DATA 301
Introduction to Data Analytics
Course Introduction

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

The Essence of the Course

The overall goal of this course is for you to:

Understand data analytics and be able to apply data analysis to data sets using a variety of software tools and techniques

This course will provide the tools for you to perform your own data analysis when encountering problems in the real-world.

My Course Goals

1) Provide the information in a simple, concise, and effective way for learning.
2) Strive for ***all*** students to understand the material and pass the course.
3) Be available for questions during class time, office hours, and at other times as needed.
4) Provide an introduction to data analytics tools and techniques so that students are able to apply data analysis to their own data sets.
5) Encourage students to continue with other data analytics or computer science courses.

Course Objectives

1) Understand data representation formats and techniques and how to use them.
2) Experience a wide-range of data analytics tools include Excel, SQL databases, R, and visualization and reporting software.
3) Develop a computational thinking approach to problem solving and use programs and scripting to solve data tasks.
4) Apply techniques to representative problems involving geographical (GIS), business, and scientific data.

Academic Dishonesty

Cheating in all its forms is strictly prohibited and will be taken very seriously by the instructor.

A guideline to what constitutes cheating:

- Assignments
  - Working in groups to solve questions and/or comparing answers to questions once they have been solved (except for group assignments).
  - Discussing HOW to solve a particular question instead of WHAT the question involves.
- Exams
  - All exams are closed book, so no course materials should be present.

Academic dishonesty may result in a "F" for the assignment or course and all instances are recorded in the Dean's office.

How to Pass This Course

The most important things to do to pass this course:

- Attend class
  - Read notes ***before*** class as preparation and try the questions.
  - Participate in class exercises and questions.
- Attend the labs and do all lab assignments
  - Labs are for marks and are practice to learn the material for the exams.

To get an "A" in this course do all the above plus:

- Practice on your own. Practice makes perfect.
  - Do more questions than in the labs. Try the techniques on your own data sets.

Systems and Tools

Connect is used for submitting assignments, posting marks, discussions, and for anonymous feedback.

All software is available in the laboratory in SCI 234.

My Expectations

You should ***SHOW UP TO CLASS AND LABS*** and put in the effort to learn the material. **Attendance ⇨ Success**

There is a wide variety in previous experience.

- Some material you may already know. Help others!!
- Build up your computer experience in labs and outside of class.
- Third-year standing means that you know how to "figure things out."

The course will be very straightforward:

Do the work and practice the techniques to do well.

The Lab Assignments

There are weekly lab assignments using computer software.

Lab assignments are worth ***30%*** of your overall grade.

Lab assignments may take more than the two hours lab time.

You have at least one week after your lab to complete it.

- No late assignments will be accepted.
- An assignment may be handed in any time before the due date.

Lab assignments are done individually or in groups of two depending on the assignment.

The lab assignments are critical to learning the material and are designed both to prepare you for the exams and build up your skills!

The In-Class Quizzes

To encourage attendance and effort, 5% of your overall grade is allocated to answering in-class questions.

These questions are answered electronically using a clicker.

- The clicker can be purchased at the bookstore.
- The clicker is personalized to you with your student number.
- At different times during all the lectures, questions reviewing material will be asked. Reponses are given using clickers.

There will be at least 75 questions throughout the semester. Each question is worth 1 mark, and you need at least 60 right answers to get the full 5%.

- That is, if you answer 45 questions right, you get 45/60 or 75%. Thus, do not worry if you must miss a class or two or forget your clicker one day!

What is Data Analysis?

Data analysis is the processing of data to yield useful insights or knowledge.

- Data processing involves finding, loading, cleaning, manipulating, transforming, modeling, and visualizing the data.
- The knowledge may be used for scientific discovery, business decision-making, or a variety of other applications.

A ***data analyst*** is a person who uses tools and applications to transform raw data into a form that will be useful.

- Data analyst jobs are projected to be one of the top jobs over the next 10 years.
  - See: http://blog.udacity.com/2014/11/data-analysts-what-youll-make.html

Why is Data Analytics Important?

Data analytics is important as society is collecting more and larger data sets all the time:

- Web: All web pages visited and links clicked, searches made, images and posts
- Business: Items purchased by date, supply chain/customers, industrial sensors
- Science: Massive data sets (biological/genomic, astronomy, physics)
- Environmental: Sensors and monitors (temperature, etc.)

and transforming this raw data into useful insights has major value:

- Web: Online advertising driven by understanding customer behaviour
- Business: Sales predictions, marketing promotions, manufacturing improvement
- Science: Scientific discoveries, new medical treatments and drugs
- Environmental: Understanding of environmental processes to allow for changing policies and behaviours

Data Analytics Toolkit

A data analyst has expertise in programming, statistics, data *munging* (transformation), and data visualization.

In this course, you will learn industrial tools and build competency in each one of these skills.

As an introductory course, the goal is to get exposure to the skills and techniques as there will not be time for mastery.

This toolkit of systems and techniques will be useful in many jobs even if they are not considered data analyst positions.

Why are you here?

A) I want to learn more about data analytics.

B) I know how important data is to my work or future work.

C) I need an upper-year elective course.

D) I already have training in computer science/statistics and want to expand my knowledge further.

E) I want an easy credit.

What Topic are You Most Interested In?

A) Excel and SQL Databases

B) Programming and Python

C) Data Visualization and GIS

D) R and Applied Statistics

E) None of the above

What is Your Major?

A) Math/Stat/Computer Science/Engineering

B) Business

C) Science (biology, chemistry, physics, environmental)

D) Arts

E) Other

What is Your Statistics Background?

A) I have taken no statistics courses.

B) I have taken a statistics course – not sure what I remember though.

C) I have taken a statistics course and can explain what a confidence interval is.

D) I have taken multiple statistics courses.

What is Your Computer Background?

A) I can use computer and mobile applications

B) I can write a formula in Excel

C) I can write a simple program in some programming language

D) I can write a query in SQL

E) I am a CS major or have taken several CS courses

What Grade are You Expecting to Get?

A) A

B) B

C) C

D) D

E) F

Why This Course is Important

Many professional jobs of the future will involve collecting, manipulating, and analyzing data. People who can understand how data can be used will have better employment opportunities.

Important results:

- Excel Proficiency – Everyone should know how to use Excel as a general data analysis and productivity software.
- Databases – Understand how they work and how to use them.
- Programming and Computational Thinking – The ability to clearly articulate a problem in a systematic way has applications beyond data analytics.
- Applied Statistics – Using R and other software makes your statistics training useful for real-world problems.
- Real-world problem solving – Your toolkit will allow you to tackle real-world data analysis problems and understand what tool to use and how to proceed.

DATA 301
Introduction to Data Analytics
Data Representation

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Computer Terminology

There is a tremendous amount of terminology related to technology.

We will introduce terminology as needed.

Using terminology precisely and correctly demonstrates ***understanding of a domain*** and simplifies communication.

Basic Computer Terminology

A ***computer*** is a device that can be programmed to solve problems.

- ***Question:*** Is a cell phone a computer? **A)** yes **B)** no

Software is programs the computer follows to perform functions.

Memory is a device which allow the computer to store data either temporarily or permanently (data is preserved even when no power).

- Many different technologies for storing data with varying performance.
- ***Flash memory*** used in mobile devices (e.g. USB drives, phones) is permanent.
- ***Question:*** Does a hard drive store data permanently? **A)** yes **B)** no

"The Cloud"

"The Cloud" is not part of your computer but rather a network of distributed computers on the Internet that provides storage, applications, and services for your computer.

These systems and services simplify tasks that otherwise would be done by programs on your computer.

Examples:

- **Dropbox** is a cloud service that allows you to store your files on machines distributed on the Internet. Automatically synchronizes any files in folder with all your machines.
- **iCloud** is an Apple service that stores and synchronizes your data, music, apps, and other content across Apple devices.

What is Data?

Data is information before it has been given any context, structure and meaning.

Raw data is produced both by people during their interactions with computers (purchases, browsing, messaging) as well as by systems and sensors (logging, monitoring, automation).

How to Measure Data Size?

Data size is measured in bytes.

- Each **byte** contains 8 **bits** - a bit is either a 0 or a 1.
- A byte can store one character of text.

Larger units:

- kilobyte (KB) - 1,000 bytes
- megabyte (MB) - $1,000^2$ bytes
- gigabyte (GB) - $1,000^3$ bytes
- terabyte (TB) - $1,000^4$ bytes
- petabyte (PB) - $1,000^5$ bytes
- exabyte (EB) - $1,000^6$ bytes
- zettabyte (ZB) - $1,000^7$ bytes

Memory Size and Data Size

Memory size is a measure of memory storage capacity in bytes.

- It represents the *maximum* capacity of data in the device.

Question: Given this flask, assume the red liquid is data and each mark represents 100 MB of data. Select a true statement.

A) Memory size is 200 MB.

B) Flask can hold at least 0.5 GB of data.

C) Data size is about 200 KB.

D) Data size of 1000 KB would "overflow device".

Massive Growth of Data – "Big Data"

Big Data is the general term used to describe the explosion of data collected and the opportunities to transform this data into useful insights to benefit society.

- "Big" as the data size challenges how the data can be processed.

Data facts:

- Over 90% of the data in world history was created over the last few years.
- Estimated that 2.5 quintillion bytes (2.5 EB) generated per day.
- Global mobile data traffic estimated at 52 million TB and tripling by 2018.
- Google processes about 3.5 billion requests/day and stores about 10 EB of data.
- Facebook collects 500 TBs/day (~2.5 billion items) and stores 100+ PB of photos.

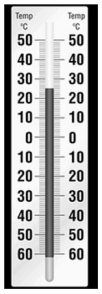
Representing Data on a Computer

Computers represent data ***digitally*** (in discrete bits).

The real-world is ***analog*** where the information is encoded on a continuous signal (spectrum of values).

Any data on a computer must be ***encoded*** as bits.

Analog versus Digital Thermometer Example



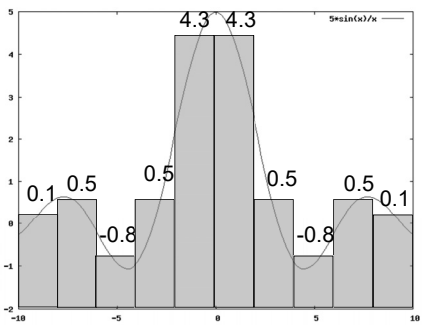
A thermometer contains liquid which expands and contracts in response to temperature changes.

The liquid level is analog, and its expansion continuous over the temperature range.

By adding marks and units to the thermometer, we are digitizing the information.

Conversion from Analog to Digital

How would you digitize this analog data into 10 discrete points?



Representing Data: Integers

An integer is a whole number. It is encoded in a computer using a fixed number of bits (usually 32 or 64).

- The first bit is a sign bit (0=positive, 1=negative).
- Negative numbers are represented in *two's complement notation*. The "largest" bit pattern is -1.

Example: 123,456,789 as a 32-bit integer:

Memory Address	0001	0002	0003	0004
	00000111	01011011	11001101	00010101

Representing Data: Doubles and Floats

A number with a decimal may be either stored as a *double* (8 bytes) or *float* (4 bytes). Values are stored using a standard IEEE 754 format:

- Represent numbers in scientific format: $N = m * 2^e$
  - m - mantissa, e - exponent, 2 - radix
  - Note that converting from base 10 to base 2 is not always precise, since real numbers cannot be represented precisely in a fixed number of bits.

Example: The number 55,125.17 stored as 4 consecutive bytes is:

0 10001110 10101110 10101010 0101011

exponent mantissa

- Stored value is: 55125.168. Note the lack of precision which may be important in scientific applications.

Representing Data: Characters

A character is mapped to a sequence of bits using a *lookup* or *translation table*.

A common encoding is ***ASCII*** *(American Standard Code for Information Interchange)*, which uses 8 bits to represent characters.

ASCII Table

First 4 bits (most significant)

Next 4 bits (least significant)

ASCII	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL
8																
9																
A		¡	¢	£		¥	¦	§	¨	©	ª	«	¬	-	®	¯
B	°	±	²	³	´	µ	¶	·	¸	¹	º	»	¼	½	¾	¿
C	À	Á	Â	Ã	Ä	Å	Æ	Ç	È	É	Ê	Ë	Ì	Í	Î	Ï
D	Ð	Ñ	Ò	Ó	Ô	Õ	Ö	×	Ø	Ù	Ú	Û	Ü	Ý	Þ	ß
E	à	á	â	ã	ä	å	æ	ç	è	é	ê	ë	ì	í	î	ï
F	ð	ñ	ò	ó	ô	õ	ö	÷	ø	ù	ú	û	ü	ý	þ	ÿ

Question:
Write your name in ASCII.

ASCII Encoding

Question: What ASCII character is `0100 0100`?

A) `A`

B) `!`

C) `@`

D) `D`

ASCII Encoding

Question: What is `Test` encoded in ASCII?

A) `01110100 01100101 01110011 01110100`

B) `01010100 01100101 01110011 01110100`

C) `01000101 01010110 00110111 01000111`

D) `01010100 01000101 01010011 01010100`

Representing Text Beyond ASCII - Unicode

Although ASCII is suitable for English text, many world languages, including Chinese, require a larger number of symbols to represent their basic alphabet.

The ***Unicode standard*** uses patterns of 16-bits (2 bytes) to represent the major symbols used in all languages.

- First 256 characters exactly the same as ASCII.
- Maximum # of symbols: 65,536.

Aside: Character Fonts

When a character is displayed on a screen, a particular font is used.

The font is how to present the data (the character).

Note that the font itself is data where each character has a mapping to a sequence of bytes representing pixels (image) on how to draw the character.

Representing Data: Strings

A ***string*** is a sequence of characters.

A string has a terminator to know when it ends:

- ***Null-terminated string*** - last byte value is 0 to indicate end of string.
- ***Byte-length string*** - length of string in bytes is specified (usually in the first few bytes before string starts).

Representing Data: Dates and Times

A ***date*** value can be represented in multiple ways:

- Integer representation - number of days past since a given date
  - Example: Julian Date (astronomy) – number of days since noon, January 1, 4713 BC
- String representation - represent a date's components (year, month, day) as individual characters of a string
  - Example: YYYYMMDD or YYYYDDD
  - Please do not reinvent Y2K by using YYMMDD!!

A ***time*** value can also be represented in similar ways:

- Integer representation - number of seconds since a given time
  - Example: # of seconds since Thursday, January 1, 1970 (UNIX)
- String representation - hours, minutes, seconds, fractions
  - Example: HHMMSSFF

Encoding Other Data

We have seen how we can encode characters, numbers, and strings using only sequences of bits (and translation tables).

The documents, music, and videos that we commonly use are much more complex. However, the principle is exactly the same. We use sequences of bits and ***interpret*** them based on the ***context*** to represent information.

As we learn more about representing information, always remember that everything is stored as bits, it is by interpreting the context that we have information.

Metadata

Metadata is data that describes other data.

Examples of metadata:

- names of files
- column names in a spreadsheet
- table and column names and types in a database

Metadata helps you understand how to interpret and manipulate the data.

Files

A ***file*** is a sequence of bytes on a storage device.

- A file has a name.
- A computer reads the file from a storage device into memory to use it.

The operating system manages how to store and retrieve the file bytes from the device.

The program using the file must know how to interpret those bytes based on its information (e.g. metadata) on what is stored in the file.

File Encoding

A ***file encoding*** is how the bytes represent data in a file.

A file encoding is determined from the file extension (e.g. .txt or .xlsx) which allows the operating system to know how to process the file.

- The extension allows the OS to select the program to use. The program understands how to process the file in its format.

File Encodings: Text Files

A ***text file*** is a file encoded in a character format such as ASCII or Unicode. These files are readable by humans.

There are many different text file encodings:

- CSV – comma-separate file – each line is a record, fields separated by commas
- tab-separated file – each line is a record, fields separated by tabs
- JSON file – data encoded in JSON format
- XML file – data encoded in XML format

Data analytics will often involve processing text files.

File Encodings: Text File Examples

CSV (comma-separated) file:

```
Id,Name,Province,Balance
1,Joe Smith,BC,345.42
```

CSV (tab-separated) file:

```
Id Name   Province  Balance
1  Joe Smith BC 345.42
```

JSON file:

```
{"Id":1, "Name":"Joe Smith", "Province":"BC", "Balance":345.42}
```

XML file:

```
<customers><customer>
   <id>1</id>   <name>Joe Smith</name>
    <province>BC</province> <balance>345.42</balance>
</customer></customers>
```

Question:
In these file encodings, what is data and what is metadata?

Note: XML and JSON are case-sensitive.

File Encodings: Binary File

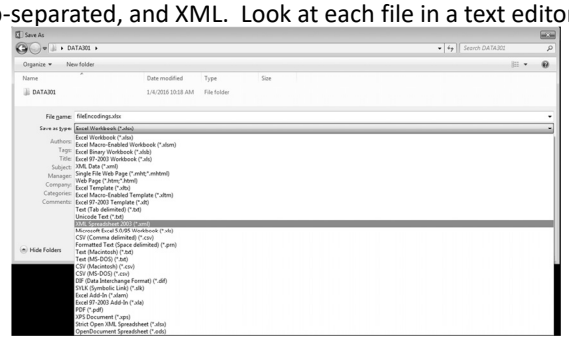
A ***binary file*** encodes data in a format that is not designed to be human-readable and is in the format used by the computer.

Binary files are often faster to process as they do not require translation from text form and may also be smaller.

Processing a binary file requires the user to understand its encoding so that the bytes can be read and interpreted properly.

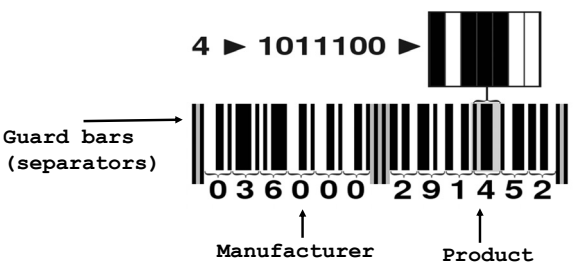
Try it: File Encodings

Question: Use the `fileEncodings.xlsx` file and save the file as CSV, tab-separated, and XML. Look at each file in a text editor.



UPC Barcodes

Universal Product Codes (UPC) encode manufacturer on left side and product on right side. Each digit uses 7 bits with different bit combinations for each side (can tell if upside down).



QR Codes

A **QR** (**Q**uick **R**esponse) code is a 2D optical encoding developed in 1994 by Toyota with support for error correction.

Hello World!

Part of Syllabus

Make your own codes at: www.qrstuff.com.

NATO Broadcast Alphabet

The code for broadcast communication is purposefully inefficient, to be distinctive when spoken amid noise.

A	Alpha	J	Juliet	S	Sierra
B	Bravo	K	Kilo	T	Tango
C	Charlie	L	Lima	U	Uniform
D	Delta	M	Mike	V	Victor
E	Echo	N	November	W	Whiskey
F	Foxtrot	O	Oscar	X	X-ray
G	Golf	P	Papa	Y	Yankee
H	Hotel	Q	Quebec	Z	Zulu
I	India	R	Romeo		

Question: Broadcast your name to a partner using the NATO broadcast alphabet.

Advanced: The Time versus Space Tradeoff

A fundamental challenge in computer science is encoding information efficiently both in terms of space and time.

At all granularities (sizes) of data representation, we want to use as little space (memory) as possible. However, saving space often makes it harder to figure out what the data means (think of compression or abbreviations). In computer terms, the data takes longer to process.

The ***time versus space tradeoff*** implies that we can often get a faster execution time if we use more memory (space). Thus, we often must strive for a balance between time and space.

Review: Memory Size

Question: Which is bigger?

A) 10 TB

B) 100 GB

C) 1,000,000,000,000 bytes

D) 1 PB

Review: Metadata and Data

Question: Select a **TRUE** statement.

A) It is possible to have data without metadata.

B) Growth rates of data generation are decreasing.

C) It is possible to represent decimal numbers precisely on a computer.

D) A character encoded in Unicode uses twice as much space as ASCII.

Conclusion

All ***data*** is encoded as bits on a computer. ***Metadata*** provides the context to understand how to interpret the data to make it useful.

- Memory capacity of devices and data sizes are measured in bytes.

Files are sequences of bytes stored on a device. A ***file encoding*** is how the bytes are organized to represent the data.

- Text files (comma/tab separated, JSON, XML) are often processed during data analytics tasks. Binary files are usually only processed by the program that creates them.

As a data analyst, understanding the different ways of representing data is critical as it is often necessary to transform data from one format to another.

Objectives

- Explain why it is important to understand and use correct terminology.
- Define: computer, software, memory, data, memory size/data size, cloud
- Explain "Big Data" and describe data growth in the coming years.
- Compare and contrast: digital versus analog
- Briefly explain how integers, doubles, and strings are encoded.
- Explain why ASCII table is required for character encoding.
- Explain why Unicode is used in certain situations instead of ASCII.
- Explain the role of metadata for interpreting data.
- Define: file, file encoding, text file, binary file
- Encode using the NATO broadcast alphabet.
- Discuss the time-versus-space tradeoff.

DATA 301
Introduction to Data Analytics
Spreadsheets: Microsoft Excel

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why Spreadsheets and Microsoft Excel?

Spreadsheets are the most common, general-purpose software for data analysis and reporting.

Microsoft Excel is the most popular spreadsheet program with hundreds of millions of installations.

- The spreadsheet concepts translate to other products.

Excel and spreadsheets are not always the best tool for data analysis, but they are great for quick analysis, reporting, and sharing.

Spreadsheet Overview

A ***spreadsheet*** organizes information into a two-dimensional array of cells (a *table*).

A ***cell*** has two components:

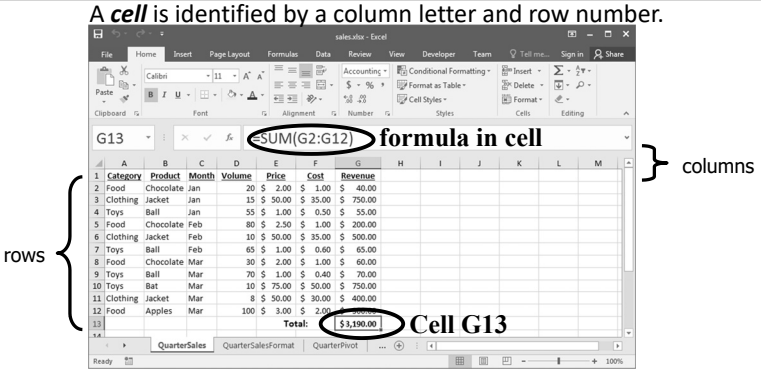
- an address - specified given a column letter and row number
- a location - that can store a number, text, or formula

The power of a spreadsheet is that we can write simple formulas (commands) to perform calculations and immediately see the results of those calculations.

Spreadsheets are very common in business and reporting applications.

Spreadsheet Addressing

A ***cell*** is identified by a column letter and row number.



Spreadsheet Addressing

The rows in a spreadsheet are numbered starting from 1.

The columns are represented by letters.

- A is column 1, B is column 2, …, Z is column 26, AA is column 27, …

A cell is identified by putting the column letter first then the row number.

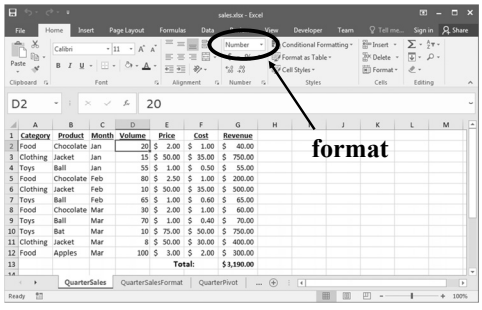
- e.g. B3 is the 2nd column and the 3rd row.

Question: What column number is AD? How about BAD?

Spreadsheet Data Entry

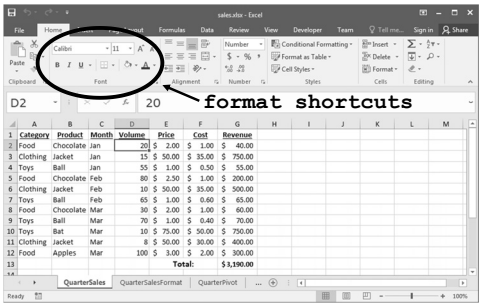
An entry is added to a cell by clicking on it and typing in the data.

- The data may be a number, text, date, etc. Type and *format* are auto-detected.



Spreadsheet Formatting

Formatting: bold, italics, underline, fonts, colors



Spreadsheet Selecting Cells

Multiple ways of selecting cells:

1) With the mouse, (left) click and drag mouse to select a rectangle region of cells.

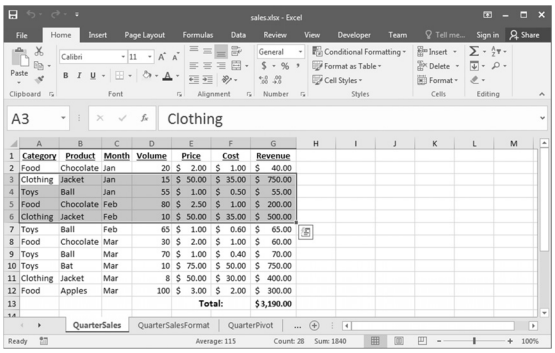
2) With keyboard, hold `SHIFT` key and use arrow keys to select a rectangle region of cells.

3) With mouse and keyboard, while holding `CTRL` key, (left) click on individual cells to select non-contiguous cells.

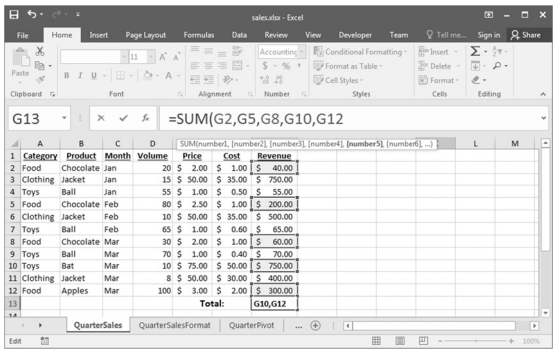
4) Click on a row number to select a whole row.

5) Click on a column header to select a whole column.

Range Selecting Cells Example



Selecting Individual Cells Example



Manipulating Cells

Once you have selected one or more cells, there are several common actions you can perform:

1) DELETE

- delete the contents of all cells by pressing delete key
- delete the contents and the cell locations (then shift remaining) by selecting `Edit` menu, `Delete...` or `Delete…` from pop-up menu (brought up by right click).

2) Cut, Copy, Paste

- cut - copies selected cells to clipboard and removes from document
- copy - copies selected cells to clipboard
- paste - copies cells in clipboard to sheet starting at currently selected cell

3) Add selected cells to a formula (requires that you were previously constructing a formula before selecting the cells).

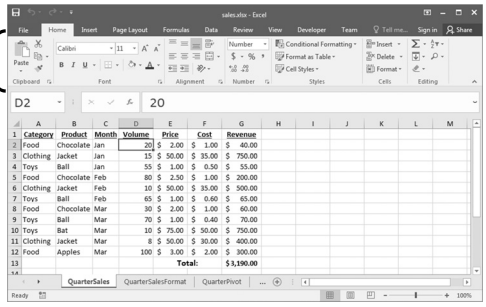
Manipulating Cells - Filling

Filling combines copy and paste.

There is a small box or tab beyond the cell's lower right corner (fill handle). Grab it with the cursor and pull to other cells.

Cut, Copy, Paste

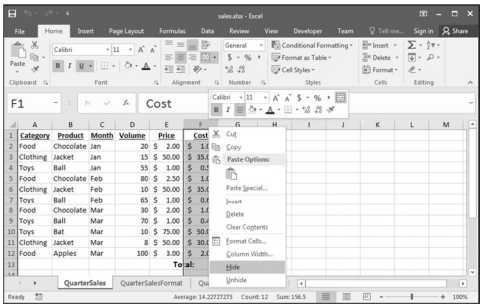
cut,
copy,
paste



Hiding Columns and Rows

Right-clicking on the column or row header and selecting **`Hide`**.

- The column/row still exists but will not be displayed or printed unless unhidden.



Selecting Cells Question

Question: Which method allows you to select non-contiguous cells in a spreadsheet?

A) hold `SHIFT` key and use arrow keys

B) With the mouse left click on a cell and drag mouse

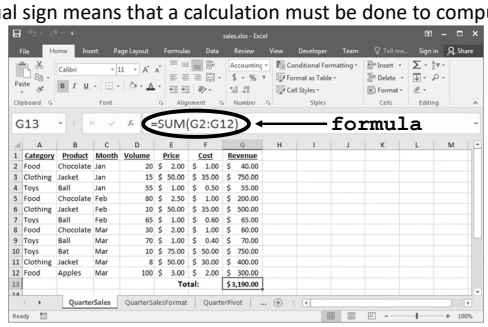
C) hold `CTRL` key and use arrow keys

D) hold `CTRL` key and left click on cells

Entering Formulas

A ***formula*** is any expression that begins with an equal sign ("**=**").

- The equal sign means that a calculation must be done to compute the cell value.



Formula Expressions

A ***formula*** expression can consist of literals (numbers, text strings), operators, functions, and cell references.

Simple mathematical expressions:

- `= 1 + 5`
- `= 1.5 * 3.14 + 42`

Common functions:

- `= ROUND(PI(),2)` // Result is 3.14
- `= CONCATENATE("Hello", " World")` // Hello World
- Other common functions for trigonometry, dates, and financial.

Formula Expressions

The power of formulas comes from using cell references (similar to variable names in programming).

Cell reference examples:

- `= A1 + A2`
- `= B1 + A3 - A4`

Formulas Question

Question: A cell contains the following: **=2+4*3** What is the value of the cell?

A) 14

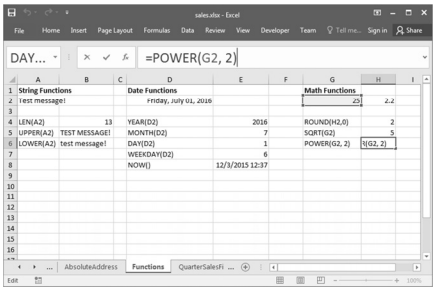
B) 18

C) =2+4*3

Using Excel Functions

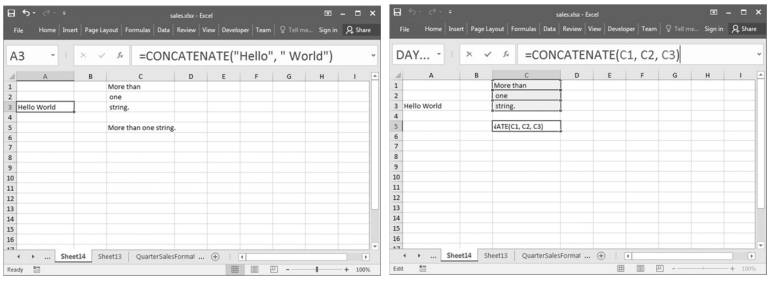
Excel has a large number of built-in functions to use.

A ***function*** takes arguments as input and produces an output.



Concatenation

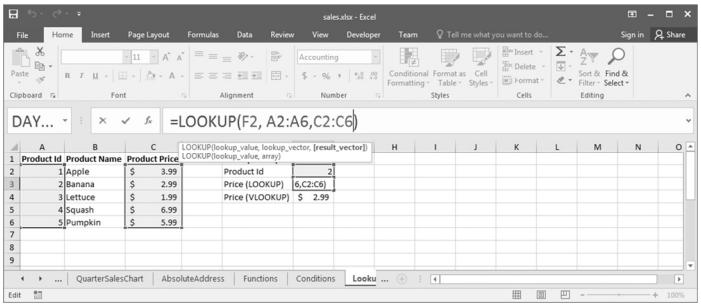
String concatenation is when two or more strings are combined by appending them in order. Function in Excel is `CONCATENATE()` or `&`.



`LOOKUP` Function

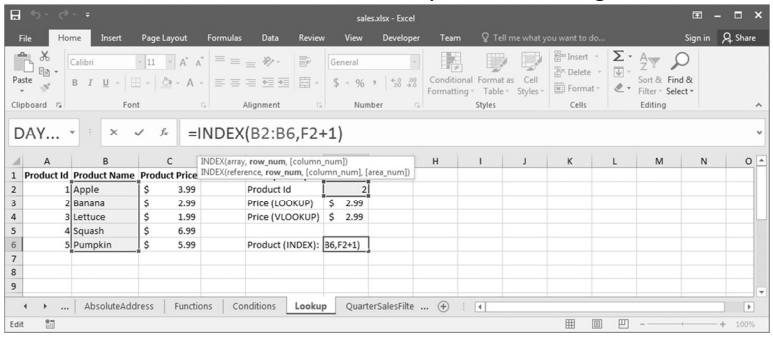
The `LOOKUP` function searches for a value in a column.

- `VLOOKUP` searches a column in a table ; `HLOOKUP` searches a row in a table.



`INDEX` Function

`INDEX()` returns the value in the array of cells at the given index.



Formulas Question

Question: A cell contains the following: **'ABC'+'DEF'**. What is the value of the cell?

A) error

B) ABCDEF

C) 'ABC'+'DEF'

Formulas Question

Question: How many of the following statements are **TRUE**?

1) `CONCATENATE` function can take 3 arguments.

2) There is an Excel function that has 0 arguments.

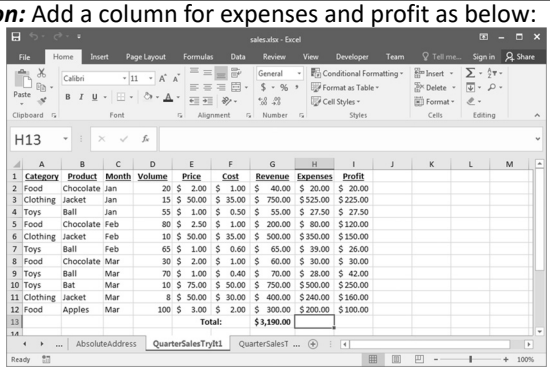
3) `=INDEX({1,3,5},2)` returns `5`.

4) `=LOOKUP(5,{1,3,5},{"a","b","c"})` returns `"c"`.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Entering Formulas

Question: Add a column for expenses and profit as below:



Category	Product	Month	Volume	Price	Cost	Revenue	Expenses	Profit
Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00	$ 20.00	$ 20.00
Clothing	Jacket	Jan	15	$ 50.00	$ 35.00	$ 750.00	$ 525.00	$ 225.00
Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00	$ 27.50	$ 27.50
Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00	$ 80.00	$ 120.00
Clothing	Jacket	Feb	10	$ 50.00	$ 35.00	$ 500.00	$ 350.00	$ 150.00
Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00	$ 39.00	$ 26.00
Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00	$ 30.00	$ 30.00
Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00	$ 28.00	$ 42.00
Toys	Bat	Mar	10	$ 75.00	$ 50.00	$ 750.00	$ 500.00	$ 250.00
Clothing	Jacket	Mar	8	$ 50.00	$ 30.00	$ 400.00	$ 240.00	$ 160.00
Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00	$ 200.00	$ 100.00
					Total:	$ 3,190.00		

Advanced Spreadsheet Addressing

The dollar sign "**$**" is a symbol that indicates an ***absolute address***.

- By default, addresses are "relative" in the sense that if they are in a formula that is copied to another cell, they will be changed relative to where they were copied from their origin.

Example:

- Cell `A1` has the formula `=A2+B1`
- Copy contents of cell `A1` to cell `C4`.
- Formula changes to `=C5+D4` because moved down three rows and over two columns.
- If cell `A1` had the formula `=$A$2+$B$1`, then the same formula would be in cell `C4`.
- Question: What if formula was `=$A2+B$1`?

Formulas and References Question

Question: Cell **A1** contains the following: **=$B2+D$4**. What is the formula if the cell is copied to cell **D3**?

A) error

B) =$B2+D$4

C) =$B4+F$4

D) =$B4+G$4

Aggregate Functions

An ***aggregate function*** computes a summary function over a range of cells. The values can either be data values or cell locations.

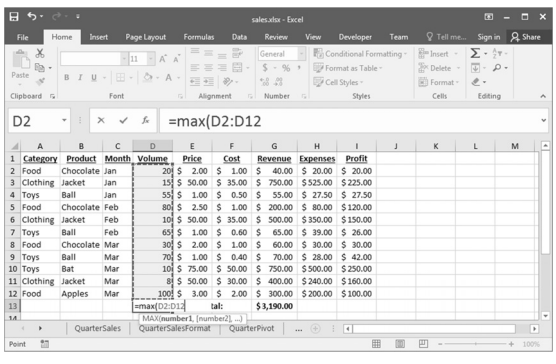
Common functions are:

- `MIN(<value list>)` - returns minimum value in list
- `MAX(<value list>)` - returns maximum value in list
- `SUM(<value list>)` - returns sum of all values in list
- `AVERAGE(<value list>)` - returns average of values in list
- `COUNT(<value list>)` - returns count of values in list
- `MEDIAN(<value list>)` - returns median value of list

If specifying a cell rectangle, give the upper left and lower right corners, separated by a colon.

- e.g. `=AVERAGE(A3:E6)` - rectangle of 4 rows and 5 columns

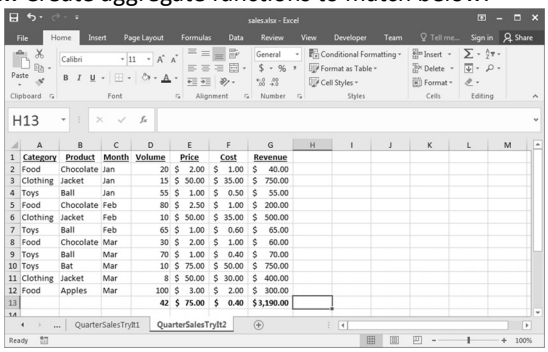
Aggregate Functions Example



Category	Product	Month	Volume	Price	Cost	Revenue	Expenses	Profit
Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00	$ 20.00	$ 20.00
Clothing	Jacket	Jan	15	$ 50.00	$ 35.00	$ 750.00	$ 525.00	$ 225.00
Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00	$ 27.50	$ 27.50
Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00	$ 80.00	$ 120.00
Clothing	Jacket	Feb	10	$ 50.00	$ 35.00	$ 500.00	$ 350.00	$ 150.00
Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00	$ 39.00	$ 26.00
Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00	$ 30.00	$ 30.00
Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00	$ 28.00	$ 42.00
Toys	Bat	Mar	10	$ 75.00	$ 50.00	$ 750.00	$ 500.00	$ 250.00
Clothing	Jacket	Mar	8	$ 50.00	$ 30.00	$ 400.00	$ 240.00	$ 160.00
Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00	$ 200.00	$ 100.00
				=max(D2:D12	tal:	$ 3,190.00		

Try it: Aggregate Functions

Question: Create aggregate functions to match below:



	A	B	C	D	E	F	G
1	Category	Product	Month	Volume	Price	Cost	Revenue
2	Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00
3	Clothing	Jacket	Jan	15	$ 50.00	$ 35.00	$ 750.00
4	Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00
5	Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00
6	Clothing	Jacket	Feb	10	$ 50.00	$ 35.00	$ 500.00
7	Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00
8	Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00
9	Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00
10	Toys	Bat	Mar	10	$ 75.00	$ 50.00	$ 750.00
11	Clothing	Jacket	Mar	8	$ 50.00	$ 30.00	$ 400.00
12	Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00
13				42	$ 75.00	$ 0.40	$ 3,190.00

Aggregate Functions Question

Question: Assume the cells in the range `A1:C4` each contain a number that is equal to their row number (e.g. `B3` contains `3`). How many of the following statements are **TRUE**?

1) The number of cells in the range is `12`.
2) The value of `SUM(A1:C4)` is `20`.
3) The value of `COUNTIF(A1:B4,">2")` is `4`.
4) `AVERAGE(A1:C4) > MAX(C2:C3)`

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Aggregate Functions Question

Question: Assume the three cells in the range `A1:C1` contain numbers. Which of these formula output results is **ALWAYS** the largest?

A) `MAX(A1:C1)`

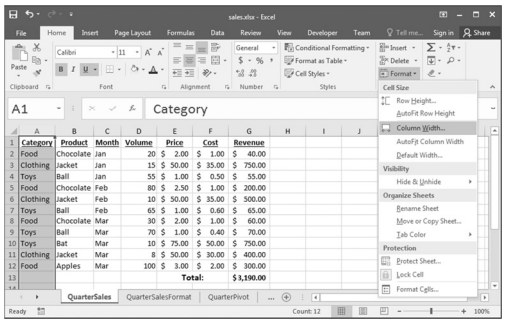
B) `MIN(A1:C1)`

C) `COUNT(A1:C1)`

D) `SUM(A1:C1)`

E) none of the above are always guaranteed to be the largest

Other Formatting: Column Width



Resizing columns/rows:

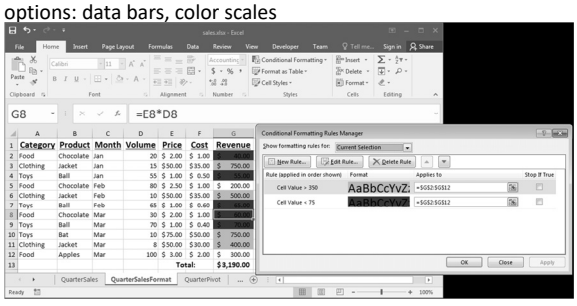
Auto-resize by double clicking on border between columns or using the `Format` option.

Drag row/column border for manual resize.

Conditional Formatting

Conditional formatting allows you to change the cell format based on data values. This is accessible under **`Styles`**.

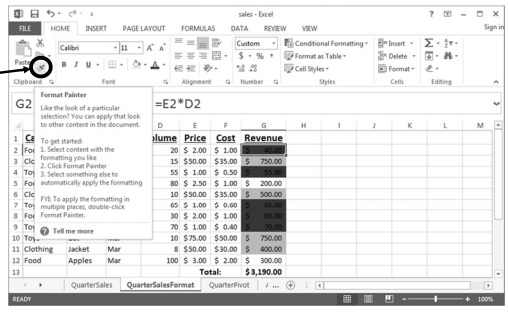
- Other options: data bars, color scales



Conditional Formatting Result

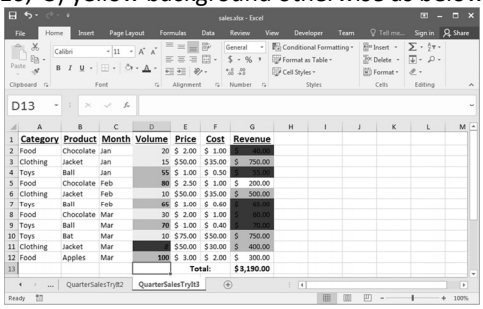
The format painter button allows you to copy formatting to many cells. Select the cell, click paint button, then highlight cells to have identical formatting.

`format painter button`



Try it: Conditional Formatting

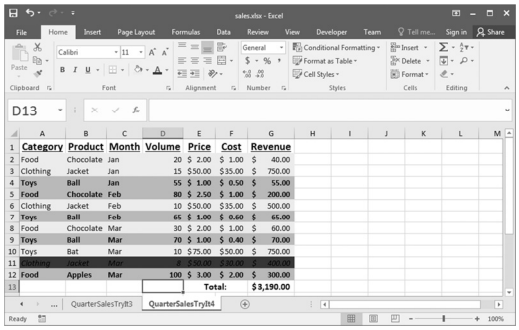
Question: Format rows so: 1) bold/green if volume > 50, 2) italics/red if volume < 10, 3) yellow background otherwise as below:



Category	Product	Month	Volume	Price	Cost	Revenue
Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00
Clothing	Jacket	Jan	15	$50.00	$35.00	$ 750.00
Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00
Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00
Clothing	Jacket	Feb	10	$50.00	$35.00	$ 500.00
Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00
Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00
Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00
Toys	Bat	Mar	10	$75.00	$50.00	$ 750.00
Clothing	Jacket	Mar	8	$50.00	$30.00	$ 400.00
Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00
				Total:		$3,190.00

Try it: Conditional Formatting Challenge

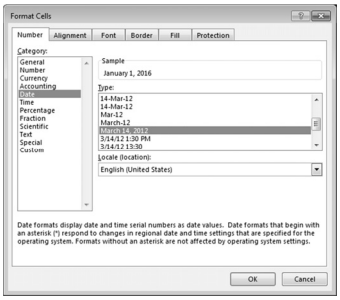
Question: Take the previous formatting and apply it to whole row:



Category	Product	Month	Volume	Price	Cost	Revenue
Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00
Clothing	Jacket	Jan	15	$50.00	$35.00	$ 750.00
Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00
Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00
Clothing	Jacket	Feb	10	$50.00	$35.00	$ 500.00
Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00
Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00
Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00
Toys	Bat	Mar	10	$75.00	$50.00	$ 750.00
Clothing	Jacket	Mar	8	$50.00	$30.00	$ 400.00
Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00
				Total:		$3,190.00

Date and Type Formats

Formatting data helps users read and understand data and is especially important for numbers and dates. Use built-in or custom formats.



Spreadsheets for Data Management

A spreadsheet is often used as a "database". A database is an organized representation of information.

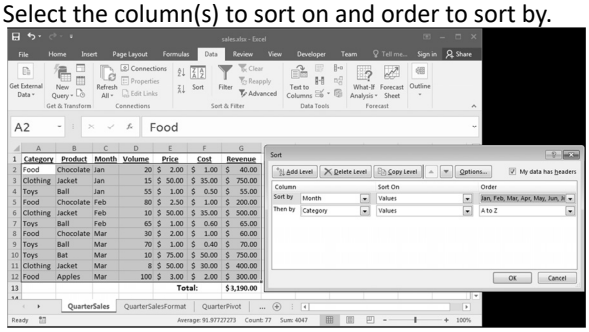
- Examples: schedules and calendars, timesheets, expenses and finances, records, notes, and recipes, data research/analysis

We can use a spreadsheet as a database by:

- Using a row to store all the information about something we want to represent.
- Giving each column a meaningful name. A column represents a property or feature of the object stored in the row.
- Using the formulas to calculate new facts from the data.
- Using sorting to organize the data by key features.
- Using simple filtering (querying) to only show the most important data or data of interest.

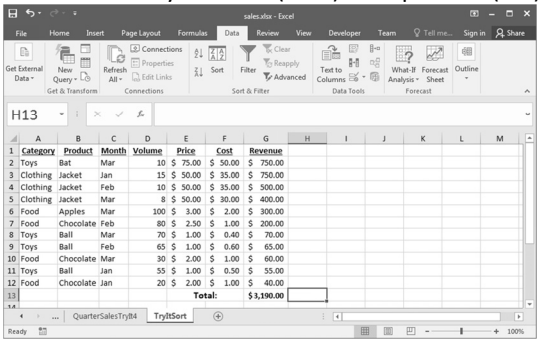
Sorting Data

Data can be sorted by selecting the **`Sort`** option under the **`Data`** menu. Select the column(s) to sort on and order to sort by.



Try it: Sort

Question: Sort the data by revenue (desc) then product (asc).



Category	Product	Month	Volume	Price	Cost	Revenue
Toys	Bat	Mar	10	$ 75.00	$ 50.00	$ 750.00
Clothing	Jacket	Jan	15	$ 50.00	$ 35.00	$ 750.00
Clothing	Jacket	Feb	10	$ 50.00	$ 35.00	$ 500.00
Clothing	Jacket	Mar	8	$ 50.00	$ 30.00	$ 400.00
Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00
Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00
Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00
Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00
Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00
Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00
Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00
				Total:		$3,190.00

Filtering

A ***filter*** shows a subset of the rows in the spreadsheet that pass a given condition (test).

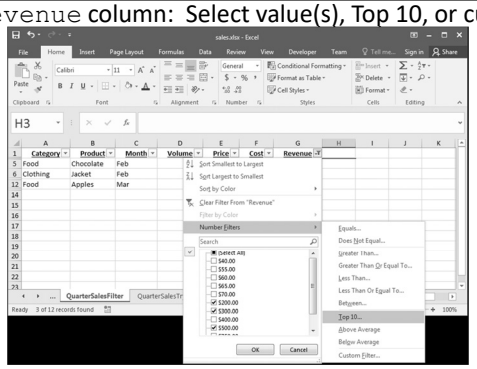
Select **`Auto Filter`** under the **`Data`** then **`Filter`** menu.

Once you select `Auto Filter`, each column heading has a drop-down list. By selecting a filtering criteria from the list, you can limit the rows that are displayed.

It is possible to filter on more than one column at the same time.

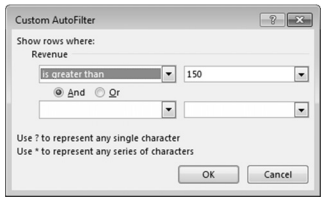
Filter Example

Filter on `Revenue` column: Select value(s), Top 10, or custom filter.



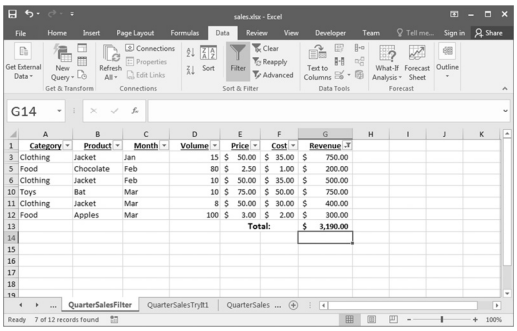
Custom Filter Example

Filter on `Revenue` column: Custom filter with **`Revenue > 150`**



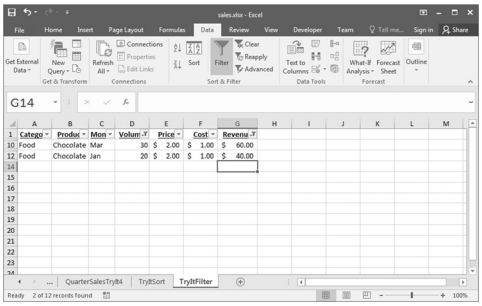
Custom Filter Result

Filter on `Revenue`: Custom filter result with **`Revenue > 150`**



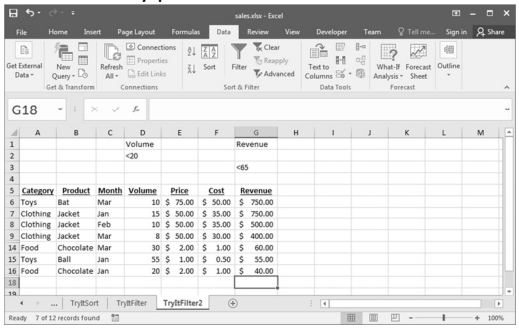
Try it: Filter

Question: Filter the data so only products with volume < 50 and revenue < $100 are shown.



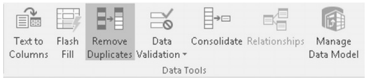
Try it: Filter Challenge

Question: Filter the data so only products with volume < 20 ***or*** revenue < $65 are shown.



Removing Duplicates

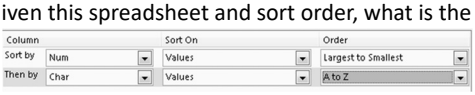
To remove duplicates, select your Data then `Remove Duplicates`.



Sorting Question

Question: Given this spreadsheet and sort order, what is the output?

Num	Char
1	A
1	a
1	B
1	b
2	A
2	b
3	a
3	B



A)

Num	Char
3	B
3	a
2	A
2	b
1	A
1	B
1	a
1	b

B)

Num	Char
3	a
3	B
2	A
2	b
1	a
1	A
1	b
1	B

C)

Num	Char
3	a
3	B
2	A
2	b
1	A
1	a
1	B
1	b

Filtering Question

Question: Given this spreadsheet, how many of these statements are **TRUE**?

	A	B
1	Number	Letter
2	1	a
3	2	b
4	3	c
5	4	d
6	5	e
7		

1) The data is sorted ascending by `Number`.

2) Filter `Number > 3` shows 3 rows.

3) Filter `Letter >= "c"` shows 3 rows.

4) Filter `Number < 3 OR Letter > "b"` shows 5 rows.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Charts

A ***chart*** is a graphical representation of spreadsheet data.

A chart is of a particular type (line, bar, etc.) and requires the user to supply the data that will be displayed in the chart.

Chart: Select Data and Type

Select **Insert**, then click **`Chart`** Icon, and pick the chart type.

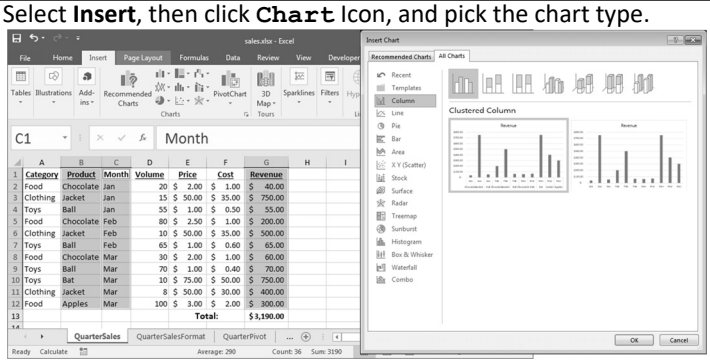
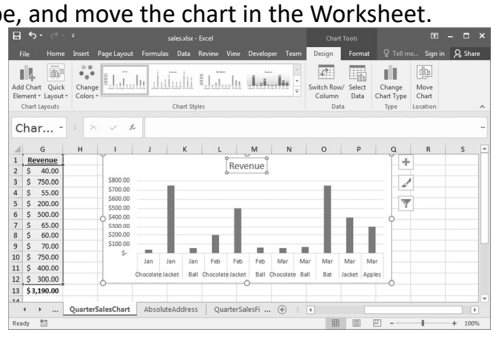


Chart Options

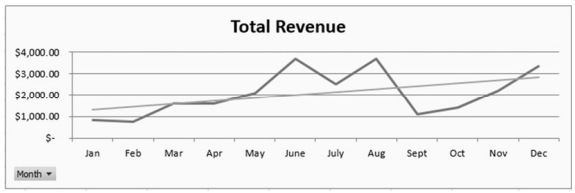
`Chart Tools` allows you to modify the data in the chart, change the chart type, and move the chart in the Worksheet.

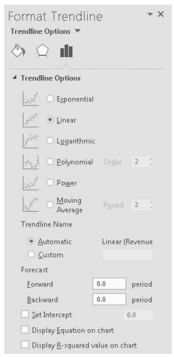


Trendlines

`Trendlines` can be easily added to any chart.

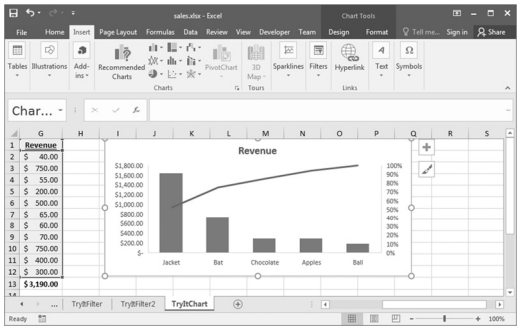
- Linear treadline for monthly revenue. Good choice?





Try it: Chart

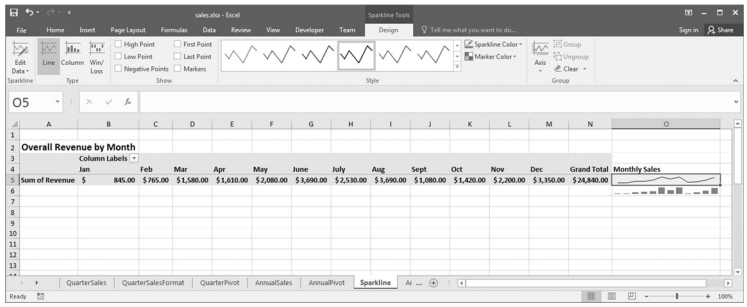
Question: Create a chart that makes it easy to see the best selling product.



Sparklines

A ***sparkline*** is a tiny chart in a worksheet cell for a quick data overview.

- `Insert` then select a Sparkline (line, column, win/loss). May put text in sparkline cell.



What-If

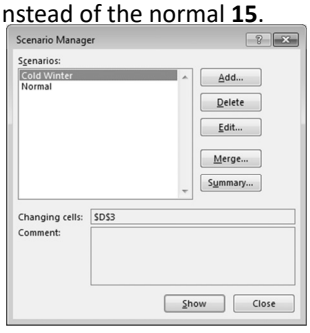
What-If scenarios help understand different possibilities.

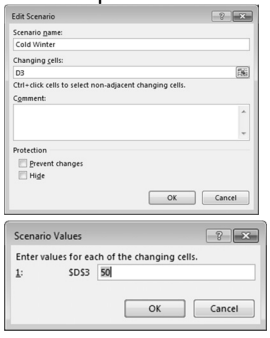
A what-if scenario is created under `Data` then `What-If Analysis` then `Scenario Manager`.

To define a scenario, give it a name and list the cells that will change with this scenario.

What-If Scenarios Example

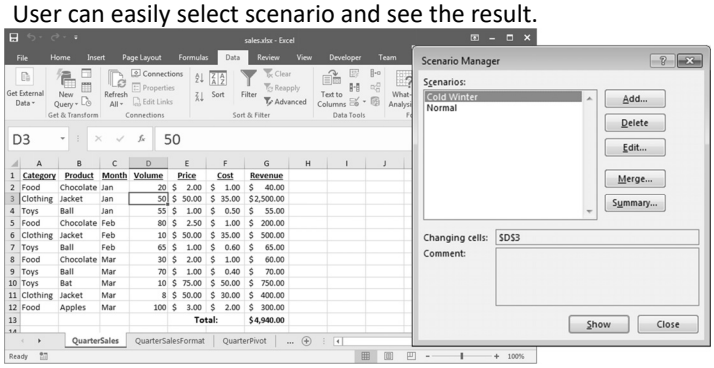
Consider what happens with a cold winter and we predict to sell **50** jackets instead of the normal **15**.





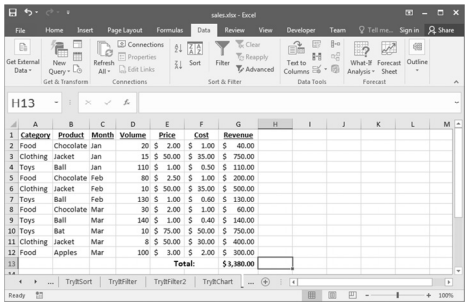
What-If Scenarios Example

User can easily select scenario and see the result.



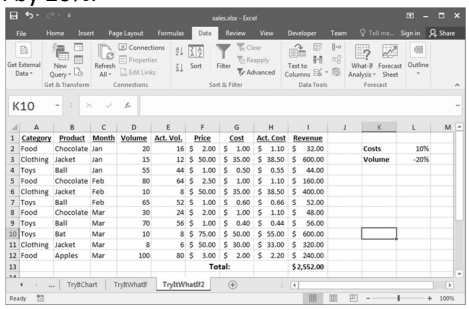
Try it: What-If Scenario

Question: Create a what-if scenario that wherever balls are sold, the volume is double than normal.



Try it: What-If Scenario Challenge

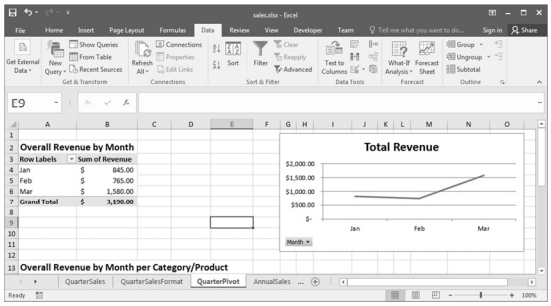
Question: Create a what-if scenario that all costs go up by 10% and volume down by 20%.



Pivot Tables

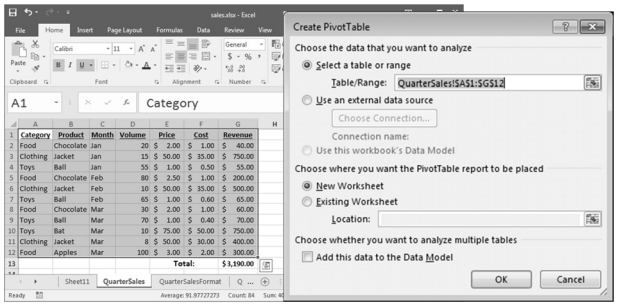
Pivot tables allow for easily aggregating and exploring large data sets.

- For example, our data set can be summarized by revenue by month.



Creating a Pivot Table

To create, select the data and then `Insert`, `Pivot Table`.

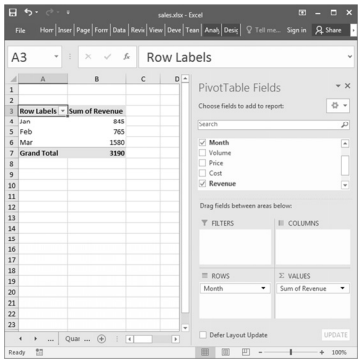


Creating a Pivot Table

Add fields to pivot table.

Field may either be:

- Row value
- Column value
- Cell value (aggregated)
- Used in a filter



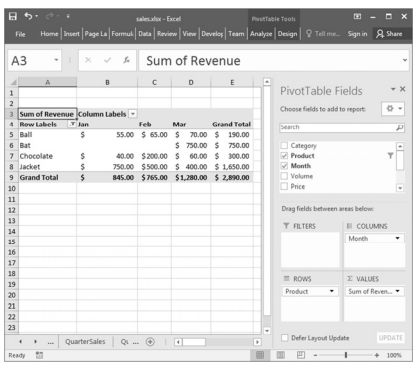
Creating a Pivot Table Example

Products are rows.

Months are columns.

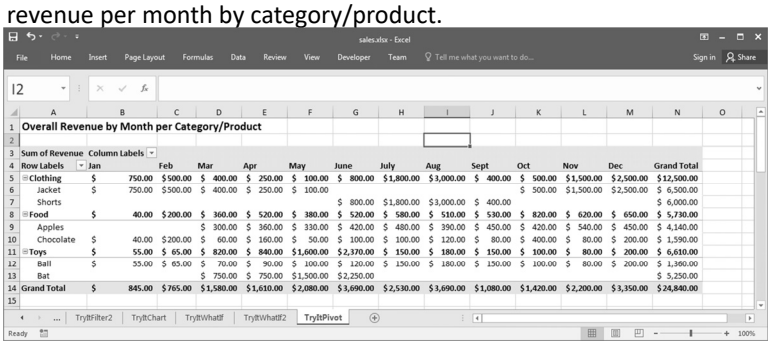
Each cell is a sum of revenue per product for that month.

Filter on product.



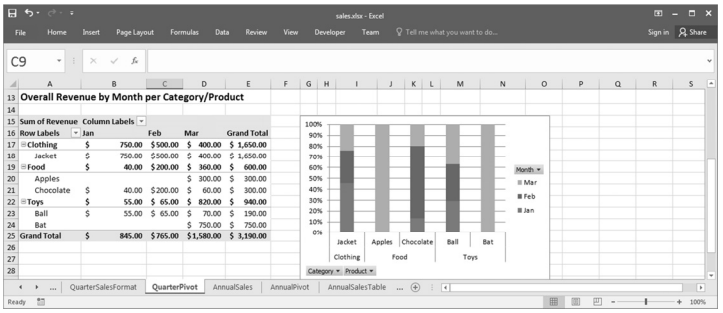
Try it: Pivot Table

Question: Create a pivot table using the annual sales data that shows revenue per month by category/product.



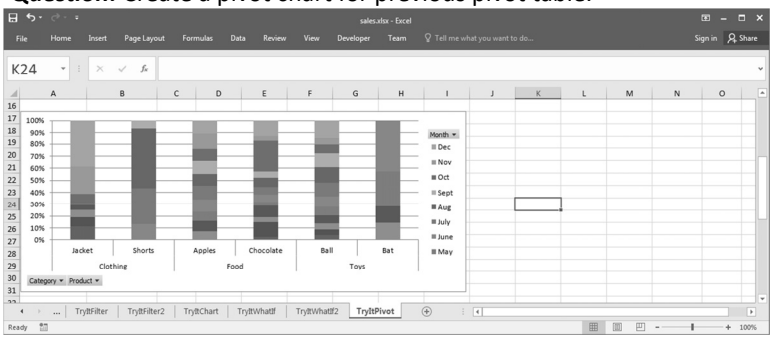
Pivot Charts

A ***pivot chart*** is a chart attached to a pivot table. Create it under `Insert` then `Pivot Chart`.



Try it: Pivot Chart

Question: Create a pivot chart for previous pivot table.



What-if and Pivot Tables Question

Question: How many of the following statements are **TRUE**?

1) A what-if scenario can have multiple cells change not just one.

2) A pivot table field can be used in `ROWS` and `COLUMNS` at the same time.

3) A pivot table field can be used in `VALUES` more than once.

4) In our sales spreadsheet example, if `Product` and `Category` are both used in `ROWS` then the order they are list does not matter.

5) It is not possible for a field that is a string to be used in `VALUES`.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Conditions and Decisions

A ***condition*** is an expression that is either `TRUE` or `FALSE`.

Conditions are used to make decisions and perform different actions depending on the condition value.

Excel condition and decision functions:

- `FALSE()` – returns `FALSE`
- `TRUE()` – returns `TRUE`
- `AND(cond1, cond2)` – returns `TRUE` if both `cond1` and `cond2` are true
- `OR(cond1, cond2)` – returns `TRUE` if either or both of `cond1` and `cond2` are true
- `NOT(cond)` – returns `TRUE` if `cond` is `FALSE`

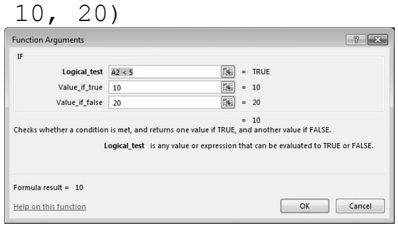
Decisions using `IF()`

The `IF()` function is used to make a decision based on a condition.

- `IF(condition, value_if_true, value_if_false)`

Example: If cell A2 is less than 5, return 10 otherwise return 20.

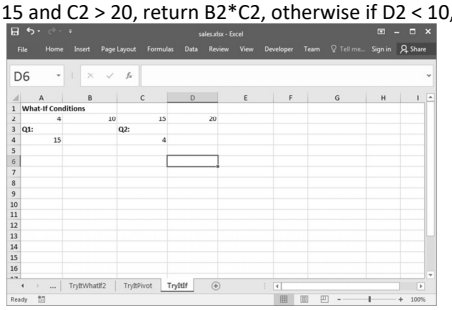
```
= IF(A2 < 5, 10, 20)
```



Try it: Conditions and `IF()`

Question: Create two conditions:

- 1) If cell B2 >= 10, then show C2, otherwise D2.
- 2) If cell B2 < 15 and C2 > 20, return B2*C2, otherwise if D2 < 10, return 1, else 4.



Decisions using `IF()` Question

Question: How many of these statements are **TRUE**? **`A1=40,A2=10`**

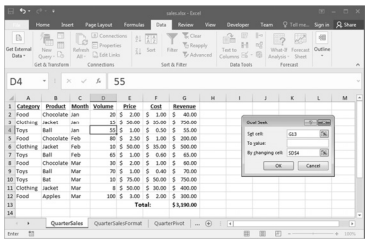
1) `=AND(FALSE(), TRUE())`
2) `=OR(FALSE(), NOT(TRUE()))`
3) `=IF(A1=40, 5, 10)` returns `10`.
4) `=IF(OR(A1=40,A2>10),1, 2)` returns `2`.
5) `=IF(A2=10,IF(A1=40,FALSE()),TRUE())`

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Goal Seek

Goal seek is used to have Excel solve for a variable given the target value of another cell.

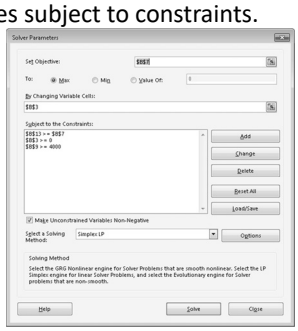
- Example: How many balls would we have to sell in January to have total revenue for first 3 months of $4000? Answer: 865



Linear Programming with Solver

Solver performs linear programming to maximize or minimize a given function by changing multiple variables subject to constraints.

	A	B
1	**Solver Example: Maximum House You Can Afford**	
2		
3	**House Value:**	**$ 500,000.00**
4	**Down payment:**	$ 20,000.00
5	**Mortgage amount:**	$ 480,000.00
6	**Amoritization in years:**	25
7	**Monthly mortgage expense:**	**$2,533.62**
8		
9	**Monthly income:**	$ 5,000.00
10	**Mortgage rate:**	4%
11		
12	**Maximum mortgage expense percentage:**	30%
13	**Maximum mortgage expense amount:**	$ 1,500.00
14		$1,033.62



Analysis ToolPak

The Analysis ToolPak is an Excel add-in that has a set of statistical and data analysis tools such as ANOVA, covariance, regression, and t-test.

Analysis ToolPak is not installed by default.

- To install: `File → Options → Add-Ins`
- Select `Excel Add-ins` inn the `Manage:` box and select `Go…`
- Choose `AnalysisToolPak` and select `OK`

You should now see `Data Analysis` under the Data tab

Regression

Linear regression models the relationship between a dependent variable *y* and explanatory variables *X*.

- Simple linear regression has one explanatory variable: $y = Bx + \varepsilon$
- Used to fit a predictor model on observed data and also used to determine the strength of the relationship between *y* and *X* variables.

Trend lines are often calculated using linear regression.

The technique provides a way to determine patterns in the data set and model the data so that new values can be predicted.

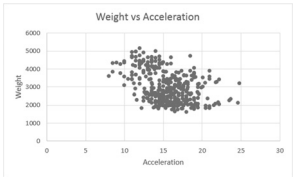
Regression in Excel

Excel provides a regression function that will calculate:

- R^2
- ANOVA table
- regression equation coefficients
- standardized and unstandardized residuals

Example: Given a data set of car weight and acceleration, determine if there is any relationship between them.

Scatterplot shows weak relationship with no strong patterns, and we would expect to see this shown in the regression analysis.



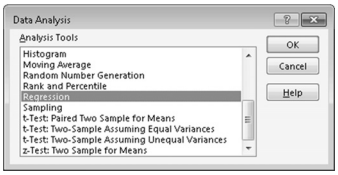
Regression Example

Regression computes constants *m* and *b* in formula:

weight = *m****acceleration** + *b*

Weight is the dependent variable and acceleration is the independent variable.

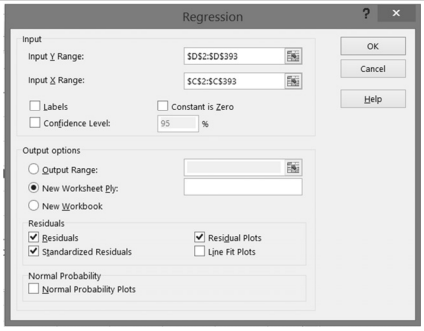
To start select, `Data Analysis` from the data tab and then select `Regression` and `OK`.



Regression Example Settings

Settings:

- Response (dependent) data for the Input Y Range
- Columns for the explanatory (independent) data (X Range).
- For residual information select, Residuals, Standardized Residuals, and Residual Plots from the Residuals section.



Regression Example Results

SUMMARY OUTPUT					
Regression Statistics					
Multiple R	0.41683992				
R Square	0.17375492				
Adjusted R Square	0.17160583				
Standard Error	2.510965983				
Observations	392				
ANOVA					
	df	*SS*	*MS*	*F*	*Significance F*
Regression	1	517.0999442	517.0999442	82.01491373	6.56562E-18
Residual	390	2458.930566	6.304950169		
Total	391	2976.03051			
	Coefficients	*Standard Error*	*t Stat*	*P-value*	*Lower 95%*	*Upper 95%*	*Lower 95.0%*	*Upper 95.0%*
Intercept	19.57266581	0.462860061	42.28635703	9.5445E-148	18.66265269	20.48267893	18.66265269	20.48267893
X Variable 1	-0.001353806	0.000149499	-9.056208574	6.56562E-18	-0.001647821	-0.001059971	-0.001647821	-0.001059971

R^2 * 100% = percentage of variation in dependent variable explained by independent variable

Coefficients for the regression equation

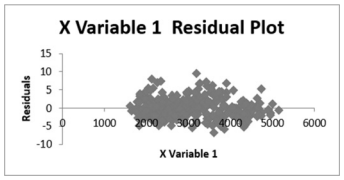
All of the output is put into a new sheet. Read the values off of the table and form the regression equation:

- weight = -0.001*acceleration + 19.572

Regression Example Results (cont.)

Below the previous tables are the predicted y values (from the regression equation) as well as the residuals and standardized residuals. All plots are placed to the right of the charts.

Observation	*Predicted Y*	*Residuals*	*Standard Residuals*
1	14.82861427	-2.828614269	-1.127947728
2	14.57272793	-3.072727927	-1.22529131
3	14.9206792	-3.920679196	-1.563423207
4	14.92474088	-2.924740884	-1.1662795
5	14.90307855	-4.403078548	-1.755786393
6	13.69540333	-3.695403327	-1.473591445
7	13.67780268	-4.67780268	-1.865336311
8	13.73466631	-5.234666311	-2.087393123
9	13.58167606	-3.581676065	-1.428241179
10	14.36016626	-5.860166257	-2.336819583
11	14.74873441	-4.748734406	-1.893621284

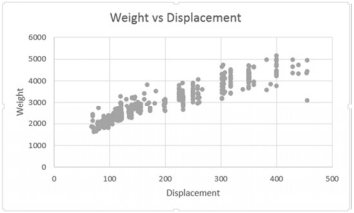


Expected a weak relationship and this is demonstated by the R^2 value.

- Only 17.4% of the variation in weight is explained by acceleration.

Try It: Regression

Question: Perform a regression analysis between weight (dependent) and displacement (independent) variable.



Conclusion

Spreadsheets are general purpose tools for data analysis that consist of a table of cells which contain data and formulas.

Formulas contain data values, cell references, and functions.

- Aggregate functions summarize multiple data values into a single value.
- Functions exist for statistics, string manipulation, lookup/indexing, and decisions.

Spreadsheets provide tools for data sorting, filtering, visualization using charts, and summarization (pivot tables).

- Also contain tools for what-if scenarios, goal seek, linear solvers, and statistical analysis tools.

Objectives

- Explain what a spreadsheet is.
- Explain how cells are addressed in a spreadsheet.
- List some of the ways to select cells in a spreadsheet.
- Define and explain: formula, function, argument, concatenation
- Use these functions: concatenate, lookup, index
- Explain the difference between an absolute and relative address.
- Explain how an aggregate function works. List some examples.
- Explain how to use conditional formatting.
- Explain how spreadsheets can be used as a database. Use sorting and filtering.
- Be able to create and edit charts and use chart features: trendlines, sparklines
- Explain the usefulness of: what-if scenarios, goal seek, solver
- Use and create pivot tables and charts.
- Evaluate and create conditions. Use IF() to make decisions.

DATA 301
Introduction to Data Analytics
Microsoft Excel VBA

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why Microsoft Excel Visual Basic for Applications?

Microsoft Excel VBA allows for automating tasks in Excel and provides a full programming environment for data analysis.

Excel VBA is commonly used in high finance and frequency trading applications for creating and validating financial models.

Using Excel VBA will be our first practice with programming and allow us to explore fundamental programming concepts of commands, variables, decisions, repetition, objects, and events.

Excel Visual Basic for Applications (VBA)

Visual Basic for Applications (VBA) is a programming language allowing users to build their own functions, automate tasks in Microsoft Office, and develop customized code.

The language has been part of almost all versions of Office for over 20 years.

VBA allows for expanding the capabilities of Excel and adding user-interface elements (buttons, lists) to your spreadsheet.

Macros

A ***macro*** is a recorded set of actions that is saved so that they can be easily executed again.

If you do the same set of actions *repetitively*, then creating a macro allows for doing all those actions with one command.

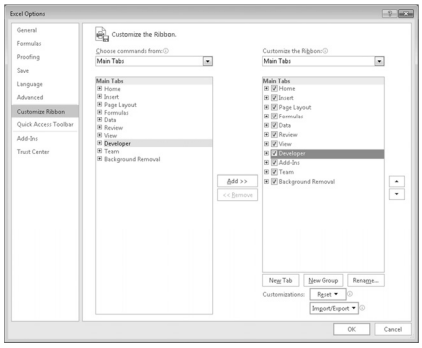
Macros are accessible under the `View` tab in the `Macros` group or the `Developer` tab.

Macros are converted into VBA programs.

Developer Tab

The ***`Developer`*** tab contains icons for performing VBA and macro development.

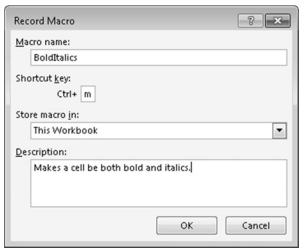
To add the `Development` tab, go to `File`, `Options`, `Customize Ribbon` and make sure it is checked beside `Developer`.



Recording a Macro

To record a macro, under `View` select, `Macros` -> `Record Macro`.

- Excel will record your actions until you select `Stop Recording`.
  - Note: Cursor movement is not captured.



Macro names cannot contain spaces or begin with a number.

It is recommended to use `Ctrl+Shift+`*`Key`* for a Shortcut key so that you do not override built-in shortcuts.

Macros can be created in a given workbook or a Personal Workbook allowing them to be used in multiple workbooks.

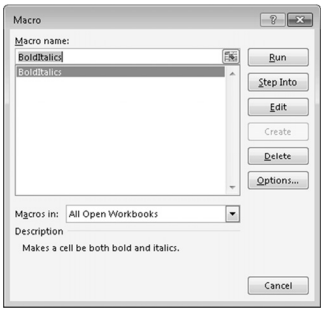
Using a Macro

Use a macro in the following ways:

1) With the shortcut key if defined

2) Assign a macro to a button or on the toolbar

3) Under `Macros`, Select `View Macros` then pick the macro and `Run`.



Macros Question

Question: Select a **TRUE** statement.

A) A macro can be created without assigning it a shortcut key.

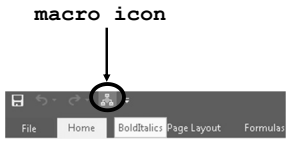
B) A macro will record cursor movements.

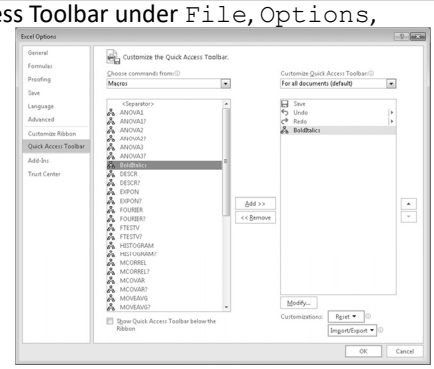
C) Macros can be created in an individual workbook or in a personal macro workbook so they can be used in multiple workbooks.

D) A macro can have only one command it executes.

Adding Macro to Quick Access Toolbar

Add a macro to The Quick Access Toolbar under `File`, `Options`, `Quick Access Toolbar`.

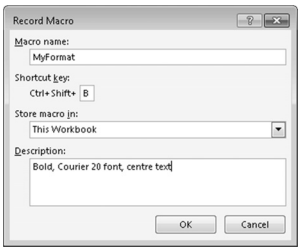




Try it: Macros

Question: Create a macro that does the following tasks:

- Bolds the cell and makes the font Courier 20.
- Sets the cell background to orange.
- Centers the text in the cell.
- Use a shortcut of `Ctrl`+`Shift`+`b`.
- Add it to the Quick Access Toolbar.

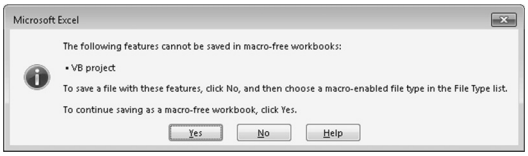


Try-out your macro using the shortcut key, toolbar, and from the macro dialog.

Saving Workbook with Macros

Excel now forces workbooks with macros to be saved in Excel Macro-Enabled Workbook (`*.xlsm`) format.

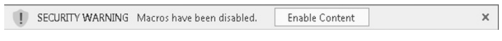
Saving a workbook with macros in regular format gives this error:



Macro Security

Since macros can execute any code, they have been a target for virus writers. Understanding the source of the Excel spreadsheet that contains macros is important when deciding to run them or not.

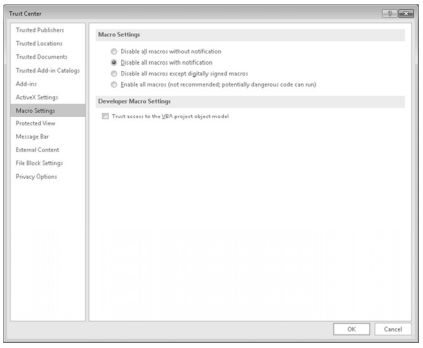
Excel has ***macro security settings*** to allow you to enable or disable running macros. Spreadsheets with macros often will generate a warning when opening them:



Macro Security Settings

The default security is **Disable all macros with notification** that prevents macros from running but displays a warning allowing you to enable them.

One of the biggest issues with macros is security and making sure you are only using macros from a trusted source.

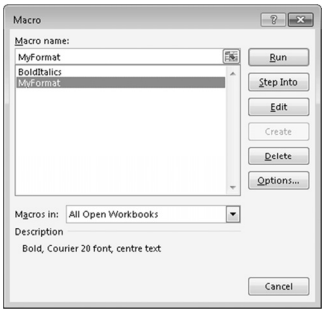


Macros: Implementation

Macros are converted to Visual Basic code.

Can edit macro code and create your own code.

Under the `Developer` tab, select `Macros` then `Edit` macro to modify the code.

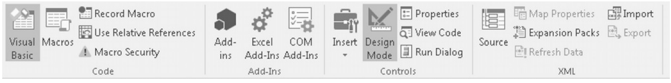


Visual Basic Editor

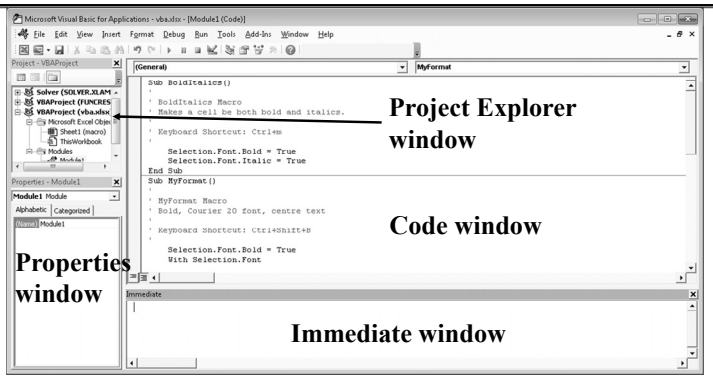
Visual Basic Editor (VBE) allows editing visual basic code and is a complete integrated development environment (IDE).

Users can create and edit macros as well as other Visual Basic code with the editor.

To open the VBE, under `Developer` tab -> `Visual Basic` or `Alt+F11`.

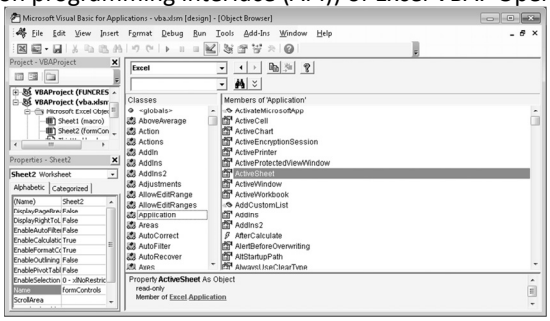


Visual Basic Editor Screenshot



Object Browser

Object browser allows for exploring objects and methods (the application programming interface (API)) of Excel VBA. Open with `F2`.



Macro Code in Visual Basic Editor

Subroutine with name and no arguments

```
Sub BoldItalics()

' BoldItalics Macro
' Makes a cell be both bold and italics.
'
' Keyboard Shortcut: Ctrl+m
'
    Selection.Font.Bold = True
    Selection.Font.Italic = True
End Sub
```

Comments start with '

Every statement is on its own line.

Dot notation to separate "items" (objects, methods, properties).

`WITH` Statement in Visual Basic Code

```
Sub MyFormat()
'
' MyFormat Macro
' Bold, Courier 20 font, centre text
'
' Keyboard Shortcut: Ctrl+Shift+B
'
    Selection.Font.Bold = True
    With Selection.Font
        .Name = "Courier New"
        .Size = 20
        .Strikethrough = False
        .Superscript = False
        .Subscript = False
        .OutlineFont = False
        .Shadow = False
        .Underline = xlUnderlineStyleNone
        .ThemeColor = xlThemeColorLight1
        .TintAndShade = 0
        .ThemeFont = xlThemeFontNone
    End With
    With Selection.Interior
        .Pattern = xlSolid
        .PatternColorIndex = xlAutomatic
        .ThemeColor = xlThemeColorAccent2
        .TintAndShade = 0
        .PatternTintAndShade = 0
    End With
```

`WITH` syntax simplifies typing same object many times.

These lines all apply to `Selection.Font`.

Visual Basic Editor: Immediate Window

The Immediate window allows entering of single line commands.

- Use `PRINT` or `?`
- In code, use `Debug.Print` to print to immediate window.

```
Immediate
? "Hello world!"
Hello world!
? Range("A2").Value
 1
Range("A2").Value = 10
? Range("A2").Value
 10
```


Try it: Immediate Window

Question: Try do these actions using the immediate window:

- 1) Print `"Hey There!"`
- 2) Calculate the answer of `765 * 39`.
- 3) Select a cell then call the macro `RedItalics`.
- 4) Change the value of cell `B4` to `"DATA"`.
- 5) Change the value of cell `A6` to `100`.

Challenge Try it: Create Macro in VBE

Question: Copy the `MyFormat` macro and edit to produce a new macro called `RedUnderline` that:

- Underlines the text in the cell.
- Makes the cell background red.
- If the cell was bold or italics before, resets to not have bold and italics.

Hints:

- Underline property in Excel is `Font.Underline` and can set to constant `xlUnderlineStyleSingle`.
- Can change background color with `Interior.Color` and set to `RGB(`*`redValue`*`,` *`greenValue`*`,` *`blueValue`*`)` where the color values are numbers from 0 to 255.

Introduction to Programming

An ***algorithm*** is a precise sequence of steps to produce a result. A ***program*** is an encoding of an algorithm in a ***language*** to solve a particular problem.

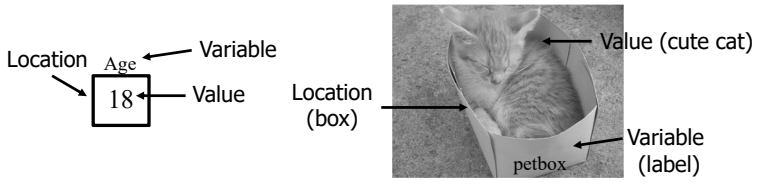
There are numerous languages that programmers can use to specify instructions. Each language has its different features, benefits, and usefulness.

- We will start with Excel VBA but also study Python and R.

The goal is to understand fundamental programming concepts that apply to all languages.

Variables

A ***variable*** is a name that refers to a location that stores a data value.



IMPORTANT: The *value* at a location can change using initialization or assignment.

Variable Assignment

Assignment using an **=** sets the value of a variable.

Example:

```
num = 10
num = Range("A1").Value
num = 20
```


Excel Variables

Every variable in Excel has a ***name*** and a ***data type***.

- Variables increase code efficiency and readability.
- Data types: `Boolean`, `Currency`, `Date`, `Double`, `Integer`, `Long`, `Object`, `String`, `Variant` (any type)

Example:

```
Dim num As Integer
```


Collections

Collections are variables that store multiple data items. Data items can either be indexed (selected) by name or number.

Example:

```
Worksheets("macro")
Worksheets(2)
```

`Worksheets` is a collection as there may be multiple worksheets in the workbook. Select one by name or number (starting with 1).

Variables Question

Question: How many of the following statements are **TRUE**?

1) A variable name cannot change during a program.
2) A variable value cannot change during a program.
3) A collection is a variable that can store multiple data items.
4) A value in a collection can be retrieved by name or by index starting from 0.
5) In Excel, variables are declared using DIM.
6) In Excel, variables are declared with a data type.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Decisions

Decisions allow the program to perform different actions in certain conditions.

- Logical operators: `AND`, `OR`, `NOT`

Excel decision syntax:

```
If condition Then
   statement
End If
```

```
If condition Then
   statement
Else
   statement
End If
```


Question: Decisions

Question: What is the output of the following code?

```
Sub TryIf()
    Dim num As Integer
    num = 20
    If num > 10 Then
        Debug.Print num * 5
    Else
        Debug.Print num * 2
    End If
End Sub
```

A) 100 **B)** 40 **C)** 20 **D)** error – no output

Try it: Decisions

Question: Create a method called `EchoDecision` that asks user a Yes and No question and outputs a message either "`Yes`" or "`No`" depending on what they chose.

```
Sub EchoDecision()
    Dim answer As Integer
    answer = MsgBox("Pick Yes or No!", vbYesNo)
    ' answer will either be vbYes or vbNo
    ' Use debug.print to output "Yes" or "No"
End Sub
```


Loops and Iteration

A ***loop*** repeats a set of statements multiple times until some condition is satisfied.

- Each time a loop is executed is called an ***iteration***.
- A `for` loop repeats statements a given number of times.

Excel `for` loop syntax:

```
Dim i as Integer
For i = 1 To 5
   Debug.Print i
Next i
```


Question: Loops

Question: How many numbers are printed with this loop?

```
Sub TestFor()
    Dim i As Integer

    For i = 0 To 10
        Debug.Print i
    Next i
End Sub
```

A) 11 **B)** 10 **C)** 0 **D)** error – no output

Try it: Loops

Question: Create a method called `TryFor` that prints the numbers 1 to 20. Challenging variants:

- Print the numbers from 10 down to 1.
- Print only the even numbers from 1 to 10.

User-Defined Functions (UDFs)

A ***user-defined function*** is your own Excel function that can be used in formulas like built-in functions.

A UDF must return a number, string, array, or Boolean.

A UDF cannot change the Excel environment including the current cells or other cells (e.g. change formatting).

UDF Example

UDF `doubleIt` will double the input argument.

```
' UDF expect a number to double
Function doubleIt(num As Integer)
    ' To return a value, assign the value to the method name
    doubleIt = num * 2
End Function
```

UDF Example – Sum Cells by Background Color

```
' Sums all the cells with the same color
Function SumColor(RangeToSum As Range, ColorID As Integer) As Long
  Dim ColorCell As Range
  Dim result As Long

  ' Loop through each cell in the range.
  For Each ColorCell In RangeToSum
    If ColorCell.Interior.ColorIndex = ColorID Then
        result = result + ColorCell.Value
    End If
  Next ColorCell

  SumColor = result
End Function
```


Try it: UDF

Question: Create a UDF called `CountNum` that will return a count of the number of digits (0 to 9) in a string.

Advanced: Object-Oriented Programming

Object-oriented programming structures code as object, classes, methods, and properties. This organization makes it easier to understand and construct large programs.

An ***object*** is an instance of a class that has its own properties and methods that define what the object is and what it can do.

A ***class*** is a generic template (blueprint) for creating an object. All objects of a class have the same methods and properties (although the property values can be different).

A ***property*** is an attribute or feature of an object.

A ***method*** is a set of statements that performs an action.

Excel Objects

Excel structures everything as a hierarchy of objects, and commands are done by running a method of an object.

An object may contain other objects as well as methods and properties. A dot "`.`" is used as a separator between objects and subobjects, methods, and properties.

Examples:

- Top-level object: `Application`
- `Workbook` – individual Excel file
- `Worksheet` - sheet in a workbook

```
Application.ActiveWorkbook.Worksheets("macro").Range("A1").Value
```


Excel Range Object

The `Range` object selects a cell or group of cells.

Example:

```
Worksheets("Sheet1")
  .Range("A1:C3").Font.Italic = True
```


Excel Object Methods

Methods perform an action.

Example:

```
Worksheets("macro").Activate
```

Object-Oriented Question

Question: How many of the following statements are **TRUE**?

1) A method can have no parameters.
2) Two objects of the same class have the same properties.
3) Two objects of the same class may have different values for their properties.
4) `Workbook` is the top-level object in Excel.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Excel Objects

Question: Using the Immediate window try to perform the following actions with methods on Excel objects:

1) Switch the active worksheet to form.
2) Switch the active cell to macro sheet A4.
3) Use `msgbox` to display value in current cell (`ActiveCell`).

Forms and Input Controls

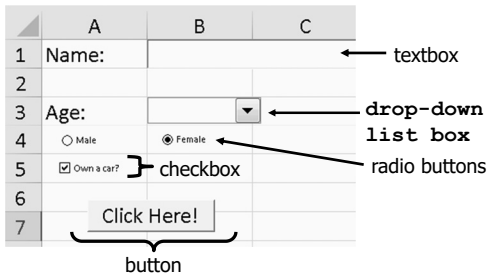
Excel allows the creation of forms with controls for a better interface.

Two types of controls in Excel:

- Form controls – default
- ActiveX controls – allow more flexibility and customization

Controls can be inserted from the Developer tab. Select `Insert`, pick control, and then click and drag the size and shape of the control on the spreadsheet.

Input Controls



Events

An ***event*** is a notification to your program that something has occurred.

Events in Excel:

- add a worksheet
- double-click on a cell
- change a cell value
- calculating a formula
- click on a button (can execute a macro)

Worksheet-level events on a particular worksheet and workbook level events for entire file.

Conclusion

Microsoft Excel VBA allows for automating tasks in Excel and provides a full programming environment for data analysis.

Macro record a set of actions so they can be easily executed again.

- Be aware of security risks when using macros.

The ***Visual Basic Editor (VBE)*** is a complete integrated development environment for editing macros, ***user-defined functions***, and adding forms and controls that dynamically respond to events.

Excel VBA uses ***object-oriented programming*** that structures code as object, classes, methods, and properties. A developer can control and automate everything with Excel using VBA.

Objectives

- List some reasons to use Excel VBA
- Define macro and explain the benefit of using macros
- Be able to record and execute a macro
- Explain the security issues with macros and how Excel deals with them
- List and explain the use of the four main windows of the Visual Basic Editor
- Explain the role of the object browser
- Explain and use the `WITH` statement syntax
- Be able to write simple macros using the VBE
- Define: algorithm, program, language
- Define: object-oriented programming, object, class, property, method
- Understand and use dot-notation
- Use the Range object to select a group of cells
- Define: variable, value, location

Objectives (2)

- Create and use Excel variables
- Explain how a collection is different from a typical variable
- Use `If/Then/Else` syntax to make decisions
- Use `For` loop for repetition
- Create user-defined functions and use them in formulas
- Define: event
- List some typical user interface controls
- Understand that Excel allows for forms and controls to be added to a worksheet which respond to events

DATA 301
Introduction to Data Analytics
Relational Databases

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why Relational Databases?

Relational databases allow for the storage and analysis of large amounts of data.

Relational databases are the most common form of database used by companies and organizations for data management.

Since a significant amount of data is stored in relational databases, understanding how to create and query these databases using the SQL standard is a very valuable skill.

What is a database?

A ***database*** is a collection of logically related data for a particular domain.

A ***database management system*** (***DBMS***) is software designed for the creation and management of databases.

- e.g. Oracle, DB2, Microsoft Access, MySQL, SQL Server, MongoDB

Bottom line: A ***database*** is the *data* stored and a ***database system*** is the *software* that manages the data.

Databases in the Real-World

Databases are everywhere in the real-world even though you do not often interact with them directly.

- $40 billion dollar annual industry

Examples:

- Retailers manage their products and sales using a database.
  - Wal-Mart has one of the largest databases in the world!
- Online web sites such as Amazon, eBay, and Expedia track orders, shipments, and customers using databases.
- The university maintains all your registration information and marks in a database that is accessible over the Internet.

Can you think of other examples?

What data do you have?

Database System Properties

A database system provides *efficient*, *convenient*, and *safe multi-user* storage and access to *massive* amounts of *persistent* data.

Efficient - Able to handle large data sets and complex queries without searching all files and data items.

Convenient - Easy to write queries to retrieve data.

Safe - Protects data from system failures and hackers.

Massive - Database sizes in gigabytes, terabytes and petabytes.

Persistent - Data exists even if have a power failure.

Multi-user - More than one user can access and update data at the same time while preserving consistency.

The Relational Model: Terminology

The ***relational model*** organizes data into tables called relations.

- Developed by E. F. Codd in 1970 and used by most database systems.

Terminology:

A ***relation*** is a table with columns and rows.

An ***attribute*** is a named column of a relation.

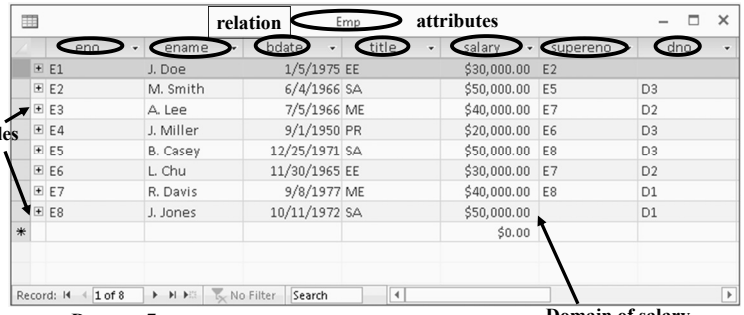
A ***tuple*** is a row of a relation.

A ***domain*** is a set of allowable values for one or more attributes.

The ***degree*** of a relation is the number of attributes it contains.

The ***cardinality*** of a relation is the number of tuples it contains.

Relation Example

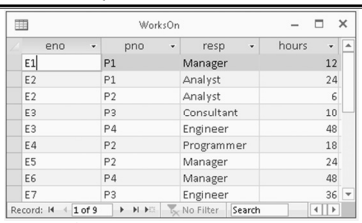


tuples

Degree =7
Cardinality = 8

Domain of salary is *currency*

Relation Practice Questions



1) What is the name of the relation?
2) What is the cardinality of the relation?
3) What is the degree of the relation?
4) What is the domain of `resp`? What is the domain of `hours`?

Database Definition Question

Question: How many of the following statements are **TRUE**?

1) A database is data.
2) A database system is software.
3) A database system will lose the data stored when the power is turned off.
4) Usually, more than one user can use a database system at a time.
5) The cardinality is the number of rows in a relation.
6) A relation's cardinality is always bigger than its degree.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Database Definition Matching Question

Question: Given the three definitions, select the ordering that contains their related definitions.

1) relation
2) tuple
3) attribute

A) column, row, table
B) row, column, table
C) table, row, column
D) table, column, row

Cardinality and Degree Question

Question: A database table has 5 rows and 10 columns. Select ***one*** true statement.

A) The table's degree is 50.
B) The table's cardinality is 5.
C) The table's degree is 5.
D) The table's cardinality is 10.

Creating and Using a Database

Typically, a data analyst will use an existing database. The database will already be created on a database system and contain data that was inserted and updated previously.

To use an existing database, the data analyst must be able to use the tools and languages to query the database. The standard is SQL.

Creating a large database is outside of the scope of this class, but we will learn how to create individual tables and load data into them which is a common data analysis task.

A Simple Query Language: Keyword Searching

Keyword (or English-language) ***search*** allows a user to type keywords or phrases and returns a best answer estimate.

This works fairly well for web searches, although we lack precision. Precision is required for many applications.

- Example: How would you return all employees with salary greater than 30,000 using keyword search?

SQL Overview

Structured Query Language or SQL is the standard database query language to retrieve *exact answers*.

- A SQL query specifies *what* to retrieve but not *how* to retrieve it.
- SQL is used by Microsoft Access and almost all other database systems.

Some basic rules for SQL statements:

- 1) There is a set of *reserved words* that cannot be used as names for database fields and tables.
  - `SELECT`, `FROM`, `WHERE`, etc.
- 2) SQL is generally *case-insensitive*.
  - Only exception is string constants. 'FRED' not the same as 'fred'.
- 3) SQL is *free-format* and white-space is ignored.

SQL `CREATE TABLE`

The **`CREATE TABLE`** command is used to create a table in the database. A table consists of a table name and a set of fields with their names and data types.

Example:

```
CREATE TABLE emp (
    eno       CHAR(5),
    ename     VARCHAR(30) NOT NULL,
    bdate     DATE,
    title     CHAR(2),
    salary    DECIMAL(9,2),
    supereno  CHAR(5),
    dno       CHAR(5),
    PRIMARY KEY (eno)
)
```

field must always have a value

Data Types:
CHAR(5) – always 5 chars long
VARCHAR(30) – up to 30 chars long
DECIMAL(9,2) – e.g. 1234567.99
DATE – e.g. 1998/01/18

What is a key?

A ***key*** is a set of attributes that uniquely identifies a tuple in a relation.

A key helps to identify a particular row (data item) and find it faster.

In the `emp` table, the key was `eno`. It was called the `primary key` because it was the main key used to find an employee in the table.

Question:

- What is a key to identify a student in this class?

Try it: `CREATE TABLE`

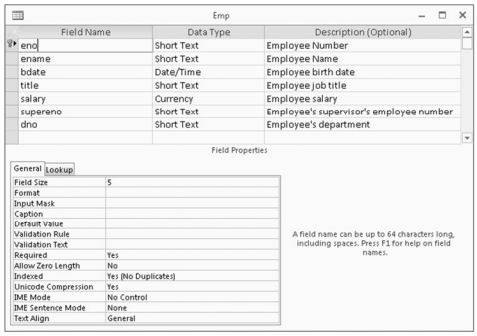
Question: Create a table called `mydata` that has three fields:

- `num` – that will store a number (use `int` as data type)
- `message` – that will store a string up to 50 characters (`varchar` data type)
- `amount` – that stores a decimal number with 8 total digits and 2 decimal digits (`decimal` data type)

Use the web site **sqlfiddle.com** to try your table creation.

`CREATE TABLE` in Microsoft Access

In Microsoft Access, use `Create` -> `Table` to build a table.



Try it: `CREATE TABLE` in Microsoft Access

Question: Create a table called `mydata` that has three fields:

- `num` – that will store a number (use `Number` as data type)
- `message` – that will store a string up to 50 characters (`Short Text` data type)
- `amount` – that stores a decimal number with 8 total digits and 2 decimal digits (`Currency` data type)

Build the table using the Microsoft Access user interface.

Schemas and Metadata

Creating tables defines the structure of the database.

The description of the structure of the database is called a ***schema***.

The schema is a type of ***metadata***.

`DROP TABLE`

The command **`DROP TABLE`** is used to delete the table and *all its data* from the database:

Example:

```
DROP TABLE emp;
```

- Note: The database does not confirm if you really want to drop the table and delete its data. The effect of the command is immediate.

`CREATE TABLE` Question

Question: How many of the following statements are **TRUE**?

1) Each field in the `CREATE TABLE` statement is separated by a comma.
2) The data type for a field is optional.
3) You can create two tables in a database with the same name.
4) A table will not be dropped (with `DROP TABLE`) if it contains data.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Adding Data using `INSERT`

Insert a row using the `INSERT` command:

```
INSERT INTO emp VALUES ('E9','S. Smith','1975-03-05',
                        'SA',60000,'E8','D1')
```

Fields: eno, ename, bdate, title, salary, supereno, dno

If you do not give values for all fields in the order they are in the table, you must list the fields you are providing data for:

```
INSERT INTO emp(eno, ename, salary)
       VALUES ('E9','S. Smith',60000)
```

Note: If any columns are omitted from the list, they are set to `NULL` (empty).

Try it: `INSERT`

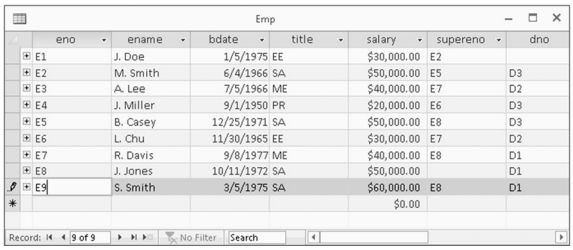
Question: Using the `mydata` table insert three rows:

- `(1, 'Hello', 99.45)`
- `(2, 'Goodbye', 55.99)`
- `(3, 'No Amount')`

Use the web site **sqlfiddle.com** to try your table creation.

Adding Data using `INSERT` in Microsoft Access

In Microsoft Access, insert a new row by entering data into the last row of the table when in data view.



Emp

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1/5/1975	EE	$30,000.00	E2	
E2	M. Smith	6/4/1966	SA	$50,000.00	E5	D3
E3	A. Lee	7/5/1966	ME	$40,000.00	E7	D2
E4	J. Miller	9/1/1950	PR	$20,000.00	E6	D3
E5	B. Casey	12/25/1971	SA	$50,000.00	E8	D3
E6	L. Chu	11/30/1965	EE	$30,000.00	E7	D2
E7	R. Davis	9/8/1977	ME	$40,000.00	E8	D1
E8	J. Jones	10/11/1972	SA	$50,000.00		D1
E9	S. Smith	3/5/1975	SA	$60,000.00	E8	D1
				$0.00		

Record: 9 of 9 | No Filter | Search

Try it: `INSERT` in Microsoft Access

Question: Using the `mydata` table insert three rows in Access:

- `(1, 'Hello', 99.45)`
- `(2, 'Goodbye', 55.99)`
- `(3, 'No Amount')`

`UPDATE` Statement

Updating existing rows using the `UPDATE` statement. Examples:

- 1) Increase all employee salaries by 10%.

```
UPDATE emp SET salary = salary*1.10;
```

- 2) Increase salary of employee E2 to $1 million and change his name:

```
UPDATE emp SET salary = 1000000, name='Rich Guy'
WHERE eno = 'E2';
```

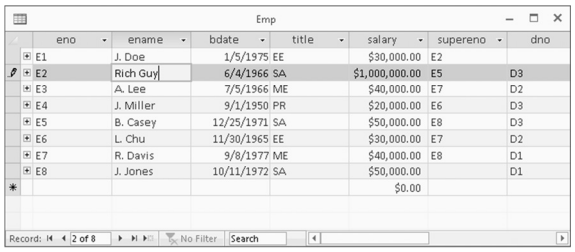
Notes:

- May change (`SET`) more than one value at a time. Separate by commas.
- Use `WHERE` to filter only the rows to update.

Updating Data in Microsoft Access

`UPDATE` command supported by Microsoft Access.

To modify individual data items, select the row and cell to update and change the data. Data is saved when you leave the row.



Emp

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	1/5/1975	EE	$30,000.00	E2	
E2	Rich Guy	6/4/1966	SA	$1,000,000.00	E5	D3
E3	A. Lee	7/5/1966	ME	$40,000.00	E7	D2
E4	J. Miller	9/1/1950	PR	$20,000.00	E6	D3
E5	B. Casey	12/25/1971	SA	$50,000.00	E8	D3
E6	L. Chu	11/30/1965	EE	$30,000.00	E7	D2
E7	R. Davis	9/8/1977	ME	$40,000.00	E8	D1
E8	J. Jones	10/11/1972	SA	$50,000.00		D1
				$0.00		

Record: 2 of 8 | No Filter | Search

Try it: `UPDATE`

Question: Using the `mydata` table and the three rows previously inserted do these updates:

- Update all `amount` fields to be `99.99`.
- Update the `num` field and set it to 10 for the record with `num = 1`.
- Update the `message` field to `'Changed'` for the record with `num = 2`.

You can use Access or sqlfiddle.com.

`DELETE` Statement

Rows are deleted using the `DELETE` statement. Examples:

- 1) Fire everyone in the company.

```
DELETE FROM emp;
```

- 2) Fire everyone making over $35,000.

```
DELETE FROM emp
WHERE salary > 35000;
```

Deleting Data in Microsoft Access

`DELETE` command supported by Microsoft Access. To delete an individual row, select the row to delete and press `Delete` key or select `Delete Record` from pop-up menu.

Try it: `DELETE`

Question: Using the `mydata` table and the three rows previously inserted do these deletes:

- Delete the row with `num = 1`.
- Delete the row(s) with `message > 'C'`.
- Delete all rows.

You can use Access or sqlfiddle.com.

`INSERT` Question

Question: How many of the following statements are **TRUE**?

1) You must always specify the fields being inserted with `INSERT` statement.
2) If you list the fields, the fields must be in the same order as the table.
3) If you do not provide a value for a number field, it will default to 1.
4) Number data items are enclosed in single quotes.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

`UPDATE` Question

Question: How many of the following statements are **TRUE**?

1) You may update more than one row at a time.
2) If the `UPDATE` has no `WHERE` clause, it always updates all rows.
3) You may update zero or more rows using a `UPDATE` statement.
4) `UPDATE` may change more than one data value (column) in a row.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

`DELETE` Question

Question: How many of the following statements are **TRUE**?

1) A `DELETE` with no `WHERE` clause will delete all rows.
2) `DELETE` statement is case-sensitive.
3) It is possible to `DELETE` zero or more rows using a `WHERE` clause.
4) A `DELETE` statement may delete zero rows when executed.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

SQL Queries using `SELECT`

A query in SQL has the form:

SELECT (list of columns or expressions)
FROM (list of tables)
WHERE (filter *conditions*)
GROUP BY (columns)
ORDER BY (columns)

Notes:

- 1) Separate the list of columns/expressions and list of tables by **commas**.
- 2) The "`*`" is used to select all columns.
- 3) Only `SELECT` required. `FROM`, `WHERE`, `GROUP BY`, `ORDER BY` are optional.

Example Data

emp Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

proj Table

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

workson Table

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

dept Table

dno	dname	mgreno
D1	Management	E8
D2	Consulting	E7
D3	Accounting	E5
D4	Development	null

SQL: Retrieving Only Some of the Columns

The ***projection operation*** creates a new table that has some of the columns of the input table. In SQL, provide the table in the `FROM` clause and the fields in the output in the `SELECT`.

Example: Return only the `eno` field from the `Emp` table:

```
SELECT eno
FROM   emp
```

emp Table

eno	ename	bdate	title	salary	supereno	dno
E1	J. Doe	01-05-75	EE	30000	E2	null
E2	M. Smith	06-04-66	SA	50000	E5	D3
E3	A. Lee	07-05-66	ME	40000	E7	D2
E4	J. Miller	09-01-50	PR	20000	E6	D3
E5	B. Casey	12-25-71	SA	50000	E8	D3
E6	L. Chu	11-30-65	EE	30000	E7	D2
E7	R. Davis	09-08-77	ME	40000	E8	D1
E8	J. Jones	10-11-72	SA	50000	null	D1

→

Result

eno
E1
E2
E3
E4
E5
E6
E7
E8

SQL Projection Examples

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT eno,ename
FROM   emp
```

eno	ename
E1	J. Doe
E2	M. Smith
E3	A. Lee
E4	J. Miller
E5	B. Casey
E6	L. Chu
E7	R. Davis
E8	J. Jones

```
SELECT title
FROM   emp
```

title
EE
SA
ME
PR
SA
EE
ME
SA

Notes: 1) Duplicates are not removed during SQL projection.
2) `SELECT *` will return all columns.

Projection Question

Question: Given this table and the query:

```
SELECT eno, ename, salary
FROM   emp
```

How many columns are returned?

A) 0

B) 1

C) 2

D) 3

E) 4

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Projection Question #2

Question: Given this table and the query:

```
SELECT salary
FROM   emp
```

How many rows are returned?

A) 0

B) 2

C) 4

D) 8

emp Table

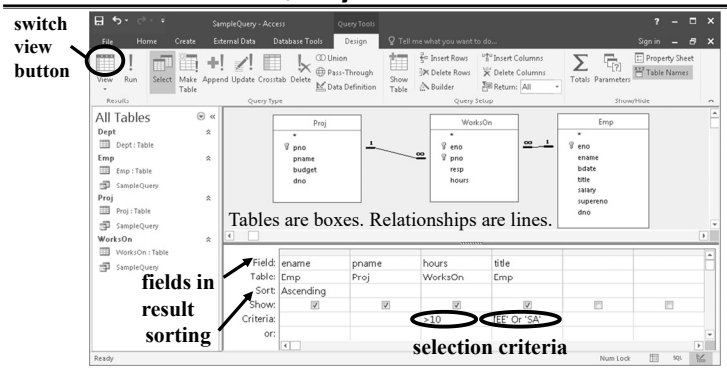
eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Building a `SELECT` SQL Query in Microsoft Access

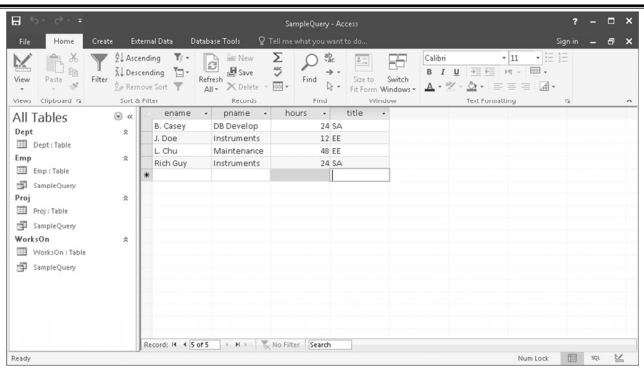
Under `Create Tab`, click on `Query Design`.

Access will pop-up a window asking what table(s) you wish to query. Select one or more.

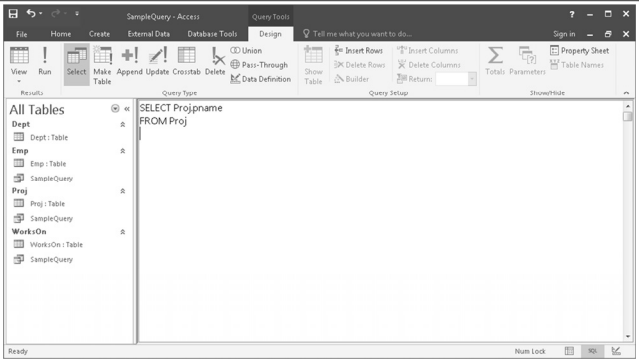
Microsoft Access Query Interface



Microsoft Access Data Sheet View

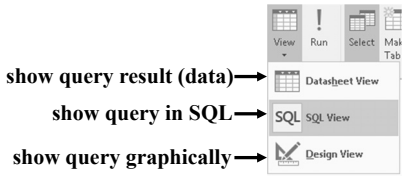


Microsoft Access SQL Design View



Microsoft Access Query Views

You may view your data, your query graphically, or your query in SQL.



Try it: SQL `SELECT` and Projection

Question: Using the `proj` table, write these three queries:

- Show all rows and all columns.
- Show all rows but only the `pno` column.
- Show all rows but only the `pno` and `budget` columns.

You can use Access or sqlfiddle.com.

Retrieving Only Some of the Rows

The ***selection operation*** creates a new table with some of the rows of the input table. A condition specifies which rows are in the new table. The condition is similar to an `if` statement.

Example: Return the projects in department `'D2'`:

```
SELECT pno, pname, budget, dno
FROM   proj
WHERE  dno = 'D2';
```

`proj` Table

pno	pname	budget	dno
P1	Instruments	150000	D1
P2	DB Develop	135000	D2
P3	Budget	250000	D3
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Result

pno	pname	budget	dno
P2	DB Develop	135000	D2
P4	Maintenance	310000	D2
P5	CAD/CAM	500000	D2

Algorithm: Scan each tuple and check if matches condition in WHERE clause.

Selection Conditions

The condition in a selection statement specifies which rows are included. It has the general form of an if statement.

The condition may consist of attributes, constants, comparison operators (<, >, =, !=, <=, >=), and logical operators (AND, OR, NOT).

SQL Selection Examples

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

```
SELECT *
FROM   emp
WHERE  title = 'EE'
```

eno	ename	title	salary
E1	J. Doe	EE	30000
E6	L. Chu	EE	30000

```
SELECT eno, ename, title, salary
FROM   emp
WHERE  salary > 35000 OR
       title = 'PR'
```

eno	ename	title	salary
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question

Question: Given this table and the query:

```
SELECT *
FROM   emp
WHERE  title='SA'
```

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question #2

Question: Given this table and the query:

```
SELECT *
FROM   emp
WHERE  salary > 50000 or title='PR'
```

How many rows are returned?

A) 0

B) 1

C) 2

D) 3

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Selection Question #3

Question: Given this table and the query:

```
SELECT *
FROM   emp
WHERE  salary > 50000 or title='PR'
```

How many columns are returned?

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

emp Table

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

Try it: SQL SELECT and Filtering Rows

Question: Using the proj table, write these three queries:

- Return all projects with budget > $250000.
- Show the pno and pname for projects in dno = 'D1'.
- Show pno and dno for projects in dno='D1' or dno='D2'.

You can use Access or sqlfiddle.com.

Join Example for Combining Tables

A ***join*** combines two tables by matching columns in each table.

workson Table

eno	pno	resp	dur
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P4	Engineer	48
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P4	Engineer	23

proj Table

pno	pname	budget
P1	Instruments	150000
P2	DB Develop	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

```
SELECT *
FROM   WorksOn INNER JOIN Proj
  ON   WorksOn.pno = Proj.pno
```

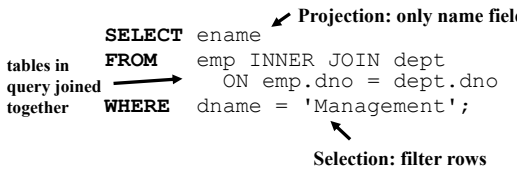
eno	pno	resp	dur	P.pno	pname	budget
E1	P1	Manager	12	P1	Instruments	150000
E2	P1	Analyst	24	P1	Instruments	150000
E2	P2	Analyst	6	P2	DB Develop	135000
E3	P4	Engineer	48	P4	Maintenance	310000
E5	P2	Manager	24	P2	DB Develop	135000
E6	P4	Manager	48	P4	Maintenance	310000
E7	P3	Engineer	36	P3	CAD/CAM	250000
E7	P4	Engineer	23	P4	Maintenance	310000

Join Query with Selection Example

You can use join, selection, and projection in the same query.

- Recall: Projection returns columns listed in `SELECT`, selection filters out rows using condition in `WHERE`, and join combines tables in `FROM` using a condition.

Example: Return the employee names who are assigned to the 'Management' department.



Result

ename
R. Davis
J. Jones

Ordering Result Data

The query result returned is not ordered on any column by default. We can order the data using the **`ORDER BY`** clause:

```
SELECT   ename, salary, bdate
FROM     emp
WHERE    salary > 30000
ORDER BY salary DESC, ename ASC;
```

- '`ASC`' sorts the data in ascending order, and '`DESC`' sorts it in descending order. The default is '`ASC`'.
- The order of sorted attributes is significant. The first column specified is sorted on first, then the second column is used to break any ties, etc.

`LIMIT` and `OFFSET`

If you only want the first *N* rows, use a **`LIMIT`** clause:

```
SELECT   ename, salary FROM emp
ORDER BY salary DESC LIMIT 5
```

To start from a row besides the first, use **`OFFSET`**:

```
SELECT   eno, salary FROM emp
ORDER BY eno DESC
LIMIT 3  OFFSET 2
```

- `LIMIT` improves performance by reducing amount of data processed and sent by the database system.
- `OFFSET` 0 is first row, so `OFFSET` 2 would return the 3rd row.
- `LIMIT`/`OFFSET` syntax supported differently by systems.
- For Access, use `SELECT TOP 5 eno, salary FROM emp`

Try it: SQL `SELECT` with Joins and Ordering

Question: Write these three queries:

- Return all projects with `budget` < $`500000` sorted by `budget` descending.
- List only the top 5 employees by `salary` descending. Show only their `name` and `salary`.
- List each project `pno`, `dno`, `pname`, and `dname` ordered by `dno` ascending then `pno` ascending. Only show projects if department `name > 'D'`. Note: This query will require a join.

You can use Access or sqlfiddle.com.

Aggregate Queries and Functions

Several queries cannot be answered using the simple form of the `SELECT` statement. These queries require a summary calculation to be performed. Examples:

- What is the maximum employee salary?
- What is the total number of hours worked on a project?
- How many employees are there in department 'D1'?

To answer these queries requires the use of aggregate functions. These functions operate on a single column of a table and return a single value.

Aggregate Functions

Five common aggregate functions are:

- `COUNT` - returns the # of values in a column
- `SUM` - returns the sum of the values in a column
- `AVG` - returns the average of the values in a column
- `MIN` - returns the smallest value in a column
- `MAX` - returns the largest value in a column

Notes:

- 1) `COUNT`, `MAX`, and `MIN` apply to all types of fields, whereas `SUM` and `AVG` apply to only numeric fields.
- 2) Except for `COUNT(*)` all functions ignore nulls. `COUNT(*)` returns the number of rows in the table.
- 3) Use `DISTINCT` to eliminate duplicates.

Aggregate Function Example

Return the number of employees and their average salary.

```
SELECT COUNT(eno) AS numEmp, AVG(salary) AS avgSalary
FROM   emp
```

Result

numEmp	avgSalary
8	38750

Note: `AS` is used to rename a column in the output.

GROUP BY Clause

Aggregate functions are most useful when combined with the `GROUP BY` clause. The **`GROUP BY`** clause groups rows based on the values of the columns specified.

When used in combination with aggregate functions, the result is a table where each row consists of unique values for the group by attributes and the result of the aggregate functions applied to the rows of that group.

GROUP BY Example

For each employee title, return the number of employees with that title, and the minimum, maximum, and average salary.

```
SELECT   title, COUNT(eno) AS numEmp,
         MIN(salary) as minSal,
         MAX(salary) as maxSal, AVG(salary) AS avgSal
FROM     emp
GROUP BY title
```

Result

title	numEmp	minSal	maxSal	avgSal
EE	2	30000	30000	30000
SA	3	50000	50000	50000
ME	2	40000	40000	40000
PR	1	20000	20000	20000

GROUP BY Facts

1) You can group by multiple attributes. To be in the same group, all attribute values must be the same.

2) Any `WHERE` conditions are applied before the `GROUP BY` and aggregate functions are calculated.

3) A column name cannot appear in the `SELECT` part of the query unless it is part of an aggregate function or in the list of group by attributes.

4) There is a `HAVING` clause that is applied *AFTER* the `GROUP BY` clause and aggregate functions are calculated to filter out groups. (We will not study that.)

GROUP BY Question

Question: Given this table and the query:

```
SELECT title, SUM(salary)
FROM   emp
GROUP BY title
```

Emp Relation

eno	ename	title	salary
E1	J. Doe	EE	30000
E2	M. Smith	SA	50000
E3	A. Lee	ME	40000
E4	J. Miller	PR	20000
E5	B. Casey	SA	50000
E6	L. Chu	EE	30000
E7	R. Davis	ME	40000
E8	J. Jones	SA	50000

How many rows are returned?

A) 1

B) 2

C) 4

D) 8

GROUP BY Question #2

Question: Given this table and the query:

```
SELECT resp, pno, SUM(hours)
FROM   workson
WHERE  hours > 10
GROUP BY resp, pno
```

workson Table

eno	pno	resp	hours
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36

How many rows are returned?

A) 9 **B)** 7 **C)** 5 **D)** 1 **E)** 0

Try it: GROUP BY

Question: Use GROUP BY and aggregation functions to answer these queries.

1) Output the number of projects in the database.

2) Return the sum of the budgets for all projects.

3) For each department (dno), return the department number (dno) and the average budget of projects in that department.

4) For each project (pno), return the project number (pno) and the sum of the number of hours employees have worked on that project.

- Challenge: Show the project name (pname) as well as the project number.

5) Challenge: Show the department name (dname), project name (pname), and sum of hours worked on that project as well as the number of employees working on the project.

You can use Access or sqlfiddle.com.

Putting it All Together

The steps to write an English query in SQL are:

- 1) Find the columns that you need and put in SELECT clause.
- 2) List the tables that have the columns in the FROM clause. If there is more than one, join them together.
- 3) If you must filter rows, add a filter criteria in WHERE clause.
- 4) If you need to create an aggregate, use aggregation functions and GROUP BY.

Example: For each project name list the sum of the hours worked by employees working as a 'Manager' on the project.

```
SELECT pname, SUM(hours) as totalHours
FROM   workson INNER JOIN proj on workson.pno=proj.pno
WHERE  resp='Manager'
GROUP BY pname
```


Microsoft Access Querying Summary

1) Projection is performed by selecting the fields in the output in the field row in the table at the bottom of the screen.

2) Selection is performed by entering the condition in the criteria box. The criteria applies to the field in that column.

3) The tables used are added to the query by the **Show Table**… option.

4) Joins (based on relationships) are often automatically added, but if not, you can add them by selecting the join field in one table, holding the mouse button, then dragging to the join field in the other table.

Conclusion

A ***database*** is a collection of related data. A ***database system*** allows storing and querying a database.

SQL is the standard query language for databases, although Microsoft Access also provides a graphical user interface.

CREATE TABLE creates a table. INSERT, DELETE, and UPDATE commands modify the data stored within the database.

The basic query operations are selection (subset of rows), projection (subset of columns), join (combine two or more tables), and grouping and aggregation.

Objectives

- Define: database, database system, schema, metadata
- Define: relation, attribute, tuple, domain, degree, cardinality
- SQL properties: reserved words, case-insensitive, free-format
- Be able to create a table using CREATE TABLE command and in Microsoft Access.
- Explain what a key is and what it is used for.
- Use DROP TABLE to delete a table and its data.
- Use INSERT/UPDATE/DELETE to add/update/delete rows of a table and perform same actions using Microsoft Access user interface.
- Execute queries using SQL SELECT and using Microsoft Access user interface.
- Sort rows using ORDER BY. Use LIMIT to keep only the first (top) N rows.
- Use GROUP BY and aggregation functions for calculating summary data.

☆Given a small database write simple English queries in SQL.

DATA 301
Introduction to Data Analytics
Command Line

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why learn the Command Line?

The ***command line*** is the text interface to the computer.

Understanding the command line allows you to interact with the computer in ways that you often cannot with the user interface.

The command line is commonly used for scripting and automation of tasks and when accessing remote systems.

What is the Command Line?

The ***command line*** is the text interface to the computer that accepts commands that the computer will execute. These commands include:

- starting programs
- navigating directories and manipulating files
- searching, sorting, and editing text files
- system and environment configuration

The command line is part of the ***operating system***, which is software that manages your computer including all devices and programs.

- Common operating systems include Windows, Mac OS, and Linux/Unix.

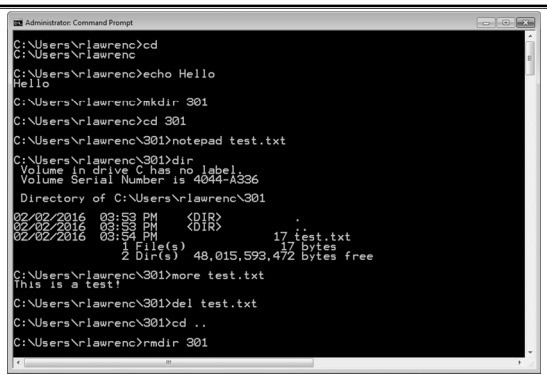
Windows Command Line

The command line on Windows dates back to the original Microsoft operating system called **DOS** (**Disk Operating System**) in 1981.

This command line interface is still part of all modern Windows operating systems and is accessible as the "Command Prompt".

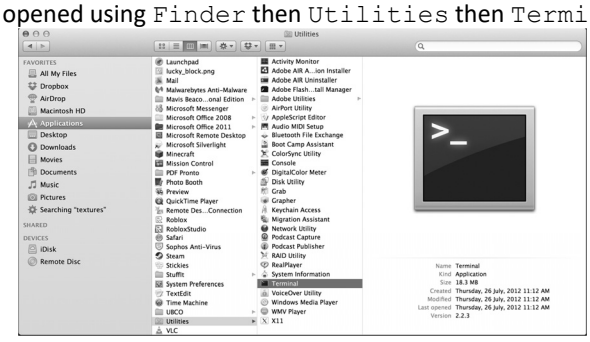
It is commonly used for system administration and scripting.

Command Line - Windows



Mac OS Command Line

The command line for Mac OS uses the same commands as Linux. It can be opened using `Finder` then `Utilities` then `Terminal`.



Command Line – Mac/Linux

```
rlawrenc — bash — 55×17
A4003829:~ rlawrenc$ pwd
/Users/rlawrenc
A4003829:~ rlawrenc$ echo Hello
Hello
A4003829:~ rlawrenc$ mkdir 301
A4003829:~ rlawrenc$ cd 301
A4003829:301 rlawrenc$ nano test.txt
A4003829:301 rlawrenc$ ls
test.txt
A4003829:301 rlawrenc$ cat test.txt
This is a test!
A4003829:301 rlawrenc$ rm test.txt
A4003829:301 rlawrenc$ cd ..
A4003829:~ rlawrenc$ rmdir 301
A4003829:~ rlawrenc$
```

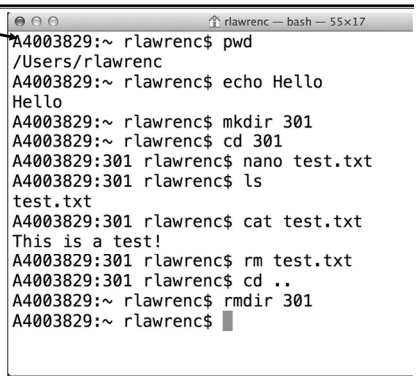

Entering a Command

Enter a ***command*** at a ***prompt***.

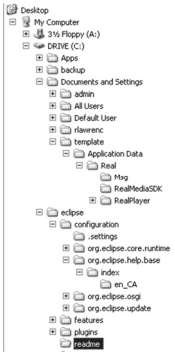
- The prompt may be a > or a $ or customized by the user.

Press ENTER to execute the command.

On Windows, commands are mostly case-insensitive while on Mac/Linux they are case-sensitive.



File System



The ***file system*** organizes data on a device as a hierarchy of directories and files.

Each ***folder*** (directory) has a name and can contain any number of files or subdirectories.

Each ***file*** has a name.

The user can change (navigate) directories in the hierarchy.

Absolute versus Relative Paths

The ***root*** of the file system is the directory "**/**".

- There is only one root of a directory hierarchy.

A path to a new location (from your current location) can be specified as an ***absolute path*** from the root:

```
cd /Users/rlawrenc/301/folder
```

or a ***relative path*** from your current location:

```
cd 301/folder
```

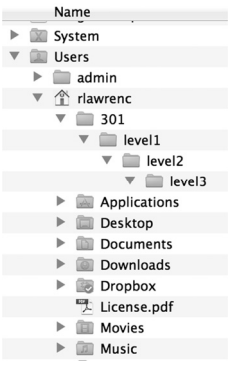
To back up one directory level, use the command: `cd ..`

Absolute versus Relative Path Question

Question: Given this directory hierarchy and that the user is currently in the directory `level2` and `level1` directory contains a file `test.txt`. How many of the following statements are **TRUE**?

1) A relative path to change to directory `301` is `..`
2) Absolute path to `test.txt` is `/Users/rlawrenc/301/level1/test.txt`
3) Relative path to `test.txt` is `../test.txt`
4) Relative path to `test.txt` is different if user was currently in `level3` directory.
5) There is only one root of the directory hierarchy.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4



Commonly Used File Navigation Commands

	Windows	Mac OS and Linux
List contents of directory	`dir`	`ls`
Change directory	`cd 301`	`cd 301`
Print working directory	`cd`	`pwd`
Make a directory	`mkdir 301`	`mkdir 301`
Remove a directory	`rmdir 301`	`rmdir 301`
Rename a file	`ren old.txt new.txt`	`mv old.txt new.txt`
Remove a file	`del file.txt`	`rm file.txt`
Copy a file	`copy src.txt dest.txt`	`cp src.txt dest.txt`

Commonly Used Text Related Commands

	Windows	Mac OS and Linux
Open a text editor	`notepad`	`nano`
Echo output	`echo Hello`	`echo Hello`
Output contents of a file	`more file.txt`	`cat file.txt`
Search text files	`find`	`grep`
Sort text files	`sort`	`sort`

Wildcards

A ***wildcard*** character allows for matching file names with more flexibility.

The `?` represents any ***one*** character in a file name.

Example: `file?.txt` would match `file1.txt`.

The `*` (asterisk) matches any number of characters (including zero).

Example: `*.txt` would match anything ending with `.txt` (`a.txt`).

Navigating the Command Line

	Windows Key	Mac OS Key
Previous command in history	`Up`	`Up`
Next command in history	`Down`	`Down`
First command in history	`PageUp`	
Last command in history	`PageDown`	
Move to start of line	`Home`	`Ctrl+A`
Move to end of line	`End`	`Ctrl+E`
Auto-compete file name	`Tab`	`Tab`

Pausing or Cancelling Commands

To **pause** a command:

- Windows: Press `Ctrl+S` or the `Pause` key. To resume, press any key.
- Mac: `Control+Esc` or `Command+.`

To **cancel** a command, press `Ctrl+C` or `Ctrl+Break`.

- The command is canceled, and the command prompt returns.
- However, any actions performed before the cancel are not undone.

Command Shortcuts Question

Question: How many of the following statements are **TRUE**?

1) To cancel a command, press `Ctrl+X`.
2) To go to the next command in the history, press `Up arrow`.
3) This wildcard expression `te*a?.txt` matches `tea12.txt`.
4) The command to change a directory is `pwd`.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Navigating Directories with Commands

Question: Using a terminal window on your computer, perform the following actions:

1) Create a directory called `301`.
2) Change into the directory `301`.
3) Echo `I am awesome!`
4) Show your current directory (print working directory).
5) Create a text file called `message.txt` with a message in it.
6) List the contents of your directory.
7) Rename the file `message.txt` to `test.txt`. Verify the name change.
8) Delete the `test.txt` file.
9) Change directory to directory above `301`.
10) Delete directory `301`.

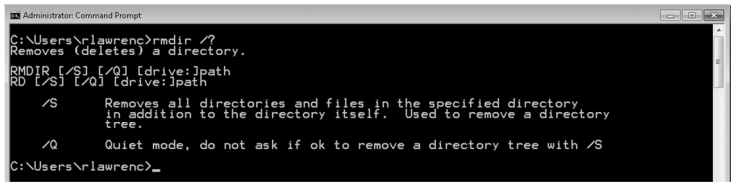
Command Arguments

A command can take ***arguments*** that changes its behavior.

- Example: Path was an argument for the `cd` command. e.g. `cd` ***301***

On Windows, commands also can be modified by a ***switch*** (or extension) which is usually a slash then a letter (e.g. `/S`).

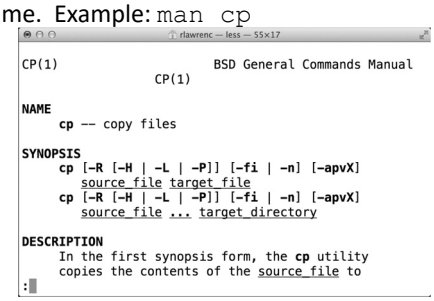
- To find out what is available, run the command with: `/?`



Command Arguments – Mac/Linux

On Mac/Linux, arguments are separated by spaces and begin with −.

An explanation of arguments can be found by using `man` then the command name. Example: `man cp`



Standard Input, Output, and Error

Standard input (`stdin`) is the default input device (usually a keyboard) into the terminal.

Standard output (`stdout`) is the location where output is sent after a command is run. The default is the terminal window.

Standard error (`stderr`) is the location where error messages are displayed (typically the terminal window).

Redirecting Input

By default, a command gets its input from standard input and outputs results to standard output.

A command can get its input from the output of another command by using the **pipe** (`|`) symbol. Example:

```
cat test.txt | wc
```

Also can use redirect input (<) to send input to a command. Example:

```
cat < test.txt
```

Note that can chain together multiple pipes.

- Note the example commands are Mac OS/Linux only: `wc` is not on Windows.

Redirecting Output

Redirect output using > which will overwrite the file:

```
sort test.txt > sorted.txt
```

Use >> to append to the existing file:

```
sort test.txt >> sorted.txt
```


Redirection Summary

	Symbol
Redirect input	<
Redirect output	>
Redirect output (append)	>>
Pipe output to input of next command	|

Escape Symbol

An ***escape symbol*** is used when a command requires input that contains a character with a special meaning. The escape symbol indicates this character is data not part of the command.

- On Windows, the caret (^) indicates that whatever character that follows it is data rather than part of the command. Example:

```
cp test.txt a^&b.txt
```

- On Linux, use the backslash (\).

This is especially common when dealing with spaces in a file name. The other way to handle file names with spaces is to enclose them in double quotes:

```
cp test.txt "c:\program files\file spaces.txt"
```


Environment Variables

Environment variables allow for customization and control of the command and system environment.

Current variables are seen using the `set` or `env` command.

Important variables:

- `$PATH` – list of directories where commands/applications will be found
- `$HOME` – user home directory

Finding Text in Files

The `grep` command allows for searching for text in files that match a pattern (Mac/Linux only, `find` on Windows).

- `grep` stands for "global regular expression print"
- Search is case-sensitive (use `-i` for case-insensitive) and can contain regular expressions.

Example:

```
grep er *.txt
```

- search for `er` in any file that ends in `.txt`

Windows: `find "er" *.txt`

Batch Files

A ***batch program*** (also commonly called a *batch file or command file*) is a text file that contains a sequence of commands to be executed.

You define the sequence of commands, name the sequence, and then execute the commands by entering the name at a command prompt. Any action you can take by typing a command at a command prompt can be encapsulated in a batch program.

In Windows files typically end in `.bat` or `.cmd` and on Mac/Linux with `.sh`.

Batch files can take arguments like other commands.

Connecting to Another Computer using SSH

Secure shell or SSH is a protocol allowing remote login to other machines to execute commands.

- The network communication is encrypted for security.
- An open-source program on campus is Putty.

Using SSH allows you to connect and execute commands on another machine even when you do not have physical access to that machine.

SSH may be used to send or retrieve data from other computers for analysis.

Try it: Using Batch Files

Question: Using a terminal window on your computer, create a batch file that performs these actions:

Before creating the batch file, create a file called `numbers.txt` that has the numbers `one`, `two`, `three`, …, `ten`.

In the batch file, called `myscript.bat` (or `.sh`):

1) Write a command to sort `numbers.txt` and output as `sorted.txt`.
2) Write a command to output the word count on `numbers.txt` to `count.txt`.
3) Write commands to take `numbers.txt` and append its data three times into the file `output.txt`.
4) Use `grep` to search for "`e`" in `output.txt` and write results as file `search.txt`.
5) Run your batch file.

Conclusion

The ***command line*** is the text interface to the computer that accepts commands that the computer will execute including running programs, manipulating files, and running scripts.

The command line allows for automation and more control than may be available in the user interface. It may also be the only way to interact with the machine if connecting via SSH.

The command environment allows for redirecting the standard input and output using input/output redirection and pipes.

Objectives

- Define command line and list some of its uses
- Explain the purpose of an operating system
- Know how to open the command line window on Mac OS and Windows
- Be able to enter commands and stop them
- Define: file system, folder, file
- Explain the difference between an absolute and relative path
- Use command line shortcuts to save time
- Be able to match wildcards involving ? and *
- Be able to cancel a command
- Explain standard input, standard output, and standard error
- Be able to use input and output redirection and pipes (?, >, < , >>)
- Explain the reason for an escape symbol
- Define and explain the purpose of environment variables.

Objectives (2)

- Be able to use grep to search text files.
- Explain the purpose of a batch program.
- Be able to connect to another machine using SSH.

DATA 301
Introduction to Data Analytics
Python

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why learn Python?

Python is increasingly the most popular choice of programming language for data analysts because it is designed to be simple, efficient, and easy to read and write.

There are many open source software and libraries that use Python and data analysis tools built on them.

We will use Python to learn programming and explore fundamental programming concepts of commands, variables, decisions, repetition, and events.

What is Python?

Python is a general, high-level programming language designed for code readability and simplicity.

Python is available for free as open source and has a large community supporting its development and associated tools.

Python was developed by Guido van Rossum and first released in 1991. Python 2.0 was released in 2000 (latest version 2.7), and a backwards-incompatible release Python 3 was in 2008.

- Our coding style will be Python 3 but most code will also work for Python 2.
- Name does refer to Monty Python.

Python Language Characteristics

Python supports:

- dynamic typing – types can change at run-time
- multi-paradigm – supports procedural, object-oriented, functional styles
- auto-memory management and garbage collection
- extendable – small core language that is easily extendable

Python core philosophies (by Tim Peters: https://www.python.org/dev/peps/pep-0020/)

- Beautiful is better than ugly
- Explicit is better than implicit
- Simple is better than complex
- Complex is better than complicated
- Readability counts

Some Quotes

"If you can't write it down in English, you can't code it."
-- Peter Halpern

"If you lie to the computer, it will get you."
-- Peter Farrar

Introduction to Programming

An ***algorithm*** is a precise sequence of steps to produce a result. A ***program*** is an encoding of an algorithm in a ***language*** to solve a particular problem.

There are numerous languages that programmers can use to specify instructions. Each language has its different features, benefits, and usefulness.

The goal is to understand fundamental programming concepts that apply to all languages.

Python: Basic Rules

To program in Python you must follow a set of rules for specifying your commands. This set of rules is called a ***syntax***.

- Just like any other language, there are rules that you must follow if you are to communicate correctly and precisely.

Important general rules of Python syntax:

- Python is ***case-sensitive***.
- Python is particular on whitespace and indentation.
- The end of command is the end of line. There is no terminator like a semi-colon.
- Use four spaces for indentation whenever in a block.

```
def spam():
    eggs = 12
    return eggs
print spam()
```


Comments

Comments are used by the programmer to document and explain the code. Comments are ignored by the computer. Two types:

- 1) One line comment: put "#" before the comment and any characters to the end of line are ignored by the computer.
- 2) Multiple line comment: put """ at the start of the comment and """ at the end of the comment. The computer ignores everything between the start and end comment indicators.

Example:

```
# Single line comment
print (1)  # Comment at end of line
""" This is a
multiple line
comment """
```


Python Programming

A Python program, like a book, is read left to right and top to bottom.

Each command is on its own line.

```
# Sample Python program
name = "Joe"
print("Hello")
print("Name: "+name)
```

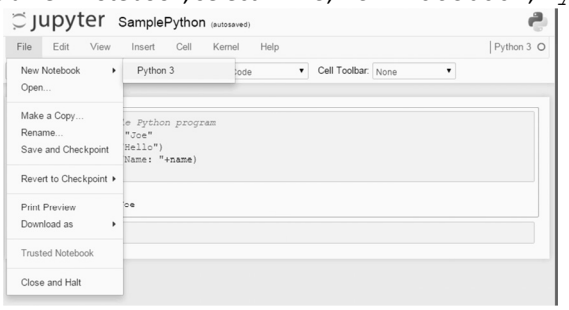
Flow of Execution
Start at first statement at top and proceed down executing each statement

A user types in a Python program in a text editor or development environment and then runs the program.

Python Editor - jupyter

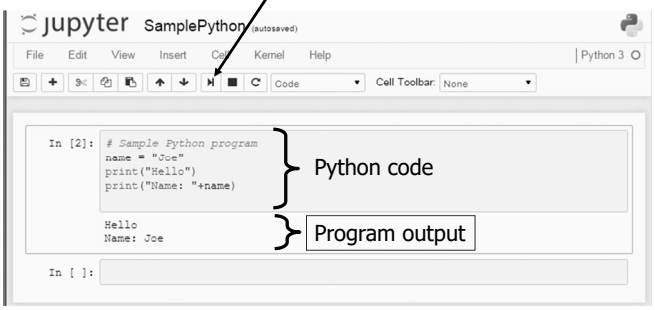
jupyter is a graphical, browser-based editor for Python.

To create a new notebook, select `File`, `New Notebook`, `Python3`.



Python Editor – jupyter notebook

Button to run code (shortcut is Ctrl+Enter)



Python: Hello World!

Simplest program:

```
print("Hello World!")
```

The `print` function will print to the terminal (standard output) whatever data (number, string, variable) it is given.

Try it: Python Printing

Question 1: Write a Python program that prints "I am fantastic!".

Question 2: Write a Python program that prints these three lines:

```
I know that I can program in Python.
I am programming right now.
My awesome program has three lines!
```


Python Question

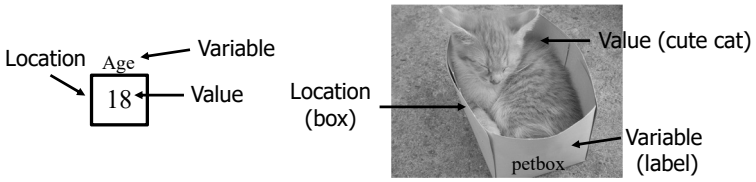
Question: How many of the following statements are **TRUE**?

1) Python is case-sensitive.
2) A command in Python is terminated by a semi-colon.
3) Indentation does not matter in Python.
4) A single line comment starts with """.
5) The `print` command prints to standard input.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Variables

A ***variable*** is a name that refers to a location that stores a data value.



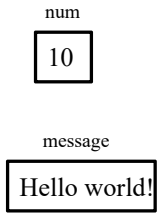
IMPORTANT: The *value* at a location can change using initialization or assignment.

Variable Assignment

Assignment using an **=** sets the value of a variable.

Example:

```
num = 10
message = "Hello world!"
```



Python Variables

To create a variable in Python, you must only provide a name.

- A variable type is dynamic. It can store numbers, strings, or Boolean at any time.

Example:

```
val = 5
val = "Hello"
```

Boolean values can be either `True` or `False`. Note case matters.

```
isAwesome = True
isAwesome = False
```


Variable Rules

Variables are a name that ***must begin with a letter*** and cannot contain spaces.

Variables are created when they are first used. There is no special syntax to declare (create) a variable.

Variable names **ARE** case-sensitive. Numbers are allowed (but not at the start). Only other symbol allowed is underscore ('_');

A programmer picks the names for variables, but try to make the names meaningful and explain their purpose.

Avoid naming variables as reserved words. A ***reserved word*** has special meaning in the language.

- e.g. `if`, `for`, `else`

Python Variables Question

Question: How many of the following variable names are valid?

1) `name`
2) `string2`
3) `2cool`
4) `under_score`
5) `spaces name`
6) `else`

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Python Math Expressions

Math ***expressions*** in Python:

Operation	Syntax	Example
Add	+	5 + 3
Subtract	-	10 – 2
Multiply	*	5 * 3
Divide	/	9 / 4
Modulus	%	9 % 4 (answer is 1)
Exponent	**	5 ** 2 (answer is 25)

Expressions - Operator Precedence

Each operator has its own priority similar to their priority in regular math expressions:

- 1) Any expression in parentheses is evaluated first starting with the inner most nesting of parentheses.
- 2) Exponents
- 3) Multiplication and division (*, /, %)
- 4) Addition and subtraction (+,-)

Python Expressions Question

Question: What is the value of this expression:

```
8 ** 2 + 12 / 4 * (3 – 1) % 5
```

A) 69 **B)** 65 **C)** 36 **D)** 16 **E)** 0

Try it: Python Variables and Expressions

Question 1: Write a program that prints the result of `35 + 5 * 10`.

Question 2: Write a program that uses at least 3 operators to end up with the value `99`.

Question 3: Write a program that has a variable called `name` with the value of your name and a variable called `age` storing your age. Print out your name and age using these variables.

Strings

Strings are sequences of characters that are surrounded by either single or double quotes.

- Use \ to escape ' E.g. There\'s
- Can use triple double quotes """ for a string that spans multiple lines.

Example:

```
name = "Joe Jones"
storeName = 'Joe\'s Store'
print("""String that is really long
with multiple lines
   and spaces is perfectly fine""")
```

Python String Indexing

Individual characters of a string can be accessed using square brackets (`[]`) with the first character at index 0.

Example:

```
str = "Hello"
print(str[1])        # e
print("ABCD"[0])     # A
print(str[-1])       # o
# Negative values start at end and go backward
```


Rules for Strings in Python

Must be surrounded by single or double quotes.

Can contain most characters except enter, backspace, tab, and backslash.

- These special characters must be escaped by using an initial "`\`".
- e.g. `\n` – new line, `\'` – single quote, `\\` - backslash, `\"` – double quote
- A string in raw mode (`r` before quote) will ignore backslash escape. May be useful if data contains escapes. Example: `st = r"slash\there\"`

Double quoted strings can contain single quoted strings and vice versa.

Any number of characters is allowed.

The minimum number of characters is zero `""`, which is called the *empty string*.

String *literals* (values) have the quotation marks removed when displayed.

Python Strings Question

Question: How many of the following are valid Python strings?

1) `""`
2) `''`
3) `"a"`
4) `" "`
5) `"""`
6) `"Joe\' Smith\""`

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5

Python String Functions

```
st  = "Hello"
st2 = "Goodbye"
```

Operation	Syntax	Example	Output
Length	`len()`	`len(st)`	`5`
Upper case	`upper()`	`st.upper()`	`HELLO`
Lower case	`lower()`	`st.lower()`	`hello`
Convert to a string	`str()`	`str(9)`	`"9"`
Concatenation	`+`	`st1 + st2`	`HelloGoodbye`
Substring	`[]`	`st[0:3]` `st[1:]`	`Hel` `ello`
String to int	`int()`	`int("99")`	`99`

String Operators: Concatenation

The ***concatenation operator*** is used to combine two strings into a single string. The notation is a plus sign '**+**'.

Example:

```
st1 = "Hello"
st2 = "World!"
st3 = st1 + st2 # HelloWorld!
print(st1+st1)
num = 5
print(st1+str(num)) # Hello5
# Must convert number to string before
# concatenation
```


String Concatenation Question

Question: What is the output of this code?

```
st1 = "Hello"
st2 = "World!"
num = 5
print(st1 + str(num) + " " + st2)
```

A) `Error`

B) `Hello5World!`

C) `Hello5 World!`

D) `Hello 5 World!`

Substring

The ***substring*** function will return a range of characters from a string.

Syntax:

```
st[start:end]   # start is included, end is not
                # first character is index 0
```

Examples:

```
st = "Fantastic"
print(st[1])        # a
print(st[0:6])      # Fantas
print(st[4:])       # astic
print(st[:5])       # Fanta
print(st[-6:-2])    # tast
```


Substring Question

Question: What is the output of this code?

```
st = "ABCDEFG"
print(st[1] + st[2:4] + st[3:] + st[:4])
```

A) ABCDCDEFGABCD

B) ABCDEFGABC

C) ACDDEFGABCD

D) BCDDEFGABCD

E) BCDECDEFGABC

Split

The ***split*** function will divide a string based on a separator.

Examples:

```
st = "Awesome coding! Very good!"
print(st.split())
# ['Awesome', 'coding!', 'Very', 'good!']
print(st.split("!"))
# ['Awesome coding', ' Very good', '']
st = 'data,csv,100,50,,25,"use split",99'
print(st.split(","))
# ['data', 'csv', '100', '50', '', '25',
#   '"use split"', '99']
```


Try it: Python String Variables and Functions

Question 1: Write a Python program that prints out your name and age stored in variables like this:

```
Name: Joe
Age: 25
```

Question 2: Write a Python program that prints out the first name and last name of `Steve Smith` like below. You must use substring.

- Bonus challenge: Use `find()` function so that it would work with any name.

```
First Name: Steve
Last Name: Smith
```


Print Formatting

The `print` method can accept parameters for formatting.

```
print("Hi", "Amy", ", your age is", 21)
print("Hi {}, your age is {}".format("Amy",21))
```

This is one of the most obvious changes between Python 2:

```
print "Hello"
```

and Python 3:

```
print("Hello")
```


Python Date and Time

Python supports date and time data types and functions.

First, import the `datetime` module:

```
from datetime import datetime
```

Functions:

```
now = datetime.now()
print(now)
current_year = now.year
current_month = now.month
current_day = now.day
print("{}-{}-{} {}:{}:{}".format(now.year, now.month,
now.day, now.hour, now.minute, now.second))
```

Python Clock

Python `time()` function returns the current time in seconds:

```
import time
startTime = time.time()
print("Start time:", startTime)
print("How long will this take?")
endTime = time.time()
print("End time:", endTime)
print("Time elapsed:", endTime-startTime)
```


Python Input

To read from the keyboard (standard input), use the method `input`:

```
name = input("What's your name?")
print(name)
age = input("What's your age?")
print(age)
```

Prompt for value from user

print out value received

- Note in Python 2 the method is called `raw_input()`.

Try it: Python Input, Output, and Dates

Question 1: Write a program that reads a name and prints out the name, the length of the name, the first five characters of the name.

Question 2: Print out the current date in `YYYY/MM/DD` format.

Comparisons

A ***comparison operator*** compares two values. Examples:

- `5 < 10`
- `N > 5  # N is a variable. Answer depends on what is N.`

Comparison operators in Python:

- `>` - Greater than
- `>=` - Greater than or equal
- `<` - Less than
- `<=` - Less than or equal
- `==` - Equal (Note: Not "=" which is used for assignment!)
- `!=` - Not equal

The result of a comparison is a ***Boolean value*** which is either **`True`** or **`False`**.

Conditions with `and`, `or`, `not`

A ***condition*** is an expression that is either `True` or `False` and may contain one or more comparisons. Conditions may be combined using: `and`, `or`, `not`.

- order of evaluation: `not`, `and`, `or` May change order with parentheses.

Operation	Syntax	Examples	Output

AND (True if both are True)	`and`	`True and True`
		`False and True`
		`False and False`
			`True`
			`False`
			`False`

OR (True if either or both are True)	`or`	`True or True`
		`False or True`
		`False or False`
			`True`
			`True`
			`False`

NOT (Reverses: e.g. True becomes False)	`not`	`not True` `not False`	`False` `True`

Condition Examples

```
n = 5
v = 8
print(n > 5)                          # False
print(n == v)                         # False
print(n != v)                         # True
print(n == v and n+4>v)               # False
print(n == v or n+4>v)                # True
print(n+1 == v-2 or not v>4)          # True
```

Python Condition Question

Question: How many of the following conditions are **TRUE**?

1) `True and False`
2) `not True or not False`
3) `3 > 5 or 5 > 3 and 4 != 4`
4) `(1 < 2 or 3 > 5) and (2 == 2 and 4 != 5)`
5) `not (True or False) or True and (not False)`

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Decisions

Decisions allow the program to perform different actions based on conditions. Python decision syntax:

```
if condition:                    if condition:
   statement  <-- Done if condition is True -->  statement
                                 else:
                                    statement  <-- Done if condition is False
```

- The statement after the `if` condition is only performed if the condition is `True`.
- If there is an `else`, the statement after the else is done if condition is `False`.
- Indentation is important! Remember the colon!

Decisions `if/elif` Syntax

If there are more than two choices, use the `if/elif/else` syntax:

```
if condition:
   statement
elif condition:
   statement
elif condition:
   statement
else:
   statement
```

```
if n == 1:
   print("one")
elif n == 2:
   print("two")
elif n == 3:
   print("three")
else:
   print("Too big!")
print("Done!")
```


Decisions: Block Syntax

Statements executed after a decision in an `if` statement are indented for readability. This indentation is also how Python knows which statements are part of the block of statements to be executed.

- If you have more than one statement, make sure to indent them. Be consistent with either using tabs or spaces. Do not mix them!

```
if age > 19 and name > "N":
   print("Not a teenager")
   print("Name larger than N")
else:
   print("This is statement #1")
   print(" and here is statement #2!")
```


Question: Decisions

Question: What is the output of the following code?

```
n = 3
if n < 1:
   print("one")
elif n > 2:
   print("two")
elif n == 3:
   print("three")
```

A) nothing **B)** `one` **C)** `two` **D)** `three`

Question: Decisions (2)

Question: What is the output of the following code?

```
n = 3
if n < 1:
   print("one")
elif n > 2
   print("two")
else:
   print("three")
```

A) nothing
B) `one`
C) `two`
D) `three`
E) error

Question: Decisions (3)

Question: What is the output of the following code?

```
n = 1
if n < 1:
   print("one")
elif n > 2:
   print("two")
else:
   print("three")
print("four")
```

A) nothing
B)
```
one
four
```
C)
```
three
```
D)
```
three
four
```
E) error

Question: Decisions (4)

Question: What is the output of the following code?

```
n = 0
if n < 1:
   print("one")
   print("five")
elif n == 0:
   print("zero")
else:
   print("three")
print("four")
```

A) nothing
B)
```
one
four
```
C)
```
one
five
four
```
D)
```
one
five
zero
four
```
E) error

Try it: Decisions

Question 1: Write a Python program that asks the user for a number then prints out if it is even or odd.

Question 2: Write a Python program that asks the user for a number between 1 and 5 and prints out the word for that number (e.g. 1 is one). If the number is not in that range, print out error.

Loops and Iteration

A ***loop*** repeats a set of statements multiple times until some condition is satisfied.

- Each time a loop is executed is called an ***iteration***.

A `for` loop repeats statements a number of times.

A `while` loop repeats statements while a condition is `True`.

The `while` Loop

The most basic looping structure is the ***while*** loop.

A while loop continually executes a set of statements **while** a condition is true.

Syntax:
```
while condition:
   statements
```

Example:
```
n = 1
while n <= 5:
   print(n)
   n = n + 1    # Shorthand: n += 1
```

Question: What does this print?

Question: `while` Loop

Question: What is the output of the following code?

```
n = 4
while n >= 0:
   n = n - 1
   print(n)
```

A) numbers 3 to -1 **B)** numbers 3 to 0 **C)** numbers 4 to 0
D) numbers 4 to -1 **E)** numbers 4 to infinity

Question: `while` Loop (2)

Question: What is the output of the following code?

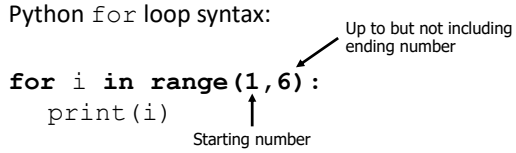
```
n = 1
while n <= 5:
   print(n)
n = n + 1
```

A) nothing **B)** numbers 1 to 5 **C)** numbers 1 to 6 **D)** lots of 1*s*

The `for` Loop

A `for` loop repeats statements a given number of times.

Python `for` loop syntax:



Using `range`

The basic form of range is:

`range(start,end)`

- start is inclusive, end is not inclusive
- default increment is 1

May also specify an increment:

`range(start, end, increment)`

or just the end:

`range(end)`

For Loop and While Loop

The `for` loop is like a short-hand for the `while` loop:

```
i=0
while i < 10:
   print(i)
   i += 1
```

```
for i in range(0, 10, 1):
   print(i)
```


Common Problems – Infinite Loops

Infinite loops are caused by an incorrect loop condition or not updating values within the loop so that the loop condition will eventually be false.

Example:

```
n = 1
while n <= 5:
   print(n)
   # Forgot to increase n -> infinite loop
```


Common Problems – Off-by-one Error

The most common error is to be "***off-by-one***". This occurs when you stop the loop one iteration too early or too late.

Example:

- This loop was supposed to print 0 to 10, but it does not.

```
for i in range(0,10):
   print(i)
```

Question: How can we fix this code to print 0 to 10?

Question: `for` Loop

Question: How many numbers are printed with this loop?

```
for i in range(1,10):
   print(i)
```

A) 0 **B)** 9 **C)** 10 **D)** 11 **E)** error

Question: `for` Loop

Question: How many numbers are printed with this loop?

```
for i in range(11,0):
   print(i)
```

A) 0 **B)** 9 **C)** 10 **D)** 11 **E)** error

Try it: `for` Loops

Question 1: Write a program that prints the numbers from 1 to 10 then 10 to 1.

Question 2: Write a program that prints the numbers from 1 to 100 that are divisible by 3 and 5.

Question 3: Write a program that asks the user for 5 numbers and prints the maximum, sum, and average of the numbers.

Lists Overview

A ***list*** is a collection of data items that are referenced by index.

- Lists in Python are similar to arrays in other programming languages

A list allows multiple data items to be referenced by one name and retrieved by index.

Python list:

```
data = [100, 200, 300, 'one', 'two', 600]
       0    1    2     3      4      5
```

list variable name

Indexes

Retrieving Items from a List

Items are retrieved by index (starting from 0) using square brackets:

```
data = [100, 200, 300, 'one', 'two', 600]
print(data[0])            # 100
print(data[4])            # 'two'
print(data[6])            # error – out of range
print(data[len(data)-1]) # 600
print(data[-1])           # 600
print(data[2:4])          # [300, 'one']

# Create an empty list:
emptyList = []
```


List Operations

```
data = [1, 2, 3, 5]
lst = []
```

Operation	Syntax	Examples	Output
Add item	*list*.append(*val*)	data.append(1)	[1, 2, 3, 5, 1]
Insert item	*list*.insert(*idx*,*val*)	data.insert(3,4)	[1, 2, 3, 4, 5]
Remove item	*list*.remove(*val*)	data.remove(5)	[1, 2, 3]
Update item	*list*[*idx*]=*val*	lst[0]=10	[10]
Length of list	len(*list*)	len(data)	4
Slice of list	*list*[x:y]	data[0:3]	[1, 2, 3]
Find index	*list*.index(*val*)	data.index(5)	3
Sort list	*list*.sort()	data.sort()	[1, 2, 3, 5]

List Details

If you provide an index outside of the valid range, Python will return an index error.

To sort in reverse order, do this:

```
data.sort(reverse=True)
```

For loops are used to iterate though items in a list:

```
for v in data:
   print(v)
```


Advanced: Python Lists Comprehensions

List comprehensions build a list using values that satisfy a criteria.

```
evenNums100 = [n for n in range(101) if n%2==0]
```

Equivalent to:

```
evenNums100 = []
for n in range(101):
   if n%2==0:
      evenNums100.append(n)
```


Advanced: Python Lists Slicing

List slicing allows for using range notation to retrieve only certain elements in the list by index. Syntax:

```
list[start:end:stride]
```

Example:

```
data = list(range(1,11))
print(data)     # [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(data[1:8:2])  # [2, 4, 6, 8]
print(data[1::3])   # [2, 5, 8]
```


Question: List

Question: At what index is item with value 3?

```
data = [1, 2, 3, 4, 5]
data.remove(3)
data.insert(1, 3)
data.append(2)
data.sort()
data = data[1:4]
```

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** not there

Try it: Lists

Question 1: Write a program that puts the numbers from 1 to 10 in a list then prints them by traversing the list.

Question 2: Write a program that will multiply all elements in a list by 2.

Question 3: Write a program that reads in a sentence from the user and splits the sentence into words using split(). Print only the words that are more than 3 characters long. At the end print the total number of words.

Python Dictionary

A ***dictionary*** is a collection of key-value pairs that are manipulated using the key.

```
dict = {1:'one', 2:'two', 3:'three'}
print(dict[1])          # one
print(dict['one'])      # error - key not found
if 2 in dict:           # Use in to test for key
   print(dict[2])       # two
dict[4] = 'four'        # Add 4:'four'
del dict[1]             # Remove key 1
dict.keys()             # Returns keys
dict.values()           # Returns values
```

Question: Dictionary

Question: What is the value printed?

```
data = {'one':1, 'two':2, 'three':3}
data['four'] = 4
sum = 0
for k in data.keys():
   if len(k) > 3:
      sum = sum + data[k]
print(sum)
```

A) 7 **B)** 0 **C)** 10 **D)** 6 **E)** error

Try it: Dictionary

Question: Write a program that will use a dictionary to record the frequency of each letter in a sentence. Read a sentence from the user then print out the number of each letter.

Code to create the dictionary of letters:

```
import string
counts = {}
for letter in string.ascii_uppercase:
    counts[letter] = 0
print(counts)
```


Functions and Procedures

A ***procedure*** (or ***method***) is a sequence of program statements that have a specific task that they perform.

- The statements in the procedure are mostly independent of other statements in the program.

A ***function*** is a procedure that returns a value after it is executed.

We use functions so that we do not have to type the same code over and over. We can also use functions that are built-in to the language or written by others.

Defining and Calling Functions and Procedures

Creating a function involves writing the statements and providing a ***function declaration*** with:

- a name (follows the same rules as identifiers)
- list of the inputs (called parameters)
- the output (return value) if any

Calling (or executing) a function involves:

- providing the name of the function
- providing the values for all arguments (inputs) if any
- providing space (variable name) to store the output (if any)

Defining and Calling a Function

Consider a function that returns a number doubled:

```
def doubleNum(num):
    num = num * 2
    print("Num: "+num)
    return num

n = doubleNum(5)          # 10
print(str(doubleNum(n)))  # ??
```

def Keyword

Function Name

Parameter Name

Call function by name

Argument

Python Built-in Math Functions

```
# Math
import math
print(math.sqrt(25))

# Import only a function
from math import sqrt
print(sqrt(25))

# Print all math functions
print(dir(math))
```

Other Python Built-in Functions

`max`, `min`, `abs`:

```
print(max(3, 5, 2))    # 5
print(min(3, 5, 2))    # 2
print(abs(-4))         # 4
```

`type()` returns the argument data type:

```
print(type(42))        # <class 'int'>
print(type(4.2))       # <class 'float'>
print(type('spam'))    # <class 'str'>
```


Python Random Numbers

Use random numbers to make the program have different behavior when it runs.

```
from random import randint
coin = randint(0, 1)        # 0 or 1
die = randint(1, 6)         # 1 to 6
print(coin)
print(die)
```


Advanced: Python Functions

Python supports functional programming allowing functions to be passed like variables to other functions.

- Lambda functions are functions that do not have a name.

Example:

```
def doFunc(func, val):
   return func(val)

print(doFunc(doubleNum, 10))        # 20
print(doFunc(lambda x: x * 3, 5))   # 15
```


Question: Functions

Question: What is the value printed?

```
def triple(num):
   return num * 3

n = 5
print(triple(n)+triple(2))
```

A) 0 **B)** 6 **C)** 15 **D)** 21 **E)** error

Practice Questions: Functions

1) Write a function that returns the largest of two numbers.

2) Write a function that prints the numbers from 1 to N where N is its input parameter.

Call your functions several times to test that they work.

Conclusion

Python is a general, high-level programming language designed for code readability and simplicity.

Programming concepts covered:

- variables, assignment, expressions, strings, string functions
- making decisions with conditions and `if`/`elif`/`else`
- repeating statements (loops) using `for` and `while` loops
- reading input with `input()` and printing with `print()`
- data structures including lists and dictionaries
- creating and calling functions, using built-in functions (math, random)

Python is a powerful tool for data analysis and automation.

Objectives

- Explain what is Python and note the difference between Python 2 and 3
- Define: algorithm, program, language, programming
- Follow Python basic syntax rules including indentation
- Define and use variables and assignment
- Apply Python variable naming rules
- Perform math expressions and understand operator precedence
- Use strings, character indexing, string functions
- String functions: split, substr, concatenation
- Use Python datetime and clock functions
- Read input from standard input (keyboard)

Objectives (2)

- Create comparisons and use them for decisions with `if`
- Combine conditions with `and`, `or`, `not`
- Use `if`/`elif`/`else` syntax
- Looping with `for` and `while`
- Create and use lists and list functions
- Advanced: list comprehensions, list slicing
- Create and use dictionaries
- Create and use Python functions
- Use built-in functions in math library
- Create random numbers
- Advanced: passing functions, lambda functions

DATA 301
Introduction to Data Analytics
Python Data Analytics

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Python File Input/Output

Many data processing tasks require reading and writing to files.

Open a file for reading:

I/O Type

```
infile = open("input.txt", "r")
```

Open a file for writing:

```
outfile = open("output.txt", "w")
```

Open a file for read/write:

```
myfile = open("data.txt", "r+")
```


Reading from a Text File (as one String)

```
infile = open("input.txt", "r")

val = infile.read()
print(val)

infile.close()
```

← Read all file as one string

← Close file

Reading from a Text File (line by line)

```
infile = open("input.txt", "r")
for line in infile:
   print(line.strip('\n'))
infile.close()


# Alternate syntax - will auto-close file
with open("input.txt", "r") as infile:
   for line in infile:
      print(line.strip('\n'))
```


Writing to a Text File

```
outfile = open("output.txt", "w")

for n in range(1,11):
    outfile.write(str(n) + "\n")

outfile.close()
```


Other File Methods

```
infile = open("input.txt", "r")

# Check if a file is closed
print(infile.closed)# False

# Read all lines in the file into a list
lines = infile.readlines()
infile.close()
print(infile.closed)# True
```

Use Split to Process a CSV File

```
with open("data.csv", "r") as infile:
   for line in infile:
      line = line.strip(" \n")
      fields = line.split(",")
      for i in range(0,len(fields)):
         fields[i] = fields[i].strip()
      print(fields)
```


Using `csv` Module to Process a CSV File

```
import csv

with open("data.csv", "r") as infile:
   csvfile = csv.reader(infile)
   for row in csvfile:
      if int(row[0]) > 1:
         print(row)
```


List all Files in a Directory

```
import os
print(os.listdir("."))
```


Python File I/O Question

Question: How many of the following statements are **TRUE**?

1) A Python file is automatically closed for you.
2) If you use the `with` syntax, Python will close the file for you.
3) To read from a file, use `w` when opening a file.
4) The `read()` method will read the entire file into a string.
5) You can use a `for` loop to iterate through all lines in a file.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Python Files

Question 1: Write a Python program that writes to the file `test.txt` the numbers from 20 to 10 in descending order.

Question 2: Write a Python program that reads your newly created `test.txt` file line by line and only prints out the value if it is even.

Question 3: Print out the contents of the CSV census file from:
https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province_population.csv

- Try to print out only the provinces with population > 1 million people and only the 2015 data. You will need to use float() and remove commas in data.

Internet Terminology Basics

An ***Internet Protocol*** (***IP***) ***address*** is an identifier for a computer on the Internet.

- IP version 4 (IPv4) address is 4 numbers in the range of 0 to 255. The numbers are separated by dots. Example: 142.255.0.1
- IP version 6 (IPv6) address has 16 numbers from 0 to 255 represented in hexadecimal. Example: 2002:CE57:25A2:0000:0000:0000:CE57:25A2

A ***domain name*** is a text name for computer(s) that are easier to remember. A ***domain*** is a related group of networked computers.

- Domain names are organized ***hierarchically***. The most general part of the hierarchy is at the end of the name.
- Example: `people.ok.ubc.ca`
  - `ca` – Canadian domain, `ubc` – University of British Columbia, `ok` – Okanagan campus, `people` – name of computer/server on campus

Internet Terminology Basics (2)

A ***uniform resource locator*** (***URL***) is an address of an item on the Internet. A URL has three parts:

- Protocol: `http://` - Hypertext Transfer Protocol
  - Tells the computer how to handle the file
- Server computer's domain name or IP address
- Item's path and name:
  - Tells the server which item (file, page, resource) is requested and where to find it.

Example:

`http://people.ok.ubc.ca/rlawrenc/teaching/301/index.html`

http protocol | server domain name | location of file/resource on server

Accessing (GET) Web Sites via URL with Python

```
import urllib.request
loc="http://people.ok.ubc.ca/rlawrenc/teaching/301"
site = urllib.request.urlopen(loc)
contents = site.read()
print(contents)
site.close()
```


Google Search with Python

```
import urllib
url = "http://www.google.com/search?hl=en&q=data+analysis"
headers={'User-Agent':'Mozilla/5.0 (Windows NT 6.1)'}
request = urllib.request.Request(url,None,headers)
response = urllib.request.urlopen(request)
data = response.read()
data = data.decode()    # Convert from Unicode to ASCII
print(data)
request.close()
```


Sending Data (PUT) to URL with Python

```
import urllib.parse
import urllib.request

url = 'http://cosc304.ok.ubc.ca/rlawrenc/tomcat/provinceState.jsp'
headers={'User-Agent':'Mozilla/5.0 (Windows NT 6.1)'}
# Build and encode data
values = {'country' : 'US'}
data = urllib.parse.urlencode(values)
data = data.encode('ascii')
req = urllib.request.Request(url, data, headers)
with urllib.request.urlopen(req) as response:
   page = response.read()
   print(page)
```


Python Web/URL Question

Question: How many of the following statements are **TRUE**?

1) An IPv4 address has 4 numbers between 0 and 256 inclusive.
2) A domain name is hierarchical with most specific part at the end.
3) Typically, a URL will reference more than one resource/item.
4) Python uses the file module for accessing URLs.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Python URLs

Question 1: Write a Python program that connects to any web page and prints its contents.

Question 2: Write a Python program that connects to:

https://people.ok.ubc.ca/rlawrenc/teaching/301/notes/code/data/province_population.csv

and outputs the CSV data.

- Modify your program to print each province and its 2015 population in descending sorted order.

Handling Errors and Exceptions

An ***exception*** is an error situation that must be handled or the program will fail.

- ***Exception handling*** is how your program deals with these errors.

Examples:

- Attempting to divide by zero
- An array index that is out of bounds
- A specified file that could not be found
- A requested I/O operation that could not be completed normally
- Attempting to follow a null or invalid reference
- Attempting to execute an operation that violates some kind of security measure

The `try-except` Statement

The ***`try-except` statement*** will handle an exception that may occur in a block of statements:

Execution flow:

- The statements in the `try` block are executed.
- If no exception occurs:
  - If there is an `else` clause, it is executed.
  - Continue on with next statement after `try`.
- If an exception occurs:
  - Execute the code after the `except`.
- If the optional `finally` block is present, it is always executed regardless if there is an exception or not.
- Keyword `pass` is used if any block has no statements.

Python Exceptions Example

```
try:
   num = int(input("Enter a number:"))
   print("You entered:",num)
except ValueError:
    print("Error: Invalid number")
else:
    print("Thank you for the number")
finally:
    print("Always do finally block")
```

try block exit if error

only execute if exception

only execute if no exception

always execute

Question: Exceptions

Question: What is the output of the following code if enter 10?

```
try:
   num = int(input("Enter num:"))
   print(num)
except ValueError:
    print("Invalid")
else:
    print("Thanks")
finally:
    print("Finally")
```

A)
```
10
```
B)
```
10
```
C)
```
Invalid
```
D)
```
10
Thanks
```
E)
```
10
Thanks
Finally
```


Question: Exceptions (2)

Question: What is the output of the following code if enter `hat`?

```
try:
   num = int(input("Enter num:"))
   print(num)
except ValueError:
    print("Invalid")
else:
    print("Thanks")
print("Finally")
```

A)
```
hat
```
B)
```
Invalid
```
C)
```
Invalid
Finally
```
D)
```
hat
Thanks
Finally
```
E)
```
Finally
```


Try it: Python Exceptions

Question: Write a Python program that reads two numbers and converts them to integers, prints both numbers, and then divides the first number by the second number and prints the result.

- If get an exception `ValueError` when converting to an integer, print `Invalid`.
- If get a `ZeroDivisionError`, print `Cannot divide by 0!`

Python Modules

A Python ***module*** or ***library*** is code written by others for a specific purpose. Whenever coding, make sure to look for modules that are already written for you to make your development faster!

Modules are imported using the import command:

```
import modulename
```

Useful modules for data analytics:

- Biopython (bioinformatics), NumPy (scientific computing/linear algebra), scikit-learn (machine learning), pandas (data structures), BeautifulSoup (HTML/Web)

Biopython

Biopython (http://biopython.org) is a Python library for biological and bioinformatics computation.

Features:

- parsers for bioinformatics file formats (BLAST, Clustalw, FASTA, Genbank)
- access to online services (NCBI - National Center for Biotechnology Information)
- sequence class
- clustering/classification (k Nearest Neighbors, Naïve Bayes, Support Vector Machines)
- Integration with BioSQL (sequence database schema)

Biopython Installation

Install in Anaconda by:

```
conda install biopython
```

Check if successfully installed and current version by:

```
import Bio
print(Bio.__version__)
```


Biopython Example - Using Sequences

```
# Create a sequence as a string
from Bio.Seq import Seq
my_seq = Seq("AGTACACTGGT")
print(my_seq)

# Read a FASTA file and print sequence info
from Bio import SeqIO
for seq_record in SeqIO.parse("sequence.fasta", "fasta"):
    print(seq_record.id)
    print(repr(seq_record.seq))
    print(len(seq_record))
    print(seq_record.seq.complement())
```


Biopython Transcription Example

```
# Transcription
from Bio.Seq import Seq
from Bio.Alphabet import IUPAC

coding_dna = Seq("TGCATTGGGTGCTGA",IUPAC.unambiguous_dna)
template_dna = coding_dna.reverse_complement()
messenger_rna = coding_dna.transcribe()

print("Coding:       ",coding_dna)
print("Template:     ",template_dna)
print("Messenger RNA:",messenger_rna)
print("Translation:  ",messenger_rna.translate())
```


Biopython - Entrez Database Search

Entrez is a federated database enabling retrieval of data from many health sciences databases hosted by the NCBI.

```
# Retrieve data from nucleotide database as FASTA
from Bio import Entrez
from Bio import SeqIO
Entrez.email = "test@test.com"
# Providing GI for single entry lookup
handle = Entrez.efetch(db="nucleotide", rettype="fasta",
retmode="text", id="3288717")
record = SeqIO.read(handle, "fasta")
handle.close()
print(record)
```

Biopython - BLAST

BLAST (Basic Local Alignment Search Tool) compares an input sequence with database and returns similar sequences.
http://blast.ncbi.nlm.nih.gov/

```
# Retrieve data from nucleotide database as FASTA
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML

sequence = "ACTATTCCAAACAGCTCATAACCAGAAA"
handle = NCBIWWW.qblast("blastn", "nt", sequence)

result = handle.read()
print(result)     # Output is in XML format
```


Biopython BLAST - Parsing Results

```
from Bio.Blast import NCBIWWW
from Bio.Blast import NCBIXML
sequence = "ACTATTCCAAACAGCTCATAACCAGAAA"
handle = NCBIWWW.qblast("blastn", "nt", sequence)
records = NCBIXML.parse(handle)
record = next(records)
for alignment in record.alignments:
    for hsp in alignment.hsps:
        print('\nsequence:', alignment.title)
        print('length:', alignment.length)
        print('e value:', hsp.expect)
        print(hsp.query[0:75] + '...')
        print(hsp.match[0:75] + '...')
        print(hsp.sbjct[0:75] + '...')
```


Try it: Biopython

Question: Write a program that has a DNA sequence that you create, performs a BLAST, and then outputs the top 3 hits.

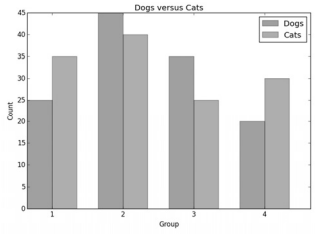
Charts

There are numerous graphing and chart libraries for Python:

- matplotlib (http://matplotlib.org/) - foundational 2D plotting library
- ggplot (http://ggplot.yhathq.com/) - based on R's ggplot2
- pygal - dynamic chart library
- Bokeh (http://bokeh.pydata.org/) - goal is to produce charts similar to D3.js for browsers
- Seaborn (http://stanford.edu/~mwaskom/software/seaborn/) - based on matplotlib and designed for statistical graphics

matplotlib - Bar Chart Example

```
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np

data1 = [25,45,35,20]
data2 = [35,40,25,30]
index = np.arange(len(data1))
bar_width = 0.35
opacity = 0.4
error_config = {'ecolor': '0.3'}
rects1 = plt.bar(index, data1, bar_width, alpha=opacity,
                 color='b', yerr=None, error_kw=error_config,
                 label='Dogs')
```

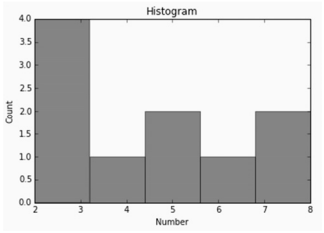


matplotlib - Bar Chart Example (2)

```
rects2 = plt.bar(index + bar_width, data2, bar_width,
                 alpha=opacity, color='r', yerr=None,
                 error_kw=error_config, label='Cats')

plt.xlabel('Group')
plt.ylabel('Count')
plt.title('Dogs versus Cats')
plt.xticks(index + bar_width, ('1', '2', '3', '4'))
plt.legend()
plt.tight_layout()
plt.show()
```

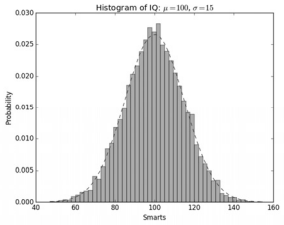
matplotlib - Histogram Example

```
%matplotlib inline

import numpy as np
import matplotlib.pyplot as plt

num_bins = 5
x = [5, 3, 8, 5, 2, 7, 2, 4, 6, 2]
n, bins, patches = plt.hist(x, num_bins,
          normed=False, facecolor='blue',
          alpha=0.5)

plt.xlabel('Number')
plt.ylabel('Count')
plt.title('Histogram')
plt.show()
```



matplotlib - Histogram Example #2

```
import numpy as np
import matplotlib.mlab as mlab
import matplotlib.pyplot as plt
mu = 100
sigma = 15
x = mu+sigma*np.random.randn(10000)
num_bins = 50
n, bins, patches = plt.hist(x, num_bins,
          normed=1, facecolor='green',
          alpha=0.5)
y = mlab.normpdf(bins, mu, sigma)
plt.plot(bins, y, 'r--')
plt.xlabel('Smarts')
plt.ylabel('Probability')
plt.title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')
plt.subplots_adjust(left=0.15)
plt.show()
```



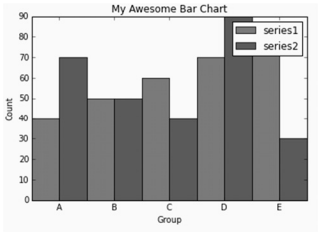
Note: Set `normed=0` to show counts rather than probabilities.

Try it: Charts

Question: Write a program to create a bar chart for this data:

- series1 = [40, 50, 60, 70, 80]
- series2 = [70, 50, 40, 90, 30]

Output:



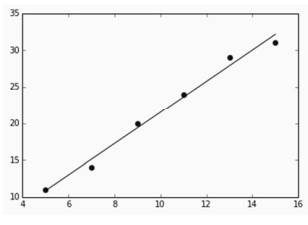
SciPy

SciPy is group of Python libraries for scientific computing:

- NumPy (http://www.numpy.org/) - N-dimensional arrays, integrating C/C++ and Fortran code, linear algebra, Fourier transform, and random numbers
- SciPy (http://www.scipy.org/) - numerical integration and optimization
- matplotlib (http://matplotlib.org/) - 2D plotting library
- IPython (http://ipython.org/) - interactive console (Jupyter)
- Sympy (http://www.sympy.org/) - symbolic mathematics (equations, calculus, statistics, combinatorics, cryptography)
- pandas (http://pandas.pydata.org/) - data structures, reading/writing data, data merging/joining/slicing/grouping, time series

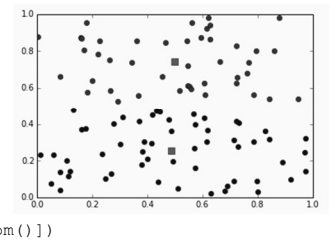
SciPy Linear Regression Example

```
from scipy import stats
import numpy as np
import matplotlib.pyplot as plt
x = np.array([5, 7, 9, 11, 13, 15])
y = np.array([11, 14, 20, 24, 29, 31])
slope, intercept, r_value, p_value,
  slope_std_error = stats.linregress(x, y)
predict_y = intercept + slope * x
print("Predicted y-values:",predict_y)
pred_error = y - predict_y
print("Prediction error:",pred_error)
degr_freedom = len(x) - 2
residual_std_error = np.sqrt(np.sum(pred_error**2) / degr_freedom)
print("Residual error:",residual_std_error)
plt.plot(x, y, 'o')
plt.plot(x, predict_y, 'k-')
plt.show()
```



SciPy k-Means Clustering Example

```
%matplotlib inline
import matplotlib.pyplot as plt
from scipy.cluster.vq import kmeans,vq
import random as rnd



# data generation
data = []
for i in range(0,100):
    data.append([rnd.random(), rnd.random()])

# Perform k-means clustering
numclusters = 2
centroids,_ = kmeans(data,numclusters) # Calculates centroids
idx,_ = vq(data,centroids)             # Puts each point in a cluster
```



SciPy k-Means Clustering Example (2)

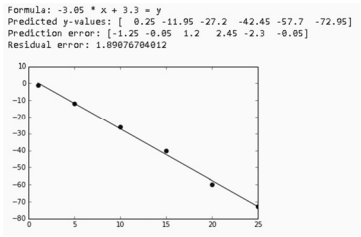
```
# Move data into individual lists based on clustering
clusters = []
for i in range(0, numclusters):
    clusters.append([[],[]])

for i in range(0,len(idx)):
    clusterIdx = idx[i]
    clusters[clusterIdx][0].append(data[i][0])
    clusters[clusterIdx][1].append(data[i][1])

# Plot data points and cluster centroids
plt.plot(clusters[0][0],clusters[0][1],'ob',
        clusters[1][0],clusters[1][1],'or')
plt.plot(centroids[:,0],centroids[:,1],'sg',markersize=8)
plt.show()
```


Try it: SciPy

Question: Write a program that uses SciPy to perform a linear regression on this data set:

- `x = [1, 5, 10, 15, 20, 25]`
- `y = [-1, -12, -26, -40, -60, -73]`

Output:



scikit-learn Library

scikit-learn (http://scikit-learn.org/) is a machine learning library for Python.

Features: classification, regression, clustering, dimensionality reduction

BeautifulSoup Library

BeautifulSoup (http://www.crummy.com/software/BeautifulSoup/) is a library to make it easy to search, navigate, and extract data from HTML and XML documents.

Databases

Python can connect to databases to retrieve data. MySQL example:

```
import mysql.connector
try:
    cnx = mysql.connector.connect(user='rlawrenc', password='test',
                  host='cosc304.ok.ubc.ca', database='WorksOn')
    cursor = cnx.cursor()
    query = ("SELECT eno, ename, salary FROM Emp WHERE title > %s "
             +"and salary < %s")
    cursor.execute(query, ('EE', 50000))
    for (eno, ename, salary) in cursor:
        print(eno, ename, salary)
    cursor.close()
except mysql.connector.Error as err:
    print(err)
finally:
    cnx.close()
```


Try it: Databases

Question: Write a program that queries the WorksOn database and returns the employees grouped by title where the employee name is after `'J'`. The output should display their title and the average salary for that title. Connection info:

- `cnx = mysql.connector.connect(user='rlawrenc', password='test', host='cosc304.ok.ubc.ca', database='WorksOn')`

Output:

```
EE 30000.000000
ME 40000.000000
PR 20000.000000
SA 50000.000000
```

Map-Reduce

Map-Reduce is a technique for processing large data sets in a functional manner.

- The technique was invented by Google and is implemented in a variety of systems including Python, NoSQL databases, and a Big Data system called Hadoop.
- In Hadoop, map takes as input key-value pairs and outputs key-value pairs. The shuffle step will move pairs to particular machines based on keys. The reduce step takes a list of key-value pairs (with same key) and reduces to one value.
- It is possible to code map/reduce functions in Python for use in Hadoop cluster.

Simpler version of Map-Reduce in Python without a cluster:

- Map function - takes as input a list and a function then applies function to each element of the list to produce a new list as output
- Filter function - only keeps list elements where filter function is `True`
- Reduce function - takes result of map/filter and produces single value from list

Python Map-Reduce Example

```
import functools     # For Reduce

data = [1, 2, 3, 4, 5, 6]

# Map function
def triple(x):
    return x*3

# Filter function
def myfilter(x):
    if x % 2 == 0:
        return True
    return False

# Reduce function
def sum(x, y):
    return x+y
```


Python Map-Reduce Example (2)

```
result = list(map(triple, data))
print("Result after map:",result)

result = list(filter(myfilter, result))
print("Result after filter:",result)

result = functools.reduce(sum, result)
print("Result after reduce:",result)
```


Try it: Map-Reduce

Question: Write a map-reduce program that during the map step will subtract 2 from each element. The reduce step should return the product of all the elements in the list.

Conclusion

Python has many libraries to help with data analysis tasks:

- reading and write to files
- `csv` module for processing CSV files
- Biopython for bioinformatics
- numerous chart libraries including `matplotlib` and `ggplot`
- SciPy - collection of libraries for scientific computing
- libraries for web access and parsing (BeautifulSoup)
- database access libraries and connectors

The ***`try-except` statement*** is used to handle exceptions so that the program may continue when an error condition occurs.

Objectives

- Open, read, write, and close text files
- Process CSV files including using the `csv` module
- Define: IPv4/IPv6 address, domain, domain name, URL
- Read URLs using `urllib.request`.
- Define: exception, exception handling
- Use `try-except` statement to handle exceptions and understand how each of `try`, `except`, `else`, `finally` blocks are used
- Import Python modules
- Use Biopython module to retrieve NCBI data and perform BLAST
- Build charts using `matplotlib`
- Perform linear regression and k-means clustering using SciPy
- Connect to and query the MySQL database using Python
- Write simple Map-Reduce programs

DATA 301
Introduction to Data Analytics
Statistics: R

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

What is R?

R is a free and open source programming language for statistical computing and graphics.

- One of the most widely used programming languages for statistical analysis.
- Popular in academia and companies like Microsoft, Google, and Facebook.
- There are currently over 8000 packages in R. (https://cran.r-project.org/web/packages/available_packages_by_name.html)
- R creates high quality graphs and visualizations.

Why learn R?

R is built to handle and analyze data.

Example: Filtering a dataset to be within a lower and upper bound and then calculating summary statistics.

Python

```
for v in data:
    # Only process data if in [lower,upper]
    if v >= lower and v <= upper:
        # Update maximum if larger
        if v > maxdata:
            maxdata = v
        # Update minimum if smaller
        if v < mindata:
            mindata = v

        # Update sum and count
        sumdata += v
        count += 1
```

R

```
#subset data to be within bounds
new_data = subset(data,x <= upper & x >= lower)

sumdata = sum(new_data)
count = length(new_data)
maxdata = max(new_data)
mindata = min(new_data)
```


Statistics Review: Types of Data

There are two types of data:

- Qualitative (Categorical)
  - Descriptions or groups
  - Can be characters or numbers
  - Observed and not measured
  - i.e. names, labels, categories, properties
- Quantitative (Numeric)
  - Strictly numeric
  - Can be measured
  - i.e. height, weight, speed, counts, temperature, volume

Numerical Summaries

A ***numerical summary*** provides an overview of data to help understand it without examining all data values.

Use a ***measure of centre*** and a ***measure of spread*** to describe quantitative data.

Measures of Centre

Mean is the average of data values (sum of values divided by count).

$$\bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

Median is the value at which half of the data lies above that value and half lies below it.

- Odd number of observations: $\tilde{y}$ is the *k*th value where $k = (n+1)/2$.
- Even number of observations: $\tilde{y}$ is the mean of the *k*th and *(k+1)* terms, where $k = n/2$

Example Calculation for Mean and Median

Data:

$y = \{1, 3, 3, 7, 9\}$

The mean and median are:

- $\bar{y} = \frac{1+3+3+7+9}{5} = 4.6$
- $\tilde{y} = 3$

In R, use the `mean()` and `median()` functions:

```
>
> mean(y)
[1] 4.6
> median(y)
[1] 3
>
```


Measures of Spread

A measure of spread indicates how far apart the values are.

Variance - is the sum of the squares of each data point's distance from the mean.

- $s^2 = \frac{\sum_{i=1}^{n}(y_i - \bar{y})^2}{n-1} = \frac{\left(\sum_{i=1}^{n} y_i^2\right) - n\bar{y}^2}{n-1}$

Standard Deviation - is the square root of the variance.

- $s = \sqrt{s^2}$

Range - is the maximum value minus the minimum value.

- $\max - \min$

Example Calculation for Variance/Standard Deviation

Data:

$y = \{1, 3, 3, 7, 9\}$

The variance and standard deviation are:

- $s^2 = \frac{(1+9+9+49+81) - 5 \cdot 4.6^2}{5-1} = 10.8$
- $s = 3.286$

In R, use the `var()` and `sd()` functions:

```
> var(y)
[1] 10.8
> sd(y)
[1] 3.286335
```


Data Measures Question

Question: Using the data y, how many of the following are **TRUE**?

$y = \{1,2,3,4,5,6\}$

1. $\bar{y} = \tilde{y}$
2. $\bar{y} = 3$
3. $s^2 = 3.5$
4. $range = 6$

A) 0 B) 1 C) 2 D) 3 E) 4

Quantiles and Quartiles

The qth quantile is the point where at least $q \cdot 100\%$ of the data values are at or below the value.

There are some special quantiles called ***Quartiles*** (quarters of the data).

- Q1 – first quartile – 0.25 quantile
- Q2 – second quartile – 0.5 quantile – median
- Q3 – third quartile – 0.75 quantile

The ***Interquartile Range*** is the difference between Q3 and Q1. It contains the centre 50% of the data.

IQR = Q3 – Q1

Example Quartiles

Data: $y = \{1,2,3,4,5,6\}$ Median: $\tilde{y} = \frac{3+4}{2} = 3.5$

Q1 and Q3 are then the 'medians' of the two subsets of data when divided at the median

- $y_1 = \{1,2,3\}$ and $y_2 = \{4,5,6\}$
- $Q1 = 2, Q3 = 5$

The function is `quantile()` in R.

Quantiles Question

Question: Given y =integers from 0:100, how many of the following are **TRUE**?

1. The median and Q3 are 50 and 75 respectively.
2. Each integer y_i is the y_i /100th quantile. i.e. 5 is the 0.05th quantile.
3. For every data set, Q2 is strictly less than Q3.
4. If the data is reversed the quantile values remain unchanged.

A) 0 B) 1 C) 2 D) 3 E) 4

Five Number Summary

A ***five number summary*** consists of the following:

- minimum
- Q1
- median
- Q3
- maximum

Using $y = \{1,2,3,4,5,6\}$ the 5 number summary would be:

Min	Q1	Median	Q3	Max
1	2	3.5	5	6

Data Summaries Question

Question: How many of the following statements are **TRUE**?

1. Variance is always non-negative.
2. Standard deviation can be 0.
3. If a > b, then quantile(a) >= quantile(b).
4. The 5 number summary uses the mean of a dataset.

A) 0 B) 1 C) 2 D) 3 E) 4

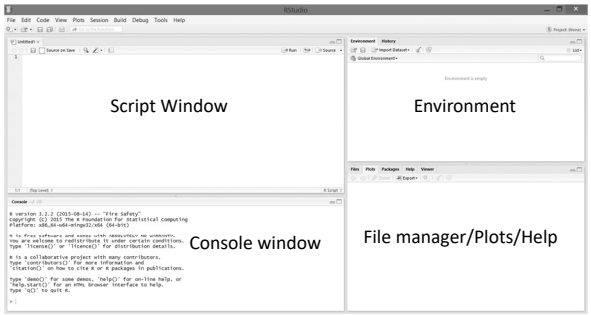
RStudio

RStudio is an integrated development environment (IDE) for R.

Install R first! Download here: https://cran.rstudio.com/

Download RStudio at:
https://www.rstudio.com/products/rstudio/download/

RStudio Environment



RStudio IDE

Script Window

- Draft and save code
- Write a script to run in the console (`CTRL+R` or `CTRL+Enter`, or pressing Run)

Console

- Where the code goes once run
- Shows input (blue), output (black) and any errors or warnings (red)

Environment

- Shows saved variables and datasets

File Browser/Plots/Help…

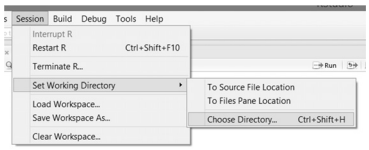
- Show files in working directory and generated plots
- Help window opens here

Working Directory

The ***working directory*** is the 'home base' of your R program. All files are written to and read from the working directory. There are two ways to do this.

1) Using the user interface

`Session → Set Working Directory → Choose Directory…`



2) Use the `setwd()` function

```
setwd("c:/tmp")
```

Get the working directory with `getwd()`.

R: Hello World!

```
print("Hello World!")
```

The `print` function will print to the console the input it is given.

Try it: R Printing

Question 1: Write a R program that prints "I am awesome!".

Question 2: Write a R program that prints these three lines:

```
I can program in R!
I can program in Python!
I can program in at least 2 languages! Can you?
```


Basics of R

R is case-sensitive.

R commands may be separated either by a semi-colon or a newline.

Brackets `{ }` are used to group commands together.

Basic Syntax of R

Commenting is done with a #. There are no multiline comments.

```
#This is a comment
```

Variables are assigned using a <-

```
x <- 4
y <- 10
```

To get help for any function use `help(function)` or `?function`

```
>
> help(c)
>
```

```
>
> ?c
>
```


Calculations in R

R has standard math operators (+, −, *, /, ^ (power)).

```
> 1+2
[1] 3
> 2-3
[1] -1
> 2*3
[1] 6
> 6/2
[1] 3
> 2^3
[1] 8
>
```

Predefined functions in R:

- Trigonometric functions: `sin`, `cos`, `tan`
- Exponential: `exp`, `log` (natural log), `log10`

Note: `pi` returns the value of π **BUT** you can (accidentally) redefine it.

R Question

Question: How many of the following statements are **TRUE**?

1) R is case-sensitive.
2) A command in R can be terminated by a semi-colon.
3) Indentation is the syntax used to group statements together.
4) A single line comment starts with #.
5) The = is the preferred syntax for variable assignment.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

R Data Types

Numeric
- Decimal values

Integer
- Can be created using `as.integer()`

Complex
- Complex values (i.e. a+bi)

Logical
- TRUE/FALSE. Can be denoted using T/F.

Character
- String values denoted with single or double quotes.

Try it: R Variables and Expressions

In a R program:
- Make a comment with your name and student number
- Calculate the following:
  - 4*5-12^3
  - e^{4*3}
  - sin(4*pi-6)
- Make the following variables. What types do you think they are?
  - var1 = TRUE
  - var2 = F
  - var3 = 3^4 - 10
  - var4 = "Hello World"
- You can print out the responses by typing the variable name in the console and pressing Enter.

Comparisons

Comparison operators in R:
- `>` - Greater than
- `>=` - Greater than or equal
- `<` - Less than
- `<=` - Less than or equal
- `==` - Equal (Note: Not "="!)
- `!=` - Not equal

The result of a comparison is a ***Boolean value*** which is either **`TRUE`** or **`FALSE`**.

Conditions with `and`, `or`, `not`

Operation	Syntax	Examples	Output

AND (True if both are True)	`&`	`TRUE & TRUE`
		`FALSE & TRUE`
		`FALSE & FALSE`
			`TRUE`
			`FALSE`
			`FALSE`

OR (TRUE if either or both are TRUE)	`|`	`TRUE | TRUE`
		`FALSE | TRUE`
		`FALSE | FALSE`
			`TRUE`
			`TRUE`
			`FALSE`

NOT (Reverses: e.g. TRUE becomes FALSE)	`!`	`!TRUE` `!FALSE`	`FALSE` `TRUE`

Decisions

Decisions allow different actions based on conditions. R syntax:

```
if (condition)                    if (condition)
{  statements   <- Done if condition is TRUE ->   {  statements
}                                                 }
                                                  else
                  Done if condition is FALSE ->   {  statements
                                                  }
```

- The statement after the `if` condition is only performed if the condition is `TRUE`.
- If there is an `else`, the statement after the else is done if condition is `FALSE`.
- Indentation is recommended but not required.
- Statements are grouped using brackets which are optional if only one statement.

Decisions `if/else if` Syntax

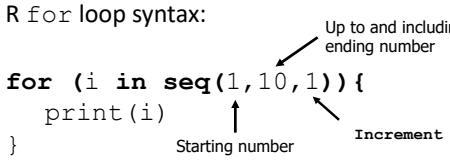
```
if (condition)
{  statement
} else if (condition)
{  statement
} else if (condition)
{  statement
}else
{  statement
}
```

```
if (n == 1)
{  print("one")
} else if (n == 2)
{  print("two")
} else if (n == 3)
{  print("three")
} else
{  print("Too big!")
}
```

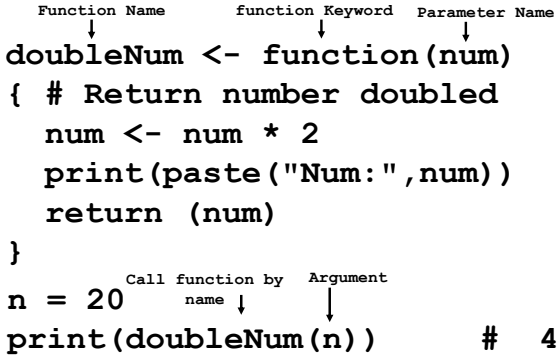

The `for` Loop

A `for` loop repeats statements a given number of times.

R `for` loop syntax:



Defining and Calling a Function in R



Try it: R Decisions, Loops, and Functions

Question: Write a R program that contains a function called `printEven` that prints the first 10 even numbers starting from an input number passed in.

- Note: Modulus is `%%`.
- Test your function with input values 5 and 10.

Reading Data Sets

Read delimited data:

```
data <- read.table("filename",sep="",header=TRUE)
```

- `Filename` – name of file to read in i.e. input.txt
- `sep` - separator character. Default "" uses any type of whitespace. Others: , \t ;
- `header` – if `TRUE` then the first row is used for variable names

Read CSV data:

```
data <- read.csv("filename", header=TRUE)
```

- Specific case of `read.table()` with `sep=","`

`head()` and `tail()`

After reading a data set, use `head()` to show the first 6 rows and `tail()` to show the last 6 rows.

```
data <- read.csv("data.csv", header=TRUE)
head(data)
tail(data)
head(data, 10)    # First 10 rows
tail(data, 20)    # Last 20 rows
```

Reading Data with R

Question: How many of the following statements are **TRUE**?

1) R can read comma separated and tab separated files.

2) If `HEADER=TRUE`, the first row of the file is assumed to be column names (i.e. not data).

3) If `HEADER=TRUE` and there is no header row, the program crashes.

4) By default, `head()` and `tail()` return 10 rows.

5) A parameter passed into `head()` can change # of rows returned.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Data Structures - Vectors

A ***vector*** is an indexed list of data of any type.

Create vectors using a colon or `seq()` (R's version of range).

- `1:10`
- `seq(5, 1, by = -0.5)` # Default by is 1

Create an empty vector with `c()`, or fill it by specifying elements.

- `c()`
- `c(4, 3, 5, 'a', 'd')`

NOTE: First index is 1!

Access elements in a vector using `[]`

- `myVector[i]` # Returns ith element of myVector
- `myVector[1]` # Returns 4

Vectors in R

Question: How many of the following statements are **TRUE**?

1) Vectors in R are indexed from 0.

2) `1:10` creates a vector of 10 numbers.

3) A vector may have data values of different types.

4) If `data <- 1:5`, then `data[2]+data[3] = 3`.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Data Structures - Matrices

A ***matrix*** is a structure of rows and columns where each data value is the same data type. All rows must have the same length. All columns must have the same length.

Create a matrix from the vector `x` using `matrix()`

- `matrix(x, nrow = 5, ncol = 3, byrow = FALSE)`

```
# Starts at [1,1] and fills the column first before
# going onto the next column.
# Need to only specify ncol or nrow
```

Access elements using `[row, col]`. Leaving one of them blank returns the whole row or column.

- `myMatrix[i,j]` `# Returns ith row and jth column`

Matrices and Vectors

Append a vector to a matrix as a row using `rbind()`:

```
myMatrix = rbind(myMatrix, vec)
```

Append a vector to a matrix as a column using: `cbind()`:

- `myMatrix = cbind(myMatrix, vec)`

Data Structures - Lists

A ***list*** is an ordered collection of objects of any type.

Create a list using `list()`. Specify names of elements by using `name=` inside the brackets.

- `myList = list(x = 1:4, y = c('a','b'))`

```
# Creates a list with two elements x and y
```

Access elements using the double square brackets

- `myList[[2]]` `# Returns 2nd item of list (y)`
- `myList[['x']]` `# Returns item with the name x`

Lists and Matrices in R

Question: How many of the following statements are **TRUE**?

1) Data values in a list may be of different types.

2) In a matrix, the number of rows and number of columns must be the same.

3) Given matrix `m`, `m[2]` would return all data in row 2.

4) Given matrix `m`, `m[,2]` would return all data in column 3.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Lists

Create a list called grades. Add in the following elements:

- *Name (containing first and last name)
- Student number
- *Assignment grades
- Midterm grade

The *'s indicate that the fields should have multiple entries.

Data Structures – Data Frames

A ***data frame*** is similar to a matrix but the columns can have different data types.

- Note: Still have uniform length of rows and columns.
- Data frames are a very common structure for data analysis.

Create a data frame by using `data.frame()`. Specify names of variables within the brackets.

```
myDF = data.frame(x = c(1:3), y = (2:4))
```

Change a matrix into a `data.frame` using `as.data.frame()`.

```
myDF = as.data.frame(myMatrix)
```


Data Structures – Accessing Data in Data Frames

Access elements using `[row, col]` or `$variable_name`.

```
myDF[i, j]        # ith row and jth column
myDF$x            # Returns the column labeled x
```

Can add new column called `vec` into the data frame using `$`

```
myDF$new_col = vec      # Adds vec as new_col
```


Data Structures - Factors

Factors are used for qualitative groups/categories (i.e. Male/Female). Use `as.factor()` to turn a vector or `data.frame` column into a factor.

```
myFactor = as.factor(x)
myDF$x = as.factor(myDF$x)
```

Access elements using `[]`:

```
myFactor[i]        # Returns ith element
```

Can use `class()` or `str()` to gain information about the type and/or structure of your variable/data. `str()` gives more detail.

Question on Data Structures

Question: How many of the following are **TRUE**?

1. Matrices must have the same number of rows as columns.
2. Vectors must contain only one data type.
3. A factor can contain only characters.
4. A Data frame's columns can be of varying length.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Subsets

Subsetting is used to extract data with particular values.

Syntax:

```
subset(data, condition)
```

Example:

```
# Only return data for province of BC
cars_bc = subset(cars, prov == 'BC')
```


Try it: Data Frame

Create a data frame `mydata` with the following column names/data:

- id - numbers 1 to 5
- location - "BC", "BC", "AB", "MB", "BC"
- value - 10, 20, 30, 40, 50
- Make location a factor.

Add one more column to your data frame that is a factor:

- success - "Y", "N", "N", "N", "Y"

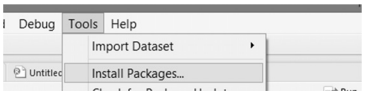
Display only the data from BC and value >= 20.

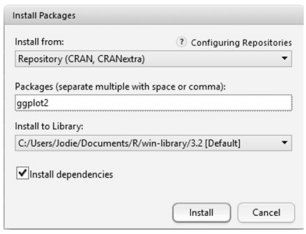
Visualizing Data in R

R supports several graphing libraries to produce graphs for qualitative and quantitative data including bar charts, histograms, and box plots.

We will use the package ggplot2. gg stands for Grammar of Graphics.

To install `tools → Install Packages...` Then input 'ggplot2'





Graphs for Qualitative Data: Frequency Table

Frequency tables summarize the number of observations in each group.

Use: `table(variable)`

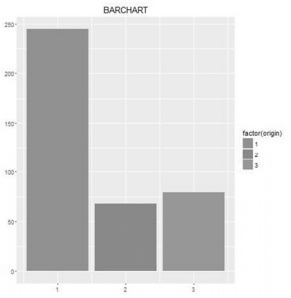
```
>
> table(Auto$origin)

  1   2   3
245  68  79
>
```


Graphs for Qualitative Data: Bar Charts

Bar charts have each group along the x-axis and a vertical bar with the height representing the number of observations of each group.

Code example:

```
ggplot(Auto, aes(x = origin))+
geom_bar(aes(fill=factor(origin)))
 + xlab("") + ylab("")
 + ggtitle("BARCHART")
```



- Using the dataset Auto in the ISLR package.

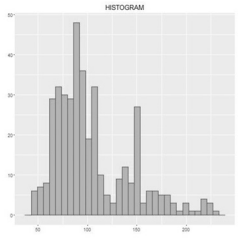
Graphs for Quantitative Data: Histogram

A ***histogram*** is similar to a bar chart, but the x-axis is divided into bins.

The variable of interest is on the x-axis, and the y-axis represents count of observations within each bin.

Visualizes the data distribution.

Code example:

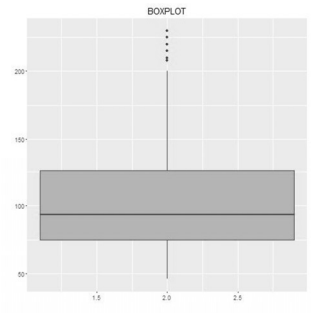



```
ggplot(Auto, aes(x = horsepower))
+ geom_histogram(color = 'mediumvioletred', fill=
'mediumaquamarine')
+ xlab("") + ylab("") + ggtitle("HISTOGRAM")
```

Graphs for Quantitative Data: Boxplot

A ***boxplot*** is a visualization of the 5 number summary.

- Groups along the x-axis
- Data values along the y-axis
- Lowest and highest points are the min and max of the data respectively.
- Bottom of box is Q1 and top is Q3
- Median is represented as the bar inside the box.
- Single points represent outliers.



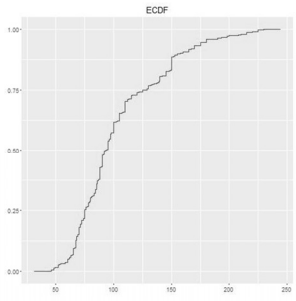
Boxplot Example Code

```
ggplot(Auto, aes(x = origin, y = horsepower))
+ geom_boxplot(color = 'mediumvioletred', fill= 'mediumaquamarine')
+ xlab("") + ylab("")+ ggtitle("BOXPLOT")
```


Graphs for Quantitative Data: ECDF

An ***Empirical Cumulative Distribution Function (ECDF) plot*** shows values along the x-axis and quantiles along the y-axis.

Each data point is plotted along with its corresponding quantile.



ECDF Example Code

```
ggplot(Auto, aes(x = horsepower))
+ stat_ecdf(color = 'mediumvioletred')
+ xlab("") + ylab("")+ ggtitle("ECDF")
```


Graphical Summary Question

Question: How many of the following are **TRUE**?

1. Bar charts and histograms will work for the same variables.
2. Boxplots show a 5 number summary.
3. Variable type does not matter, any graph can be used.
4. Histograms can give an idea of the distribution of a variable.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it!

1) Using the car data from the data.frame example, create a bar chart for the variable `prov`.

2) Use the cars data and create a histogram of any variable.

3) Create a boxplot for the variable of your choice!
- What are the median and minimum values? Can you estimate the IQR?

4) Make an ECDF for the variable of your choice.
- Recalling that Q1, median, and Q3 are the 0.25, 0.5, and 0.75th quantiles, what is your best guess at these values from reading off of the graphs?

Confidence Intervals

62 percent of US College students miss a class due to excessive drinking. The result is accurate within 1.7 percentage points 19 times out of 20.

Taking the pieces out of the above statement we have:

- 62 is the estimated percentage
- 1.7 is the margin of error
- 19 times out of 20 is the stated confidence -> 100%(19/20) = 95%

This is a 95% confidence interval: (60.3, 63.7)

Confidence Intervals (2)

General form:

$$(\mu - me, \mu + me)$$

Interpret a 95% confidence interval as that we are 95% confident that the interval will contain the true value of the parameter.

Hypothesis Testing

Hypothesis testing is used to determine if a relationship exists between two sets of data and make decisions/conclusions about that relationship.

Hypothesis testing is useful for:

- business - determining effectiveness of marketing, identifying customer buying properties, online advertising optimization
- science/social science - determining if data sets match a model, understanding scientific process based on collected data values, analysis of study data

Hypothesis Testing Steps

1) Declare hypotheses statement and null hypothesis

2) Decide on test statistic

3) Use P-value and/or confidence interval to make decision/conclusion

- A p-value of 0.05 "signifies that if the null hypothesis is true, and all other assumptions made are valid, there is a 5% chance of obtaining a result at least as extreme as the one observed" (http://www.nature.com/news/statisticians-issue-warning-over-misuse-of-p-values-1.19503)

Data is used as evidence. Perform a test in order to make a decision: reject the null hypothesis or fail to reject the null hypothesis.

NOTE: We cannot prove if the null hypothesis is true or false. We can only show that there is evidence to suggest one conclusion or another.

Assumptions

There are assumptions that need to be met before performing statistical tests.

For the one sample case:

- Population of interest is normally distributed
- Independent random samples are taken

For the two sample case:

- The two samples are independent
- Populations of interest are normally distributed

One Sample Test

A ***one sample test*** is used when a sample is compared to a model or known population/estimate.

As an example, using the car data test if the average mileage is different than 10 km/L.

One Sample Test: Hypotheses Statements

Null hypothesis (H_0) always contains a statement of no change (=).

Alternative hypothesis (H_A) can be one sided (< or >) or two sided (≠).

H_0: μ = test_number

H_A: μ ≠ test_number

Car mileage example:

H_0: μ = 10

H_A: μ ≠ 10

One Sample Test: Calculate Test Statistic

For the one sample test the t-test statistic is calculated as:

$$t = \frac{\bar{y} - \mu}{s/\sqrt{n}}$$

- $\bar{y}$ is sample mean, s is sample standard deviation, n is sample size, μ is specified mean value

R code:

```
t.test(x = car_data$km.L,
       alternative = c("two.sided"), mu = 10)
```

One Sample Test: Decision and Conclusion (using P-value)

If p-value > 0.05, the probability of seeing a sample mean more extreme is not that unlikely.

- Fail to reject the null hypothesis
- There is no evidence to suggest that the mean value of VARIABLE is less than, greater than, or different than the test value.

If p-value < 0.05,

- Reject the null hypothesis
- There is evidence to suggest that the mean value of VARIABLE is less than, greater than, or different than the test value.

One Sample Test: Decision and Conclusion (using P-value) Example


```
> t.test(x = car_data$km.L,alternative = c("two.sided"),mu = 10)

        One Sample t-test

data:  car_data$km.L
t = 1.608, df = 29, p-value = 0.1187
alternative hypothesis: true mean is not equal to 10
95 percent confidence interval:
  9.90338 10.80729
sample estimates:
mean of x
 10.35533
```

P-value = **0.1187 > 0.05 => Fail to reject the null hypothesis**

There is no evidence to suggest that the mean mileage is not 10 km/L.

Note: Unable to claim that either the null or alternative hypothesis is true. Can only reject or fail to reject the null hypothesis.

One Sample Test: Decision and Conclusion (using CI) Example


```
> t.test(x = car_data$km.L,alternative = c("two.sided"),mu = 10)

        One Sample t-test

data:  car_data$km.L
t = 1.608, df = 29, p-value = 0.1187
alternative hypothesis: true mean is not equal to 10
95 percent confidence interval:
  9.90338 10.80729
sample estimates:
mean of x
 10.35533
```

Can also make a conclusion (reject or fail to reject) based on the confidence interval. We are 95% confident that the true mean mileage of the car lies within those bounds.

Since 10 km/L is within those bounds, fail to reject the null hypothesis.

Two Sample Unpaired

An unpaired (independent) ***two sample test*** compares two independent samples to determine if there is a difference between the groups.

Examples:

- Compare effectiveness of two different drugs tested on two sets of patients
- Experiment versus control samples

Two Sample Unpaired Example Hypothesis Statement

Using the beaver2 dataset in R, test the hypothesis that there is no difference between the mean active temperature and the mean non-active temperatures.

H_0: $\mu_1 = \mu_2 \rightarrow \mu_1 - \mu_2 = 0$

H_A: $\mu_1 \neq \mu_2 \rightarrow \mu_1 - \mu_2 \neq 0$

Two Sample Unpaired Example Test Statistic

Use t-test statistic.

R code:

```
# Need to set active to be a factor first
beaver2$activ = as.factor(beaver2$activ)
# Perform unpaired test
t.test(temp~activ, data=beaver2,
  alternative=c("two.sided"), mu=0,
  paired=FALSE)
```

Two Sample Unpaired Example Decision and Conclusion (using P-value)


```
> t.test(temp~activ, data = beaver2, alternative = c("two.sided"),mu = 0, paired = FALSE)

        Welch Two Sample t-test

data:  temp by activ
t = -18.548, df = 80.852, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.8927106 -0.7197342
sample estimates:
mean in group 0 mean in group 1
       37.09684        37.90306
```

The p-value << 0.05.

Reject the null hypothesis. There is evidence to suggest that there is a difference between active and non active temperatures.

Two Sample Unpaired Example Decision and Conclusion (using CI)


```
> t.test(temp~activ, data = beaver2, alternative = c("two.sided"),mu = 0, paired = FALSE)

        Welch Two Sample t-test

data:  temp by activ
t = -18.548, df = 80.852, p-value < 2.2e-16
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.8927106 -0.7197342
sample estimates:
mean in group 0 mean in group 1
       37.09684        37.90306
```

The two sample case tests a DIFFERENCE between the groups ($\mu_1 - \mu_2 \neq 0$). The CI stated above is the CI for the difference, $\mu_1 - \mu_2$.

We reject the null hypothesis because 0 is not contained in the interval.

If 0 was contained we would fail to reject the null hypothesis.

Two Sample Paired Test

A ***paired (dependent) two sample test*** compares two dependent samples to see if there is a difference between the groups.

- This test typically uses multiple measurements on one subject.
- Also called a "repeated measures" test.

Examples:

- Affect of treatment on a patient (before and after)
- Apply something to test subjects to see if there is an effect
- Car example: Do cars get better mileage with different grades of gasoline?

Two Sample Paired Test Example Hypothesis Statement

The athlete.csv dataset contains data on ten athletes and their speeds for the 100m dash before training (Training = 0) and after (Training = 1).

Test the hypothesis that their training has no affect on the times of the athletes. Test to see if the mean of the difference is different than 0.

H_0: $d = 0$

H_A: $d \neq 0$

Two Sample Paired Test Example Test Statistic - R Code

```
# Read in the data
athlete = read.csv("athlete.csv", header=TRUE)

# Perform paired test
t.test(Time~Training, data = athlete,
alternative=c("two.sided"), mu=0, paired=TRUE)
```


Two Sample Paired Test Example Decision and Conclusion (using P-value)

```
> t.test(Time~Training, data = athlete, alternative = c("two.sided"),mu =
0, paired = TRUE)

        Paired t-test

data:  Time by Training
t = -0.12031, df = 9, p-value = 0.9069
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5544647  0.4984647
sample estimates:
mean of the differences
                -0.028
```

The p-value >> 0.05.

Fail to reject the null hypothesis. There is no evidence to suggest that there is a difference between pre and post training times.

Two Sample Paired Test Example Decision and Conclusion (using CI)

```
> t.test(Time~Training, data = athlete, alternative = c("two.sided"),mu =
0, paired = TRUE)

        Paired t-test

data:  Time by Training
t = -0.12031, df = 9, p-value = 0.9069
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.5544647  0.4984647
sample estimates:
mean of the differences
                -0.028
```

The two sample case tests for a difference between the groups ($d \neq 0$). The CI is for the difference.

Fail to reject the null hypothesis because 0 is contained in the confidence interval.

Sampling Question 1

Question: How many of the following are **TRUE**?

1. Paired and unpaired t-tests are the same thing.
2. Confidence intervals can be of any level of confidence (not just 95%).
3. Confidence intervals can be used to make a conclusion about a hypothesis test.
4. Confidence intervals can be used to prove that the null hypothesis is false.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Sampling Question 2

Question: How many of the following are **TRUE**?

1. Unpaired t-tests test the difference between two means μ_1 and μ_2.
2. Paired t-tests can be used to compare the difference between two measurements on the same subject.
3. In both the paired and unpaired two sample cases, a confidence interval containing 0 would result in a decision of: fail to reject the null hypothesis.
4. In the one sample t-test, a confidence interval containing 0 would result in a decision of: fail to reject the null hypothesis.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Hypothesis Testing Question

Question: How many of the following hypothesis questions should use ***two sample unpaired tests***?

1. Is the average student mark in courses 70%?
2. Does a student's mark improve after studying?
3. Has the average student height increased since 1990?
4. Does radiation reduce the size of tumors when used to treat patients?
5. Is aspirin more effective than Tylenol for treating headaches?
6. Are college graduates better than high school graduates at standardized tests?

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try It: Hypothesis Testing

1. Using the car data, test the hypothesis that the mean distance at each fill up is less than 450km.
2. Use the car data to see if the mean distance for Alberta fill ups is different than the mean distance for B.C. fill ups.

Linear models in R

A linear model is an equation that relates a response variable (*y*) to some explanatory variables (*x*'s). The general form of the model is:

$$y = b_0 + b_1x_1 + b_2x_2 + \cdots + b_nx_n$$

Not all of the data points can fall on this line so the full equation is

$$y_i = b_0 + b_1x_{1i} + b_2x_{2i} + \cdots + b_nx_{ni} + \varepsilon_i$$

Where ε_i denotes the error term associated with observation *i*.

Fitting a Linear Model

```
lm(km.L~Litres+Distance, data = car_data)
```

```
> model = lm(km.L~Litres+Distance, data = car_data)
> model

Call:
lm(formula = km.L ~ Litres + Distance, data = car_data)

Coefficients:
(Intercept)       Litres     Distance
   10.35447     -0.33295      0.03251
```

The formula can then be created using the values stored in `model$coefficients`

```
Km.L = 10.35447 -0.33295*Litres + 0.03251*Distance
```


Conclusion

R is a free and open source programming language for statistical computing and graphics.

R contains many useful features for data analysis including data structures such as vectors and data frames that make it easy to perform statistical analysis and visualization.

R is often used for hypothesis testing and understanding how to properly setup and interpret a test is an important skill.

Objectives

- Understand purpose and usefulness of R
- Types of data: qualitative, quantitative
- Describe data use numerical summaries (measure of centre/spread)
- Define and calculate: mean, median, variance, standard deviation, range
- Define: quantile, quartile, interquartile range, five number summary
- Perform matrix addition, subtraction, and multiplication
- Install and use RStudio
- Set and get the working directory
- Write small programs/commands in R that may use variables, conditions, loops, and functions
- Read in data sets from files
- Use head and tail to explore a data set
- Create and use data structures: vectors, matrices, lists

Objectives (2)

- Use data frames/factors for data analysis
- Create graphs/visualizations: frequency table, bar chart, histogram, boxplot, ECDF using ggplot2
- Explain the purpose of confidence intervals
- Perform hypothesis testing using R
- Understand assumptions inherent in a t-test
- Compute linear models using R

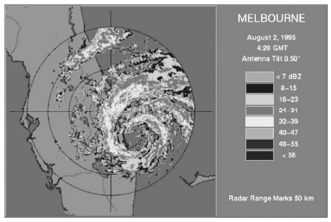
DATA 301
Introduction to Data Analytics
Geographic Information Systems

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why learn Geographic Information Systems?

Geographic Information Systems (GIS) are used in a wide variety of areas for the analysis and display of spatial and geographical data:

- City and infrastructure planning
- Business development, planning, forecasting
  - Store placement, sales trends
- Population forecasting and analysis
- Environmental and water



What is a Geographic Information System?

Geographic Information Systems are systems designed for storing, manipulating, analyzing, and displaying spatial and geographical data.

A GIS will contain components:

- for importing data from various sources in different formats
- organizing the data into layers or groups
- composing and integrating data to produce new information
- displaying the data visually as maps or 3D visualizations to help interpret the data

GIS History

The technique of ***overlaying*** information on maps dates back long before computers.

In the 1960s geographic mapping software developed the basic GIS concepts. First GIS developed by Dr. Roger Tomlinson for Canadian Department of Forestry and Rural Development.

In 1969, Environmental Systems Research Institute (ESRI) founded by Laura and Jack Dangermond and developed suite of products.

- Software linked spatial representation of features with table attributes
- ESRI now the de facto standard for commercial GIS products (e.g. ArcGIS, ArcView).

GIS Features

A GIS allows a user to add (or overlay) data on a map including:

- Annotation (text) - name or description of item/feature (e.g. city name)
- Point - a single (x,y) co-ordinate on the map
- Line - a connected pair of two (x,y) points
- Polygon - three or more (x,y) points connected to form a closed shape

Each feature can have one or more additional attributes (data items) describing it.

For example, a city could be represented as a point on the map (with coordinates) and additional attributes include its name and population.

GIS Data Types

The GIS feature consists of coordinates placing it on the map.

Each feature can have attributes providing additional information.

These attributes may have data types such as:

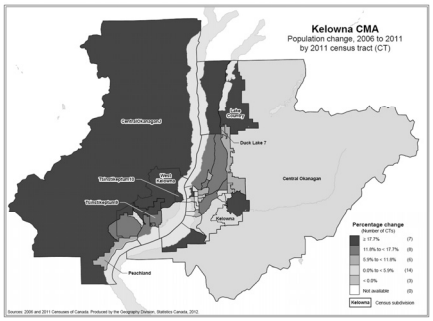
- Text - for names and labels
- Categories - for grouping similar features/classes (road, land use, etc.)
- Numbers - for measurements (population, rainfall, etc.)

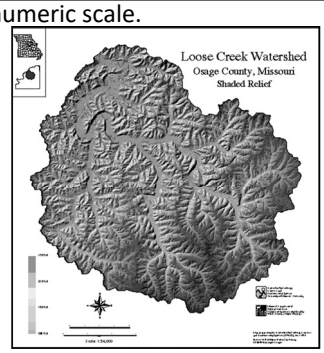
Data is often placed in categories for display. Each item in the category has the same symbol.

- Measurement data is also grouped into categories to ease understanding even if the data is continuous. For example, ordinal data has categories ranked according to a scale: low, medium, high

GIS Interval Data

Interval data has values along a regular numeric scale.





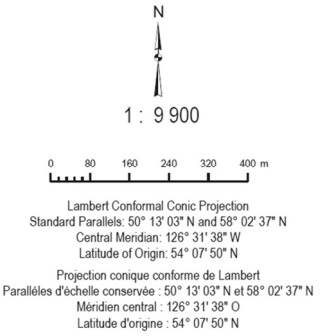
GIS Terminology: Scale and Precision

Scale is the ratio of size on the ground to size on the map.

Precision is a measure of how accurate the map representation is compared to the real-world.

- If map symbol is 1 point (72 points=1 inch) and scale is 1:100000, then symbol S = 100,000 points = 1388 inches = 115 feet is uncertainty in placement based on representation.

Resolution is the sampling distance of the stored x-y values.



Scale Question

Question: If the scale of the map is 1:100000, and the feature on the map is 3 cm long, how long is the feature in the real-world?

A) 1 cm **B)** 3 cm **C)** 300000 m **D)** 300 m **E)** 3 km

Precision Question

Question: If the scale of the map is 1:100000, and a road is represented on the map by a line that is 2 mm think, what is the error in representation (i.e. how far off can the road be in real-life)?

A) 2 km **B)** 200 m **C)** 20 m **D)** 2 m **E)** No error

Feature Classes and Layers

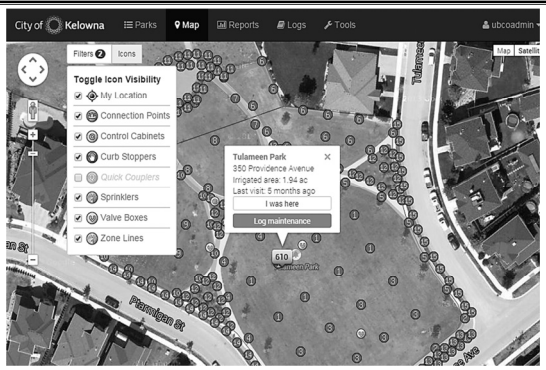
A ***feature class*** is a collection of objects with the same attributes.

- May be stored as individual rows of a single table
- Have the same geometry (e.g. all points or all polygons).
- Example classes: states, cities, rivers

A ***layer*** is a grouping of features that can be added or removed from the map (and its display visualization).

- A layer will often reference or use a feature class.

Feature Class and Layer Screenshot



Representing GIS Data: Raster and Vector

There are two common methods for representing GIS data.

Raster representation uses a matrix of data values.

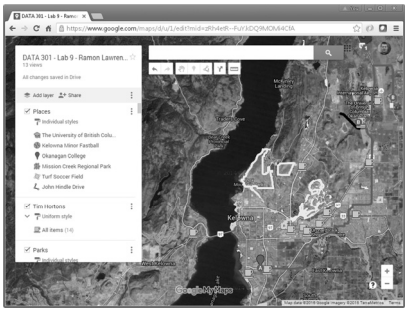
Vector representation adds features (points, polygons) onto a map each with its own coordinates and attributes.

Vector Representation

Vector representation adds features onto a map with their own coordinates and attributes.

- Features stored as series of x-y coordinates and may be points, lines, polygons.
- Features are linked to a row in a data table which may have multiple attributes describing it.

Allows for very precise specification of features by coordinates which may have multiple attributes.



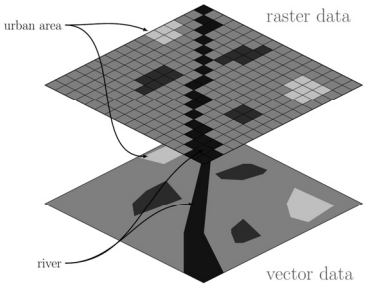
Raster Representation

A ***raster*** stores data as a matrix of data points that is georeferenced to earth's surface.

- The value at each data point may be discrete or continuous.
- Scanned images are continuous.
- Often used to store continuously changing values such as elevation.

Resolution measured by cell size.

- Since space is N^2, smaller cell sizes result in much large raster files
- Raster formats: GRID, geoTIFF (both georeferenced), TIFF



Raster vs. Vector

Vector - Advantages

- precision of coordinates
- may have multiple attributes per feature
- less storage space required
- flexible cartography

Disadvantages:

- hard to perform surface analysis
- not easy for continuous data storage

Raster - Advantages:

- simple, robust format
- implicit georeferencing
- stores continuous data
- surface analysis, faster analysis

Disadvantages:

- storage space
- lower precision

Raster versus Vector Question

Question: How many of the following statements are **TRUE**?

1) In a raster if the cell size decreases by half, the space increases by 4 times.
2) Rasters are often useful for scanning or remote sensing applications.
3) A raster typically stores only one numeric data value per cell.
4) A vector format may allow multiple attributes to describe each feature.
5) Vector representation is better suited for continuous data than rasters.

A) 1 **B)** 2 **C)** 3 **D)** 4 **E)** 5

Representing Geographical Data

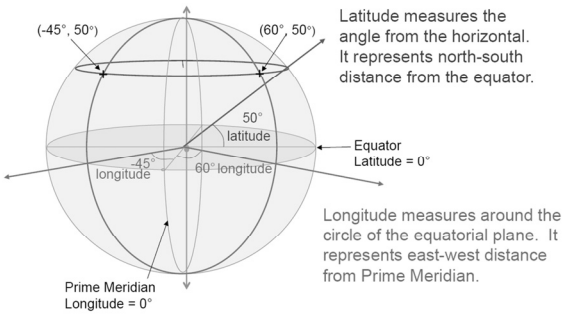
A geographic data set requires a description of its coordinate system for display and analysis, often called the spatial reference.

Components:

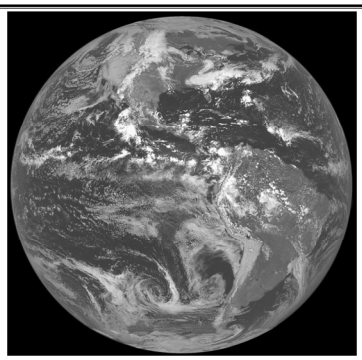
- Geographical coordinate system (GCS) / datum - for assigning coordinates to points on the earth's surface
- Projection - for mapping 3D spherical view to 2D plane
- Storage units (degrees, meters, etc.)
- Resolution - accuracy of the measurements

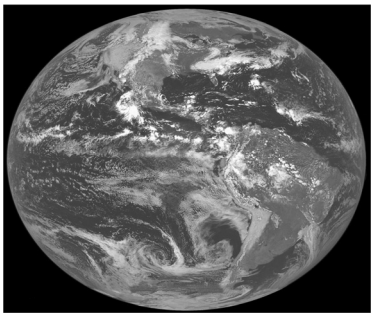
Geographic Coordinate System

Latitude and longitude:



GCS - Earth is not a Perfect Sphere



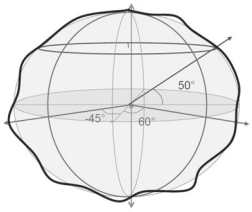


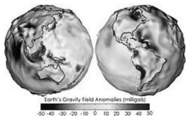
Earth is Not Even a Perfect Ellipsoid

The earth has been approximated by various ellipsoids over time. Current standard is: (Maling, 1989)

Still not perfect as affected by topography which is the height of surface features.

A ***geoid*** is an earth model that takes into account surface height from the centre of earth. (defined by gravity measurements - see https://en.wikipedia.org/wiki/Geoid)





Datum

A ***datum*** is a mapping to minimize the difference between geoid and ellipsoid. Shifts ellipsoid relative to geoid for a particular location.

Datum components:

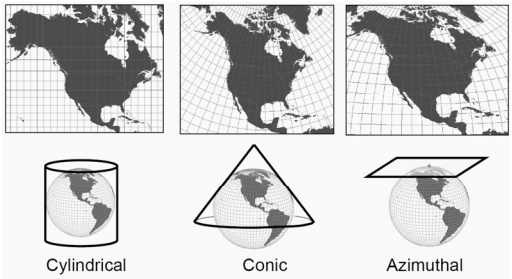
- ellipsoid used
- adjustment or fit (translation of center)

Note this means different datums are incompatible. Make sure you know your datum.

- Example: WGS 84 - reference coordinate system for GPS

Projections

A ***projection*** transforms a spherical coordinate system to a planar coordinate system. Each projection has different benefits and distortions.



GPS Data

GPS units collect points with a datum and projection. (eg. Lat-Lon NAD 1983).

Important to record datum when perform analysis later.

Map Design Process

Determine objectives
- Know audience and purpose, use case

Decide on data and layers required
- What types of data: points, line, area, volume, temporal?

Plan map layout

Choose colors and symbols
- Use colors consistent with understanding (red/green) and real-world
- Use bold colors sparingly. Color/size use strategically for emphasis.

Create!

Coordinate Systems Question

Question: How many of the following statements are **TRUE**?

1) The earth can be modeled as a perfect spheroid.
2) Latitude measures the angle from the horizontal.
3) The zero degree for longitude is the equator.
4) A projection will cause a distortion when representing 3D as 2D.
5) A datum consists of a mapping between a geoid and an ellipsoid.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Google Maps

There are a variety of GIS tools and software to use. We will use Google Maps, specifically Google My Maps, as it is easy to use and handles many of the details for us.

Google My Maps Link: https://www.google.com/maps/d/u/1

Representation is using vector format and layers consisting of single or groups of objects (features classes) can be easily added.

Supports importing data from files, including KML, CSV, and others as well as entering data via searching or map exploring.

KML

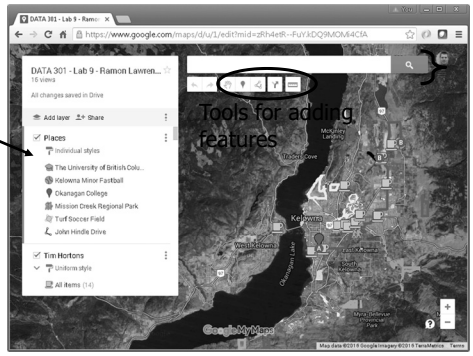
KML or Keyhole Markup Language uses XML to represent geographic information for visualization.
- Supported by Google and international standard in 2008.
- KML file represents features for display on maps with latitude/longitude coordinates.
- Data file is often in zipped form (KMZ files).

Example:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Document>
<Placemark>
  <name>New York City</name>
  <description>New York City</description>
  <Point>
    <coordinates>-74.006393,40.714172,0</coordinates>
  </Point>
</Placemark>
</Document>
</kml>
```


Google My Maps

Define layers with features



Search to find features

Try it: Google My Maps - Pokémon!

Build a map that looks like this.

Note the library has a picture of the library.

Use walking path and custom icons.

Try it: Google My Maps (Create Your Own)

Build a map that has the following:

- Two layers
- Three features per layer
- At least one marker with an image
- An area
- A driving route
- Import an open data set. City of Kelowna data: http://www.kelowna.ca/CM/Page3936.aspx

Suggestion: Add a map with places you have been or would like to visit.

Google Maps API with Python

The Google Maps API can be used with Python to access and manipulate geographical data using a Python program.

- https://developers.google.com/maps/web-services/client-library

Services and features:

- Geocoding and reverse geocoding
- Directions (walking, driving, transit)
- Distance calculations and routes
- Elevations
- Geolocation (based on WIFI and cell towers)
- Road information and speed limits
- Times zones and places (points of interest)

Google Maps API - Getting an API Key

The first step is to get an API key that allows access to the Google services. This API key should be kept private and not shared!

- To get a key you will need a Google account.
- Securing API keys: https://support.google.com/cloud/answer/6310037

Get an API key using Google Developer Console.

- https://developers.google.com/maps/documentation/directions/get-api-key

With directions API, test with:

- https://maps.googleapis.com/maps/api/directions/json?origin=Toronto&destination=Montreal&key=yourkey

Installing Google Maps API for Python

Command:

```
pip install -U googlemaps
```


Python Google Maps API Example

```
import googlemaps
from datetime import datetime

# TODO: Replace the API key below with a valid API key.
gmaps = googlemaps.Client(key='yourkey')

# Use Geocoding API to look up latitude, longitude
address = '3333 University Way, Kelowna, BC, Canada'
geocode_result = gmaps.geocode(address)

print("Geocoding address...")
print("Address:",address,
"Coordinates:",geocode_result[0]["geometry"]["location"])
```


Python Google Maps API Example (2)

```
# Look up an address with reverse geocoding (UBC Van)
lat = 49.2683043
lon = -123.2489377
reverse_geocode_result=gmaps.reverse_geocode((lat, lon))

print("\nReverse geocoding...")
print("Coordinates: ",lat,lon,"Address:",
reverse_geocode_result[0]["formatted_address"])
```

Python Google Maps API Example (3)

```
# Request driving directions between UBCO and UBCV
directions_result = gmaps.directions(address,
         reverse_geocode_result[0]["formatted_address"],
         mode="driving", departure_time=datetime.now())
print("\nDriving directions...")
leg = directions_result[0]['legs'][0]
print("Start address:",leg['start_address'],
      "\nDestination address:",leg['end_address'])
print("Distance:",leg['distance']['text'],
      "Time:",leg['duration']['text'])

for step in leg['steps']:
    print("Step:",step['duration']['text'],
          step['html_instructions'])
```


Conclusion

Geographic Information Systems are systems designed for storing, manipulating, analyzing, and displaying spatial and geographical data.

A GIS supports:

- importing data from various sources in different formats
- organizing the data into layers or groups and integrating data from sources
- displaying the data visually as maps or 3D visualizations to help interpret the data

Understanding how GIS data is encoded using a geographical coordinate system and datums is important when interpreting and combining data from sources.

Google My Maps is an easy-to-use tool for displaying geographical information. The Google Maps API can be used with Python.

Objectives

- Provide examples where a GIS is used
- Define GIS and list some of its features/components
- Appreciate history of GIS including Canadian connection
- List and use GIS features: text, point, line, polygon
- Explain the relationship between features, coordinates, and attributes
- Provide an example on how interval and categorical data is displayed
- Define: scale, precision, resolution and perform simple calculations
- Define: feature class, layer
- Compare and contrast raster versus vector representations
- Define and use latitude and longitude
- Explain the challenge in modeling a point on the earth's surface given that it is not a perfect sphere and has topography
- Explain role and connection between a geoid, spheroid, datum

Objectives (2)

- Explain the purpose of a projection and understand different projections have different benefits and distortions
- Apply a map design process to produce visually appealing maps
- Define and use KML
- Create a map with Google My Maps with various features
- Write a program to access the Google Maps API using Python

DATA 301
Introduction to Data Analytics
Visualization

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Why learn Visualization?

Visualization allows people to understand and extract information faster and with more accuracy than displaying text and numbers.

A good visualization makes data more understandable and reachable to more people.

High quality visualization encourages confidence in the data analysis and inspires people to utilize the data more effectively.

What is Data Visualization? What is Tableau?

Data visualization is the creation and presentation of visual representations of data with the goal to communicate information clearly and efficiently.

- Data visualizations include graphs, charts, images, plots, and tables.
- Data visualization is both an art and a science as it relies on both scientific data analysis and techniques as well as artistic creativity and presentation.

Tableau is a software package designed to make data visualization easy for non-expert users.

Data Visualization with Previous Tools

We have seen data visualization in a variety of other tools including Excel, Python charts, and R.

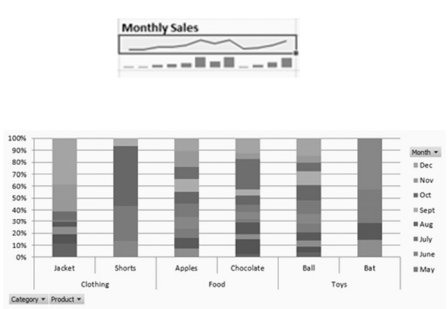
A data visualization package is selected based on its ability to effectively communicate the information to end users and the simplicity in creating the visualizations.

There is no one perfect software package for data visualization as you must trade-off experience, time, and appearance.

Data Visualization in Excel

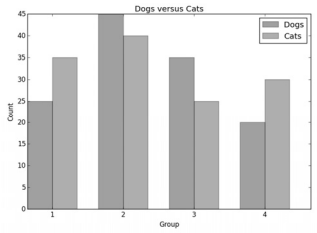
Charts including pivot charts, spark lines, and visual formatting of cells

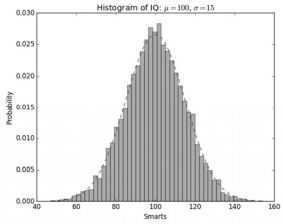
	A	B	C	D	E	F	G
1	Category	Product	Month	Volume	Price	Cost	Revenue
2	Food	Chocolate	Jan	20	$ 2.00	$ 1.00	$ 40.00
3	Clothing	Jacket	Jan	15	$50.00	$35.00	$ 750.00
4	Toys	Ball	Jan	55	$ 1.00	$ 0.50	$ 55.00
5	Food	Chocolate	Feb	80	$ 2.50	$ 1.00	$ 200.00
6	Clothing	Jacket	Feb	10	$50.00	$35.00	$ 500.00
7	Toys	Ball	Feb	65	$ 1.00	$ 0.60	$ 65.00
8	Food	Chocolate	Mar	30	$ 2.00	$ 1.00	$ 60.00
9	Toys	Ball	Mar	70	$ 1.00	$ 0.40	$ 70.00
10	Toys	Bat	Mar	10	$75.00	$50.00	$ 750.00
11	Clothing	Jacket	Mar	8	$50.00	$30.00	$ 400.00
12	Food	Apples	Mar	100	$ 3.00	$ 2.00	$ 300.00
13					Total:		$3,190.00



Data Visualization in Python

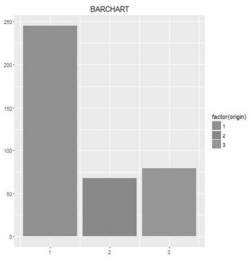
Variety of charting libraries including matplotlib and ggplot





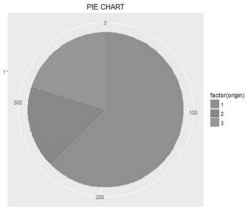
Data Visualization in R - Qualitative Data

Qualitative data: bar chart, frequency table, pie chart



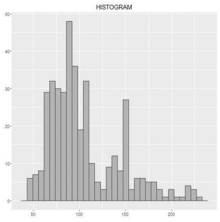
Frequency Table

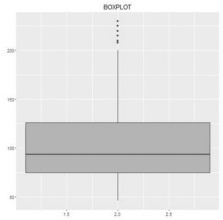
```
  1   2   3
245  68  79
>
```

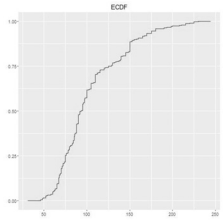


Data Visualization in R - Quantitative Data

Quantitative data: histogram, box plot, ECDF

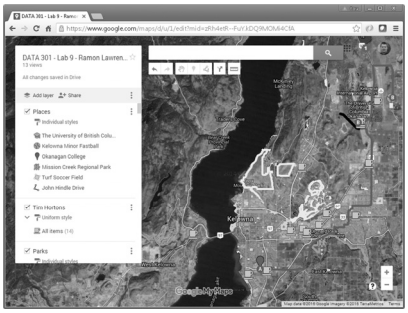


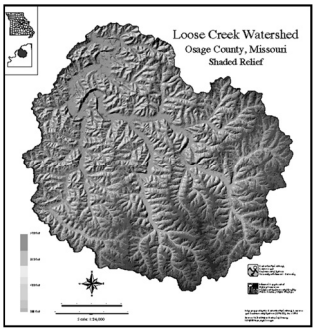




Data Visualization in GIS

Maps of coordinates with overlays (markers, points, lines, regions)





Types of Data

Data can be considered of three types:

- 1) **Known data** - monitoring and regular reporting data for visibility
- 2) **Data You Know You Need** - for understanding outliers or trends in the known data and deciding on how to act on them
- 3) **Data You Need but Do Not Know It** - information that you would have not thought about but knowing it would be very valuable (data to discover!)

Visual analytics helps with all three types of data, but especially the last two to understand trends and discover important information.

Introduction to Tableau

Tableau (http://www.tableau.com/) was founded in 2003 as a spin-off from Stanford University by Chris Stolte, Christian Chabot and Pat Hanrahan.

- 2015 revenue was over $650 million with over 3000 employees

The goal of Tableau is "to help people see and understand their data." - Christian Chabot, Tableau CEO

Tableau has desktop and server (enterprise) products as well as Tableau Public allowing sharing of data sets.

Tableau - Home Page

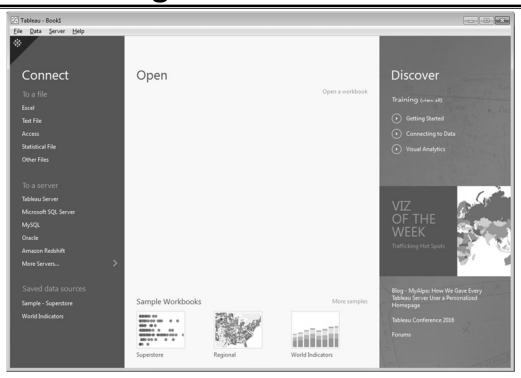


Tableau Workspace

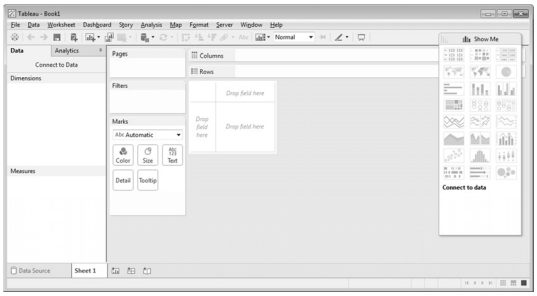


Tableau Features

Supported data types: text, dates, numbers, geographical coordinates (latitude/longitude), Boolean

Aggregation functions: sum, average, max, count, variance, etc.

Many built-in functions for numeric and string manipulation.

Calculated fields can be created and are proceeded by an =.

Tableau Terminology

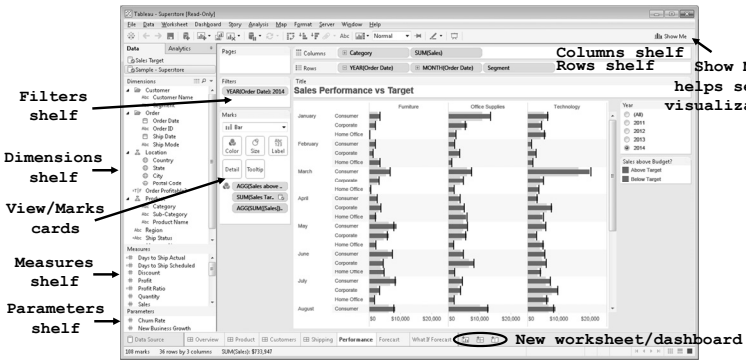
A ***pill*** is a dimension, attribute, or measure that can be placed in the visualization.

- Blue pills are discrete. Green pills are continuous.

A ***shelf*** is a location to put a pill.

- Column shelf, row shelf, filter shelf
- Row and column shelves are similar to Pivot tables in Excel but with built-in visualization.

Tableau Workspace Items



View Cards

View or shape cards allows control of color, shape, and size. They also enable filtering, labeling, and ability to add details on demand.

- Color—expresses discrete or continuous values
- Size—expresses discrete or continuous values
- Label—one or more fields can be expressed as label on marks
- Detail—disaggregates the marks plotted
- Tooltip/tooltips—makes fields available to tooltips without disaggregating data
- Shape—expresses discrete or continuous fields

Multiple fields can be placed on the color, label, detail, and tooltip buttons.

Show Me Button

The Show Me button suggests visualization to use based on your current dimensions and measures.

It will also place pills on shelves automatically.

Tableau Visualization: Show Me

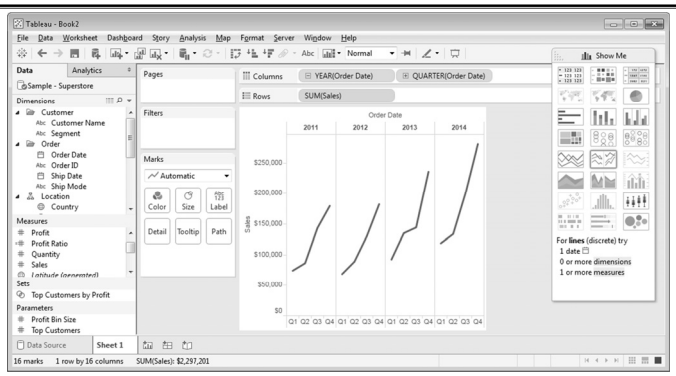


Tableau Question

Question: How many of the following statements are **TRUE**?

1) In Tableau blue pills are continuous.
2) The View Cards interface allows for changing color and size of features in the visualization.
3) A shelf is a location to place a pill.
4) The Show Me button will suggest visualizations for you.
5) A pill for a dimension may be on more than one shelf at the same time.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Tableau Visualizations

1) Install Tableau. Use trial version or student license provided in Connect.

2) Start Tableau. Use the sample.twbx file or the Superstore example and explore the visualizations.

3) Try create any visualization of the data.

Tableau - Data Sources

Tableau can connect to a wide variety of data sources including:

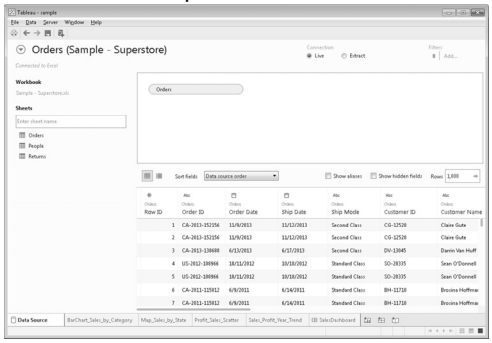
- Microsoft Excel and Access
- Text files (txt, csv)
- Relational databases (MySQL, SQL Server, Oracle, PostgreSQL)
- NoSQL databases (MongoDB)
- Parallel and analytical databases (Greenplum, Vertica, Teradata)
- Other ODBC sources (note JDBC is not supported)

A sample data source called Superstore is available in the Tableau/defaults/Datasources directory.

- File: Sample - Superstore.tds (Tableau Data Source) or
- File: Sample - Superstore.xls (Excel file)

Example Connecting to Excel

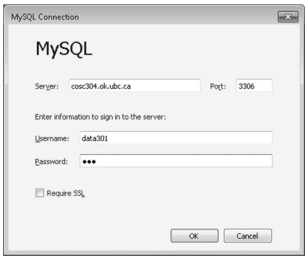
Select Excel file and then pick sheets.



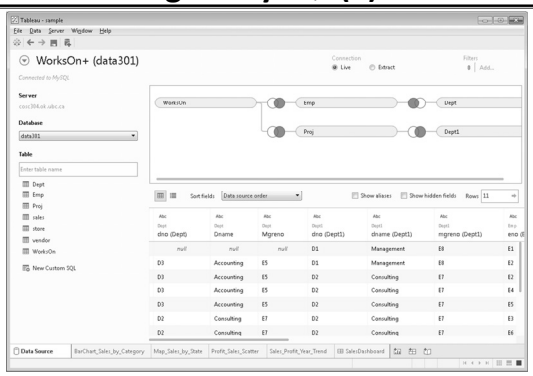
Example Connecting to MySQL

Connecting to a relational database like MySQL requires:

- 1) Driver (often need to download from database vendor)
  - https://www.tableau.com/support/drivers
- 2) Database connection information



Example Connecting to MySQL (2)



Connect or Extract Data

Tableau has its own internal data engine. There are two options when retrieving data to visualize:

1) Direct connect to source to get live data
- Can refresh data using F5 or selecting refresh menu item
- May be faster depending on data set/visualization

2) Extract and import data into Tableau's data engine
- May get a performance improvement as data is local
- May set certain times to extract
- Portability (as consumer of report does not need access to data source)
- Support for functions not supported by source (e.g. Excel)

Tableau Data Sources Question

Question: How many of the following statements are **TRUE**?

1) Tableau can connect to many relational databases.
2) Tableau can process data in text and Excel files.
3) Tableau can either leave data in data source or extract it locally.
4) Tableau can connect to data sources using JDBC.
5) Tableau will try to identify types and relationships from the data sources.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Tableau Data Sources

Use Tableau to connect to Excel and MySQL data sources.
- Start Tableau. Open up Superstore Excel data source (either XLS or TDS file) in Tableau/defaults/Datasources directory.
- Install the MySQL ODBC connector from: https://dev.mysql.com/downloads/connector/odbc/
- Server: cosc304.ok.ubc.ca Database: data301 User: data301 Password: ubc

Superstore visualizations:
- Map showing profit by state.
- Visualization to indicate what is the best selling product category per store.

WorksOn visualizations:
- Visualize the number of projects, employees, and hours worked per department.
- Visualize employee ages to see if age impacts if they are supervisors.

Tableau Files

Tableau Workbook (.twb) (default) - saves workbook but no data

Tableau Packaged Workbook (.twbx) - contains data and visualization for easier sharing

Tableau Datasource (.tds) - metadata on a data source

Tableau Bookmark (.twb) - one worksheet within workbook

Tableau Data Extract (.tde) - compressed snapshot of data stored in column format

Note similarities with Excel/spreadsheet terminology.

Joining Tables

When connecting tables **R** and **S**, there are four types of joins:
- **INNER JOIN** - row in result for each row of R that matches a row of S
- **LEFT OUTER JOIN** - row in result for each row of R that matches a row of S OR a row of R that does not match anything in S
- **RIGHT OUTER JOIN** - row in result for each row of R that matches a row of S OR a row of S that does not match anything in R
- **FULL OUTER JOIN** - row in result for each row of R that matches a row of S OR a row of R that does not match anything in S OR a row of S that does not match anything in R

Join Example

Boys

Bid	BoyName
1	Joe
2	Steve
3	Fred
5	James

Girls

Gid	GirlName
2	Jane
4	Sarah
5	Fran
6	Julie

Boys INNER JOIN Girls

Bid	BoyName	Gid	GirlName
2	Steve	2	Jane
5	James	5	Fran

Boys LEFT OUTER JOIN Girls

Bid	BoyName	Gid	GirlName
1	Joe		
2	Steve	2	Jane
3	Fred		
5	James	5	Fran

Boys FULL OUTER JOIN Girls

Bid	BoyName	Gid	GirlName
1	Joe		
2	Steve	2	Jane
3	Fred		
		4	Sarah
5	James	5	Fran
		6	Julie

Boys RIGHT OUTER JOIN Girls

Bid	BoyName	Gid	GirlName
2	Steve	2	Jane
		4	Sarah
5	James	5	Fran
		6	Julie

Join Question

Question: Given these tables, how many rows are in the result of Boys LEFT OUTER JOIN Girls ON Bid=Gid?

Boys

Bid	BoyName
1	Joe
1	Steve
3	Fred
5	James
7	Ben
7	Bishop

Girls

Gid	GirlName
1	Jane
1	Sarah
5	Fran
6	Julie

A) 9 **B)** 8 **C)** 7 **D)** 6 **E)** 0

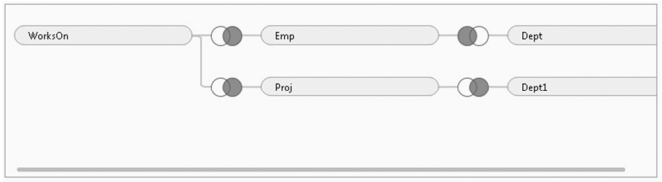
Data Blending

Data blending allows "joining" data that does not reside in a single source. There are automatic and manual methods.

- Automatic - field names must match across sources. Will link secondary data source with primary data source.
- Manual methods include ability to specify SQL statement to perform with join.

Try it: Tableau Data Sources - Joins

Using the MySQL tables in the data301 database, create some joins to connect them so it looks like this:



Create a visualization with this data set.

Dynamic Grouping/Renaming

Dynamic grouping (also called ad hoc groups) can be created by using Ctrl+Select to select elements in visualization and select Group from menu.

It is also possible to rename values/labels and correct value errors.

Geographic Data

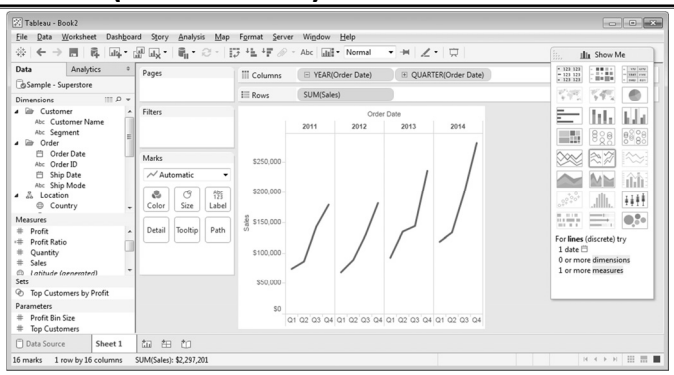
For geographic data (small globe icon), Tableau automatically generates center-point geocodes (longitude/latitude).

Tableau Chart Types

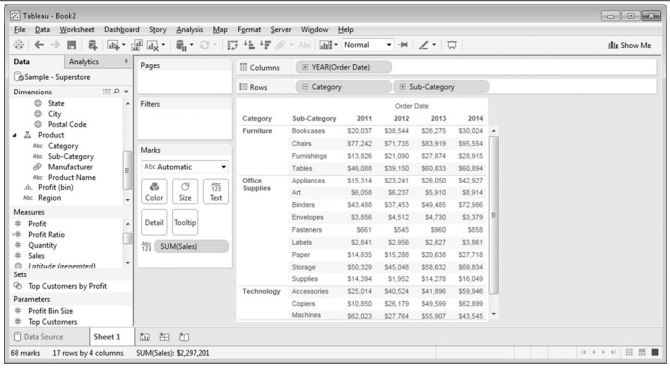
Chart types:

- text tables/crosstabs
- maps
- heat maps, highlight tables, tree maps
- line charts
- area fill charts and pie charts
- scatter plot, circle view, side-by-side plots (identify outliers)
- bullet graph, packed bubble, histogram, Gantt charts

Line Chart (discrete time)

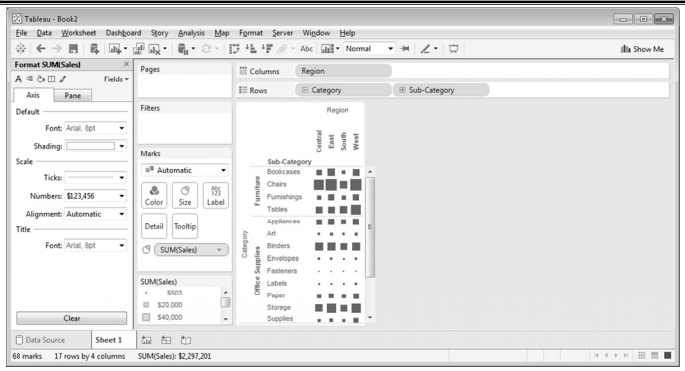


Text Table (Crosstab)

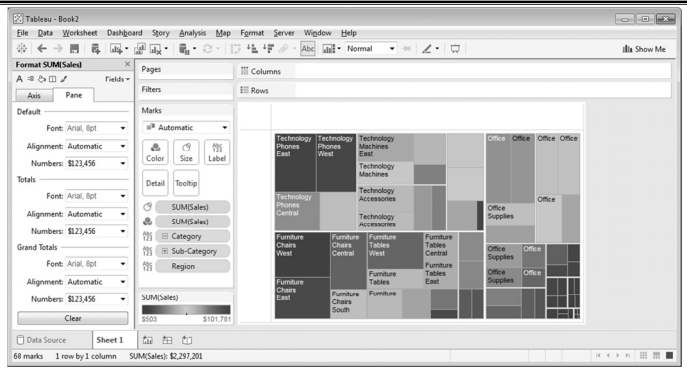


Maps

Heat Map

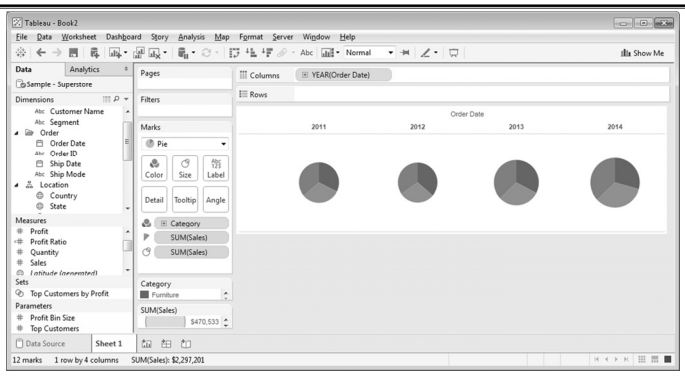


Tree map

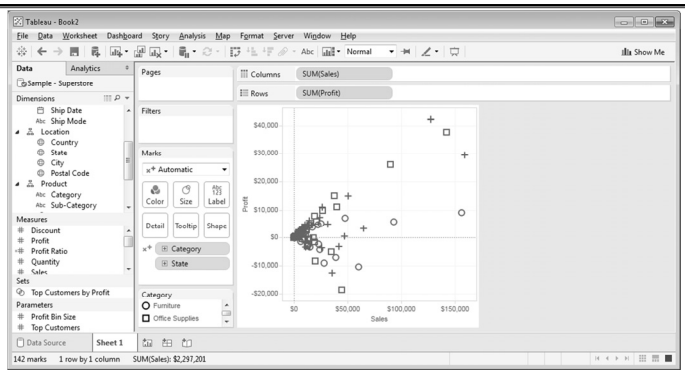


Bar Charts

Pie Charts



Scatter Plots

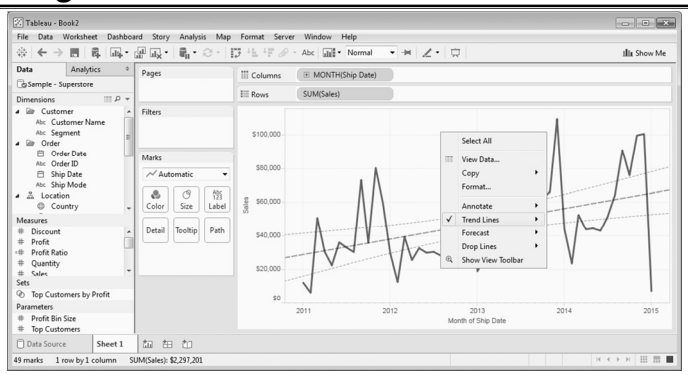


Trend Lines and Reference Lines

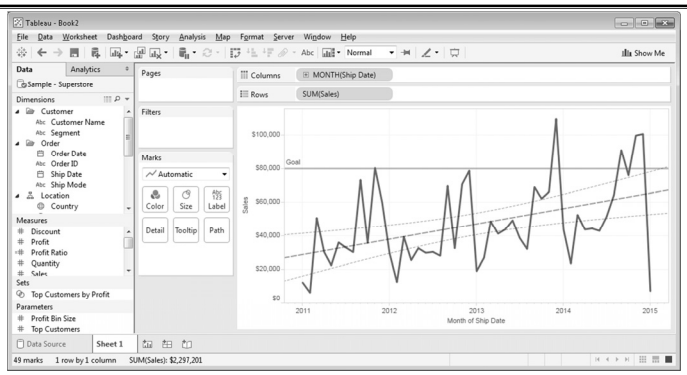
Trend lines show patterns in data using a line of best fit.

Reference lines allow comparison with a reference (detect trends and outliers).

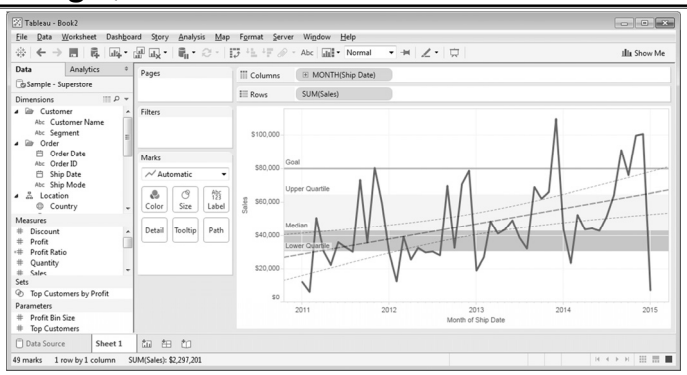
Adding Trend Lines



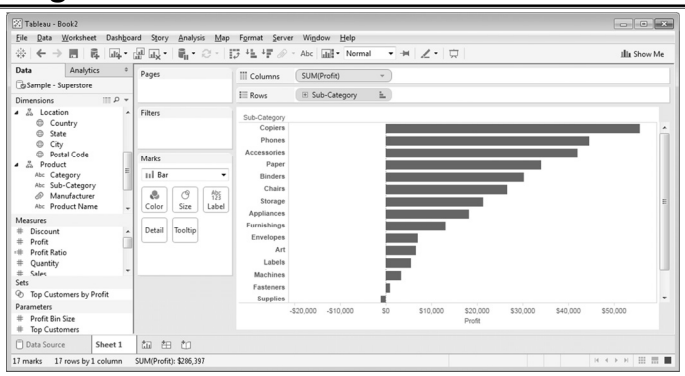
Trend Lines and Reference Lines



Adding Quantiles



Sorting



Hierarchies

Hierarchies are groupings of data that make it easier to roll-up and drill-down into data.

Examples:

- category and subcategory
- year, quarter, month
- country, state, city

Can create own hierarchies by dragging dimensions on top of each other.

Filters

There are multiple ways to define filters:

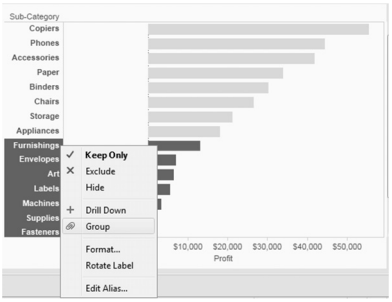
1) Drag dimension into filter shelf

2) Quick filters allow people using report to change filters dynamically. (click on item in filter shelf and select Show Filter option)

Grouping

Grouping allows summarizing data without using a hierarchy.

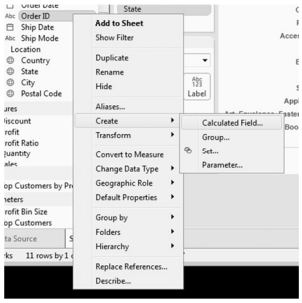
- Multi-select elements then in pop-up menu select Group



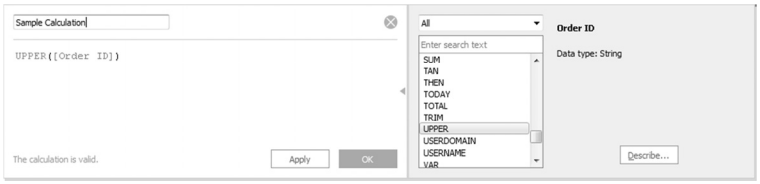
Calculations

Calculated fields are performed on data source when possible.

Table calculations are performed locally in Tableau.



Creating a Calculated Field



Parameters

Calculations may have parameters.

Parameters may be exposed in the visualization so the user can control them.

Forecasting

Right click, select Forecast then Show Forecast.

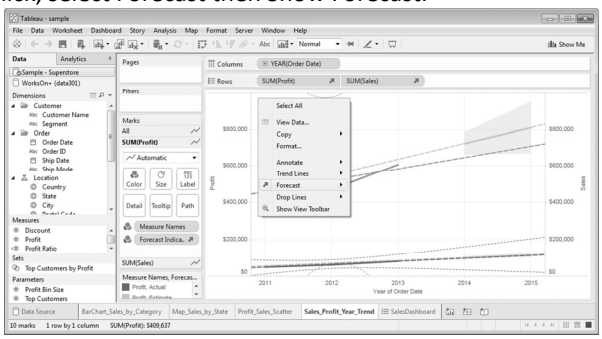


Tableau Charts Question

Question: How many of the following statements are **TRUE**?

1) There can only be one pill on the row shelf.
2) A trend line can only be linear.
3) A user can group multiple items into a group in the visualization.
4) Calculated fields are calculated on the data source if possible.
5) Filters may be exposed to the user of the visualization just like parameters.

A) 0 **B)** 1 **C)** 2 **D)** 3 **E)** 4

Try it: Tableau Charts

Using the Superstore data set, create a visualization for each of these chart types:

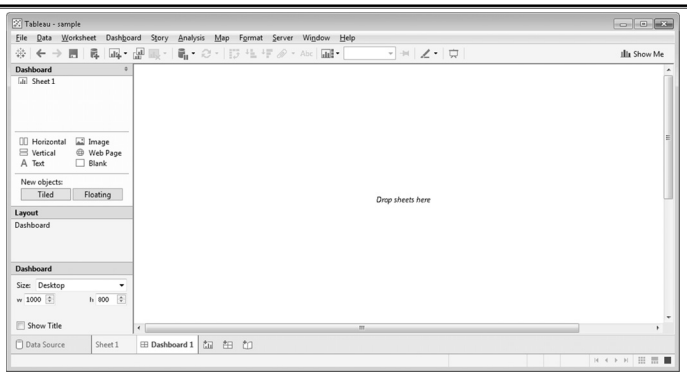
- line chart (with forecast and trend line)
- bar chart (with filters and sorting)
- pie chart (with a parameter)
- heat map (with grouping)
- scatter plot (with a calculated field)
- histogram
- circle view

Dashboards

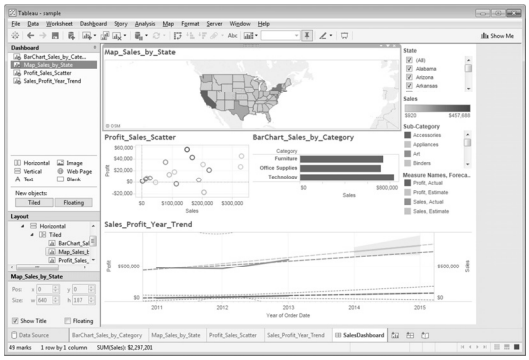
A ***dashboard*** consists of multiple sheets organized to make information and its relationships more understandable.

Tableau recommendation: 4-pane dashboard designs

Dashboard Starter View



Dashboard Populated with Worksheets



Try it: Tableau Dashboard

Using the Superstore data set, create your own dashboard with multiple visualizations.

Conclusion

Tableau is a software system for visualizing data sets from multiple sources using a wide-range of visualization techniques.

- line charts, bar charts, scatter plots, heat maps, pie charts, histograms

Visualization of data sets is critical for communicating meaning and understanding, especially for people with less understanding of the data set.

Objectives

- Explain the purpose of visualization
- List different types of visualizations available in Excel, Python, R, GIS
- List the three "types of data"
- Define: pill, shelf, view card (as used in Tableau)
- Explain the purpose of the Show Me button
- Be able to connect to Excel and relational databases using Tableau
- Compare/contrast connecting to versus extracting data with Tableau
- List and explain the different Tableau file types
- Define and compute: inner join, left outer join, right outer join, full outer join
- Use dynamic grouping and renaming to clean and correct data values in a visualization

Objectives (2)

- List and use the different Tableau chart types: text tables, maps, heat maps, tree maps, line charts, pie charts, area charts, scatter plot, circle view, histogram, Gantt charts
- Add trend lines, references lines, quantiles to a visualization
- Create and use hierarchies
- Create and use filters
- Create calculated fields
- Use parameters to allow user-controlled visualizations
- Add forecasts to a visualization
- Organize visualizations into a dashboard

DATA 301
Introduction to Data Analytics
Open Data

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

What is Open Data?

Open Data is the movement to make data freely available to all with no restrictions on use or copyright.

Governments have been major supporters and providers of open data as data collected by governments is primarily done to benefit its citizens.

Corporations and other organizations are both producers and consumers of open data.

Open Data in Canada

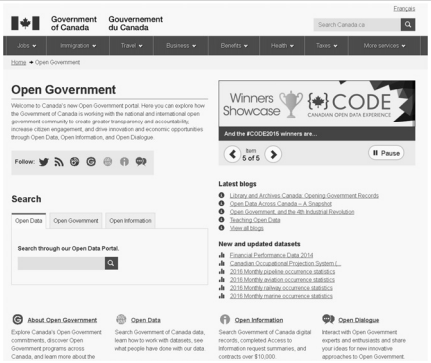
Federal, provincial, and local governments have all been involved in the open data movement.

Canadian Federal government: http://open.canada.ca/en
- How to use: http://open.canada.ca/en/working-data
- Statistics Canada: http://www.statcan.gc.ca/eng/rdc/data

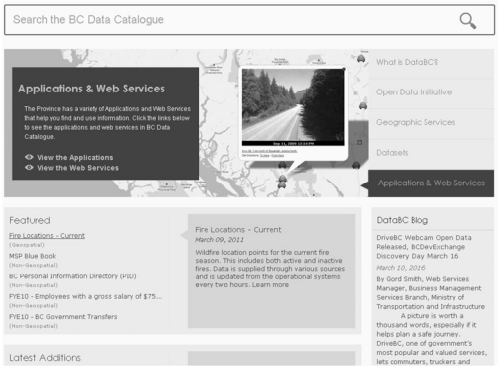
British Columbia government: http://www.data.gov.bc.ca/

City of Kelowna:
https://www.kelowna.ca/city-services/city-maps-open-data/open-data-catalogue

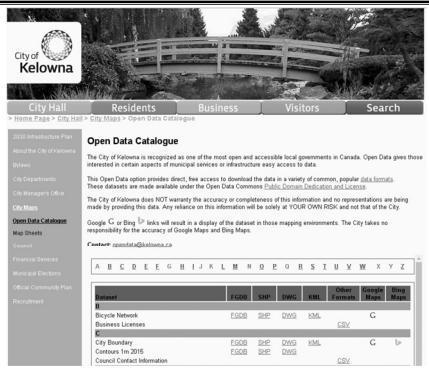
Open Data in Canada



Open Data in BC



Open Data in Kelowna



Open Data in United States

United States government: https://www.data.gov/

Individual states have their own open data sites as well.

- Example: Washington state: https://data.wa.gov/

United States: Data.gov

Open Data Worldwide

UK: http://data.gov.uk

The World Bank: http://data.worldbank.org/

- Financial information and statistics

United Nations: http://data.un.org/

OECD: https://data.oecd.org/

Open Data Aggregators

There are many sites that aggregate open data sets (and some data sets for a cost). A Canadian based site is Quandl (http://www.quandl.com).

Kaggle provides many data sets and competitions and techniques for data analytics and machine learning.
https://www.kaggle.com/datasets

Open Data from Companies

Many companies either have public data or application programming interfaces (APIs) that allow people to use their data.

- Google: https://www.google.com/publicdata/directory (public data explorer) and https://developers.google.com/maps/ (Google Maps API)
- Facebook: https://developers.facebook.com/ (API)
- reddit: https://www.reddit.com/dev/api (API)
- Twitter: https://dev.twitter.com/rest/public (API)
- Amazon: https://aws.amazon.com/public-data-sets/ (public data sets) and https://developer.amazon.com/ (API for developers)
- Best Buy: https://developer.bestbuy.com/ (API)

Try it: Open Data

Explore the federal, provincial, and City of Kelowna data sets to discover "something interesting". Report to your neighbors and to the class.

From any Canadian government open data site, retrieve a data set and analyze and visualize it using one of our tools: Excel, R, Python, Tableau.

Open Data for Researchers

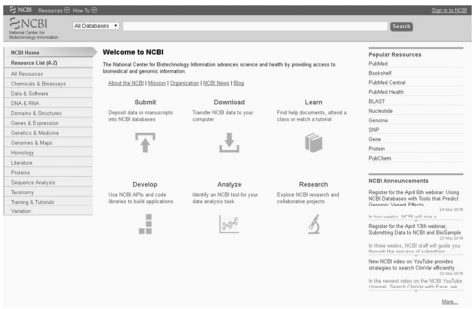
Increasingly publicly funded researchers are responsible for making their data sets, procedures, and results available to the public (and other researchers).

- Canadian researchers funded by NSERC, SSHRC, CIHR must make their publications freely available within 12 months of publication.
- Researchers in bioinformatics and other fields must make their data sets publically available in a database or repository.

Researchers benefit by having access to public data sets and data sets of other researchers, but there is also a challenge as producing data sets (and perhaps commercializing results) may restrict open access.

Open Data Biology/Bioinformatics

Huge number of databases with most related to NCBI (http://www.ncbi.nlm.nih.gov/) but distributed world-wide.



Open Data in Chemistry

ChEMBL (https://www.ebi.ac.uk/chembl/) stores structures and properties of pharmacologically active molecules.

- Over 1.5 million compounds.

SureChEMBL (https://www.surechembl.org) is a database extracted automatically from patent applications.

- Growing at 80,000 compounds a month and has 16 million compounds from over 13 million annotated patents.

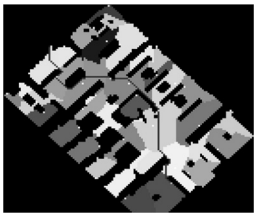
ChemSpider (http://www.chemspider.com/) is a free chemical structure database containing over 43 million structures.

- Supported and hosted by Royal Society of Chemistry.

Open Data in Computer Science

Computer scientists in various fields create standardized data sets for experimentation and research.

- Databases: Standard performance benchmarks such as TPC (www.tpc.org).
- Machine learning/data mining: UCI ML repository http://archive.ics.uci.edu/ml/
- Game path finding: http://www.movingai.com/benchmarks/



Open Data in Earth/Environment Science

- Climate Change Data Portal: http://sdwebx.worldbank.org/climateportal/
- National Climatic Data Center: https://www.ncdc.noaa.gov/cdo-web/
- National Geographic Data Center: http://www.nodc.noaa.gov/submit/
- Polar Data Catalog: https://www.polardata.ca/

Open Data in Physics

Modern physics produces a HUGE amount in data in experiments like astronomical observations and the Large Hadron Collider.

- New research systems developed to handle the large amount of data produced.
- CERN open data portal: http://opendata.cern.ch/
- Data produced is tens of petabytes/year. Large distributed computing of 170 facilities in 36 countries.

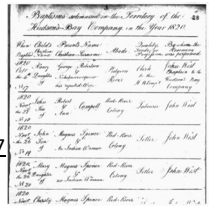
Astronomy:

- Canadian Astronomy Data Centre: http://www3.cadc-ccda.hia-iha.nrc-cnrc.gc.ca/cadc/
- National Space Science Data Center: http://nssdc.gsfc.nasa.gov/

Open Data in Psychology and Social Sciences

Archaeology:

- Archaeology Data Service: http://archaeologydataservice.ac.uk/
- Many museums have online exhibits and open data.



Psychology:

- Journals increasing requiring open data sets.
- List of open data sites at: http://guides.library.ucla.edu/c.php?g=180221&p=1188487

History

- Digital Archive Database Project (UBC): http://dadp.ok.ubc.ca

Google Analytics

Google Analytics is an analysis service for tracking, optimizing, and understanding user interaction with a web site/service.

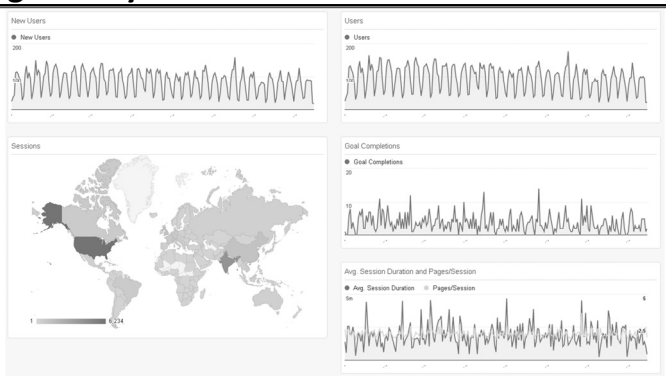
Using Google analytics is important for all business, but especially web companies, that rely on users interacting with their site to generate revenue and sales.

Google analytics helps identify and improve content to make it more accessible to potential customers.

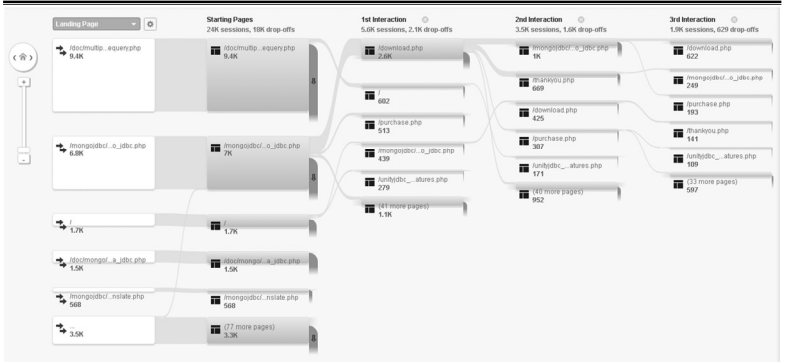
- Very important skill set for business owners and managers.

Google Analytics - Audience Overview

Google Analytics - Traffic Dashboard



Google Analytics - Behaviour Flow



Google Adwords

Google AdWords is a service to provide advertisements during searches and as display advertisements on web sites and in apps.

- Primary source of revenue for Google. https://www.google.ca/adwords/
- Companies bid on *keywords* and display opportunities that are presented by Google and affiliated sites.

Terminology:

- *Ad Impression* - display of an advertisement. Pricing in cost-per-thousand impressions or cost per mille (CPM).
- *Click through* - user clicks on an advertisement (and directly to new location)
- *Click through rate* - fraction of impressions that are clicked on
- *Pay-per-click (PPC)* - companies are billed on each click of an advertisement. The pricing depends on the bid amount and the desirability of the ad location.

Conclusion

Open Data is the movement to make data freely available to all with no restrictions on use or copyright.

Open data has been widely supported by governments and companies wishing to engage users (and developers) with their services.

Data analysts should use open data to help with their analysis whenever available.

Researchers are often responsible for making their publications and data available in an open fashion.

Google provides services for analytics and advertising that are valuable to understand as a business or site looking for user traffic.

Objectives

- Define open data and explain the motivations for making data "open".
- List some of the governments and organizations that provide data in an open fashion.
- Use open data sets when applicable when performing data analysis.
- Explain the role of Google Analytics and Google AdWords. Compare and contrast what these two services provide.

DATA 301
Introduction to Data Analytics
Course Summary

Dr. Ramon Lawrence
University of British Columbia Okanagan
ramon.lawrence@ubc.ca

Data Analytics

Data analytics is the processing of data to yield useful insights or knowledge.

- Data processing involves finding, loading, cleaning, manipulating, transforming, modeling, and visualizing the data.
- The knowledge may be used for scientific discovery, business decision-making, or a variety of other applications.
- Vital in a "data-driven" world with larger and more critical data sets.

A ***data analyst*** is a person who uses tools and applications to transform raw data into a form that will be useful.

- Data analyst jobs are projected to be one of the top jobs over the next 10 years, and this course has provided training in the skills needed for those jobs.

Skills and Capabilities

Skills are tools, software, and techniques that you can use **today** to solve your problems.

- Excel, Excel VBA, SQL/databases, command line, Python and Python libraries for data analysis/visualization, R, GIS (Google Maps), Tableau

Capabilities and ***concepts*** are fundamental principles and knowledge that applies to many situations. They are the building blocks of future learning.

- programming concepts (Python), data representation/metadata, thinking algorithmically, designing, manipulating and cleaning data, querying and filtering data, statistical analysis, visualizing information

Course Summary

The course goal was to:

Understand data analytics and be able to apply data analysis to data sets using a variety of software tools and techniques

We developed a variety of skills and capabilities and applied them to scientific and business data sets.

The most exciting aspect of data analytics is discovering and presenting useful data/information that can have an impact on business, society, etc. You have the ability and skill set to perform data analysis.

Putting it all together … Going Forward

This course has started you on the path of data analytics.

Computer systems and technology ***will*** change (the skills), but it is the ***attitude*** and the ***concepts*** that are most important.

How much information in the course will you remember?

How much do you *need* to remember to apply the concepts?

As an experienced data analyst, you can:

- Solve real-world problems on data sets using skills and techniques learned.
- Learn and use new systems with confidence by applying gained knowledge, experience, and fundamental concepts.
- Critically evaluate data analysis done by others and determine when and how to apply tools for your own data analysis problems.

Next Steps

At UBC:

- COSC 101 (Digital Citizenship), COSC 122 (Computer Fluency), COSC 123 (Computer Creativity), COSC 111/121 (Java), COSC 304 (DB), COSC 341 (HCI)
- Stats related: GEOG 271/GEOG 272, BIOL 202, HMKN 205, PSYO 270/271, PSYO 372/373, STAT 230, Data Science program
- VISA 108 (Media studies), MGMT 350 (IT Mgmt.), BIOL 420 (Bioinformatics), GEOG 370 (GIS)
- New Masters of Data Analytics (2018) and follow-on to DATA 301 (future).

Online:

- Coursera and EdX have several courses on data analytics and R.

The best next step is to apply your knowledge to real-world problems.

Go analyze some data!

Thank you for a great course!

Good luck on the exam!