Tools for Data Science

TS4DS

Achraf Cohen, PhD

12/31/22

Table of contents

Welcome to TFDS

This eBook is used for **Tools and Statistics for Data Science** (TS4DS) courses offered at the University of West Florida, Hal Marcus College of Engineering and Science.

An 8-Week course on **Tools for data science** using R, Python, and SQL. Throughout the course, there will be hands-on exercises with computing resources. The course will include introductions to several packages in R, particularly `Tidyverse`, libraries in Python such as `Pandas`, `NumPy`, and `matplotlib`. SQL clauses including `joins`, `sub-queries`, and summary statistics.

Topics

- Introduction to R/RStudio/Quarto
- R Programming
- Python Programming
- Introduction to SQL
- Data input and output
- Data manipulation

- Summary statistics
- Graphics and Data visualization

Readings

In addition to material provided in this course, it is highly encouraged reading and reviewing some of the material, I will be pointing out throughout the course, including:

- *R for Data Science* (Wickham and Grolemund 2016). It is available free online.
- *Hands-On Programming with R* (Grolemund 2014). It is available free online.
- *Exploring Enterprise Databases with R: A Tidyverse Approach* (John David et al. 2020)
- *Mastering Spark with R* (Luraschi, Kuo, and Ruiz 2019). It is available free online.
- *Practical Guide for Oracle SQL, T-SQL and MySQL* (Zhang 2017)
- *Think Python* (Downey 2015)
- *Data Science and Analytics with Python* (Rogel-Salazar 2018)

R Basics

At the end of this week, you will be able to:

- Show an understanding of data science workflow
- Start your computing environment
- Write your first R script
- Practice with basic R programming tools such as `for loop`, `data frames`, etc.

Data Science?

The use of data as evidence is crucial but, it is not something novel. If we examine a definition of the field of *statistics*, we can observe that is given as four subtopics:

- Data Collection
- Data Analysis
- Results Interpretation
- Data Visualization

Originally, *statistics* was viewed as the analysis and interpretation of information about states. And science is understood as organized knowledge in the form of testable explanations and predictions about the universe.

So, what is data science? Data science is more than just using statistics and data to answer scientific questions.

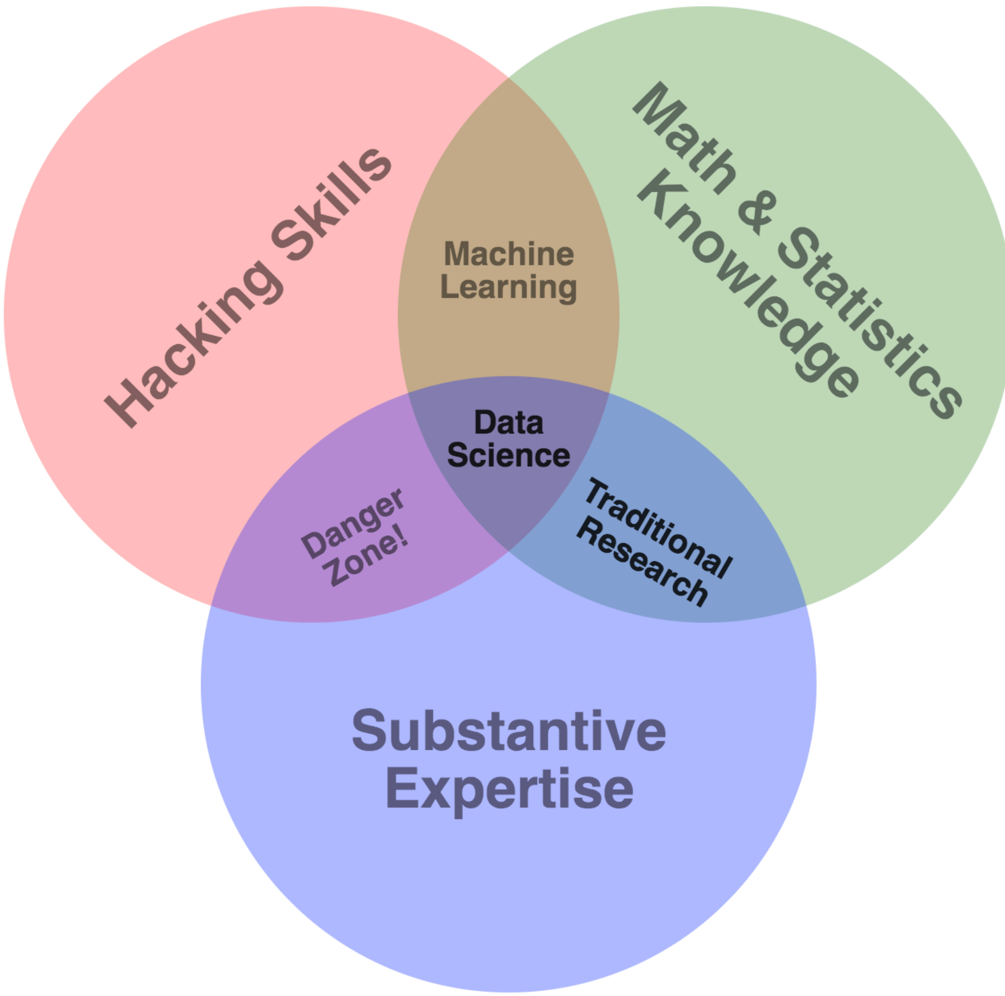
> Nowadays, data science is viewed as the use of various sources of data to extract knowledge and provide insights using multiple skills including programming, math and statistics, and communication.

Venn diagram by Drew Conway provides a visualization on data science.

Data Science Venn diagram by Drew Conway

Typical examples of data science projects:

- **Market analysis** What product will sell better in conjunction with another popular product


Hacking Skills
Math & Statistics Knowledge
Substantive Expertise
Machine Learning
Data Science
Danger Zone!
Traditional Research

- **Market segmentation** Are there distinguishable features that characterize different groups of sales agents, customers or businesses?
- **Advertising and marketing** What advertisement should be placed on what site?
- **Fraud** How to detect if a retail/finance transaction is valid or not?
- **Demand forecasting** What is the demand for a particle service at a specific time/place?
- **Classification** Emails classification (spam vs. valid email)

Tools for Data Science

Data science helps managers, engineers, policymakers, and researchers - almost everybody - to make informed decisions based on evidence from data. Computers and technologies have empowered how much data we can store, manipulate, and analyze. To enable these functions, technologies and tools are developed to help us to be more productive and efficient when conducting data science projects.

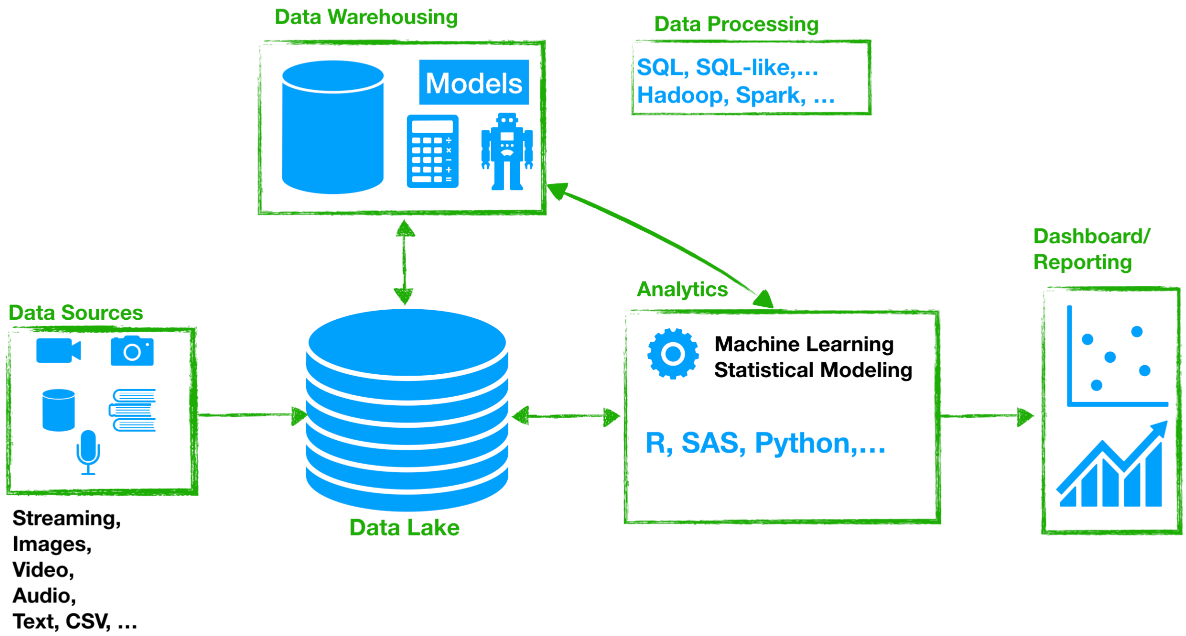


Figure 1: Data Science Workflow

The technologies deployed in the analytics and data science have advanced very fast and multiple open source projects exist, for example:

- Data framework: Hadoop, Spark,...
- Query Languages: SQL, SQL-like,...
- Data manipulation, modeling, and graphing: R, Python,...
- Software management: Git, GitHub,...

Data Science Workflow

Often, the data science process is iterative. Some steps in the data science workflow include:

1. Specify the question of interest (business understanding, scientific goal, predict or estimate,…)
2. Collect data (internal, external, sampled, relevant, ethics,…)
3. Manipulate data (explore, transform, merge, filter,…)
4. Model data (machine learning, statistics, probability, fit, validate,…)
5. Communicate and interpret the results (storytelling, visualization, dashboard, reports,…)
6. Deploy and monitor models

Introduction to R / RStudio

The two programming languages we cover in this course are R and Python. These are both open source programming languages. Let's start off with R.

A few features of R are:

- **R** is a free software environment for statistical computing and graphics.
- It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS.
- The R project website contains a lot of useful information about: download R, documentation and manuals, The R journal, books related to R, and R packages by topics
- There is a large active community of R users.

RStudio is an integrated development environment (IDE) for R and Python, with a console, syntax-highlighting editor that supports direct code execution, and tools for plotting, history, debugging and workspace management. It compiles and runs on a wide variety of UNIX platforms, Windows and MacOS. The RStudio website. There is an open source license for both the desktop and server versions that you install for free from here: download RStudio

R Markdown provides an authoring framework for data science. You can use a single R Markdown file to both save and execute code generate high quality reports that can be shared with an audience. R Markdown documents are fully reproducible and support dozens of static and dynamic output formats. The R Markdown 1-minute video provides an overview of what R Markdown can do!

Install your R/RStudio

For TFDS, we will be using RStudio Server hosted at UWF. This is the link https://rstudio.hmcse.uwf.edu/. **Login using your UWF account**.

You don't need to install R and RStudio on your computer. But, you are welcome to do so if you wish so.

Getting started with R

Recordings of this week provide lessons about R, RStudio, and GitHub. The following will be covered:

- RStudio (editor, console, global Env., and etc.)
- R (scripts, packages, help)
- GitHub and connection to RStudio
- R Markdown - Cheet Sheet
- My first R script - the basics
    - Values, vectors, matrices, factors, data.frames, lists. Here is an example of code:

```
# assign a value to object named "x"
x = 1
# or
x <- 1
1 -> x
# Calculator
x=10^2
y=2*x
# vectors
c(1,21,50,80,45,0)
```

```
[1]  1 21 50 80 45  0
```

```
c("d","4","r")
```

```
[1] "d" "4" "r"
```

```
# characters
"R is useful and cool"
```

```
[1] "R is useful and cool"
```

```r
# boolean - TRUE or FALSE
45>96
```

```
[1] FALSE
```

```r
# built-in functions
sum(1,3,5)
```

```
[1] 9
```

- Statistical and mathematical functions: An example of code:

```r
# a vector / array
vec1= c(1,21,50,80,45,0)
# minimun
min(vec1)
```

```
[1] 0
```

```r
# maximum
max(vec1)
```

```
[1] 80
```

```r
# exponential function
exp(vec1)
```

```
[1] 2.718282e+00 1.318816e+09 5.184706e+21 5.540622e+34 3.493427e+19
[6] 1.000000e+00
```

```r
# cosine function
cos(vec1)
```

```
[1]  0.5403023 -0.5477293  0.9649660 -0.1103872  0.5253220  1.0000000
```

```r
# sine function
sin(vec1)
```

```
[1]  0.8414710  0.8366556 -0.2623749 -0.9938887  0.8509035  0.0000000
```

```r
# logarithm function of base e
log(vec1,0.5)
```

```
[1]  0.000000 -4.392317 -5.643856 -6.321928 -5.491853       Inf
```

```r
# square root
sqrt(vec1)
```

```
[1] 1.000000 4.582576 7.071068 8.944272 6.708204 0.000000
```

```r
# logarithm function of base 10
log10(10)
```

```
[1] 1
```

```r
# logarithm function of base 2
log2(2)
```

```
[1] 1
```

```r
# logarithm function of base 45
log(45,base = 45)
```

```
[1] 1
```

```r
# factorial
factorial(3)
```

```
[1] 6
```

```r
# binomial coefficient / combination
choose(10,5)
```

```
[1] 252
```

- Summary statistics, random number generation. An example:

```r
# a set of values
vec1= c(1,21,50,80,45,0)
# summation
sum(vec1)
```

```
[1] 197
```

```r
# arithmetic mean
mean(vec1)
```

```
[1] 32.83333
```

```r
# standard deviation
sd(vec1)
```

```
[1] 31.30122
```

```r
# summary statistics
summary(vec1)
```

```
 Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
 0.00    6.00   33.00   32.83   48.75   80.00 
```

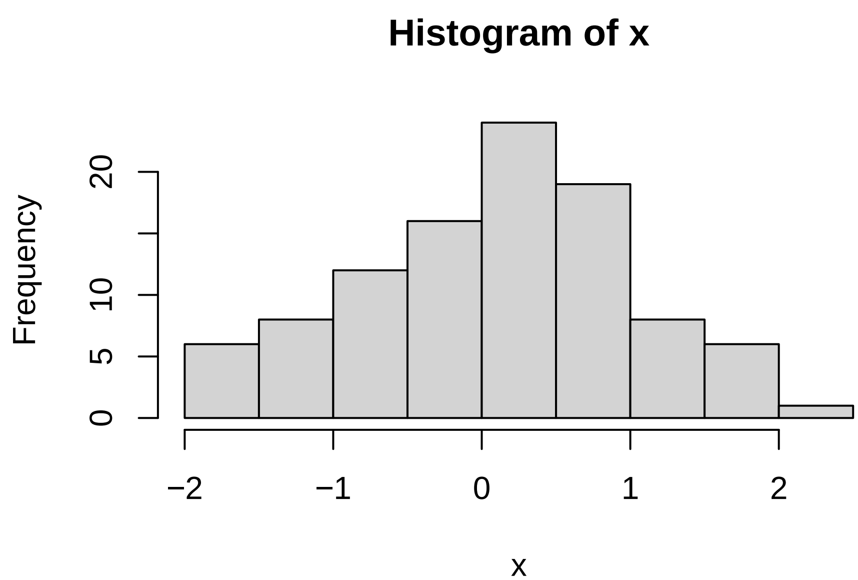
```r
# variance
var(vec1)
```

```
[1] 979.7667
```

```
# quantile
quantile(vec1,0.5)
```

```
50%
 33
```

```
# 100 Standard normal random numbers
x=rnorm(100,mean=0,sd=1)
# histogram
hist(x)
```



- Functions, conditional statements: *if*, *for* and *while*. A code example:

```
# create your own function
  myfunction=function(){
    return(print("Hello there!"))
  }
# if statement
lucky.number=100
if(lucky.number<=54){
print("You win!")
```

```
  }else{
  print("You lost!")
}
```

```
[1] "You lost!"
```

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignment - R basics

Instructions are posted on Canvas.

R Tidyverse

At the end of this week, you will be able to:

- Use R packages especially Tidyverse
- Identify `Tidy` data
- Practice with pipe operator `%>%`, `select()`, `filter()`,... for data wrangling
- Visualization using `ggplot2` package.

All Cheat Sheets are very useful!

Tidy data

A data is said to be tidy (Wickham 2014) format if each column represents a variable and each row represents an observation. Example of data that is *NOT* `tidy` is the `relig_income` data set in `tidyr` package:

```
# load a libraries
library(knitr) # fancy table
library(tidyverse) # load library tidyverse
# To display fancy tables
kable(head(relig_income,10))
```

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
Agnostic	27	34	60	81	76	137	122	109	84	96
Atheist	12	27	37	52	35	70	73	59	74	76
Buddhist	27	21	30	34	33	58	62	39	53	54
Catholic	418	617	732	670	638	1116	949	792	633	1489
Don't know/refused	15	14	15	11	10	35	21	17	18	116
Evangelical Prot	575	869	1064	982	881	1486	949	723	414	1529
Hindu	1	9	7	9	11	34	47	48	54	37

religion	<$10k	$10-20k	$20-30k	$30-40k	$40-50k	$50-75k	$75-100k	$100-150k	>150k	Don't know/refused
Historically Black Prot	228	244	236	238	197	223	131	81	78	339
Jehovah's Witness	20	27	24	24	21	30	15	11	6	37
Jewish	19	19	25	25	30	95	69	87	151	162

It is obvious that each column does not represent a variable. Variable `salary` could be a better fit to the values we have in the columns headings (<$10k, etc.). Another variable can be created to store values in the entry table (27, 34,...). These are the number of time we have a response - counts -. To make it `tidy` we need then to *pivot* the values columns into a two-column key-value pair. Let's name the values in the header `income` and values in the table `counts.` To do that we can run the following code:

```
# pivot a table/data frame
pivot_longer(relig_income,-religion,names_to='income',values_to = "count") -> tidydata
# To display fancy tables
kable(head(tidydata,n = 12))
```

religion	income	count
Agnostic	<$10k	27
Agnostic	$10-20k	34
Agnostic	$20-30k	60
Agnostic	$30-40k	81
Agnostic	$40-50k	76
Agnostic	$50-75k	137
Agnostic	$75-100k	122
Agnostic	$100-150k	109
Agnostic	>150k	84
Agnostic	Don't know/refused	96
Atheist	<$10k	12
Atheist	$10-20k	27

Manipulating data

`dplyr` package is designed to perform some of the widely used operations when working with `data.frame` or `tibble`. - The [`dplyr` Cheet Sheet](). When manipulating data, you may want to:

- Subset the data to contain only row (observations) you are interested in
- Subset the data to contain only columns (variables) you are interested in
- Create new variables and add them to the data
- aggregate the data

To achieve these operations and more, the package `dplyr`offers the following functions:

Function	Action
filter()	subset rows
select()	subset variables
mutate()	create a new variable
arrange()	sort
summarize()	aggregate the data
———-	———

Here is an example:

```
# pivot a table/data frame
pivot_longer(relig_income,-religion,names_to='income',values_to = "count") -> tidydata
# Select data where income is < $10k
kable(head(filter(tidydata,income=="<$10k")))
```

religion	income	count
Agnostic	<$10k	27
Atheist	<$10k	12
Buddhist	<$10k	27
Catholic	<$10k	418
Don't know/refused	<$10k	15
Evangelical Prot	<$10k	575

```
# Select data where income is < $10k
kable(head(arrange(tidydata,desc(count))))
```

religion	income	count
Evangelical Prot	Don't know/refused	1529
Catholic	Don't know/refused	1489
Evangelical Prot	$50-75k	1486
Mainline Prot	Don't know/refused	1328

religion	income	count
Catholic	$50-75k	1116
Mainline Prot	$50-75k	1107

Pipe operator %>%

The pipe operator `%>%` allows us to perform a series of functions without storing the outcomes of each function. For example:

```
library(dplyr)
sqrt(log(25))
```

[1] 1.794123

```
#is the same as
25 %>%
  log %>%
  sqrt
```

[1] 1.794123

We often start with our data and then apply functions sequentially. The benefit of the pipe operator is more evident when dealing with complex operations.

Summarizing data

One of the tasks in statistics is to summarize data. Let's look into this example using data `chickwts` about Chicken weights and diet. It has two variables `weight` and `feed`:

```
# See what is in the data
str(chickwts)
```

```
'data.frame':   71 obs. of  2 variables:
 $ weight: num  179 160 136 227 217 168 108 124 143 140 ...
 $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2 2 2 2 2 2 2 2 2 ...
```

```
# Mean and standard deviation of the weight
chickwts %>%
```

```r
  summarise(mean.weight=mean(weight),s.weight=sd(weight))
```

```
  mean.weight s.weight
1    261.3099  78.0737
```

```r
# Mean and standard deviation of the weight by group
chickwts %>%
  group_by(feed) %>%
  summarise(mean.weight=mean(weight),s.weight=sd(weight), nbr.chick=n())
```

```
# A tibble: 6 x 4
  feed      mean.weight s.weight nbr.chick
  <fct>           <dbl>    <dbl>     <int>
1 casein           324.     64.4        12
2 horsebean        160.     38.6        10
3 linseed          219.     52.2        12
4 meatmeal         277.     64.9        11
5 soybean          246.     54.1        14
6 sunflower        329.     48.8        12
```

```r
# Select groups `casein`, `linseed`, and `soybean`
chickwts %>%
  filter(feed %in% c("casein","linseed","soybean")) %>%
  group_by(feed) %>%
  summarise(mean.weight=mean(weight),s.weight=sd(weight), nbr.chick=n())
```

```
# A tibble: 3 x 4
  feed    mean.weight s.weight nbr.chick
  <fct>         <dbl>    <dbl>     <int>
1 casein         324.     64.4        12
2 linseed        219.     52.2        12
3 soybean        246.     54.1        14
```

Data visualization using `ggplot2`

`ggplot2` package is dedicated to data visualization. It can greatly improve the quality and aesthetics of your graphics, and will make you much more efficient in creating them. `gg` stands for *grammar of graphics*.

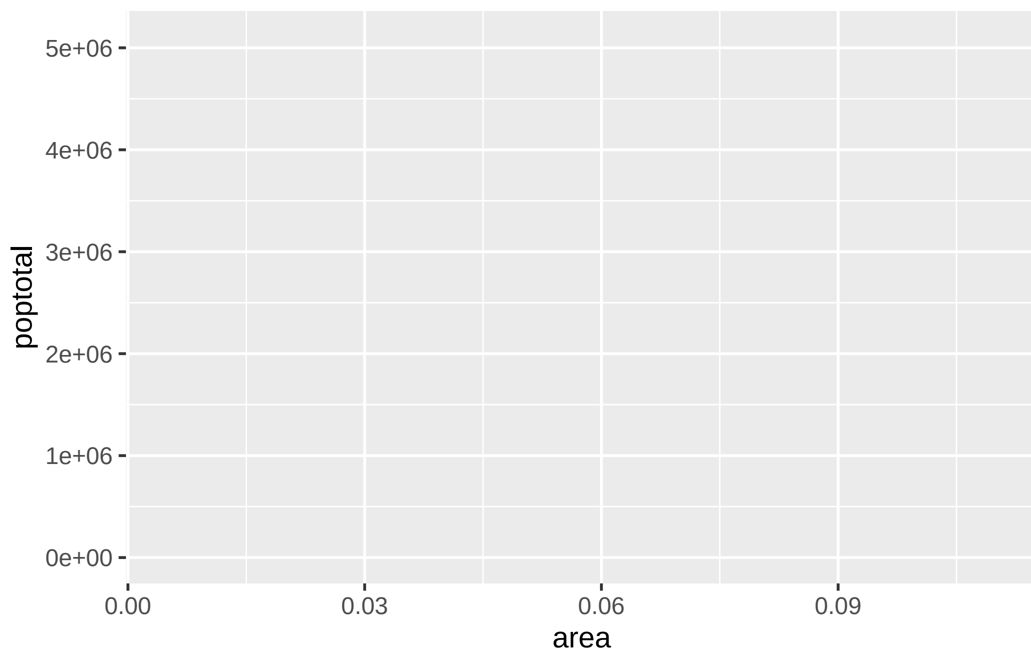
This link The R Graph Gallery provides a gallery of graphs created using R. A good place to get inspired and learn some advanced visualizations.

Let consider the following example:

```
#Demographic information of midwest counties from 2000 US census
kable(head(midwest))
```

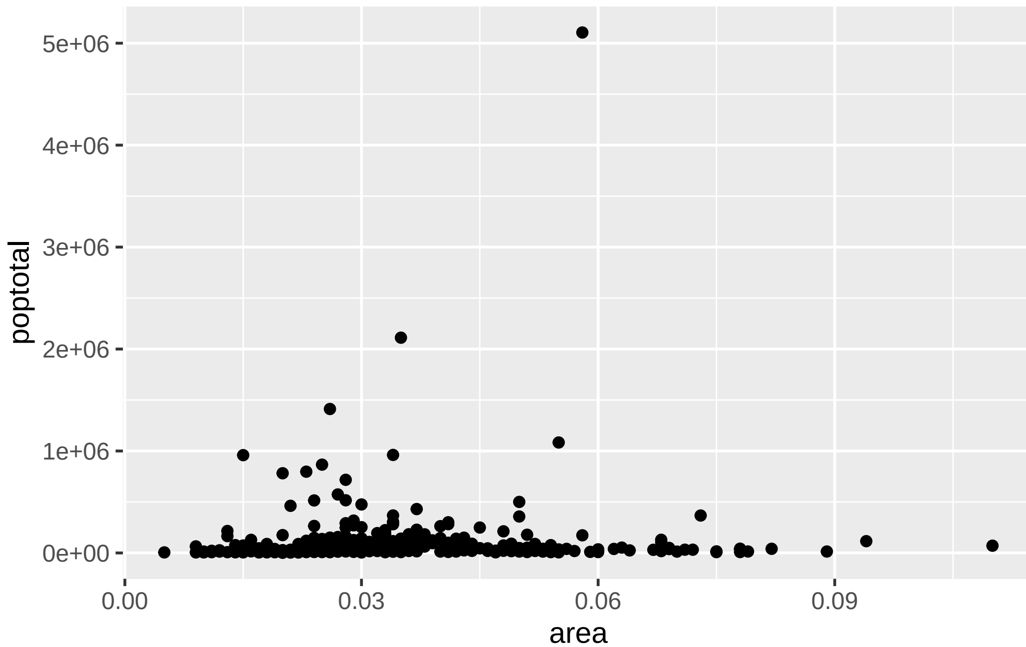
PID	county	state	[illegible]	poverty
561	ADAMS	IL	[illegible]	AAR
562	ALEXANDER	IL	[illegible]	LHR
563	BOND	IL	[illegible]	AAR
564	BOONE	IL	[illegible]	ALU
565	BROWN	IL	[illegible]	AAR
566	BUREAU	IL	[illegible]	AAR

```
# Get started - `area` and `poptotal` are variable in `midwest`
ggplot(midwest,aes(x=area,y=poptotal))
```

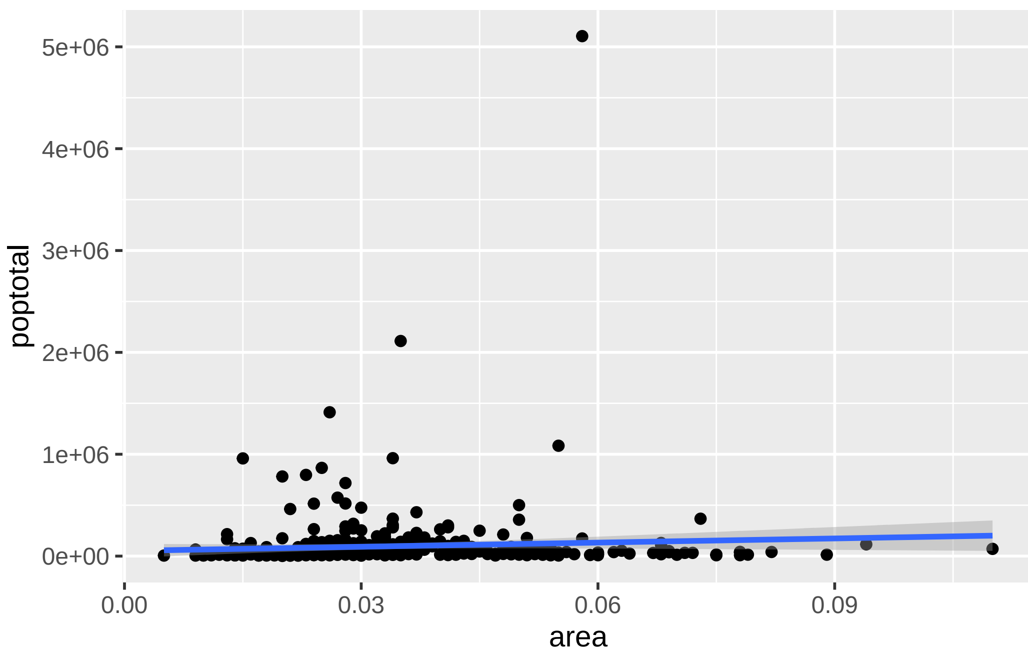


What we see here is a blank ggplot! ggplot does not plot by default a scatter or a line chart! We would need to decide next what should we plot! Let's make a scatter plot.

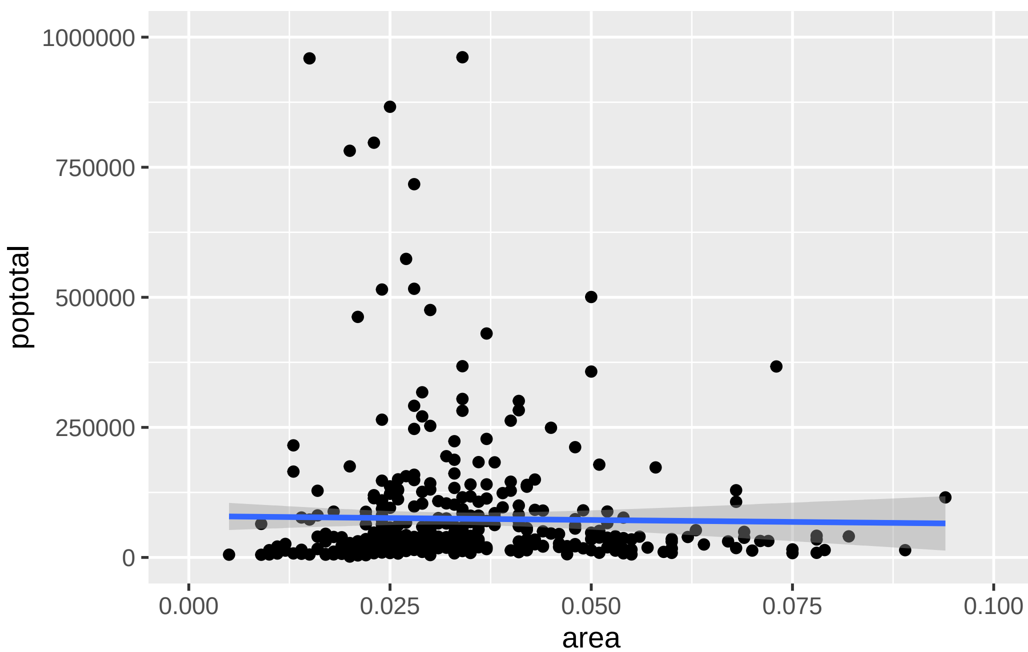
```
ggplot(midwest,aes(x=area,y=poptotal)) +
  geom_point()
```



Yaay! we did it. Next, let's add a linear regression model: $poptotal = \beta_0 + \beta_1 area$.

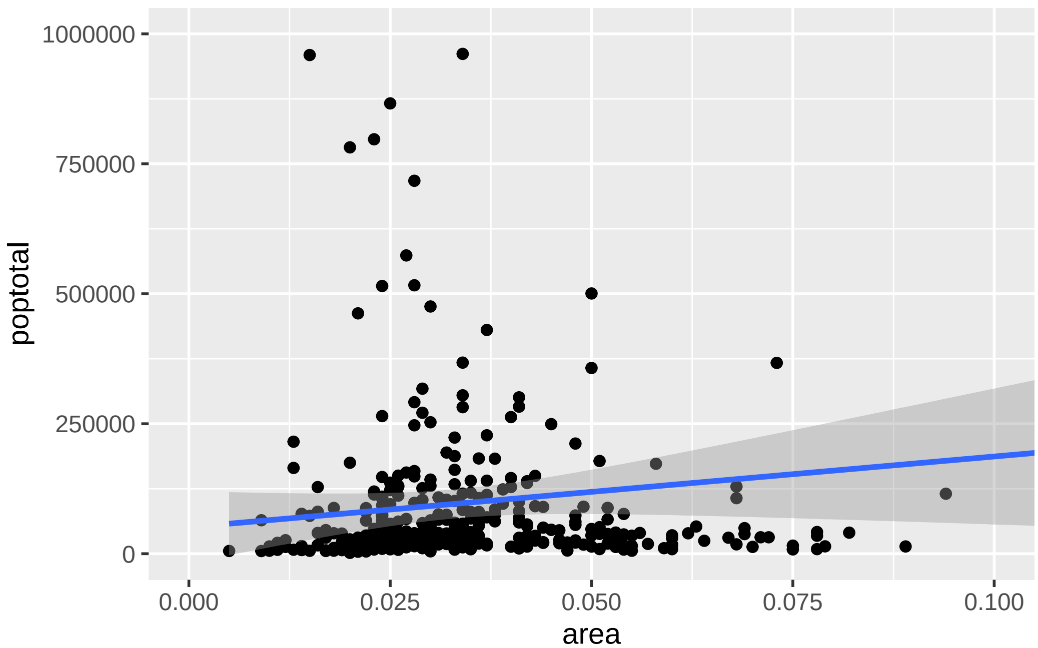


To control x and y axis limits, we can use `xlim()` and `ylim()` as follows:

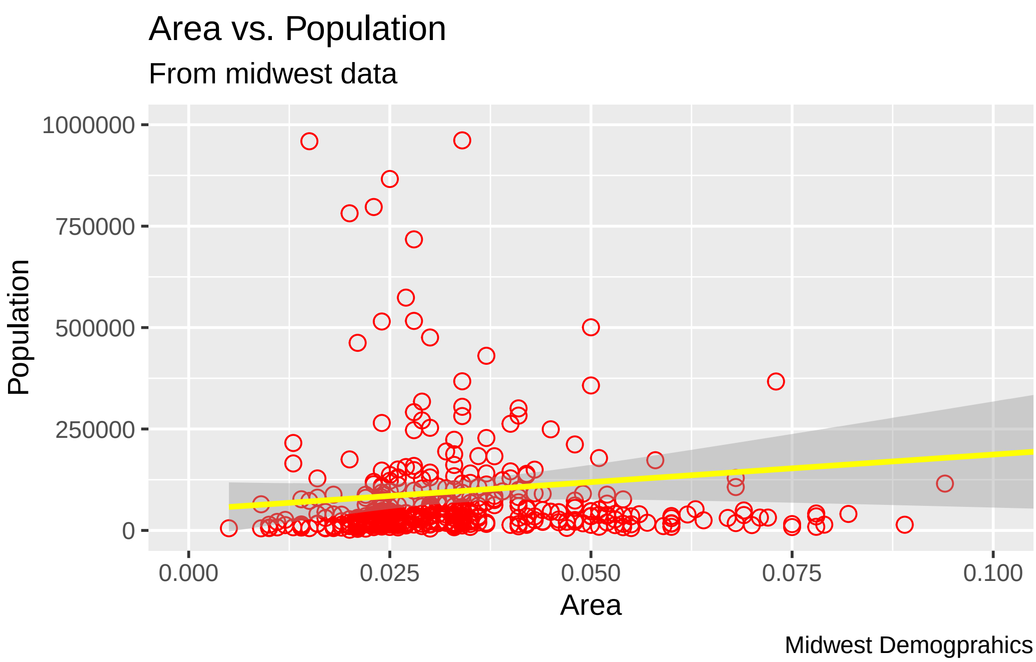


Notice that the line we obtain here is different from the line from the first fit (all data included). This happens because ggplot will refit the model `lm()` to data without the observations that are outside the ranges. This is useful if we want to examine changes in the model line when extreme values (or outliers) are removed.

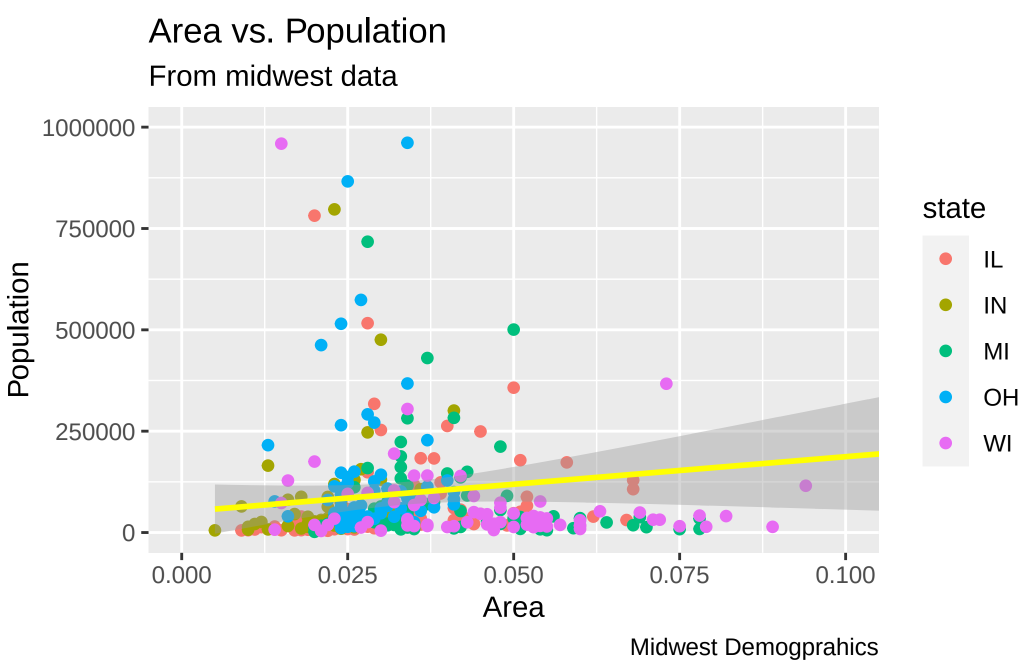
We can also keep the model as the original plot and zoom in using:



Let's Add some fancy options:



Wow! What about adding a new variable to the plot! For example, adding `state` variable. Let's change the color to match the `state` where a data point belongs to; `state` is a variable in the `midwest` dataset.



And more...

Lessons of this week provide more about `tidyverse`. The following will be covered more in details:

- Data manipulation (`filter`, `select`, `mutate`, `arrange`, `summarize`, and etc.)
- `ggplot2` package for data visualization.
- An extended example

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignment - R Tidyverse

Instructions are posted on Canvas.

SQL Basics

At the end of this week, you will be able to:

- Identify *Structured Query Language* queries
- Write your first SQL queries

Let's start with defining the basics.

Database

A *database* is an organized collection of data stored and accessed electronically from a computer system. A Database Management System (DBMS) is a software that is used to manage databases.

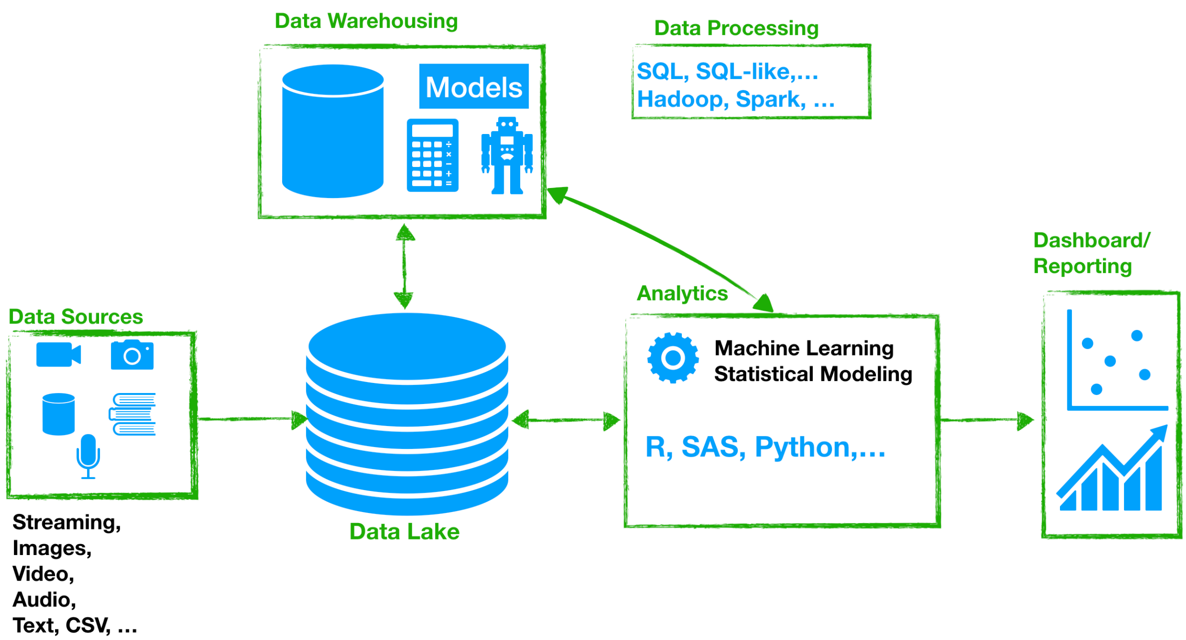


Figure 2: Data Science

In order to work with data that are stored in databases we need a language. *SQL* is a standard computer language for relational database management systems (RDBMS). It is used for storing, manipulating and retrieving data in databases.

SQL has various dialects such as PL/SQL (Oracle), T-SQL (Microsoft), and others.

In this course, we will use **SQL Server Management Studio** hosted at UWF servers. We will use the fictional company Adventure Works data.

Information about accessing the SQL Server is posted on Canvas.

Basic concepts

When dealing with databases we will need to know what is:

- **Entity**: is any thing the data represents in a database. For example, `Students`, `Employees`, `Schools`, `Departments`, etc. There are given as tables.
- **Data Type**: We need to pick a data type for each column when creating a table. There are common data types including `INTEGER`, `FLOAT`, `CURRENCY`, `DATE`, `BOOLEAN`, and etc.
- **Data Definition Language (DDL)**: DDL commands are used to create or modify database structures. `CREATE`, `ALTER`, and `DROP` are examples of DDL commands.
- **Data Manipulation Language (DML)**: DML commands are used to insert, retrieve, or modify data. `INSERT`, `DELETE`, and `UPDATE` are examples of DML commands.
- **Data Control Language (DCL)**: DCL commands are used to create rights and permission. `GRANT` and `REVOKE` are examples of DCL commands.
- **Query**: Data scientists use a query to get data or information from database tables.

Data Language

Now that we have access to SQL server system, we are ready to manipulate some data and execute *SQL* queries. *SQL* statements are divided into **3 categories**: DDL, DML, and DCL. We can execute *SQL* queries using *SQL Command* or using Graphic User Interface (*GUI*). We shall present next common statements for DDL and DML.

Data Definition Language (DDL)

The DDL statements are used to create databases and tables. Here is a list of some of the statements:

- SQL commands to create a *database*:

`CREATE DATABASEdatabase_name;`

- SQL commands to delete a *database*:

`DROP DATABASEdatabase_name;`

be very careful to drop databases or tables!

- SQL commands to create a *Table*:

`CREATE TABLEtable_name;`

- SQL commands to create a *Table* from an existing table:

`SELECT... INTOtable_name FROM Orginal_table`

- SQL commands to drop a *Table*:

`DROP TABLEtable_name;`

- SQL commands to truncating (remove all records from a table) a *Table*:

`TRUNCATE TABLEtable_name;`

Data Maniplulation Language (DML)

The DDL statements are used to insert data, update records, and delete records. Data Manipulation Language is used to manipulate data. Here is a list of the main statements:

- SQL commands to insert one or more records into a *Table*:

`INSERT INTOtable_name(col1,col2,...) VALUES(exp1,exp2,...);`

`INSERT INTOtable_name VALUES(exp1,exp2,...);`

Make sure you insert data in the same order as that in the table for the second syntax.

- SQL commands to select records from one or more *Tables*:

`SELECTcolumn(s) FROMtables WHEREconditions` (optional) `ORDER BY column(s)ASC | DESC;` (optional)

- `DISTINCT` clause to eliminate duplicates:

`SELECT DISTINCTcolumn_name FROMtable_name;`

- `WHERE` clause to filter if the condition is true:

`SELECTcolumn(s) FROMtable_name WHEREconditions;`

- Arithmetic operators

`SELECTcolumn_name1, column_name2, column_name2*2 AS 'twicecolumn2' FROMtable_name;`

Basic **arithmetic** operators include: `%modulo`, `/division`, `*multiplication`, `+addition`, and `-substraction`.

Basic **comparison** operators include: `=equal to`, `<>not equal to`, `>greater than`, `>=greater than equal to`, and more.

Basic **condition** operators include: `ANDall conditions must be true to get true`, `ORAny one of the conditions must be true to get true`, `INtest if an expression matches any value in a list of VALUES`, `BETWEENcheck if an experession is within a range of VALUES`, and more.

- `ORDER BY` clause to sort the records:

`SELECTcolumn(s) FROMtable_name WHEREconditions ORDER BYexpression (by default ASC);`

- `UPDATE` statement to update records:

`UPDATEtable SETcol1 = value1, col2 = value2, ... WHEREconditions [optional];`

- `DELETE` statement to delete records:

`DELETE FROMtable WHEREconditions [optional];`

Functions and GROUP BY

Often you will be asked to answer questions that involve writing queries for summaries using aggregate function and `GROUP BY` clause.

- **SQL commands for Aggregate statements**:

`SELECT Aggregate Functioncolumn_name FROMtable_name;`

Below are the main aggregate functions:

Function	Action
`AVG()`	average values
`COUNT()`	count the number of rows in a table
`MAX()`	select the highest value select the latest date select the last record for a character
`MIN()`	select the lowest value select the earliest date select the first record for a character
`SUM()`	return the total for a numeric column
`ROUND()`	round a number to specific decimal

In addition to aggregate functions, there are other type of functions:

- *The **number** functions* take a numeric as an input and return a numeric value. The common number functions include `CEILING()`, `FLOOR()`, `%`, `POWER(m,n)` $[m^n]$, `SQRT()`, and `ROUND()`.

- *The **string** functions.* The common string functions include `CONCAT()`, `LEFT()`, `LEN()`, `LOWER()`, `REPLACE()`, `RIGHT()`, `UPPER()`, and `SUBSTRING()`.

- *The **Date and Time** functions.* The common date and time functions include `CURRENT_TIMESTAMP()`, `DATEADD()`, `DATEPART()`, `GETDATE()`, `DATEDIFF()`, and `SYSDATETIME()`.

- *The **Conversion** functions.* The common conversion functions include `CAST()` and `CONVERT()`.

- **GROUP BY and HAVING Clause**:

The GROUP BY statement is used to group data from a column. HAVING clause is used with a GROUP BY to add conditions on groups.

```
SELECT Aggregate Functioncolumn_name FROMtable_name WHEREconditions - optional
GROUP BYcolumn_name HAVINGconditions - optional ORDER BYcolumn(s) [ASC | DESC]
- optional;
```

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignments - SQL basics

Instructions are posted on Canvas.

Advanced SQL

At the end of this week, you will be able to:

- Practice with advanced *SQL*
- Evaluate `UNION`, `Subqueries`, `EXCEPT`, and other *SQL* commands.
- Apply `JOINS`

Advanced SQL commands

SQL commands to return all rows from two tables:

```
SELECT column(s) FROM table1 UNION ALL SELECT column(s) FROM table2;
```

SQL commands to return only rows that exist in both tables:

```
SELECT column(s) FROM table1 INTERSECT SELECT column(s) FROM table2;
```

SQL commands to return all rows in the first SELECT but excludes those by the second SELECT:

```
SELECT col1,col2,... FROM table1 EXCEPT SELECT col1,col2,... FROM table2;
```

SQL command to specify the number of records to return:

```
SELECTTOPnumber | percent column_names(s) <br/>FROM table_name;‘
```

Subqueries

A *Subquery* is a SQL query nested inside a SQL query. Very useful to create a virtual table usable by the main query.

```
SELECT column(s) FROM table1 WHERE value IN (SELECT column_name FROM tables2 WHERE conditions);
```

Joins

Relational databases are defined with tables or entities such *Employee* and *Department*. To create a link between the two tables a column is defined as *Department_ID* in both tables. Now, if you would like to extract employee names and departments names you need to SQL JOIN.

There are four main type of joins:

JOIN	Action
`INNER JOIN`	return records that have matching values in both tables
`LEFT JOIN`	return all records from table1 (**LEFT** table1) and the matched records from table2
`RIGHT JOIN`	return all records from table2 (**RIGHT** table1) and the matched records from table1.
`FULL JOIN()`	return all rows from both tables

Syntax:

```
SELECT table1.col_name, table2.col_name FROM table1 INNER JOIN table2ON
table1.col_name = table2.col_name;
```

OR

```
SELECT table1.col_name, table2.col_name FROM table1 LEFT JOIN table2ON table1.col_name
= table2.col_name;
```

OR

```
SELECT table1.col_name, table2.col_name FROM table1 RIGHT JOIN table2ON
table1.col_name = table2.col_name;
```

OR

```
SELECT table1.col_name, table2.col_name FROM table1 FULL JOIN table2ON table1.col_name
= table2.col_name;
```

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignments - Advanced SQL

Instructions are posted on Canvas.

Python Basics - NumPy and Pandas

At the end of this week, you will be able to:

- Practice with *Python* Basics
- Practice using **NumPy** and **Pandas** libraries
- Write your first *Python* script!

References

- *Python Data Science Handbook* (VanderPlas 2016). Free access
- *Think Python* (Downey 2015). Free access
- *Data Science and Analytics with Python* (Rogel-Salazar 2018)

Introduction to Python

Python has emerged over the last recent years as one of the most used tools for data science projects. It is known for code readability and interactive features. Similar to **R**, **Python** is supported by a large number of packages that extend its features and functions. Common packages are, to name few:

- NumPy: provides functions for manipulating arrays
- Pandas: provides functions for manipulating data frames
- Matplotlib: provides functions for visualizations and plotting
- Statsmodels: provides functions for statistical models
- Scikit-learn: provides functions for machine learning algorithms

Getting started with Python

We will use RStudio IDE to run Python but, there are other IDEs that you may want to check for your information such as Pycharm, Jupyter, and others. We will be using `Python 3`. We will see that there are multiple similarities between R and Python.

Indentation refers to the spaces at the beginning of a code line. The indentation in Python is very important.

Recordings of this week provide lessons about the following concepts:

Python Basics

- Python Variables:

```
# This is Python Code
print("Hello World!")
```

```
Hello World!
```

You can name a variable following these rules:

- One word
- Use only letters, numbers, and the underscore (_) character
- Can't begin with a number
- Python is case-sensitive

```
x = "HeyHey"
y = 40
x
y
x, y = "Hey", 45 # Assign values to multiple variables
print(x)
print(y)
ranks = ["first","second","third"] # list
x, y, z = ranks
print(ranks)
x
y
z

def myf():
  x="Hello"
  print(x)

myf()
```

```python
def myf():
  global x # x to be global - outside the function
  x="Hello"
  print(x)

myf()
```

```
Hey
45
['first', 'second', 'third']
Hello
Hello
```

Data Types:

```python
x = str(3)    # x will be '3'
x = int(3)    # x will be 3
x = float(3)  # x is a float - 3.0
x = 1j       # x is complex
x = range(5,45)    # x is a range type
x = [1,2,1,24,54,45,2,1]  # x is a list
x = (1,2,1,24,54,45,2,1)  # x is a tuple
x = {"name" : "Ach", "age" : 85}  # x is a dict (mapping)
```

Math operations:

```python
5+4   # Addition
5*4   # Multiplication
5**4  # power / exponent
print("Hey"*3) # String operations
import math as mt # More more math functions using package *math*
mt.cos(556) # cosine function
import random # generate random numbers
print(random.randrange(1, 10))
import numpy as np # generate random numbers
print(np.random.normal(loc=0,scale=1,size=2))
```

```
HeyHeyHey
2
[ 1.57610916 -0.33948029]
```

Strings operations:

```
word = "Hello There!"
word[1] # accessing characters in a String
for z in word:
  print(z)

len(word) # strings length

"or" in word # check if "or" is in word
word1 = "Do you use Python or R or both!"
"or" in word1 # check if "or" is in word1
```

```
H
e
l
l
o
 
T
h
e
r
e
!
```

```
True
```

Python assignment operators:

Operator	Example	Results
=	x = 10	x = 10
+=	x += 10	x = x+10
-=	x -= 10	x = x-10
*=	x *= 10	x = x*10
/=	x /= 10	x = x/10
%=	x %= 10	x = x%10
**=	x **= 10	x = x**10

If-Else Statements:

```
h = 2
if h > 2:
 print("Yes!") # indentation very important other ERROR
elif h > 50:
 print("Yes Yes!")
else:
  print("No")
```

```
No
```

For Loop Statements:

```
for k in range(1,10):
  print(str(k)) # does not show up 10; goes up to 9
```

```
1
2
3
4
5
6
7
8
9
```

Python Numpy

`NumPy` is a Python library. It stands for Numerical Python and very useful for manipulating arrays. It is faster than using Lists and quite useful for machine learning applications.

```
import numpy # this code import NumPy library
arr1 =  numpy.array([1,2,45,564,98]) # create array using NumPy
print(arr1)
```

```
[  1   2  45 564  98]
```

Usually, we give a Library an **alias** such as `np` for the NumPy library. Array objects in NumPy are called `ndarray`. We can pass any array (list, tuple, etc.) to the function `array()`:

```python
import numpy as np
arr1 = np.array([1,2,45,564,98])
print(arr1)

# Multidimensional arrays!
d0 = np.array(56)
d1 = np.array([15, 52, 83, 84, 55])
d2 = np.array([[1, 2, 3], [4, 5, 6]])
d3 = np.array([[[1, 2, 3], [4, 5, 6]], [[11, 21, 31], [41, 51, 61]]])

print(d0.ndim) # print dimension
print(d1.ndim)
print(d2.ndim)
print(d3.ndim)
```

```
[  1   2  45 564  98]
0
1
2
3
```

Array Indexing:

```python
import numpy as np

D2 = np.array([[1,2,3,4,5], [6,7,8,9,10]], dtype=float)

print('4th element on 1st dim: ', D2[0, 3])
print('4th element on 2nd dim: ', D2[1, 3])
print('1st dim: ', D2[0, :])

arr = np.array([1, 2, 3, 4, 5, 6, 7])

print("From the start to index 2 (not included): ", arr[:2])
print("From the index 2 (included) to the end: ", arr[2:])
```

```
4th element on 1st dim:  4.0
4th element on 2nd dim:  9.0
1st dim:  [1. 2. 3. 4. 5.]
From the start to index 2 (not included):  [1 2]
From the index 2 (included) to the end:  [3 4 5 6 7]
```

Arithmetic operations and Math/Stat functions:

```
import numpy as np

a = np.array([[1,2,3,4,5], [6,7,8,9,10]], dtype="f")
b = np.array([[10,20,30,40,50], [60,70,80,90,100]], dtype="i")

np.subtract(b,a) # b-a
np.add(b,a) # b+a
np.divide(b,a) # b/a
np.multiply(b,a) # b*a
np.exp(a) # exponential function
np.log(a) # natural logarithm function
np.sqrt(a) # square root function
np.full((3,3),5) # 3x3 constant array
a.mean() # mean
a.std() # standard deviation
a.var() # variance
a.mean(axis=0) # mean across axis 0 (rows)
np.median(a) # median
np.median(a,axis=0) # median
```

```
array([3.5, 4.5, 5.5, 6.5, 7.5], dtype=float32)
```

Random numbers generation:

Random is a module in `NumPy` to offer functions to work with random numbers.

```
from numpy import random

x = random.randint(100) # a random integer from 0 to 100
print(x)

x = random.rand(10) # 10 random numbers float from 0 to 1
print(x)

x = random.randint(100,size=(10)) # 10 random integers from 0 to 100
print(x)

x = random.randint(100,size=(10,10)) # 10x10 random integers from 0 to 100
print(x)
```

```python
x = random.choice([100,12,0,45]) # sample one value from an array
print(x)

x = random.choice([100,12,0,45],size=(10)) # sample one value from an array
print(x)

x = random.choice([100, 12, 0, 45], p=[0.1, 0.3, 0.6, 0.0], size=(10)) # Probability sampl
print(x)

x = random.normal(loc=1, scale=0.5, size=(10)) # Normal distribution
print(x)

x = random.normal(loc=1, scale=0.5, size=(10)) # Normal distribution
print(x)
```

```
15
[0.19713905 0.23444829 0.991311   0.22672133 0.58365635 0.80819133
 0.42944713 0.04866871 0.03334703 0.92529126]
[93 27 64 34  9 68 27 45 59 66]
[[89 66 96 99 91 37 82 97 33 80]
 [ 6  7 83 87 29 74 70 84 74 91]
 [57 97 85 94 82 12 88  0  9 44]
 [79 48 61 76 22 96  5 93  3 94]
 [39 22  4 33 13 16 52 14  0 29]
 [28 44 11 42 86 92 16 75 43  5]
 [87 13 59 55 24 70 95  3  1 77]
 [89 81 36 77 57 78 77 58 46 42]
 [15 21 76 45 42 76 42 61 43 87]
 [68 14 41 19 26 45 25 33 73 70]]
100
[100  12 100  12 100  45  12   0 100  45]
[  0   0 100 100   0   0   0   0   0   0]
[1.05120275 0.18006357 0.7898641  1.24505707 1.34247441 0.43188103
 1.27766879 1.14041786 0.51785556 0.61957845]
[0.22007981 0.53797539 0.69221536 1.33064321 0.64695599 1.80440136
 1.03245788 0.94828179 1.22113946 0.36958332]
```

For more reading visit Introduction to NumPy.

Python Pandas

`Pandas` is a Python library. It is useful for data wrangling and working with data sets. `Pandas` refers to both *Panel Data* and *Python Data Analysis*. This is a handy Cheat Sheet for Pandas for data wrangling.

```python
import pandas as pd

a = [1,6,8]
series = pd.Series(a) # this is a panda series
print(series)

mydata = {
  "calories": [1000, 690, 190],
  "duration": [50, 40, 20]
}
mydataframe = pd.DataFrame(mydata) # data frame
mydataframe
```

```
0    1
1    6
2    8
dtype: int64
```

	calories	duration
0	1000	50
1	690	40
2	190	20

Read CSV Files

CSV files are a simple way to store large data sets. Data Frame Pandas can read CSV files easily:

```python
import pandas as pd
import numpy as np

df = pd.read_csv("../datasets/mycars.csv")
print(df.info()) # Info about Data

df.head()
```

```python
df.loc[3,"speed"] = np.NaN # insert NaN in the row 10 in speed column
df.head()

newdf = df.dropna() # drop NA cells
newdf.head()

df.dropna(inplace = True) # drop NA cells and replace "df" with the new data
df.head()

df = pd.read_csv("../datasets/mycars.csv")
df.fillna(100, inplace = True) # replace NA values with 100.

df["speed"].fillna(10, inplace = True) # replace NA values with 10 only in column "speed"

x = df["speed"].mean() # find mean of speed
df["speed"].fillna(x, inplace = True) # replace NA values with mean only in column "speed"


print(df.duplicated().head()) # show duplicates
df.drop_duplicates().head() # drop duplicates
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 50 entries, 0 to 49
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Unnamed: 0  50 non-null     int64
 1   speed       50 non-null     int64
 2   dist        50 non-null     int64
dtypes: int64(3)
memory usage: 1.3 KB
None
0    False
1    False
2    False
3    False
4    False
dtype: bool
```

	Unnamed: 0	speed	dist
0	1	4	2
1	2	4	10
2	3	7	4
3	4	7	22
4	5	8	16

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignments - Python Basics

Instructions are posted on Canvas.

More Python (Stat/ML/Viz)

We continue our journey with *Python*. At the end of this week, you will be able to:

- Practice using **statsmodels** library for statistical analysis
- Exercise using **Scikit-learn** library for machine learning
- Create plots using **Matplotlib**

Statistical Models in Python

statsmodels is a Python package that provides functions for fitting statistical models, conducting statistical tests, and statistical data exploration.

Let's read a data set from the list provided in this link. We use the `mtcars` data set in R package `datasets`.

```
import statsmodels.api as stat # allow to access easily to most of the functions
import statsmodels.formula.api as statf # allow to use formula style to fit the models
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt

df = stat.datasets.get_rdataset("mtcars", "datasets").data # load data "mtcars" from the R
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 32 entries, Mazda RX4 to Volvo 142E
Data columns (total 11 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   mpg     32 non-null     float64
 1   cyl     32 non-null     int64
 2   disp    32 non-null     float64
 3   hp      32 non-null     int64
 4   drat    32 non-null     float64
```

```
 5   wt      32 non-null    float64
 6   qsec    32 non-null    float64
 7   vs      32 non-null    int64
 8   am      32 non-null    int64
 9   gear    32 non-null    int64
 10  carb    32 non-null    int64
dtypes: float64(5), int64(6)
memory usage: 3.0+ KB
None
```

```
fit_olsregression = statf.ols("mpg ~ wt + cyl",data=df).fit()
print(fit_olsregression.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.830
Model:                            OLS   Adj. R-squared:                  0.819
Method:                 Least Squares   F-statistic:                     70.91
Date:                Sun, 25 Jun 2023   Prob (F-statistic):           6.81e-12
Time:                        23:29:32   Log-Likelihood:                -74.005
No. Observations:                  32   AIC:                             154.0
Df Residuals:                      29   BIC:                             158.4
Df Model:                           2
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     39.6863      1.715     23.141      0.000      36.179      43.194
wt            -3.1910      0.757     -4.216      0.000      -4.739      -1.643
cyl           -1.5078      0.415     -3.636      0.001      -2.356      -0.660
==============================================================================
Omnibus:                        4.628   Durbin-Watson:                   1.671
Prob(Omnibus):                  0.099   Jarque-Bera (JB):                3.426
Skew:                           0.789   Prob(JB):                        0.180
Kurtosis:                       3.287   Cond. No.                         27.9
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

```python
pred_ols = fit_olsregression.get_prediction()
pred_ols.summary_frame().head()
```

```
                        mean   mean_se  ...  obs_ci_lower  obs_ci_upper
Mazda RX4          22.279145  0.601167  ...     16.885964     27.672326
Mazda RX4 Wag      21.465447  0.497629  ...     16.116566     26.814327
Datsun 710         26.252026  0.725244  ...     20.795395     31.708658
Hornet 4 Drive     20.380516  0.460267  ...     15.045648     25.715384
Hornet Sportabout  16.646958  0.775271  ...     11.161630     22.132285

[5 rows x 6 columns]
```

```python
df["cyl"] = df["cyl"].astype("category") # make cyl categorical variable
fit_olsregression = statf.ols("mpg ~ wt + cyl",data=df).fit() # refit with a categorical v
print(fit_olsregression.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                    mpg   R-squared:                       0.837
Model:                            OLS   Adj. R-squared:                  0.820
Method:                 Least Squares   F-statistic:                     48.08
Date:                Sun, 25 Jun 2023   Prob (F-statistic):           3.59e-11
Time:                        23:29:32   Log-Likelihood:                -73.311
No. Observations:                  32   AIC:                             154.6
Df Residuals:                      28   BIC:                             160.5
Df Model:                           3
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept     33.9908      1.888     18.006      0.000      30.124      37.858
cyl[T.6]      -4.2556      1.386     -3.070      0.005      -7.095      -1.416
cyl[T.8]      -6.0709      1.652     -3.674      0.001      -9.455      -2.686
wt            -3.2056      0.754     -4.252      0.000      -4.750      -1.661
==============================================================================
Omnibus:                        2.709   Durbin-Watson:                   1.806
Prob(Omnibus):                  0.258   Jarque-Bera (JB):                1.735
Skew:                           0.559   Prob(JB):                        0.420
Kurtosis:                       3.222   Cond. No.                         18.9
==============================================================================
```

```
Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

For more statistics with Python consult the following links: - Statistical tests - Generalized Linear Models - Contingency Tables

Machine Learning

The `scikit-learn` provides function that support machine learning techniques and practices including model fitting, predicting, cross-validation, etc. It also provides various supervised and unsupervised methods. The website of the package is https://scikit-learn.org

Linear models

Fitting regression models is relevant when the target value or response variable is assumed to be a linear combinations of some predictors. The following code will allow you to fit various linear models using sklearn module.

```
from sklearn import linear_model
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_absolute_percentage_error

df = stat.datasets.get_rdataset("mtcars", "datasets").data # load data "mtcars" from the R

# split data
training_data, testing_data = train_test_split(df, test_size=0.2, random_state=25)

# Create X and Y from training
Y = training_data["mpg"] # response variable / outcome
X = training_data.drop(columns=["mpg"]) #predictors / features
reg =  linear_model.LinearRegression().fit(X,Y)

# Create X and Y from testing
Y_test = testing_data["mpg"] # response variable / outcome
X_test = testing_data.drop(columns=["mpg"]) #predictors / features
mpg_y_pred = reg.predict(X_test) # predictions

print(reg.coef_)
```

```
[-0.52365917  0.01668511 -0.02157865  0.12362249 -3.46329267  0.70433502
  1.1100029   3.90616473  0.28676536 -0.16588934]
```

```
mean_absolute_percentage_error(y_true=Y_test,y_pred=mpg_y_pred)
```

```
0.12447286077371422
```

Visualization with Python

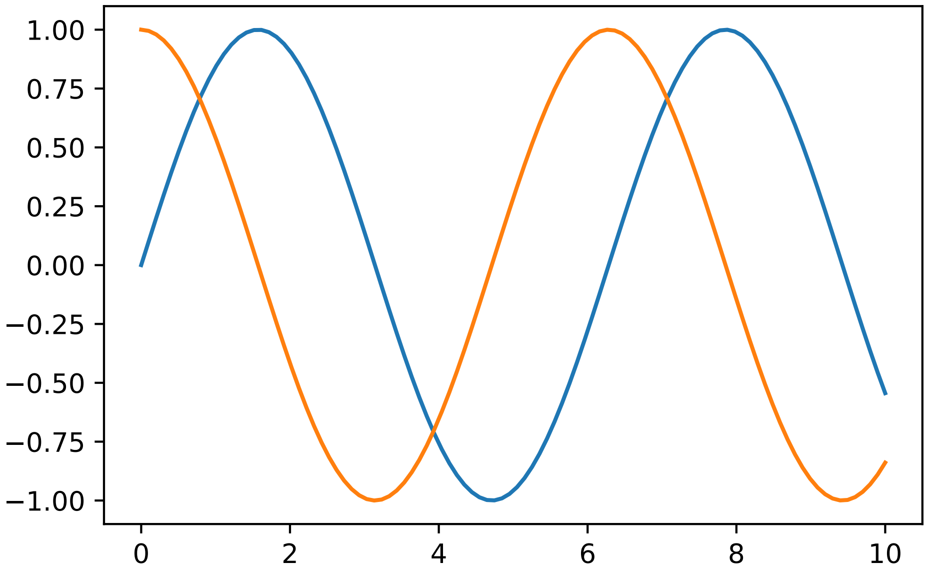
Python offers multiple tools and libraries that come with a lot of features for data vizualisation and plotting. Among the popular libraries we have:

- Matplotlib
- Seaborn
- Plotly

The `matplotlib.pyplot` module is a collection of command style functions that make matplotlib work like `MATLAB`.

A few plots!

```
import matplotlib.pyplot as plt
import numpy as np
import matplotlib
matplotlib.use('Agg') # To plot with Markdown

x = np.linspace(0, 10, 100)
plt.figure();
plt.plot(x, np.sin(x))
plt.plot(x, np.cos(x))
plt.show()
```




```
plt.close()
```

Read data from sklearn and vizualize

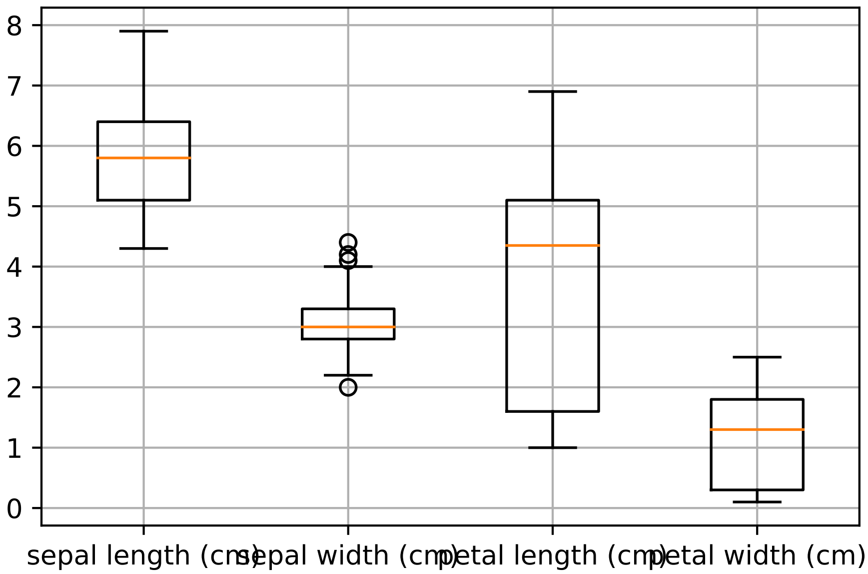
```
import matplotlib.pyplot as plt
import pandas as pd
from sklearn.datasets import load_iris
import matplotlib
matplotlib.use('Agg') # To plot with Markdown

iris = load_iris()
df_iris = pd.DataFrame(iris.data)
df_iris.columns = iris.feature_names

# Boxplot
plt.figure();
plt.boxplot(df_iris)
```

```
{'whiskers': [<matplotlib.lines.Line2D object at 0x141e57b70>, <matplotlib.lines.Line2D obje
```

```
plt.xticks([1, 2, 3, 4], iris.feature_names)
```

```
([<matplotlib.axis.XTick object at 0x141e300b8>, <matplotlib.axis.XTick object at 0x141ba3b38
```

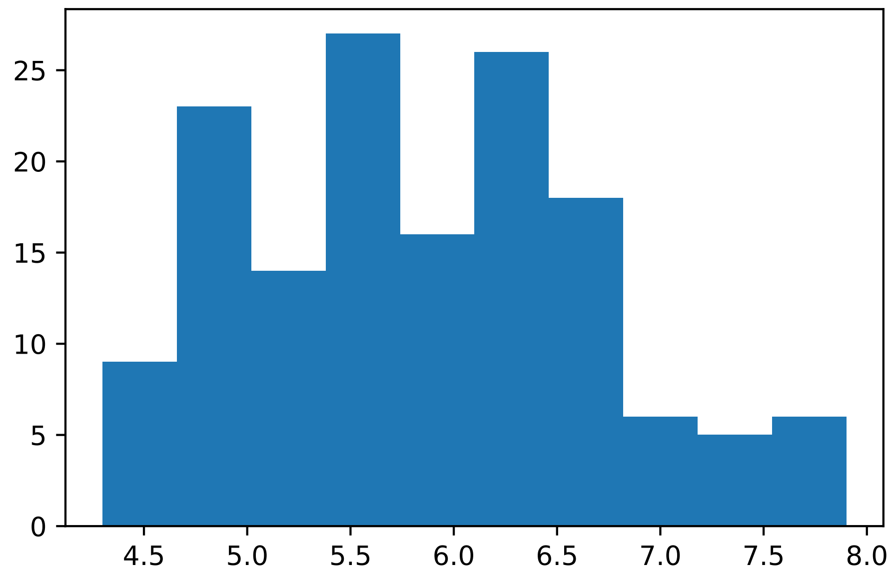
```
plt.grid()
plt.show()
```




```
plt.close()

#  histogram
plt.figure();
plt.hist(df_iris.iloc[:,0])
```

```
(array([ 9., 23., 14., 27., 16., 26., 18.,  6.,  5.,  6.]), array([4.3 , 4.66, 5.02, 5.38, 5
```

```
plt.show()
```




```
plt.close()
```

Recordings on Canvas will cover more details and examples! Have fun learning and coding ! Let me know how I can help!

Assignment - Python Stat/ML/Viz

Instructions are posted on Canvas.

Final Project

Final Exam Project

Final Project instructions are posted on Canvas.

References

Downey, Allen. 2015. *Think Python How to Think Like a Computer Scientist*. Green Tea Press Needham, Massachusetts.

Grolemund, Garrett. 2014. *Hands-on Programming with r: Write Your Own Functions and Simulations*. " O'Reilly Media, Inc.".

John David, Smith, Yang Sophie, Borasky M. Edward (Ed), Tyhurst Jim, Came Scott, Thygesen Mary Anne, and Frantz Ian. 2020. *Exploring Enterprise Databases with r: A Tidyverse Approach*. https://smithjd.github.io/sql-pet/.

Luraschi, Javier, Kevin Kuo, and Edgar Ruiz. 2019. *Mastering Spark with r: The Complete Guide to Large-Scale Analysis and Modeling*. O'Reilly Media.

Rogel-Salazar, Jesus. 2018. *Data Science and Analytics with Python*. CRC Press.

VanderPlas, Jake. 2016. *Python Data Science Handbook*. O'Reilly Media, Inc.

Wickham, Hadley. 2014. "Tidy Data." *The Journal of Statistical Software* 59. http://www.jstatsoft.org/v59/i10/.

Wickham, Hadley, and Garrett Grolemund. 2016. *R for Data Science: Import, Tidy, Transform, Visualize, and Model Data*. " O'Reilly Media, Inc.".

Zhang, Preston. 2017. *Practical Guide for Oracle SQL, t-SQL and MySQL*. CRC Press.