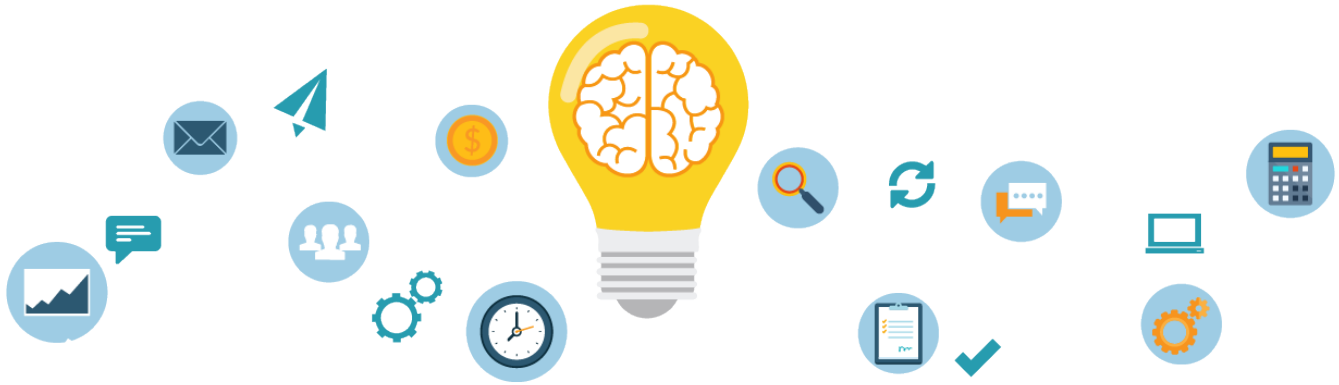


Python Programming

Day 1

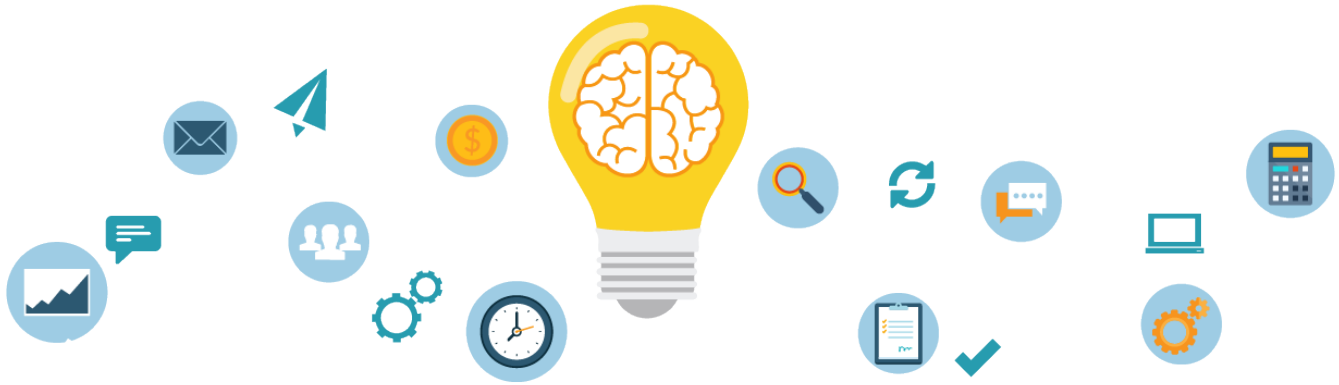
Learning Outcomes

A. To be able to gain the confidence to use Python as a programming language

B. To do basic operations in Numpy and Pandas
   1. Indexing, Slicing, and Querying
   2. Vector and matrix computations

C. To be able to do basic preprocessing using Python, Numpy and Pandas:
   1. Type conversion
   2. Missing values and standardization
   3. String cleaning
   4. Cleaning outliers

D. To be able to visualize the data with Matplotlib, Pandas and Seaborn



Jupyter Notebook Familiarization

Python Programming

Familiarization


Save and Checkpoint


Insert Cell Below


Cut, Copy, or Paste Cells


Move Selected Cell Up or Down


Run Cell, Interrupt Kernel, or Restart the Kernel

Shortcuts:

Command Mode (press `Esc` to enable)

- `F` : find and replace
- `Ctrl-Shift-F` : open the command palette
- `Ctrl-Shift-P` : open the command palette
- `Enter` : enter edit mode
- `P` : open the command palette
- `Shift-Enter` : run cell, select below
- `Ctrl-Enter` : run selected cells
- `Alt-Enter` : run cell, insert below
- `Y` : to code
- `M` : to markdown
- `R` : to raw
- `1` : to heading 1
- `2` : to heading 2
- `3` : to heading 3
- `4` : to heading 4
- `5` : to heading 5
- `6` : to heading 6
- `K` : select cell above
- `Up` : select cell above
- `Down` : select cell below
- `J` : select cell below
- `Shift-K` : extend selected cells above
- `Shift-Up` : extend selected cells above
- `Shift-Down` : extend selected cells below
- `Shift-J` : extend selected cells below
- `A` : insert cell above
- `B` : insert cell below
- `X` : cut selected cells
- `C` : copy selected cells
- `Shift-V` : paste cells above
- `V` : paste cells below
- `Z` : undo cell deletion
- `D`, `D` : delete selected cells
- `Shift-M` : merge selected cells, or current cell with cell below if only one cell selected
- `Ctrl-S` : Save and Checkpoint
- `S` : Save and Checkpoint
- `L` : toggle line numbers
- `O` : toggle output of selected cells
- `Shift-O` : toggle output scrolling of selected cells
- `H` : show keyboard shortcuts
- `I`, `I` : interrupt kernel
- `0`, `0` : restart the kernel (with dialog)
- `Esc` : close the pager
- `Q` : close the pager
- `Shift-L` : toggles line numbers in all cells, and persist the setting
- `Shift-Space` : scroll notebook up
- `Space` : scroll notebook down

Shortcuts:

Edit Mode (press `Enter` to enable)

- `Tab` : code completion or indent
- `Shift-Tab` : tooltip
- `Ctrl-]` : indent
- `Ctrl-[` : dedent
- `Ctrl-A` : select all
- `Ctrl-Z` : undo
- `Ctrl-Shift-Z` : redo
- `Ctrl-Y` : redo
- `Ctrl-Home` : go to cell start
- `Ctrl-Up` : go to cell start
- `Ctrl-End` : go to cell end
- `Ctrl-Down` : go to cell end
- `Ctrl-Left` : go one word left
- `Ctrl-Right` : go one word right
- `Ctrl-Backspace` : delete word before
- `Ctrl-Delete` : delete word after
- `Ctrl-M` : command mode
- `Ctrl-Shift-F` : open the command palette
- `Ctrl-Shift-P` : open the command palette
- `Esc` : command mode
- `Shift-Enter` : run cell, select below
- `Ctrl-Enter` : run selected cells
- `Alt-Enter` : run cell, insert below
- `Ctrl-Shift-Minus` : split cell
- `Ctrl-S` : Save and Checkpoint
- `Down` : move cursor down
- `Up` : move cursor up

Tips

1.) **CTRL-M** then **H** to see help

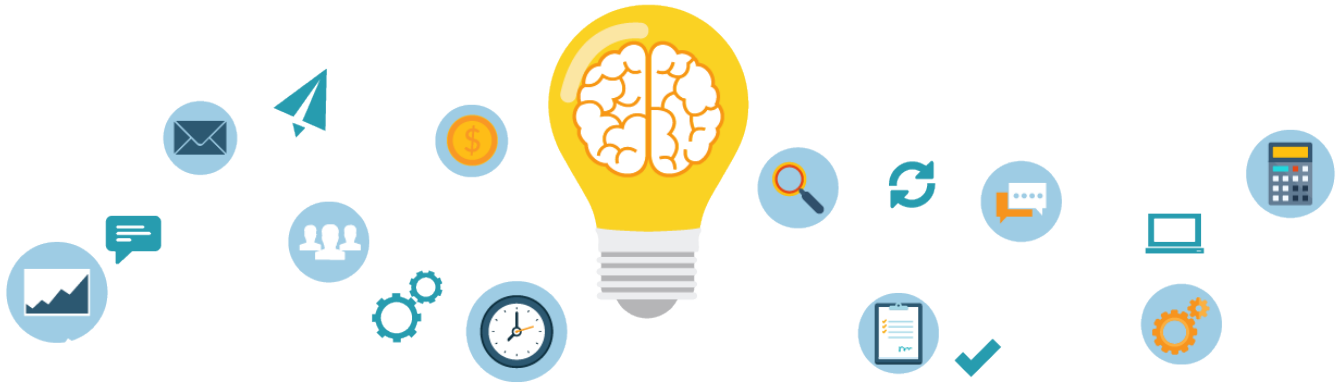
2.) **CTRL-M** then **S** to save the notebook

3.) **CTRL-Enter** to Run the Code but stay in the same cell

4.) **SHIFT-Enter** to Run the Code and advance to the next cell

5.) Using **%pylab inline** preceding everything else in the notebook, imports **matplotlib** and **numpy**. It also enables graphics to be part of the notebook.

6.) You can use **TAB** to see available functions and **SHIFT-TAB** repeatedly for the documentation



Variables and Data Types

Python Programming

Variables and Data Types

Python uses five standard data types:

Numbers

```
varNum = 123
pi = 3.14159
```

Strings

```
varString = "Hello World!"
varText = 'This is a String'
```

Lists

```
varList = ["abc", 123]
```

Tuples

```
varTuple = ('abc', 123, "HELLO")
```

Dictionaries

```
var = 3
varDict = {'first':1, '2':'2nd', 3:var}
```

```
varDict = {}
varDict['first']= 1
varDict['2'] = '2nd'
varDict[3] = var
```

Arithmetic

Addition

```
a = 5 + 3
```

```
> 8
```

Subtraction

```
a = 5 - 3
```

```
> 2
```

Multiplication

```
a = 5 * 3
```

```
> 15
```

Exponent

```
a = 5 ** 3
```

```
> 125
```

Division

```
a = 5 / 3
```

```
> 1.6666666666666667
```

```
a = 5 % 3
```

```
> 2
```

```
a = 5 // 3
```

```
> 1
```

Arithmetic

Increment/Decrement

```
a = 5
a += 1
```

```
> 6
```

Decrement

```
a = 5
a -= 1
```

```
> 4
```

String Concatenation

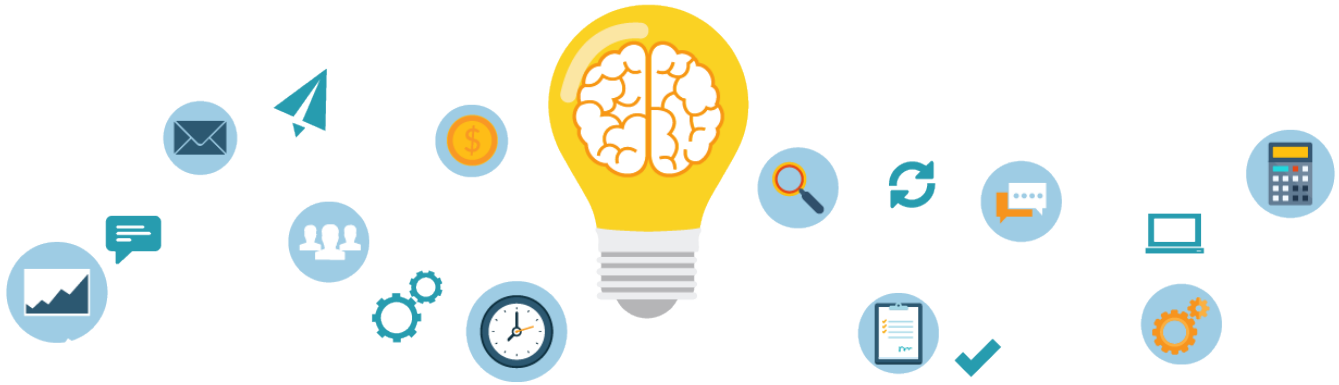
```
a = 'Hello ' + 'World!'
```

```
> Hello World!
```

Complex Expressions

```
a = 3 + 5 - 6 * 2 / 4
```

```
> 5.0
```



Challenge 1!

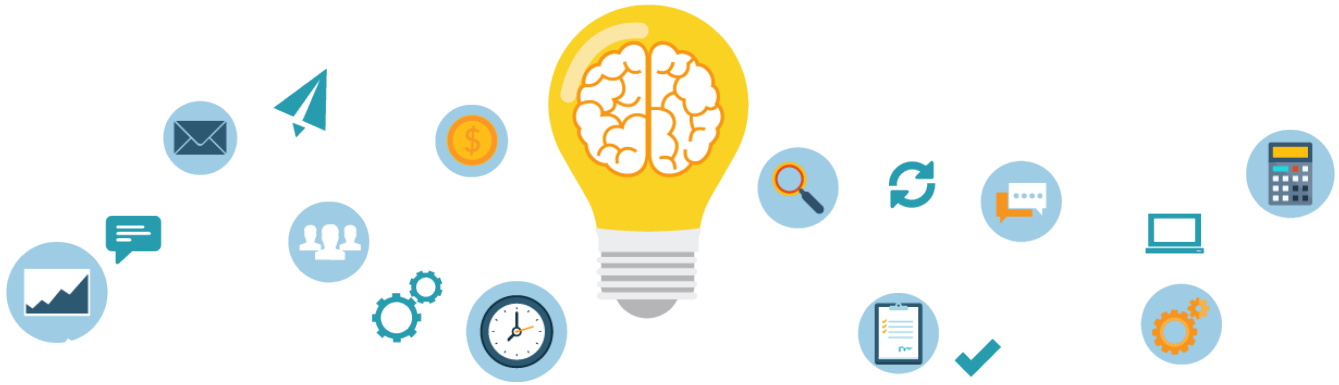
Python Programming

Challenge 1: Convert the following formula into a code

$$g(z) = \frac{1}{1 + e^{-z}}$$

1.) When z = **8** and e = **2.718**; $g(z)$ should be equal to **0.0003**

2.) When z = **2** and e = **2.718**; $g(z)$ should be equal to **0.1192246961081721**



Control Statements and Data Structures

Python Programming

Conditional Statements

Boolean Conditions

```
x = True

if x:
    print ("var x is True")
else:
    print("var x is False")
```

```
> var x is True
```

String Conditions

```
x = "Hello World!"

if x == 'Hello World!':
    print ("var x is Hello World!")
else:
    print("var x is not Hello World!")
```

```
> var x is Hello World!
```

Numerical Conditions

```
x = 10

if x == '10':
    print ("var x is a string")
elif x == 10:
    print ("var x is an integer")
else:
    print("var x is none of the above")
```

```
> var x is an integer
```

Loops

For Loops

```
for var in range(0, 5, 2):
    print(var)
```

```
> 0
> 2
> 4
```

While Loops

```
var = 0
while var < 5:
    print(var)
    var += 2
```

```
> 0
> 2
> 4
```

Nested Loops

```
x = 0
while x < 5:
    for y in range (0, x):
        print(y, end='')
    x += 1
    print()
```

```
> 0
> 01
> 012
> 0123
```

*Note: The loop declaration ends with a colon (:) while its contents inside are indented

Lists

Can contain a number of values comprised of different datatypes.

```
pi = 3.14159
varList = [1, 2, 'A', 'B', 'Hello!', pi]
print(varList[0])
```

```
> 1
```

```
print(varList[4])
```

```
> Hello!
```

```
varList.append('World!')
print(varList[6])
```

```
> World!
```

```
len(varList)
```

```
> 7
```

```
print(varList[5])
```

```
> 3.14159
```

```
varList.remove(pi)
print(varList[5])
```

```
> World!
```

*Note: Lists starts with an index of 0. The *remove()* method erases the first value that matches the parameter.

Dictionaries

```
var = "Hello World!"
varDict = {'first' : 123, 2 : 'abc',
           '3' : var, 4 : ['lista', 'listb']}
print(varDict['first'])
```

```
> 123
```

```
print(varDict[2])
```

```
> abc
```

```
print(varDict['3'])
```

```
> Hello World!
```

```
print(varDict[4])
```

```
> ['lista', 'listb']
```

```
print(varDict[4][1])
```

```
> listb
```

```
len(varDict)
```

```
> 4
```

List Generators and Comprehension

Generators can be used to build list in the memory as objects

```
def gen_num_up_to(n):
    num = 0
    while num < n:
        yield num
        num += 1
```

```
gen_num_up_to(5)
```

```
> <generator object gen_num_up_to at 0x0000>
```

*Note: This creates an object

```
varList = gen_num_up_to(5)
print([var for var in varList])
```

```
> [0, 1, 2, 3, 4]
```

```
def gen_num_up_to(n):
    num = 0
    while num < n:
        yield num
        num += 2

varList = gen_num_up_to(5)
print([var for var in varList])
```

```
> [0, 2, 4]
```

```
varList = range(0, 5, 2)
print([var for var in varList])
```

```
> [0, 2, 4]
```

*Note: Python's built in range function
range([start], stop, [step])

Slicing

```
varList = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
print(varList[:5])
```

```
> [1, 2, 3, 4, 5]
```

```
print(varList[5:])
```

```
> [6, 7, 8, 9, 10]
```

```
print(varList[:-2])
```

```
> [1, 2, 3, 4, 5, 6, 7, 8]
```

```
print(varList[-2:])
```

```
> [9, 10]
```

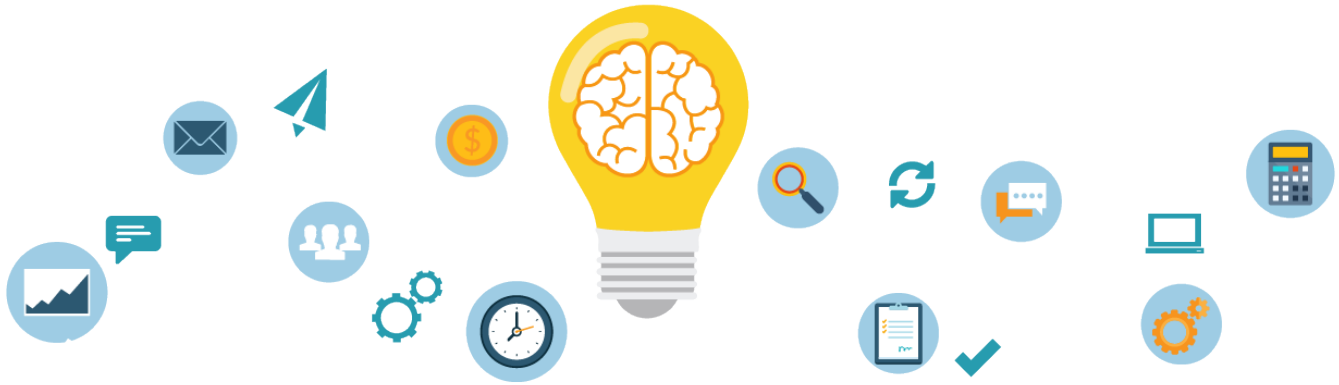
```
print(varList[2:-2])
```

```
> [3, 4, 5, 6, 7, 8]
```

```
print(varList[2:8:2])
```

```
> [3, 5, 7]
```

*Note: *list(start :)*
list(: end)
list(start : end)
list(start : end : step)



Functions

Python Programming

Functions

Python functions use the following notation:

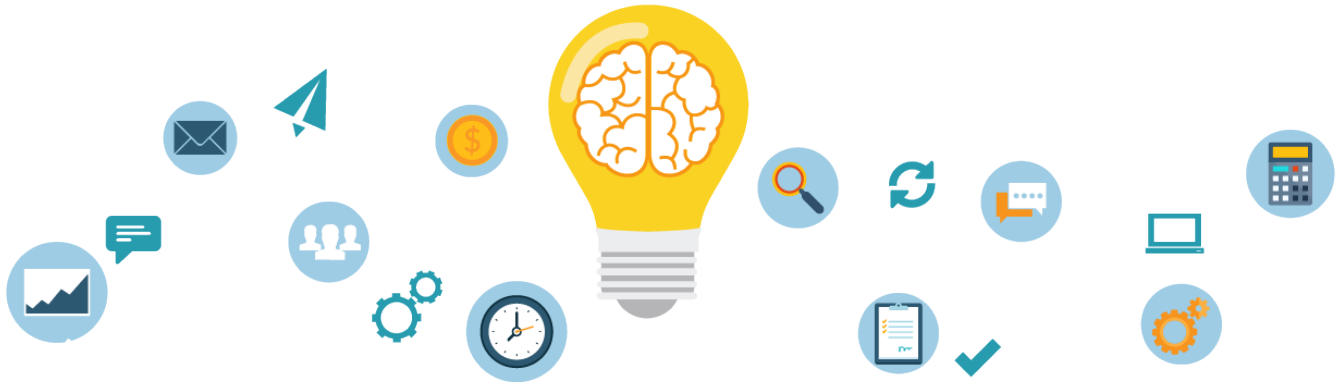
def ***function_name:***

commands

```
def remainder(n, m):
    while True:
        if n - m < 0:
            return n
        else:
            n = n - m

remainder(10, 4)
```

```
> 2
```

*Note: Functions are defined with the keyword **def** before the ***function_name***. Similar to loops, the function definition ends with a colon (:) while its contents inside are indented



Challenge 2!

Python Programming

Challenge 2: Coin Flip

Create a function ***coin_flip()*** that simulates coin flips repeated ***n*** times, with **0** representing tails and 1 representing heads. Make the function a generator with the parameter ***n*** as the number of coin flips.

Use the ***numpy*** library as ***np*** and the function of ***np.random.randint()*** in order to simulate a coin flip. After defining this function, return the result as a ***List*** using List Comprehension discussed previously.

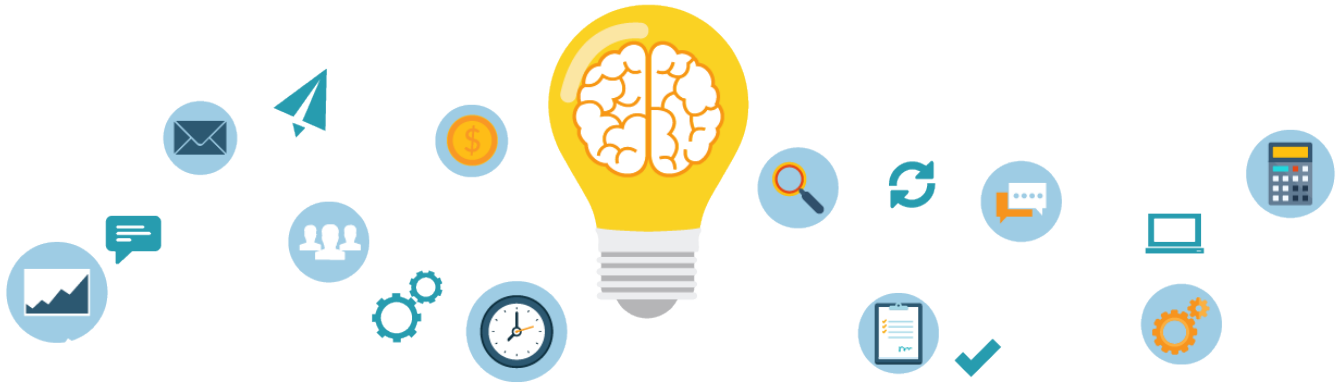
```
import numpy as np
np.random.randint(0, 5)
```

```
> 4
```

*Note: *numpy.random.randint([start], stop)*

```
def coin_flip(n):
    <function commands>

print([var for var in coin_flip(8)])
```

```
> [0, 1, 0, 0, 1, 1, 0, 0]
```



Vectors, Matrices and Computation

Python Programming

Array Range

Declaring an Array containing a Range

```
import numpy as np
np.array(range(7))
```

```
> array([0, 1, 2, 3, 4, 5, 6])
```

```
np.arange(0, 7)
```

```
> array([0, 1, 2, 3, 4, 5, 6])
```

```
print(np.arange(0, 7))
```

```
> [0 1 2 3 4 5 6]
```

Using the *coin_flip()* from Challenge 2

```
np.array([var for var in coin_flip(7)])
```

```
> array([0, 0, 1, 1, 1, 0, 0])
```

Vector Computations

Vector to Scalar Computation

```
varArray = np.arange(0, 5)
varArray * 2
```

```
> array([0, 2, 4, 6, 8])
```

Vector to Vector: Element-wise Multiplication

```
varArrayA * varArrayB
```

```
> array([ 0,  6, 14, 24, 36])
```

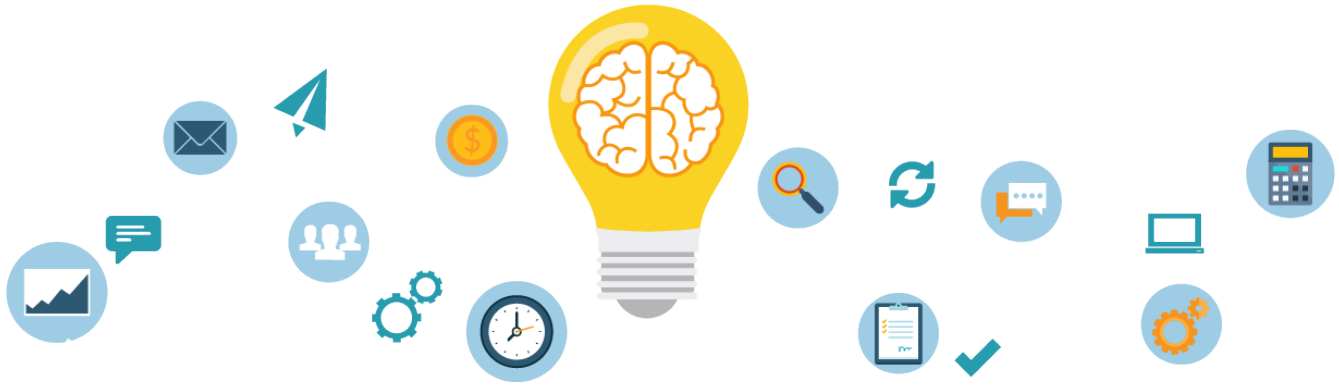
Vector to Vector: Dot Product

```
varArrayA = np.arange(0,5)
varArrayB = np.arange(5,10)

print(varArrayA)
print(varArrayB)
print(np.dot(varArrayA, varArrayB))
```

```
> [0 1 2 3 4]
> [5 6 7 8 9]
> 80
```

*Note: Dot product multiples the elements on the same index and sums the result

$$\begin{array}{rccccc} & [& 0 & 1 & 2 & 3 & 4 &] \\ x & [& 5 & 6 & 7 & 8 & 9 &] \\ \hline & & 0 & + 6 & + 14 & + 24 & + 36 & = \mathbf{80} \end{array}$$



Challenge 3

Python Programming

Challenge 3: Compute the mean of 1000 coin flips

Using the ***coin_flip()*** function from Challenge 2, compute for the result of 1000 coin flips. Recall that the ***mean*** is computed thus:

$$mean(\vec{X}) = \frac{1}{|\vec{X}|} \sum_k X_k$$

*Note: Since a coin flip would randomize from a value of 0 and 1, as the number of flips increase, the ***mean*** of the result should get closer to about 0.5

Matrix Computations

Matrix Initialization

```
varMat = np.random.randint(0, 5, size=(3,3))
varMat
```

```
> array([[2, 3, 4],
         [1, 1, 1],
         [3, 0, 3]])
```

Matrix to Matrix: Matrix Multiplication

```
varMat * varMat
```

```
> array([[ 4,  9, 16],
         [ 1,  1,  1],
         [ 9,  0,  9]])
```

Matrix to Scalar: Element-wise Multiplication

```
varMat * 2
```

```
> array([[4, 6, 8],
         [2, 2, 2],
         [6, 0, 6]])
```

```
print(varMat * 2)
```

```
> [[4 6 8]
   [2 2 2]
   [6 0 6]]
```

Advanced Matrix Operations

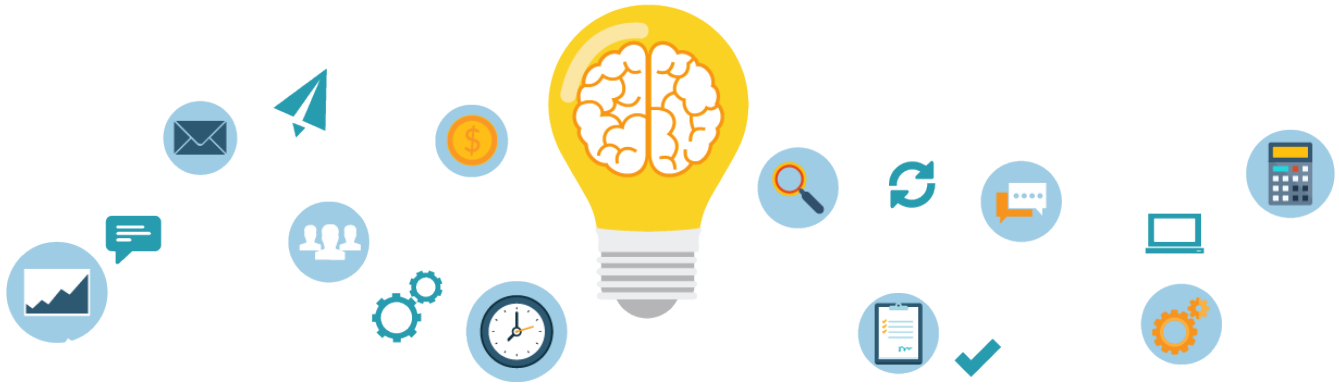
The SciPy Library of Python contains easy-to-use function for formulas and computations used in mathematics, Science, and Engineering. The core packages used are not limited to NumPy, SciPy Library, Matplotlob, Ipython, Sympy, and Pandas.

```
from scipy.linalg import eig
eig(varMat)
```

```
(array([ 10.33653671+0.j,  -2.34683972+0.j,  -0.93324972+0.j,
          1.94355273+0.j]),
 array([[-0.44089993, -0.16783873, -0.70858236, -0.23796553],
        [-0.52872189,  0.82045426, -0.29335679,  0.11803188],
        [-0.48229798, -0.5189514 ,  0.60233167, -0.66084107],
        [-0.5417094 , -0.17138963,  0.221471  ,  0.70194727]]))
```

*Note: The following **SciPy** code on the left solves for an ordinary or generalized eigenvalue problem of a square matrix **varMat**

Visit the **SciPy** home page from the url below for more information:
https://scipy.org/



Pandas: Python Data Analysis Library

Python Programming

Introducing Pandas

Reading CSV files and storing them in a variable

```
import pandas as pd
data = pd.read_csv("movie_metadata.csv")
```

Row and column count

```
data.shape
```

```
(5044, 28)
```

Enumerating the column names

```
data.columns
```

```
Index([u'movie_title', u'color', u'director_name', u'num_critic_for_reviews',
       u'duration', u'director_facebook_likes', u'actor_3_facebook_likes',
       u'actor_2_name', u'actor_1_facebook_likes', u'gross', u'genres',
       u'actor_1_name', u'num_voted_users', u'cast_total_facebook_likes',
       u'actor_3_name', u'facenumber_in_poster', u'plot_keywords',
       u'movie_imdb_link', u'num_user_for_reviews', u'language', u'country',
       u'content_rating', u'budget', u'title_year', u'actor_2_facebook_likes',
       u'imdb_score', u'aspect_ratio', u'movie_facebook_likes'],
      dtype='object')
```

*Note: Pandas is a Python library which allows the use of data-structures and analysis tools.

Please use the movie_metadata.csv dataset provided. You may also obtain the dataset from:

https://www.kaggle.com/deepmatrix/imdb-5000-movie-dataset

Slicing Data Frames

Slicing Data

```
data[:3]
```

	movie_title	color	director_name	num_critic_for_reviews	duration
0	Avatar	Color	James Cameron	723.0	178.0
1	Pirates of the Caribbean: At World's End	Color	Gore Verbinski	302.0	169.0
2	Spectre	Color	Sam Mendes	602.0	148.0

3 rows × 28 columns

```
data[5:8]
```

	movie_title	color	director_name	num_critic_for_reviews	duration
5	John Carter	Color	Andrew Stanton	462.0	132.0
6	Spider-Man 3	Color	Sam Raimi	392.0	156.0
7	Tangled	Color	Nathan Greno	324.0	100.0

3 rows × 28 columns

Indexing Columns and Rows

Indexing Columns

```
data.director_name[:3]
```

```
0     James Cameron
1    Gore Verbinski
2        Sam Mendes
Name: director_name, dtype: object
```

Indexing Columns from a List

```
cols = ["movie_title","director_name"]
data[cols][:5]
```

	movie_title	director_name
0	Avatar	James Cameron
1	Pirates of the Caribbean: At World's End	Gore Verbinski
2	Spectre	Sam Mendes
3	The Dark Knight Rises	Christopher Nolan
4	Star Wars: Episode VII - The Force Awakens ...	Doug Walker

Indexing Rows

```
data.loc[10:12]
```

	movie_title	color	director_name	num_critic_for_reviews	duration
10	Batman v Superman: Dawn of Justice	Color	Zack Snyder	673.0	183.0
11	Superman Returns	Color	Bryan Singer	434.0	169.0
12	Quantum of Solace	Color	Marc Forster	403.0	106.0

3 rows × 28 columns

*Note: *data.loc[var]* uses labels as parameters as opposed to using *data.iloc[var]* or *data[var]* which uses indexes

Finding Movies by James Cameron

```
data[data.director_name == 'James Cameron']
```

	movie_title	color	director_name	num_critic_for_reviews	duration
0	Avatar	Color	James Cameron	723.0	178.0
26	Titanic	Color	James Cameron	315.0	194.0
288	Terminator 2: Judgment Day	Color	James Cameron	210.0	153.0
291	True Lies	Color	James Cameron	94.0	141.0
606	The Abyss	Color	James Cameron	82.0	171.0
2486	Aliens	Color	James Cameron	250.0	154.0
3575	The Terminator	Color	James Cameron	204.0	107.0

7 rows × 28 columns

*Note: **data['director_name']** is also the same as **data.director_name**

The code tries to obtain all movies within the dataset where the **director_name** is **James Cameron**

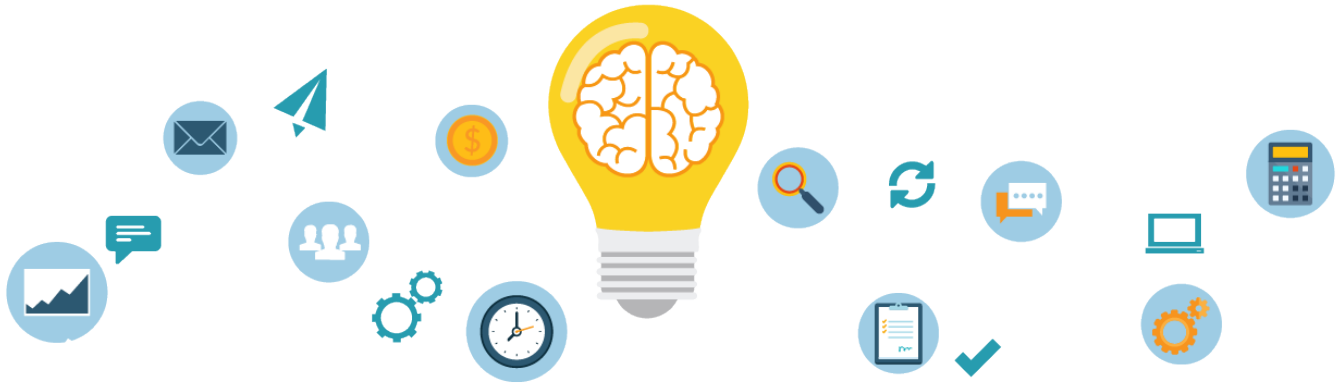
Sort Films by Gross Earnings

```
sorted_data = data.sort_values(by="gross", ascending=False)
sorted_data[:5]
```

	movie_title	color	director_name	num_critic_for_reviews	duration
0	Avatar	Color	James Cameron	723.0	178.0
26	Titanic	Color	James Cameron	315.0	194.0
29	Jurassic World	Color	Colin Trevorrow	644.0	124.0
794	The Avengers	Color	Joss Whedon	703.0	173.0
17	The Avengers	Color	Joss Whedon	703.0	173.0

5 rows × 28 columns

*Note: Sorting values are helpful in being able to arrange which ones go first. With the inclusion of slicing, there are plenty of applications and queries one could implement with the use of these two functions together in one code



Challenge 4!

Python Programming

Challenge 4: Get the Top 5 films of Michael Bay

Create a variable named **sorted_data** which would contain the sorted values by gross of each film of Michael Bay. Output the top 5 films into a table. Your answer should be similar to that of the output below.

	movie_title	color	director_name	num_critic_for_reviews	duration
36	Transformers: Revenge of the Fallen	Color	Michael Bay	366.0	150.0
53	Transformers: Dark of the Moon	Color	Michael Bay	428.0	154.0
112	Transformers	Color	Michael Bay	396.0	144.0
37	Transformers: Age of Extinction	Color	Michael Bay	378.0	165.0
151	Armageddon	Color	Michael Bay	167.0	153.0

5 rows × 28 columns

*Note: You should be able to obtain an output similar to that of the table on the left.

You would need to implement a form of sorting as well as slicing in order to complete this challenge

Multiple Conditions

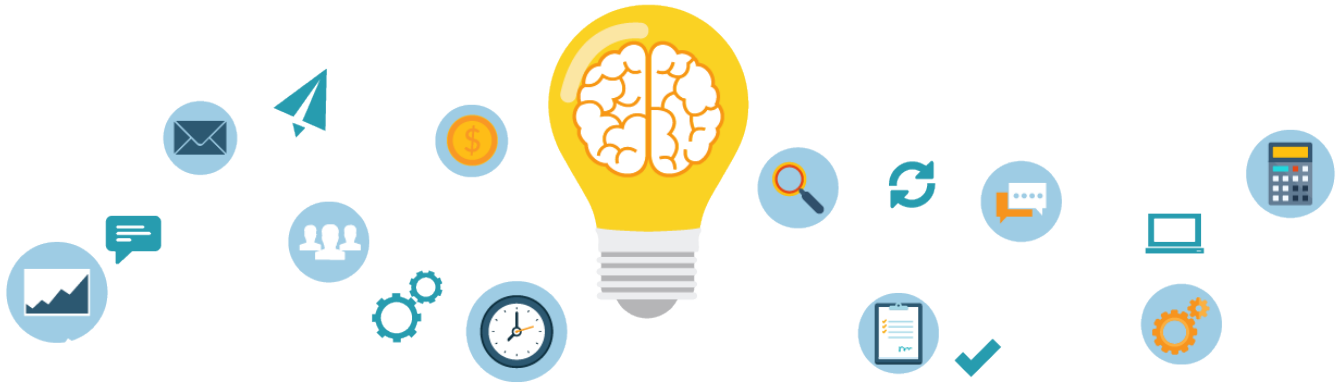
Find Canadian films that have **Hugh Jackman** as the ***actor_1_name***

```
data[(data['country'] == 'Canada') & (data['actor_1_name'] == 'Hugh Jackman')]
```

	movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes
34	X-Men: The Last Stand	Color	Brett Ratner	334.0	104.0	420.0
210	X-Men 2	Color	Bryan Singer	289.0	134.0	0.0

2 rows × 28 columns

*Note: There is a difference between the syntax ***and*** as well as the symbol ***&***, it Is that the syntax ***and*** does a **Boolean** operation and checks if both statements are **True** then the result would be **True**, while the symbol ***&***, does a **Bit-wise AND** operator



Challenge 5!

Python Programming

Challenge 5: Actor with the movie that grossed an exact value

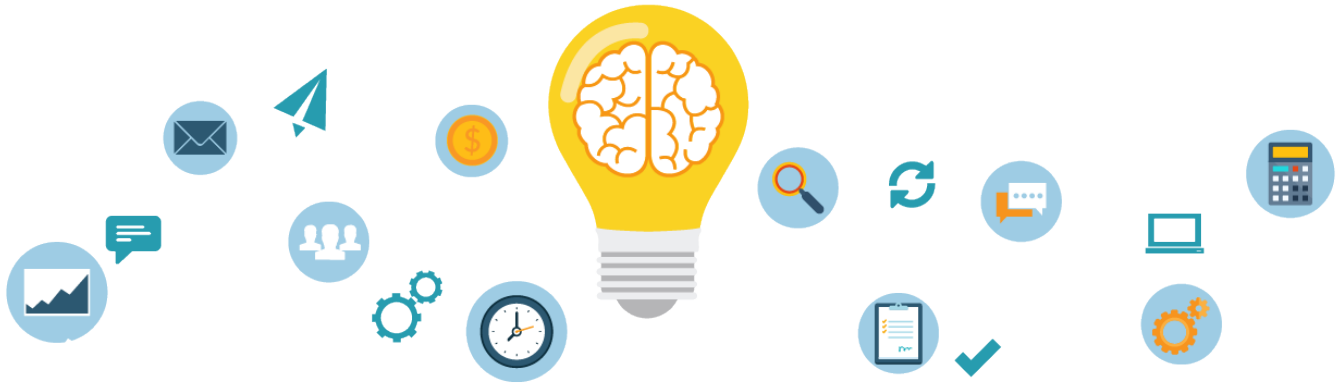
Find the actor who is the ***actor_1_name*** for the movie that grossed exactly **67344392**. The resulting output should be similar to the one below.

	movie_title	color	director_name	num_critic_for_reviews	duration	director_facebook_likes
347	A Good Day to Die Hard	Color	John Moore	412.0	101.0	212.0

1 rows × 28 columns

Once completed, try to find the films with whose ***actor_3_name*** is **Piolo Pascual** and the ***actor_1_name*** is the person from **Armageddon**

*Note: You may check the previously discussed topics on Multiple Conditions for Pandas



Preprocessing

Python Programming

Datatypes of the Columns

```
data.dtypes
```

```
movie_title                 object
color                       object
director_name               object
num_critic_for_reviews     float64
duration                   float64
director_facebook_likes    float64
actor_3_facebook_likes     float64
actor_2_name                object
actor_1_facebook_likes     float64
gross                      float64
genres                      object
actor_1_name                object
num_voted_users            float64
cast_total_facebook_likes  float64
actor_3_name                object
facenumber_in_poster       float64
plot_keywords               object
movie_imdb_link             object
num_user_for_reviews       float64
language                    object
country                     object
content_rating              object
budget                     float64
title_year                 float64
actor_2_facebook_likes     float64
imdb_score                 float64
aspect_ratio               float64
movie_facebook_likes       float64
dtype: object
```

*Note: Since the datatypes are automatically labelled as they are read from the dataset, numbers are typically assigned to have a **float64** value even if they do not possess any fractions

Filling up Blank values

Converting ***actor_1_facebook*** from a datatype of float64 to int64

```
# will have an error
data.actor_1_facebook_likes.astype(np.int64)
```

```
ValueError: Cannot convert non-finite values (NA or inf) to integer
```

```
data['actor_1_facebook_likes'] = data['actor_1_facebook_likes'].fillna(0)
data['actor_1_facebook_likes'] = data['actor_1_facebook_likes'].astype(np.int64)
data.dtypes
```

```
movie_title                 object
color                       object
director_name               object
num_critic_for_reviews     float64
duration                   float64
director_facebook_likes    float64
actor_3_facebook_likes     float64
actor_2_name                object
actor_1_facebook_likes       int64
gross                      float64
```

*Note: An error has occurred due to the reason that *conversion cannot be done when at least one of the values are NA or Infinite*. If this happens, fill up or initialize the NA values before converting

The parameter used by ***fillna()*** replaces all NA values to the specified value

Lambda or the Nameless Function

Applying a Lambda

```
data['actor_1_facebook_likes'].apply(lambda x: np.sqrt(x))[:5]
```

```
0     31.622777
1    200.000000
2    104.880885
3    164.316767
4     11.445523
Name: actor_1_facebook_likes, dtype: float64
```

*Note: **Lambda** is commonly used in the event that a function would only be used once as opposed to it being continuously reused. This better utilized the memory being used by the application.

Cleaning the Dataset

```
data.movie_title.tolist()[:5]
```

```
['Avatar\xc2\xa0',
 "Pirates of the Caribbean: At World's End\xc2\xa0",
 'Spectre\xc2\xa0',
 'The Dark Knight Rises\xc2\xa0',
 'Star Wars: Episode VII - The Force Awakens\xc2\xa0            ']
```

*Note: The \xc2\xa0 after each title are unicode **Escape Characters**

```
data.movie_title.apply(
    lambda x: x.encode().decode('unicode_escape').encode('ascii','ignore').decode('utf-8')).tolist()[:5]
```

```
['Avatar',
 "Pirates of the Caribbean: At World's End",
 'Spectre',
 'The Dark Knight Rises',
 'Star Wars: Episode VII - The Force Awakens            ']
```

*Note: The problem with this statement is that it removes the **ñ**

Cleaning the Dataset

```
print(data[data['movie_title'].str.contains("ñ")]["movie_title"])
print("The following removed the enye:")
print(data[data['movie_title'].str.contains("ñ")]["movie_title"].apply(
    lambda x: x.encode().decode('unicode_escape').encode('ascii','ignore').decode('utf-8')).tolist())
```

```
4796    Quinceañera
Name: movie_title, dtype: object
The following removed the enye:
['Quinceaera']
```

*Note: The problem with this statement is that it removes the **ñ**

```
data["movie_title"] = data["movie_title"].apply(
    lambda x: x.encode().decode('unicode_escape').encode('ascii','ignore').decode('utf-8'))
data["movie_title"][:4]
```

```
0                                     Avatar
1    Pirates of the Caribbean: At World's End
2                                      Spectre
3                      The Dark Knight Rises
Name: movie_title, dtype: object
```

Data Summaries

Referencing the original data

```
cleaned_data = data.fillna(0)
```

```
cleaned_data.describe()
```

	num_critic_for_reviews	duration	director_facebook_likes
count	5044.000000	5044.000000	5044.000000
mean	138.776764	106.861023	672.218279
std	121.795659	25.869713	2785.611681
min	0.000000	0.000000	0.000000
25%	48.000000	93.000000	6.000000
50%	108.500000	103.000000	45.000000
75%	194.000000	118.000000	189.000000
max	813.000000	511.000000	23000.000000

Saving the output to a CSV file

```
cleaned_data.to_csv('movie_metadata_cleaned.csv')
```

*Note: When adding a new reference to the original data, only new cells are allocated memory, while the unchanged cells are referenced to the original

The describe function allows an easy-to-use summarization of aggregated date

Data Correlations

```
cleaned_data.corr()
```

	num_critic_for_reviews	duration	director_facebook_likes
num_critic_for_reviews	1.000000	0.269957	0.184942
duration	0.269957	1.000000	0.161057
director_facebook_likes	0.184942	0.161057	1.000000
actor_3_facebook_likes	0.273130	0.123878	0.121308
actor_1_facebook_likes	0.193322	0.089381	0.092385
gross	0.524884	0.254773	0.149975
num_voted_users	0.626649	0.313770	0.298072
cast_total_facebook_likes	0.266189	0.123360	0.121131
facenumber_in_poster	-0.033755	0.004985	-0.041191
movie_facebook_likes	0.682596	0.195677	0.162117

*Note: When using a dataframe from Pandas, such as that of the variable **cleaned_data**, you may also look for a number of functions that are readily available. What is shown here is how to easily obtain the correlation of one column to the other

Outliers: Clipping to the n^{th} percentile

Removing outliers may be adjusted based on the percentile.

```
cleaned_data.duration.quantile(1)
```

```
511.0
```

```
cleaned_data.duration.quantile(0.99)
```

```
189.0
```

```
cleaned_data.duration.quantile(0.95)
```

```
146.0
```

```
cleaned_data.duration.quantile(0.90)
```

```
134.0
```

*Note: Clipping to the 99^{th} percentile is just one of the many rules-of-thumb used in practice. However it does not always work, especially if there are too many outliers. The percentile should be adjusted based on the dataset being used

Aggregation: Dataframe Grouping by **title_year**

```
cleaned_data["title_year"] = cleaned_data["title_year"].astype(np.int64)
movies_per_year = cleaned_data.groupby("title_year").size()
movies_per_year[-3:]
```

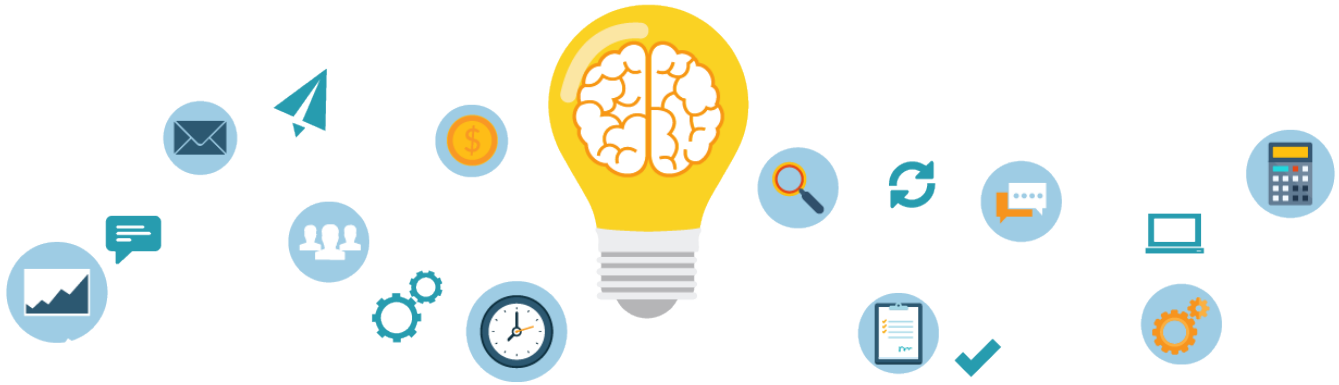
```
title_year
2014    253
2015    226
2016    106
dtype: int64
```

Group together the **title_year** as the *index*

```
like_per_year = cleaned_data.groupby("title_year")["movie_facebook_likes"].mean()
like_per_year[-3:]
```

```
title_year
2014    19850.422925
2015    19775.920354
2016    17610.669811
Name: movie_facebook_likes, dtype: float64
```

*Note: There are also readily available aggregation functions present such as **size()**, **mean()**, and the like



Data Visualization

Python Programming

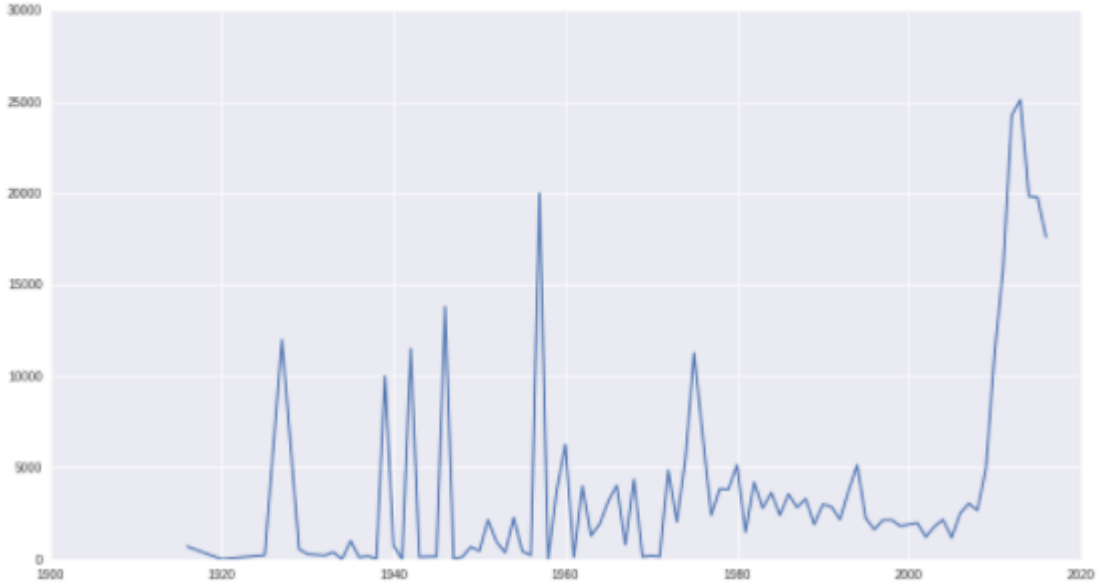
Matplotlib: Line Plot of Avaerage Facebook Likes per Year

Importing Seaborn

```
# seaborn allows for a better look-and-feel of the plots
import seaborn as sns
```

Average Facebook Likes per Year

```
fig = plt.figure(figsize=(15,8))
years = like_per_year.index.tolist()[1:]
likes = like_per_year[1:]
plt.plot(years, likes)
plt.show()
```

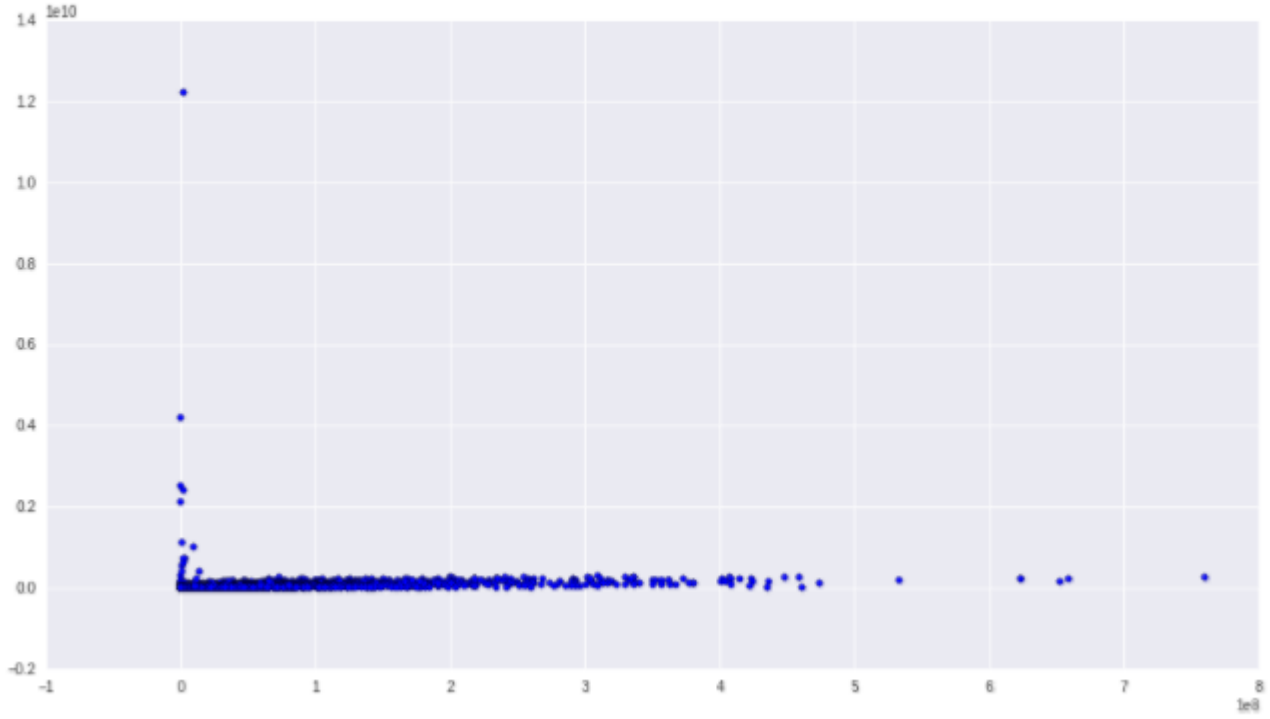


*Note: Plot takes in the x-axis as the first parameter and the second, the y-axis values

figsize=(15,8) is the image size for the plot

Matplotlib: Scatterplot of Gross vs Budget

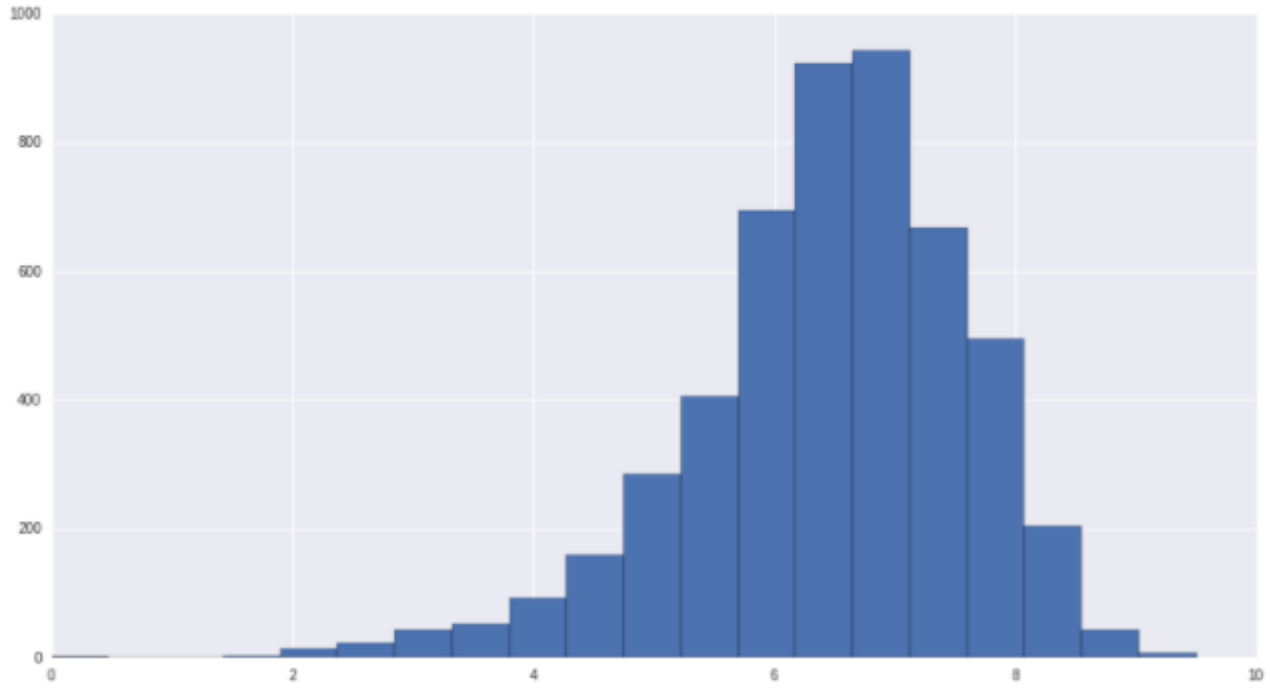
```
fig = plt.figure(figsize=(15,8))
plt.scatter(cleaned_data["gross"], cleaned_data["budget"])
plt.show()
```



*Note: plt.figure(figsize=(15,8)) is used to render the size of the graph. You may try to play with the parameters in the size in order to create a better fit for your visualization

Matplotlib: Histogram of IMDB Scores

```
fig = plt.figure(figsize=(15,8))
plt.hist(cleaned_data["imdb_score"], bins=20)
plt.show()
```

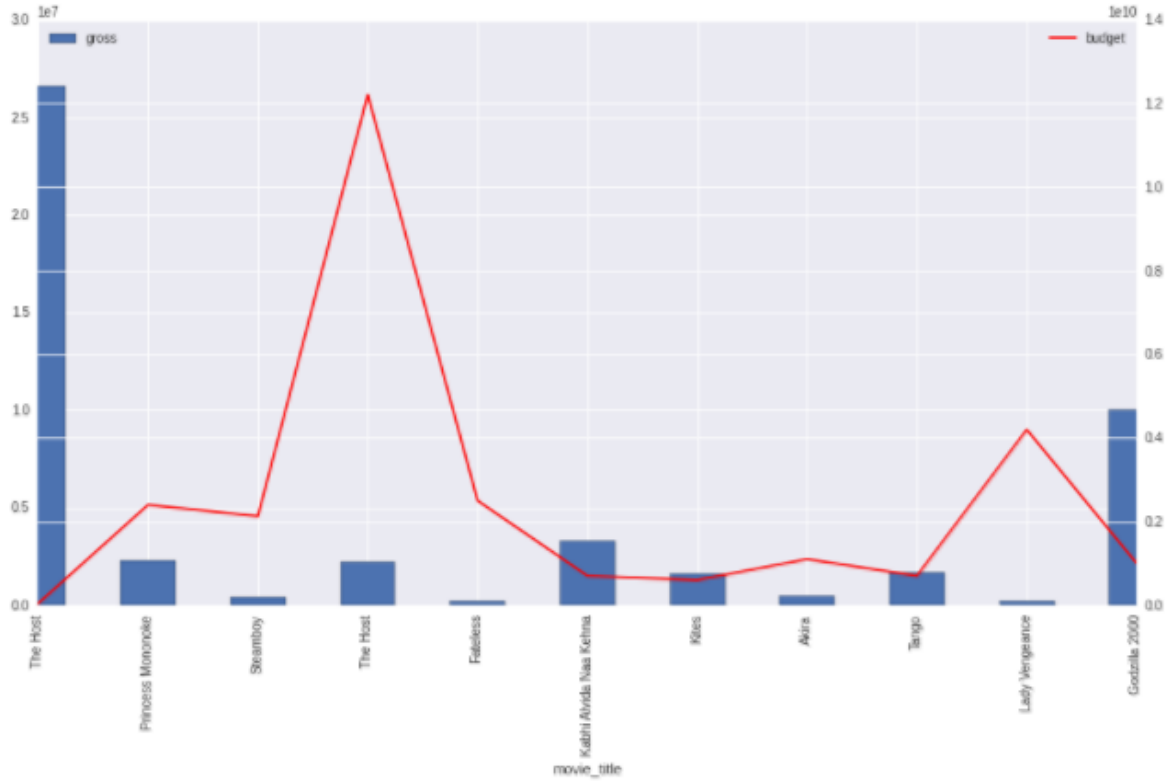


*Note: When using Matplotlib, you may modify a number of parameters in your graphs. In this example, the bin is set to bin=20 however if you set that to bin=50, you would get a different data summarization for your visualization. You may press **SHIFT + TAB** in order to see other parameters you can modify

Pandas: Overlay Barplot and Line Graph

Barplot of the gross earning of the 10 movies with the highest budget superimposed with their budget as a line graph

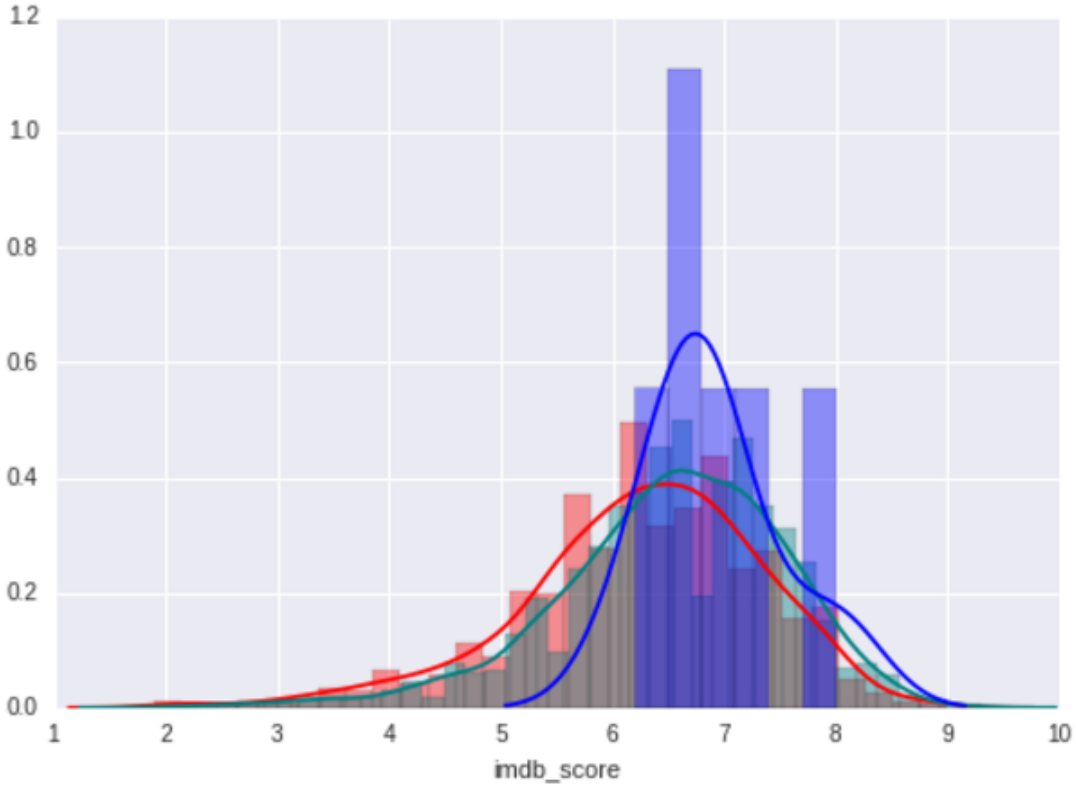
```
top_budget = cleaned_data.sort_values(by="budget",
    ascending=False)["movie_title"][:10].tolist()
top_budget_data = cleaned_data[cleaned_data["movie_title"]
    .isin(top_budget)]

ax1 = top_budget_data.plot(x="movie_title", y="gross",
    kind="bar", figsize=(15,8))
ax2 = ax1.twinx()
top_budget_data.plot(x="movie_title", y="budget",
    kind="line", color='red', figsize=(15,8), ax=ax2)
ax1.legend(loc='upper left')
ax2.legend(loc='upper right')
plt.show()
```



Histograms of imdb_scored

Histograms of imdb_scored according to the different content rating types

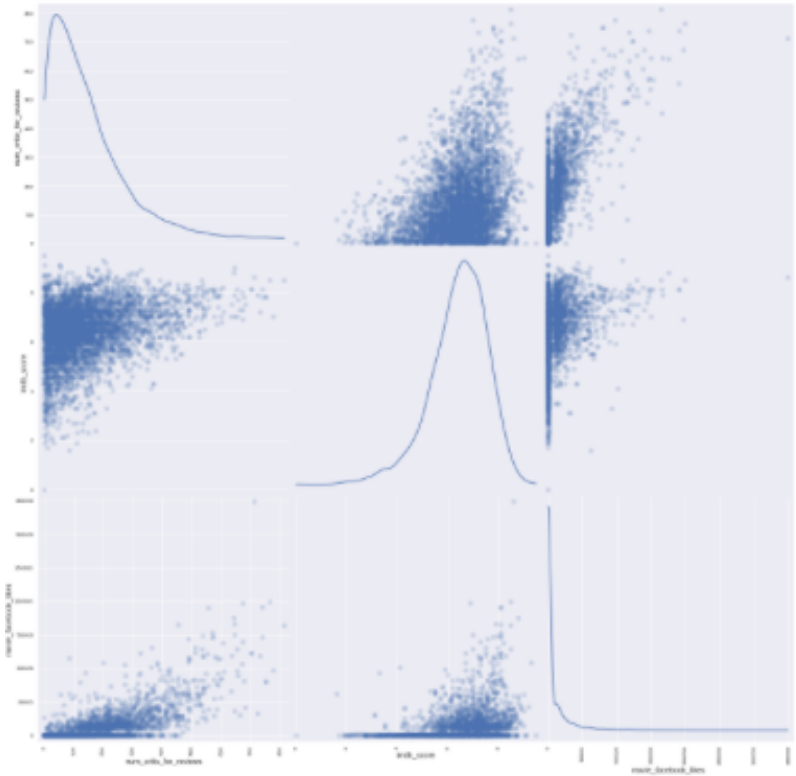
```
import seaborn as sns

ax = sns.distplot(cleaned_data[cleaned_data['content_rating']
    == 'PG-13']["imdb_score"], color='red')
sns.distplot(cleaned_data[cleaned_data['content_rating']
    == 'R']["imdb_score"], color='teal', ax=ax)
sns.distplot(cleaned_data[cleaned_data['content_rating']
    == 'GP']["imdb_score"], color='blue', ax=ax)
```



*Note: Using overlay graphs allow better differentiation and comparison of columns. Take note that some data may only be comparable when normalized.

Scatter Matrix

```
from pandas.tools.plotting import scatter_matrix
cols = ["num_critic_for_reviews", "imdb_score", "movie_facebook_likes"]
scatter_matrix(cleaned_data[cols], alpha=0.2, figsize=(20, 20),
               diagonal='kde', marker='o')
plt.show()
```



*Note: Scatter Matrix are good to use when trying to determine the correlation of two columns in the dataset

Seaborn: Conditional Formatting

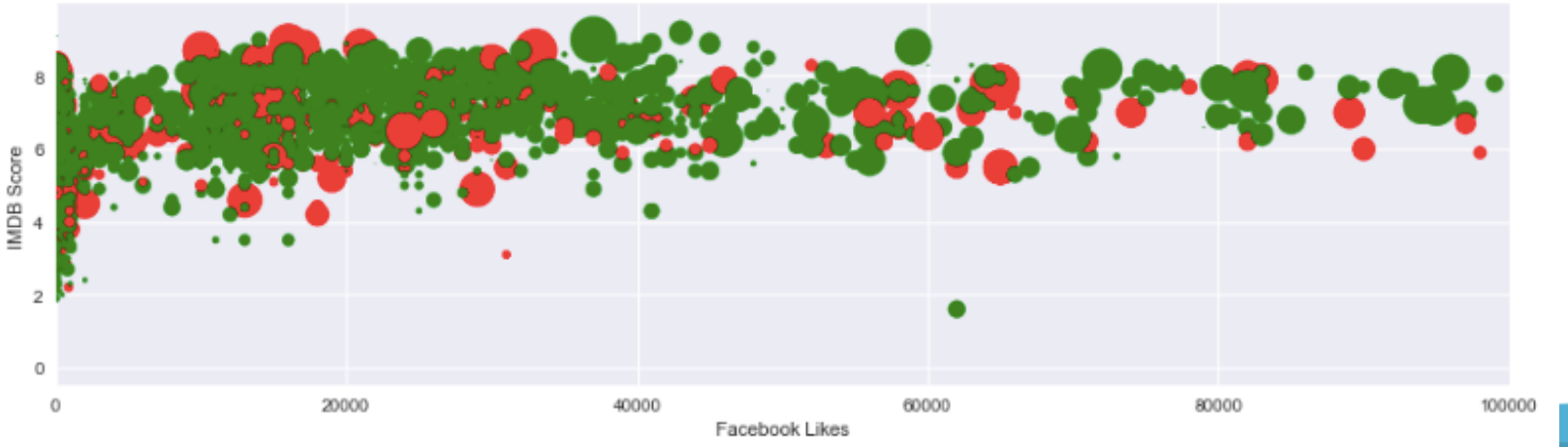
```
import seaborn as sns

cm = sns.light_palette("green", as_cmap=True)
cols = ["movie_title", "imdb_score","gross"]
color_me = cleaned_data[cols][:10]
s = color_me.style.background_gradient(cmap=cm)
s
```

*Note: Seaborn allows better visualization of graphs and one of its useful functions is to render the visualization with conditional formatting to easily determine the difference in values of your dataset

	movie_title	imdb_score	gross
0	b'Avatar'	7.9	7.60506e+08
1	b"Pirates of the Caribbean: At World's End"	7.1	3.09404e+08
2	b'Spectre'	6.8	2.00074e+08
3	b'The Dark Knight Rises'	8.5	4.48131e+08
4	b'Star Wars: Episode VII - The Force Awakens '	7.1	0
5	b'John Carter'	6.6	7.30587e+07
6	b'Spider-Man 3'	6.2	3.3653e+08
7	b'Tangled'	7.8	2.00807e+08
8	b'Avengers: Age of Ultron'	7.5	4.58992e+08
9	b'Harry Potter and the Half-Blood Prince'	7.5	3.01957e+08

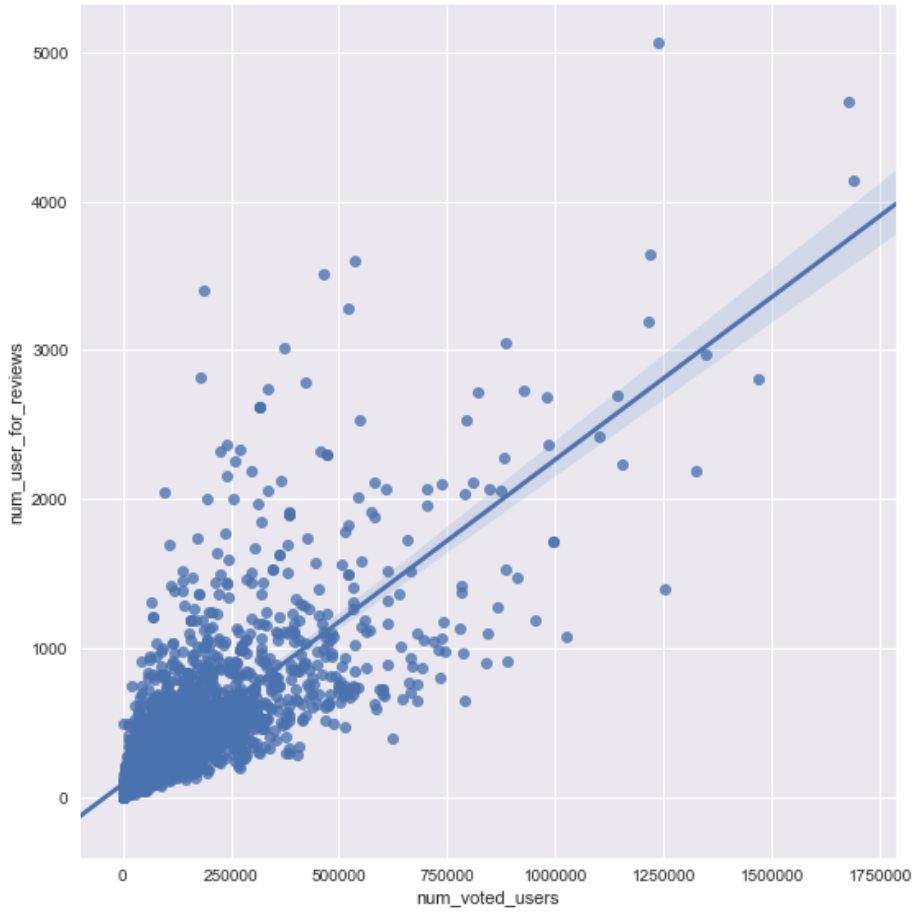
Advanced Graphs: Plotting more than 2 variables

```
plt.figure(figsize=(15,8))
sizes = 1000*((cleaned_data["gross"] - min(cleaned_data["gross"]))
              / max(cleaned_data["gross"] ) - min(cleaned_data["gross"] ))
colors = np.where(cleaned_data.genres.str.contains("Fantasy"), 'red', 'green')
plt.scatter(x=cleaned_data["movie_facebook_likes"],
            y=cleaned_data["imdb_score"], c=colors,
            s = sizes)

plt.xlabel("Facebook Likes")
plt.ylabel("IMDB Score")
plt.xlim((0,100000))
plt.show()
```



*Note: This visualization shows 4 variables. The red nodes designate Fantasy while the green nodes are movies which are not classified as Fantasy. The size of each node is proportional to the gross of the movie. The x-axis shows the number of Facebook Likes. Lastly, the y-axis shows the IMDB Score

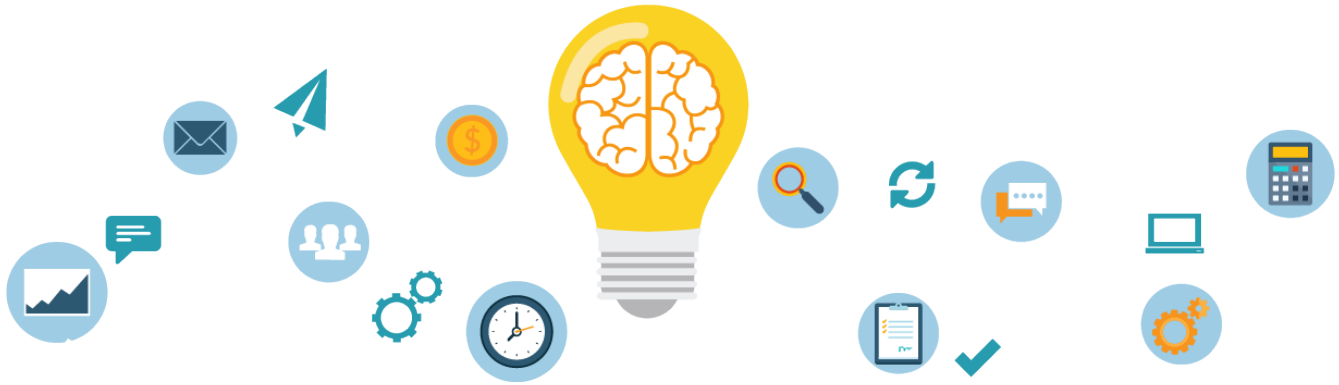
Seaborn: Line Plot with Regression



Voted Users and Reviews

```
sns.lmplot(x="num_voted_users",
           y="num_user_for_reviews",
           data=cleaned_data, size=8)
```

*Note: This visualization shows 4 variables. The red nodes designate Fantasy while the green nodes are movies which are not classified as Fantasy. The size of each node is proportional to the gross of the movie. The x-axis shows the number of Facebook Likes. Lastly, the y-axis shows the IMDB Score



Challenge 6!

Python Programming

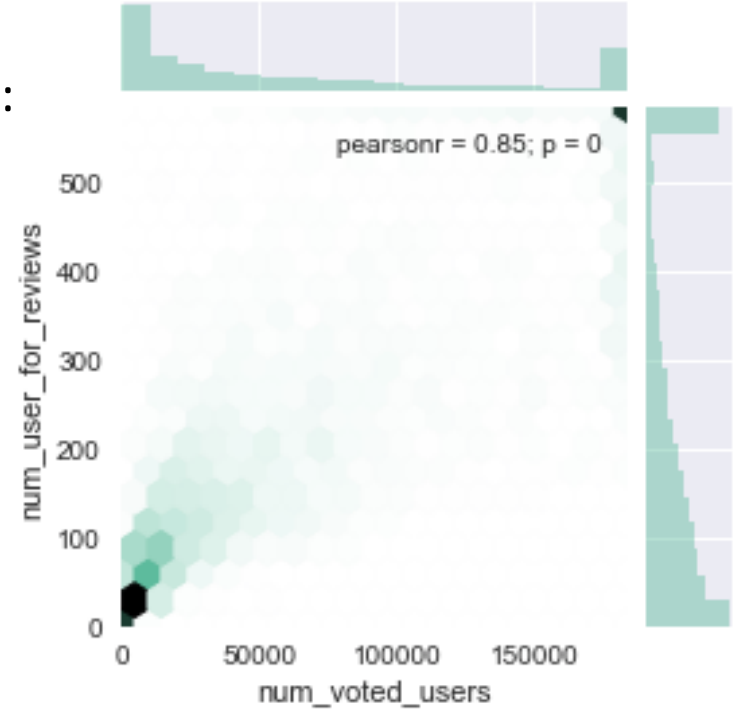
Challenge 6: Fix the plot by clipping with the Tukey's Test (k=1)

```
sns.jointplot(x="num_voted_users", y="num_user_for_reviews", data=cleaned_data, size=4,
              kind="hex", color="#4CB391")
```

$$[Q_1 - k(Q_3 - Q_1), Q_3 + k(Q_3 - Q_1)]$$

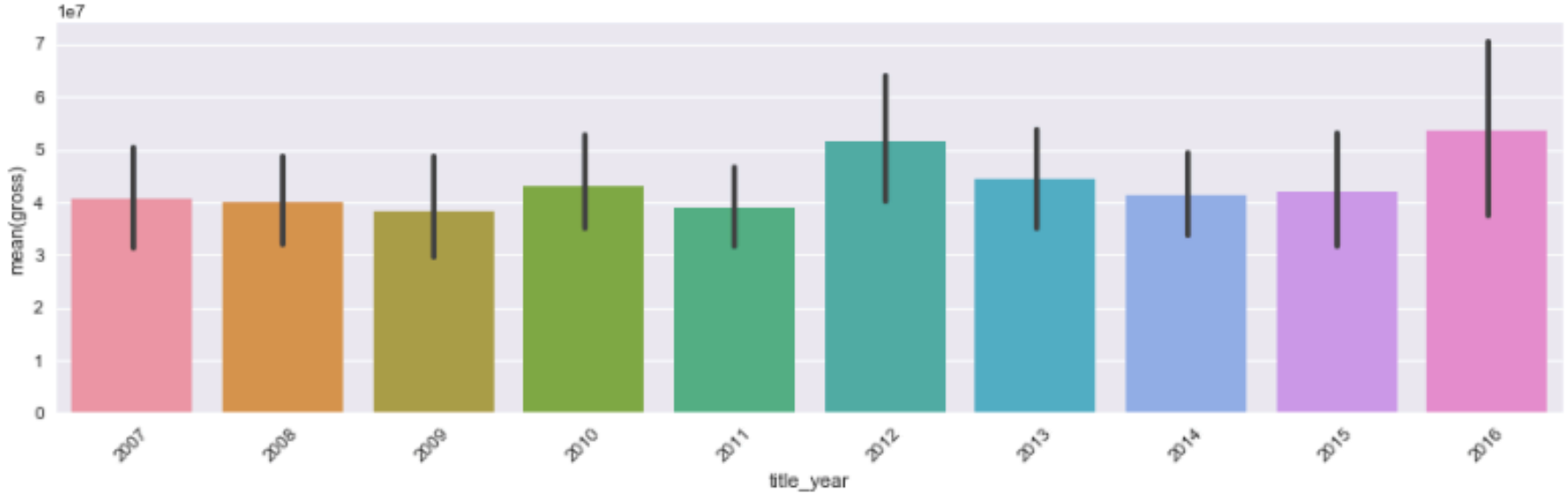
*Note: The original output of the code would result in a graph that is You may have to research about the function ***np.clip()*** which limits the value in an array.

Answer :

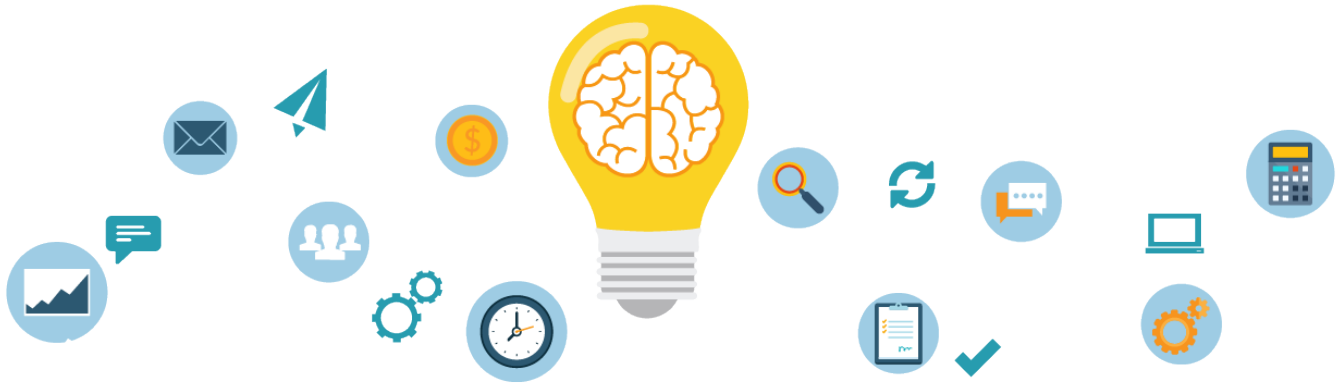


Seaborn: Barplot Gross earnings of the last 10 years

```
cleaned_data["title_year"] = cleaned_data["title_year"].astype(np.int64)
latest_10_years = np.sort(cleaned_data["title_year"].unique())[-10:]

latest_movies_data=cleaned_data[cleaned_data["title_year"].isin(latest_10_years)]
plt.figure(figsize=(15,4))
sns.barplot(x="title_year", y="gross", data=latest_movies_data)
plt.xticks(rotation=45)
plt.show()
```



*Note: The list of years were obtained first and saved in a list as **last_10_years** then it was used as the y-axis labels

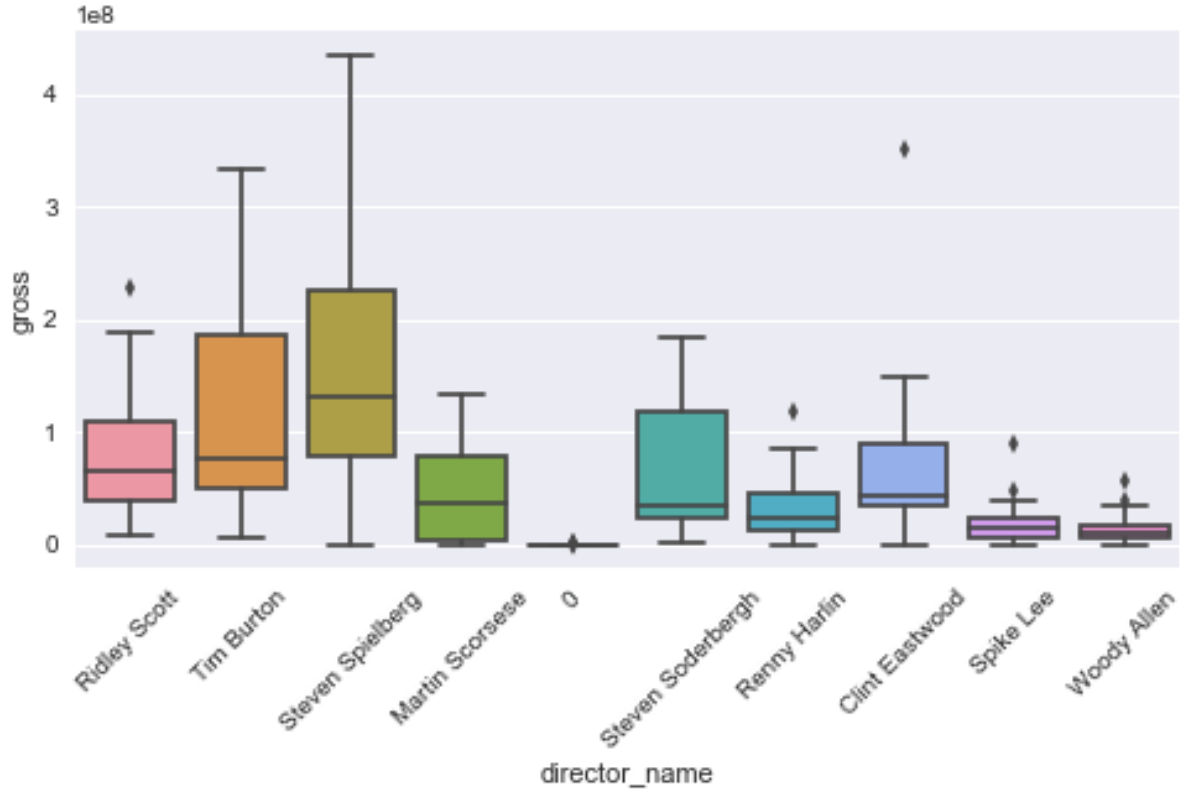


Challenge 7!

Python Programming

Challenge 7: Get the top 10 directors with most movies directed and use a boxplot for their gross earnings

The result of your code should be similar to this:



*Note: You may have to use groupings, sortings, and slicing in order to be able to complete this task.

Try to obtain each parameter needed one at a time in order to build your data. Understanding which one is aggregated, sorted, and sliced to get the top 10.