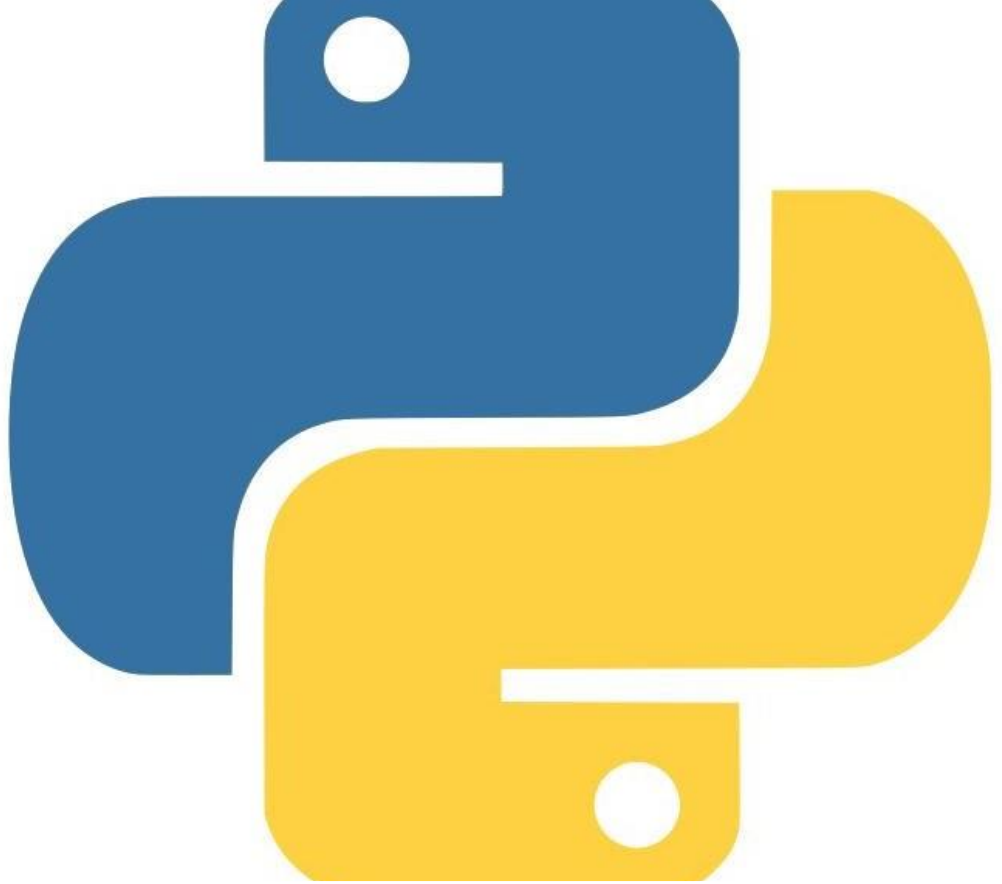


Python Workshop for Data Processing and Visualization

Nov 2020

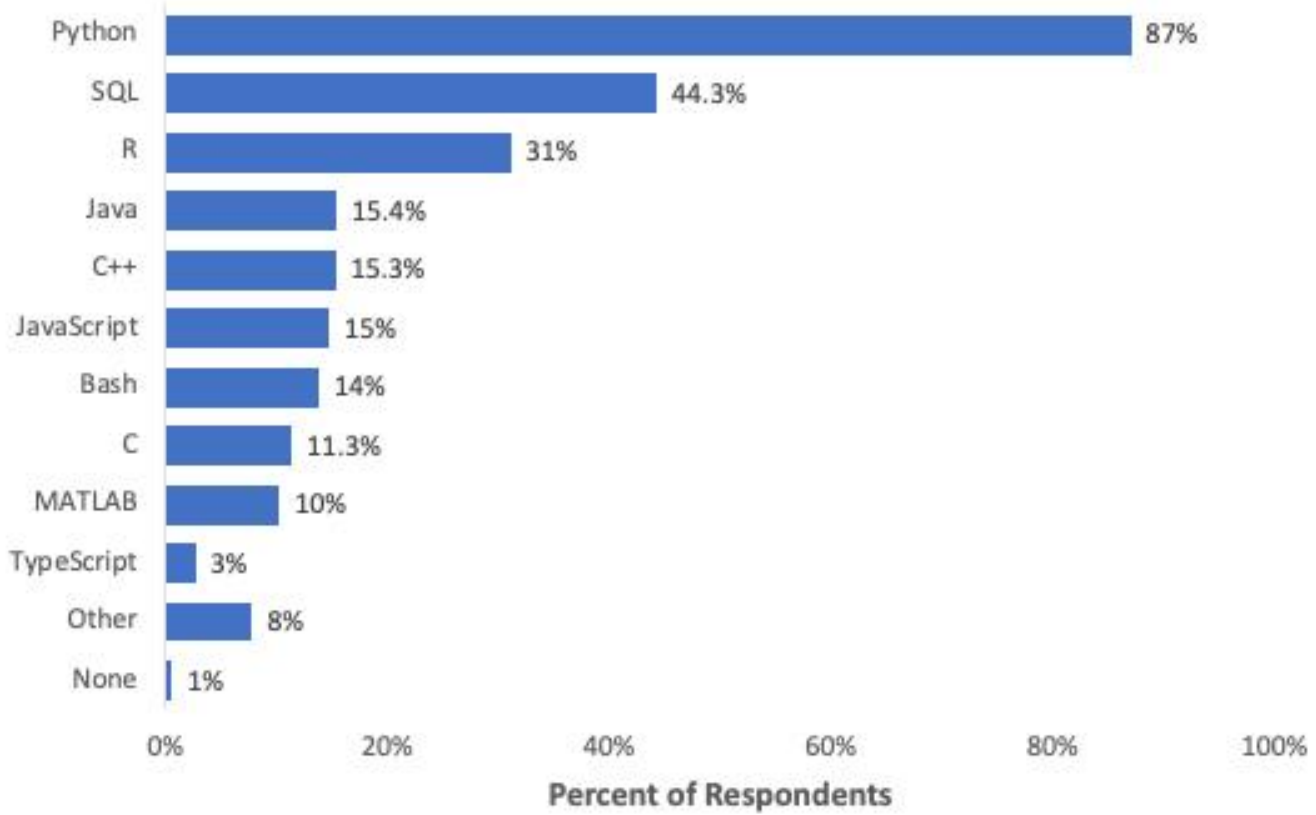
Workshop Topics

- Python in data science ecosystem
- Basic usage of Jupyter Notebook
- Basic Python Concepts & Syntax
- Essential Python Libraries
- Data Loading and Processing with Pandas
- Data Cleaning and Preparation with Pandas
- Plotting and Visualization

Python in data science ecosystem

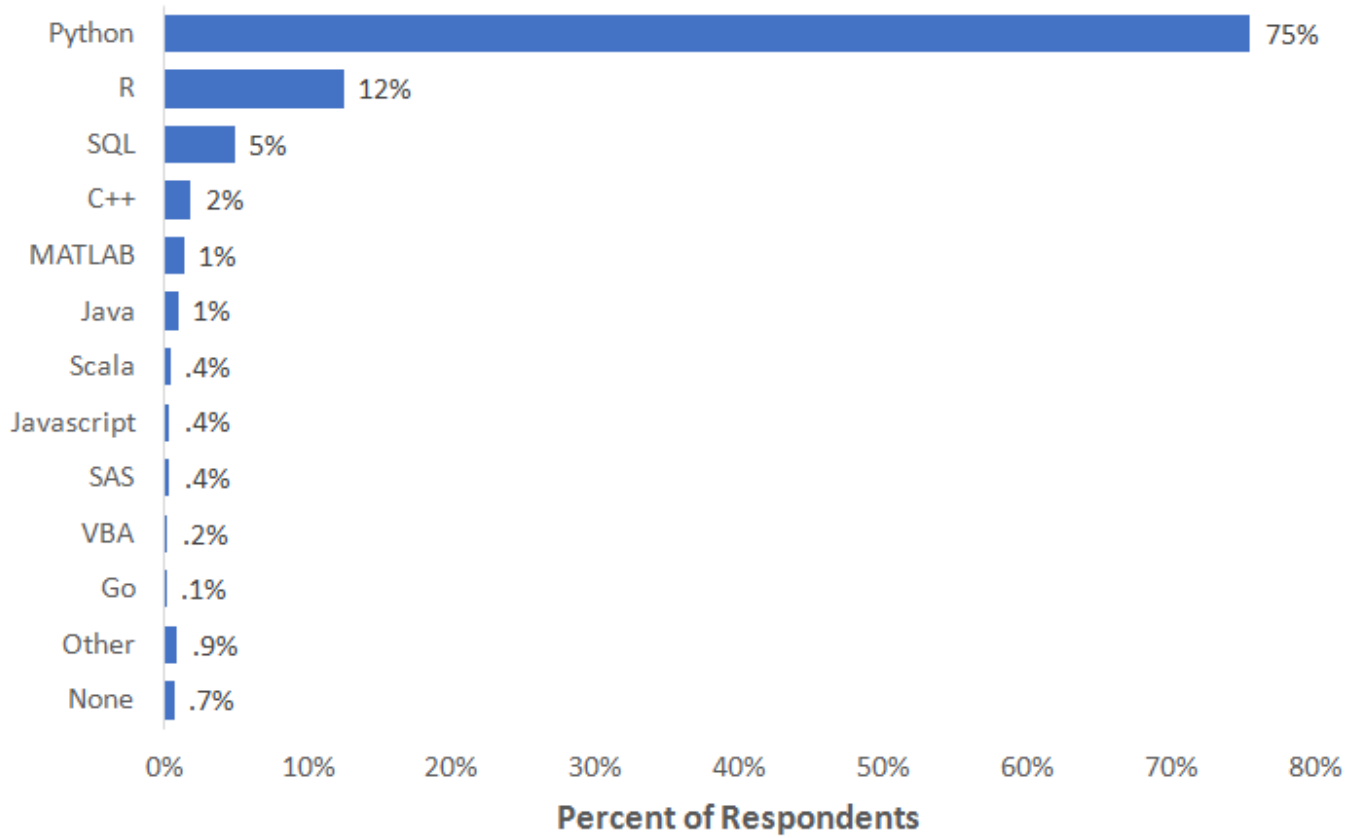
Platform	
Tools	
Frameworks	
Language	
Format	
SaaS	

What programming languages do you use on a regular basis?



Note: Data are from the 2019 Kaggle ML and Data Science Survey. You can learn more about the study here: https://www.kaggle.com/c/kaggle-survey-2019.
A total of 19717 respondents completed the survey; the percentages in the graph are based on a total of 14762 respondents who provided an answer to this question.

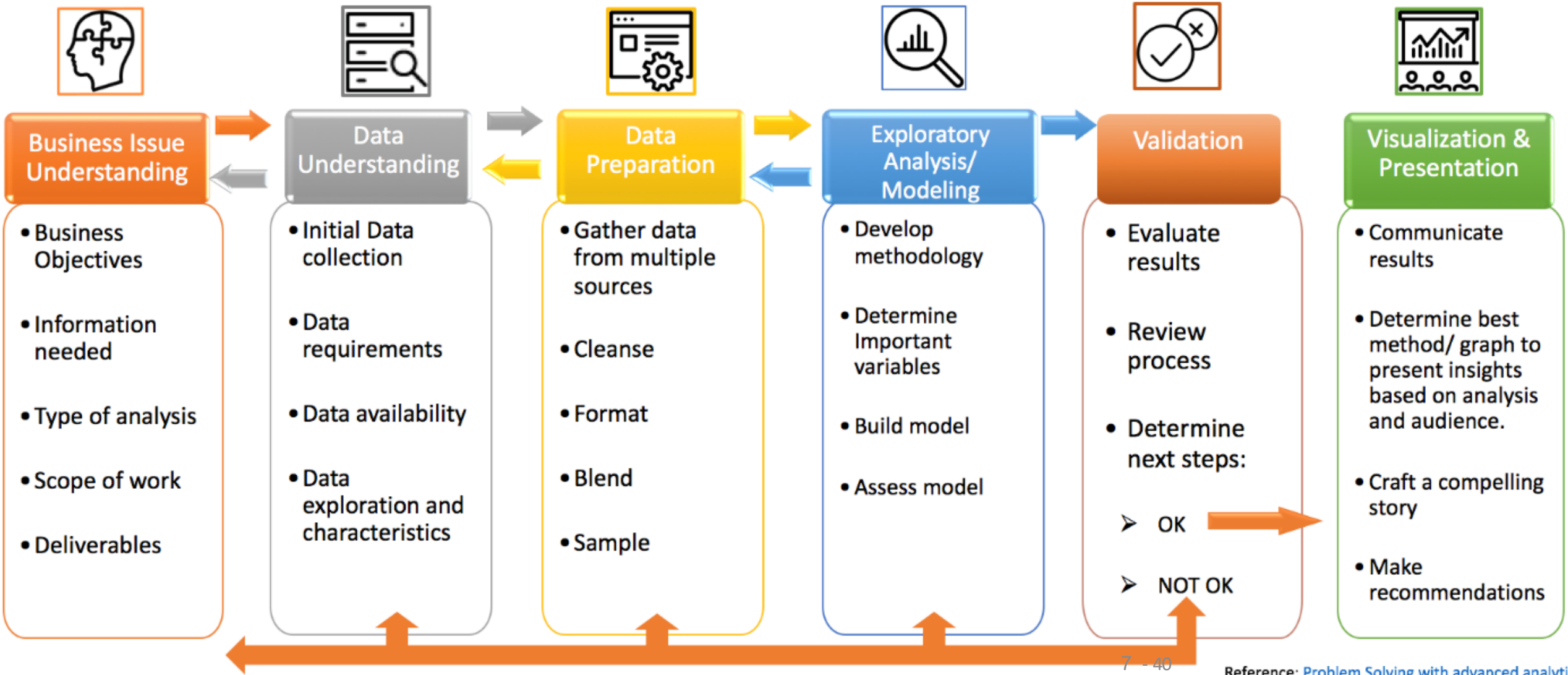
What programming language would you recommend an aspiring data scientist to learn first?



Note: Data are from the 2018 Kaggle ML and Data Science Survey. You can learn more about the study here: http://www.kaggle.com/kaggle/kaggle-survey-2018.
A total of 23859 respondents completed the survey; the percentages in the graph are based on a total of 18788 respondents who provided an answer to this question.

BUSINESS BROADWAY
DATA SCIENCE | CUSTOMER ANALYTICS | MACHINE LEARNING

Life Cycle of a data analysis project based on CRISP- DM methodology



Reference: Problem Solving with advanced analyti

Reference: https://en.wikipedia.org/wiki/Cross-industry_standard_process_for_data_minir

Basic usage of Jupyter Notebook

Python Editors/IDEs

General Editors and IDEs with Python Support

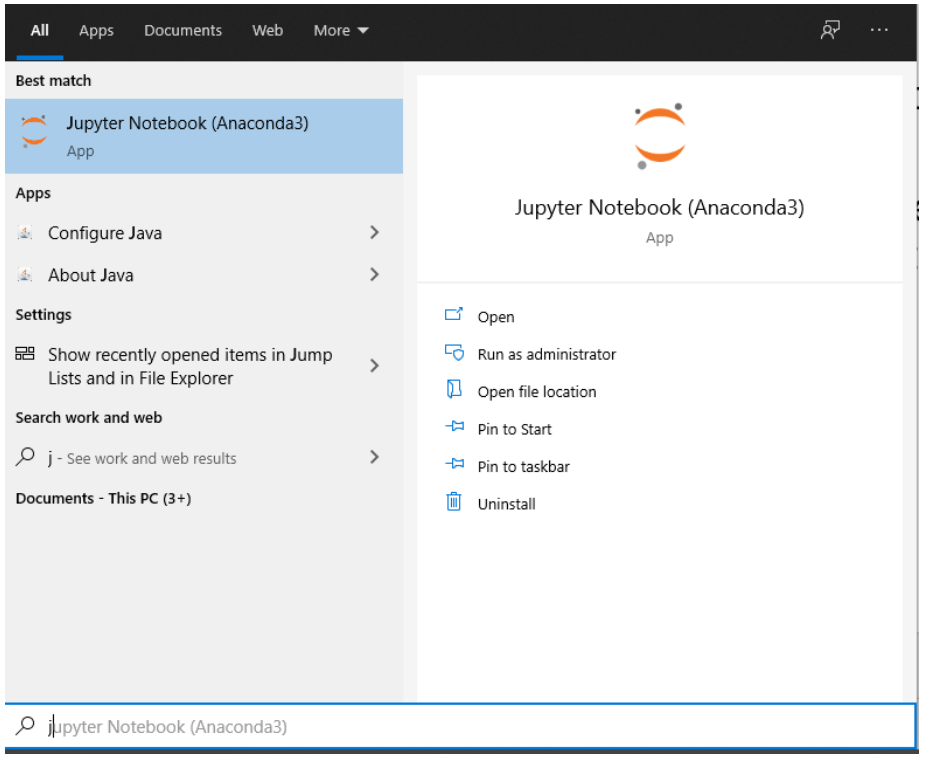
- Eclipse + PyDev
- Atom
- Vi / Vim
- Visual Studio Code

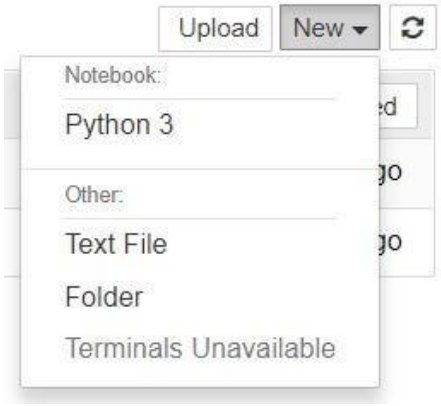
Python-Specific Editors and IDEs

- PyCharm
- Spyder
- IPython/Jupyter

Starting Jupyter

- .ipynb file is notebook file.
- A **kernel** is a "computational engine" that executes the code contained in a notebook document.
- A **cell** is a container for text to be displayed in the notebook or code to be executed by the notebook's kernel.
-





jupyter Workshop (unsaved changes)

Logout

File Edit View Insert Cell Kernel Widgets Help Trusted Python 3

Run Code

In [5]:
```
print('Hello World!')
```

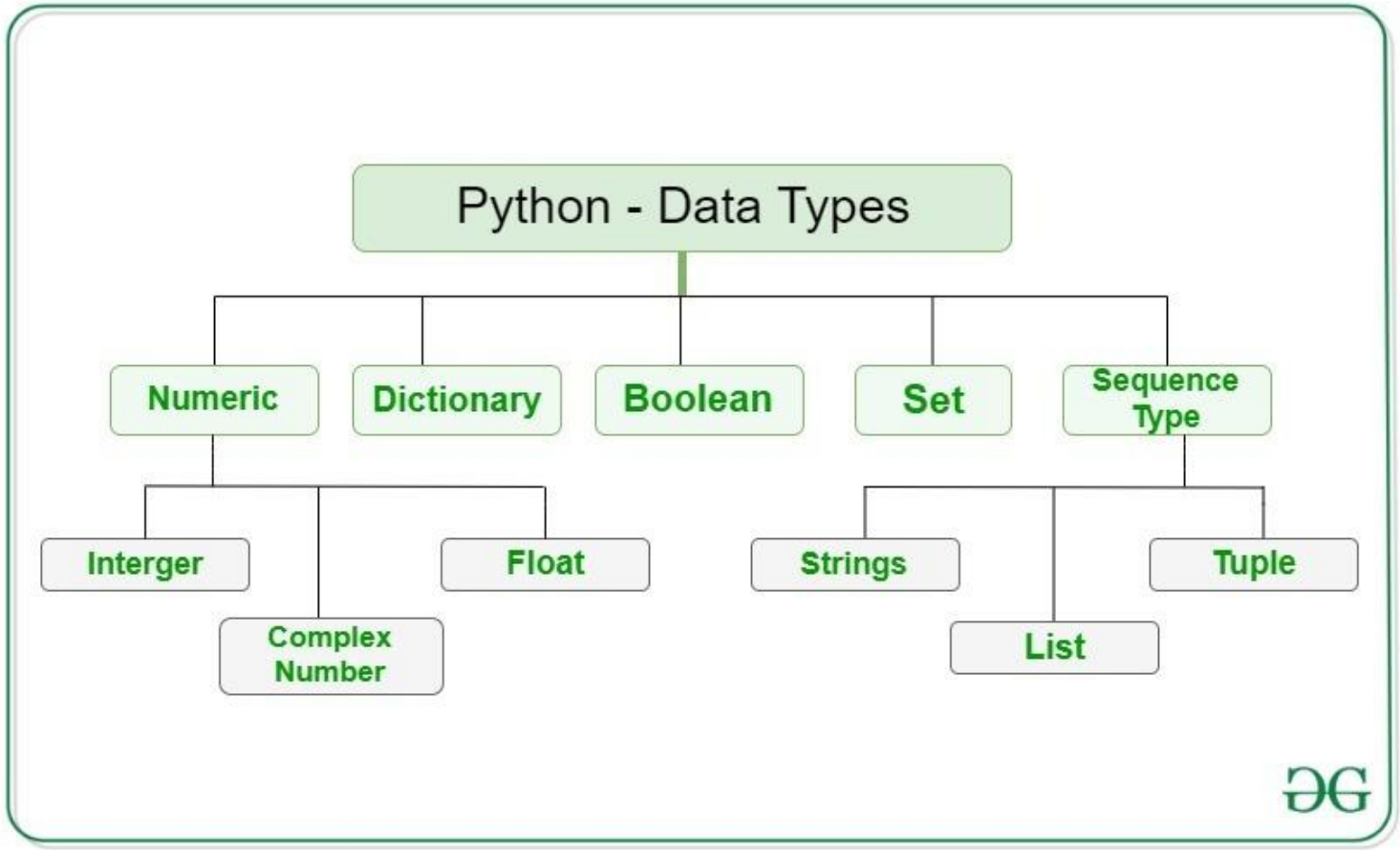
```
Hello World!
```

In [11]:
```
import numpy as np
x = np.random.randint(1, 10)
print (x)
```

```
1
```

Basic Python Concepts & Syntax

Basic Data types



Int, float, Boolean

In [ ]:

```
1  a = 5 #int
2  print(a, "is of type", type(a))
3
4  a = 2.0 #float
5  print(a, "is of type", type(a))
6
7  a= True
8  print(a, "is of type", type(a))
9
```

String

In [ ]:

```
# Strings immutable
s= "Hello World"
print (type(s))

print("s[4] = ", s[4])

print("s[6:11] = ", s[6:11])

s2 = s.replace("World", "universe")
print(s)
print(s2)

# Strings are immutable in Python
s[5] ='d'

```

List, Tuple, Set, Dict

List
General purpose
Most widely used data structure
Grow and shrink size as needed
Sequence type
Sortable

Tuple
Immutable (can't add/change)
Useful for fixed data
Faster than Lists
Sequence type

Set
Store non-duplicate items
Very fast access vs Lists
Math Set ops (union, intersect)
Unordered

Dict
Key/Value pairs
Associative array, like Java HashMap
Unordered

List, Tuple, Set, Dict

Data Structure	Ordered	Mutable	Constructor	Example
List	Yes	Yes	`[ ]` or `list()`	`[5.7, 4, 'yes', 5.7]`
Tuple	Yes	No	`( )` or `tuple()`	`(5.7, 4, 'yes', 5.7)`
Set	No	Yes	`{}`* or `set()`	`{5.7, 4, 'yes'}`
Dictionary	No	Yes**	`{ }` or `dict()`	`{'Jun': 75, 'Jul': 89}`

List

In [ ]:

```
#list (mutable)
a = [5,10,15,20,25,30,35,40]

print("a[2] = ", a[2])

print("a[0:3] = ", a[0:3])

print("a[5:] = ", a[5:])

a[2]= 1500
print(a)

a.append(50)
print(a)

a.pop(3)
print(a)

len(a)
```

Tuple

In [ ]:

```
# Tuple immutable
a = (1, 2.2, 'name')
print (a)

print("a[1] = ", a[1])

print("a[0:3] = ", a[0:3])

# Tuples are immutable
t[0] = 150

```

Set

```
1  #Set unordered & unique
2  a = {5,2,3,1,4}
3
4  print("a = ", a)
5
6  print(type(a))
7
8  a.add(4)
9  a.add(5)
10
11 for x in a:
12   print(x)
13
14 print (a[1])
```

Dict

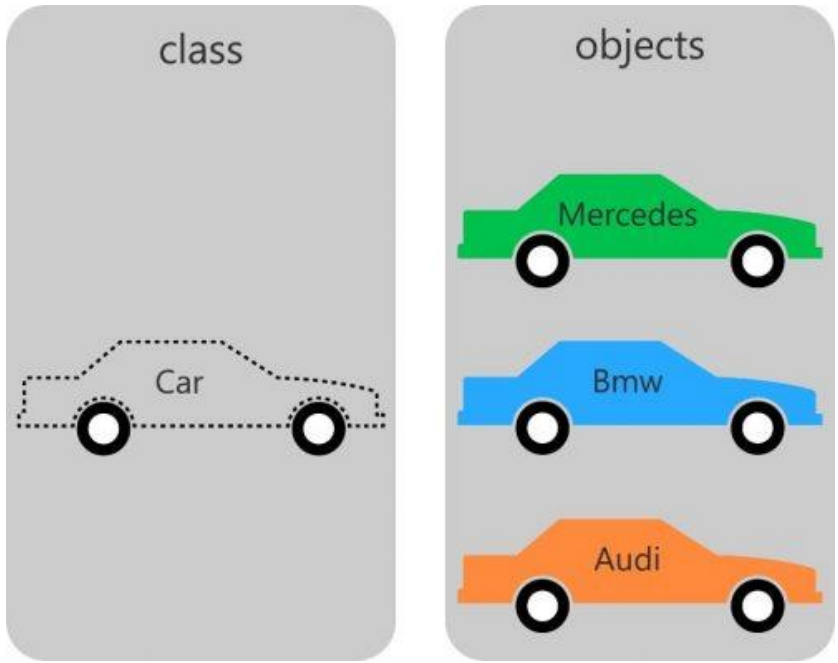
```python
# Dictionary key:value

# empty dictionary
my_dict = {}

# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}

# get vs [] for retrieving elements
my_dict = {'name': 'Jack', 'age': 26}

# Output: Jack
print(my_dict['name'])

# Output: 26
print(my_dict.get('age'))

# Trying to access keys which doesn't exist throws error
# Output None
print(my_dict.get('address'))

# update value
my_dict['age'] = 27

#Output: {'age': 27, 'name': 'Jack'}
print(my_dict)

# add item
my_dict['address'] = 'Downtown'

# Output: {'address': 'Downtown', 'age': 27, 'name': 'Jack'}
print(my_dict)

```

Import package, comment, Indentation

In [ ]:

```
# import package, comment, Identation
import numpy as np
x = np.random.randint(1, 10)
print (x)

if x > 5:
    print ("x > 5")
elif x < 5:
    print ("x < 5")
else:
    print ("x = 5")


```

Class, Object, Function & Method



Function	vs.	Method
can have many parameters		the object is one of its parameters
exists on its own		belongs to a certain class
function()		object.method()

Class, Object, Function & Method

```python
#Class, Object, Function & Method
class Car: # class
    def __init__ (self,car_type,avg_speed): # constructor
        self.type = car_type
        self.avg_speed = avg_speed
        self.kilometrage = 0

    def move(self, duration_h): # Method
        self.kilometrage += (self.avg_speed*duration_h)
        return self.kilometrage

    def __str__(self):
        return self.type + " "  + str(self.kilometrage)
    def __repr__(self):
        return self.type + " "  + str(self.kilometrage)


def add(x,y): # Function
    return x+y

ford_car = Car('Ford',100) # creat object
print (ford_car.type)

ford_kilos = ford_car.move(100)
print (ford_kilos)

honda_car = Car('Honda',200)
print (honda_car.type)

honda_kilos = honda_car.move(20)
print(honda_kilos)

total_kilos = add(ford_kilos,honda_kilos)
print(total_kilos)
```

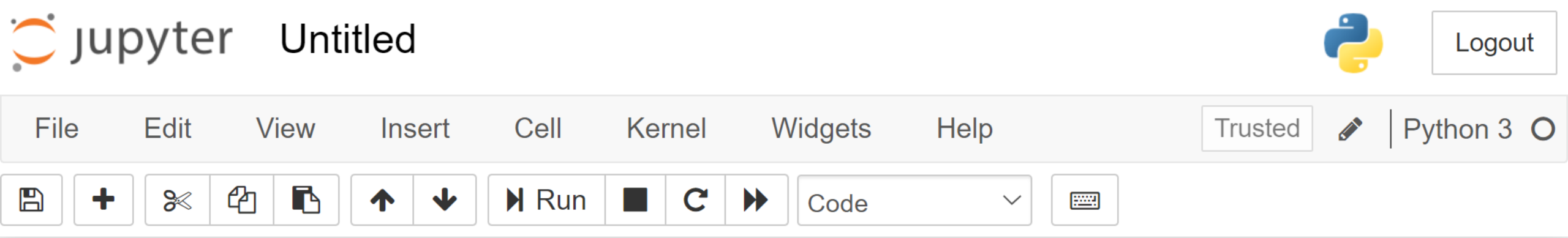
Essential Python Libraries

Important Libraries

- Basic Libraries
  - Numpy (mathematical functions)
  - Pandas (data analysis)
  - Scipy (on top of Numpy)
  - Statsmodels (statistical models)
- Visualizations
  - Matplotlib
  - Seaborn (beautiful visualizations on top of Matplotlib)
- Machine Learning
  - Scikit-learn
  - Pytorch (by Facebook)
  - Tensorflow (by Google)
  - Keras (interface for Tensorflow)

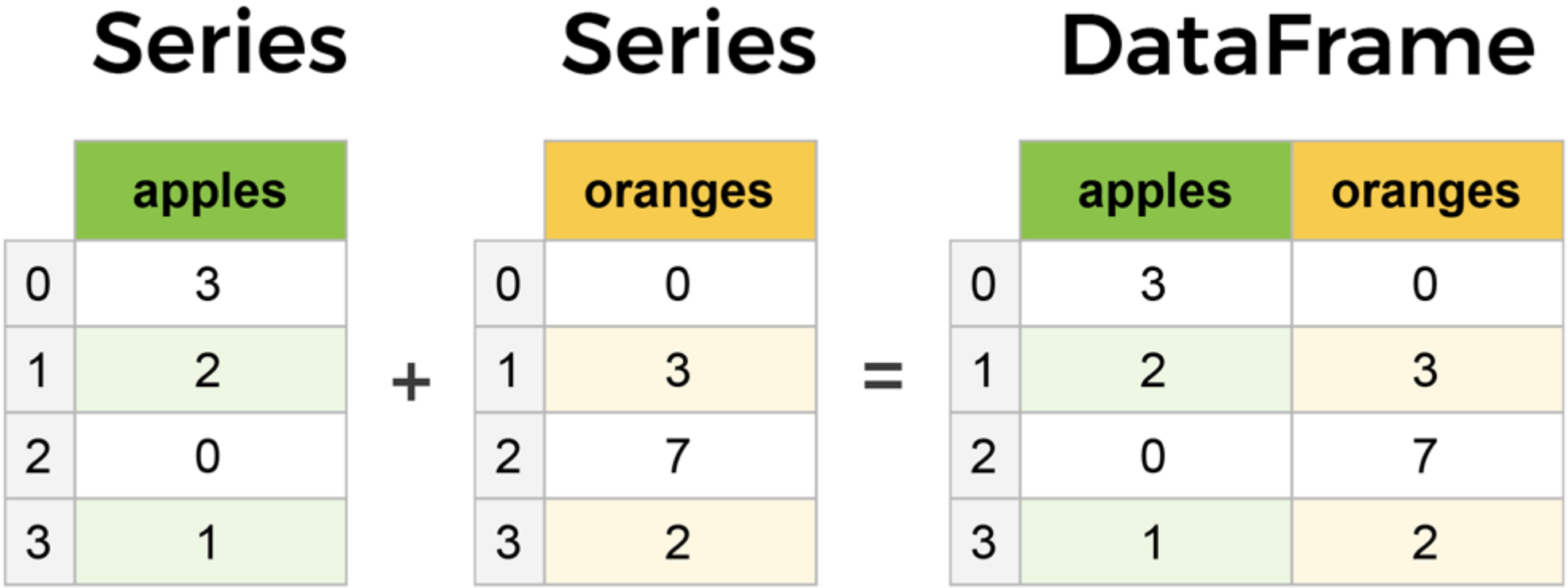
Installing Packages

- conda install -c anaconda seaborn
- pip install seaborn


jupyter
Untitled
Logout
File
Edit
View
Insert
Cell
Kernel
Widgets
Help
Trusted
Python 3
Run
Code
In [1]:
import pandas as pd
import numpy as np
In [ ]:

Data Loading, Storage, and File Formats

Series & DataFrame



Creating DataFrame

In [2]:
```
import pandas as pd

list_a = [[1,2,3,4,5],
          [6,7,8,9,10]]
columns = ['a','b','c','d','e']
df_a = pd.DataFrame(list_a, columns=columns)
df_a.head()
print(df_a)
```

```
   a  b  c  d   e
0  1  2  3  4   5
1  6  7  8  9  10
```

Loading Files and basic DF info

In [36]:
```
import pandas as pd
import numpy as np

url_data = 'https://raw.githubusercontent.com/owid/covid-19-data/master/public/data/owid-covid-data.csv'
df = pd.read_csv(url_data, low_memory=False)

print(df.shape)
print(df.columns)
print(df.describe())
print (df.info())
```

Sub DataFrame based on columns

In [51]:

```
df_sub_col= df[['location','total_cases','total_deaths','date'] ]
print(df_sub_col.shape)
print (df_sub_col.head(10))
print (df_sub_col.tail(10))
```

DataFrame basic operations

In [ ]:
```
print(df[['total_deaths_per_million','hospital_beds_per_thousand', 'human_development_index']].corr())

print(df['location'].value_counts().head(10))

print(df['new_cases'].sum()/2)
```

Data Cleaning and Preparation with Pandas

DataFrame slicing

```
fd_eg = df_sub_col.loc[15180]
print (fd_eg)


fd_i = df_sub_col.iloc[15180]
print (fd_i)
fd_i = df_sub_col.iloc[15180:15185]
print (fd_i)
```

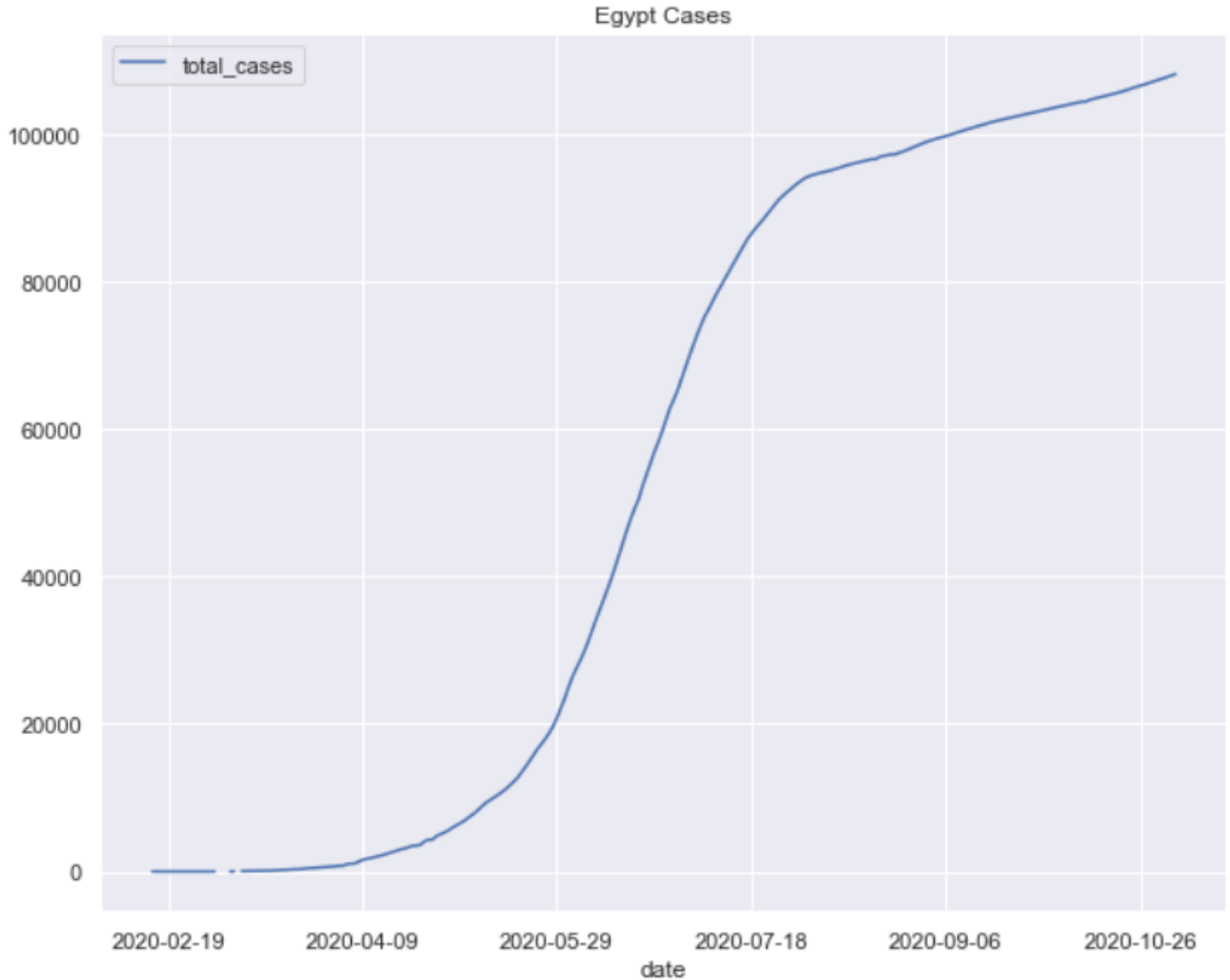
DataFrame Conditional selections

```
condition = (df_sub_col['date'] == "2020-11-04")
print(condition)

df_last = df_sub_col[df_sub_col['date'] == "2020-11-04"]
print(df_last.shape)
print(df_last.head(10))

df_last = df_sub_col[ (df_sub_col['date'] == "2020-11-04" ) & (df_sub_col['total_deaths']>10000)]
print(df_last.shape)
print(df_last.head(10))

df_last = df_sub_col[ (df_sub_col['date'] == "2020-11-04" ) & (df_sub_col['location'].isin (['Egypt','Japan']))]

print(df_last.shape)
print(df_last.head(10))
```

Plotting and Visualization

In [72]:
```
import matplotlib.pyplot as plt
plt.rcParams.update({'font.size': 20, 'figure.figsize': (10, 8)}) # set font and plot size to be larger
fd_eg =  df_sub_col[df_sub_col['location'] == "Egypt"]
fd_eg.plot(kind='line',x='date',y='total_cases',  title='Egypt Cases')
fd_eg.plot(kind='line',x='date',y='total_deaths',  title='Egypt deaths')
```

Out[72]: <matplotlib.axes._subplots.AxesSubplot at 0x2618d8930d0>



In [75]:
```
plt.plot( 'date', 'total_cases', data=fd_eg, marker='o', color='skyblue', linewidth=4)
plt.plot( 'date', 'total_deaths', data=fd_eg, marker='', color='olive', linewidth=2)
plt.legend()
```

Out[75]: <matplotlib.legend.Legend at 0x2618a6eb820>