Python

Download: python.org/downloads -> download Python 3

Download: jetbrains.com/pycharm/ -> download communityversion of pycharm

String – stores plain text, which needs parentheses

Number – duh

Boolean Value –TRUE/FALSE

Import existing codes:

From math import * -> to allow usage of a few more maths functions

Commandlist:

Basic operations:

Print(“”) – prints what is between the parentheses onto the console

!ls – shows what files are in the current working directory

name_var = ”name” – creates a macro which writes what is inbetween the parentheses, whenever you write the following into your code: -> print (“My name is” + name_var +” and I am 12”)

\n inserts a new line inside a string

\ can be used as an “escape character” and tells python to use the next character literally

or ‘’ starts a comment so that you can write notes in your code that are not part of your code

.lower – behind a string converts the entire string to lower case

.upper - behind a string converts the entire string to upper case

len(“string”) – will give you the amount of characters in a string

string[0] – gives you the first character inside the string

string[1] - gives you the second character inside the string

PhraseA.index(“string”) – gives you the position where “string” is in PhraseA

.replace(“StringA”, “StringB”) – replaces StringA with StringB

str(number) – converts a number into a string

abs(number) – absolute value of a number

pow(3, 2) – takes 3 to the power of 2

max(4,6) – takes the max of 4 and 6

min(4,6) – takes the min of 4 and 6

round(3.7) – rounds the number to the nearest integer

sqrt(number) – squareroots the number in the brackets

Inputs

input() – python will allow the user to input some information

example: name = input("enter your name: ")

print ("Hello" +name + "!")

- ➔ Python will get input from the user and save it under the variable "name", so whenever you use "name" in the program it will use whatever the user put in

num1 = input("enter a number: ")

num2 = input("enter another number: ")

result = float(num1+num2) -> converts the string from the input into number

print(result)

Lists

To store a bunch of values (strings, numbers, bullions) into a variable

Range(a,b,c) <- creates a list with the following characteristics:

- A = endpoint
- B = startpoint <- zero if empty
- C = in steps of "c"

friends = ["Kevin", "Karen", "Jim", "Hans", "Mike"]

lucky_numbers = [4, 8, 15, 16, 23, 42]

- ➔ print(friends[1]) will output position 1 (Karen)
- ➔ print(friends[-1]) will output the last element (Mike)
- ➔ print(friends[2:]) will output everything after position 2
- ➔ print(friends[1:3]) will output the range from positions 1-3

friends.extend(lucky_numbers) -> adds the lucky_numbers list to friends list

friends.append("James") -> adds James to the friends list to the end

friends.insert(1, "James") -> adds James to Position 1 to the list

friends.remove("James") -> removes James from Friends

friends.pop() -> removes the last item from the list

friends.index("James") -> finds James in the list and gives you the position

friends.count("James") -> counts the number of "James" in the list

friends.sort()-> puts the list in alphabetic/ascending order

friends.reverse() -> puts the list in reverse order

list2 = friends.copy() -> makes list2 a copy of friends

Tuples

A type of data structure we can store information in (similar to a list)

Tuple is inmutable -> once it is created it can't be changed

This is a tuple: Coordinates = (7,3)

To normalize the data before feature selection:

Z

Min-max

Functions:

You can take a couple lines of code into a function which performs a task and whenever you want to perform the task, then you just need to write the function and not all lines of codes again

def sayhi(): -> name your function

print ("Hello User") -> anything in a function needs to be indented

sayhi() -> to execute the function just write its name

you can also specify parameters inside the parentheses

Example: def say_hi(name, age):

Print("Hello,"+name+ ",you are" +age)

Say_hi("Mike", "30")

➔ the output will be "Hello, Mike, you are 30"

Return Statement:

Allows python to return information from a function

Example: def cube(num):

return num*num*num

print(cube(3)) -> this will give us 27, without the “return” it wouldn’t work

➔ whenever pythons sees a return statement, it is the end of the function, so afterwards there cant be anything added inside the function

If Statements:

is_male = False -> create a bullion variable

is_male = False

if is_male *and* is_tall:

    print(“you are a tall male”)

elif is_male *and not* is_tall:

    print(“you are a short male”)

elif not(is_male) *and* is_tall:

    print(“you are a tall woman”)

else:

    print(“you are a short woman”)

If Statements and Comparisons:

Compare 3 different inputvalues of a defined variable

def max_num(num1, num2, num3):

    If num1>=num2 *and* num1>=num3:

        Return num1

    elif num2<= num1 and num2 >=num3:

        return num2

    else:

```
        return num3

print(max_num(3,4,5))
```

other comparison operators

== - equal

!= - not equal

>= bigger than

<= smaller than

How to build a basic calculator:

Get input from the user:

Num1 = float(input(“Enter first number: ”))

op = input(“Enter operator: ”)

Num3 = float(input(“Enter second number: ”))

Float converts the string into a number

Input is always naturally stored as string

```
If op == “+”:
        Print(num1 + num 2)
Elif op==”-“:
        Print(num1-num2)
Elif op==“*“:
        Print(num1*num2)
Elif op ==“/“
        Print(num1/num2)
Else:
        Print(„invalid operator“)
```

Using a Dictionary in Python:

Dictionaries are special structures in python that allows us to store information in specific “key value pairs”

- we need to define “keys” and store the value in it -> for example key= “Jan” and value = “January”

in order to create a dictionary, this is how you do it:

Dictionary_name = {

“jan”: “January”,

“feb”: “February”,

“mar”: ”March”

}

Print(Dictionaryname[“jan”])

Print(Dictionaryname.get[“jan”, “not a valid key”]) <- with get it can give a default function

While Loop

Allows you to make a code repeat itself a few times until a certain condition is fulfilled

- a loop can be started with “while” and then specify the condition that has to be met to repeat the code that is indented

Example:

i=1

while i<= 10: <- as long as this condition is true, we will loop through this code

print (i)

i = i+1

Building a guessing Game (with 5 tries)

```
print('Hello, Do you want to play a game?')
secret_word= "giraffe"
guess = ""
guess_count = 0
guess_limit = 5
out_of_guesses = False

while guess != secret_word and not (out_of_guesses):
  if guess_count < guess_limit:
     guess = input("guess the secret word: ")
     guess_count += 1
  else:
     out_of_guesses = True
```

```
if out_of_guesses:
    print("Out of Guesses, YOU LOSE!")
else:
    print("CORRECT!!! YOU WIN!!!")
```

For Loop

A for loop is a special type of loop in python that allows us to loop through a different array of code

```
friends = ["jim", "Karen", "Kevin"]
for friend in friends:
    print (friend) <- this will print out every of the “friends” list

for index in range(10):
    print(index)   <- this will print out numbers from 0 to 9

for letter in "PwC":
    print(letter)    <- this will print out PwC one letter per line
```

for loop over a dictionary:

```
for key, value in dictionary.items():
            print(key+ str(value))
```

for a loop over a numpy array:

```
for val in np.nditer(my_array):
            print(val)
```

for a loop over a Pandas DataFrame:

```
for lab,row in df.iterrows():
                print(lab)
                print(row[“column”])
```

Exponent Function

```
def raise_to_power(base_num, pow_num):

  result = 1
  for index in range(pow_num):
    result = result*base_num
  return result

print(raise_to_power(3,4))
```

2D Lists

```
number_grid = [
  [1,2,3],
  [4,5,6],
  [7,8,9],
  [0]
]

print(number_grid[1][1]) <- this would give you “5”
```

Nested Loops

```
number_grid = [
  [1,2,3],
  [4,5,6],
  [7,8,9],
  [0]
]
```

```
for row in number_grid:
  for col in row:
    print(col)
```

Build a basic encoder

Switch every vowel with the letter “g” and every other letter of “Daniel” with “f”

```
def translate(x):
    translation=""

    for letter in x:
      if letter.lower() in "aeiou":
          if letter.isupper():
            translation = translation +"G"
          else:
            translation = translation +"g"
      elif letter.lower() in "dnl":
          if letter.isupper():
            translation = translation + "F"
          else:
            translation = translation +"f"
      else:
          translation = translation + letter
    return translation

print(translate(input("Enter a phrase: ")))
```

Try Except

Allows your programme to try a certain code and if it is wrong it wont break down

```
try:
  number = int(input("Enter a number: "))
  print(number)
except ValueError as err:                    <- ValueError is a predefined error
  print(err)
```

Reading from external Files

How to work with a file that is outside of python, open, close, read, alter etc.

Testfile = Open("file.txt", "r") <- read only

Testfile = Open("file.txt", "w") <- write

Testfile = Open("file.txt", "a") <- append but cant change

Testfile = Open("file.txt", "r+") <- read and write

Print(testfile.readable()) <- to find out if we can read the file or not

Print(testfile.read()) <- opens the file

Print(testfile.readline()) <- opens only the first line of the file

Print(testfile.readline**s**()) <- takes every line and puts them in a list

Testfile.close()

Writing and appending to external files

Appending to a file

Testfile= open("testfile", "a")

Print(testfile.read())

Testfile.write("**\n**Value1, Value2") <-appends a new line in the file

Testfile.close()

Overwriting a file

Testfile= open("testfile", "**w**")

Print(testfile.read())

Testfile.write("**\n**Value1, Value2") <-appends a new line in the file

Testfile.close()

Modules and Pip

Module is a python file that we can import into our current file in order to access it

You can find modules here: https://docs.python.org/3/py-modindex.html

```
Import filename
```

```
Print(filename.action())
```
<- where “action” is a predefined function from “filename”

How to install PIP

1. go to windows earch bar and type in “cmd”
2. make sure you have pip on your computer “pip – version”
3. pip install python-docx <- python-docx is the PIP we wanted to install

How to uninstall PIP

1. go to windows earch bar and type in “cmd”
2. pip uninstall python-docx

Classes and Objects in Python:

Help to make programmes more organized and powerful

A class defines what an “object” should be

```
Class Student:
      Def __init__(self, name, major, gpa, is_on_probation) <- always include “self”
```

```
            Self.name = name
            Self.major = major
            Self.gpa = gpa
            Self.is_on_probation = is_on_probation

From file import Student
Student1= Student(“Jim”, “Business”, 3.1, False)
Student2= Student(“Pam”, “Art”, 3.3, True)
```

Now, if you want to access the information about the object, you can refer to the class:

`Print(student1.gpa)` <- this gives you the GPA of student 1

Class or Object Function in Python

Can either modify an object in a class or give information about an object in a class

Example:

```
      Class Student:
            Def __init__(self, name, major, gpa):
                  Self.name = name
```

```
        Self. Major = major
        Self.gpa = gpa
```

Create a Function that tells you if a student is a good student or not

```
    Def good_student(self):
        If self.gpa >=3.5
            Return True
        Else:
            Return False

    Print(student1.good_student())
```

How to build a multiple-choice quiz

1. **make a Questions "Class" in another file**

```
class Question:
  def __init__(self, prompt, answer):
    self.prompt = prompt
    self.answer = answer
```

2. **write the Quiz:**

```
from classfile import Question
question_prompts = [
  "How old is Daniel?\n(a) 21\n(b) 22\n(c) 23\n\n",
  "what is Daniels favorite Sport?\n(a) Swimming\n(b) Snowboarding\n(c) Football
\n\n",
  "What did Daniel study for his Bachelor?\n(a) Engineering\n(b) Economics\n(c)
Maths \n\n"
]

questions =[
  Question(question_prompts[0],"a"),
  Question(question_prompts[1],"c"),
  Question(question_prompts[2],"b")
]

def run_test(questions):
  score = 0
  for question in questions:
    answer = input(question.prompt)
    if answer == question.answer:
      score +=1
  if score == 3:
    print("Wow, you really know Daniel well! You got " + str(score) + "/" +
str(len(questions)) + " Questions correct!")
  elif score == 2:
    print("Room for improvement...You got " + str(score) + "/" +
str(len(questions)) + " Questions correct!")
  elif score == 1:
    print("Poor Effort, you only got " + str(score) + "/" + str(len(questions)) +
" Questions correct!")
  else:
    print("And you call yourself a friend?! You got " + str(score) + "/" +
str(len(questions)) + " Questions correct!")
run_test(questions)
```

Inheritance of Classes:

Allows you to make a class that has all the functionality of another class, but added other functionalities as well

1. make generic Class:

```
Class Chef:
	Def make_chicken(self):
		Print("The chef makes a chicken")
	Def make_salad(self):
		Print("the chef makes a salad")
	Def make_special(self):
		Print("the chef makes his special dish: BBQ")
```

2. make 2nd inheritance Class:

```
from Chef import Chef
Class ChineseChef(Chef):
	Def make_special(self):
		Print("the chef makes his special dish: Sushi") <- override "Chef class"
	Def make_rice(self):
		Print("the chef makes rice")
```

3. refer to the classes in your code

```
From Chef import Chef
From ChineseChef import ChineseChef
	myChef = Chef()
	myChef.make_special()		<- will make BBQ
	myChineseChef = ChineseChef()
	myChineseChef.make_special()	<- will make Sushi
```

Matplotlib

```
From matplotlib import pyplot as plt
```

Plt.scatter(x,y)

Plt.xscale("log") <- transforms the x-axis to logarithmic scale

Plt.hist(x,y) <- plots a histogram

Plt.show() <- shows the actual plot

Plt.clf() <- cleans up the plot so that you can restart again

Plt.xlabel(“title) <- x axis lavel

Plt.ylabel(“title) <- y axis lavel

Plt. Title(“title”) <- title of the plot

Plt.yticks([1,2,3,4,5,6]) <- change the intervals of the y axis

plt.scatter(x = gdp_cap, y = life_exp, s = np.array(pop) * 2, c = col, alpha = 0.8)

- s= “var” <- changes the size of the scatter depending on “var”
- c=col <- changes the colors of the plot
- alpha = 0-1 <- changes the opacity of the plot

plt.legend([“y”, “z”]) <- takes a list as an argument and labels the plot in the order in which you plt.plot()

Dictionaries

The Dictionary is like a list, but the index is a “key” instead of numbers

Dictionary[“key”] <- gives you the value to the respective key

Dictionary.index(“key”) <- gives you the index of the respective key

Dictionary.keys() <- prints our the keys of your dictionary

Del(dictionary[“key”]) <- deletes a key from your dictionary

Print(“keyvalue” in “dictionary”) <- prints out the result from one keyvalue

print(europe["france"]["capital"]) <- prints out the result from one keyvalue

Pandas

Import pandas as pd

Read_csv(“path/to/dataset.csv”, index_col = 0)

- opens csv file
- puts an index on the columns

my_array[rows, columns] <- slices the array according to the rows and columns

my_array.loc[[“label”, “label2”, “label3”], [“column1”, “column2”] <- label-based

- use a double dot [:] to select all

my.array.iloc <- integer position based

- results are the same as in loc, but instead of “label” just put the index number in the square brackets

my_array[“column”] > X <- gets a Boolean

my_array[is_X] <- splits the dataframe to all the values that are True in the previous Boolean

print(my_array>=X) <- gets a Boolean for all values who fullfil the condition

my_array[my_array[“column”]>=X] <- slices the array and leaves all the values who fullfil the condition

my_array["new_column"] = my_array["existing_column"] > X <- makes new_column based on a condition

df[“new_column”] = df[“column”].apply(len) <- adds a new_column with the length of the characters in another column (instead of len, other commands are also possible

example: cars["COUNTRY"] = cars["country"].apply(str.upper)

df.iloc[first_row : last_row , first_column : last_column] <- slice a dataframe

df.info() <- find out how many NaN’s there are

Numpy

np.logical_and(array>x , array < y) <- gets a Boolean for all values who fullfil the and condition

- same thing works with *np.logical_or()* and *np.logical_not()*

Random Numbers

Inside the numpy package there is a random package

Np.random.rand()

Np.random.randint(start,end) <- generates a random integer between interval

Iterating over a Pandas Dataframe with a function

```
# Define count_entries()
def count_entries(df, col_name):
    """Return a dictionary with counts of occurrences as value for each key."""
```

```
    # Initialize an empty dictionary: langs_count
    langs_count = {}
    # Extract column from DataFrame: col
    col = df[col_name]

    # Iterate over lang column in DataFrame
    for entry in col:

        # If the language is in langs_count, add 1
        if entry in langs_count.keys():
            langs_count[entry] +=1
        # Else add the language to langs_count, set the value to 1
        else:
            langs_count[entry] = 1

    # Return the langs_count dictionary
    return(langs_count)

# Call count_entries(): result
result = count_entries(tweets_df, "lang")

# Print the result
print(result)
```

Dice Rolling Game with Empire State Building

```
# numpy and matplotlib imported, seed set

# Simulate random walk 250 times
all_walks = []
for i in range(10) :
    random_walk = [0]
    for x in range(100) :
        step = random_walk[-1]
        dice = np.random.randint(1,7)
        if dice <= 2:
            step = max(0, step - 1)
        elif dice <= 5:
            step = step + 1
        else:
            step = step + np.random.randint(1,7)

        # Implement clumsiness
        if ___ :
            step = 0
```

```
      random_walk.append(step)
   all_walks.append(random_walk)

# Create and plot np_aw_t
np_aw_t = np.transpose(np.array(all_walks))
plt.plot(np_aw_t)
plt.show()
```

Local, Global, etc.

Default and flexible arguments

```
Def add_all(*args):
        Sum_all = 0
        For num in args:
                Sum_all += num
        Return sum_all
```

* <- before an argument allows us to pass as many arguments in the variable as we want
** <- before and argument allows us to specify any amount of keys in the formula

For index, value in enumerate(list, start=5):
    Print(index, value)
<- this function unpacks the list and prints them with an index that starts at 5 with the first value

Zip() <- takes an arbitrary number of iterables and returns an iterable number of tuples

Data Visualisation

Import matplotlib as plt
Plt.plot() <- can plot numpy arrays and lists
Plt.show() <- shows the plot
Close_series.plot() <- when you have a timeseries on the x-axis then this is nicer, with titles etc.
df.plot() <- plots all columns in a dataframe on one diagram
plt.yscale("log") <- makes the scale logarithmic
df.plot(kind="scatter") <- changes the plot to a scatter plot
df.plot(kind="hist")
df.plot(kind="area")

other options for "hist":

- bins(int) <- number of intervals or bins
- range(tuple) <- extremes of bins (min to max)
- normed(boolean) <- normalize or not
- cumulative(Boolean) <- compute cdf
- alpha = () <- specify the transparency of the histograms, which is useful when plotting overlapping things

df.plot(kind="box") <- plots a boxplot
df.plot(ax=axes[0]) <- plots the graph on the highest position, in case you plot multiple graphs

Indexing timeseries

Read_csv(parse_dates=True)
ISO 8601 = yyyy-mm-dd hh:mm:ss <- the most common format

Pd.to_datetime(df, format = "%Y-%m-%d %H:%M") <- set a datetime pd
Pd.Series(df, index=my_datetime) <- convert a df into a series

df.loc['2010-10-11 21:00:00':'2010-10-11 22:00:00'] <- splitting a df by time index

df["column"]["Date-time"] <- splitting df by time and column

df = df.rolling(window=24).mean() <- applying a rolling mean with a 24-hour window

daily_highs = august.resample('D').max() <- resampling to daily data

Filter Timeseries

from datetime import datetime

df = df[df['Date'].dt.month == 11] <- month

- `dt.quarter`
- `dt.week`
- `dt.weekday`
- `dt.day_name`
- `dt.is_month_end`
- `dt.is_month_start`
- `dt.is_year_end`
- `dt.is_year_start`

Manipulating time series data

Df[“column].str.contains(“substring”) <- searching to find all of the rows that contain a substring

Df[“column].str.contains(“substring”).sum() <- counts all the rows that contain certain substring

Df[“date_time”].dt.hour <- changes 23:00 to 11pm or vice versa

Df[“date_time”].dt.tz_localize(“US/Central”) <- changes the timezone to US central

Dt.tz.convert(“US/Eastern”) <- transforms the times and puts it into a new column

Column.resample(‘A’).first() <- gets the first value of every year and puts every value in between to NaN
Column.resample(‘D’).first() <- gets the first value of every day and puts every value in between to NaN

Column.resample(‘D’).first().interpolate(“linear”) <- replaces the NaN linearly inbetween the first values of each day

```
# Build a Boolean mask to filter for the 'LAX' departure flights: mask
mask = df['Destination Airport'] == "LAX"

# Use the mask to subset the data: la
la = df[mask]
```

working with numeric Columns

df['column'] = pd.tonumeric(df ['column], errors='coerce') <- convert all the value in a column to numeric

df = df.drop('column_name', 1) <- to drop a column from a df

listname = df['column'].tolist() <- #Convert the dataframe column into a list

listname = str(listname) <- converts all elements in the list to string

tokenized = word_tokenize(tweet_eng) <- #Convert list of tweets into list of all words within the tweets

print(df.isnull().any()) <- check if there is any missing entries in the dataframe

print(df.isnull().sum()/len(df)) <- checks the % of null values in each column

print(df.column.value_counts(normalize=True)) <- % of values in each column (like pie-chart but in table)

print(df.isnull().sum) <- counts the number of null values in each column

df = df.drop('Column', axis =1) <- drops the column that you want to get rid off

df=df.dropna() <- drops all columns that contain null values

```
# Drop rows in df with how='any' and print the shape
df.dropna(how="any")

train['production_companies'] = train['production_companies'].str.split(',').str[0]
train['production_companies'] = train['production_companies'].str.split(':').str[1]

visitors_pivot = users.pivot(index ="weekday", columns="city", values="visitors")
```

stacking and unstacking dataframes

```
# Stack df by 'column' and print it
df.stack("column")
```

3 ways to merge dataframes

```
merged_df= pd.merge(df1, df2, on="column")
combined = pd.merge(df1,df2,left_on="column1", right_on= "column2")
combined = pd.merge(df1, df2, on=["branch_id", "city", "state"])
```

Use a pivot table to display the count of each column by category/index

```
count_by_column= users.pivot_table(aggfunc="count", index="weekday")
```

map a new column to the df

```
# Create the dictionary: red_vs_blue
red_vs_blue = {"Obama":"blue", "Romney":"red"}
# Use the dictionary to map the 'winner' column to the new column: election['color']
election['color'] = election["winner"].map(red_vs_blue)
```

fill column based on condition

```
# Create the boolean array: too_close
too_close = election["margin"]<1
# Assign np.nan to the 'winner' column where the results were too close to call
election["winner"].loc[too_close] = np.nan
```

how to iterate through a column to make a list based on conditions

```
List = [ ]

for element in df["OLSAT Verbal Score"]:
   if type(element)==type("a"):
      if  "-" in element or "N" in element or "'-'" in element:
         element=99
```

```
    else:
        element=float(element[:2])
  listmean.append(element)
```

df.loc[df.column > df.column.quantile(0.95), 'column] = df.column.quantile(0.95) <- to eliminate outliers

df.loc[df.column.isnull(), 'column'] = df.column.median() <- convert missing values to median or mean

transform categorical data into numbers

```
class_categories = {'First':1, 'Second':2, 'Third':3}
df['class'] = df['class'].apply(lambda x: class_categories[x])
```

transform binary categories into dummy

```
columns_binary = ['sex', 'alive', 'over 20']
titanic = pd.get_dummies(df, drop_first=True, columns=columns_binary)
```

```
f['Sex'] = df["Sex"].str.get_dummies()
```

transform categorical data into multiple dummies columns

```
columns_binary = ['origin', 'generation']
titanic = pd.get_dummies(titanic, columns=columns_binary)
```

Transform dummies into binary columns and categoricals into multiple dummies

```
for column in data.select_dtypes('object'):
  if len(data[column].unique()) == 2:
    data[column] = pd.get_dummies(data[column], dtype='int64')
  else:
    data = pd.get_dummies(data, prefix=column, columns=[column], drop_first=True, dtype='int64')
```

Fill Nan with a Function

```
def fill_nan(data, method='mean'):
  for column in data.select_dtypes(['int64', 'float64']):
    if method == 'mean':
      data[column] = data[column].fillna(data[column].mean())
    elif method == 'median':
      data[column] = data[column].fillna(data[column].median())
  return data
```

Get rid of outliers

```
df = pd.DataFrame(np.random.randn(100, 3))

from scipy import stats
df[(np.abs(stats.zscore(df)) < 3).all(axis=1)]
```

Replacing and Entry with another value

```
df.replace({'?': np.nan}, inplace = True)
```

SEABORN for Plotting

Matrix of plots

```
sns.set(style="ticks")
sns.pairplot(df1, hue="category")
```

PANDAS DATASET DESCRIPTOR

```
import pandas_profiling
pandas_profiling.ProfileReport(df)

print(df["Column"].value_counts(dropna=False)) <- count all the entries per category in a column
```

Piechart

```
df["Column"].value_counts().plot('pie')

#we program colors
colors =
['#3AA2F3','#A93226','#884EA0','#5DADE2','#17A589','#27AE60','#7DCEA0','#F1C40F','#F39C12','#E67E22','#D35400']
```

Histogram with Kernel Density

```
sns.distplot(df['Overall Score'])
plt.show()
```

Boxplot

```
sns.boxplot(data=df, y="column")
```

Boxplot with multiple categories

```
sns.boxplot(data=df,x="District" ,y="Overall Score")
```

Correlation Plots

```
# Create scatterplot of dataframe
sns.lmplot('OLSAT Verbal Score',
        'Overall Score',
        data=df,
        fit_reg=False,
        scatter_kws={"marker": "D",
                "s": 90})
# Set title
plt.title('Correlation between verbal score and Score ')

# Set x-axis label
plt.xlabel('Verbal')

# Set y-axis label
plt.ylabel('Score')

# Show graph
plt.show()
```

Webscraping with BeautifulSoup

```
import requests
from bs4 import BeautifulSoup

r = packages.get(url)                <- Package the request, send the request and catch the response: r

html_doc = r.text                    <- Extracts the response as html: html_doc

soup = BeautifulSoup(html_doc) <- # Create a BeautifulSoup object from the HTML: soup

pretty_soup = soup.prettify()    <- # Prettify the BeautifulSoup object: pretty_soup

guido_title = soup.title             <- # Get the title of Guido's webpage: guido_title

guido_text = soup.get_text()     <- # Get Guido's text: guido_text

a_tags = soup.find_all("a")      <- # Find all 'a' tags (which define hyperlinks): a_tags

for link in a_tags:
   print(link.get('href'))           <- # Print the URLs to the shell
```

Twitter Webscraping with Tweepy

```
# Import package
import tweepy

# Store OAuth authentication credentials in relevant variables
access_token = "1092294848-aHN7DcRP9B4VMTQIhwqOYiB14YkW92fFO8k8EPy"
access_token_secret = "X4dHmhPfaksHcQ7SCbmZa2oYBBVSD2g8uIHXsp5CTaksx"
consumer_key = "nZ6EA0FxZ293SxGNg8g8aP0HM"
consumer_secret = "fJGEodwe3KiKUnsYJC3VRndj7jevVvXbK2D5EiJ2nehafRgA6i"

# Pass OAuth details to tweepy's OAuth handler
auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_token_secret)

# Initialize Stream listener
l = MyStreamListener()

# Create your Stream object with authentication
stream = tweepy.Stream(auth, l)

# Filter Twitter Streams to capture data by the keywords:
stream.filter(track=['clinton', 'trump', 'sanders', 'cruz'])

# String of path to file: tweets_data_path
tweets_data_path = 'tweets.txt'

# Initialize empty list to store tweets: tweets_data
tweets_data = []

# Open connection to file
```

```
tweets_file = open(tweets_data_path, "r")

# Read in tweets and store in list: tweets_data
for line in tweets_file:
    tweet = json.loads(line)
    tweets_data.append(tweet)

# Close connection to file
tweets_file.close()

# Print the keys of the first tweet dict
print(tweets_data[0].keys())
```

Data Exploration

Load the dataset with numpy or Pandas

Pd.read_csv("filename.csv", sep=",", usecols=[1:3], header=True, skiprows = 1)

Peak at the data

Data.head()

Get the datatypes

Data.info()

Check duplicate values

```
duplicates = df.duplicated(subset=column_names)
df.drop_duplicates()
```

Get the %-age & number of NaN for each column

```
print(df.isnull().sum()/len(df))
print(df.isnull().sum) <- counts the number of null values in each column
```

Fill NaN with something else

```
df.replace({'?': np.nan}, inplace = True)
```

Fill Nan with a Function

```
def fill_nan(data, method='mean'):
    for column in data.select_dtypes(['int64', 'float64']):
        if method == 'mean':
            data[column] = data[column].fillna(data[column].mean())
        elif method == 'median':
            data[column] = data[column].fillna(data[column].median())
    return data
```

Transform dummies into binary columns and categoricals into multiple dummies

```
for column in data.select_dtypes('object'):
    if len(data[column].unique()) == 2:
        data[column] = pd.get_dummies(data[column], dtype='int64')
    else:
        data = pd.get_dummies(data, prefix=column, columns=[column], drop_first=True, dtype='int64')
```

find categorical columns

```
for column in data.select_dtypes('object'):
   print(f'{column:17s}: {data[column].unique()}')
```

transform categorical data into numbers

```
class_categories = {'First':1, 'Second':2, 'Third':3}
df['class'] = df['class'].apply(lambda x: class_categories[x])
```

or use this method

```
for column in data.select_dtypes('object'):
   if len(data[column].unique()) == 2:
      data[column] = pd.get_dummies(data[column], dtype='int64')
   else:
      data = pd.get_dummies(data, prefix=column, columns=[column], drop_first=True, dtype='int64')
```