20012531022
ALAUKI PATEL
CEIT-B
AI1

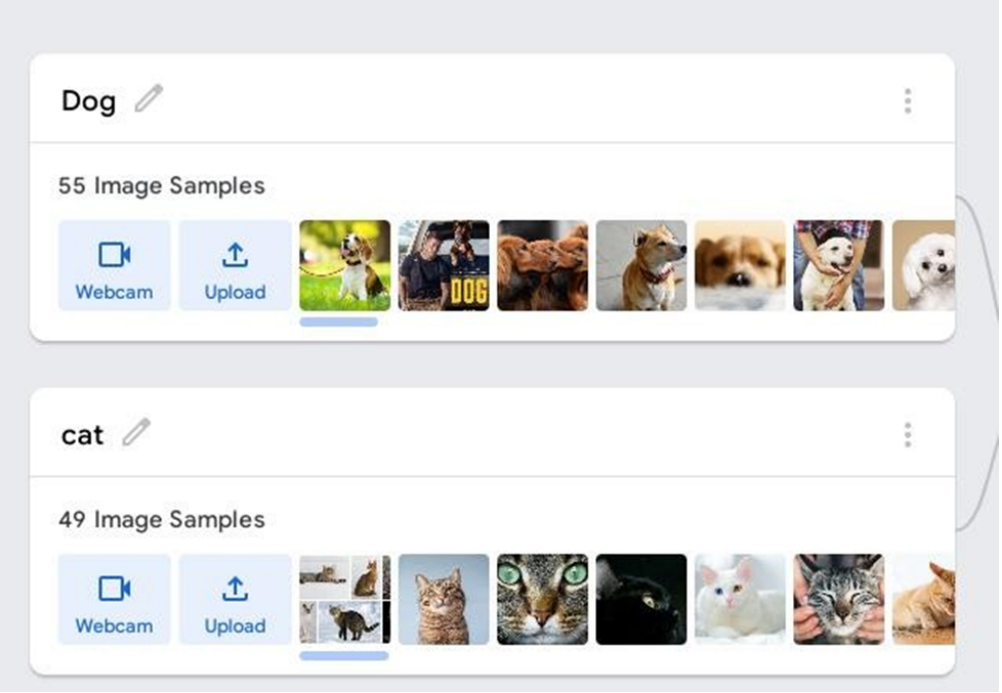
AIM:Handson practice on Teachable machine

Teachable Machine

IMAGE PROJECT

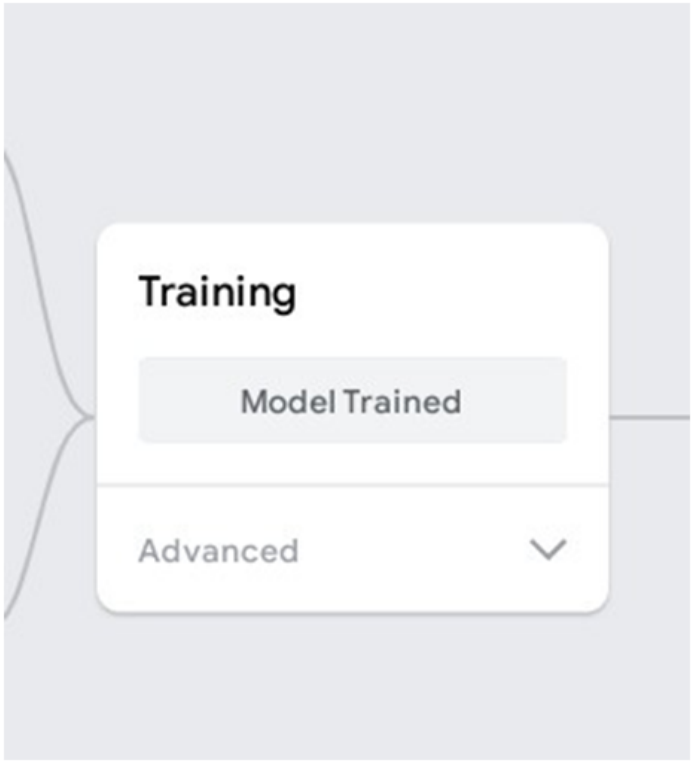
Step 1: Gather all the files

Upload all the images in class 1 , named Dog Upload all the images in class 2 , named Cat



Step 2: Train the model

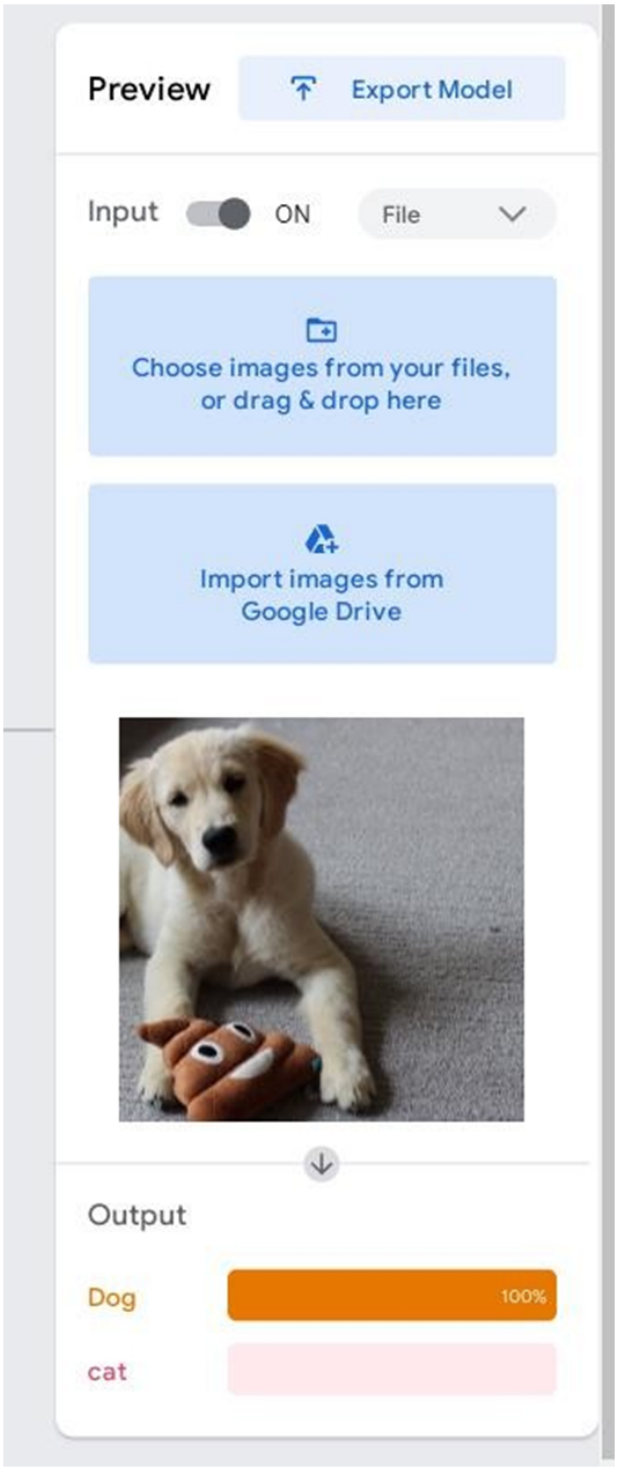
20012531022
ALAUKI PATEL
CEIT-B
AI1



Step 3: Export your model

Step 4: Preview the model , checking the model if it is working correctly

20012531022
ALAUKI PATEL
CEIT-B
AI1



20012531022

ALAUKI PATEL

```
import numpy as np
#import matplotlib.pyplot as plt
import pandas as pd
```

```
# Classes and inbuilt methods

# Importing the dataset
dataset = pd.read_csv('/content/dataset1.csv')   # XLSX read_excel  .tsv     read_csv('d1.tsv
# https://docs.google.com/a/ganpatuniversity.ac.in/viewer?a=v&pid=sites&srcid=Z2FucGF0dW5pdmV
```

```
dataset
```

	Country	Age	Salary	Purchased
0	France	44.0	72000.0	No
1	Spain	27.0	48000.0	Yes
2	Germany	30.0	54000.0	No
3	Spain	38.0	61000.0	No
4	Germany	40.0	NaN	Yes
5	France	35.0	58000.0	Yes
6	Spain	NaN	52000.0	No
7	France	48.0	79000.0	Yes
8	Germany	50.0	83000.0	No
9	France	37.0	67000.0	Yes

```
# divide my dataset into 2 parts - input and output

X = dataset.iloc[:, :-1].values
y = dataset.iloc[:, -1].values
```

https://stackoverflow.com/questions/14419206/what-does-1-mean-in-python

```
X
```

```
array([['France', 44.0, 72000.0],
       ['Spain', 27.0, 48000.0],
       ['Germany', 30.0, 54000.0],
       ['Spain', 38.0, 61000.0],
       ['Germany', 40.0, nan],
       ['France', 35.0, 58000.0],
       ['Spain', nan, 52000.0],
       ['France', 48.0, 79000.0],
       ['Germany', 50.0, 83000.0],
       ['France', 37.0, 67000.0]], dtype=object)
```

```
y
```

```
array(['No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes'],
      dtype=object)
```

```
from sklearn.impute import SimpleImputer
```

```
imputer = SimpleImputer(missing_values = np.nan, strategy = 'mean')
imputer = imputer.fit(X[:, 1:3])
# imputation is the process of replacing missing data with substituted values.
#When substituting for a data point, it is known as "unit imputation";
#when substituting for a component of a data point, it is known as "item imputation"
```

https://scikit-learn.org/stable/modules/generated/sklearn.impute.SimpleImputer.html

Sample

```
X[:, 1:3] = imputer.transform(X[:, 1:3])
```

```
from sklearn.preprocessing import LabelEncoder

labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
print(X)
```

```
[[0 44.0 72000.0]
 [2 27.0 48000.0]
 [1 30.0 54000.0]
 [2 38.0 61000.0]
 [1 40.0 63777.77777777778]
 [0 35.0 58000.0]
 [2 38.77777777777778 52000.0]
 [0 48.0 79000.0]
```

```
 [1 50.0 83000.0]
 [0 37.0 67000.0]]
```

https://www.geeksforgeeks.org/python-create-test-datasets-using-sklearn/

https://www.geeksforgeeks.org/ml-one-hot-encoding-of-datasets-in-python/

```
# On dataset1

from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size = .3, random_state = 0)
```

```
X_train
```

```
array([[-0.8660254 , -0.2029809 ,  0.44897083],
       [ 1.15470054, -1.82168936, -1.41706417],
       [ 1.15470054,  0.08478949, -1.0242147 ],
       [-0.8660254 ,  1.5775984 ,  1.62751925],
       [ 1.15470054, -0.04111006, -0.14030338],
       [-0.8660254 ,  0.93011502,  0.94003267],
       [-0.8660254 , -0.52672259, -0.43494049]])
```

```
X_test
```

```
array([[ 0.14433757, -1.33607682, -0.82778996],
       [ 0.14433757,  1.90134009,  2.02036872],
       [ 0.14433757,  0.28263164,  0.13250875]])
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()

X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
X_train
```

```
array([[-0.8660254 , -0.2029809 ,  0.44897083],
       [ 1.15470054, -1.82168936, -1.41706417],
       [ 1.15470054,  0.08478949, -1.0242147 ],
       [-0.8660254 ,  1.5775984 ,  1.62751925],
       [ 1.15470054, -0.04111006, -0.14030338],
       [-0.8660254 ,  0.93011502,  0.94003267],
       [-0.8660254 , -0.52672259, -0.43494049]])
```

A value is standardized as follows:

y = (x – mean) / standard_deviation Where the mean is calculated as:

mean = sum(x) / count(x) And the standard_deviation is calculated as:

standard_deviation = sqrt(sum((x – mean)^2) / count(x)) We can guesstimate a mean of 10.0 and a standard deviation of about 5.0. Using these values, we can standardize the first value of 20.7 as follows:

y = (x – mean) / standard_deviation y = (20.7 – 10) / 5 y = (10.7) / 5 y = 2.14

```
# Feature Scaling
from sklearn.preprocessing import MinMaxScaler
sc_X = MinMaxScaler()

X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

A value is normalized as follows:

y = (x – min) / (max – min) Where the minimum and maximum values pertain to the value x being normalized.

For example, for a dataset, we could guesstimate the min and max observable values as 30 and -10. We can then normalize any value, like 18.8, as follows:

y = (x – min) / (max – min) y = (18.8 – (-10)) / (30 – (-10)) y = 28.8 / 40 y = 0.72

```
X_train
```

```
array([[0.        , 0.47619048, 0.61290323],
       [1.        , 0.        , 0.        ],
       [1.        , 0.56084656, 0.12903226],
       [0.        , 1.        , 1.        ],
       [1.        , 0.52380952, 0.41935484],
       [0.        , 0.80952381, 0.77419355],
       [0.        , 0.38095238, 0.32258065]])
```

```
X_test
```

```
array([[1, 30.0, 54000.0],
       [1, 50.0, 83000.0],
       [1, 40.0, 63777.77777777778]], dtype=object)
```

```
data1 = [[12, 3, 34], [10, 32, 4],  [7, 11, 20]]
```

```
data1
```

```
     [[12, 3, 34], [10, 32, 4], [7, 11, 20]]
```

```
# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()

X_train = sc_X.fit_transform(data1)
```

```
X_train
```

```
     array([[ 1.13554995, -1.00850764,  1.19664225],
            [ 0.16222142,  1.36284817, -1.25103507],
            [-1.29777137, -0.35434052,  0.05439283]])
```

https://www.calculator.net/standard-deviation-calculator.html?numberinputs=12%2C10%2C7&ctype=p&x=39&y=17

https://machinelearningmastery.com/standardscaler-and-minmaxscaler-transforms-in-python/

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

```
# Data Preprocessing

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd


# Importing the dataset
dataset = pd.read_csv('/content/dataset2.csv')
X = dataset.iloc[:, :].values
#y = dataset.iloc[:, 3].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
print(X.describe())

transform = ColumnTransformer([('Outlook_OL0_OL1',OneHotEncoder(),[0],)], remainder = 'passth

# columnTransformer = ColumnTransformer([('encoder',
#                                         OneHotEncoder(),
#                                         [0])],
#                                       remainder='passthrough')
```

```
X = transform.fit_transform(X)
X
```

```
array([[0.0, 0.0, 1.0, 1, 'Sunny'],
       [0.0, 0.0, 1.0, 2, 'Sunny'],
       [1.0, 0.0, 0.0, 3, 'Overcast'],
       [0.0, 1.0, 0.0, 4, 'Rain'],
       [0.0, 1.0, 0.0, 5, 'Rain'],
       [0.0, 1.0, 0.0, 2, 'Rain'],
       [1.0, 0.0, 0.0, 3, 'Overcast'],
       [0.0, 0.0, 1.0, 1, 'Sunny'],
       [0.0, 0.0, 1.0, 2, 'Sunny'],
       [0.0, 1.0, 0.0, 3, 'Rain'],
       [0.0, 0.0, 1.0, 4, 'Sunny'],
       [1.0, 0.0, 0.0, 2, 'Overcast'],
       [1.0, 0.0, 0.0, 1, 'Overcast'],
       [0.0, 1.0, 0.0, 3, 'Rain']], dtype=object)
```

```
array([[0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 2],
       [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 3],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 4],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 5],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 2],
       [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 3],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 2],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 3],
       [0.0, 0.0, 1.0, 0.0, 0.0, 1.0, 4],
       [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 2],
       [1.0, 0.0, 0.0, 1.0, 0.0, 0.0, 1],
       [0.0, 1.0, 0.0, 0.0, 1.0, 0.0, 3]], dtype=object)
```

```
transform =  ColumnTransformer([("Outlook_OL0_OL1",OneHotEncoder(),[6])], remainder = 'passth
X = transform.fit_transform(X)
print(X.astype(int))
```

```
[[1 0 0 0 0 0 0 1 0 0 1]
 [0 1 0 0 0 0 0 1 0 0 1]
 [0 0 1 0 0 1 0 0 1 0 0]
 [0 0 0 1 0 0 1 0 0 1 0]
```

```
[0 0 0 0 1 0 1 0 0 1 0]
[0 1 0 0 0 0 1 0 0 1 0]
[0 0 1 0 0 1 0 0 1 0 0]
[1 0 0 0 0 0 0 1 0 0 1]
[0 1 0 0 0 0 0 1 0 0 1]
[0 0 1 0 0 0 1 0 0 1 0]
[0 0 0 1 0 0 0 1 0 0 1]
[0 1 0 0 0 1 0 0 1 0 0]
[1 0 0 0 0 1 0 0 1 0 0]
[0 0 1 0 0 0 1 0 0 1 0]]
```

https://www.calculator.net/standard-deviation-calculator.html?numberinputs=12%2C10%2C7&ctype=p&x=39&y=17

```
# Program for demonstration of one hot encoding

# import libraries
import numpy as np
import pandas as pd

# import the data required
data = pd.read_csv("/content/dataset2.csv")
print(data)

# importing one hot encoder from sklearn
# There are changes in OneHotEncoder class
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer

# creating one hot encoder object with categorical feature 0
# indicating the first column
columnTransformer = ColumnTransformer([('encoder',
                                        OneHotEncoder(),
                                        [0])],
                                      remainder='passthrough')

data = np.array(columnTransformer.fit_transform(data), dtype = np.str)
print(data)
```

```
      Outlook  Outlook0  Outlook1
0       Sunny         1     Sunny
1       Sunny         2     Sunny
2    Overcast         3  Overcast
3        Rain         4      Rain
4        Rain         5      Rain
5        Rain         2      Rain
6    Overcast         3  Overcast
7       Sunny         1     Sunny
8       Sunny         2     Sunny
9        Rain         3      Rain
```

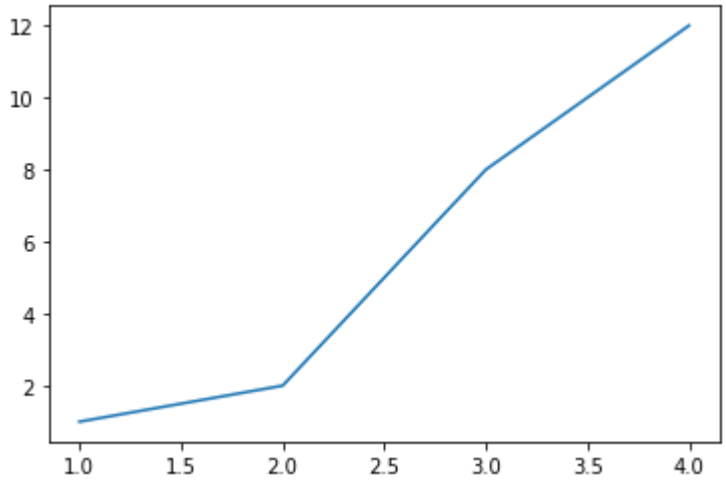
```
     10     Sunny         4     Sunny
     11  Overcast         2  Overcast
     12  Overcast         1  Overcast
     13      Rain         3      Rain
     [['0.0' '0.0' '1.0' '1' 'Sunny']
      ['0.0' '0.0' '1.0' '2' 'Sunny']
      ['1.0' '0.0' '0.0' '3' 'Overcast']
      ['0.0' '1.0' '0.0' '4' 'Rain']
      ['0.0' '1.0' '0.0' '5' 'Rain']
      ['0.0' '1.0' '0.0' '2' 'Rain']
      ['1.0' '0.0' '0.0' '3' 'Overcast']
      ['0.0' '0.0' '1.0' '1' 'Sunny']
      ['0.0' '0.0' '1.0' '2' 'Sunny']
      ['0.0' '1.0' '0.0' '3' 'Rain']
      ['0.0' '0.0' '1.0' '4' 'Sunny']
      ['1.0' '0.0' '0.0' '2' 'Overcast']
      ['1.0' '0.0' '0.0' '1' 'Overcast']
      ['0.0' '1.0' '0.0' '3' 'Rain']]
```

```
# Importing the SimpleImputer class
from sklearn.impute import SimpleImputer

# Imputer object using the mean strategy and
# missing_values type for imputation
imputer1 = SimpleImputer(missing_values = np.nan,
                         strategy ='mean')

data = [[12, np.nan, 34], [10, 32, np.nan],
        [np.nan, 11, 20]]

print("Original Data : \n", data)
# Fitting the data to the imputer object
imputer1 = imputer.fit(data)

# Imputing the data
data = imputer1.transform(data)

print("Imputed Data : \n", data)
```

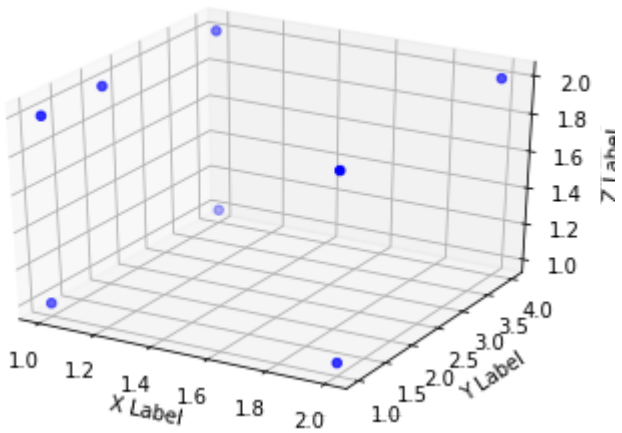
```
     Original Data :
      [[12, nan, 34], [10, 32, nan], [nan, 11, 20]]
     Imputed Data :
      [[12.  21.5 34. ]
      [10.  32.  27. ]
      [11.  11.  20. ]]
```

```
#2D Plot with Interpolation
import matplotlib.pyplot as plt
import numpy as np
from scipy.interpolate import make_interp_spline
x=np.array([1,2,3,4])
y=np.array([1,2,8,12])
```

```
x_smooth = np.linspace(x.min(),x.max(),4)
y_smooth=make_interp_spline(x,y)(x_smooth)
plt.plot(x_smooth,y_smooth)
plt.show()
```



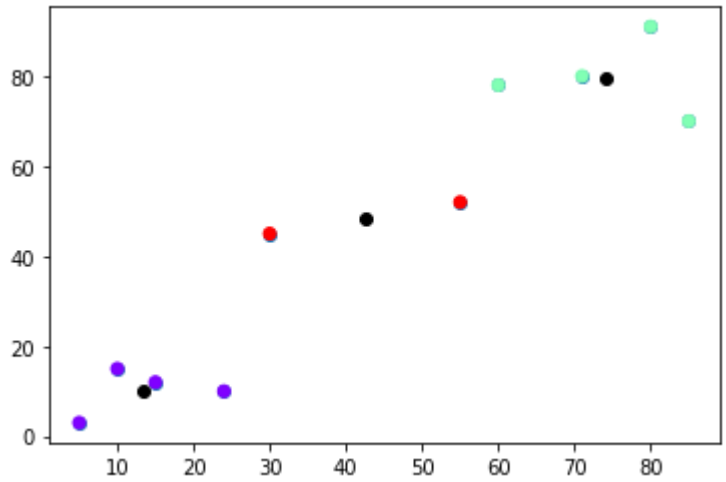

```
#3D Scatter Plot
import matplotlib.pyplot as plt
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
x =[1,1,1,2,1,2,1,2]
y= [4,2,4,1,1,4,1,1]
z =[1,2,2,2,1,2,2,1]
ax.scatter(x, y, z, c='b', marker='o')
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_zlabel('Z Label')
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np
```

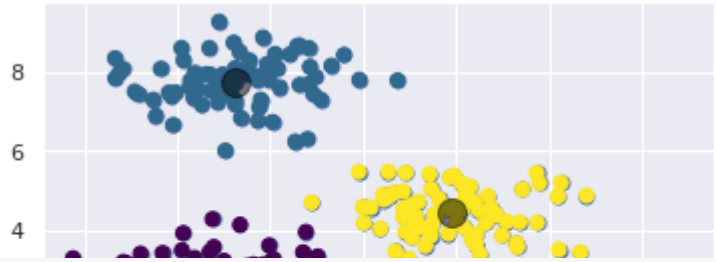
```
from sklearn.cluster import KMeans
X = np.array([[5,3],[10,15], [15,12], [24,10], [30,45], [85,70], [71,80], [60,78], [55,52], [
plt.scatter(X[:,0],X[:,1], label='True Position')
kmeans = KMeans(n_clusters=3)
kmeans.fit(X)
print(kmeans.cluster_centers_)
print(kmeans.labels_)
plt.scatter(X[:,0],X[:,1], c=kmeans.labels_, cmap='rainbow')
plt.scatter(kmeans.cluster_centers_[:,0] ,kmeans.cluster_centers_[:,1], color='black')
```

```
[[13.5  10.  ]
 [74.   79.75]
 [42.5  48.5 ]]
[0 0 0 0 2 1 1 1 2 1]
<matplotlib.collections.PathCollection at 0x7f1730b70910>
```




```
#K-means clustering on random generated values using K-means functionalities of  scikit-learn
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.datasets.samples_generator import make_blobs
from sklearn.cluster import KMeans
X, y_true = make_blobs(n_samples=300, centers=4, cluster_std=0.60,  random_state=0)
plt.scatter(X[:, 0], X[:, 1], s=50);
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
y_kmeans = kmeans.predict(X)
plt.scatter(X[:, 0], X[:, 1], c=y_kmeans, s=50, cmap='viridis')
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0], centers[:, 1], c='black', s=200, alpha=0.5);
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:144: FutureWarning:
  warnings.warn(message, FutureWarning)
```




```
## 20012531022
## ALAUKI PATEL
```

20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

```
import pandas as pd
```

```
df = pd.read_csv("/content/Covid cases in India.csv")
```

```
df
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228	15649	3800	779
1	2.0	Gujarat	7797	5234	2091	472
2	3.0	Delhi	6542	4454	2020	68
3	4.0	Tamil Nadu	6535	4667	1824	44
4	5.0	Rajasthan	3741	1458	2176	107
5	6.0	Madhya Pradesh	3457	1766	1480	211
6	7.0	Uttar Pradesh	3373	1800	1499	74
7	8.0	Andhra Pradesh	1930	999	887	44
8	9.0	West Bengal	1786	1243	372	171
9	10.0	Punjab	1762	1574	157	31
10	11.0	Telangana	1163	382	751	30
11	12.0	Jammu and Kashmir	836	459	368	9

```
df.head()
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228	15649	3800	779
1	2.0	Gujarat	7797	5234	2091	472
2	3.0	Delhi	6542	4454	2020	68
3	4.0	Tamil Nadu	6535	4667	1824	44
4	5.0	Rajasthan	3741	1458	2176	107

```
df.tail(7)
```

```
S. No.  Name of State / UT  TotalConfirmedCases  Active  Recovere
```

```
df1=pd.read_csv("/content/Covid cases in India_2.csv")
```

```
df1.head(3)
```

	S. No.	Name of State / UT	Total Confirmed cases (Indian National)	Total Confirmed cases (Foreign National)	Cured/	Death	Unnamed: 6	S. No..1	Name of State / UT.1	N
0	NaN	NaN	NaN	NaN	Discharged	NaN	NaN	NaN	NaN	
1	1	Andhra Pradesh	1.0	0.0	0	0.0	NaN	1.0	Andhra Pradesh	

```
columns = df.columns
```

```
columns
```

```
Index(['S. No.', 'Name of State / UT', 'TotalConfirmedCases', 'Active',
       'Recovered', 'Deaths'],
      dtype='object')
```

```
df.columns[1:4]
```

```
Index(['Name of State / UT', 'TotalConfirmedCases', 'Active'], dtype='object')
```

```
df['TotalConfirmedCases']
```

```
0     20228
1      7797
2      6542
3      6535
4      3741
5      3457
6      3373
7      1930
8      1786
9      1762
10     1163
11      836
12      794
13      675
14      629
15      506
16      352
17      169
18      156
19      135
```

```
20       67
21       62
22       59
23       52
24       42
25       33
26       13
27       10
28        7
29        2
30        1
31        1
32        1
33    62916
Name: TotalConfirmedCases, dtype: int64
```

```
df['Active']
```

```
0     15649
1      5234
2      4454
3      4667
4      1458
5      1766
6      1800
7       999
8      1243
9      1574
10      382
11      459
12      377
13      376
14      306
15       17
16      281
17      143
18       75
19      133
20       20
21       26
22       16
23       11
24       24
25        0
26        2
27        2
28        0
29        0
30        0
31        0
32        1
33    41495
Name: Active, dtype: int64
```

```
df['Name of State / UT']
```

```
0                                   Maharashtra
1                                       Gujarat
2                                         Delhi
3                                    Tamil Nadu
4                                     Rajasthan
5                                Madhya Pradesh
6                                 Uttar Pradesh
7                                Andhra Pradesh
8                                   West Bengal
9                                        Punjab
10                                    Telangana
11                            Jammu and Kashmir
12                                    Karnataka
13                                      Haryana
14                                        Bihar
15                                       Kerala
16                                       Odisha
17                                   Chandigarh
18                                    Jharkhand
19                                      Tripura
20                                  Uttarakhand
21                                        Assam
22                                 Chhattisgarh
23                              Himachal Pradesh
24                                       Ladakh
25                  Andaman and Nicobar Islands
26                                    Meghalaya
27                                   Puducherry
28                                          Goa
29                                      Manipur
30                                      Mizoram
31                             Arunachal Pradesh
32    Dadra and Nagar Haveli and Daman and Diu
33                                        Total
Name: Name of State / UT, dtype: object
```

```
new_df = df[df.columns[1:3]]
new_df.head(3)
```

	Name of State / UT	TotalConfirmedCases
0	Maharashtra	20228
1	Gujarat	7797
2	Delhi	6542

```
new_df=df[10:15]
new_df.head(10)
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
10	11.0	Telangana	1163	382	751	30
11	12.0	Jammu and Kashmir	836	459	368	9
12	13.0	Karnataka	794	377	386	30
13	14.0	Haryana	675	376	290	9
14	15.0	Bihar	629	306	318	5

```
new_columns = ['Name of State / UT', 'Active']
new_df = df[new_columns]
new_df.head(15)
```

	Name of State / UT	Active
0	Maharashtra	15649
1	Gujarat	5234
2	Delhi	4454
3	Tamil Nadu	4667
4	Rajasthan	1458
5	Madhya Pradesh	1766
6	Uttar Pradesh	1800
7	Andhra Pradesh	999
8	West Bengal	1243
9	Punjab	1574
10	Telangana	382
11	Jammu and Kashmir	459
12	Karnataka	377
13	Haryana	376
14	Bihar	306

```
df
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovere
0	1.0	Maharashtra	20228	15649	380
1	2.0	Gujarat	7797	5234	209
2	3.0	Delhi	6542	4454	202
3	4.0	Tamil Nadu	6535	4667	182
4	5.0	Rajasthan	3741	1458	217
5	6.0	Madhya Pradesh	3457	1766	148
6	7.0	Uttar Pradesh	3373	1800	149
7	8.0	Andhra Pradesh	1930	999	88
8	9.0	West Bengal	1786	1243	37
9	10.0	Punjab	1762	1574	15
10	11.0	Telangana	1163	382	75
11	12.0	Jammu and Kashmir	836	459	36
12	13.0	Karnataka	794	377	38
13	14.0	Haryana	675	376	29
14	15.0	Bihar	629	306	31
15	16.0	Kerala	506	17	48
16	17.0	Odisha	352	281	6
17	18.0	Chandigarh	169	143	2
18	19.0	Jharkhand	156	75	7
19	20.0	Tripura	135	133	
20	21.0	Uttarakhand	67	20	4
21	22.0	Assam	62	26	3
22	23.0	Chhattisgarh	59	16	4
23	24.0	Himachal Pradesh	52	11	3
24	25.0	Ladakh	42	24	1
25	26.0	Andaman and Nicobar Islands	33	0	3
26	27.0	Meghalaya	13	2	1
27	28.0	Puducherry	10	2	
28	29.0	Goa	7	0	
29	30.0	Manipur	2	0	

```
new_columns = ['Name of State / UT', 'Recovered']
new_df = df[new_columns]
new_df.head(3)
```

	Name of State / UT	Recovered
0	Maharashtra	3800
1	Gujarat	2091
2	Delhi	2020

```
df['Active'] >500
```

```
0      True
1      True
2      True
3      True
4      True
5      True
6      True
7      True
8      True
9      True
10    False
11    False
12    False
13    False
14    False
15    False
16    False
17    False
18    False
19    False
20    False
21    False
22    False
23    False
24    False
25    False
26    False
27    False
28    False
29    False
30    False
31    False
32    False
33     True
Name: Active, dtype: bool
```

```
df[df['Active'] >500,[]]  # this will not work
```

```
df.loc[df['Active'] < 500, ['Name of State / UT','Active', 'Deaths']]
```

	Name of State / UT	Active	Deaths
10	Telangana	382	30
11	Jammu and Kashmir	459	9
12	Karnataka	377	30
13	Haryana	376	9
14	Bihar	306	5
15	Kerala	17	4
16	Odisha	281	3
17	Chandigarh	143	2
18	Jharkhand	75	3
19	Tripura	133	0
20	Uttarakhand	20	1
21	Assam	26	1
22	Chhattisgarh	16	0
23	Himachal Pradesh	11	3
24	Ladakh	24	0
25	Andaman and Nicobar Islands	0	0
26	Meghalaya	2	1
27	Puducherry	2	0
28	Goa	0	0
29	Manipur	0	0
30	Mizoram	0	0
31	Arunachal Pradesh	0	0
32	Dadra and Nagar Haveli and Daman and Diu	1	0

```
df.loc[df['Active'] <500 and df['Active']>100],[['Name of State / UT','Active', 'Deaths']]
```

```
df.iloc[(df['Deaths'] > 400).values, [1,3,5]]
```

	Name of State / UT	Active	Deaths
0	Maharashtra	15649	779
1	Gujarat	5234	472
33	Total	41495	2102

```
df.iloc[(df['Deaths'] > 400).values & (df['Active']>100).values, [0,1,3,5]]
```

	S. No.	Name of State / UT	Active	Deaths
0	1.0	Maharashtra	15649	779
1	2.0	Gujarat	5234	472
33	NaN	Total	41495	2102

```
df.head(1)
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228	15649	3800	779

```
df.nlargest(10,'Deaths')
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
33	NaN	Total	62916	41495	19315	2102
0	1.0	Maharashtra	20228	15649	3800	779
1	2.0	Gujarat	7797	5234	2091	472
5	6.0	Madhya Pradesh	3457	1766	1480	211
8	9.0	West Bengal	1786	1243	372	171
4	5.0	Rajasthan	3741	1458	2176	107
6	7.0	Uttar Pradesh	3373	1800	1499	74
2	3.0	Delhi	6542	4454	2020	68
3	4.0	Tamil Nadu	6535	4667	1824	44
7	8.0	Andhra Pradesh	1930	999	887	44

```
df.nsmallest(10,'Active')
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovere
25	26.0	Andaman and Nicobar Islands	33	0	3
28	29.0	Goa	7	0	
29	30.0	Manipur	2	0	
30	31.0	Mizoram	1	0	
31	32.0	Arunachal Pradesh	1	0	
32	33.0	Dadra and Nagar Haveli and Daman and Diu	1	1	
26	27.0	Meghalaya	13	2	1
27	28.0	Puducherry	10	2	
23	24.0	Himachal Pradesh	52	11	3
22	23.0	Chhattisgarh	59	16	4

```
mylist = list(df.nsmallest(5, 'Recovered')['Name of State / UT'])
mylist
```

```
['Dadra and Nagar Haveli and Daman and Diu',
 'Mizoram',
 'Arunachal Pradesh',
 'Tripura',
 'Manipur']
```

```
df.info
```

```
<bound method DataFrame.info of     S. No.                    Name of State / UT
...  Recovered  Deaths
0      1.0                         Maharashtra  ...       3800     779
1      2.0                             Gujarat  ...       2091     472
2      3.0                               Delhi  ...       2020      68
3      4.0                           Tamil Nadu  ...       1824      44
4      5.0                            Rajasthan  ...       2176     107
5      6.0                       Madhya Pradesh  ...       1480     211
6      7.0                        Uttar Pradesh  ...       1499      74
7      8.0                       Andhra Pradesh  ...        887      44
8      9.0                          West Bengal  ...        372     171
9     10.0                               Punjab  ...        157      31
10    11.0                            Telangana  ...        751      30
11    12.0                      Jammu and Kashmir  ...        368       9
12    13.0                            Karnataka  ...        386      30
13    14.0                              Haryana  ...        290       9
14    15.0                                Bihar  ...        318       5
15    16.0                               Kerala  ...        485       4
16    17.0                               Odisha  ...         68       3
17    18.0                           Chandigarh  ...         24       2
18    19.0                            Jharkhand  ...         78       3
19    20.0                              Tripura  ...          2       0
20    21.0                          Uttarakhand  ...         46       1
21    22.0                                Assam  ...         35       1
```

```
22    23.0                              Chhattisgarh  ...         43       0
23    24.0                          Himachal Pradesh  ...         35       3
24    25.0                                   Ladakh  ...         18       0
25    26.0               Andaman and Nicobar Islands  ...         33       0
26    27.0                                 Meghalaya  ...         10       1
27    28.0                                Puducherry  ...          8       0
28    29.0                                       Goa  ...          7       0
29    30.0                                   Manipur  ...          2       0
30    31.0                                   Mizoram  ...          1       0
31    32.0                            Arunachal Pradesh  ...          1       0
32    33.0  Dadra and Nagar Haveli and Daman and Diu  ...          0       0
33     NaN                                     Total  ...      19315    2102

[34 rows x 6 columns]>
```

```
df.describe
```

```
<bound method NDFrame.describe of     S. No.                        Name of State / UT
...  Recovered  Deaths
0      1.0                                Maharashtra  ...       3800     779
1      2.0                                    Gujarat  ...       2091     472
2      3.0                                      Delhi  ...       2020      68
3      4.0                                  Tamil Nadu  ...       1824      44
4      5.0                                  Rajasthan  ...       2176     107
5      6.0                               Madhya Pradesh  ...       1480     211
6      7.0                                Uttar Pradesh  ...       1499      74
7      8.0                               Andhra Pradesh  ...        887      44
8      9.0                                  West Bengal  ...        372     171
9     10.0                                       Punjab  ...        157      31
10    11.0                                   Telangana  ...        751      30
11    12.0                            Jammu and Kashmir  ...        368       9
12    13.0                                   Karnataka  ...        386      30
13    14.0                                     Haryana  ...        290       9
14    15.0                                       Bihar  ...        318       5
15    16.0                                      Kerala  ...        485       4
16    17.0                                      Odisha  ...         68       3
17    18.0                                  Chandigarh  ...         24       2
18    19.0                                   Jharkhand  ...         78       3
19    20.0                                     Tripura  ...          2       0
20    21.0                                 Uttarakhand  ...         46       1
21    22.0                                       Assam  ...         35       1
22    23.0                                Chhattisgarh  ...         43       0
23    24.0                            Himachal Pradesh  ...         35       3
24    25.0                                     Ladakh  ...         18       0
25    26.0                 Andaman and Nicobar Islands  ...         33       0
26    27.0                                   Meghalaya  ...         10       1
27    28.0                                  Puducherry  ...          8       0
28    29.0                                         Goa  ...          7       0
29    30.0                                     Manipur  ...          2       0
30    31.0                                     Mizoram  ...          1       0
31    32.0                              Arunachal Pradesh  ...          1       0
32    33.0  Dadra and Nagar Haveli and Daman and Diu  ...          0       0
33     NaN                                       Total  ...      19315    2102

[34 rows x 6 columns]>
```

```
df['Deaths'].sum()
```

```
4204
```

```
df['TotalConfirmedCases'].sum()
```

```
125832
```

```
df['Active'].sum()
```

```
82990
```

```
df['Active'].count()
```

```
34
```

```
avg= df['Active'].sum() / df['Active'].count()
```

```
avg
```

```
2440.8823529411766
```

```
print("no. of active cases",avg)
```

```
no. of active cases 2440.8823529411766
```

```
df.drop(labels='Active',axis=1)
df.head()
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228	15649	3800	779
1	2.0	Gujarat	7797	5234	2091	472
2	3.0	Delhi	6542	4454	2020	68
3	4.0	Tamil Nadu	6535	4667	1824	44
4	5.0	Rajasthan	3741	1458	2176	107

```
import numpy as np
new_df = df[:]
```

```
new_df
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovere
0	1.0	Maharashtra	20228	15649	380
1	2.0	Gujarat	7797	5234	209
2	3.0	Delhi	6542	4454	202
3	4.0	Tamil Nadu	6535	4667	182
4	5.0	Rajasthan	3741	1458	217

```
new_df.iloc[1,2] = np.NaN
new_df.head()
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228.0	15649	3800	779
1	2.0	Gujarat	NaN	5234	2091	472
2	3.0	Delhi	6542.0	4454	2020	68
3	4.0	Tamil Nadu	6535.0	4667	1824	44
4	5.0	Rajasthan	3741.0	1458	2176	107

15	16.0	Kerala	306	17		4

```
new_df = new_df.dropna()
```

17	18.0	Chandigarh	169	143		2

```
new_df
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovere
0	1.0	Maharashtra	20228.0	15649	380
2	3.0	Delhi	6542.0	4454	202
3	4.0	Tamil Nadu	6535.0	4667	182
4	5.0	Rajasthan	3741.0	1458	217
5	6.0	Madhya Pradesh	3457.0	1766	148
6	7.0	Uttar Pradesh	3373.0	1800	149
7	8.0	Andhra Pradesh	1930.0	999	88
8	9.0	West Bengal	1786.0	1243	37
9	10.0	Punjab	1762.0	1574	15
10	11.0	Telangana	1163.0	382	75
11	12.0	Jammu and Kashmir	836.0	459	36
12	13.0	Karnataka	794.0	377	38
13	14.0	Haryana	675.0	376	29
14	15.0	Bihar	629.0	306	31
15	16.0	Kerala	506.0	17	48
16	17.0	Odisha	352.0	281	6
17	18.0	Chandigarh	169.0	143	2
18	19.0	Jharkhand	156.0	75	7
19	20.0	Tripura	135.0	133	
20	21.0	Uttarakhand	67.0	20	4
21	22.0	Assam	62.0	26	3
22	23.0	Chhattisgarh	59.0	16	4

```
new_df.reset_index(inplace=True, drop=True)
new_df.head()
```

	S. No.	Name of State / UT	TotalConfirmedCases	Active	Recovered	Deaths
0	1.0	Maharashtra	20228.0	15649	3800	779
1	3.0	Delhi	6542.0	4454	2020	68
2	4.0	Tamil Nadu	6535.0	4667	1824	44
3	5.0	Rajasthan	3741.0	1458	2176	107
4	6.0	Madhya Pradesh	3457.0	1766	1480	211

31 32 () Arunachal Pradesh 1 () ()

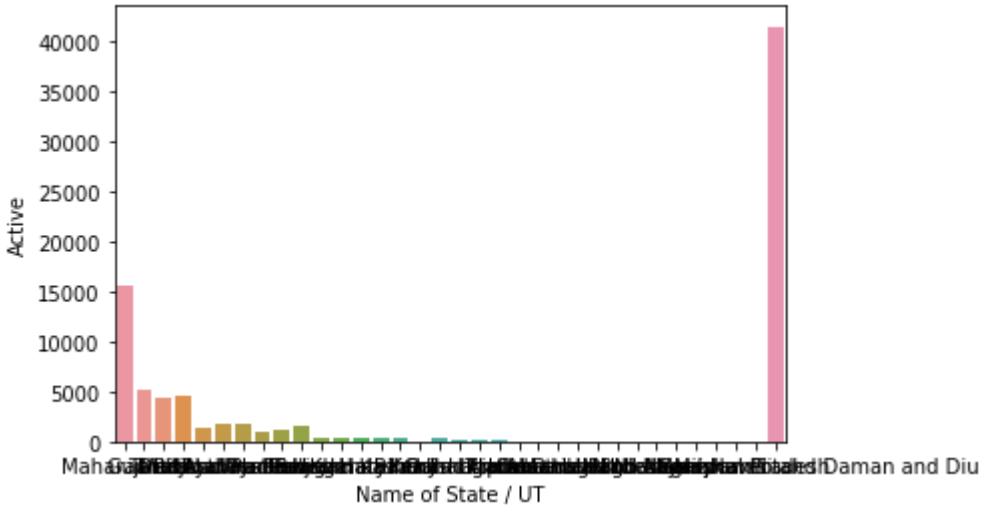
```
new_df.isnull().sum()
```

```
S. No.                0
Name of State / UT    0
TotalConfirmedCases   0
Active                0
Recovered             0
Deaths                0
dtype: int64
```

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
sns.barplot(x="Name of State / UT", y="Active", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f6ac0687610>
```

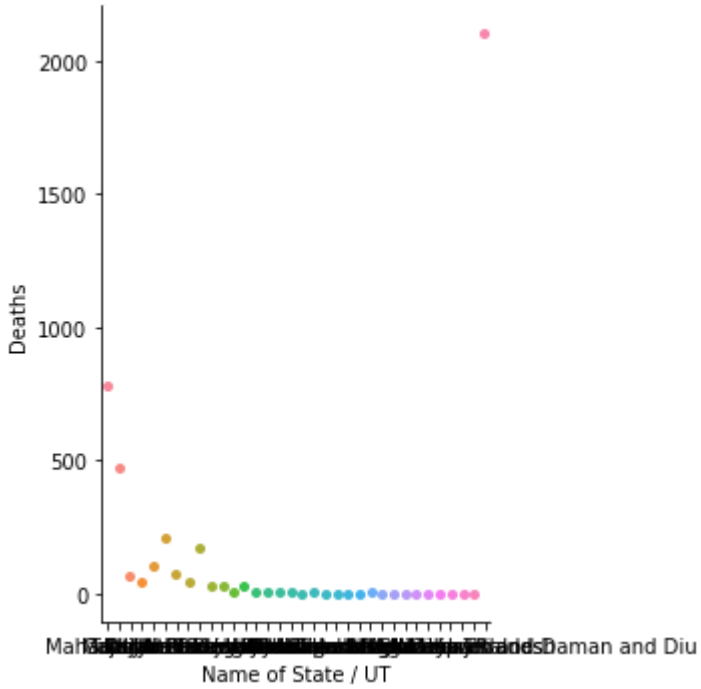



```
sns.boxplot(x="Name of State / UT", y="Deaths", data=df)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fa9af6f9a90>
```

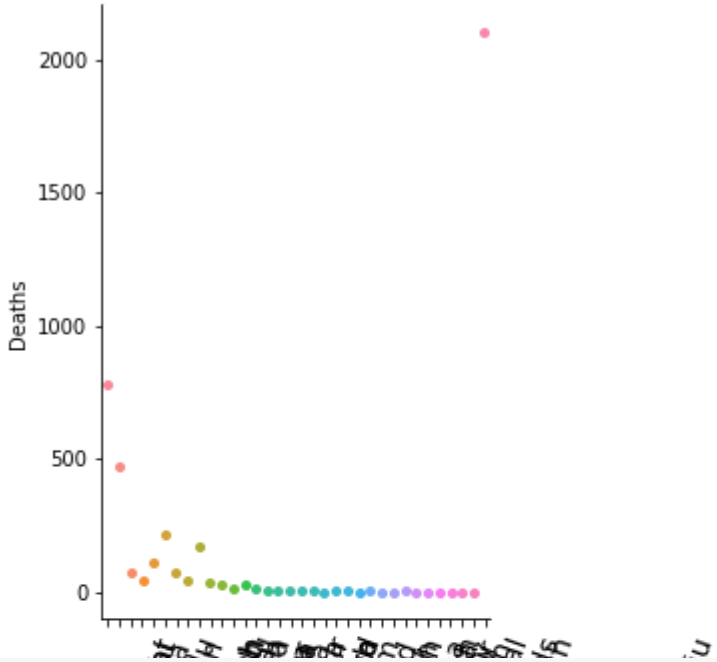

```
sns.catplot(x="Name of State / UT", y="Deaths", data=df)
```

```
<seaborn.axisgrid.FacetGrid at 0x7fa9afea1990>
```



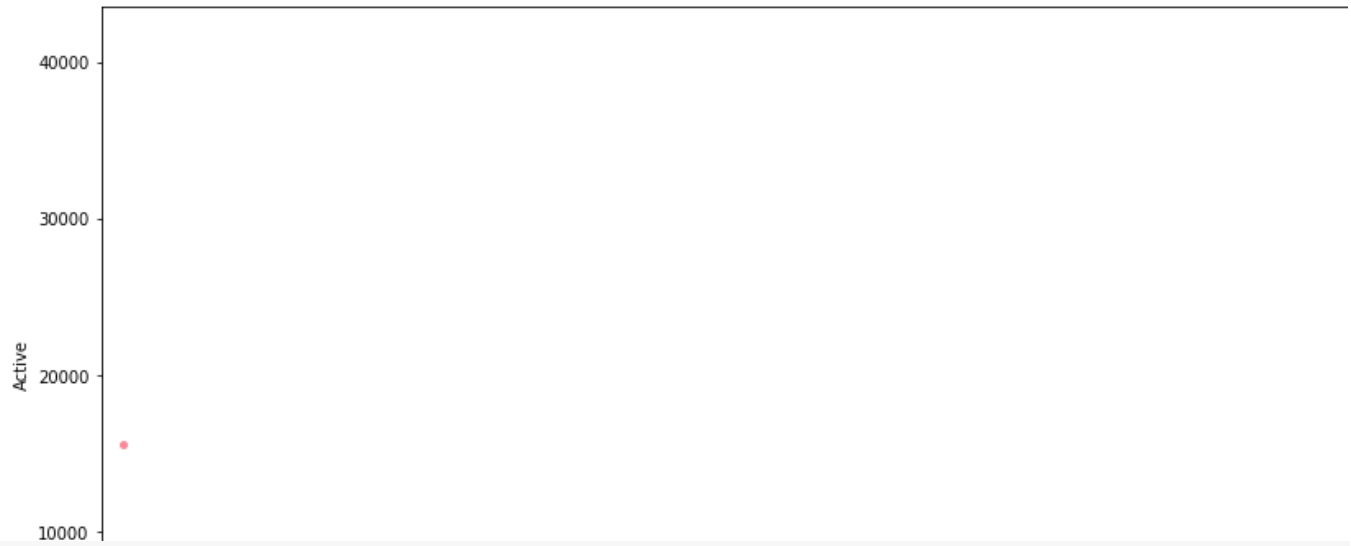

```
plt.figure(figsize=(16,8))
sns.catplot(x="Name of State / UT", y="Deaths", data=df)

plt.xticks(rotation=70, horizontalalignment='left', fontweight='light', fontsize='x-large')
plt.show()
```

```
<Figure size 1152x576 with 0 Axes>
```



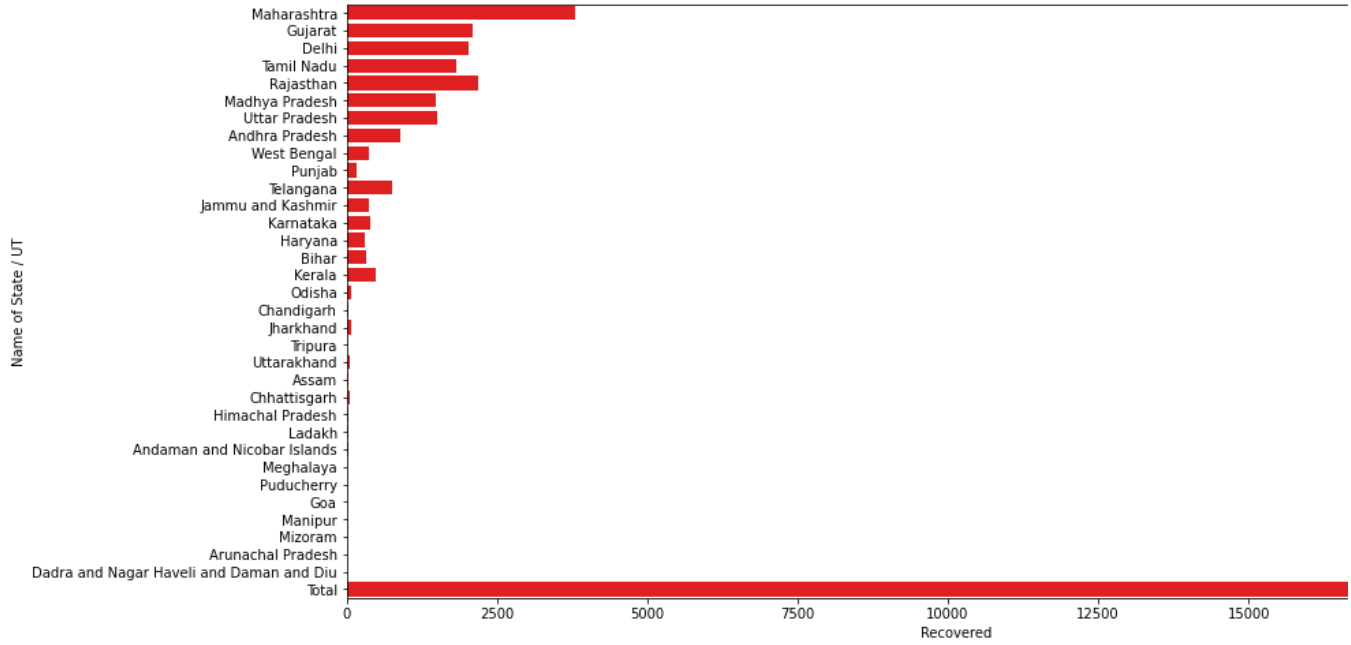

```
plt.figure(figsize=(16,8))
sns.stripplot(x="Name of State / UT", y="Active", data=df)
plt.xticks(rotation=70, horizontalalignment='right', fontweight='light', fontsize='x-large')
```

```
(array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16,
        17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33]),
 <a list of 34 Text major ticklabel objects>)
```




```
plt.figure(figsize=(16,8))
sns.barplot( x=df["Recovered"], y=df["Name of State / UT"], color="red", label="Recovered")
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7ff85aaa7e10>
```



20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

```
# importing the libraries

import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#import the dataset
dataset = pd.read_csv('/content/db5.csv')
# https://drive.google.com/file/d/1wjq1Un5qPT_ftNm3xJ2CdfpygLkayzzw/view?usp=sharing

X = dataset.iloc[:, 1:2].values
Y = dataset.iloc[:, -1].values
```

```
X
```

```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10]])
```

```
Y
```

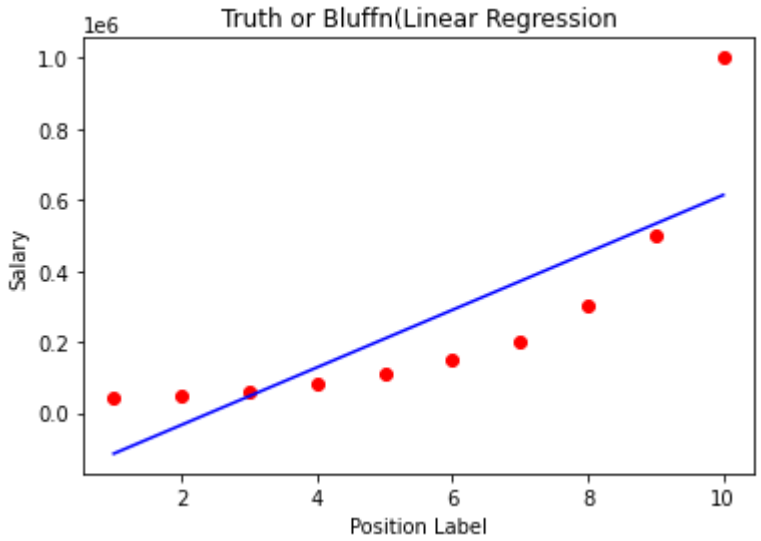
```
array([  45000,   50000,   60000,   80000,  110000,  150000,  200000,
        300000,  500000, 1000000])
```

```
# fitting linear regression to the dataset
from sklearn.linear_model import LinearRegression
lin_reg = LinearRegression()
lin_reg.fit(X, Y)
```

```
LinearRegression()
```

```
# Visualising the Linear Regression results
plt.scatter(X, Y, color = 'red')
plt.plot(X, lin_reg.predict(X), color = 'blue')
plt.title('Truth or Bluffn(Linear Regression')
```

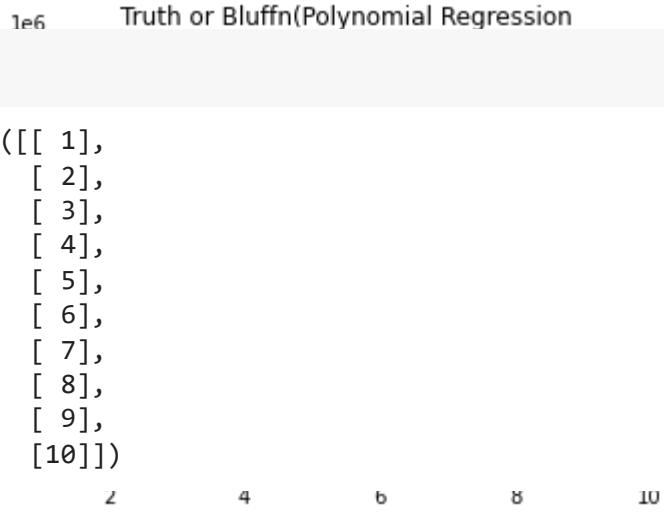
```
plt.xlabel('Position Label')
plt.ylabel('Salary')
plt.show()
```




```
# fitting polynomial regression to the dataset
from sklearn.preprocessing import PolynomialFeatures
poly_reg = PolynomialFeatures(degree = 10)
X_poly = poly_reg.fit_transform(X)
lin_reg_2 = LinearRegression()
lin_reg_2.fit(X_poly, Y)
```

```
LinearRegression()
```

```
plt.scatter(X, Y, color = 'red')
plt.plot(X, lin_reg_2.predict(poly_reg.fit_transform(X)), color = 'blue')
plt.title('Truth or Bluffn(Polynomial Regression')
plt.xlabel('Position Label')
plt.ylabel('Salary')
plt.show()
```




```
X
```

```
array([[ 1],
       [ 2],
       [ 3],
       [ 4],
       [ 5],
       [ 6],
       [ 7],
       [ 8],
       [ 9],
       [10]])
```

```
X_poly
```

```
array([[1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00, 1.00000000e+00,
        1.00000000e+00, 1.00000000e+00, 1.00000000e+00],
       [1.00000000e+00, 2.00000000e+00, 4.00000000e+00, 8.00000000e+00,
        1.60000000e+01, 3.20000000e+01, 6.40000000e+01, 1.28000000e+02,
        2.56000000e+02, 5.12000000e+02, 1.02400000e+03],
       [1.00000000e+00, 3.00000000e+00, 9.00000000e+00, 2.70000000e+01,
        8.10000000e+01, 2.43000000e+02, 7.29000000e+02, 2.18700000e+03,
        6.56100000e+03, 1.96830000e+04, 5.90490000e+04],
       [1.00000000e+00, 4.00000000e+00, 1.60000000e+01, 6.40000000e+01,
        2.56000000e+02, 1.02400000e+03, 4.09600000e+03, 1.63840000e+04,
        6.55360000e+04, 2.62144000e+05, 1.04857600e+06],
       [1.00000000e+00, 5.00000000e+00, 2.50000000e+01, 1.25000000e+02,
        6.25000000e+02, 3.12500000e+03, 1.56250000e+04, 7.81250000e+04,
        3.90625000e+05, 1.95312500e+06, 9.76562500e+06],
       [1.00000000e+00, 6.00000000e+00, 3.60000000e+01, 2.16000000e+02,
        1.29600000e+03, 7.77600000e+03, 4.66560000e+04, 2.79936000e+05,
        1.67961600e+06, 1.00776960e+07, 6.04661760e+07],
       [1.00000000e+00, 7.00000000e+00, 4.90000000e+01, 3.43000000e+02,
        2.40100000e+03, 1.68070000e+04, 1.17649000e+05, 8.23543000e+05,
        5.76480100e+06, 4.03536070e+07, 2.82475249e+08],
       [1.00000000e+00, 8.00000000e+00, 6.40000000e+01, 5.12000000e+02,
        4.09600000e+03, 3.27680000e+04, 2.62144000e+05, 2.09715200e+06,
        1.67772160e+07, 1.34217728e+08, 1.07374182e+09],
       [1.00000000e+00, 9.00000000e+00, 8.10000000e+01, 7.29000000e+02,
        6.56100000e+03, 5.90490000e+04, 5.31441000e+05, 4.78296900e+06,
        4.30467210e+07, 3.87420489e+08, 3.48678440e+09],
       [1.00000000e+00, 1.00000000e+01, 1.00000000e+02, 1.00000000e+03,
        1.00000000e+04, 1.00000000e+05, 1.00000000e+06, 1.00000000e+07,
        1.00000000e+08, 1.00000000e+09, 1.00000000e+10]])
```

```
np.round(X_poly,0) *10
```

```
array([[1.00000000e+01, 1.00000000e+01, 1.00000000e+01, 1.00000000e+01,
        1.00000000e+01, 1.00000000e+01, 1.00000000e+01, 1.00000000e+01,
        1.00000000e+01, 1.00000000e+01, 1.00000000e+01],
       [1.00000000e+01, 2.00000000e+01, 4.00000000e+01, 8.00000000e+01,
```

```
        1.60000000e+02, 3.20000000e+02, 6.40000000e+02, 1.28000000e+03,
        2.56000000e+03, 5.12000000e+03, 1.02400000e+04],
       [1.00000000e+01, 3.00000000e+01, 9.00000000e+01, 2.70000000e+02,
        8.10000000e+02, 2.43000000e+03, 7.29000000e+03, 2.18700000e+04,
        6.56100000e+04, 1.96830000e+05, 5.90490000e+05],
       [1.00000000e+01, 4.00000000e+01, 1.60000000e+02, 6.40000000e+02,
        2.56000000e+03, 1.02400000e+04, 4.09600000e+04, 1.63840000e+05,
        6.55360000e+05, 2.62144000e+06, 1.04857600e+07],
       [1.00000000e+01, 5.00000000e+01, 2.50000000e+02, 1.25000000e+03,
        6.25000000e+03, 3.12500000e+04, 1.56250000e+05, 7.81250000e+05,
        3.90625000e+06, 1.95312500e+07, 9.76562500e+07],
       [1.00000000e+01, 6.00000000e+01, 3.60000000e+02, 2.16000000e+03,
        1.29600000e+04, 7.77600000e+04, 4.66560000e+05, 2.79936000e+06,
        1.67961600e+07, 1.00776960e+08, 6.04661760e+08],
       [1.00000000e+01, 7.00000000e+01, 4.90000000e+02, 3.43000000e+03,
        2.40100000e+04, 1.68070000e+05, 1.17649000e+06, 8.23543000e+06,
        5.76480100e+07, 4.03536070e+08, 2.82475249e+09],
       [1.00000000e+01, 8.00000000e+01, 6.40000000e+02, 5.12000000e+03,
        4.09600000e+04, 3.27680000e+05, 2.62144000e+06, 2.09715200e+07,
        1.67772160e+08, 1.34217728e+09, 1.07374182e+10],
       [1.00000000e+01, 9.00000000e+01, 8.10000000e+02, 7.29000000e+03,
        6.56100000e+04, 5.90490000e+05, 5.31441000e+06, 4.78296900e+07,
        4.30467210e+08, 3.87420489e+09, 3.48678440e+10],
       [1.00000000e+01, 1.00000000e+02, 1.00000000e+03, 1.00000000e+04,
        1.00000000e+05, 1.00000000e+06, 1.00000000e+07, 1.00000000e+08,
        1.00000000e+09, 1.00000000e+10, 1.00000000e+11]])
```

```
# Predicting a new result with Linear Regression
lin_reg.predict([[11]])

# Predicting a new result with Polynomial Regression
lin_reg_2.predict(poly_reg.fit_transform([[6.5]]))
```

```
array([172421.91746118])
```

```
from sklearn.metrics import classification_report
# print(classification_report(Y_test,Y_pred))
print(classification_report(Y,Y))
```

```
              precision    recall  f1-score   support

       45000       1.00      1.00      1.00         1
       50000       1.00      1.00      1.00         1
       60000       1.00      1.00      1.00         1
       80000       1.00      1.00      1.00         1
      110000       1.00      1.00      1.00         1
      150000       1.00      1.00      1.00         1
      200000       1.00      1.00      1.00         1
      300000       1.00      1.00      1.00         1
      500000       1.00      1.00      1.00         1
     1000000       1.00      1.00      1.00         1

    accuracy                           1.00        10
```

```
   macro avg       1.00      1.00      1.00        10
weighted avg       1.00      1.00      1.00        10
```

20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

```
!git clone https://github.com/ksarvakar/Social_Network_Ads
```

```
Cloning into 'Social_Network_Ads'...
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (3/3), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), done.
```

AIM: https://drive.google.com/file/d/1ywXGXYuj9xCOm8sRVvzLgfjOchoTvkCV/view?usp=sharing

DATASET: https://drive.google.com/file/d/12px9dEfDwEPDI3SHjlplD7g18BtiQ072/view?usp=sharing

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('/content/Social_Network_Ads/Social_Network_Ads.csv')
X = dataset.iloc[:, [2,3]].values
Y = dataset.iloc[:, 4].values
```

```
dataset
```

	User ID	Gender	Age	EstimatedSalary	Purchased
0	15624510	Male	19	19000	0
1	15810944	Male	35	20000	0
2	15668575	Female	26	43000	0
...	...	...	...	...	...

```
# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
# Fitting logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, Y_train)
```

```
LogisticRegression(random_state=0)
```

```
# Predicting the Test set results
Y_pred = classifier.predict(X_test)
```

```
# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
```

```
cm
```

```
array([[65,  3],
       [ 8, 24]])
```

```
print((cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1]) * 100)
```

```
89.0
```

```
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.96      0.92        68
```

```
              1       0.89      0.75      0.81        32

       accuracy                           0.89       100
      macro avg       0.89      0.85      0.87       100
   weighted avg       0.89      0.89      0.89       100
```

K-NN

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

dataset = pd.read_csv('/content/Social_Network_Ads/Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
Y = dataset.iloc[:, 4].values
```

```
# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state = 0)
```

```
# MinMaxScaler
from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

p = 1, when p is set to 1 we get Manhattan distance p = 2, when p is set to 2 we get Euclidean distance

Double-click (or enter) to edit

```
# Fitting classifier to the training set
from sklearn.neighbors import KNeighborsClassifier
classifier = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
classifier.fit(X_train, Y_train)
```

```
KNeighborsClassifier()
```

```
# Predicting the Test set results
Y_pred = classifier.predict(X_test)
```

```
# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
```

```
print((cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1]) * 100)
```

```
93.0
```

```
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

```
                  precision    recall  f1-score   support

               0       0.96      0.94      0.95        68
               1       0.88      0.91      0.89        32

        accuracy                           0.93       100
       macro avg       0.92      0.92      0.92       100
    weighted avg       0.93      0.93      0.93       100
```

SVM

```
##### SVM
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

```
dataset = pd.read_csv('/content/Social_Network_Ads/Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
Y = dataset.iloc[:, 4].values

# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state = 0)

from sklearn.preprocessing import StandardScaler
sc_X = StandardScaler()
X_train = sc_X.fit_transform(X_train)
X_test = sc_X.transform(X_test)
```

```
# Fitting SVM to the training set
from sklearn.svm import SVC
classifier = SVC(kernel = 'linear', random_state = 0)
classifier.fit(X_train, Y_train)
```

```
SVC(kernel='linear', random_state=0)
```

```
# Predicting the Test set results
Y_pred = classifier.predict(X_test)
```

```
# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)

print((cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1]) * 100)
```

```
90.0
```

```
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

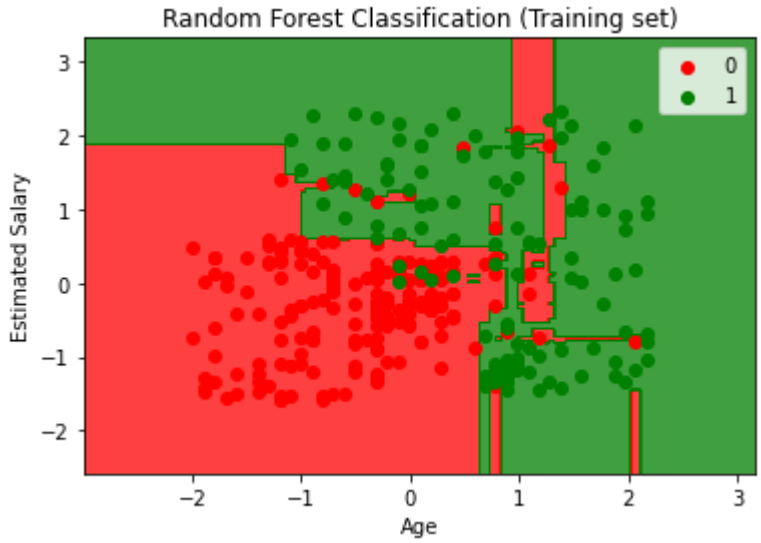
```
                  precision    recall  f1-score   support

               0       0.89      0.97      0.93        68
               1       0.92      0.75      0.83        32

        accuracy                           0.90       100
       macro avg       0.91      0.86      0.88       100
    weighted avg       0.90      0.90      0.90       100
```

```
# Random Forest Classification

# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

# Importing the dataset
dataset = pd.read_csv('/content/Social_Network_Ads/Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values

# # Splitting the dataset into the Training set and Test set
# from sklearn.cross_validation import train_test_split
```

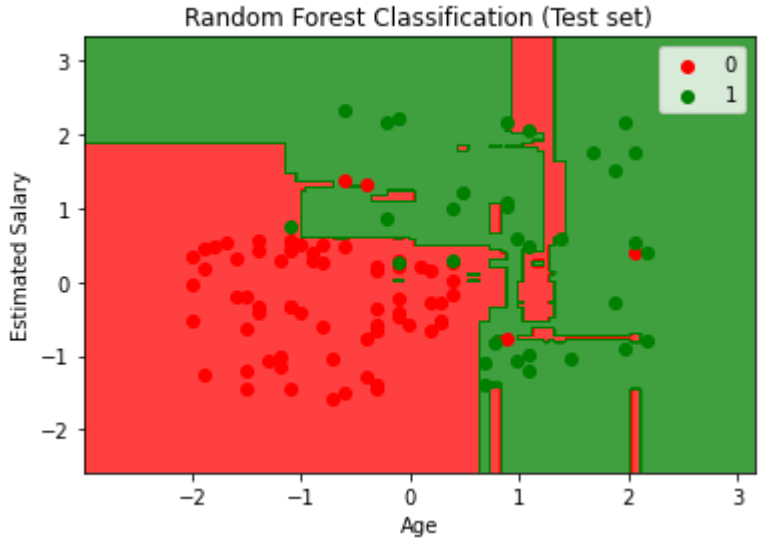
```
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.25, random_state =


# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.25, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)

# Fitting Random Forest Classification to the Training set
from sklearn.ensemble import RandomForestClassifier
classifier = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state =
classifier.fit(X_train, Y_train)

# Predicting the Test set results
y_pred = classifier.predict(X_test)

# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, y_pred)

# Visualising the Training set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_train, Y_train
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Training set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()

# Visualising the Test set results
from matplotlib.colors import ListedColormap
X_set, y_set = X_test, Y_test
X1, X2 = np.meshgrid(np.arange(start = X_set[:, 0].min() - 1, stop = X_set[:, 0].max() + 1, s
                     np.arange(start = X_set[:, 1].min() - 1, stop = X_set[:, 1].max() + 1, s
plt.contourf(X1, X2, classifier.predict(np.array([X1.ravel(), X2.ravel()]).T).reshape(X1.shap
             alpha = 0.75, cmap = ListedColormap(('red', 'green')))
plt.xlim(X1.min(), X1.max())
```

```
plt.ylim(X2.min(), X2.max())
for i, j in enumerate(np.unique(y_set)):
    plt.scatter(X_set[y_set == j, 0], X_set[y_set == j, 1],
                c = ListedColormap(('red', 'green'))(i), label = j)
plt.title('Random Forest Classification (Test set)')
plt.xlabel('Age')
plt.ylabel('Estimated Salary')
plt.legend()
plt.show()
```

```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```




```
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
*c* argument looks like a single numeric RGB or RGBA sequence, which should be avoided a
```



20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

Practical 4

To study and implement about Chi-Squared Test.

Chi-Squared Goodness-Of-Fit Test

```
import numpy as np
import pandas as pd
import scipy.stats as stats


national = pd.DataFrame(["white"]*100000 + ["hispanic"]*60000 +\
                        ["black"]*50000 + ["asian"]*15000 + ["other"]*35000)


minnesota = pd.DataFrame(["white"]*600 + ["hispanic"]*300 + \
                         ["black"]*250 +["asian"]*75 + ["other"]*150)

national_table = pd.crosstab(index=national[0], columns="count")
minnesota_table = pd.crosstab(index=minnesota[0], columns="count")

print( "National")
print(national_table)
print(" ")
print( "Minnesota")
print(minnesota_table)
```

```
     National
     col_0      count
     0
     asian      15000
     black      50000
     hispanic   60000
     other      35000
     white     100000

     Minnesota
     col_0     count
     0
     asian        75
     black       250
     hispanic    300
```

```
other       150
white       600
```

Chi-squared tests are based on the so-called chi-squared statistic. You calculate the chi-squared statistic with the following formula:

$sum((observed-expected)^2/expected)$

In the formula, observed is the actual observed count for each category and expected is the expected count based on the distribution of the population for the corresponding category. Let's calculate the chi-squared statistic for our data to illustrate:

```
observed = minnesota_table


national_ratios = national_table/len(national)


expected = national_ratios * len(minnesota)


chi_squared_stat = (((observed-expected)**2)/expected).sum()


print(chi_squared_stat)
```

```
     col_0
     count    18.194805
     dtype: float64
```

```
crit = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 4)   # Df = number of variable categories - 1

print("Critical value")
print(crit)

p_value = 1 - stats.chi2.cdf(x=chi_squared_stat,  # Find the p-value
                             df=4)
print("P value")
print(p_value)
```

```
     Critical value
     9.487729036781154
     P value
     [0.00113047]
```

```
stats.chisquare(f_obs= observed,   # Array of observed counts
                f_exp= expected)   # Array of expected counts
```

```
Power_divergenceResult(statistic=array([18.19480519]), pvalue=array([0.00113047]))
```

Chi-Squared Test of Independence

```
np.random.seed(10)

# Sample data randomly at fixed probabilities
voter_race = np.random.choice(a= ["asian","black","hispanic","other","white"],
                              p = [0.05, 0.15 ,0.25, 0.05, 0.5],
                              size=1000)

# Sample data randomly at fixed probabilities
voter_party = np.random.choice(a= ["democrat","independent","republican"],
                              p = [0.4, 0.2, 0.4],
                              size=1000)

voters = pd.DataFrame({"race":voter_race,
                       "party":voter_party})

voter_tab = pd.crosstab(voters.race, voters.party, margins = True)

voter_tab.columns = ["democrat","independent","republican","row_totals"]

voter_tab.index = ["asian","black","hispanic","other","white","col_totals"]

observed = voter_tab.iloc[0:5,0:3]   # Get table without totals for later use
voter_tab
```

	democrat	independent	republican	row_totals
asian	21	7	32	60
black	65	25	64	154
hispanic	107	50	94	251
other	15	8	15	38
white	189	96	212	497
col_totals	397	186	417	1000

```
expected =  np.outer(voter_tab["row_totals"][0:5],
                     voter_tab.loc["col_totals"][0:3]) / 1000

expected = pd.DataFrame(expected)

expected.columns = ["democrat","independent","republican"]
expected.index = ["asian","black","hispanic","other","white"]

expected
```

	democrat	independent	republican
asian	23.820	11.160	25.020
black	61.138	28.644	64.218
hispanic	99.647	46.686	104.667
other	15.086	7.068	15.846

```
chi_squared_stat = (((observed-expected)**2)/expected).sum().sum()

print(chi_squared_stat)
```

```
7.169321280162059
```

```
crit = stats.chi2.ppf(q = 0.95, # Find the critical value for 95% confidence*
                      df = 8)   # *

print("Critical value")
print(crit)

p_value = 1 - stats.chi2.cdf(x=chi_squared_stat,  # Find the p-value
                             df=8)
print("P value")
print(p_value)
```

```
Critical value
15.50731305586545
P value
0.518479392948842
```

```
stats.chi2_contingency(observed= observed)
```

```
(7.169321280162059, 0.518479392948842, 8, array([[ 23.82 ,  11.16 ,  25.02 ],
        [ 61.138,  28.644,  64.218],
        [ 99.647,  46.686, 104.667],
        [ 15.086,   7.068,  15.846],
        [197.309,  92.442, 207.249]]))
```

20012531022

ALAUKI PATEL

```
import pandas as pd
```

```
msg = pd.read_csv('/content/text_doc.CSV', names=['message', 'label'])
```

```
print("Total Instances of Dataset: ", msg.shape[0])
```

```
Total Instances of Dataset:  18
```

```
msg['labelnum'] = msg.label.map({'pos': 1, 'neg': 0})
```

```
X = msg.message
```

```
y = msg.labelnum
```

```
from sklearn.model_selection import train_test_split
```

```
Xtrain, Xtest, ytrain, ytest = train_test_split(X, y)
```

```
from sklearn.feature_extraction.text import CountVectorizer
```

```
count_v = CountVectorizer()
```

```
Xtrain_dm = count_v.fit_transform(Xtrain)
```

```
Xtest_dm = count_v.transform(Xtest)
```

```
df = pd.DataFrame(Xtrain_dm.toarray(),columns=count_v.get_feature_names())
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/deprecation.py:87: FutureWarning: F
  warnings.warn(msg, category=FutureWarning)
```

```
print(df[0:5])
```

```
   about  am  amazing  an  and  awesome  ...  tomorrow  very  we  went  will  with
```

```
     0      0   0       1   1    0       0  ...       0     0   0     0     0     0
     1      0   1       0   0    0       0  ...       0     0   0     0     0     0
     2      0   0       0   0    0       0  ...       0     0   0     0     0     1
     3      0   0       0   0    0       0  ...       1     0   1     0     1     0
     4      0   0       0   1    0       1  ...       0     0   0     0     0     0

     [5 rows x 45 columns]
```

```
from sklearn.naive_bayes import MultinomialNB
```

```
clf = MultinomialNB()
```

```
clf.fit(Xtrain_dm, ytrain)
```

```
     MultinomialNB()
```

```
pred = clf.predict(Xtest_dm)
```

```
for doc, p in zip(Xtrain, pred):
    p = 'pos' if p == 1 else 'neg'
    print("%s -> %s" % (doc, p))
```

```
     This is an amazing place -> pos
     I am tired of this stuff -> neg
     I can't deal with this -> neg
     We will have good fun tomorrow -> neg
     This is an awesome place -> neg
```

```
from sklearn.metrics import accuracy_score, confusion_matrix, precision_score, recall_score
```

```
print('Accuracy Metrics: \n')
print('Accuracy: ', accuracy_score(ytest, pred))
```

```
     Accuracy Metrics:

     Accuracy:  0.6
```

```
print('Recall: ', recall_score(ytest, pred))
```

```
     Recall:  0.3333333333333333
```

```
print('Precision: ', precision_score(ytest, pred))
```

```
     Precision:  1.0
```

```
print('Confusion Matrix: \n', confusion_matrix(ytest, pred))
```

```
Confusion Matrix:
 [[2 0]
 [2 1]]
```

20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

Using Machine Learning KNN (K-Nearest Neighbors) to Solve Problems

The main objective of this article is to demonstrate the the best practices of solving a problem through the surpervioned machine learning algorithm **KNN** (K-Nearest Neighbors).

To comply with this goal the **IRIS** dataset is used, a very common dataset for data scientists for tests and studies in **ML** (Machine Learning). Furthermore, this is a built-in dataset for ***Scikit-Learn*** or ***Seaborn*** library.

Iris Dataset

- Iris is a genus of species of flowering plants with showy flowers.
- The iris species here studied, **setosa**, **versicolor** and **virginica** share a similar color: dark violet/blue.
- A way to distinguish visually the species among them is through their petals and sepals dimensions (lenght and width).

> **The first step is to import *Seaborn* Library and then load the iris dataset from it.**

Differently from the Scikit-Learn that loads the dataset as arrays, Seaborn load it as Panda's Dataframe what saves some steps to build the dataframe, and so, some lines of codes.

```
import seaborn as sns
iris = sns.load_dataset("iris")
```

```
type(iris)
```

```
pandas.core.frame.DataFrame
```

The method **sample** of a Panda's Dataframe is great way to visualize how the data is stored, like the features and the kind of data of them, with random samples of the Dataframe, non ordered.

```
iris.sample(5)
```

	sepal_length	sepal_width	petal_length	petal_width	species
85	6.0	3.4	4.5	1.6	versicolor
120	6.9	3.2	5.7	2.3	virginica
41	4.5	2.3	1.3	0.3	setosa
117	7.7	3.8	6.7	2.2	virginica
145	6.7	3.0	5.2	2.3	virginica

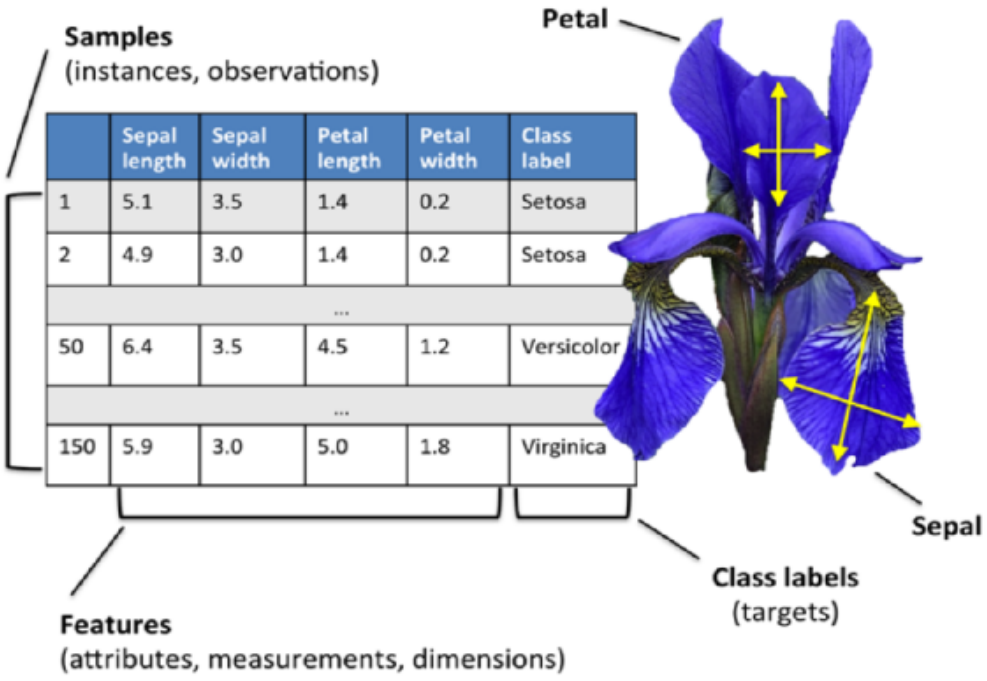
There are 4 categories listed: "sepal length (cm)", "sepal width (cm)", "petal length (cm)" and "petal width (cm)".

To better understand the dataset look to the picture below.

> **This method is specific for Colab to import files, in this case a picture.**

```
from google.colab import files
iris_image = files.upload()
```

```
from IPython.display import Image
Image(filename ="iris-data-set.png", width=500, height=350)
```



> **To perform some extra checks to the Dataframe, the Pandas library itself must be imported.** Before starting any more in-deep work on any data it is essencial to do a prelimiary analysis of the data: the types of data, its sizes, some statistical figures, any missing values. Then, take the first conclusions and correct any issue* in appropriate way.

*this is not the case of this well-known data but the checks are to be performed anyway.

```
import pandas as pd
```

```
pd.value_counts(iris.species)
```

```
setosa        50
versicolor    50
virginica     50
Name: species, dtype: int64
```

The dataset is made of 150 flowers (150 rows), 50 of each specie.

> **The statistical basic characteristics of the numeric columns.**

Through **describe** method the average, standard deviation, max and min, and the percentiles for each of the features.

```
iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

It is clear each feature constrast with the others. However, at this time the data of the 3 species is

The method **info** gives extra information about the dataframe and the type of data for each column:

- the features that are dimension of type float64
- the target, or species, of type object

```
iris.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
sepal_length    150 non-null float64
sepal_width     150 non-null float64
petal_length    150 non-null float64
petal_width     150 non-null float64
species         150 non-null object
dtypes: float64(4), object(1)
memory usage: 6.0+ KB
```

The method **isnull** combined with the method **sum** returns if there is any null value in the dataframe that should be managed, otherwise, could imply in the impossibility of applying any machine learning algorithm.

```
iris.isnull().sum()
```

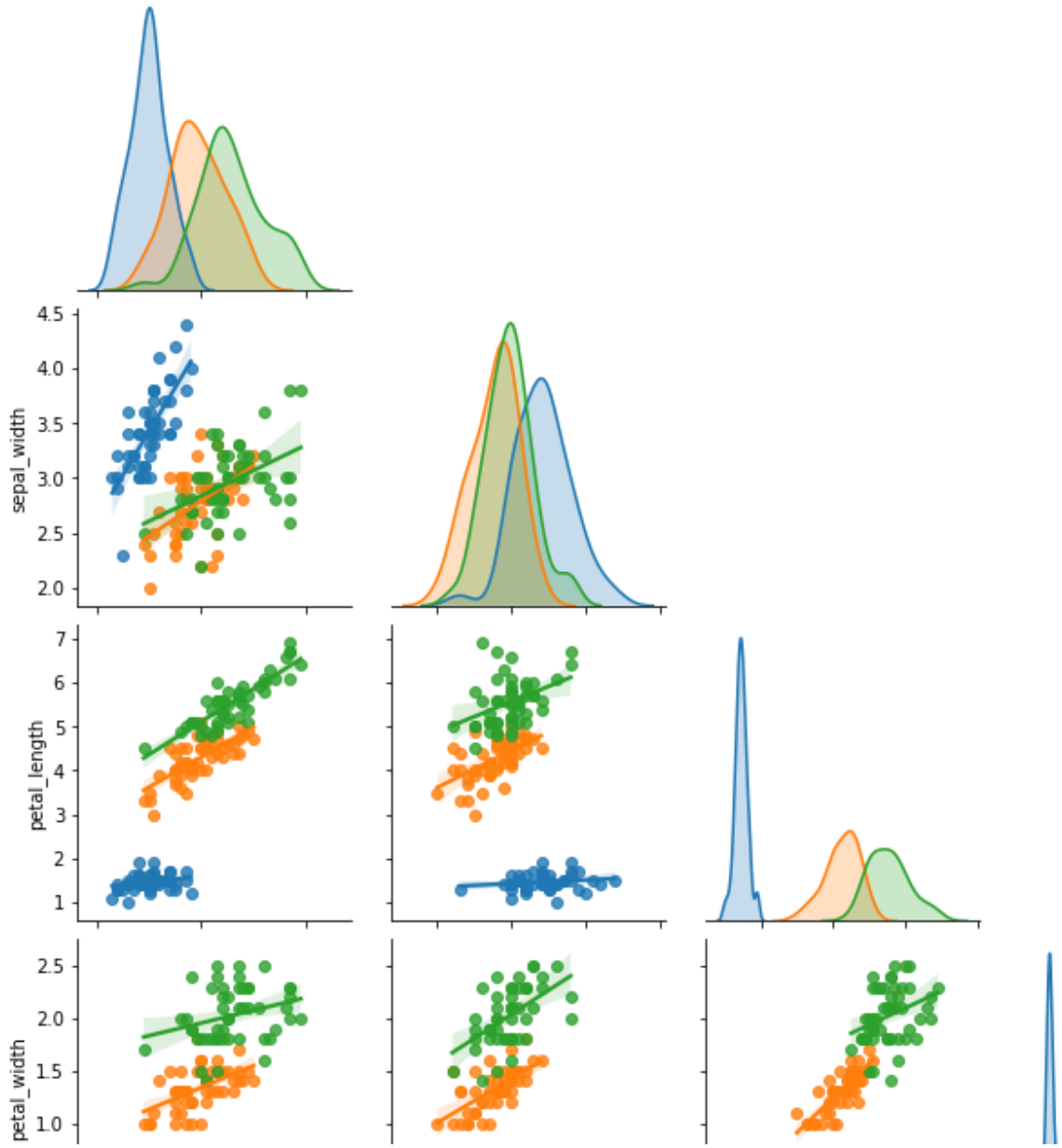
```
sepal_length    0
sepal_width     0
petal_length    0
petal_width     0
species         0
dtype: int64
```

Exploratory Analysis through Charts

Seaborn is a Python data visualization library based on **Matplotlib**. It provides a high-level interface for drawing attractive and informative statistical graphics.

```
sns.pairplot(iris, hue="species", corner=True, kind='reg')
```

```
<seaborn.axisgrid.PairGrid at 0x7f34b34a9b70>
```



Setosa Identification

Iris-setosa, **iris-virginica** and **iris-versicolor** have the same violet/blue color(sorry, my genre can't distinguish one from another). A way to distinguish the species is analyzing its petal and sepal dimensions and the proportions these them. Along with an exploratory analysis of the data it is possible to get valuable information and some conclusions. Follows.

As seen in the charts above, the data from **iris-setosa** is very distinguish from the other two species. With a data distribution like this, it is expected from any good predictive machine learning algorithm a high accuracy, maybe 100%. Or better, for the classification of this specie against the other two, **simply check it petal dimensions**; if its length is up to 2 cm or its width is up to 0.7 cm it is for sure iris-setosa.

[ ] ↳ *2 cells hidden*

Virginica and Versicolor Identification

From the other two species. The data from **iris-versicolor** and **iris-virginica**, at the same time they are close, it is notorious a clear normal value for all its dimensions, except for the sepal width that is almost the same for both species. Due to this, it is possible to distinguish with a high percentage of accuracy when referring to the following relationships:

- Petal Length **vs** Sepal Length or Sepal Width
- Petal Width **vs** Sepal Width or Sepal Width or Petal Length

The only relationship that is almost the same, and could not, through an exploratory analysis conclude which specie is between this two, as the dimensions and proportion between length and width, are quite the same is:

- Sepal Length **vs** Sepal Width

An algorithm of Machine Learning is of great use for this classification, reducing any human mistake when comparing all this dimensions and relationships.

Both algorithms are used for the classification in this issue are optimal for this kind of data analysis.

They have some differences that could be said are simply their characteristics.

[ ] ↳ *2 cells hidden*

MACHINE LEARNING - Predictive Analysis

After **performing the exploratory** analysis, what is basically, evaluate and reaching some conclusions looking to the charts of distribution and scatter of the data.

Now, the predictive analysis is going to be performed not by a person itself but with the aid of a computer, mathematical algorithms, **machine learning**.

The problem is going to be solved through the **KNN** algorithm. The best pratices are going to be performed and explained.

Scikit-learn is the library used here for machine learning in Python. From it several modulus are imported as needed.

> *Scikit-learn is an open source machine learning library that supports supervised and unsupervised learning. It also provides various tools for model fitting, data preprocessing, model selection and evaluation, and many other utilities.*

TRAIN and TEST Data

The first step to perform a predictive analysis through a Machine Learning algorithm is to separate the data in two parts. One to be **trained**, so the coeficients are calculated to the best fit to this data, according to the algorithm chosen. The second part is reserved to be **tested**, so it is possible to ***evaluate the accuracy*** of the method with a different data, so that you can extrapolate to any data.

> This selection, of which part of the data is to be used as train and test is very important. We are going to use **70% as train** and **30% as test**. It is safe not to pick any skewed data to make it randomly.

> ***Scikit-learn*** has a very good method for this, **train_test_split**; however, everytime you run this method it returns different samples as it makes it ramdomly, what would make the same problem impossible to replicate. The selection of the data can produces slight different results to the algorithm.**

> ***Panda*** has a solution for this, through a seeded random split of the database. The method **sample** set with **random_state=500** would meake it replicable by anyone. You are free to test with a different number, or even with *train_test_split*.

> The final result is:

>> - **X_train** and **y_train -** for training, data and target, respectively
>> - **X_test -** data for predicting with the algorithm
>> - **y_test -** target for validation the model

```
train = iris.sample(frac=0.7, random_state=500) # random state is a seed value
```

```
test = iris.drop(train.index)
```

```
X_train = train.drop(labels='species', axis=1)
```

```
y_train = train.species
```

```
X_test = test.drop(labels='species', axis=1)
```

```
y_test = test.species
```

> **later was checked in the documentation, and the train_test_split method also has a seeded random split method, yet not the same, so same seed represents different data from Panda's sample method.

> Follows the option to use the train_test_split: simple, one line code:

> > **from** sklearn.model_selection **import** train_test_split

> > X_train, X_test, y_train, y_test = train_test_split (iris.drop ('species' ,axis = 1) , iris ['species'] , test_size = 0.3, random_state = 1)

> Following is a sample of the first 5 rows of the train data after being splitted.

- Every time the *train_test_split* method is run a *different sample* for test and train is selected as it works randomly with the data. Like this, everytime you have a different train data the performance can differ. To minimize this issue some techniques for training/testing with different parts of the dataset is recommended; this is called **cross validation** and is going to be used further ahead.

```
print(X_train.head(),"\n", y_train.head())
```

```
     sepal_length  sepal_width  petal_length  petal_width
7             5.0          3.4           1.5          0.2
126           6.2          2.8           4.8          1.8
136           6.3          3.4           5.6          2.4
133           6.3          2.8           5.1          1.5
66            5.6          3.0           4.5          1.5
 7          setosa
126     virginica
136     virginica
133     virginica
66     versicolor
Name: species, dtype: object
```

> Following, the shapes of the splitted data.

```
X_train.shape,X_test.shape
```

```
((105, 4), (45, 4))
```

105 rows and 4 columns for the train against 45 rows (30%) and 4 columns for the test.

```
y_train.shape,y_test.shape
```

```
((105,), (45,))
```

105 rows for the train against 45 rows for the test, to the targets.

KNN - K Nearest Neighbors*

In pattern recognition, the k-nearest neighbors algorithm (k-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the k closest training examples in the feature space. The output depends on whether k-NN is used for classification or regression:

> - In k-NN classification, the output is a class membership. An object is classified by a plurality vote of its neighbors, with the object being assigned to the class most common among its k nearest neighbors (k is a positive integer, typically small). If k = 1, then the object is simply assigned to the class of that single nearest neighbor.

*from Wikipedia.

Instantiating the KNN Algorithm

The KNN Classifier (KNeighborsClassifier) is imported from Scikit-learn.

The **KNeighborsClassifier** has some parameters to improve to improve its performance. At first only n_neighbors is going to be set, the others are to be as default. Later, an optimization analysis could be performed to adjust them.

[ ] ↳ *3 cells hidden*

Trainning the KNN algorithm

At this step the algorithm is trained to estimate the best representative function for the data, according to the parameters being used.

Fit is the method to adjust the classifier to the data.

```
knn.fit(X_train,y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

Before predicting the results, it is possible to have the estimate mean accuracy on the given test data and labels.

```
knn.score(X_test,y_test)
```

```
0.9555555555555556
```

Predicting with the Test Data and the KNN Algorithm

With the algorithm trained and the function stablished it is possible to get the first results predicting against the test data (X_test).

Predict is the method to infer results from any data compatible to the ones of the train.

[ ] ↳ *2 cells hidden*

Validation

Technics to to calculate and improve the accuracy and other performance metrics.

Cross Tabulation

[ ] ↳ *3 cells hidden*

Classification Metrics

[ ] ↳ *4 cells hidden*

Cross Validation

As discussed, previously, in the **Train and Test Data**, the selection of data to be used for these two important parts of supervisioned machine learning can alter the efficiency of the model. In other words, different trainnind data mean a different fitted algorithm that gives in different prediction that could lead to different metrics.

This, only varying the selected samples, all parameters remaining the same.

To minimize this issue, a good solution is to use the **Cross Validation Method**, which splits the training data into **n** folds (in this case n = 5), and repeats the analysis of **fitting** and **predicting n** times, using at each time the amount of data reduced by one folder.

It is almost the same it was selected different data for the trainning, however, a smaller one. For more details on how this method works, consult the official documentation.

It optimizes the choosen metrics (in this case, accuracy) reducing the risk of **overfitting** when the classifier works wonderful with the trained data but not with the test by **averaging** the resultant metrics of eacho loop.

[ ] ↳ *7 cells hidden*

Optimizing the Parameter K (n_neighbors)

Defining a List for Parameter K

It is possible with the **GridSearch Objetc** to optimize the parameters of a classifier.

For the classification of the best parameters the Cross Validation Metrics shows up as a great metrics for the reasons formerly explained.

[ ] ↳ *3 cells hidden*

Instantiating the GridSearch Object

The **GridSearch Objetc** performs an exhaustive search over specified parameter values for an estimator.

Its process performs in-built **fit** and **predict** methos, together with the **Cross Validation** Strategie.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

[ ] ↳ *2 cells hidden*

Training the GridSearch Object

[ ] ↳ *1 cell hidden*

Checking for the Scores

Time to check the performance of the classifier over all the parameters listed to the GridSearch Objetc.

[ ] ↳ *10 cells hidden*

Charts

Graphically, the list of results of the k number of neighbors versus the accuracy.

[ ] ↳ *2 cells hidden*

Conclusions and Next Steps

The issue is exaustive, other parameters should be tested with the **GridSearch Object**. However this database is very simple and these changes could indicate no differ.

For sure, the best way for performing the optimization with these sensibility analysis, is through **Pipelines**.

The next article is programmed to be about the **SVM (Support Vectors Machine)** Machine Learning Algorithm for classifiation, where I would solve the same problem discussed herein in a shorter way going direct to the classifier laying aside the exploratory analysis and the sample selection, accomplishing the optimization along Pipelines.

Later, a comparison between these two approaches (**KNN x SVM**) is of great concern, to confront which performs better to this problem. Or at least, more appropriately. Checking how both works.

Acknowledgment

This article had the collaboration of some data scientist colleagues:

- Karinne Cristina, Tutor of Minerando's Data Science do Zero
- Maykon Schots, Data Scientist at Volvo Group
- Plínio Mendes, Software Engineer at Google
- Rodringo Santana, Co-Founder of Minerando Dados

20012531022

20012531022

ALAUKI PATEL

```
!git clone https://github.com/ksarvakar/data.git
```

```
Cloning into 'data'...
remote: Enumerating objects: 7, done.
remote: Counting objects: 100% (7/7), done.
remote: Compressing objects: 100% (7/7), done.
remote: Total 7 (delta 1), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (7/7), done.
```

```
import pandas as pd
dataset = pd.read_csv('/content/Market_Basket_Optimisation (1).csv', header = None)
```

```
dataset
```

	0	1	2	3	4	5	6	7	8	
0	shrimp	almonds	avocado	vegetables mix	green grapes	whole weat flour	yams	cottage cheese	energy drink	tom ju
1	burgers	meatballs	eggs	NaN	NaN	NaN	NaN	NaN	NaN	N
2	chutney	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
3	turkey	avocado	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
4	mineral water	milk	energy bar	whole wheat rice	green tea	NaN	NaN	NaN	NaN	N
...	...	...	...	...	...	...	...	...	...	
7496	butter	light mayo	fresh bread	NaN	NaN	NaN	NaN	NaN	NaN	N
7497	burgers	frozen vegetables	eggs	french fries	magazines	green tea	NaN	NaN	NaN	N
7498	chicken	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7499	escalope	green tea	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
7500	eggs	frozen smoothie	yogurt cake	low fat yogurt	NaN	NaN	NaN	NaN	NaN	N

7501 rows × 20 columns

```
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i, j]) for j in range(0, 20)])
```

```
transactions
```

```
#import numpy as np
#import matplotlib.pyplot as plt
import pandas as pd
# https://drive.google.com/drive/u/0/folders/1hTHQL5bUJuIiIU7rJ0qRLwekWXNfTj4M
# https://towardsdatascience.com/underrated-machine-learning-algorithms-apriori-1b1d7a8b7bc

dataset = pd.read_csv('/content/Market_Basket_Optimisation (1).csv', header = None)
transactions = []
for i in range(0, 7501):
    transactions.append([str(dataset.values[i, j]) for j in range(0, 20)])

# Training Apriori on the dataset
from apyori import apriori
# !pip install apyori
rules = apriori(transactions, min_support = 0.003, min_confidence = 0.2, min_lift = 3, min_le

# Visualising the result
results = list(rules)
results_list = []
for i in range(0, len(results)):
    results_list.append('RULE:\t' + str(results[i][0]) + '\nSUPPORT:\t' + str(results[i][1])
```

```
transactions
```

```
results_list
```

```
["RULE:\tfrozenset({'chicken', 'light
cream'})\nSUPPORT:\t0.004532728969470737\n\t[OrderedStatistic(items_base=frozenset({'
 cream'}), items_add=frozenset({'chicken'}), confidence=0.29059829059829057,
lift=4.84395061728395)]",
 "RULE:\tfrozenset({'escalope', 'mushroom cream
sauce'})\nSUPPORT:\t0.005732568990801226\n\t[OrderedStatistic(items_base=frozenset({'
 cream sauce'}), items_add=frozenset({'escalope'}), confidence=0.3006993006993007,
lift=3.790832696715049)]",
 "RULE:\tfrozenset({'pasta',
'escalope'})\nSUPPORT:\t0.005865884548726837\n\t[OrderedStatistic(items_base=frozense
 items_add=frozenset({'escalope'}), confidence=0.3728813559322034,
lift=4.700811850163794)]",
 "RULE:\tfrozenset({'honey', 'fromage
blanc'})\nSUPPORT:\t0.003332888948140248\n\t[OrderedStatistic(items_base=frozenset({'
 blanc'}), items_add=frozenset({'honey'}), confidence=0.2450980392156863,
lift=5.164270764485569)]",
 "RULE:\tfrozenset({'ground beef', 'herb &
pepper'})\nSUPPORT:\t0.015997866951073192\n\t[OrderedStatistic(items_base=frozenset(
```

20012531022

ALAUKI PATEL

Practical -8

Aim :- Implementation of Confusion - Matrix for Multi-Class Machine Learning Model.

```
#importing packages
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
#Importing of dataset to dataframe.
df = pd.read_csv("/content/drive/MyDrive/Dataset/IRIS.csv")
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
#To see first 5 rows of the dataset
df.head()
#To know the data types of the variables.
df.dtypes
```

```
sepal_length    float64
sepal_width     float64
petal_length    float64
petal_width     float64
species          object
dtype: object
```

```
#Speceis is the output class,to know the count of each class we use value_counts()
df['species'].value_counts()
```

```
Iris-setosa        50
Iris-versicolor    50
Iris-virginica     50
Name: species, dtype: int64
```

```
#Separating independant variable and dependent variable("Species")
X = df.drop(['species'], axis=1)
y = df['species']
# print(X.head())
print(X.shape)
# print(y.head())
print(y.shape)
```

```
(150, 4)
(150,)
```

```
# Splitting the dataset to Train and test
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
#to know the shape of the train and test dataset.
print(X_train.shape)
print(y_train.shape)
print(X_test.shape)
print(y_test.shape)
```

```
(105, 4)
(105,)
(45, 4)
(45,)
```

```
#We use Support Vector classifier as a classifier
from sklearn.svm import SVC
from sklearn.metrics import confusion_matrix
```

```
#training the classifier using X_Train and y_train
clf = SVC(kernel = 'linear').fit(X_train,y_train)
clf.predict(X_train)
```

```
array(['Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-virginica',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-virginica', 'Iris-virginica', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-versicolor', 'Iris-virginica', 'Iris-virginica',
       'Iris-setosa', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-versicolor',
```

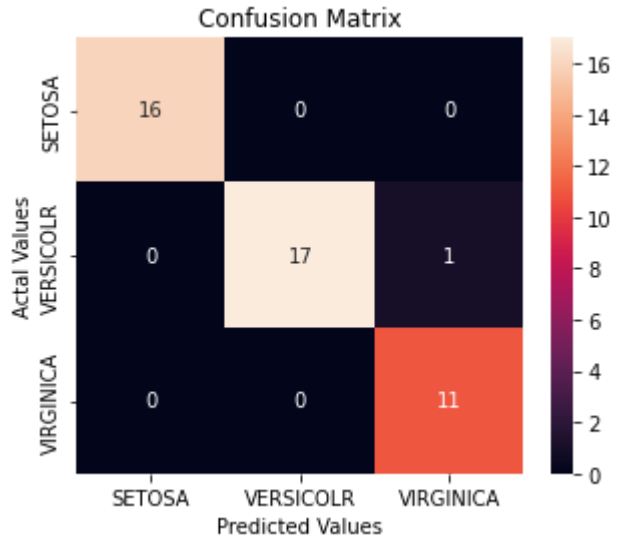
```
       'Iris-setosa', 'Iris-virginica', 'Iris-versicolor',
       'Iris-virginica', 'Iris-versicolor', 'Iris-setosa',
       'Iris-virginica', 'Iris-setosa', 'Iris-virginica', 'Iris-setosa',
       'Iris-setosa', 'Iris-virginica', 'Iris-setosa', 'Iris-virginica',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-versicolor',
       'Iris-virginica', 'Iris-virginica', 'Iris-setosa',
       'Iris-versicolor', 'Iris-versicolor', 'Iris-virginica',
       'Iris-versicolor', 'Iris-setosa', 'Iris-setosa', 'Iris-setosa',
       'Iris-virginica', 'Iris-versicolor', 'Iris-virginica',
       'Iris-setosa'], dtype=object)
```

```
#Testing the model using X_test and storing the output in y_pred
y_pred = clf.predict(X_test)
```

```
# Creating  a confusion matrix,which compares the y_test and y_pred
cm = confusion_matrix(y_test, y_pred)
```

```
# Creating a dataframe for a array-formatted Confusion matrix,so it will be easy for plott
cm_df = pd.DataFrame(cm,index = ['SETOSA','VERSICOLR','VIRGINICA'],columns = ['SETOSA','VE
```

```
#Plotting the confusion matrix
plt.figure(figsize=(5,4))
sns.heatmap(cm_df, annot=True)
plt.title('Confusion Matrix')
plt.ylabel('Actal Values')
plt.xlabel('Predicted Values')
plt.show()
```



20012531022

ALAUKI PATEL

Covered following things

1.Exploring the Dataset

2.Vizualizing the Data Set

3.K Means Algorithm | Elbow Method | Customer Segmentation

4.Relation between Age and Spending Score

5.Relation between Age and Income

6.Relation between Age,Annual Income and Spending Score

How to use Data Science to better understand your customers

How much prominence do customers hold in your business layout? Well, this was a rhetorical question. We all know that the majority of businesses thrive only because of their customers. Therefore it is imperative that you understand your customers well before serving them. Knowing your customers helps you provide tailored services. This results in enhanced customer engagement and increased sales.

Do you know your customers? Well, this question is very vague. If you are not able to answer this with certain qualitative aspects of your customers, then you need to get to work now. I am sure that all business owners have an image of their ideal customer in their minds, regardless of how obscure it may be. Often this image is fabricated from intuition. It may not be supported by any tautological evidence.

Data never lies. It is nothing more than a collection of facts and figures, and at times it can show us a mirror. This article will explain how to use the “magic” of data science to gain a coherent understanding of your customers. Precisely, we will learn how to apply a clustering algorithm to this mall customer dataset. We will then draw inferences from the output to gain a better understanding of the customers that frequent the mall. Thank you for bearing with such a lengthy prelude, and you get the project source code for your patience.

What is customer bucketing? Customer segmentation or customer bucketing is the practice of dividing a company’s customers into groups (a.k.a. buckets) that reflect similarity among

customers in each group. The goal of segmenting customers is to decide how to relate to customers in each segment in order to maximize the value of each customer to the business. Bucketing customers enables you to cater to each customer group in a way that can maximise your sales. For marketers, segmenting your target customers allows you to shape your communications in a way that causes maximum impact. In this project, we will use cluster analysis to segment customers into clusters based on their annual income. For this, we will use Kmeans, which is one of the finest clustering algorithms out there. K-means clustering is an unsupervised learning algorithm which finds groups in data. The number of groups is represented by the letter K.

```
#KETAN SARVAKAR DEMO PROJECT

import numpy as np # linear algebra
import matplotlib.pyplot as plt # data processing, CSV file I/O (e.g. pd.read_csv)
import pandas as pd

import warnings
import seaborn as sns
```

Peeking at the Data. The mall customer dataset is a relatively small dataset as it contains only 200 rows and 5 columns. If you glance at the picture below this paragraph, then you will notice that these five columns titles are CustomerID, Genre, Age, Annual Income (k$), and Spending Score (1–100).

```
data = pd.read_csv('/content/Mall_Customers.csv')
# https://www.kaggle.com/ksarvakar/mall-customers  Download it
X = data.iloc[:, [3,4]].values
```

In this project, we will be clustering the customers using their annual income and their spending score (between 1 and 100). Therefore we will only be using those two columns.

```
data.head(4)
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77

So in the data set we have the Customer Id, Genre, Age, Annual Income and the Spending Score of the customer making purchase at the mall.Spending score of 1 means less purchase and 100 means more purchase being done by the customer

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 5 columns):
 #   Column                  Non-Null Count  Dtype
---  ------                  --------------  -----
 0   CustomerID              200 non-null    int64
 1   Genre                   200 non-null    object
 2   Age                     200 non-null    int64
 3   Annual Income (k$)      200 non-null    int64
 4   Spending Score (1-100)  200 non-null    int64
dtypes: int64(4), object(1)
memory usage: 7.9+ KB
```

```
data.info()
```

	CustomerID	Age	Annual Income (k$)	Spending Score (1-100)
count	200.000000	200.000000	200.000000	200.000000
mean	100.500000	38.850000	60.560000	50.200000
std	57.879185	13.969007	26.264721	25.823522
min	1.000000	18.000000	15.000000	1.000000
25%	50.750000	28.750000	41.500000	34.750000
50%	100.500000	36.000000	61.500000	50.000000
75%	150.250000	49.000000	78.000000	73.000000
max	200.000000	70.000000	137.000000	99.000000

```
df= data.describe().loc['mean']
df
```

```
CustomerID               100.50
Age                       38.85
Annual Income (k$)        60.56
Spending Score (1-100)    50.20
Name: mean, dtype: float64
```

1.Mean Age of the cusomers is 38.85

2.Mean Annual Income 60.56 $

3.Mean Spening Score is 50.20

```
data.tail()
```

	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
195	196	Female	35	120	79
196	197	Female	45	126	28
197	198	Male	32	126	74
198	199	Male	32	137	18
199	200	Male	30	137	83

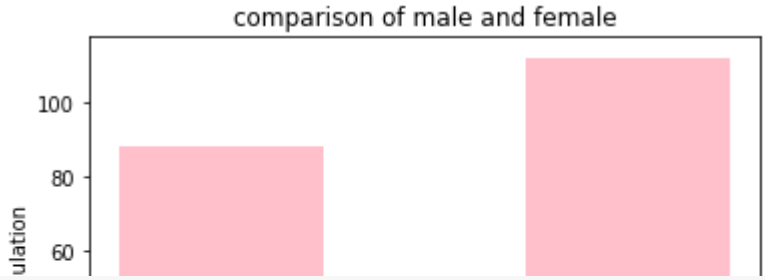
```
data.isnull().sum()
```

```
CustomerID                0
Genre                     0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

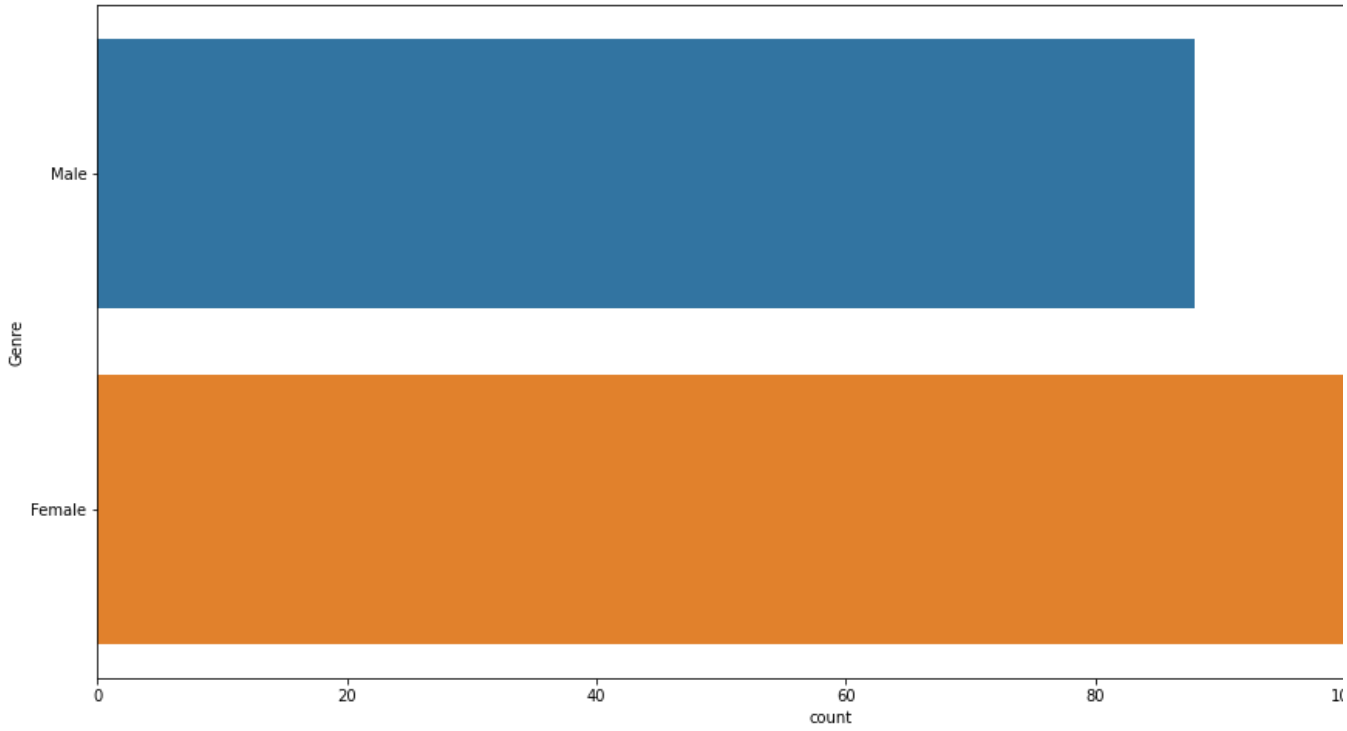
```
data['Genre'].value_counts()
```

```
Female    112
Male       88
Name: Genre, dtype: int64
```

```
# Plotting the features of the dataset to see the correlation between them

plt.hist(x = data.Genre, bins = 3, color = 'pink')
plt.title('comparison of male and female')
plt.xlabel('Genre')
plt.ylabel('population')
plt.show()
```

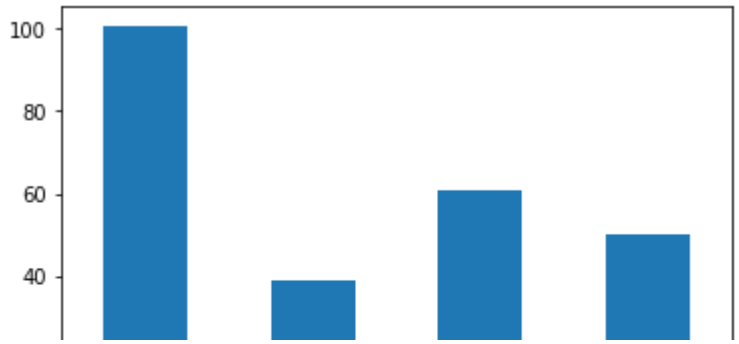



```
plt.figure(1 , figsize = (17 , 8))
sns.countplot(y = 'Genre' , data = data)
plt.show()
```




```
data.describe().loc['mean'].plot.bar()
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f01d4788410>
```



Mean of custmer if is not important as te value doesnt have any signifance.

This plot would be very useful to compare he features which have same units.

```
plt.figure(1 , figsize = (17 , 8))
n = 0
for x in ['Age' , 'Annual Income (k$)' , 'Spending Score (1-100)']:
    n += 1
    plt.subplot(1 , 3 , n)
    sns.distplot(data[x] , bins = 20)
    plt.title('Distplot of {}'.format(x))
plt.show()
```

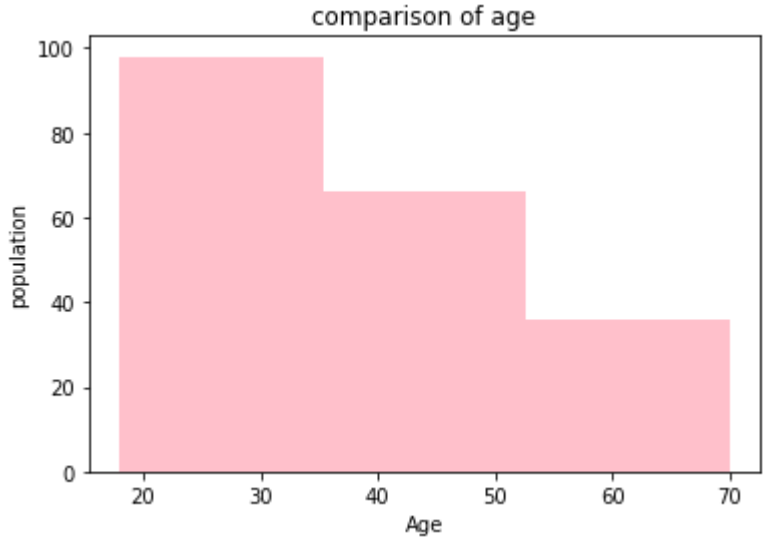
```
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
/usr/local/lib/python3.7/dist-packages/seaborn/distributions.py:2557: FutureWarning: `di
  warnings.warn(msg, FutureWarning)
```


```
data['Age'].value_counts()
```

```
32    11
35     9
19     8
31     8
30     7
49     7
27     6
47     6
40     6
23     6
36     6
38     6
50     5
48     5
29     5
21     5
20     5
34     5
18     4
28     4
59     4
24     4
67     4
54     4
39     3
25     3
33     3
22     3
37     3
43     3
68     3
45     3
46     3
60     3
41     2
57     2
66     2
65     2
63     2
58     2
26     2
70     2
42     2
```

```
53     2
52     2
51     2
44     2
55     1
64     1
69     1
56     1
Name: Age, dtype: int64
```

```
# Plotting the features of the dataset to see the correlation between them

plt.hist(x = data.Age, bins = 3, color = 'pink')
plt.title('comparison of age')
plt.xlabel('Age')
plt.ylabel('population')
plt.show()
```




```
data['Annual Income (k$)'].value_counts()
```

```
54     12
78     12
60      6
87      6
62      6
       ..
61      2
126     2
59      2
58      2
15      2
Name: Annual Income (k$), Length: 64, dtype: int64
```

```
# Using the elbow method to find the optimal number of clusters
from sklearn.cluster import KMeans
```

```
wcss = []
for i in range (1, 11):
    kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_s
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
```

```
#Let's predict the x
y_kmeans = kmeans.fit_predict(X)
```

```
print(y_kmeans)
#We convert our prediction to dataframe so we can easily see this prediction in table form
df_pred = pd.DataFrame(y_kmeans)
df_pred.head()
```

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

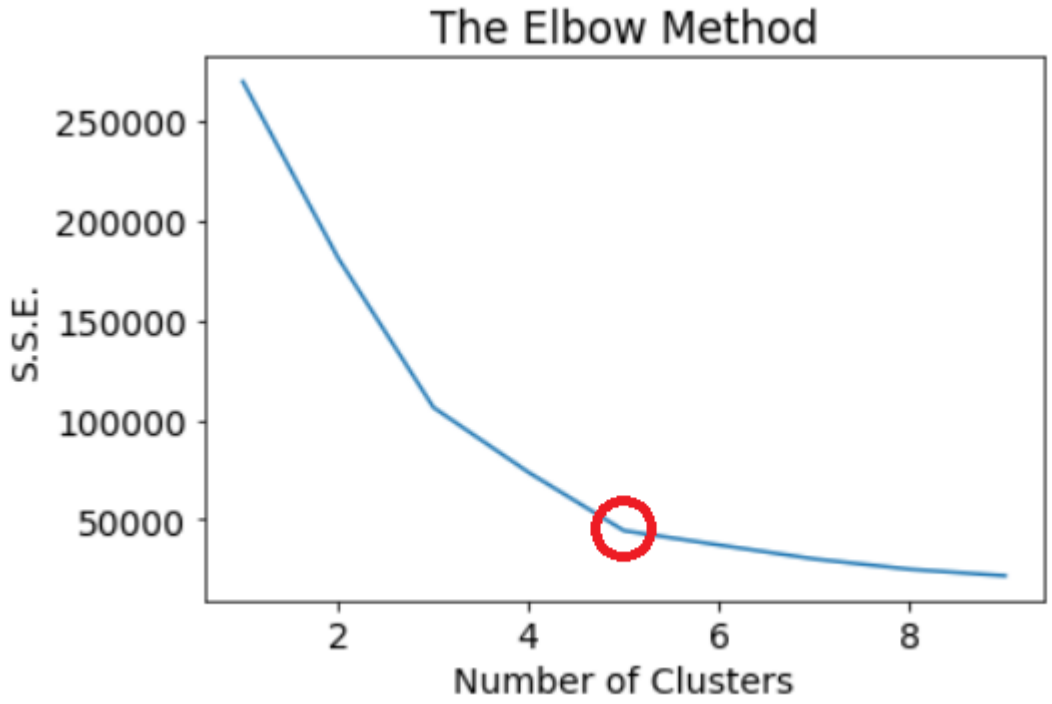
	0
0	7
1	0
2	4
3	0
4	7

Now that we are all set on the data front, it is time to start with our clustering. Before we run our clustering algorithm, it is imperative to determine the number of clusters to divide our customers into. There are a few different methods of determining the ideal number of clusters for this dataset. For that, we will be using the **elbow method.**

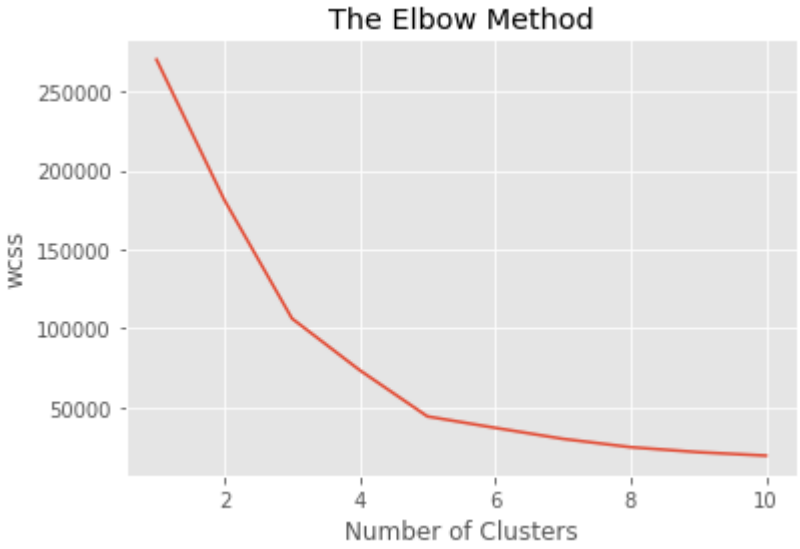
Elbow method

One method to figure out the number of clusters is by using the elbow method. This method involves running the K-means clustering algorithm on the data for different values of K and calculating the Sum of Squared Errors (S.S.E.) for each value of K. Then, these values are plotted on a graph, and we can see that S.S.E. tends to decrease as the value of K increases. S.S.E. becomes 0 when the values of K is equal to the number of data points, because then each data point is its own cluster. Our aim is to find a point with a small value of K and that has a low S.S.E. In this experiment,

we will run K-means for different values of K in the range of 0 to 10 and store the S.S.E. in a list



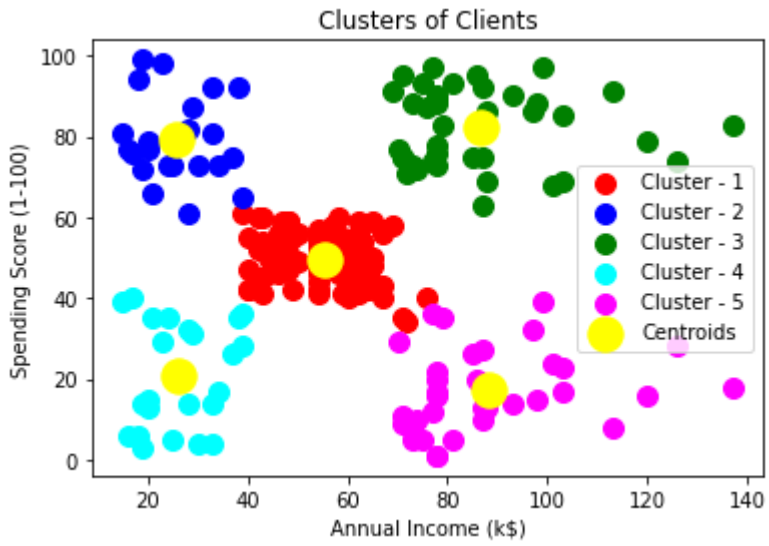

```
plt.plot(range(1,11), wcss)
plt.title('The Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('wcss') #Within-Cluster-Sum-of-Squares
plt.show()
```



In this graph, you can observe that the S.S.E. steeply drops after every iteration of K.

You can also observe that after **K reaches 5**, it is one downhill slope from there. So, 5 seems to be an optimum value for K, and this means that we will be dividing the customers into 5 clusters. Now that we have figured out the number of clusters we can go ahead and create these clusters.

```
# Applying k-means to the mall dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_state
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans==0, 0], X[y_kmeans==0, 1], s = 100, c = 'red', label = 'Cluster - 1')
plt.scatter(X[y_kmeans==1, 0], X[y_kmeans==1, 1], s = 100, c = 'blue', label = 'Cluster - 2')
plt.scatter(X[y_kmeans==2, 0], X[y_kmeans==2, 1], s = 100, c = 'green', label = 'Cluster - 3'
plt.scatter(X[y_kmeans==3, 0], X[y_kmeans==3, 1], s = 100, c = 'cyan', label = 'Cluster - 4')
plt.scatter(X[y_kmeans==4, 0], X[y_kmeans==4, 1], s = 100, c = 'magenta', label = 'Cluster -
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c='yellow'
plt.title('Clusters of Clients')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```




```
print(y_kmeans)
#We convert our prediction to dataframe so we can easily see this prediction in table form
df_pred = pd.DataFrame(y_kmeans)
df_pred.head()
```

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

	0
0	3
1	1

it's our prediction means 0 number customer belongs to 3 number cluster 1 number customer belongs to 1 number cluster

4	3

Since the dataset was small, all these processes take no time to finish. Once the clusters are created, we can plot them on a graph. Each cluster point is marked using a different sign, and the centroids of each cluster are marked using solid red dots.

Just looking at the graph tells us about the five different types of customers that frequent the mall. If we were to name them, then they could be named as follows:

i. Low income, High spenders (Red).

ii. Low income, Low Spenders (Blue).

iii. Average income, average expenditure(Orange).

iv. High income, High spenders, and(Green)

v. High income, Low spenders(Purple).

Members of each of these groups would have more features common with each other, and therefore we have a homogeneous group. People of each of these clusters may have similar needs and desires. By keeping that in mind all marketing /sales activities can accommodate these needs and **desires to attract more such customers.** For example, a weekly **discount sale** that caters to the low-income group or reward points for purchases which will cater to the high spenders, turning them into regular customers. The possibilities are limitless and are only bounded by our imagination.

Conclusion Understanding a business's customer base is of utmost importance. One of the ways to gain deeper insight into customer behaviour is by segmenting them into different buckets based on their behaviour (income and expenditure in this experiment.) Similar people tend to behave simi ,

and this is the crux of customer segmentation. Therefore, by planning all the sales and marketing activities around these buckets, it would promise a higher return on investment and enjoyable customer experience.

```
dataset = pd.read_csv('Mall_Customers.csv')
X = dataset.iloc[:, [1,4]].values
X
from sklearn.preprocessing import LabelEncoder

labelencoder_X = LabelEncoder()
X[:, 0] = labelencoder_X.fit_transform(X[:, 0])
# print(X)

# Applying k-means to the mall dataset
kmeans = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_state
y_kmeans = kmeans.fit_predict(X)

# Visualising the clusters
plt.scatter(X[y_kmeans==0, 0], X[y_kmeans==0, 1], s = 100, c = 'red', label = 'Cluster - 1')
plt.scatter(X[y_kmeans==1, 0], X[y_kmeans==1, 1], s = 100, c = 'blue', label = 'Cluster - 2')
plt.scatter(X[y_kmeans==2, 0], X[y_kmeans==2, 1], s = 100, c = 'green', label = 'Cluster - 3'
plt.scatter(X[y_kmeans==3, 0], X[y_kmeans==3, 1], s = 100, c = 'cyan', label = 'Cluster - 4')
plt.scatter(X[y_kmeans==4, 0], X[y_kmeans==4, 1], s = 100, c = 'magenta', label = 'Cluster -
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 300, c='yellow'
plt.title('Clusters of Clients')
plt.xlabel('Genre')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```

20012531022

20012531022

ALAUKI PATEL

```
# Feature Selection - Forward Feature Selection, Backward Elimination,.....
# Feature Extraction - PCA - Principal Component Analysis
                    #  - LDA - Linear Discreminent Analysis
                    #  - Kernel PCA
```

```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

#/content/drive/MyDrive/
#/content/drive/MyDrive/machine learning 2021/LAB DATA/Wine.csv
# https://www.youtube.com/watch?v=MLaJbA82nzk&ab_channel=EduFlairKTUCS
# https://www.youtube.com/watch?v=FgakZw6K1QQ&ab_channel=StatQuestwithJoshStarmer
# https://www.kaggle.com/questions-and-answers/174690
# https://www.bi4all.pt/en/news/en-blog/what-is-pca/
# https://www.youtube.com/watch?reload=9&v=o0NNUeWNnL4&ab_channel=5MinutesEngineering-Princip
# https://www.youtube.com/watch?v=_ZkFfrCfIws&ab_channel=5MinutesEngineering- Principal Compo
```

```
!WGET https://drive.google.com/open?id=112-yODBtpM4bk8iK7kDNGI1BJPRVBg0u
```

```
/bin/bash: WGET: command not found
```

```
dataset = pd.read_csv('/content/Wine.CSV')
```

```
dataset.sample(10)
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Phenols	Flavanoids	Nonf
162	12.85	3.27	2.58	22.0	106	1.65	0.60	
158	14.34	1.68	2.70	25.0	98	2.80	1.31	

```
dataset.describe
```

```
<bound method NDFrame.describe of      Alcohol  Malic_Acid   Ash  Ash_Alcanity
Magnesium  Total_Phenols  \
0      14.23        1.71  2.43          15.6        127           2.80
1      13.20        1.78  2.14          11.2        100           2.65
2      13.16        2.36  2.67          18.6        101           2.80
3      14.37        1.95  2.50          16.8        113           3.85
4      13.24        2.59  2.87          21.0        118           2.80
..       ...         ...   ...           ...        ...            ...
173    13.71        5.65  2.45          20.5         95           1.68
174    13.40        3.91  2.48          23.0        102           1.80
175    13.27        4.28  2.26          20.0        120           1.59
176    13.17        2.59  2.37          20.0        120           1.65
177    14.13        4.10  2.74          24.5         96           2.05

     Flavanoids  Nonflavanoid_Phenols  Proanthocyanins  Color_Intensity   Hue  \
0          3.06                  0.28             2.29             5.64  1.04
1          2.76                  0.26             1.28             4.38  1.05
2          3.24                  0.30             2.81             5.68  1.03
3          3.49                  0.24             2.18             7.80  0.86
4          2.69                  0.39             1.82             4.32  1.04
..          ...                   ...              ...              ...   ...
173        0.61                  0.52             1.06             7.70  0.64
174        0.75                  0.43             1.41             7.30  0.70
175        0.69                  0.43             1.35            10.20  0.59
176        0.68                  0.53             1.46             9.30  0.60
177        0.76                  0.56             1.35             9.20  0.61

     OD280  Proline  Customer_Segment
0     3.92     1065                 1
1     3.40     1050                 1
2     3.17     1185                 1
3     3.45     1480                 1
4     2.93      735                 1
..     ...      ...               ...
173   1.74      740                 3
174   1.56      750                 3
175   1.56      835                 3
176   1.62      840                 3
177   1.60      560                 3

[178 rows x 14 columns]>
```

```
dataset.corr()
# https://archive.ics.uci.edu/ml/datasets/wine   Discription
# The attributes are (dontated by Riccardo Leardi, riclea '@' anchem.unige.it )
# 1) Alcohol
```

```
# 2) Malic acid
# 3) Ash
# 4) Alcalinity of ash
# 5) Magnesium
# 6) Total phenols
# 7) Flavanoids
# 8) Nonflavanoid phenols
# 9) Proanthocyanins
# 10)Color intensity
# 11)Hue
# 12)OD280/OD315 of diluted wines
# 13)Proline
```

	Alcohol	Malic_Acid	Ash	Ash_Alcanity	Magnesium	Total_Ph
Alcohol	1.000000	0.094397	0.211545	-0.310235	0.270798	0.2
Malic_Acid	0.094397	1.000000	0.164045	0.288500	-0.054575	-0.3
Ash	0.211545	0.164045	1.000000	0.443367	0.286587	0.1
Ash_Alcanity	-0.310235	0.288500	0.443367	1.000000	-0.083333	-0.3
Magnesium	0.270798	-0.054575	0.286587	-0.083333	1.000000	0.2
Total_Phenols	0.289101	-0.335167	0.128980	-0.321113	0.214401	1.0
Flavanoids	0.236815	-0.411007	0.115077	-0.351370	0.195784	0.8
Nonflavanoid_Phenols	-0.155929	0.292977	0.186230	0.361922	-0.256294	-0.4
Proanthocyanins	0.136698	-0.220746	0.009652	-0.197327	0.236441	0.6
Color_Intensity	0.546364	0.248985	0.258887	0.018732	0.199950	-0.0
Hue	-0.071747	-0.561296	-0.074667	-0.273955	0.055398	0.4
OD280	0.072343	-0.368710	0.003911	-0.276769	0.066004	0.6
Proline	0.643720	-0.192011	0.223626	-0.440597	0.393351	0.4
Customer_Segment	-0.328222	0.437776	-0.049643	0.517859	-0.209179	-0.7

```
dataset.mean()
```

```
Alcohol                 13.000618
Malic_Acid               2.336348
Ash                      2.366517
Ash_Alcanity            19.494944
Magnesium               99.741573
Total_Phenols            2.295112
Flavanoids               2.029270
Nonflavanoid_Phenols     0.361854
Proanthocyanins          1.590899
```

```
Color_Intensity           5.058090
Hue                       0.957449
OD280                     2.611685
Proline                 746.893258
Customer_Segment          1.938202
dtype: float64
```

```
X = dataset.iloc[:, :-1].values
Y = dataset.iloc[:, -1].values
```

```
# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state = 0)
```

```
from sklearn.preprocessing import StandardScaler
sc_x = StandardScaler()
X_train = sc_x.fit_transform(X_train)
X_test = sc_x.transform(X_test)
```

```
from sklearn.decomposition import PCA
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,Y_train)
```

```
LogisticRegression()
```

```
Y_pred = lr.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,Y_pred)
cm
```

```
array([[16,  0,  0],
       [ 1, 20,  0],
       [ 0,  0,  8]])
```

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
lda = LinearDiscriminantAnalysis(n_components=2)
X_train = lda.fit_transform(X_train,Y_train)
X_test = lda.transform(X_test)
```

```
from sklearn.linear_model import LogisticRegression
lr = LogisticRegression()
lr.fit(X_train,Y_train)
```

```
LogisticRegression()
```

```
Y_pred = lr.predict(X_test)
```

```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test,Y_pred)
cm
```

```
array([[16,  0,  0],
       [ 0, 21,  0],
       [ 0,  0,  8]])
```

```
X_train
```

```
array([[1.364e+01, 3.100e+00, 2.560e+00, ..., 9.600e-01, 3.360e+00,
        8.450e+02],
       [1.260e+01, 2.460e+00, 2.200e+00, ..., 7.300e-01, 1.580e+00,
        6.950e+02],
       [1.196e+01, 1.090e+00, 2.300e+00, ..., 9.900e-01, 3.130e+00,
        8.860e+02],
       ...,
       [1.242e+01, 1.610e+00, 2.190e+00, ..., 1.060e+00, 2.960e+00,
        3.450e+02],
       [1.390e+01, 1.680e+00, 2.120e+00, ..., 9.100e-01, 3.330e+00,
        9.850e+02],
       [1.416e+01, 2.510e+00, 2.480e+00, ..., 6.200e-01, 1.710e+00,
        6.600e+02]])
```

```
# Applying PCA
from sklearn.decomposition import PCA
pca = PCA(n_components =1)
X_train = pca.fit_transform(X_train)   # only X_train  #unsupervise
X_test = pca.transform(X_test)
```

```
X_train
```

```
array([[ 9.54445410e+01],
       [-5.49002358e+01],
       [ 1.36165698e+02],
       [ 1.05359747e+02],
       [ 5.30422666e+02],
       [-2.19731379e+02],
       [-1.69985774e+02],
       [ 4.54666236e+01],
       [-8.98470753e+01],
```

```
       [ 2.85172934e+02],
       [-4.37883536e+02],
       [ 8.54854060e+01],
       [ 8.01848402e+01],
       [-3.40017938e+02],
       [-1.79716691e+02],
       [-2.69946807e+02],
       [ 5.35022188e+02],
       [-2.35039955e+02],
       [-1.00031692e+02],
       [-3.21992319e+02],
       [-3.78066625e+02],
       [-7.47138656e+01],
       [-1.19889840e+02],
       [ 4.35144435e+02],
       [-2.29936780e+02],
       [ 1.88123054e+02],
       [-1.24290223e+02],
       [-6.97517968e+01],
       [ 7.60165233e+02],
       [-6.99093158e+01],
       [ 2.35700511e+02],
       [-2.79992385e+02],
       [-1.29973804e+02],
       [-1.87687451e+02],
       [-3.72013187e+02],
       [-2.55017468e+02],
       [-5.46490595e+01],
       [-2.39850313e+02],
       [ 4.00045953e+02],
       [-2.49381860e+01],
       [-1.25301785e+02],
       [-3.94703749e+02],
       [-3.84779077e+02],
       [-3.70059915e+02],
       [-3.10078236e+01],
       [ 5.10258814e+02],
       [ 3.10194354e+02],
       [-2.99788219e+02],
       [ 4.00035371e+02],
       [-3.00062954e+01],
       [ 2.65073002e+02],
       [-2.29991802e+02],
       [ 3.06102538e+01],
       [ 5.60309412e+02],
       [ 7.65907135e-01],
       [ 4.40331409e+02],
       [-3.98543692e+01],
       [-3.57967979e+02],
```

```
# Fitting logistic Regression to the training set
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state = 0)
classifier.fit(X_train, Y_train)
```

```
LogisticRegression(random_state=0)
```

```
#Predicting the Test set results
Y_pred = classifier.predict(X_test)
```

```
Y_pred
```

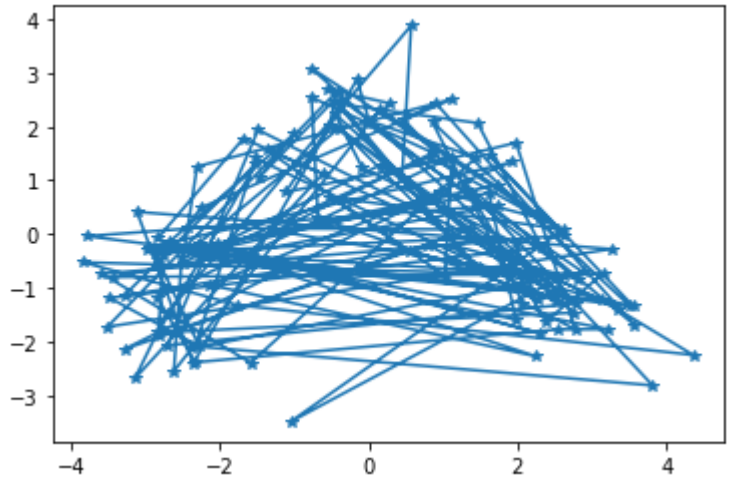
```
array([1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 3, 2, 1, 3, 3, 2, 1, 1, 2, 1, 2, 1,
       3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 3, 2, 2, 3, 1, 1, 2, 2,
       2])
```

```
Y_test,Y_pred
```

```
(array([1, 3, 2, 1, 2, 2, 1, 3, 2, 2, 3, 3, 1, 2, 3, 2, 1, 1, 2, 1, 2, 1,
        1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1, 3, 2, 2, 3, 1, 1, 2, 2,
        2]),
 array([1, 2, 2, 1, 2, 2, 1, 2, 2, 2, 3, 2, 1, 3, 3, 2, 1, 1, 2, 1, 2, 1,
        3, 3, 3, 2, 2, 2, 2, 2, 1, 1, 2, 1, 1, 1, 3, 2, 2, 3, 1, 1, 2, 2,
        2]))
```

```
plt.plot(X_train[:,0],X_train[:,1],marker='*')
```

```
[<matplotlib.lines.Line2D at 0x7ff674a32090>]
```




```
# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)
```

```
cm
```

```
array([[16,  0,  0],
       [ 1, 20,  0],
       [ 0,  0,  8]])
```

```
from sklearn.metrics import classification_report
print (classification_report(Y_test, Y_pred))
```

```
                  precision    recall  f1-score   support

               1       1.00      0.94      0.97        16
               2       0.82      0.86      0.84        21
               3       0.50      0.50      0.50         8

        accuracy                           0.82        45
       macro avg       0.77      0.76      0.77        45
    weighted avg       0.83      0.82      0.82        45
```

20012531022

ALAUKI PATEL

20012531022

ALAUKI PATEL

Importing the libraries

```
import numpy as np
import pandas as pd

import matplotlib.pyplot as plt

#  ketan sarvakar
```

```
!git clone https://github.com/ksarvakar/Churn-Modelling-Dataset.git
```

```
Cloning into 'Churn-Modelling-Dataset'...
remote: Enumerating objects: 29, done.
remote: Total 29 (delta 0), reused 0 (delta 0), pack-reused 29
Unpacking objects: 100% (29/29), done.
```

```
!wget https://github.com/ksarvakar/Churn-Modelling-Dataset/blob/master/Churn_Modelling.csv
```

```
--2022-03-31 09:01:10--  https://github.com/ksarvakar/Churn-Modelling-Dataset/blob/maste
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: ‘Churn_Modelling.csv’

Churn_Modelling.csv     [ <=>                ]   2.96M  15.8MB/s    in 0.2s

2022-03-31 09:01:11 (15.8 MB/s) - ‘Churn_Modelling.csv’ saved [3101461]
```

Importing the Dataset

```
data = pd.read_csv('/content/Churn-Modelling-Dataset/Churn_Modelling.csv')
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
```

```
Data columns (total 14 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   RowNumber        10000 non-null  int64
 1   CustomerId       10000 non-null  int64
 2   Surname          10000 non-null  object
 3   CreditScore      10000 non-null  int64
 4   Geography        10000 non-null  object
 5   Gender           10000 non-null  object
 6   Age              10000 non-null  int64
 7   Tenure           10000 non-null  int64
 8   Balance          10000 non-null  float64
 9   NumOfProducts    10000 non-null  int64
 10  HasCrCard        10000 non-null  int64
 11  IsActiveMember   10000 non-null  int64
 12  EstimatedSalary  10000 non-null  float64
 13  Exited           10000 non-null  int64
dtypes: float64(2), int64(9), object(3)
memory usage: 1.1+ MB
```

```
data.describe()
```

	RowNumber	CustomerId	CreditScore	Age	Tenure	Balance
count	10000.00000	1.000000e+04	10000.000000	10000.000000	10000.000000	10000.000000
mean	5000.50000	1.569094e+07	650.528800	38.921800	5.012800	76485.889288
std	2886.89568	7.193619e+04	96.653299	10.487806	2.892174	62397.405202
min	1.00000	1.556570e+07	350.000000	18.000000	0.000000	0.000000
25%	2500.75000	1.562853e+07	584.000000	32.000000	3.000000	0.000000
50%	5000.50000	1.569074e+07	652.000000	37.000000	5.000000	97198.540000
75%	7500.25000	1.575323e+07	718.000000	44.000000	7.000000	127644.240000
max	10000.00000	1.581569e+07	850.000000	92.000000	10.000000	250898.090000

```
data.tail()
```

	RowNumber	CustomerId	Surname	CreditScore	Geography	Gender	Age	Tenure	Ba
9995	9996	15606229	Obijiaku	771	France	Male	39	5	
9996	9997	15569892	Johnstone	516	France	Male	35	10	57
9997	9998	15584532	Liu	709	France	Female	36	7	
9998	9999	15682355	Sabbatini	772	Germany	Male	42	3	75
9999	10000	15628319	Walker	792	France	Female	28	4	130

```
# Checking if our dataset contains any NULL values

data.isnull().sum()
```

```
RowNumber          0
CustomerId         0
Surname            0
CreditScore        0
Geography          0
Gender             0
Age                0
Tenure             0
Balance            0
NumOfProducts      0
HasCrCard          0
IsActiveMember     0
EstimatedSalary    0
Exited             0
dtype: int64
```

Data Analysis

```
data['Gender'].value_counts()
```

```
Male      5457
Female    4543
Name: Gender, dtype: int64
```

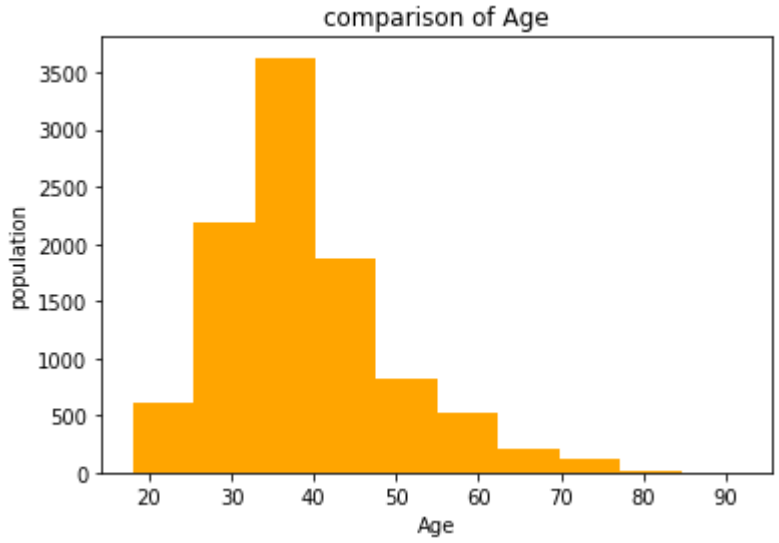
```
# Plotting the features of the dataset to see the correlation between them

plt.hist(x = data.Gender, bins = 3, color = 'pink')
plt.title('comparison of male and female')
plt.xlabel('Gender')
plt.ylabel('population')
plt.show()
```

comparison of male and female

```
data['Age'].value_counts()
```

```
37    478
38    477
35    474
36    456
34    447
     ...
92      2
82      1
88      1
85      1
83      1
Name: Age, Length: 70, dtype: int64
```

```
# comparison of age in the dataset

plt.hist(x = data.Age, bins = 10, color = 'orange')
plt.title('comparison of Age')
plt.xlabel('Age')
plt.ylabel('population')
plt.show()
```

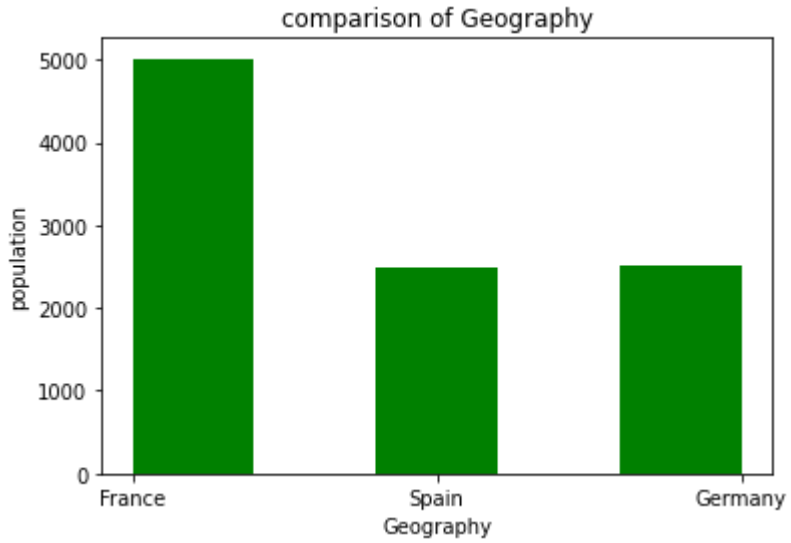



```
data['Geography'].value_counts()
```

```
France     5014
Germany    2509
Spain      2477
Name: Geography, dtype: int64
```

```
# comparison of geography

plt.hist(x = data.Geography, bins = 5, color = 'green')
```

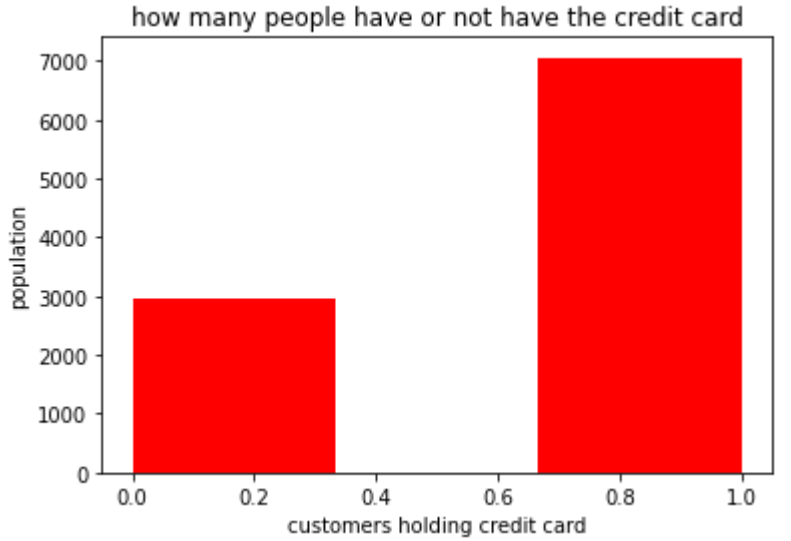
```
plt.title('comparison of Geography')
plt.xlabel('Geography')
plt.ylabel('population')
plt.show()
```




```
data['HasCrCard'].value_counts()
```

```
1    7055
0    2945
Name: HasCrCard, dtype: int64
```

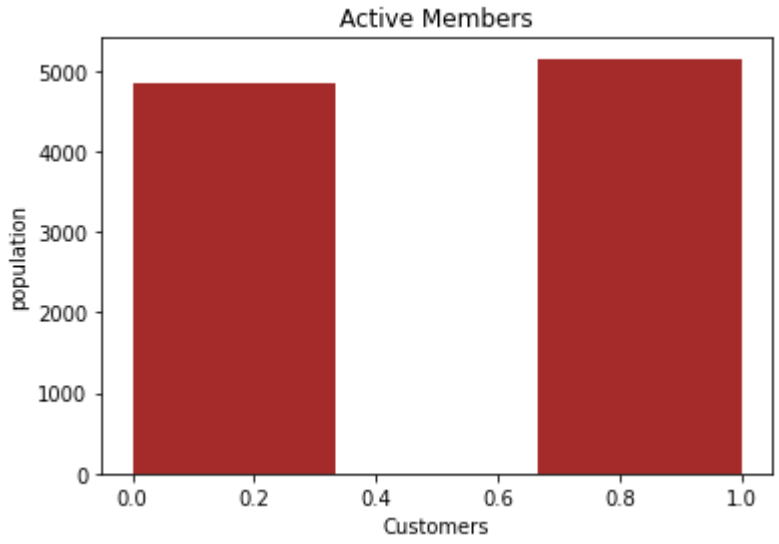
```
# comparision of how many customers hold the credit card

plt.hist(x = data.HasCrCard, bins = 3, color = 'red')
plt.title('how many people have or not have the credit card')
plt.xlabel('customers holding credit card')
plt.ylabel('population')
plt.show()
```



```
data['IsActiveMember'].value_counts()
```

```
1    5151
0    4849
Name: IsActiveMember, dtype: int64
```

```
# How many active member does the bank have ?

plt.hist(x = data.IsActiveMember, bins = 3, color = 'brown')
plt.title('Active Members')
plt.xlabel('Customers')
plt.ylabel('population')
plt.show()
```

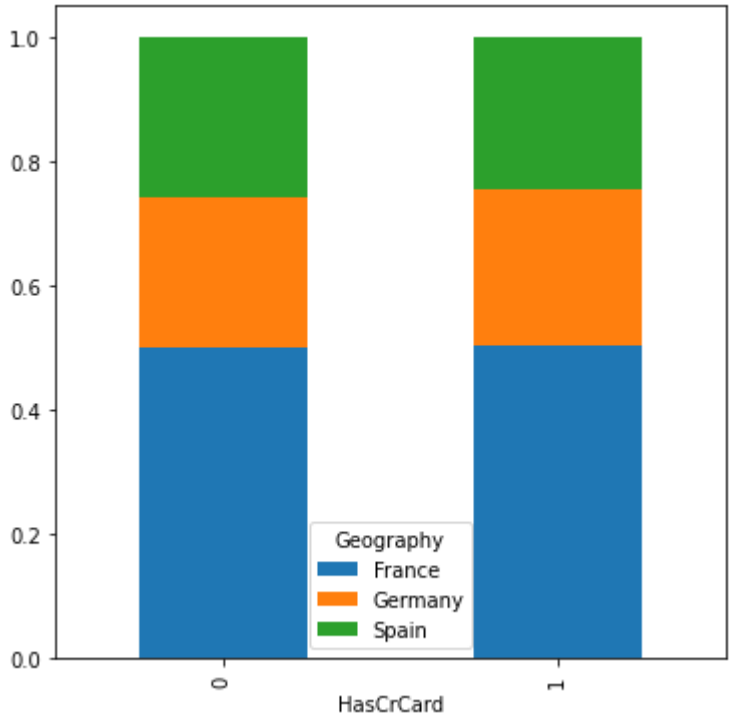



```
# comparison between Geography and Gender

Gender = pd.crosstab(data['Gender'],data['Geography'])
Gender.div(Gender.sum(1).astype(float), axis=0).plot(kind="bar", stacked=True, figsize=(6, 6)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59d6dc5290>
```

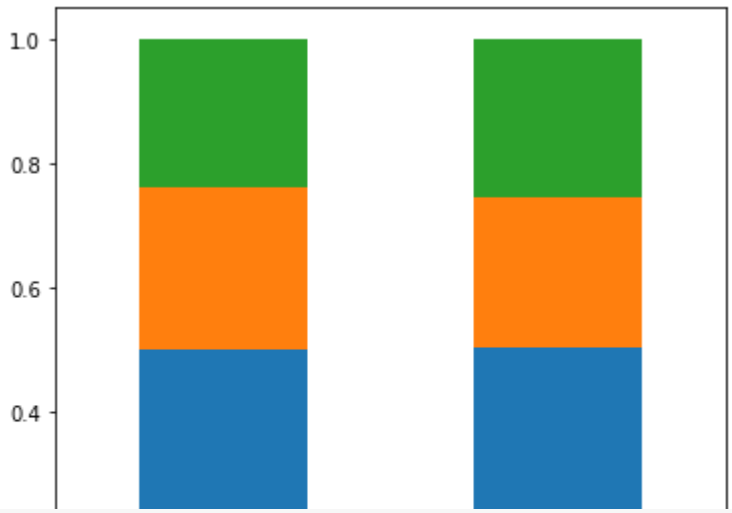

```
# comparison between geography and card holders

HasCrCard = pd.crosstab(data['HasCrCard'], data['Geography'])
HasCrCard.div(HasCrCard.sum(1).astype(float), axis = 0).plot(kind = 'bar',
                                                stacked = True,figsize = (6, 6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59d6d47250>
```



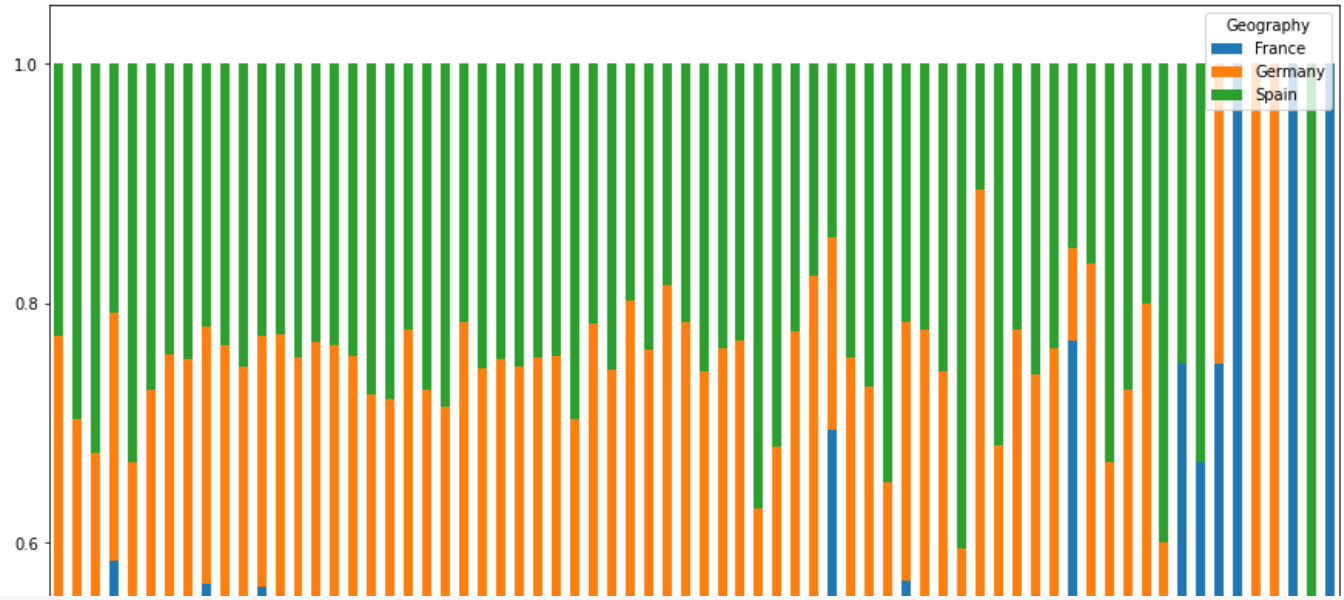

```
# comparison of active member in differnt geographies

IsActiveMember = pd.crosstab(data['IsActiveMember'], data['Geography'])
IsActiveMember.div(IsActiveMember.sum(1).astype(float), axis = 0).plot(kind = 'bar',
                                                  stacked = True, figsize= (6, 6))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59d6c64e10>
```




```
# comparing ages in different geographies

Age = pd.crosstab(data['Age'], data['Geography'])
Age.div(Age.sum(1).astype(float), axis = 0).plot(kind = 'bar',
                                                 stacked = True, figsize = (15,15))
```

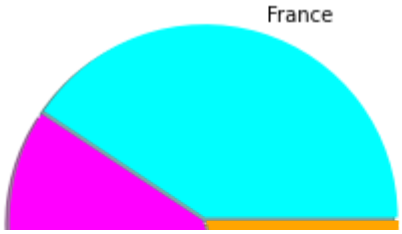
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f59d6e77c50>
```




```
# calculating total balance in france, germany and spain

total_france = data.Balance[data.Geography == 'France'].sum()
total_germany = data.Balance[data.Geography == 'Germany'].sum()
total_spain = data.Balance[data.Geography == 'Spain'].sum()

print("Total Balance in France :",total_france)
print("Total Balance in Germany :",total_germany)
print("Total Balance in Spain :",total_spain)
```

```
Total Balance in France : 311332479.49
Total Balance in Germany : 300402861.38
Total Balance in Spain : 153123552.01
```

```
# plotting a pie chart

labels = 'France', 'Germany', 'Spain'
colors = ['cyan', 'magenta', 'orange']
sizes =  [311, 300, 153]
explode = [ 0.01, 0.01, 0.01]

plt.pie(sizes, colors = colors, labels = labels, explode = explode, shadow = True)

plt.axis('equal')
plt.show()
```



Data Preprocessing

```
# Removing the unnecassary features from the dataset

data = data.drop(['CustomerId', 'Surname', 'RowNumber'], axis = 1)



print(data.columns)
```

```
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary',
       'Exited'],
      dtype='object')
```

```
data.shape
```

```
(10000, 11)
```

```
# splitting the dataset into x(independent variables) and y(dependent variables)

x = data.iloc[:,0:10]
y = data.iloc[:,10]

print(x.shape)
print(y.shape)

print(x.columns)
#print(y)
```

```
(10000, 10)
(10000,)
Index(['CreditScore', 'Geography', 'Gender', 'Age', 'Tenure', 'Balance',
       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary'],
      dtype='object')
```

```
# Encoding Categorical variables into numerical variables
# One Hot Encoding

x = pd.get_dummies(x)

x.head()
```

	CreditScore	Age	Tenure	Balance	NumOfProducts	HasCrCard	IsActiveMember	Estima
0	619	42	2	0.00	1	1	1	
1	608	41	1	83807.86	1	0	1	
2	502	42	8	159660.80	3	1	0	
3	699	39	1	0.00	2	0	0	
4	850	43	2	125510.82	1	1	1	

```
x.shape
```

```
(10000, 13)
```

```
# splitting the data into training and testing set

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.25, random_state = 0)

print(x_train.shape)
print(y_train.shape)
print(x_test.shape)
print(y_test.shape)
```

```
(7500, 13)
(7500,)
(2500, 13)
(2500,)
```

```
# Feature Scaling
# Only on Independent Variable to convert them into values ranging from -1 to +1

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)

x_train = pd.DataFrame(x_train)
x_train.head()
```

	0	1	2	3	4	5	6	7	
0	-0.735507	0.015266	0.008860	0.673160	2.535034	-1.553624	-1.034460	-1.640810	-1.0

Modelling

2	0.808295	-0.461788	1.393293	-0.356937	0.804242	0.643657	0.966688	-0.996840	-1.0

Decision Tree

4	0.467915	1.255605	0.701077	1.207724	0.804242	0.643657	0.966688	1.283302	0.9

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix

model = DecisionTreeClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuaracy :", model.score(x_test, y_test))

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Training Accuracy : 1.0
Testing Accuaracy : 0.806
[[1718  273]
 [ 212  297]]
```

Random Forest

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Training Accuracy : 0.9998666666666667
Testing Accuracy : 0.8736
[[1921   70]
 [ 246  263]]
```

```
# k fold cross validatio
```

```
from sklearn.model_selection import cross_val_score

cvs = cross_val_score(estimator = model, X = x_train, y = y_train, cv = 10)
print(cvs)
```

```
[0.86533333 0.844      0.85866667 0.864      0.86       0.852
 0.86133333 0.85333333 0.84666667 0.85733333]
```

```
print("Mean Accuracy :", cvs.mean())
print("Variance :", cvs.std())
```

```
Mean Accuracy : 0.8562666666666667
Variance : 0.006767241354906402
```

*** *italicized text*Logistic Regression**

```
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Training Accuracy : 0.8096
Testing Accuracy : 0.8092
[[1916   75]
 [ 402  107]]
```

Support Vector Machine

```
from sklearn.svm import SVC

model = SVC()
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))
```

```
cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Training Accuracy : 0.8625333333333334
Testing Accuracy : 0.8616
[[1951   40]
 [ 306  203]]
```

```
# k fold cross validatio

from sklearn.model_selection import cross_val_score

cvs = cross_val_score(estimator = model, X = x_train, y = y_train, cv = 10)
print(cvs)
```

```
[0.864      0.852      0.864      0.85733333 0.84266667 0.844
 0.852      0.85333333 0.84533333 0.85066667]
```

```
print("Mean Accuracy :", cvs.mean())
print("Variance :", cvs.std())
```

```
Mean Accuracy : 0.8525333333333333
Variance : 0.007160384843785353
```

Multi Layer Perceptron

```
from sklearn.neural_network import MLPClassifier

model = MLPClassifier(hidden_layer_sizes = (100, 100), activation ='relu',
                      solver = 'adam', max_iter = 50)
model.fit(x_train, y_train)

y_pred = model.predict(x_test)

print("Training Accuracy :", model.score(x_train, y_train))
print("Testing Accuracy :", model.score(x_test, y_test))

cm = confusion_matrix(y_test, y_pred)
print(cm)
```

```
Training Accuracy : 0.8873333333333333
Testing Accuracy : 0.8612
[[1881  110]
 [ 237  272]]
/usr/local/lib/python3.7/dist-packages/sklearn/neural_network/_multilayer_perceptron.py
  ConvergenceWarning,
```

** **bold text**Aritificial Neural Networks**

```
import keras
from keras.models import Sequential
from keras.layers import Dense
```

```
#dataset = pd.read_csv('Churn_Modelling.csv')
dataset = pd.read_csv('/content/Churn-Modelling-Dataset/Churn_Modelling.csv')
```

```
X = dataset.iloc[:, 3:-1].values
Y = dataset.iloc[:, -1].values

# Encoding categorical data
from sklearn.preprocessing import LabelEncoder, OneHotEncoder
from sklearn.compose import ColumnTransformer

transform =  ColumnTransformer([("Cs",OneHotEncoder(),[1])], remainder = 'passthrough')
X = transform.fit_transform(X)
X = X[:,1:]

transform =  ColumnTransformer([("Gender",OneHotEncoder(),[3])], remainder = 'passthrough')
X = transform.fit_transform(X)
X = X[:,1:]

# Splitting the dataset into the training set and test set
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state = 0)

# Feature Scaling
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

```
# Intitialising the ANN
classifier = Sequential()


# Adding the input layer and First hidden layer
classifier.add(Dense(units=6, kernel_initializer = 'uniform', activation = 'relu',
                     input_shape = (11,)))
#classifier.add(Dense(output_dim=6, init = 'uniform', activation = 'relu', input_dim = 11))

# Adding 2nd hidden layer
#classifier.add(Dense(output_dim=6, init = 'uniform', activation = 'relu'))
classifier.add(Dense(units=6, kernel_initializer = 'uniform', activation = 'relu'))

# Adding the output layer
```

```
#classifier.add(Dense(output_dim=1, init = 'uniform', activation = 'sigmoid'))
classifier.add(Dense(units=1, kernel_initializer = 'uniform', activation = 'sigmoid'))

# Compiling the ANN - Adadelta - categorical_crossentropy
classifier.compile(optimizer = 'adam', loss='binary_crossentropy', metrics = ['accuracy'])

# Fitting the ANN to the training dataset
classifier.fit(X_train, Y_train, batch_size = 10, epochs = 100)

# Predicting the Test set results
Y_pred = classifier.predict(X_test)
Y_pred = (Y_pred > 0.5)

# Making the confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(Y_test, Y_pred)

print((cm[0][0]+cm[1][1])/(cm[0][0]+cm[0][1]+cm[1][0]+cm[1][1]) * 100)



##### Using ANN perform project for house price prediction
##### Integrate the model with flask web app
```

```
Epoch 1/100
800/800 [==============================] - 4s 3ms/step - loss: 0.4900 - accuracy: 0.7
Epoch 2/100
800/800 [==============================] - 3s 3ms/step - loss: 0.4295 - accuracy: 0.7
Epoch 3/100
800/800 [==============================] - 3s 3ms/step - loss: 0.4252 - accuracy: 0.7
Epoch 4/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4209 - accuracy: 0.7
Epoch 5/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4175 - accuracy: 0.8
Epoch 6/100
800/800 [==============================] - 1s 2ms/step - loss: 0.4149 - accuracy: 0.8
Epoch 7/100
800/800 [==============================] - 1s 2ms/step - loss: 0.4136 - accuracy: 0.8
Epoch 8/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4119 - accuracy: 0.8
Epoch 9/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4106 - accuracy: 0.8
Epoch 10/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4094 - accuracy: 0.8
Epoch 11/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4088 - accuracy: 0.8
Epoch 12/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4083 - accuracy: 0.8
Epoch 13/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4075 - accuracy: 0.8
Epoch 14/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4068 - accuracy: 0.8
```

```
Epoch 15/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4066 - accuracy: 0.8
Epoch 16/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4059 - accuracy: 0.8
Epoch 17/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4057 - accuracy: 0.8
Epoch 18/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4047 - accuracy: 0.8
Epoch 19/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4046 - accuracy: 0.8
Epoch 20/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4045 - accuracy: 0.8
Epoch 21/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4038 - accuracy: 0.8
Epoch 22/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4040 - accuracy: 0.8
Epoch 23/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4040 - accuracy: 0.8
Epoch 24/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4033 - accuracy: 0.8
Epoch 25/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4031 - accuracy: 0.8
Epoch 26/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4031 - accuracy: 0.8
Epoch 27/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4025 - accuracy: 0.8
Epoch 28/100
800/800 [==============================] - 1s 1ms/step - loss: 0.4025 - accuracy: 0.8
Epoch 29/100
```

```
# creating the model
model = Sequential()

from keras.layers import Dropout

# first hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu', input_dim = 14))
model.add(Dropout(0.5))

# second hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu'))
model.add(Dropout(0.5))

# output layer
model.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN
# binary_crossentropy loss function used when a binary output is expected
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

model.fit(x_train, y_train, batch_size = 10, nb_epoch = 50)
```

```
# creating the model
model = Sequential()

# first hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu', input_dim = 14))
model.add(Dropout(0.1))

# second hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu'))
model.add(Dropout(0.1))

# output layer
model.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN
# binary_crossentropy loss function used when a binary output is expected
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

model.fit(x_train, y_train, batch_size = 10, nb_epoch = 50)
```

```
# creating the model
model = Sequential()

# first hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu', input_dim = 13))

# second hidden layer
model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu'))

# output layer
model.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

# Compiling the NN
# binary_crossentropy loss function used when a binary output is expected
model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

model.fit(x_train, y_train, batch_size = 10, nb_epoch = 49)
```

```
data.columns
```

```
'''
predicting if the costumer having following information will leave the bank or not ?
```

```
Geography : france
Age = 50
Credit score = 850
Tenure = 4
Balance = 150000
Number of Products = 5
Gender = Female
Has Credit Card = yes
Is Active Member = yes
Estimated Salary = 85000
'''

new_prediction = model.predict(sc.transform(np.array([[850, 50, 4, 150000, 5, 1, 1, 85000, 1,

new_prediction = (new_prediction > 0.5 )
print(new_prediction)
```

```
from keras.wrappers.scikit_learn import KerasClassifier
from sklearn.model_selection import cross_val_score

from keras.layers import Dense
from keras.models import Sequential

def build_classifier():
  # creating the model
  model = Sequential()

  # first hidden layer
  model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu', input_dim = 13))

  # second hidden layer
  model.add(Dense(output_dim = 8, init = 'uniform', activation = 'relu'))

  # output layer
  model.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

  # Compiling the NN
  # binary_crossentropy loss function used when a binary output is expected
  model.compile(optimizer = 'adam', loss = 'binary_crossentropy', metrics = ['accuracy'])

  return model

model = KerasClassifier(build_fn = build_classifier, batch_size = 10, nb_epoch = 50)
accuracies = cross_val_score(estimator = model, X = x_train, y = y_train, cv = 10, n_jobs = -
```

```
print("Accuracies :", accuracies)

print("Mean Accuracy :", accuracies.mean())
print("Variance :", accuracies.std())
```

20012531022

ALAUKI PATEL

PRACTICAL 12

Hierarchical Clustering

> A Summary of lecture "Cluster Analysis in Python", via datacamp

- toc: true
- badges: true
- comments: true
- author: Chanseok Kang
- categories: [Python, Datacamp, Machine Learning]
- image: images/fifa_cluster.png

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

Basics of hierarchical clustering

- Creating a distance matrix using linkage
  - `method` : how to calculate the proximity of clusters
  - `metric` : distance metric
  - `optimal_ordering` : order data points
- Type of Methods
  - single: based on two closest objects
  - complete: based on two farthest objects
  - average: based on the arithmetic mean of all objects
  - centroids: based on the geometric mean of all objects
  - median: based on the median of all objects
  - ward: based on the sum of squares

Hierarchical clustering: ward method

It is time for Comic-Con! Comic-Con is an annual comic-based convention held in major cities in the world. You have the data of last year's footfall, the number of people at the convention ground at a given time. You would like to decide the location of your stall to maximize sales. Using the ward method, apply hierarchical clustering to find the two points of attraction in the area.

- Preprocess

```
comic_con = pd.read_csv('./dataset/comic_con.csv', index_col=0)
comic_con.head()
```

	x_coordinate	y_coordinate
0	17	4
1	20	6
2	35	0
3	14	0
4	37	4

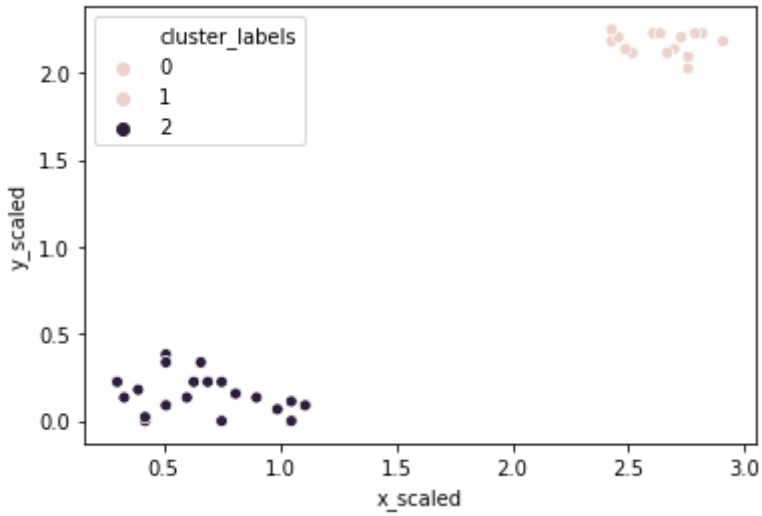
```
from scipy.cluster.vq import whiten

comic_con['x_scaled'] = whiten(comic_con['x_coordinate'])
comic_con['y_scaled'] = whiten(comic_con['y_coordinate'])
```

```
from scipy.cluster.hierarchy import linkage, fcluster

# Use the linkage()
distance_matrix = linkage(comic_con[['x_scaled', 'y_scaled']], method='ward', metric='eucl

# Assign cluster labels
comic_con['cluster_labels'] = fcluster(distance_matrix, 2, criterion='maxclust')

# Plot clusters
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con);
```

Hierarchical clustering: single method

Let us use the same footfall dataset and check if any changes are seen if we use a different method for clustering.

```
# Use the linkage()
distance_matrix = linkage(comic_con[['x_scaled', 'y_scaled']], method='single', metric='eu

# Assign cluster labels
comic_con['cluster_labels'] = fcluster(distance_matrix, 2, criterion='maxclust')

# Plot clusters
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con);
```



Hierarchical clustering: complete method

For the third and final time, let us use the same footfall dataset and check if any changes are seen if we use a different method for clustering.

```
# Use the linkage()
distance_matrix = linkage(comic_con[['x_scaled', 'y_scaled']], method='complete', metric='

# Assign cluster labels
comic_con['cluster_labels'] = fcluster(distance_matrix, 2, criterion='maxclust')

# Plot clusters
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con);
```

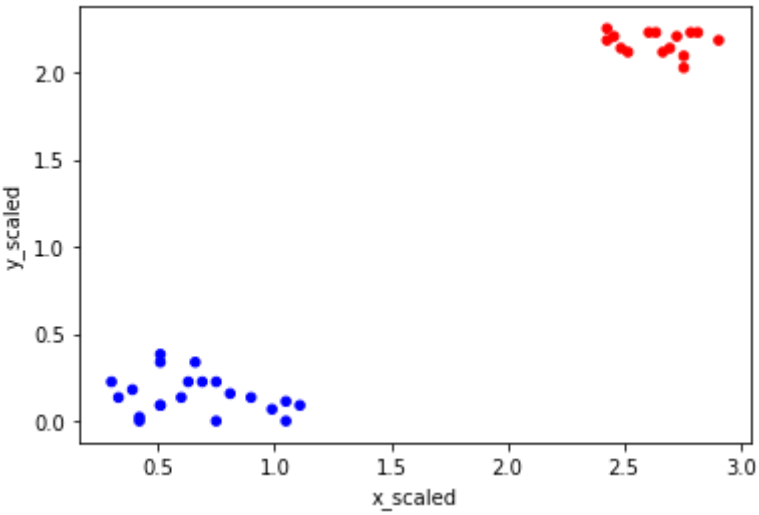
Visualize clusters

- Why visualize clusters?
  - Try to make sense of the clusters formed
  - An additional step in validation of clusters
  - Spot trends in data

Visualize clusters with matplotlib

We have discussed that visualizations are necessary to assess the clusters that are formed and spot trends in your data. Let us now focus on visualizing the footfall dataset from Comic-Con using the matplotlib module.

```
# Define a colors dictionary for clusters
colors = {1:'red', 2:'blue'}

# Plot the scatter plot
comic_con.plot.scatter(x='x_scaled', y='y_scaled', c=comic_con['cluster_labels'].apply(lam
```

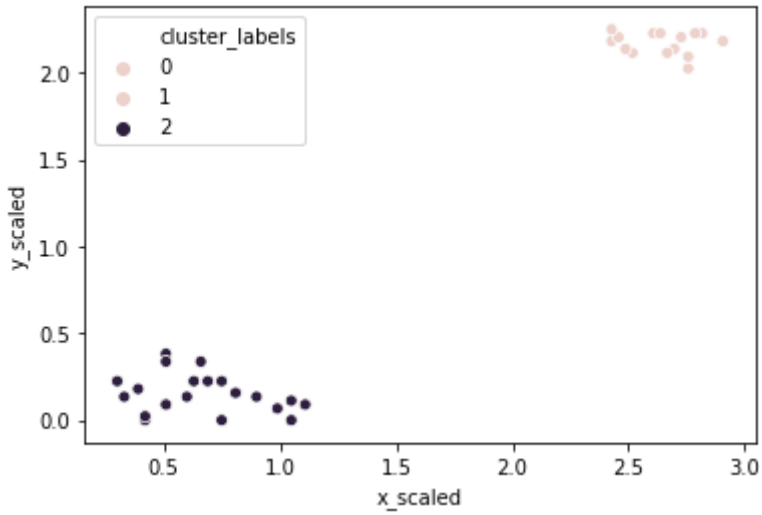


Visualize clusters with seaborn

Let us now visualize the footfall dataset from Comic Con using the seaborn module. Visualizing clusters using seaborn is easier with the inbuild `hue` function for cluster labels.

```
# Plot a scatter plot using seaborn
sns.scatterplot(x='x_scaled', y='y_scaled', hue='cluster_labels', data=comic_con)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x27d3a82d1c8>
```



How many clusters?

- Introduction to dendrograms
  - Strategy till now - decide clusters on visual inspection
  - Dendrograms help in showing progressions as clusters are merged
  - A dendrogram is a branching diagram that demonstrates how each cluster is composed by branching out into its child nodes

Create a dendrogram

Dendrograms are branching diagrams that show the merging of clusters as we move through the distance matrix. Let us use the Comic Con footfall data to create a dendrogram.

```
from scipy.cluster.hierarchy import dendrogram

# Create a dendrogram
dn = dendrogram(distance_matrix)
```

Limitations of hierarchical clustering

- Comparison of runtime of linkage method
  - Increasing runtime with data points
  - Quadratic increase of runtime
  - Not feasible for large datasets

Timing run of hierarchical clustering

In earlier exercises of this chapter, you have used the data of Comic-Con footfall to create clusters. In this exercise you will time how long it takes to run the algorithm on DataCamp's system.

Remember that you can time the execution of small code snippets with:

```
%timeit sum([1, 3, 2])
```

```
%timeit linkage(comic_con[['x_scaled', 'y_scaled']], method='ward', metric='euclidean')
```

```
459 µs ± 377 ns per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

FIFA 18: exploring defenders

In the FIFA 18 dataset, various attributes of players are present. Two such attributes are:

- sliding tackle: a number between 0-99 which signifies how accurate a player is able to perform sliding tackles
- aggression: a number between 0-99 which signifies the commitment and will of a player These are typically high in defense-minded players. In this exercise, you will perform clustering based on these attributes in the data.

This data consists of 5000 rows, and is considerably larger than earlier datasets. Running hierarchical clustering on this data can take up to 10 seconds.

- Preprocess

```
fifa = pd.read_csv('./dataset/fifa_18_dataset.csv')
fifa.head()
```

	sliding_tackle	aggression
0	23	63
1	26	48
4	11	29

```
fifa['scaled_sliding_tackle'] = whiten(fifa['sliding_tackle'])
fifa['scaled_aggression'] = whiten(fifa['aggression'])

# Fit the data into a hierarchical cluster
distance_matrix = linkage(fifa[['scaled_sliding_tackle', 'scaled_aggression']], method='wa

# Assign cluster labels to each row of data
fifa['cluster_labels'] = fcluster(distance_matrix, 3, criterion='maxclust')

# Display cluster centers of each cluster
print(fifa[['scaled_sliding_tackle', 'scaled_aggression', 'cluster_labels']].groupby('clus

# Create a scatter plot through seaborn
sns.scatterplot(x='scaled_sliding_tackle', y='scaled_aggression', hue='cluster_labels', da
plt.savefig('../images/fifa_cluster.png')
```

```
                scaled_sliding_tackle  scaled_aggression
cluster_labels
1                            0.987373           1.849142
2                            3.013487           4.063492
3                            1.934455           3.210802
```