ex02-data-preprocessing

October 16, 2024

1 Importing Libraries

We begin by importing the necessary libraries for numerical operations and data manipulation.

[1]:
```
import pandas as pd
```

2 Loading the Dataset

Here, we load the dataset from a CSV file and separate the features (independent variables) from the target (dependent variable).

[2]:
```
dataset = pd.read_csv("datasets/ShopSellData.csv")
```

3 Splitting the inputs and outputs

`dataset.iloc` can be used to index into rows and columns using integers.

The first parameter is rows and the second parameter is columns.

`dataset.iloc[:, :-1]` indexes all rows and columns from 0 to before last, thus excluding the last one.

if column names are known, then we can also use `dataset.loc` to perform indexing using only column names.

It is notable that using column names will index inclusively (last column is included in result unlike integers).

[3]:
```
x = dataset.loc[:, :"Salary"].values
```

[4]:
```
x
```

[4]:
```
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)
```

[5]:
```
dataset.iloc[:, :-1]
```

[5]:

	Country	Age	Salary
0	France	44.0	72000.0
1	Spain	27.0	48000.0
2	Germany	30.0	54000.0
3	Spain	38.0	61000.0
4	Germany	40.0	NaN
5	France	35.0	58000.0
6	Spain	NaN	52000.0
7	France	48.0	79000.0
8	Germany	50.0	83000.0
9	France	37.0	67000.0

[6]:
```
y = dataset.iloc[:, 3].values
```

[7]:
```
dataset.iloc[:, 3]
```

[7]:
```
0     No
1    Yes
2     No
3     No
4    Yes
5    Yes
6     No
7    Yes
8     No
9    Yes
Name: Purchased, dtype: object
```

4 Viewing the Dataset

We display the entire dataset and a quick overview of the first two rows to understand its structure.

[8]:
```
dataset
```

[8]:

	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
```

[9]:
```
dataset.head()
```

[9]:
```
   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
```

[10]:
```
dataset.head(2)
```

[10]:
```
  Country   Age   Salary Purchased
0  France  44.0  72000.0        No
1   Spain  27.0  48000.0       Yes
```

[11]:
```
dataset.tail(2)
```

[11]:
```
   Country   Age   Salary Purchased
8  Germany  50.0  83000.0        No
9   France  37.0  67000.0       Yes
```

5 Label Encoding

Label Encoding is used to convert categorical data into numeric form. Here, we encode the 'Country' column.

`LabelEncoder` is a class we import from scikit-learn (imported as `sklearn`) library

[12]:
```
from sklearn.preprocessing import LabelEncoder
```

[13]:
```
label_encode_x = LabelEncoder()
```

[14]:
```
x[:, 0] = label_encode_x.fit_transform(x[:, 0])
```

[15]:
```
x
```

[15]:
```
array([[0, 44.0, 72000.0],
       [2, 27.0, 48000.0],
       [1, 30.0, 54000.0],
       [2, 38.0, 61000.0],
       [1, 40.0, nan],
       [0, 35.0, 58000.0],
       [2, nan, 52000.0],
```

```
[0, 48.0, 79000.0],
[1, 50.0, 83000.0],
[0, 37.0, 67000.0]], dtype=object)
```

6 One Hot Encoding

One Hot Encoding is used to create dummy variables for categorical data. This step ensures that the encoded categorical data does not imply any ordinal relationship.

[16]:
```
from sklearn.preprocessing import OneHotEncoder
```

[17]:
```
onehotencoder = OneHotEncoder()
```

[18]:
```
onehotencoder.fit_transform(x[:, 0].reshape(-1, 1)).toarray()
```

[18]:
```
array([[1., 0., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 1., 0.],
       [1., 0., 0.],
       [0., 0., 1.],
       [1., 0., 0.],
       [0., 1., 0.],
       [1., 0., 0.]])
```

7 Encoding the Target Variable

Similar to the feature encoding, we encode the target variable 'Purchased' to convert it into numeric form.

[19]:
```
label_encode_y = LabelEncoder()
```

[20]:
```
y = label_encode_y.fit_transform(y)
```

[21]:
```
y
```

[21]:
```
array([0, 1, 0, 0, 1, 1, 0, 1, 0, 1])
```

8 Splitting the Dataset

We split the dataset into training and test sets. This allows us to train our model on one set of data and test it on another to evaluate its performance.

[22]:
```
from sklearn.model_selection import train_test_split
```

[23]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.2,␣
↪random_state=0)
```

[24]:
```
x_train
```

[24]:
```
array([[1, 40.0, nan],
       [0, 37.0, 67000.0],
       [2, 27.0, 48000.0],
       [2, nan, 52000.0],
       [0, 48.0, 79000.0],
       [2, 38.0, 61000.0],
       [0, 44.0, 72000.0],
       [0, 35.0, 58000.0]], dtype=object)
```

[25]:
```
x_test
```

[25]:
```
array([[1, 30.0, 54000.0],
       [1, 50.0, 83000.0]], dtype=object)
```

[26]:
```
y_train
```

[26]:
```
array([1, 1, 1, 0, 1, 0, 0, 1])
```

[27]:
```
y_test
```

[27]:
```
array([0, 0])
```

9 Feature Scaling

Feature scaling is performed to standardize the range of independent variables. It ensures that each feature contributes equally to the model.

[28]:
```
from sklearn.preprocessing import StandardScaler
```

[29]:
```
sc_x = StandardScaler()
```

[30]:
```
x_train_scaled = sc_x.fit_transform(x_train)
```

[31]:
```
x_test_scaled = sc_x.transform(x_test)
```

[32]:
```
x_train_scaled
```

[32]:
```
array([[ 0.13483997,  0.25315802,         nan],
       [-0.94387981, -0.23014365,  0.44897083],
       [ 1.21355975, -1.84114924, -1.41706417],
       [ 1.21355975,         nan, -1.0242147 ],
       [-0.94387981,  1.54196248,  1.62751925],
       [ 1.21355975, -0.0690431 , -0.14030338],
```

```
        [-0.94387981,  0.89756025,  0.94003267],
        [-0.94387981, -0.55234477, -0.43494049]])
```

[33]:
```
x_test_scaled
```

[33]:
```
array([[ 0.13483997, -1.35784756, -0.82778996],
       [ 0.13483997,  1.8641636 ,  2.02036872]])
```

ex03-multivariate-linear-regression

October 16, 2024

[1]:
```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
```

1 Load the Boston Housing DataSet

[2]:
```
boston = pd.read_csv("./datasets/boston_house_prices.csv")

boston.head()
```

[2]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	MEDV
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

2 Check if our data has null values and count them up for each column

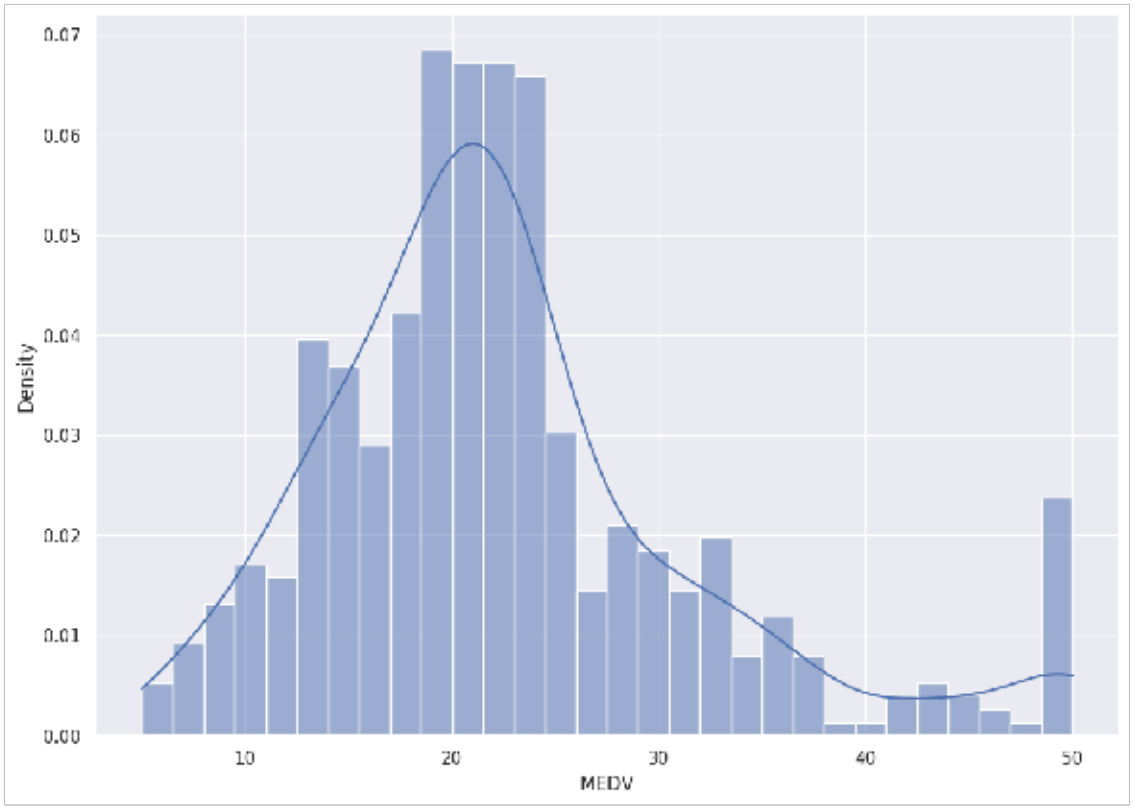
[3]:
```
boston.isnull().sum()
```

[3]:
```
CRIM       0
ZN         0
INDUS      0
CHAS       0
NOX        0
RM         0
AGE        0
```

```
DIS        0
RAD        0
TAX        0
PTRATIO    0
B          0
LSTAT      0
MEDV       0
dtype: int64
```

3 Data Visualization

[4]:
```
# set the size of the figure
sns.set(rc={"figure.figsize": (11.7, 8.27)})
# plot a histogram showing the distribution of the target values
sns.histplot(boston["MEDV"], bins=30, kde=True, stat="density")
plt.show()
```



4 Correlation matrix

[5]:
```
# compute the pair wise correlation for all columns
correlation_matrix = boston.corr().round(2)
```

[6]:
```
# use the heatmap function from seaborn to plot the correlation matrix
# annot = True to print the values inside the square
sns.heatmap(data=correlation_matrix, annot=True)
```

[6]: <Axes: >

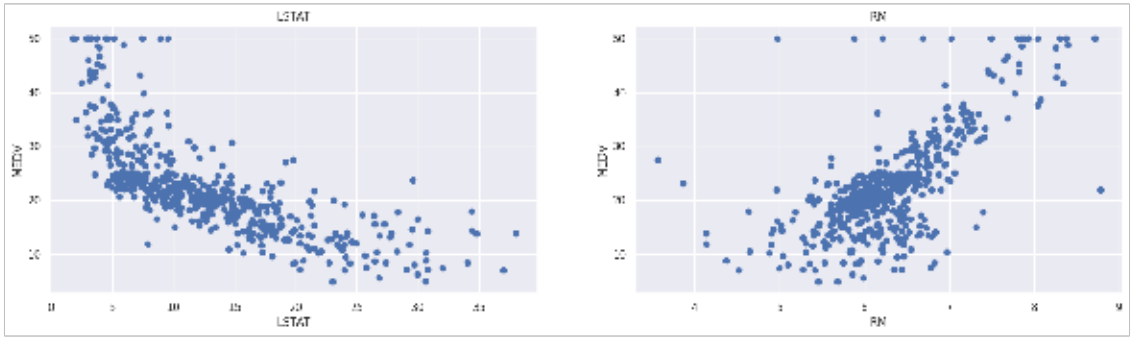
5 Observations

From the above coorelation plot we can see that MEDV is strongly correlated to LSTAT, RM

RAD and TAX are stronly correlated, so we don't include this in our features together to avoid multi-colinearity

[7]:
```
plt.figure(figsize=(20, 5))

features = ["LSTAT", "RM"]
target = boston["MEDV"]
```

```
for i, col in enumerate(features):
    plt.subplot(1, len(features), i + 1)
    x = boston[col]
    y = target
    plt.scatter(x, y, marker="o")
    plt.title(col)
    plt.xlabel(col)
    plt.ylabel("MEDV")
```



ex04-logistic-regression

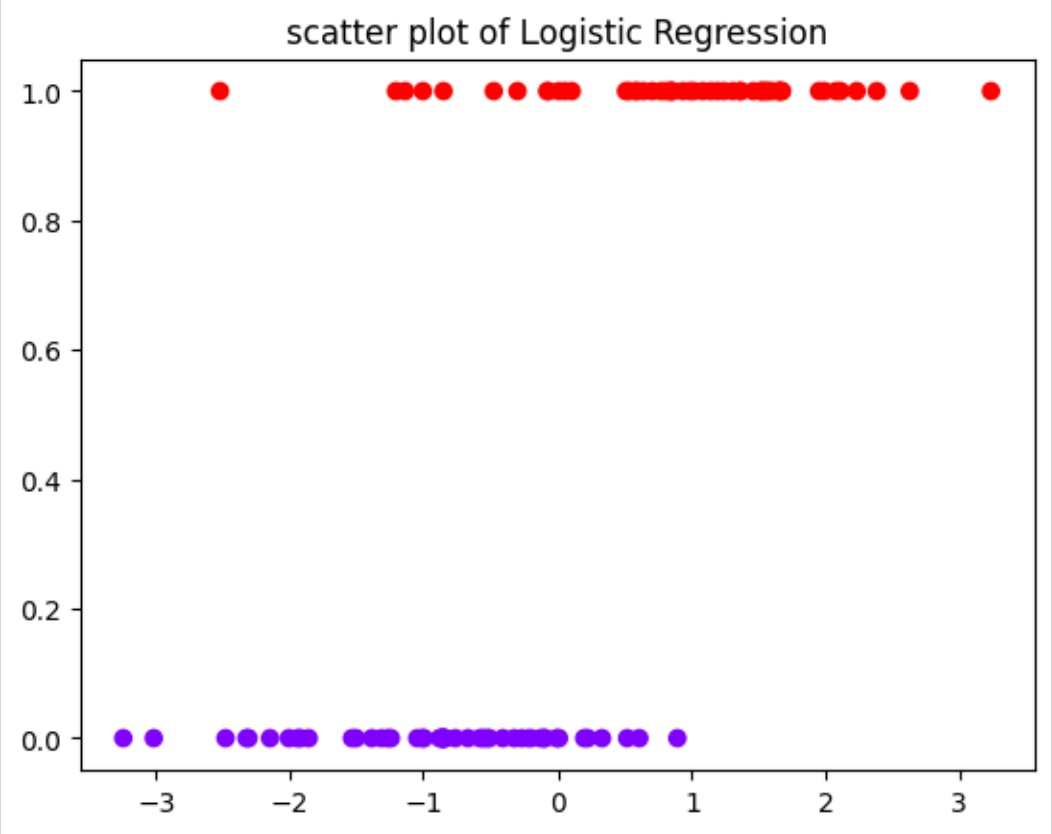
October 16, 2024

[1]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix

from sklearn.datasets import make_classification
from matplotlib import pyplot as plt
```

[2]:
```
x, y = make_classification(
    n_samples=100,
    n_features=1,
    n_classes=2,
    n_clusters_per_class=1,
    flip_y=0.03,
    n_informative=1,
    n_redundant=0,
    n_repeated=0,
)
print(y)
```

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

[3]:
```
plt.scatter(x, y, c=y, cmap="rainbow")
plt.title("scatter plot of Logistic Regression")
plt.show()
```



[4]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y, random_state=1)
```

[5]:
```
x_train.shape
```

[5]:
```
(75, 1)
```

[6]:
```
# step 5 perform logidtic regression
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

[6]:
```
LogisticRegression()
```

[7]:
```
# step 6 Make prediction using thr model
# performs prediction using the test dataset
y_pred = log_reg.predict(x_test)
```

[8]:
```
# step 7 Display the Confusion matrix
confusion_matrix(y_test, y_pred)
```

[8]:
```
array([[ 8,  1],
       [ 6, 10]])
```

[9]:
```
df = pd.read_csv("datasets/insurance_data.csv")
```

[10]:
```
df
```

[10]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1
5	56	1
6	55	0
7	60	1
8	62	1
9	61	1
10	18	0
11	28	0
12	27	0
13	29	0
14	49	1
15	55	1
16	25	1
17	58	1
18	19	0
19	18	0
20	21	0
21	26	0
22	40	1
23	45	1
24	50	1
25	54	1
26	23	0

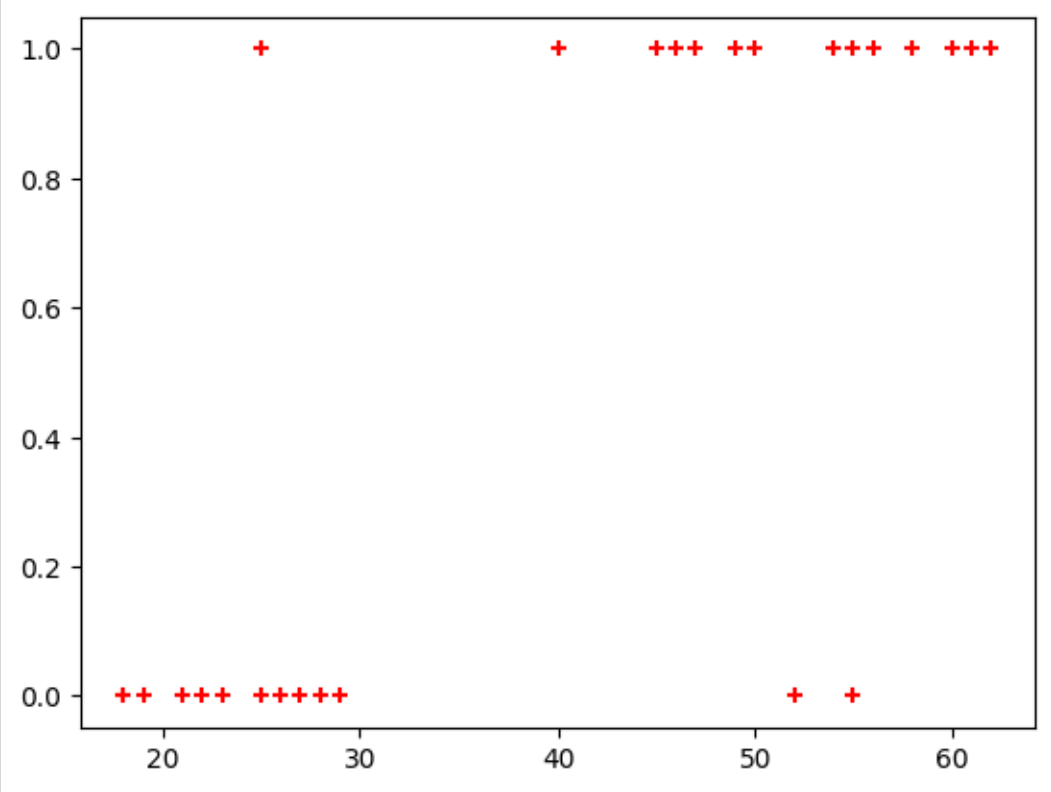
[11]:
```
df.head()
```

[11]:

	age	bought_insurance
0	22	0
1	25	0
2	47	1
3	52	0
4	46	1

[12]:
```
plt.scatter(df.age, df.bought_insurance, marker="+", color="red")
```

[12]:
```
<matplotlib.collections.PathCollection at 0x7f4ce327e510>
```



[13]:
```
df.shape
```

[13]:
```
(27, 2)
```

[14]:
```
x_train, x_test, y_train, y_test = train_test_split(
    df[["age"]], df.bought_insurance, test_size=0.1
)
```

[15]:
```
x_test
```

[15]:
```
    age
21   26
5    56
18   19
```

[16]:
```
model = LogisticRegression()
```

[17]:
```
model.fit(x_train, y_train)
```

[17]:
```
LogisticRegression()
```

[18]:
```
model.predict(x_test)
```

[18]:
```
array([0, 1, 0])
```

[19]:
```
model.score(x_test, y_test)
```

[19]:
```
1.0
```

[20]:
```
model.predict_proba(x_test)
```

[20]:
```
array([[0.82649559, 0.17350441],
       [0.09917355, 0.90082645],
       [0.91983146, 0.08016854]])
```

[21]:
```
df.describe()
```

[21]:

	age	bought_insurance
count	27.000000	27.000000
mean	39.666667	0.518519
std	15.745573	0.509175
min	18.000000	0.000000
25%	25.000000	0.000000
50%	45.000000	1.000000
75%	54.500000	1.000000
max	62.000000	1.000000

ex05-k-means-hierarchical-clustering

October 16, 2024

[1]:
```
import pandas as pd
from matplotlib import pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
```

[2]:
```
df = pd.read_csv("datasets/Mall_Customers.csv")
# loads the csv file into a pandas dataframe
df
```

[2]:

	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
4	5	Female	31	17	40
..	…	…	…	…	…
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

[200 rows x 5 columns]

[3]:
```
df.head()
```

[3]:

	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
4	5	Female	31	17	40

[4]:
```
df.shape
```

[4]:
```
(200, 5)
```

[5]:
```
df.info()  # with the help of it we get brief information about our dataset
```

```
<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
```

[6]:
```
# one way to access the annual income and spending score column
df.iloc[:, [3, 4]]
```

[6]:

	Annual Income (k$)	Spending Score (1-100)
0	15	39
1	15	81
2	16	6
3	16	77
4	17	40
..	…	…
195	120	79
196	126	28
197	126	74
198	137	18
199	137	83

[200 rows x 2 columns]

[7]:
```
x = df.loc[:, "Annual Income (k$)":"Spending Score (1-100)"].values
```

[8]:
```
x
```

[8]:
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
```

```
[ 20,  77],
[ 20,  13],
[ 20,  79],
[ 21,  35],
[ 21,  66],
[ 23,  29],
[ 23,  98],
[ 24,  35],
[ 24,  73],
[ 25,   5],
[ 25,  73],
[ 28,  14],
[ 28,  82],
[ 28,  32],
[ 28,  61],
[ 29,  31],
[ 29,  87],
[ 30,   4],
[ 30,  73],
[ 33,   4],
[ 33,  92],
[ 33,  14],
[ 33,  81],
[ 34,  17],
[ 34,  73],
[ 37,  26],
[ 37,  75],
[ 38,  35],
[ 38,  92],
[ 39,  36],
[ 39,  61],
[ 39,  28],
[ 39,  65],
[ 40,  55],
[ 40,  47],
[ 40,  42],
[ 40,  42],
[ 42,  52],
[ 42,  60],
[ 43,  54],
[ 43,  60],
[ 43,  45],
[ 43,  41],
[ 44,  50],
[ 44,  46],
[ 46,  51],
[ 46,  46],
```

```
[ 46,  56],
[ 46,  55],
[ 47,  52],
[ 47,  59],
[ 48,  51],
[ 48,  59],
[ 48,  50],
[ 48,  48],
[ 48,  59],
[ 48,  47],
[ 49,  55],
[ 49,  42],
[ 50,  49],
[ 50,  56],
[ 54,  47],
[ 54,  54],
[ 54,  53],
[ 54,  48],
[ 54,  52],
[ 54,  42],
[ 54,  51],
[ 54,  55],
[ 54,  41],
[ 54,  44],
[ 54,  57],
[ 54,  46],
[ 57,  58],
[ 57,  55],
[ 58,  60],
[ 58,  46],
[ 59,  55],
[ 59,  41],
[ 60,  49],
[ 60,  40],
[ 60,  42],
[ 60,  52],
[ 60,  47],
[ 60,  50],
[ 61,  42],
[ 61,  49],
[ 62,  41],
[ 62,  48],
[ 62,  59],
[ 62,  55],
[ 62,  56],
[ 62,  42],
[ 63,  50],
```

```
[ 63,  46],
[ 63,  43],
[ 63,  48],
[ 63,  52],
[ 63,  54],
[ 64,  42],
[ 64,  46],
[ 65,  48],
[ 65,  50],
[ 65,  43],
[ 65,  59],
[ 67,  43],
[ 67,  57],
[ 67,  56],
[ 67,  40],
[ 69,  58],
[ 69,  91],
[ 70,  29],
[ 70,  77],
[ 71,  35],
[ 71,  95],
[ 71,  11],
[ 71,  75],
[ 71,   9],
[ 71,  75],
[ 72,  34],
[ 72,  71],
[ 73,   5],
[ 73,  88],
[ 73,   7],
[ 73,  73],
[ 74,  10],
[ 74,  72],
[ 75,   5],
[ 75,  93],
[ 76,  40],
[ 76,  87],
[ 77,  12],
[ 77,  97],
[ 77,  36],
[ 77,  74],
[ 78,  22],
[ 78,  90],
[ 78,  17],
[ 78,  88],
[ 78,  20],
[ 78,  76],
```

```
[ 78,  16],
[ 78,  89],
[ 78,   1],
[ 78,  78],
[ 78,   1],
[ 78,  73],
[ 79,  35],
[ 79,  83],
[ 81,   5],
[ 81,  93],
[ 85,  26],
[ 85,  75],
[ 86,  20],
[ 86,  95],
[ 87,  27],
[ 87,  63],
[ 87,  13],
[ 87,  75],
[ 87,  10],
[ 87,  92],
[ 88,  13],
[ 88,  86],
[ 88,  15],
[ 88,  69],
[ 93,  14],
[ 93,  90],
[ 97,  32],
[ 97,  86],
[ 98,  15],
[ 98,  88],
[ 99,  39],
[ 99,  97],
[101,  24],
[101,  68],
[103,  17],
[103,  85],
[103,  23],
[103,  69],
[113,   8],
[113,  91],
[120,  16],
[120,  79],
[126,  28],
[126,  74],
[137,  18],
[137,  83]])
```

1 Exploratory Data Analysis (EDA)

[9]:
```
# Renaming a column in the dataset
df.rename(
    columns={"Genre": "Gender"}, inplace=True
)  # To rename column 2 from Genre to Gender
df.head()  # Checking if the correction has been effected
```

[9]:

	CustomerID	Gender	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
4	5	Female	31	17	40

[10]:
```
# Checking data types and shape
df.dtypes  # returns the data types of the variables
```

[10]:
```
CustomerID                int64
Gender                   object
Age                       int64
Annual Income (k$)        int64
Spending Score (1-100)    int64
dtype: object
```

[11]:
```
# Descriptive statistics
df.describe()  # returns the descriptive statistics of the dataset.
```

[11]:

	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

[12]:
```
# Looking for null or missing values
df.isnull().sum()  # returns the number of missing values
```

[12]:
```
CustomerID                0
Gender                    0
Age                       0
Annual Income (k$)        0
Spending Score (1-100)    0
dtype: int64
```

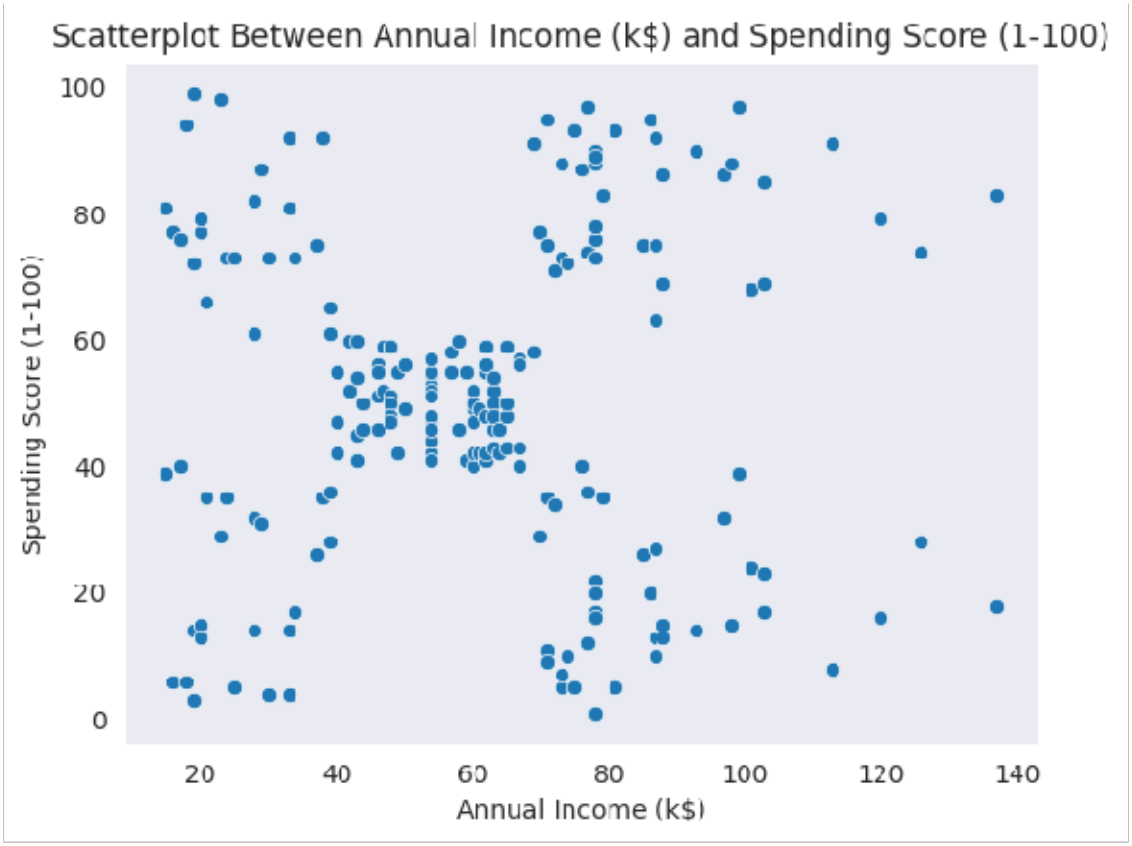
[13]:
```
# Looking for duplicated values
df.duplicated()  # Checking for duplicate values.
```

[13]:
```
0      False
1      False
2      False
3      False
4      False
       …
195    False
196    False
197    False
198    False
199    False
Length: 200, dtype: bool
```

2 Bivariate Analysis — Scatterplot

[14]:
```
sns.set_style("dark")
sns.scatterplot(x="Annual Income (k$)", y="Spending Score (1-100)", data=df)
plt.xlabel("Annual Income (k$)")
plt.ylabel("Spending Score (1-100)")
plt.title("Scatterplot Between Annual Income (k$) and Spending Score (1-100)")
```

[14]:
```
Text(0.5, 1.0, 'Scatterplot Between Annual Income (k$) and Spending Score
(1-100)')
```



[15]:
```
# Feature Selection(Choosing the columns of interest for clustering)
X = df.loc[:, ["Annual Income (k$)", "Spending Score (1-100)"]].values
X
```

[15]:
```
array([[ 15,  39],
       [ 15,  81],
       [ 16,   6],
       [ 16,  77],
       [ 17,  40],
       [ 17,  76],
       [ 18,   6],
       [ 18,  94],
       [ 19,   3],
       [ 19,  72],
       [ 19,  14],
       [ 19,  99],
       [ 20,  15],
       [ 20,  77],
       [ 20,  13],
       [ 20,  79],
       [ 21,  35],
```

```
[ 21,  66],
[ 23,  29],
[ 23,  98],
[ 24,  35],
[ 24,  73],
[ 25,   5],
[ 25,  73],
[ 28,  14],
[ 28,  82],
[ 28,  32],
[ 28,  61],
[ 29,  31],
[ 29,  87],
[ 30,   4],
[ 30,  73],
[ 33,   4],
[ 33,  92],
[ 33,  14],
[ 33,  81],
[ 34,  17],
[ 34,  73],
[ 37,  26],
[ 37,  75],
[ 38,  35],
[ 38,  92],
[ 39,  36],
[ 39,  61],
[ 39,  28],
[ 39,  65],
[ 40,  55],
[ 40,  47],
[ 40,  42],
[ 40,  42],
[ 42,  52],
[ 42,  60],
[ 43,  54],
[ 43,  60],
[ 43,  45],
[ 43,  41],
[ 44,  50],
[ 44,  46],
[ 46,  51],
[ 46,  46],
[ 46,  56],
[ 46,  55],
[ 47,  52],
[ 47,  59],
```

```
[ 48,  51],
[ 48,  59],
[ 48,  50],
[ 48,  48],
[ 48,  59],
[ 48,  47],
[ 49,  55],
[ 49,  42],
[ 50,  49],
[ 50,  56],
[ 54,  47],
[ 54,  54],
[ 54,  53],
[ 54,  48],
[ 54,  52],
[ 54,  42],
[ 54,  51],
[ 54,  55],
[ 54,  41],
[ 54,  44],
[ 54,  57],
[ 54,  46],
[ 57,  58],
[ 57,  55],
[ 58,  60],
[ 58,  46],
[ 59,  55],
[ 59,  41],
[ 60,  49],
[ 60,  40],
[ 60,  42],
[ 60,  52],
[ 60,  47],
[ 60,  50],
[ 61,  42],
[ 61,  49],
[ 62,  41],
[ 62,  48],
[ 62,  59],
[ 62,  55],
[ 62,  56],
[ 62,  42],
[ 63,  50],
[ 63,  46],
[ 63,  43],
[ 63,  48],
[ 63,  52],
```

```
[ 63,  54],
[ 64,  42],
[ 64,  46],
[ 65,  48],
[ 65,  50],
[ 65,  43],
[ 65,  59],
[ 67,  43],
[ 67,  57],
[ 67,  56],
[ 67,  40],
[ 69,  58],
[ 69,  91],
[ 70,  29],
[ 70,  77],
[ 71,  35],
[ 71,  95],
[ 71,  11],
[ 71,  75],
[ 71,   9],
[ 71,  75],
[ 72,  34],
[ 72,  71],
[ 73,   5],
[ 73,  88],
[ 73,   7],
[ 73,  73],
[ 74,  10],
[ 74,  72],
[ 75,   5],
[ 75,  93],
[ 76,  40],
[ 76,  87],
[ 77,  12],
[ 77,  97],
[ 77,  36],
[ 77,  74],
[ 78,  22],
[ 78,  90],
[ 78,  17],
[ 78,  88],
[ 78,  20],
[ 78,  76],
[ 78,  16],
[ 78,  89],
[ 78,   1],
[ 78,  78],
```

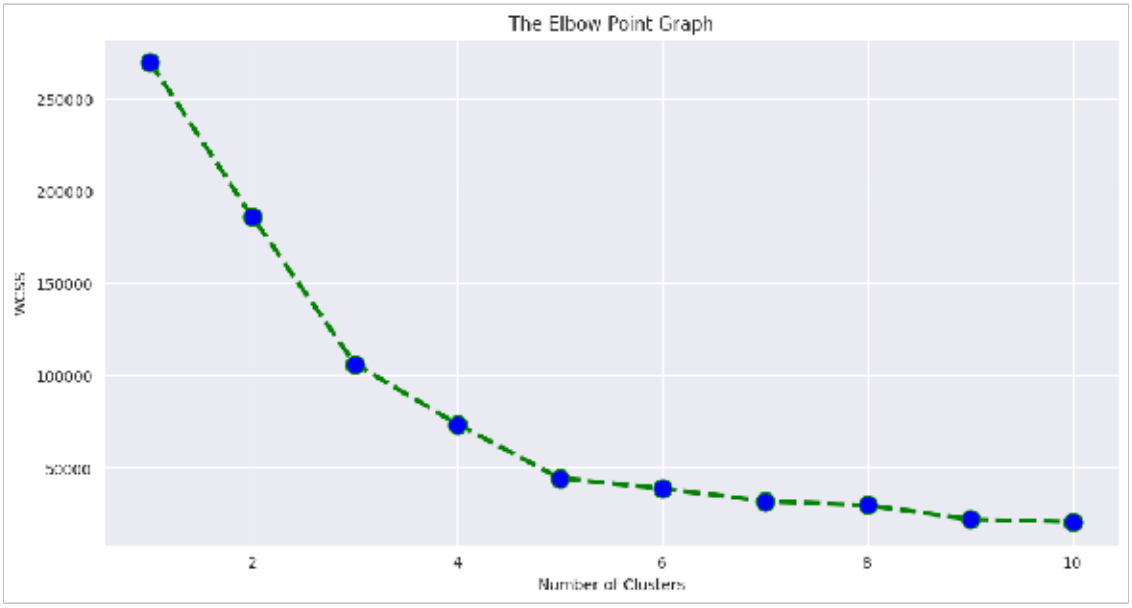
```
[ 78,   1],
[ 78,  73],
[ 79,  35],
[ 79,  83],
[ 81,   5],
[ 81,  93],
[ 85,  26],
[ 85,  75],
[ 86,  20],
[ 86,  95],
[ 87,  27],
[ 87,  63],
[ 87,  13],
[ 87,  75],
[ 87,  10],
[ 87,  92],
[ 88,  13],
[ 88,  86],
[ 88,  15],
[ 88,  69],
[ 93,  14],
[ 93,  90],
[ 97,  32],
[ 97,  86],
[ 98,  15],
[ 98,  88],
[ 99,  39],
[ 99,  97],
[101,  24],
[101,  68],
[103,  17],
[103,  85],
[103,  23],
[103,  69],
[113,   8],
[113,  91],
[120,  16],
[120,  79],
[126,  28],
[126,  74],
[137,  18],
[137,  83]])
```

3 Step 2: Perform Elbow Method To Find Optimal No.Of Clusters

[16]:
```
wcss = []
```

[17]:
```
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init="k-means++", random_state=0)
    kmeans.fit(x)
    wcss.append(kmeans.inertia_)
```

[18]:
```
plt.figure(figsize=(12, 6))
plt.grid()
plt.plot(
    range(1, 11),
    wcss,
    color="green",
    linestyle="dashed",
    linewidth=3,
    marker="o",
    markerfacecolor="blue",
    markersize=12,
)
plt.title("The Elbow Point Graph")
plt.xlabel("Number of Clusters")
plt.ylabel("WCSS")
plt.show()
```



4 Training the K-Means Clustering Model

[19]:
```
kmeans = KMeans(n_clusters=5, init="k-means++")  # initialize the class object
label = kmeans.fit_predict(X)  # returns a cluster number for each of the data␣
 ↪points
print(label)
```

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

5 Checking the centers of out clusters (Also known as Centroids)

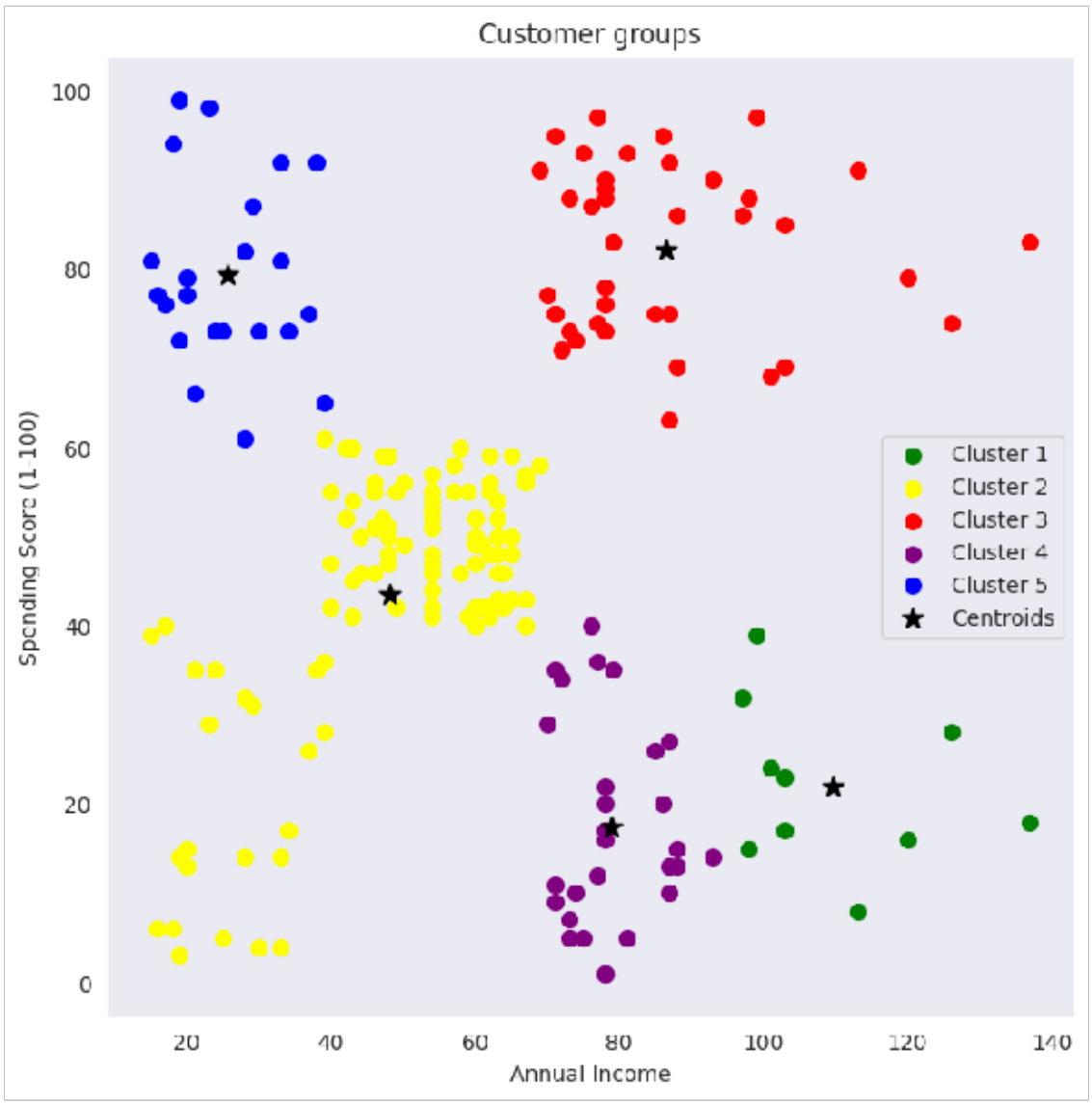
[20]:
```
print(kmeans.cluster_centers_)
```

```
[[109.7         22.        ]
 [ 48.16831683  43.3960396 ]
 [ 86.53846154  82.12820513]
 [ 78.89285714  17.42857143]
 [ 25.72727273  79.36363636]]
```

6 Visualizing all the clusters

[21]:
```
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 8))

# Scatter plot for 5 clusters
plt.scatter(X[label == 0, 0], X[label == 0, 1], s=50, c="green", label="Cluster␣
 ↪1")
plt.scatter(X[label == 1, 0], X[label == 1, 1], s=50, c="yellow",␣
 ↪label="Cluster 2")
plt.scatter(X[label == 2, 0], X[label == 2, 1], s=50, c="red", label="Cluster␣
 ↪3")
plt.scatter(X[label == 3, 0], X[label == 3, 1], s=50, c="purple",␣
 ↪label="Cluster 4")
plt.scatter(X[label == 4, 0], X[label == 4, 1], s=50, c="blue", label="Cluster␣
 ↪5")

# Scatter plot for cluster centers
plt.scatter(
    kmeans.cluster_centers_[:, 0],
    kmeans.cluster_centers_[:, 1],
```

```
    s=100,
    c="black",
    label="Centroids",
    marker="*",
)

plt.title("Customer groups")
plt.xlabel("Annual Income")
plt.ylabel("Spending Score (1-100)")
plt.legend()

plt.show()
```



ex06-dimensionality-reduction

October 16, 2024

[1]:
```
# Import necessary libraries
from sklearn import datasets  # to retrieve the iris Dataset
import pandas as pd  # to load the dataframe
from sklearn.preprocessing import StandardScaler  # to standardize the features
from sklearn.decomposition import PCA  # to apply PCA
import seaborn as sns  # to plot the heat maps
```

[2]:
```
# Load the Dataset
iris = datasets.load_iris()
# convert the dataset into a pandas data frame
df = pd.DataFrame(iris["data"], columns=iris["feature_names"])
# display the head (first 5 rows) of the dataset
df.head()
```

[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

[3]:
```
# Standardize the features
# Create an object of StandardScaler which is present in sklearn.preprocessing
scalar = StandardScaler()
scaled_data = pd.DataFrame(scalar.fit_transform(df))  # scaling the data
scaled_data
```

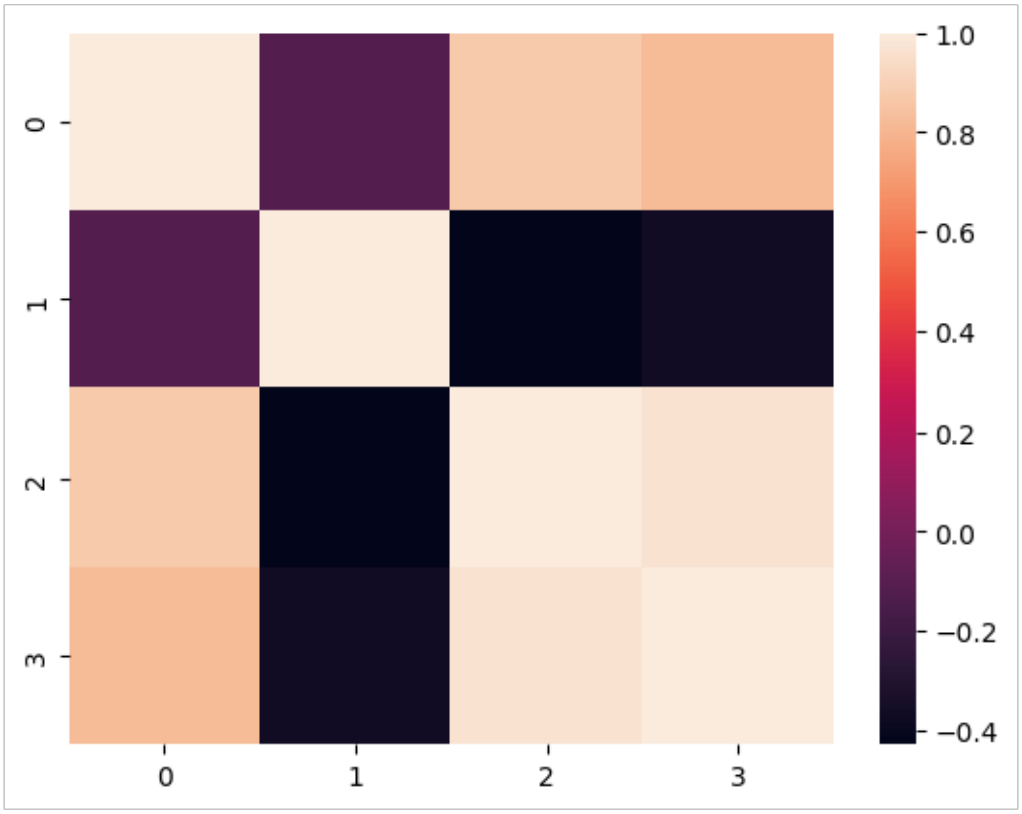
[3]:

	0	1	2	3
0	-0.900681	1.019004	-1.340227	-1.315444
1	-1.143017	-0.131979	-1.340227	-1.315444
2	-1.385353	0.328414	-1.397064	-1.315444
3	-1.506521	0.098217	-1.283389	-1.315444
4	-1.021849	1.249201	-1.340227	-1.315444
..	…	…	…	…
145	1.038005	-0.131979	0.819596	1.448832
146	0.553333	-1.282963	0.705921	0.922303
147	0.795669	-0.131979	0.819596	1.053935
148	0.432165	0.788808	0.933271	1.448832

```
149  0.068662 -0.131979  0.762758  0.790671

[150 rows x 4 columns]
```

[4]:
```
# Check the Co-relation between features without PCA
sns.heatmap(scaled_data.corr())
```

[4]:
```
<Axes: >
```



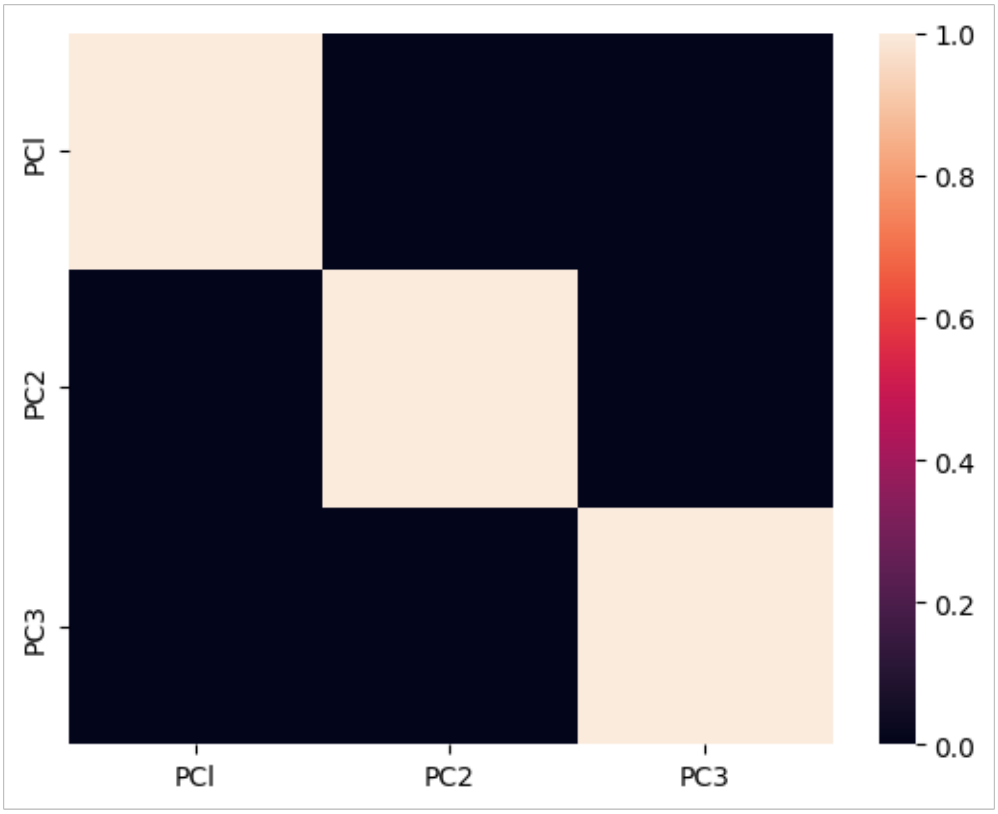
[5]:
```
# Applying PCA
# Taking no. of Principal Components as 3
pca = PCA(n_components=3)
pca.fit(scaled_data)
data_pca = pca.transform(scaled_data)
data_pca = pd.DataFrame(data_pca, columns=["PC1", "PC2", "PC3"])
data_pca.head()
```

[5]:
```
        PC1       PC2       PC3
0 -2.264703  0.480027 -0.127706
1 -2.080961 -0.674134 -0.234609
2 -2.364229 -0.341908  0.044201
3 -2.299384 -0.597395  0.091290
```

```
4 -2.389842  0.646835  0.015738
```

[6]:
```
# Checking Co-relation between features after PCA
sns.heatmap(data_pca.corr())
```

[6]:
```
<Axes: >
```



ex07-text-recognition

October 16, 2024

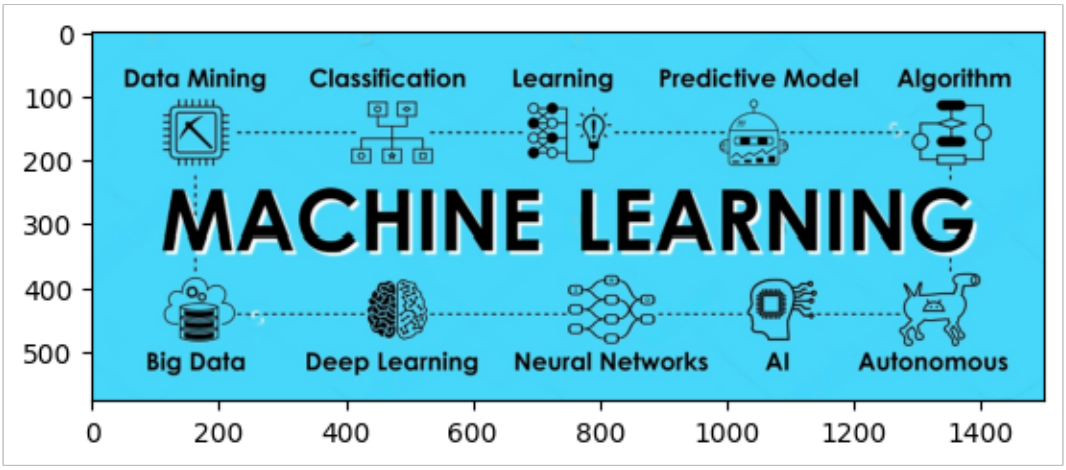
[1]:
```
import cv2
from matplotlib import pyplot as plt
import easyocr
```

[2]:
```
%%capture
# the %%capture will supress download progress messages
reader = easyocr.Reader(
    ["en"]
)  # this needs to run only once to load the model into memory
```

[3]:
```
IMAGE_PATH = "datasets/ocr.jpeg"
```

[4]:
```
img = cv2.imread(IMAGE_PATH)
plt.imshow(img)
```

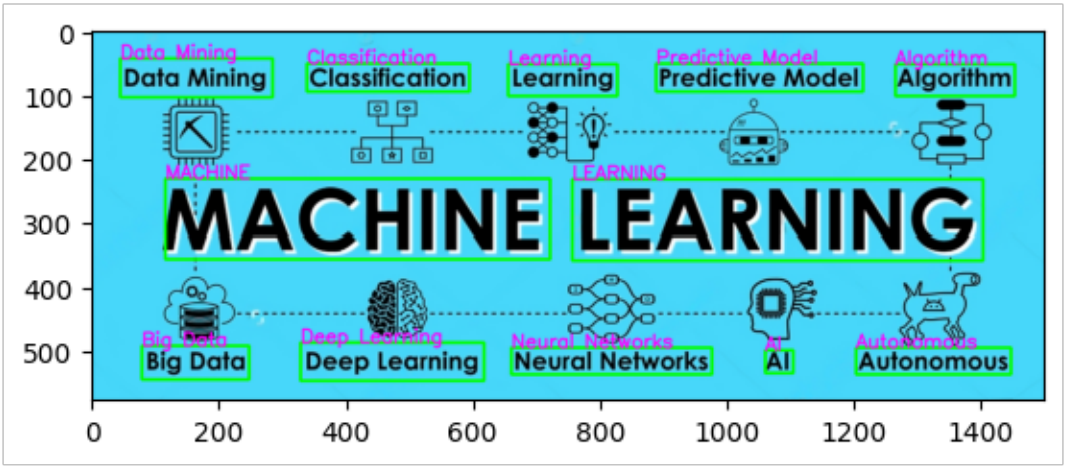
[4]:
```
<matplotlib.image.AxesImage at 0x7f78f3d3dd50>
```



[5]:
```
result = reader.readtext(IMAGE_PATH)
img = cv2.imread(IMAGE_PATH)
for finding in result:
    top_left = tuple(finding[0][0])
    bottom_right = tuple(finding[0][2])
```

```
    text = finding[1]
    font = cv2.FONT_HERSHEY_SIMPLEX
    img = cv2.rectangle(img, top_left, bottom_right, (0, 255, 0), 3)
    img = cv2.putText(img, text, top_left, font, 1, (255, 0, 255), 2, cv2.
 ↪LINE_AA)

print([r[1] for r in result])

plt.imshow(img)
plt.show()
```

```
['Data Mining', 'Classification', 'Learning', 'Predictive Model', 'Algorithm',
'MACHINE', 'LEARNING', 'Big Data', 'Deep Learning', 'Neural Networks', 'Al',
'Autonomous']
```



ex08-support-vector-machine

October 16, 2024

1 Import the dataset

[1]:
```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```
spam = pd.read_csv("datasets/spam.csv")
spam.head()
```

[2]:

	Label	EmailText
0	ham	Go until jurong point, crazy.. Available only …
1	ham	Ok lar… Joking wif u oni…
2	spam	Free entry in 2 a wkly comp to win FA Cup fina…
3	ham	U dun say so early hor… U c already then say…
4	ham	Nah I don't think he goes to usf, he lives aro…

2 Check the shape of the dataset

[3]:
```
spam.shape
```

[3]:
```
(5572, 2)
```

3 Check the columns present in the dataset

[4]:
```
spam.columns
```

[4]:
```
Index(['Label', 'EmailText'], dtype='object')
```

4 Check the descriptive statistics of the dataset

[5]:
```
spam.describe()
```

[5]:

	Label	EmailText
count	5572	5572
unique	2	5169

```
top      ham  Sorry, I'll call later
freq    4825                      30
```

5 Check the info of the dataset

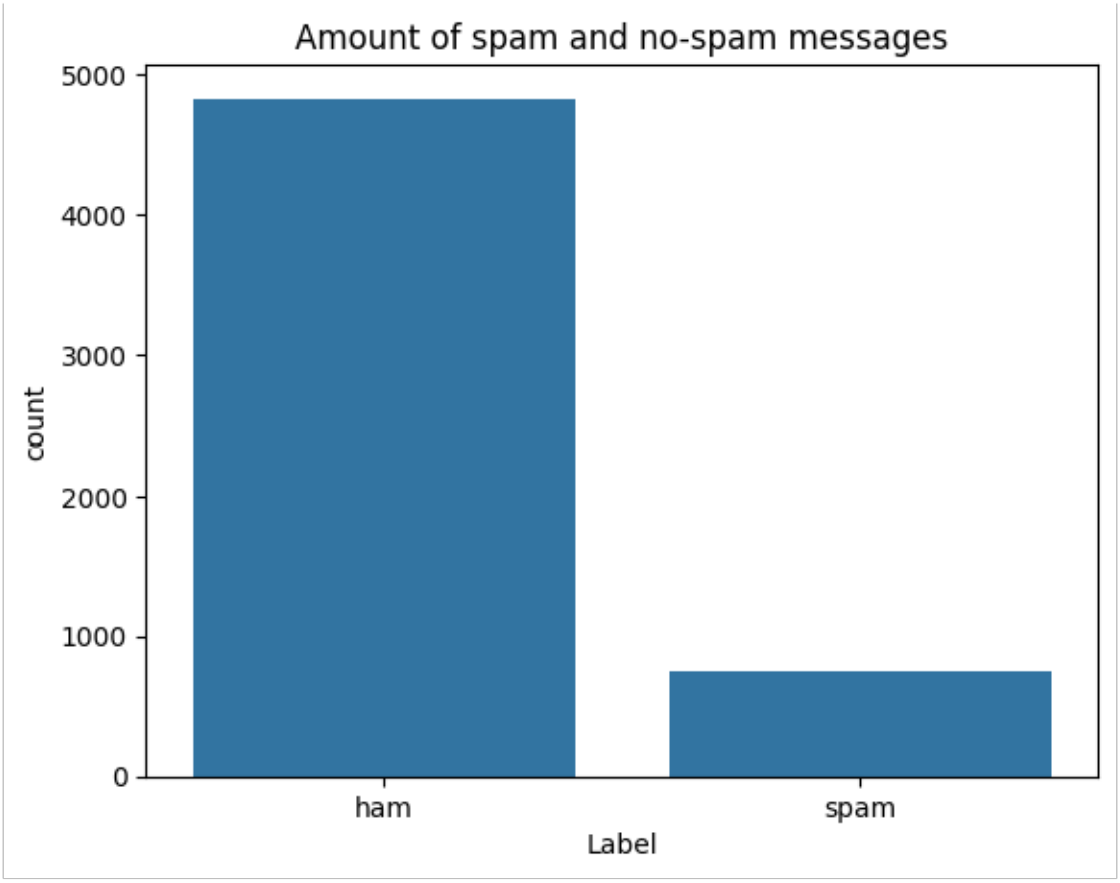
[6]:
```
spam.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5572 entries, 0 to 5571
Data columns (total 2 columns):
 #   Column     Non-Null Count  Dtype 
---  ------     --------------  ----- 
 0   Label      5572 non-null   object
 1   EmailText  5572 non-null   object
dtypes: object(2)
memory usage: 87.2+ KB
```

[7]:
```
spam["Label"].value_counts()
```

[7]:
```
Label
ham     4825
spam     747
Name: count, dtype: int64
```

[8]:
```
sns.countplot(data=spam, x=spam["Label"]).set_title(
    "Amount of spam and no-spam messages"
)
plt.show()
```

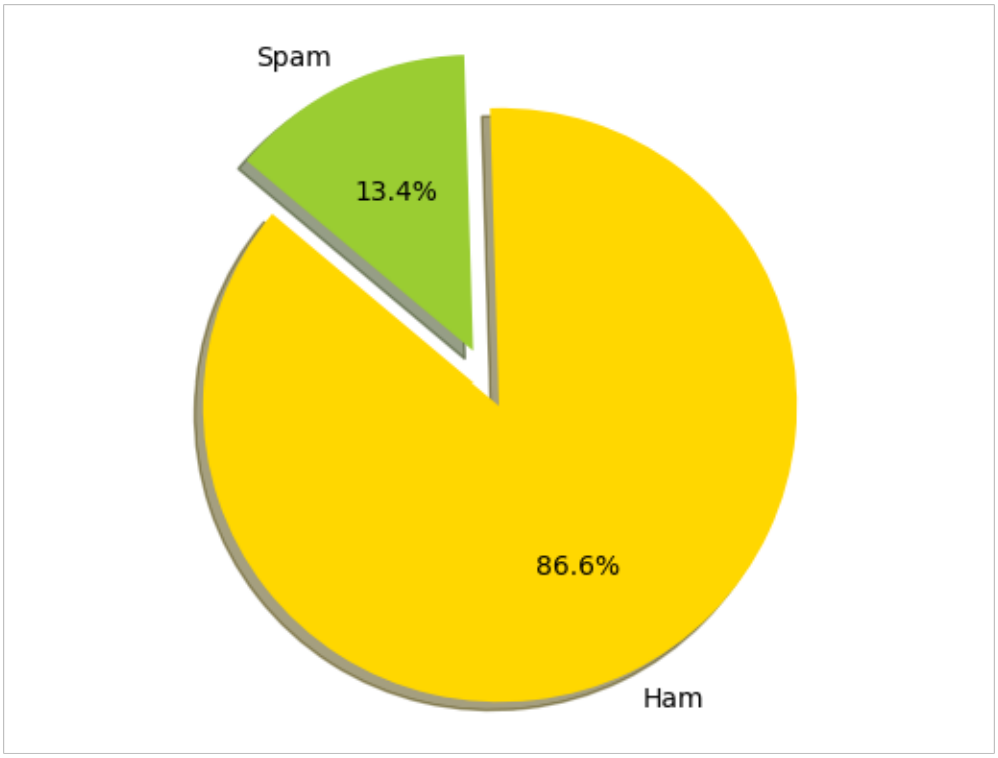


6 Plotting Pie-Chart

[9]:
```
count_Class = pd.value_counts(spam.Label, sort=True)
# Data to plot
labels = "Ham", "Spam"
sizes = [count_Class[0], count_Class[1]]
colors = ["gold", "yellowgreen"]  # 'lightcoral', 'lightskyblue'
explode = (0.1, 0.1)  # explode 1st slice

plt.pie(
    sizes,
    explode=explode,
    labels=labels,
    colors=colors,
    autopct="%1.1f%%",
    shadow=True,
    startangle=140,
)
plt.axis("equal")
```

```
plt.show()
```

```
/tmp/ipykernel_6150/3052618870.py:1: FutureWarning: pandas.value_counts is
deprecated and will be removed in a future version. Use
pd.Series(obj).value_counts() instead.
  count_Class = pd.value_counts(spam.Label, sort=True)
/tmp/ipykernel_6150/3052618870.py:4: FutureWarning: Series.__getitem__ treating
keys as positions is deprecated. In a future version, integer keys will always
be treated as labels (consistent with DataFrame behavior). To access a value by
position, use `ser.iloc[pos]`
  sizes = [count_Class[0], count_Class[1]]
```



7 Extract the independent variables to create a dataframe X

[10]:
```
X = spam["EmailText"]
X.head()
```

[10]:
```
0    Go until jurong point, crazy.. Available only …
1                        Ok lar… Joking wif u oni…
2    Free entry in 2 a wkly comp to win FA Cup fina…
3    U dun say so early hor… U c already then say…
4    Nah I don't think he goes to usf, he lives aro…
Name: EmailText, dtype: object
```

8 Extract the dependent variables to create a dataframe y

[11]:
```
y = spam["Label"]
y.head()
```

[11]:
```
0     ham
1     ham
2    spam
3     ham
4     ham
Name: Label, dtype: object
```

9 Split X and y into train and test dataset with test_size = 0.20, random_state=0

[12]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.20, random_state=0
)
```

10 Check the shape of X and y of train dataset

[13]:
```
print(X_train.shape)
print(y_train.shape)
```

```
(4457,)
(4457,)
```

11 Check the shape of X and y of test dataset

[14]:
```
print(X_test.shape)
print(y_test.shape)
```

```
(1115,)
(1115,)
```

12 Applying various models of Machine Learning

[15]:
```
from sklearn.feature_extraction.text import CountVectorizer

cv = CountVectorizer()
trainCV = cv.fit_transform(X_train)
testCV = cv.transform(X_test)
```

[16]:
```
from sklearn.naive_bayes import MultinomialNB

naive_bayes = MultinomialNB()
naive_bayes.fit(trainCV, y_train)
pred_NB = naive_bayes.predict(testCV)
```

[17]:
```
from sklearn.metrics import accuracy_score

Accuracy_Score_NB = accuracy_score(y_test, pred_NB)
Accuracy_Score_NB
```

[17]:
```
0.9874439461883409
```

[18]:
```
from sklearn.neighbors import KNeighborsClassifier

classifier_knn = KNeighborsClassifier()
classifier_knn.fit(trainCV, y_train)
pred_knn = classifier_knn.predict(testCV)
```

[19]:
```
Accuracy_Score_knn = accuracy_score(y_test, pred_knn)
Accuracy_Score_knn
```

[19]:
```
0.9085201793721973
```

[20]:
```
from sklearn.svm import SVC

classifier_svm_linear = SVC(kernel="linear")
classifier_svm_linear.fit(trainCV, y_train)
pred_svm_linear = classifier_svm_linear.predict(testCV)
```

[21]:
```
Accuracy_Score_SVM_Linear = accuracy_score(y_test, pred_svm_linear)
Accuracy_Score_SVM_Linear
```

[21]:
```
0.9811659192825112
```

[22]:
```
classifier_svm_rbf = SVC(kernel="rbf")
classifier_svm_rbf.fit(trainCV, y_train)
pred_svm_rbf = classifier_svm_rbf.predict(testCV)
```

[23]:
```
Accuracy_Score_SVM_Gaussion = accuracy_score(y_test, pred_svm_rbf)
Accuracy_Score_SVM_Gaussion
```

[23]:
```
0.9766816143497757
```

[24]:
```
classifier_svm_poly = SVC(kernel="poly")
classifier_svm_poly.fit(trainCV, y_train)
pred_svm_poly = classifier_svm_poly.predict(testCV)
```

[25]:
```
Accuracy_Score_SVM_Polynomial = accuracy_score(y_test, pred_svm_poly)
Accuracy_Score_SVM_Polynomial
```

[25]: 0.9417040358744395

[26]:
```
classifier_svm_sigmoid = SVC(kernel="sigmoid")
classifier_svm_sigmoid.fit(trainCV, y_train)
pred_svm_sigmoid = classifier_svm_sigmoid.predict(testCV)
```

[27]:
```
Accuracy_Score_svm_Sigmoid = accuracy_score(y_test, pred_svm_sigmoid)
Accuracy_Score_svm_Sigmoid
```

[27]: 0.9300448430493273

[28]:
```
from sklearn.tree import DecisionTreeClassifier

classifier_dt = DecisionTreeClassifier()
classifier_dt.fit(trainCV, y_train)
pred_dt = classifier_dt.predict(testCV)
```

[29]:
```
Accuracy_Score_dt = accuracy_score(y_test, pred_dt)
Accuracy_Score_dt
```

[29]: 0.9659192825112107

[30]:
```
from sklearn.ensemble import RandomForestClassifier

classifier_rf = RandomForestClassifier()
classifier_rf.fit(trainCV, y_train)
pred_rf = classifier_rf.predict(testCV)
```

[31]:
```
Accuracy_Score_rf = accuracy_score(y_test, pred_rf)
Accuracy_Score_rf
```

[31]: 0.9713004484304932

[32]:
```
print("K-Nearest Neighbors =", Accuracy_Score_knn)
print("Naive Bayes =", Accuracy_Score_NB)
print("Support Vector Machine Linear =", Accuracy_Score_SVM_Linear)
print("Support Vector Machine Gaussion =", Accuracy_Score_SVM_Gaussion)
print("Support Vector Machine Polynomial =", Accuracy_Score_SVM_Polynomial)
print("Support Vector Machine Sigmoid =", Accuracy_Score_svm_Sigmoid)
print("Decision Tree =", Accuracy_Score_dt)
print("Random Forest =", Accuracy_Score_rf)
```

```
K-Nearest Neighbors = 0.9085201793721973
Naive Bayes = 0.9874439461883409
Support Vector Machine Linear = 0.9811659192825112
Support Vector Machine Gaussion = 0.9766816143497757
```

```
Support Vector Machine Polynomial = 0.9417040358744395
Support Vector Machine Sigmoid = 0.9300448430493273
Decision Tree = 0.9659192825112107
Random Forest = 0.9713004484304932
```

ex09-naive-bayes-classifier

October 16, 2024

[1]:
```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.feature_extraction.text import TfidfTransformer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
```

[2]:
```
# We defined the categories which we want to classify
categories = ["rec.motorcycles", "sci.electronics", "comp.graphics", "sci.med"]
# sklearn provides us with subset data for training and testing
train_data = fetch_20newsgroups(
    subset="train", categories=categories, shuffle=True, random_state=42
)
print(train_data.target_names)
print("\n".join(train_data.data[0].split("\n")[:3]))
print(train_data.target_names[train_data.target[0]])
# Let's Look at categories of our first ten training data
for t in train_data.target[:10]:
    print(train_data.target_names[t])
```

```
['comp.graphics', 'rec.motorcycles', 'sci.electronics', 'sci.med']
From: kreyling@lds.loral.com (Ed Kreyling 6966)
Subject: Sun-os and 8bit ASCII graphics
Organization: Loral Data Systems
comp.graphics
comp.graphics
comp.graphics
rec.motorcycles
comp.graphics
sci.med
sci.electronics
sci.electronics
comp.graphics
rec.motorcycles
sci.electronics
```

[3]:
```
# Builds a dictionary of features and transforms documents to feature vectors␣
↪anc
```

```
# matrix of token counts (CountVectorizer)
count_vect = CountVectorizer()
X_train_counts = count_vect.fit_transform(train_data.data)
# transform a count matrix to a normalized tf-idf representation
tfidf_transformer = TfidfTransformer()
X_train_tfidf = tfidf_transformer.fit_transform(X_train_counts)
```

[4]:
```
# training our classifier ; train_data.target will be having numbers assigned␣
 ↪for
clf = MultinomialNB().fit(X_train_tfidf, train_data.target)
# Input Data to predict their classes of the given categories
docs_new = ["I have a Harley Davidson and Yamaha.", "I have a GTX 1050 GPU"]
# building up feature vector of our input
X_new_counts = count_vect.transform(docs_new)
# We call transform instead of fit_transform because it's already been fit
X_new_tfidf = tfidf_transformer.transform(X_new_counts)
```

[5]:
```
# predicting the category of our input text: Will give out number for category
predicted = clf.predict(X_new_tfidf)
for doc, category in zip(docs_new, predicted):
    print("%r => %s" % (doc, train_data.target_names[category]))
```

```
'I have a Harley Davidson and Yamaha.' => rec.motorcycles
'I have a GTX 1050 GPU' => sci.med
```

[6]:
```
text_clf = Pipeline(
    [
        ("vect", CountVectorizer()),
        ("tfidf", TfidfTransformer()),
        ("clf", MultinomialNB()),
    ]
)

# Fitting our train data to the pipeline
text_clf.fit(train_data.data, train_data.target)

# Test data
test_data = fetch_20newsgroups(
    subset="test", categories=categories, shuffle=True, random_state=42
)
docs_test = test_data.data

# Predicting our test data
predicted = text_clf.predict(docs_test)
print("We got an accuracy of", np.mean(predicted == test_data.target) * 100,␣
 ↪"%")
```

```
We got an accuracy of 91.49746192893402 %
```

ex10-online-fraud-detection

October 16, 2024

[1]:
```
import pandas as pd

# Load the creditcard.csv using pandas
datainput = pd.read_csv("datasets/creditcard.csv")
# https://www.Rkaggle.com/mlg-ulb/creditcardfraud
# Print the top 5 records
print(datainput[0:5])
# Print the complete shape of the dataset
print("Shape of Complete Data Set")
print(datainput.shape)
```

```
   Time        V1        V2        V3        V4        V5        V6        V7  \
0   0.0 -1.359807 -0.072781  2.536347  1.378155 -0.338321  0.462388  0.239599
1   0.0  1.191857  0.266151  0.166480  0.448154  0.060018 -0.082361 -0.078803
2   1.0 -1.358354 -1.340163  1.773209  0.379780 -0.503198  1.800499  0.791461
3   1.0 -0.966272 -0.185226  1.792993 -0.863291 -0.010309  1.247203  0.237609
4   2.0 -1.158233  0.877737  1.548718  0.403034 -0.407193  0.095921  0.592941

         V8        V9  …       V21       V22       V23       V24       V25  \
0  0.098698  0.363787  … -0.018307  0.277838 -0.110474  0.066928  0.128539
1  0.085102 -0.255425  … -0.225775 -0.638672  0.101288 -0.339846  0.167170
2  0.247676 -1.514654  …  0.247998  0.771679  0.909412 -0.689281 -0.327642
3  0.377436 -1.387024  … -0.108300  0.005274 -0.190321 -1.175575  0.647376
4 -0.270533  0.817739  … -0.009431  0.798278 -0.137458  0.141267 -0.206010

        V26       V27       V28  Amount  Class
0 -0.189115  0.133558 -0.021053  149.62      0
1  0.125895 -0.008983  0.014724    2.69      0
2 -0.139097 -0.055353 -0.059752  378.66      0
3 -0.221929  0.062723  0.061458  123.50      0
4  0.502292  0.219422  0.215153   69.99      0

[5 rows x 31 columns]
Shape of Complete Data Set
(284807, 31)
```

[2]:
```
cls = datainput.get("Class")
false = datainput[cls == 1]
```

```
true = datainput[cls == 0]
n = len(false) / float(len(true))
print(n)
print("False Detection Cases: {}".format(len(datainput[cls == 1])))
print("True Detection Cases: {}".format(len(datainput[cls == 0])))
```

```
0.0017304750013189597
False Detection Cases: 492
True Detection Cases: 284315
```

[3]:
```
# False Detection Cases
print("False Detection Cases")
print("----------------------")
print(false.Amount.describe())
# True Detection Cases
print("True Detection Cases")
print("----------------------")
print(true.Amount.describe())
```

```
False Detection Cases
----------------------
count     492.000000
mean      122.211321
std       256.683288
min         0.000000
25%         1.000000
50%         9.250000
75%       105.890000
max      2125.870000
Name: Amount, dtype: float64
True Detection Cases
----------------------
count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

[4]:
```
# separating features(X) and Label(y)
# Select all columns except the last for all rows
X = datainput.iloc[:, :-1].values
# Select the last column of all rows
Y = datainput.iloc[:, -1].values
print(X.shape)
```

```
print(Y.shape)
```

```
(284807, 30)
(284807,)
```

[5]:
```
from sklearn.model_selection import train_test_split

# train_test_split method
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

[6]:
```
from sklearn import metrics

# DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier

classifier = DecisionTreeClassifier(max_depth=4)
classifier.fit(X_train, Y_train)
predicted = classifier.predict(X_test)
print("predicted values :", predicted)

# Accuracy
DT = metrics.accuracy_score(Y_test, predicted) * 100
print("The accuracy score using the DecisionTreeClassifier : ", DT)
```

```
predicted values : [0 0 0 … 0 0 0]
The accuracy score using the DecisionTreeClassifier :  99.9367999719111
```

[7]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score

# Precision
print("precision")
# Precision = TP / (TP + FP) (Where TP = True Positive, TN = True Negative, FP =
precision = precision_score(Y_test, predicted, pos_label=1)
print(precision_score(Y_test, predicted, pos_label=1))
# Recall
print("recall")
# Recall = TP / (TP + FN)
recall = recall_score(Y_test, predicted, pos_label=1)
print(recall_score(Y_test, predicted, pos_label=1))
# f1-score
print("f-Score")
# F - scores are a statistical method for determining accuracy accounting for␣
 ↪bot
fscore = f1_score(Y_test, predicted, pos_label=1)
print(f1_score(Y_test, predicted, pos_label=1))
```

```
precision
```

```
0.8160919540229885
recall
0.7802197802197802
f-Score
0.797752808988764
```