Introduction to AI and ML (CSE326)
Practical File

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

Submitted to:	Submitted by:
Dr Aakanshi Gupta	Shubham Garg
	A2305219420
	7CSE-6Y

AMITY SCHOOL OF ENGINEERING AND TECHNOLOGY
AMITY UNIVERSITY UTTAR PRADESH
NOIDA – 201301

INDEX

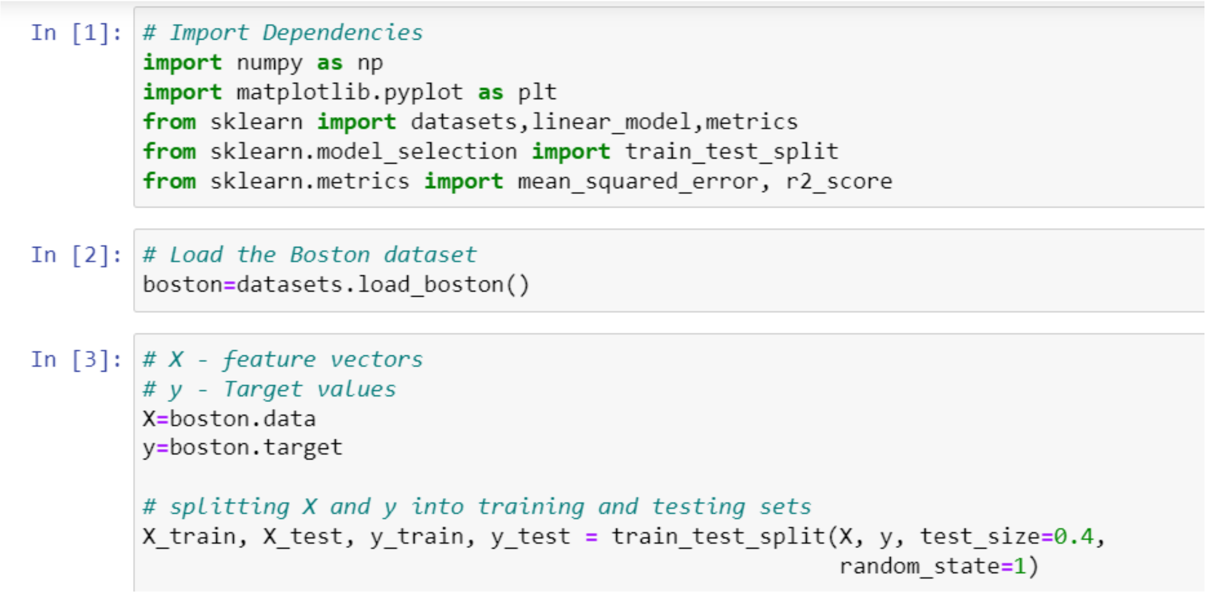
Experiment-1

Aim: WAP to Implement Linear Regression on Given Data Set.

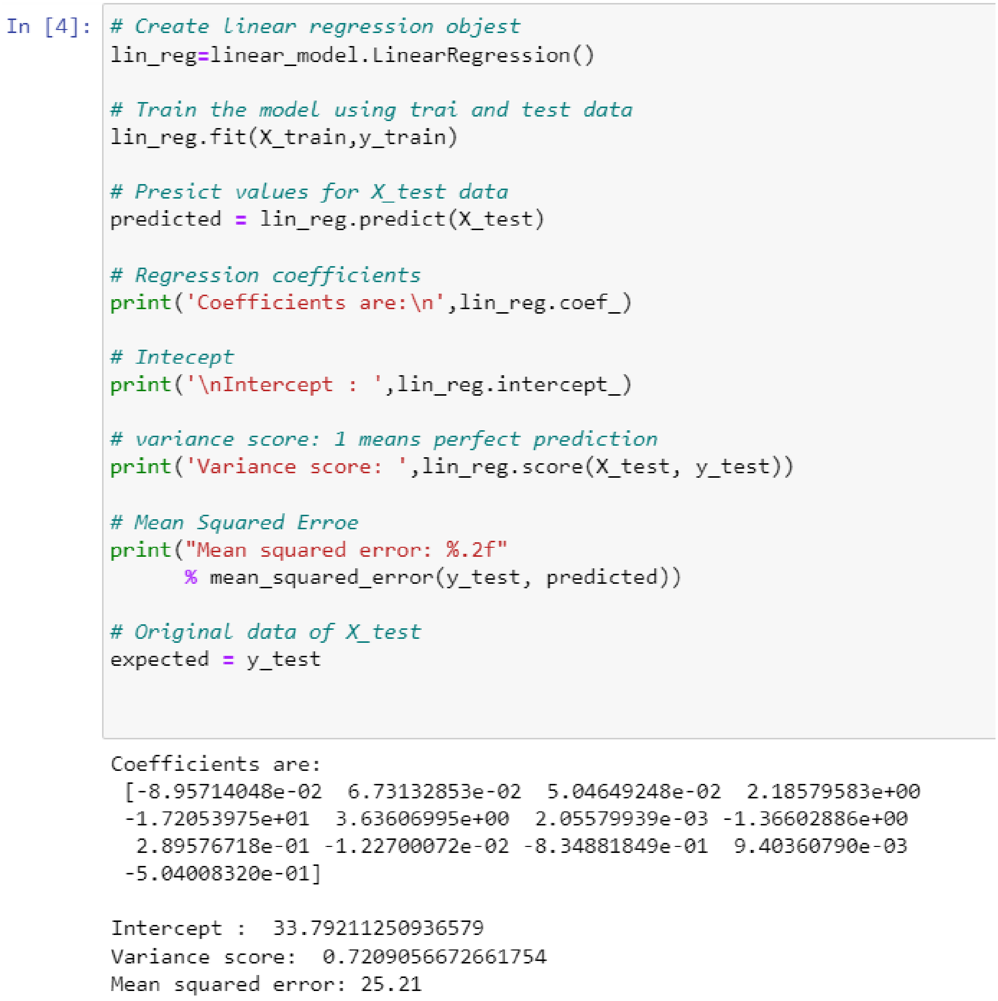
Theory:

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things: (1) does a set of predictor variables do a good job in predicting an outcome (dependent) variable? (2) Which variables in particular are significant predictors of the outcome variable, and in what way do they–indicated by the magnitude and sign of the beta estimates–impact the outcome variable? These regression estimates are used to explain the relationship between one dependent variable and one or more independent variables. The simplest form of the regression equation with one dependent and one independent variable is defined by the formula y = c + b*x, where y = estimated dependent variable score, c = constant, b = regression coefficient, and x = score on the independent variable.

Implementation:

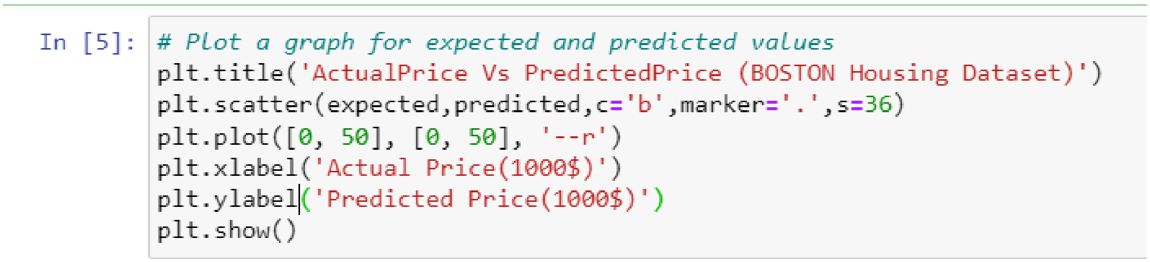


```
In [1]: # Import Dependencies
        import numpy as np
        import matplotlib.pyplot as plt
        from sklearn import datasets,linear_model,metrics
        from sklearn.model_selection import train_test_split
        from sklearn.metrics import mean_squared_error, r2_score

In [2]: # Load the Boston dataset
        boston=datasets.load_boston()

In [3]: # X - feature vectors
        # y - Target values
        X=boston.data
        y=boston.target

        # splitting X and y into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.4,
                                                            random_state=1)
```

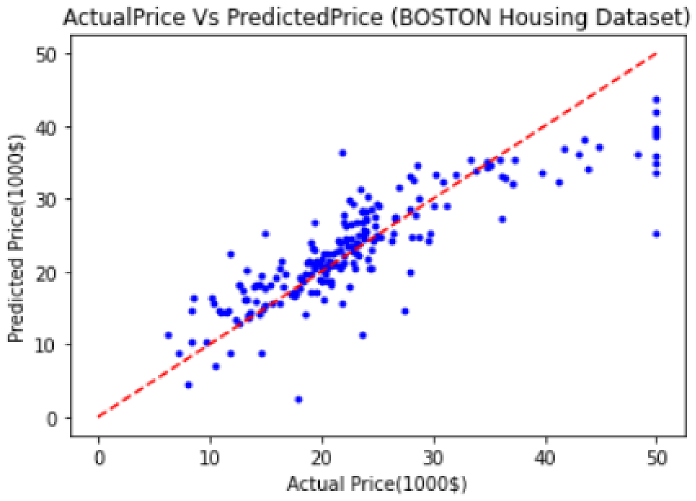


```
In [4]: # Create linear regression objest
        lin_reg=linear_model.LinearRegression()

        # Train the model using trai and test data
        lin_reg.fit(X_train,y_train)

        # Presict values for X_test data
        predicted = lin_reg.predict(X_test)

        # Regression coefficients
        print('Coefficients are:\n',lin_reg.coef_)

        # Intecept
        print('\nIntercept : ',lin_reg.intercept_)

        # variance score: 1 means perfect prediction
        print('Variance score: ',lin_reg.score(X_test, y_test))

        # Mean Squared Erroe
        print("Mean squared error: %.2f"
              % mean_squared_error(y_test, predicted))

        # Original data of X_test
        expected = y_test
```

```
Coefficients are:
 [-8.95714048e-02  6.73132853e-02  5.04649248e-02  2.18579583e+00
 -1.72053975e+01  3.63606995e+00  2.05579939e-03 -1.36602886e+00
  2.89576718e-01 -1.22700072e-02 -8.34881849e-01  9.40360790e-03
 -5.04008320e-01]

Intercept :  33.79211250936579
Variance score:  0.7209056672661754
Mean squared error: 25.21
```



```
In [5]: # Plot a graph for expected and predicted values
        plt.title('ActualPrice Vs PredictedPrice (BOSTON Housing Dataset)')
        plt.scatter(expected,predicted,c='b',marker='.',s=36)
        plt.plot([0, 50], [0, 50], '--r')
        plt.xlabel('Actual Price(1000$)')
        plt.ylabel('Predicted Price(1000$)')
        plt.show()
```



Experiment-2

Aim: WAP to Implement Logistic Regression on Given Data Set.

Theory:

In statistics, the Logistic Regression model is a widely used statistical model which is primarily used for classification purposes. It means that given a set of observations, Logistic Regression algorithm helps us to classify these observations into two or more discrete classes. So, the target variable is discrete in nature.

Logistic Regression algorithm works by implementing a linear equation with independent or explanatory variables to predict a response value. This predicted response value, denoted by z is then converted into a probability value that lie between 0 and 1. We use the sigmoid function in order to map predicted values to probability values. This sigmoid function then maps any real value into a probability value between 0 and 1.

The sigmoid function returns a probability value between 0 and 1. This probability value is then mapped to a discrete class which is either “0” or “1”. In order to map this probability value to a discrete class (pass/fail, yes/no, true/false), we select a threshold value. This threshold value is called Decision boundary. Above this threshold value, we will map the probability values into class 1 and below which we will map values into class 0.

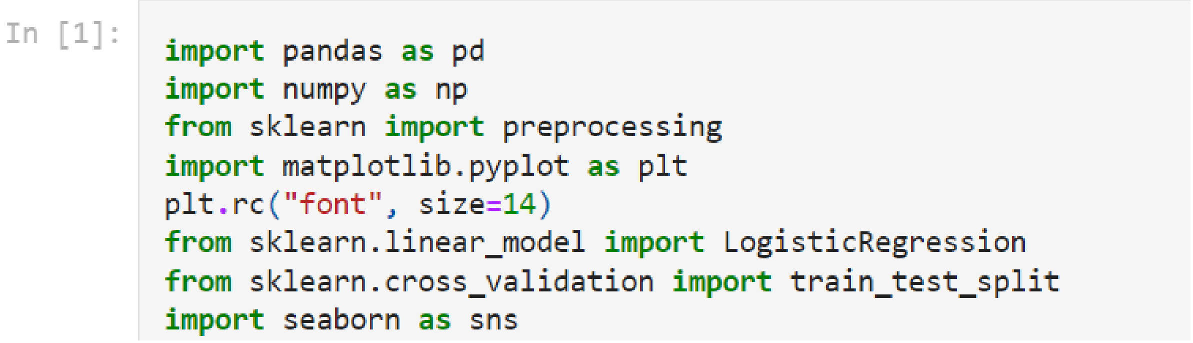
Mathematically, it can be expressed as follows:-

$p \geq 0.5 \Rightarrow class = 1$

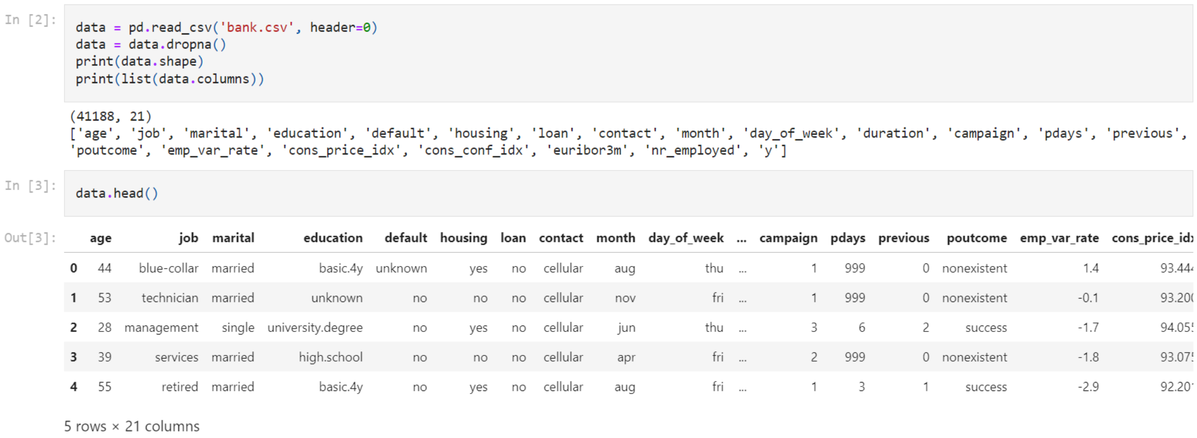
$p < 0.5 \Rightarrow class = 0$

Generally, the decision boundary is set to 0.5. So, if the probability value is 0.8 (> 0.5), we will map this observation to class 1. Similarly, if the probability value is 0.2 (< 0.5), we will map this observation to class 0.

Implementation:



```
In [1]:
import pandas as pd
import numpy as np
from sklearn import preprocessing
import matplotlib.pyplot as plt
plt.rc("font", size=14)
from sklearn.linear_model import LogisticRegression
from sklearn.cross_validation import train_test_split
import seaborn as sns
```

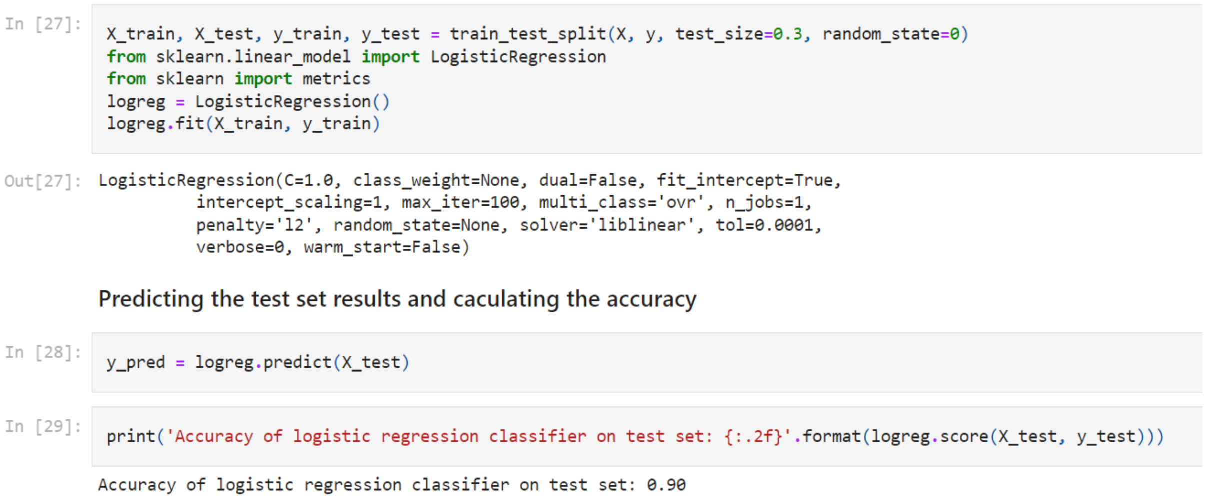


```
In [2]: data = pd.read_csv('bank.csv', header=0)
        data = data.dropna()
        print(data.shape)
        print(list(data.columns))

        (41188, 21)
        ['age', 'job', 'marital', 'education', 'default', 'housing', 'loan', 'contact', 'month', 'day_of_week', 'duration', 'campaign', 'pdays', 'previous',
        'poutcome', 'emp_var_rate', 'cons_price_idx', 'cons_conf_idx', 'euribor3m', 'nr_employed', 'y']

In [3]: data.head()
```

Out[3]:

	age	job	marital	education	default	housing	loan	contact	month	day_of_week	...	campaign	pdays	previous	poutcome	emp_var_rate	cons_price_id
0	44	blue-collar	married	basic.4y	unknown	yes	no	cellular	aug	thu	...	1	999	0	nonexistent	1.4	93.44
1	53	technician	married	unknown	no	no	no	cellular	nov	fri	...	1	999	0	nonexistent	-0.1	93.20
2	28	management	single	university.degree	no	yes	no	cellular	jun	thu	...	3	6	2	success	-1.7	94.05
3	39	services	married	high.school	no	no	no	cellular	apr	fri	...	2	999	0	nonexistent	-1.8	93.07
4	55	retired	married	basic.4y	no	yes	no	cellular	aug	fri	...	1	3	1	success	-2.9	92.20

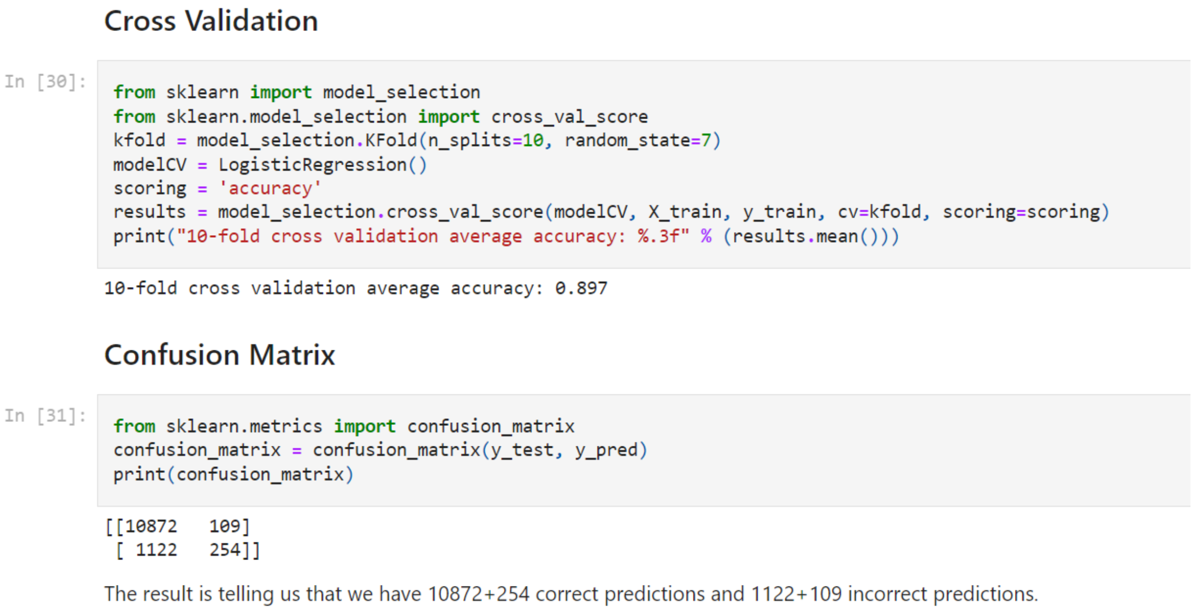
5 rows × 21 columns



```
In [27]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
         from sklearn.linear_model import LogisticRegression
         from sklearn import metrics
         logreg = LogisticRegression()
         logreg.fit(X_train, y_train)

Out[27]: LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
                   penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
                   verbose=0, warm_start=False)
```

Predicting the test set results and caculating the accuracy

```
In [28]: y_pred = logreg.predict(X_test)

In [29]: print('Accuracy of logistic regression classifier on test set: {:.2f}'.format(logreg.score(X_test, y_test)))

         Accuracy of logistic regression classifier on test set: 0.90
```



Cross Validation

```
In [30]: from sklearn import model_selection
         from sklearn.model_selection import cross_val_score
         kfold = model_selection.KFold(n_splits=10, random_state=7)
         modelCV = LogisticRegression()
         scoring = 'accuracy'
         results = model_selection.cross_val_score(modelCV, X_train, y_train, cv=kfold, scoring=scoring)
         print("10-fold cross validation average accuracy: %.3f" % (results.mean()))

         10-fold cross validation average accuracy: 0.897
```

Confusion Matrix

```
In [31]: from sklearn.metrics import confusion_matrix
         confusion_matrix = confusion_matrix(y_test, y_pred)
         print(confusion_matrix)

         [[10872   109]
          [ 1122   254]]
```

The result is telling us that we have 10872+254 correct predictions and 1122+109 incorrect predictions.

Experiment-3

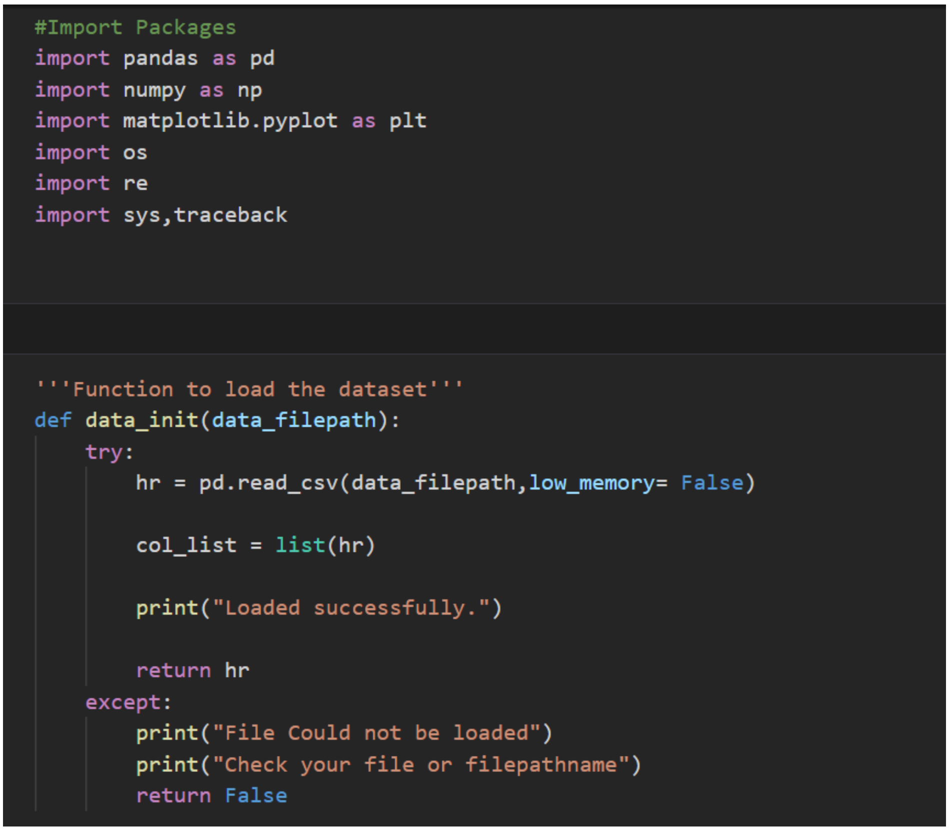
Aim: WAP to predict Employee attrition in a firm and help them plan their manpower

Theory:

Employee attrition prediction system is a really helpful and an important aspect for an organisation. Its is useful in following manner:

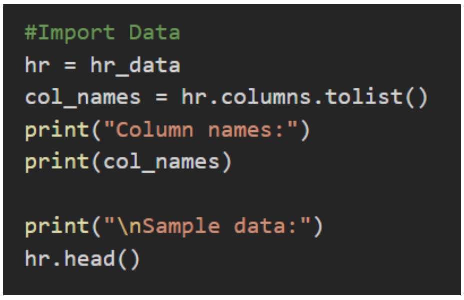
- Managing workforce: If the supervisors or HR came to know about some employees that they will be planning to leave the company then they could get in touch with those employees which can help them to stay back or they can manage the workforce by hiring the new alternative of those employees.
- Smooth pipeline: If all the employees in the current project are working continuously on a project, then the pipeline of that project will be smooth but if suppose one efficient asset of the project(employee) suddenly leave that company then the workflow will be not so smooth
- Hiring Management: If HR of one particular project came to know about the employee who is willing to leave the company, then he/she can manage the number of hiring and they can get the valuable asset

Implementation:

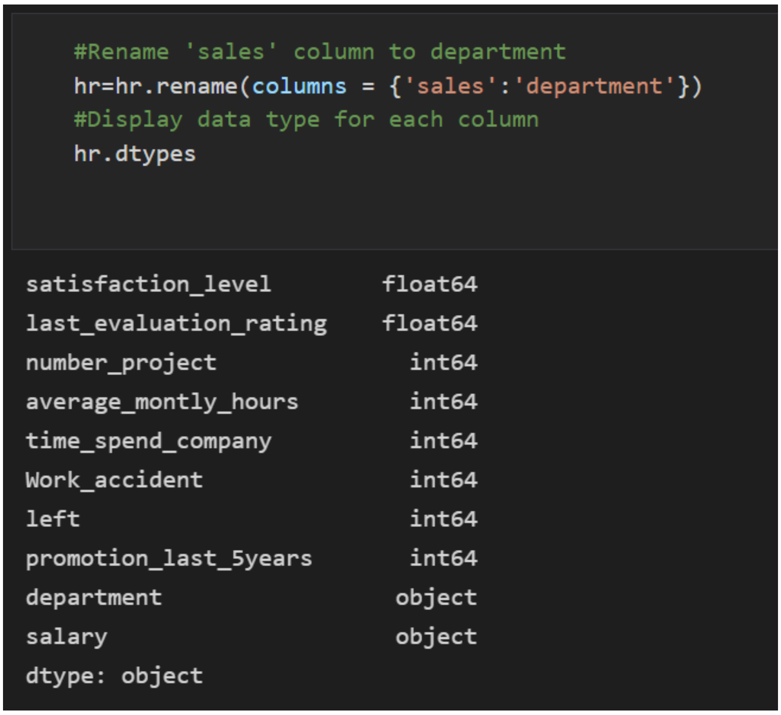


```python
#Import Packages
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import os
import re
import sys,traceback
```

```python
'''Function to load the dataset'''
def data_init(data_filepath):
    try:
        hr = pd.read_csv(data_filepath,low_memory= False)

        col_list = list(hr)

        print("Loaded successfully.")

        return hr
    except:
        print("File Could not be loaded")
        print("Check your file or filepathname")
        return False
```

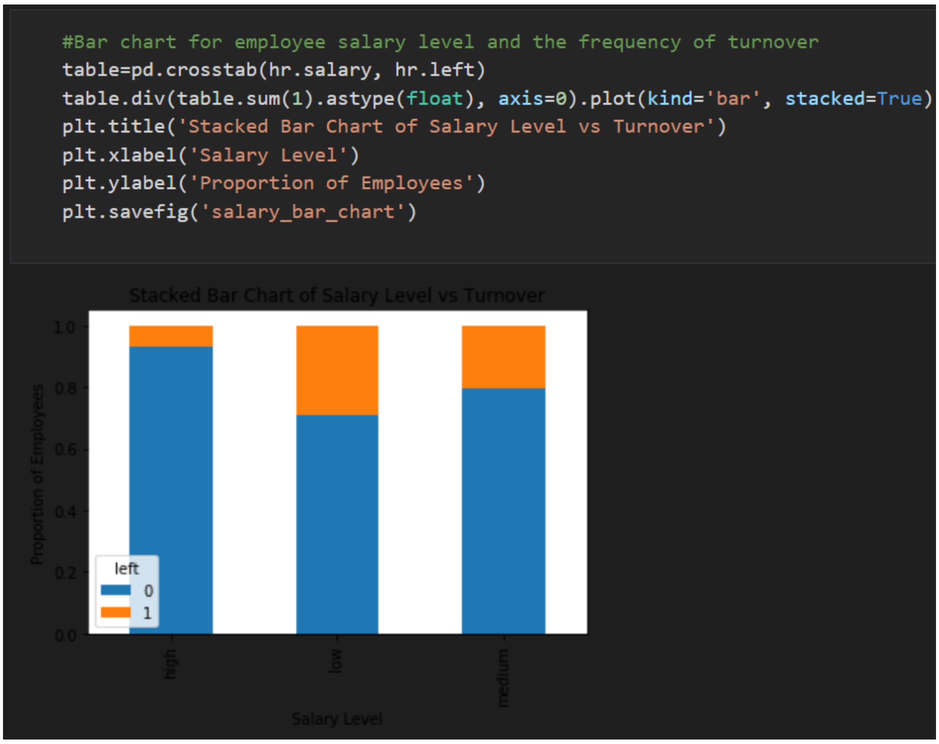


```
#Import Data
hr = hr_data
col_names = hr.columns.tolist()
print("Column names:")
print(col_names)

print("\nSample data:")
hr.head()
```

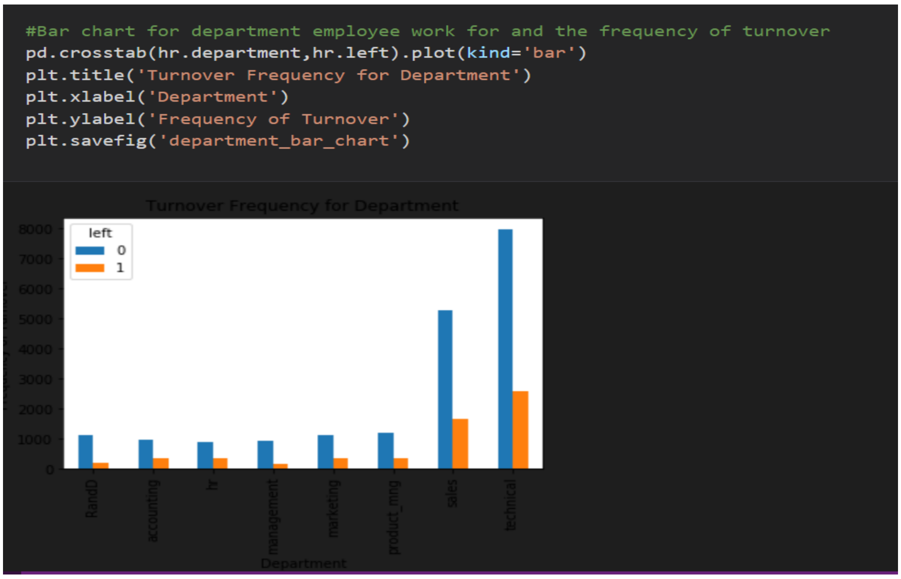


```
#Rename 'sales' column to department
hr=hr.rename(columns = {'sales':'department'})
#Display data type for each column
hr.dtypes
```

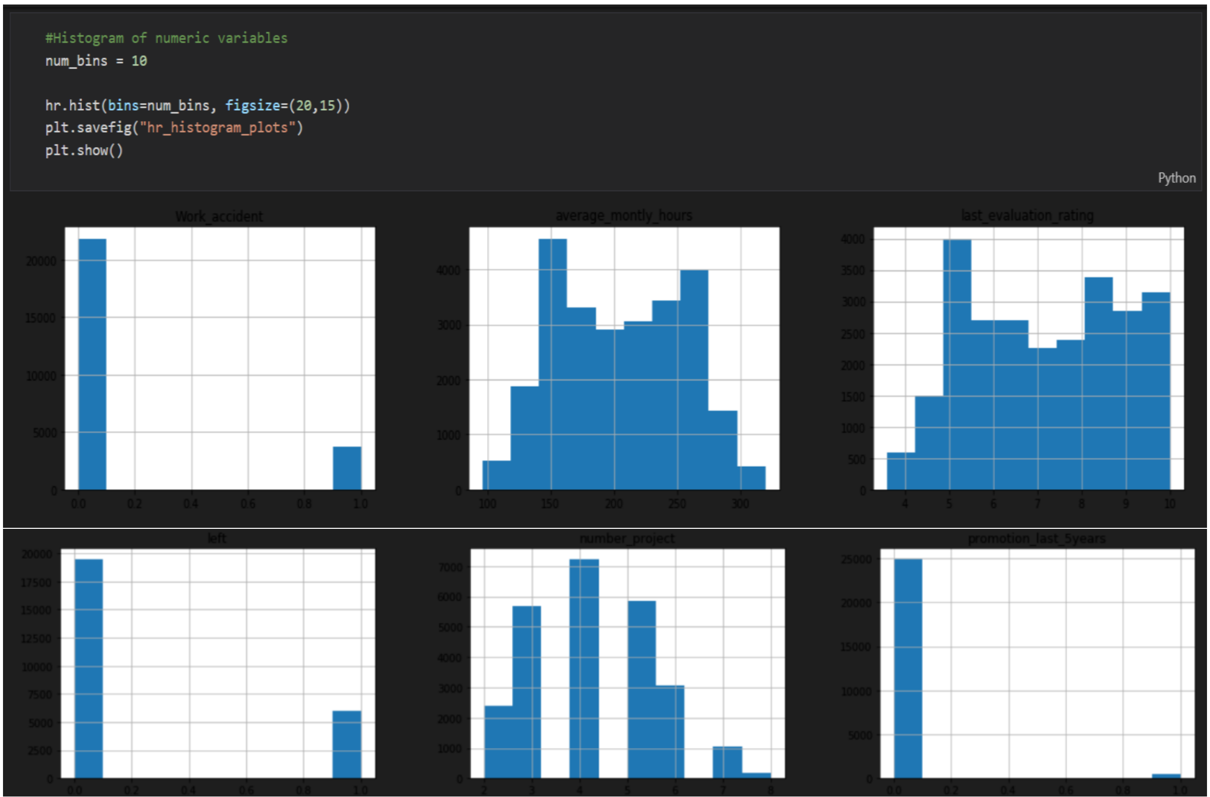
```
satisfaction_level        float64
last_evaluation_rating    float64
number_project              int64
average_montly_hours        int64
time_spend_company          int64
Work_accident               int64
left                        int64
promotion_last_5years       int64
department                 object
salary                     object
dtype: object
```



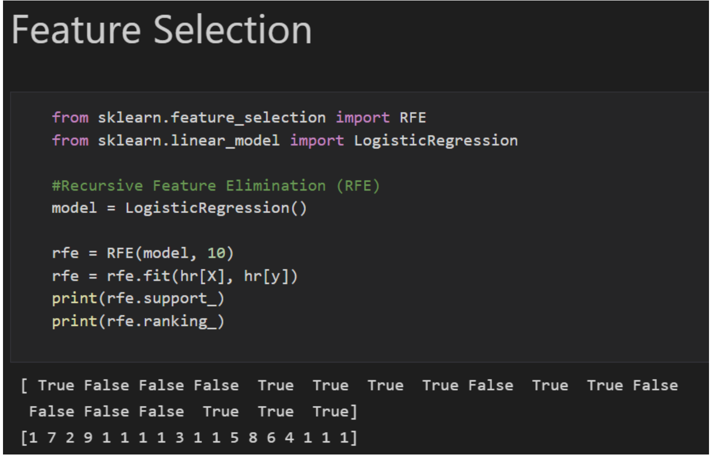
```
#Bar chart for employee salary level and the frequency of turnover
table=pd.crosstab(hr.salary, hr.left)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Salary Level vs Turnover')
plt.xlabel('Salary Level')
plt.ylabel('Proportion of Employees')
plt.savefig('salary_bar_chart')
```



```
#Bar chart for department employee work for and the frequency of turnover
pd.crosstab(hr.department,hr.left).plot(kind='bar')
plt.title('Turnover Frequency for Department')
plt.xlabel('Department')
plt.ylabel('Frequency of Turnover')
plt.savefig('department_bar_chart')
```



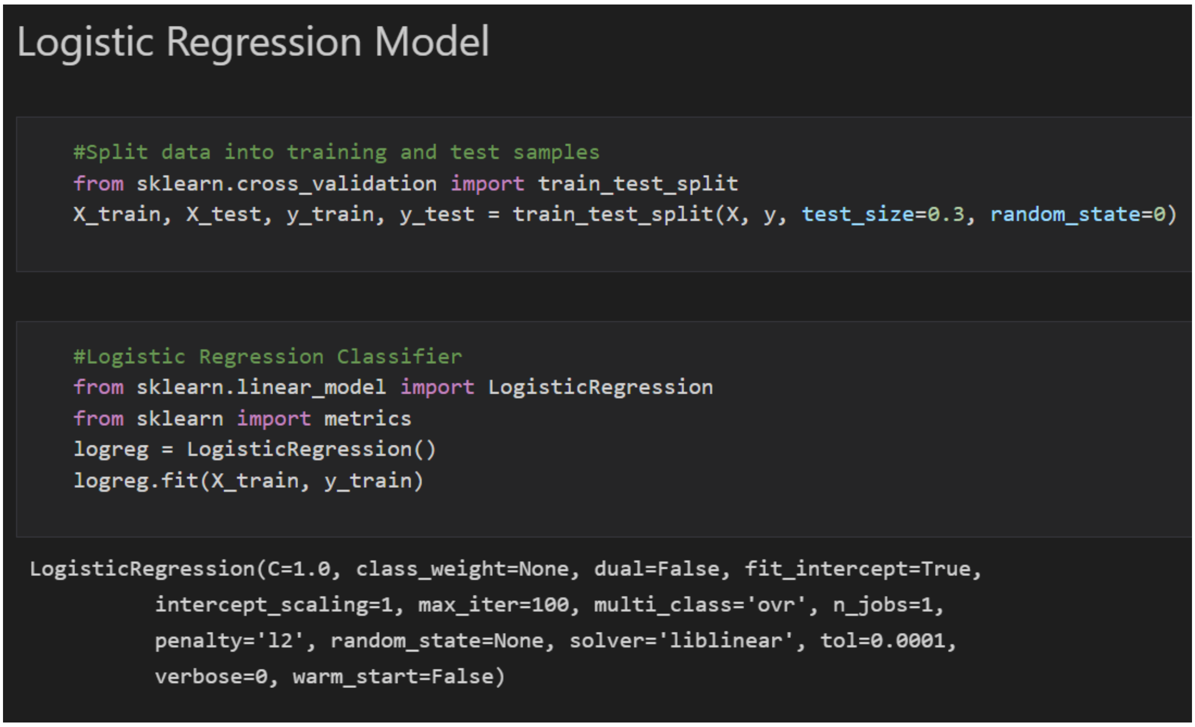
```
#Histogram of numeric variables
num_bins = 10

hr.hist(bins=num_bins, figsize=(20,15))
plt.savefig("hr_histogram_plots")
plt.show()
```



Feature Selection

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

#Recursive Feature Elimination (RFE)
model = LogisticRegression()

rfe = RFE(model, 10)
rfe = rfe.fit(hr[X], hr[y])
print(rfe.support_)
print(rfe.ranking_)
```

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

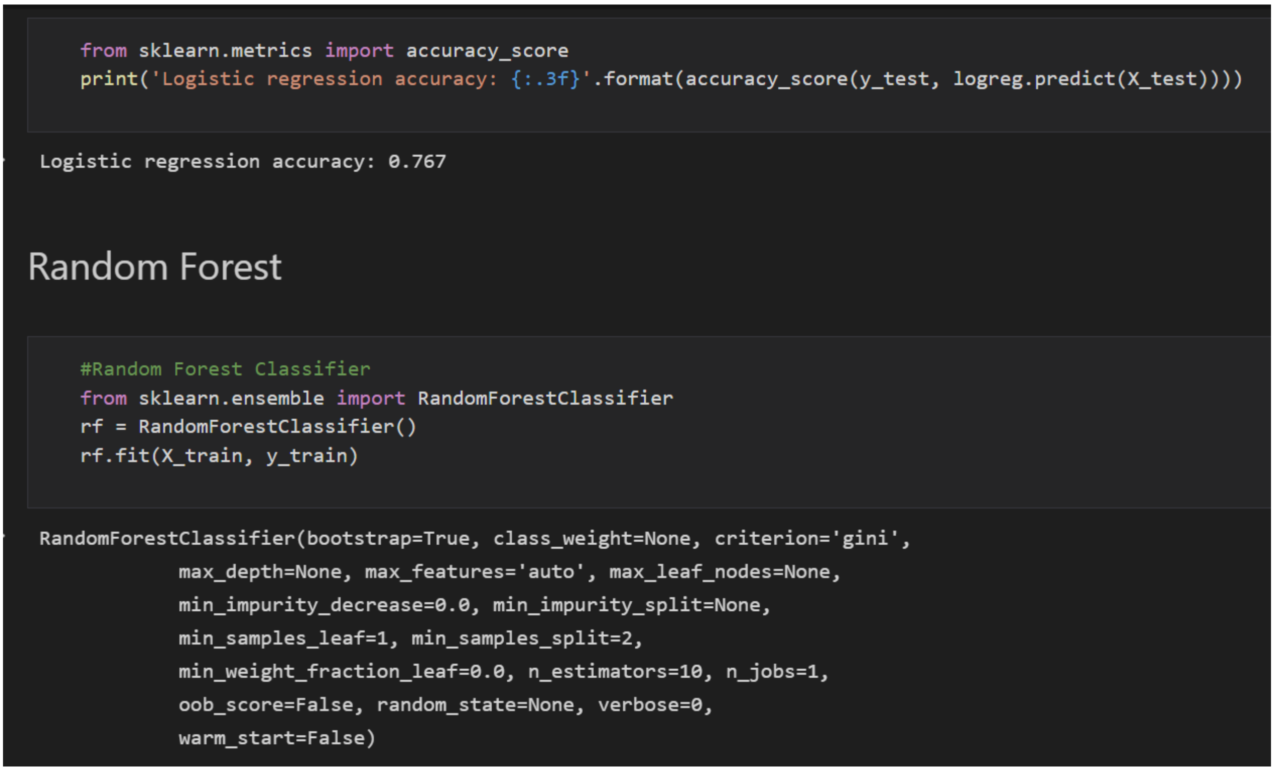


Logistic Regression Model

```
#Split data into training and test samples
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
```

```
#Logistic Regression Classifier
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```



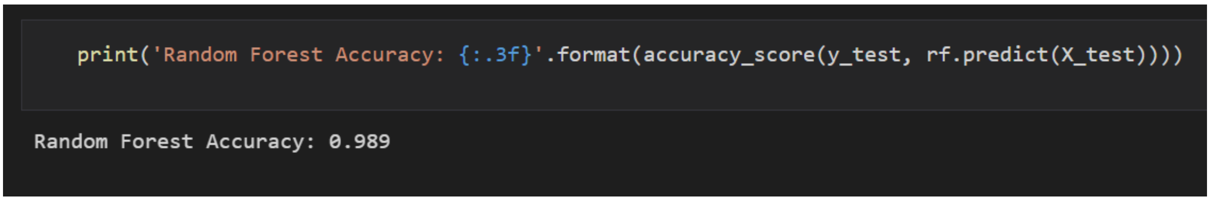
```
from sklearn.metrics import accuracy_score
print('Logistic regression accuracy: {:.3f}'.format(accuracy_score(y_test, logreg.predict(X_test))))
```

```
Logistic regression accuracy: 0.767
```

Random Forest

```
#Random Forest Classifier
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```



```
print('Random Forest Accuracy: {:.3f}'.format(accuracy_score(y_test, rf.predict(X_test))))
```

```
Random Forest Accuracy: 0.989
```

Experiment-4

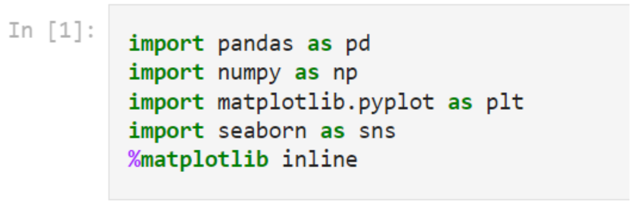
Aim: WAP to Implement Decision Tree Classification Algorithm.

Theory:

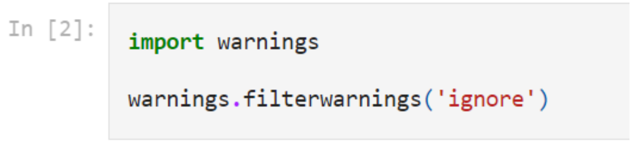
Employee attrition prediction system is a really helpful and an important aspect for an Decision Tree is the most powerful and popular tool for classification and prediction. A Decision tree is a flowchart-like tree structure, where each internal node denotes a test on an attribute, each branch represents an outcome of the test, and each leaf node (terminal node) holds a class label. Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute as shown in the above figure. This process is then repeated for the subtree rooted at the new node.

The decision tree in above figure classifies a particular morning according to whether it is suitable for playing tennis and returns the classification associated with the particular leaf.(in this case Yes or No).

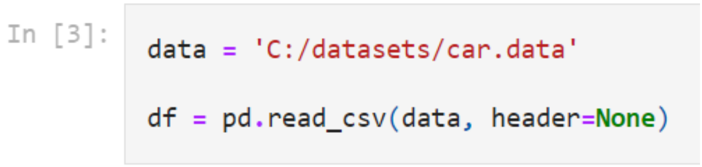
Implementation:



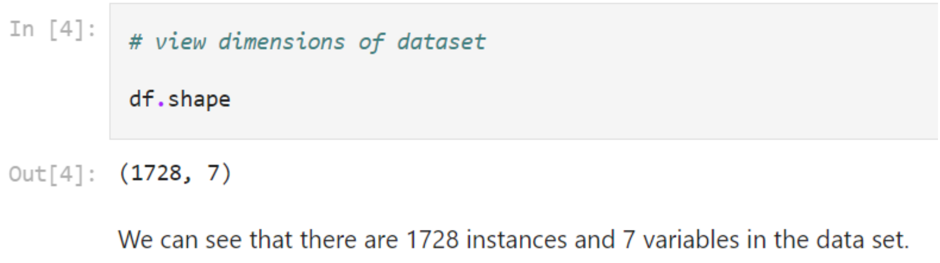
```
In [1]:
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```



```
In [2]:
import warnings

warnings.filterwarnings('ignore')
```



```
In [3]:
data = 'C:/datasets/car.data'

df = pd.read_csv(data, header=None)
```



```
In [4]:
# view dimensions of dataset

df.shape

Out[4]: (1728, 7)
```

We can see that there are 1728 instances and 7 variables in the data set.

```
df.head()
```

Out[7]:

	buying	maint	doors	persons	lug_boot	safety	class
0	vhigh	vhigh	2	2	small	low	unacc
1	vhigh	vhigh	2	2	small	med	unacc
2	vhigh	vhigh	2	2	small	high	unacc
3	vhigh	vhigh	2	2	med	low	unacc
4	vhigh	vhigh	2	2	med	med	unacc

We can see that the column names are renamed. Now, the columns have meaningful names.

View summary of dataset

In [8]:

```
df.info()
```

```
RangeIndex: 1728 entries, 0 to 1727
Data columns (total 7 columns):
buying      1728 non-null object
maint       1728 non-null object
doors       1728 non-null object
persons     1728 non-null object
lug_boot    1728 non-null object
safety      1728 non-null object
class       1728 non-null object
dtypes: object(7)
memory usage: 94.6+ KB
```

In [10]:

```
df['class'].value_counts()
```

Out[10]:

```
unacc    1210
acc       384
good       69
vgood      65
Name: class, dtype: int64
```

The `class` target variable is ordinal in nature.

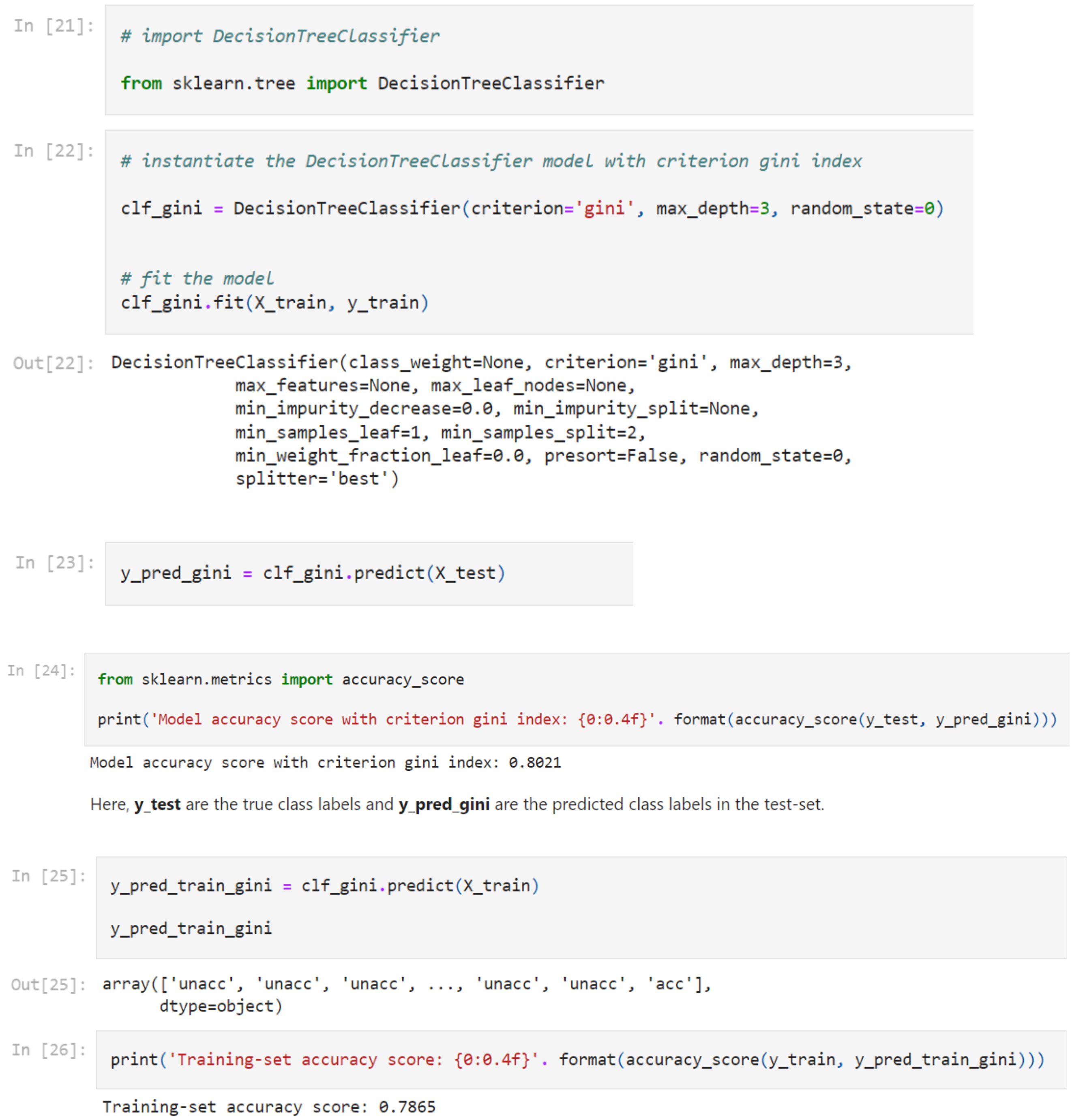
Missing values in variables

In [11]:

```
# check missing values in variables

df.isnull().sum()
```

Out[11]:

```
buying      0
maint       0
doors       0
persons     0
lug_boot    0
safety      0
class       0
dtype: int64
```



In [21]:
```
# import DecisionTreeClassifier

from sklearn.tree import DecisionTreeClassifier
```

In [22]:
```
# instantiate the DecisionTreeClassifier model with criterion gini index

clf_gini = DecisionTreeClassifier(criterion='gini', max_depth=3, random_state=0)


# fit the model
clf_gini.fit(X_train, y_train)
```

Out[22]:
```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=3,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=0,
            splitter='best')
```

In [23]:
```
y_pred_gini = clf_gini.predict(X_test)
```

In [24]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score with criterion gini index: {0:0.4f}'. format(accuracy_score(y_test, y_pred_gini)))
```

```
Model accuracy score with criterion gini index: 0.8021
```

Here, **y_test** are the true class labels and **y_pred_gini** are the predicted class labels in the test-set.

In [25]:
```
y_pred_train_gini = clf_gini.predict(X_train)

y_pred_train_gini
```

Out[25]:
```
array(['unacc', 'unacc', 'unacc', ..., 'unacc', 'unacc', 'acc'],
      dtype=object)
```

In [26]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train_gini)))
```

```
Training-set accuracy score: 0.7865
```

Experiment-5

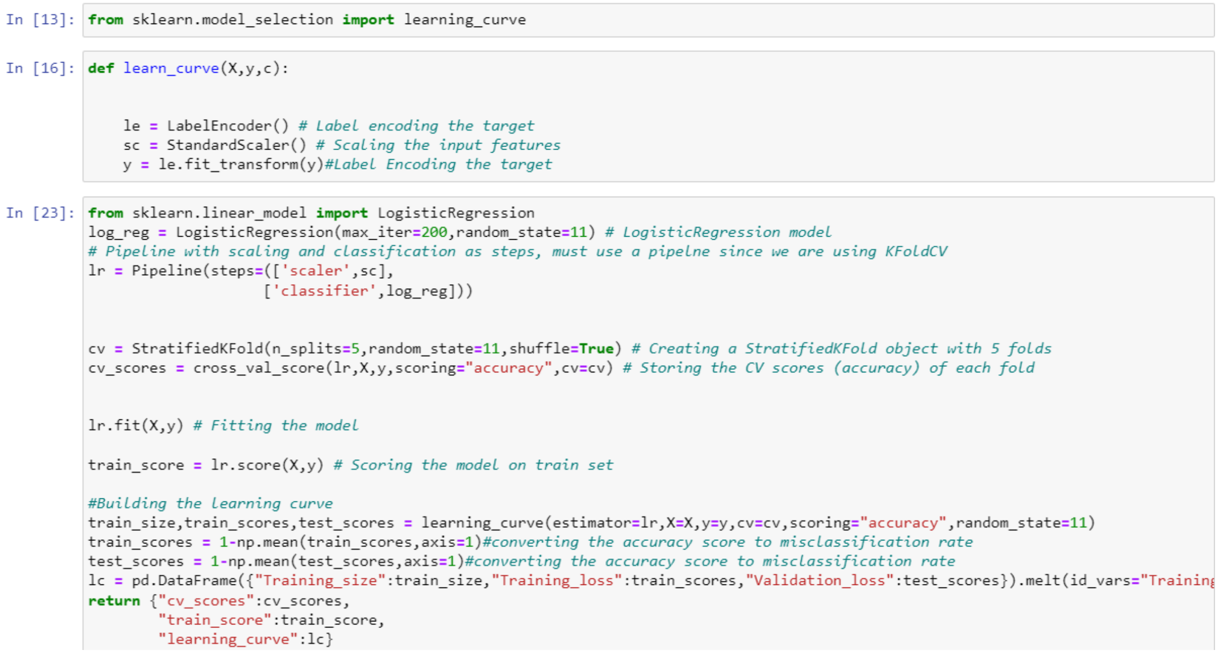
Aim: WAP To implement and plot graphs for overfitting and Underfitting of the curve.

Theory:

Underfitting: A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data, i.e., it only performs well on training data but performs poorly on testing data. (It's just like trying to fit undersized pants!) Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have fewer data to build an accurate model and when we try to build a linear model with fewer non-linear data. In such cases, the rules of the machine learning model are too easy and flexible to be applied to such minimal data and therefore the model will probably make a lot of wrong predictions. Underfitting can be avoided by using more data and reducing the features by feature selection.

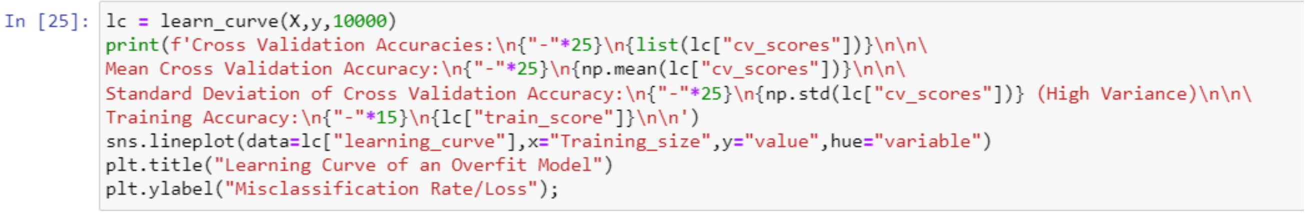
Overfitting: A statistical model is said to be overfitted when the model does not make accurate predictions on testing data. When a model gets trained with so much data, it starts learning from the noise and inaccurate data entries in our data set. And when testing with test data results in High variance. Then the model does not categorize the data correctly, because of too many details and noise. The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore, they can really build unrealistic models. A solution to avoid overfitting is using a linear algorithm if we have linear data or using the parameters like the maximal depth if we are using decision trees.

Implementation:

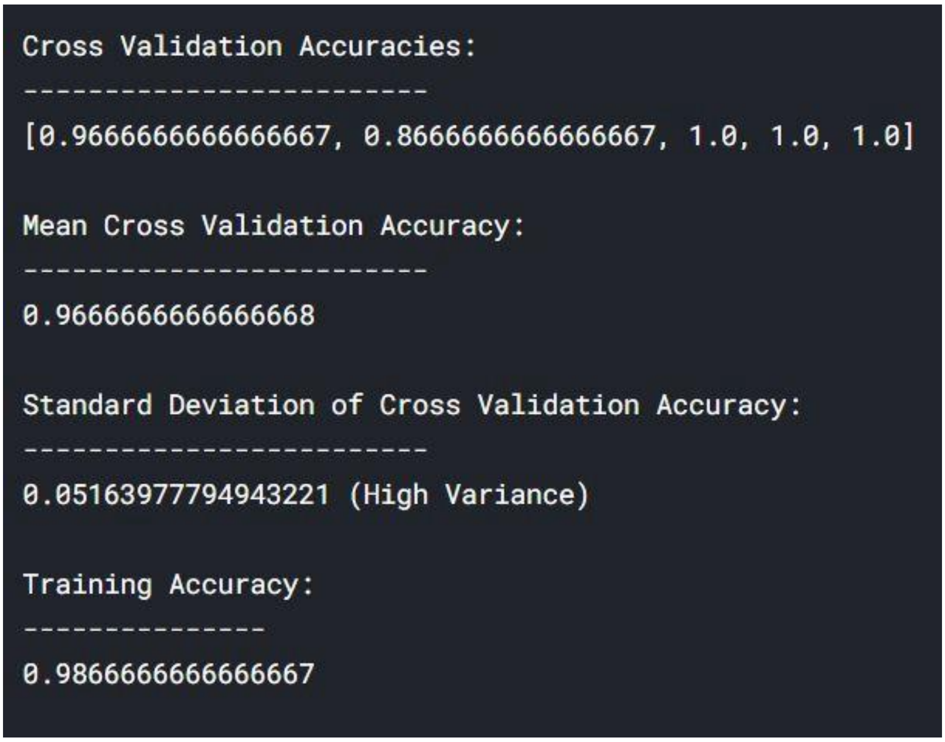


```
In [13]: from sklearn.model_selection import learning_curve

In [16]: def learn_curve(X,y,c):


             le = LabelEncoder() # Label encoding the target
             sc = StandardScaler() # Scaling the input features
             y = le.fit_transform(y)#Label Encoding the target

In [23]: from sklearn.linear_model import LogisticRegression
         log_reg = LogisticRegression(max_iter=200,random_state=11) # LogisticRegression model
         # Pipeline with scaling and classification as steps, must use a pipelne since we are using KFoldCV
         lr = Pipeline(steps=(['scaler',sc],
                             ['classifier',log_reg]))


         cv = StratifiedKFold(n_splits=5,random_state=11,shuffle=True) # Creating a StratifiedKFold object with 5 folds
         cv_scores = cross_val_score(lr,X,y,scoring="accuracy",cv=cv) # Storing the CV scores (accuracy) of each fold


         lr.fit(X,y) # Fitting the model

         train_score = lr.score(X,y) # Scoring the model on train set

         #Building the learning curve
         train_size,train_scores,test_scores = learning_curve(estimator=lr,X=X,y=y,cv=cv,scoring="accuracy",random_state=11)
         train_scores = 1-np.mean(train_scores,axis=1)#converting the accuracy score to misclassification rate
         test_scores = 1-np.mean(test_scores,axis=1)#converting the accuracy score to misclassification rate
         lc = pd.DataFrame({"Training_size":train_size,"Training_loss":train_scores,"Validation_loss":test_scores}).melt(id_vars="Training
         return {"cv_scores":cv_scores,
                 "train_score":train_score,
                 "learning_curve":lc}
```

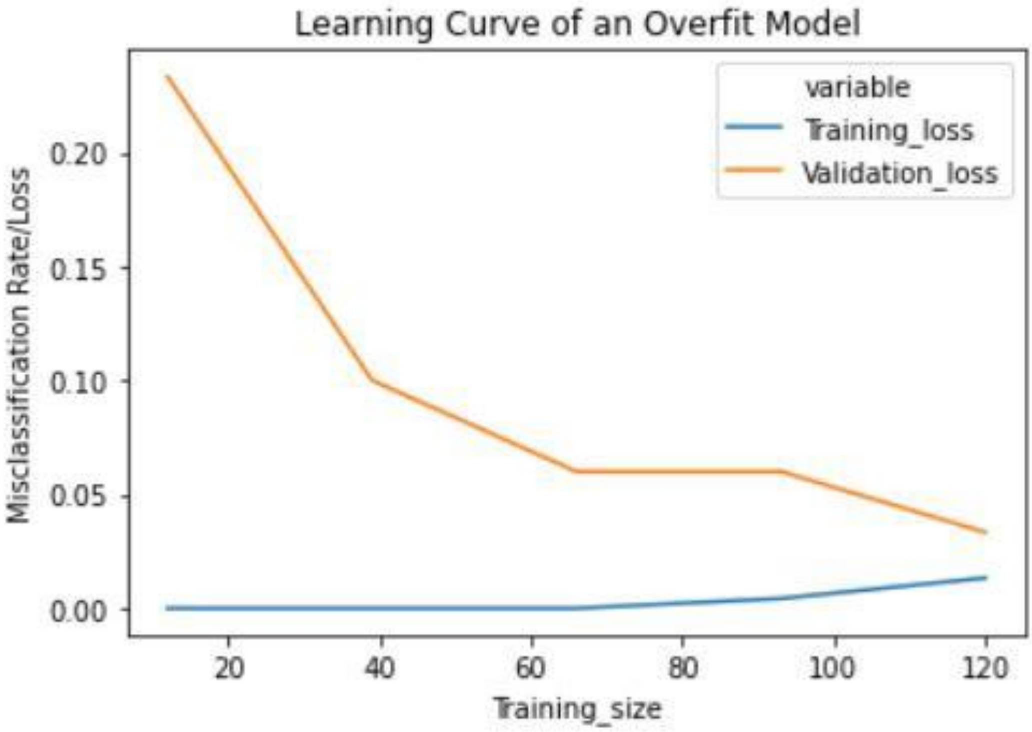
Overfit Model:



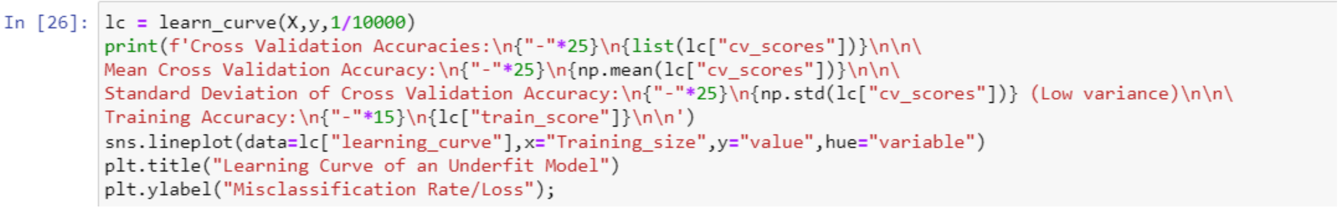
```
In [25]: lc = learn_curve(X,y,10000)
         print(f'Cross Validation Accuracies:\n{"-"*25}\n{list(lc["cv_scores"])}\n\n\
         Mean Cross Validation Accuracy:\n{"-"*25}\n{np.mean(lc["cv_scores"])}\n\n\
         Standard Deviation of Cross Validation Accuracy:\n{"-"*25}\n{np.std(lc["cv_scores"])} (High Variance)\n\n\
         Training Accuracy:\n{"-"*15}\n{lc["train_score"]}\n\n')
         sns.lineplot(data=lc["learning_curve"],x="Training_size",y="value",hue="variable")
         plt.title("Learning Curve of an Overfit Model")
         plt.ylabel("Misclassification Rate/Loss");
```



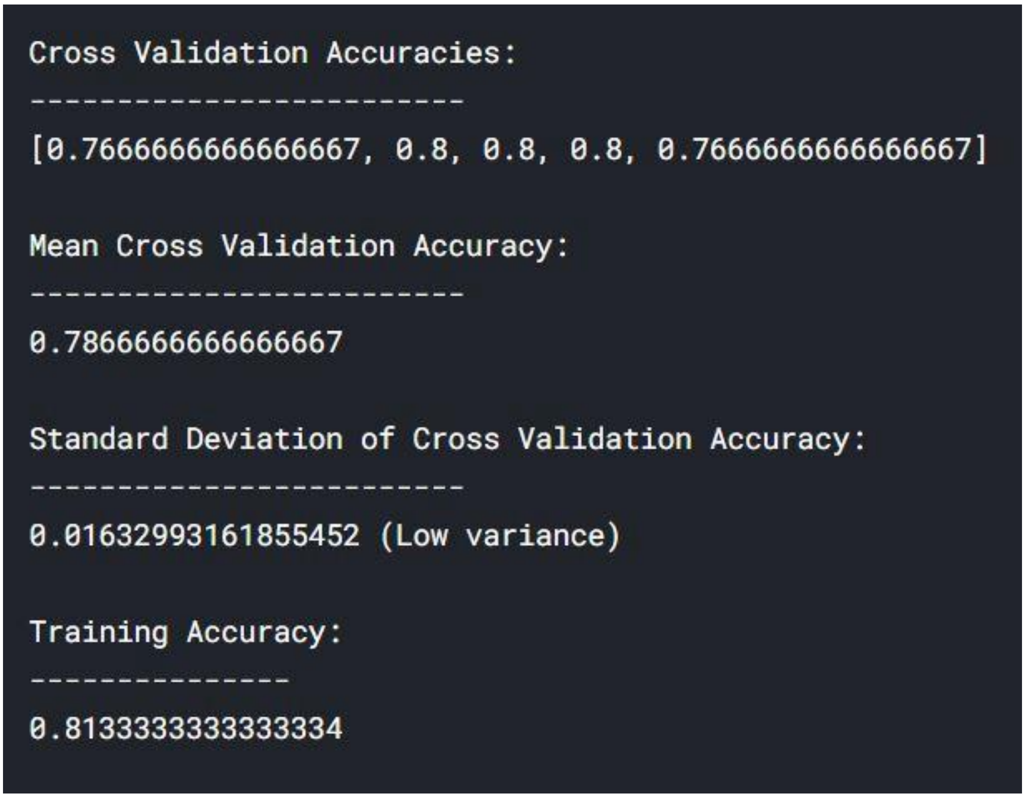
```
Cross Validation Accuracies:
-------------------------
[0.9666666666666667, 0.8666666666666667, 1.0, 1.0, 1.0]

Mean Cross Validation Accuracy:
-------------------------
0.9666666666666668

Standard Deviation of Cross Validation Accuracy:
-------------------------
0.05163977794943221 (High Variance)

Training Accuracy:
---------------
0.9866666666666667
```



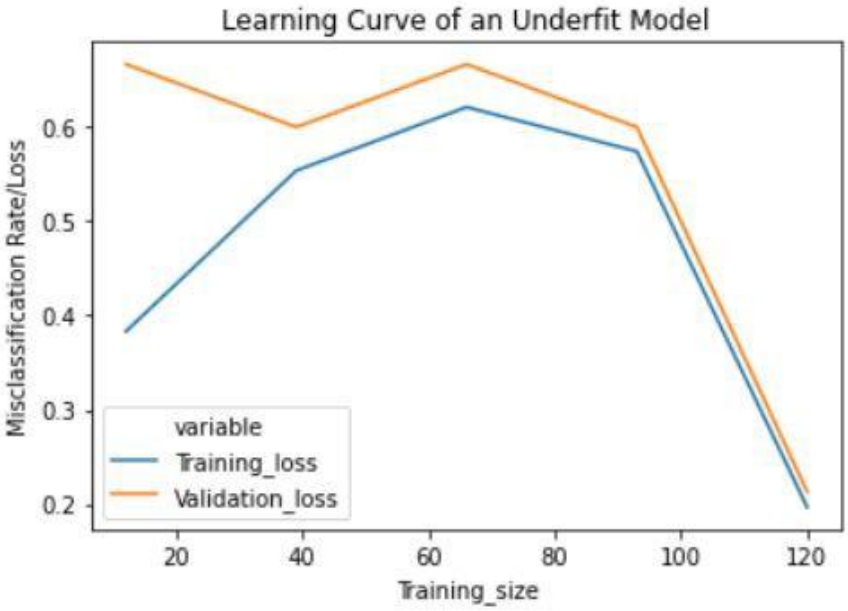
Underfit Model:



```
In [26]: lc = learn_curve(X,y,1/10000)
         print(f'Cross Validation Accuracies:\n{"-"*25}\n{list(lc["cv_scores"])}\n\n\
         Mean Cross Validation Accuracy:\n{"-"*25}\n{np.mean(lc["cv_scores"])}\n\n\
         Standard Deviation of Cross Validation Accuracy:\n{"-"*25}\n{np.std(lc["cv_scores"])} (Low variance)\n\n\
         Training Accuracy:\n{"-"*15}\n{lc["train_score"]}\n\n')
         sns.lineplot(data=lc["learning_curve"],x="Training_size",y="value",hue="variable")
         plt.title("Learning Curve of an Underfit Model")
         plt.ylabel("Misclassification Rate/Loss");
```



```
Cross Validation Accuracies:
-------------------------
[0.7666666666666667, 0.8, 0.8, 0.8, 0.7666666666666667]

Mean Cross Validation Accuracy:
-------------------------
0.7866666666666667

Standard Deviation of Cross Validation Accuracy:
-------------------------
0.01632993161855452 (Low variance)

Training Accuracy:
---------------
0.8133333333333334
```



Experiment-6

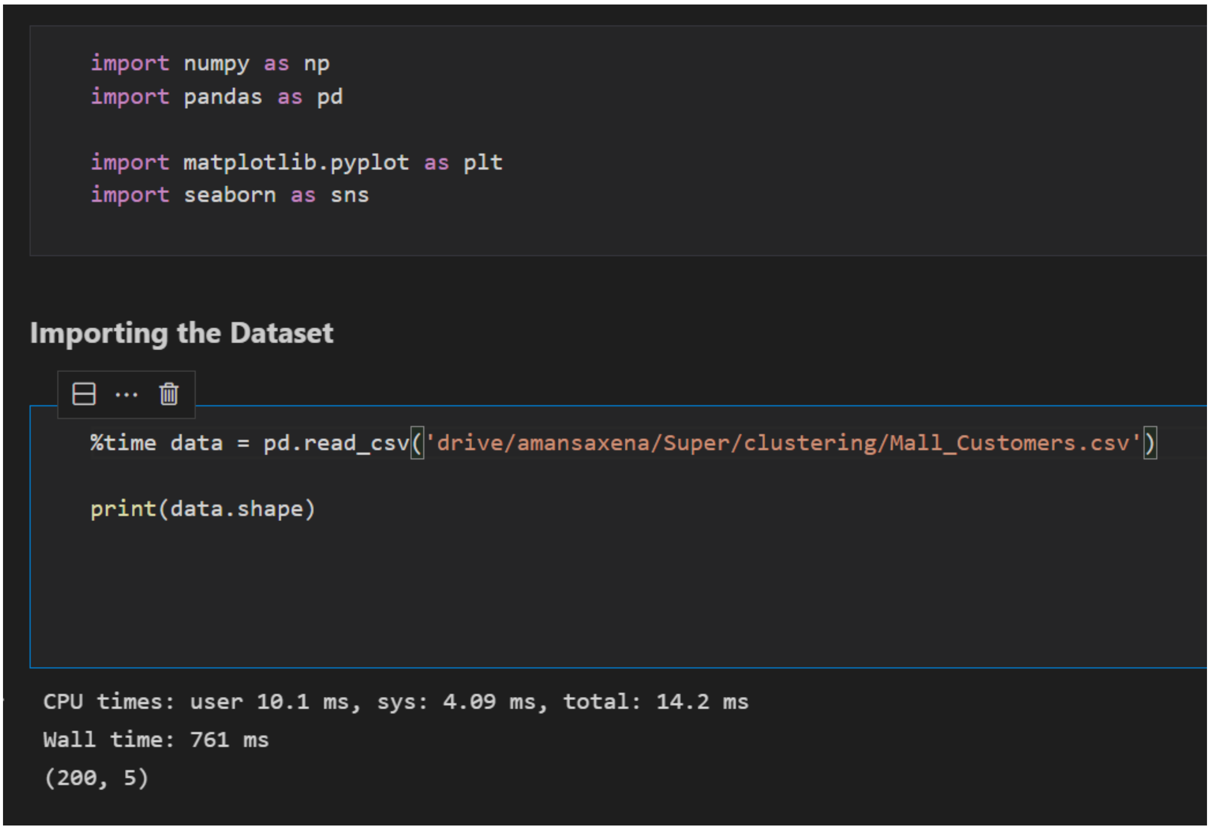
Aim: WAP To Create Customer Clusters using different market strategies on a dataset.

Theory:

Customer clustering or segmentation is the process of dividing an organisation's customers into groups or 'clusters' that reflect similarity amongst customers in that group. The goal of such a process of clustering is to decide how to relate to customers in each of these customer cluster to maximise the benefits those customers bring to the business and make for a more engaging transaction with them. Compared to rule-based segmentation, AI powered customer clustering finds closer affinity among customers within a cluster. In the context of customer clustering, cluster analysis is the use of mathematical modelling to achieve such goals. These homogenous groups of customers are known as 'customer archetypes' or 'personas'.

A common cluster analysis process is an algorithm known as 'k-means cluster analysis'. Since the process is not based on predetermined rules, the data reveals customer archetypes and trends that exist inherently within customer populations. Customer clustering and analysis can thus be used to effectively target customers with offers and incentives personalised to their needs and wants. It is an important tool for any business to operate in the modern day.

Implementation:

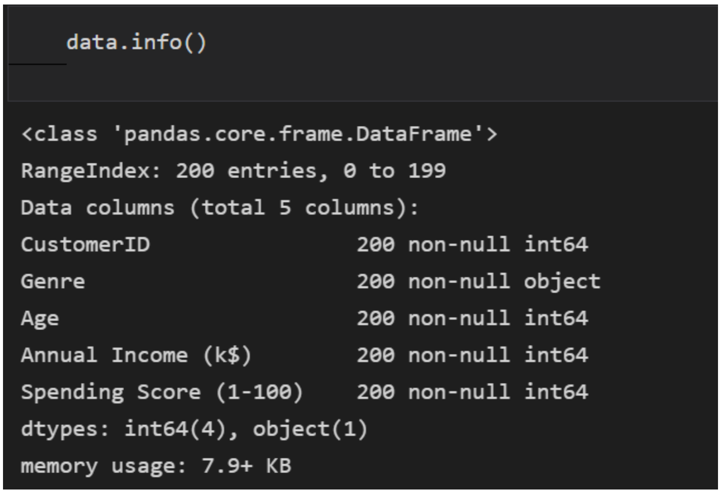


```
import numpy as np
import pandas as pd

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

Importing the Dataset

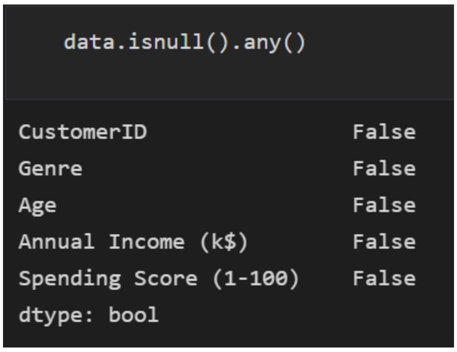
```
%time data = pd.read_csv('drive/amansaxena/Super/clustering/Mall_Customers.csv')

print(data.shape)
```

```
CPU times: user 10.1 ms, sys: 4.09 ms, total: 14.2 ms
Wall time: 761 ms
(200, 5)
```



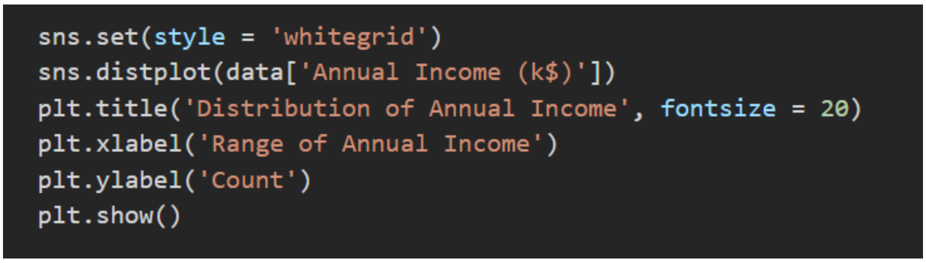
```
data.info()
```

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

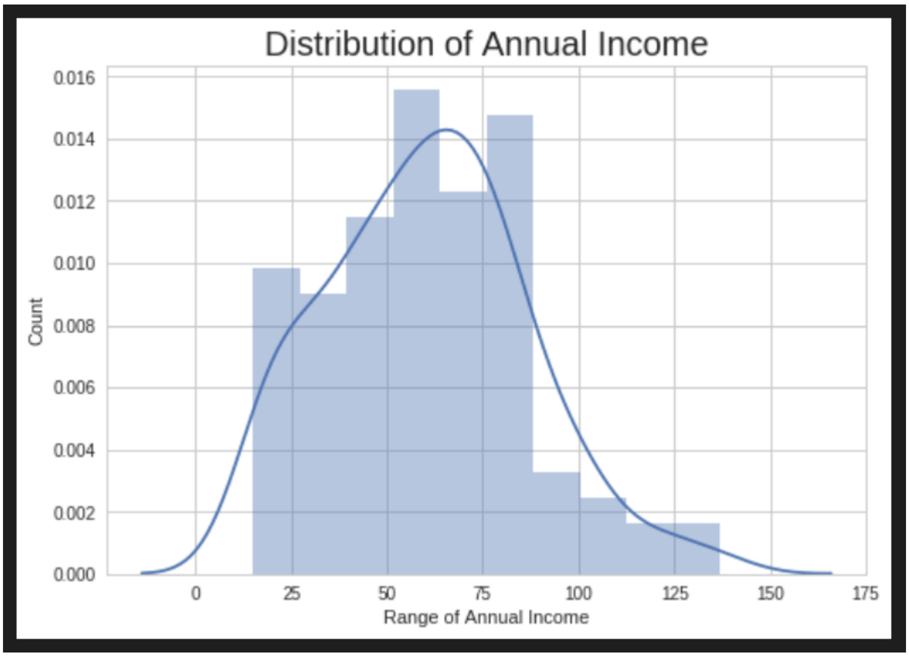


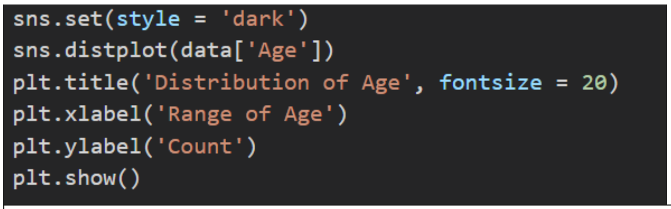
```
data.isnull().any()
```

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

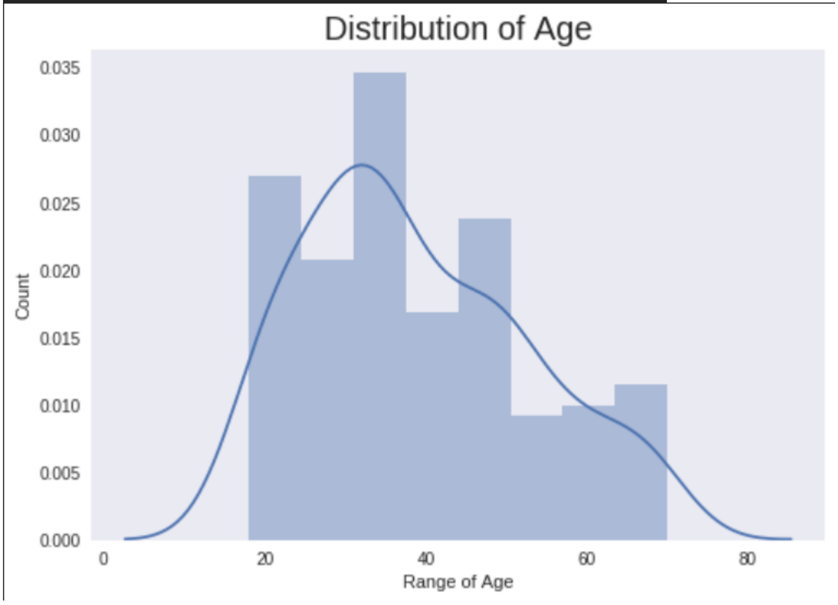


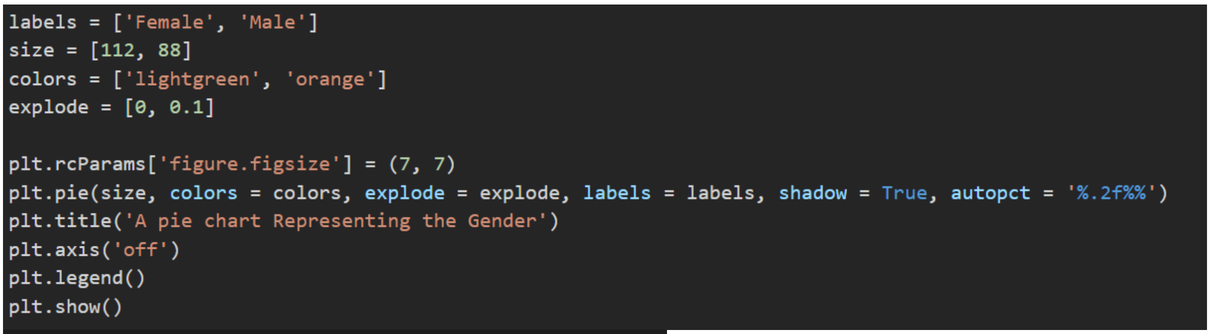
```
sns.set(style = 'whitegrid')
sns.distplot(data['Annual Income (k$)'])
plt.title('Distribution of Annual Income', fontsize = 20)
plt.xlabel('Range of Annual Income')
plt.ylabel('Count')
plt.show()
```



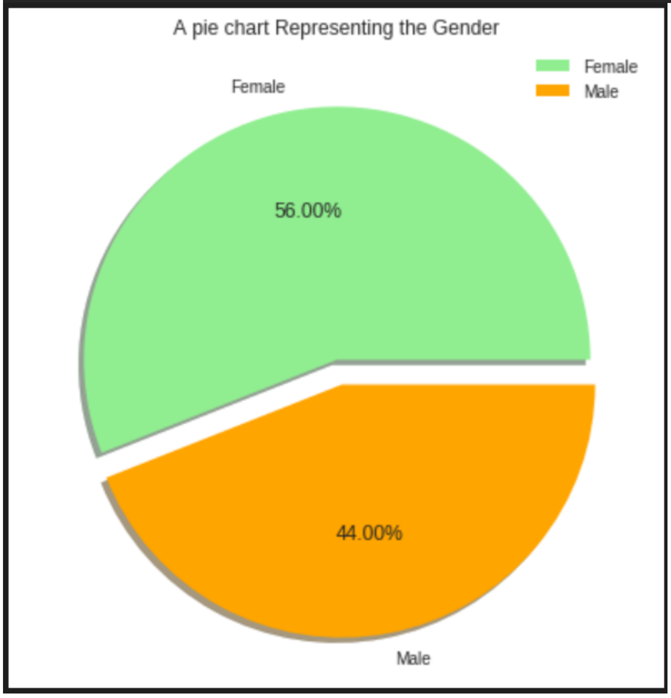


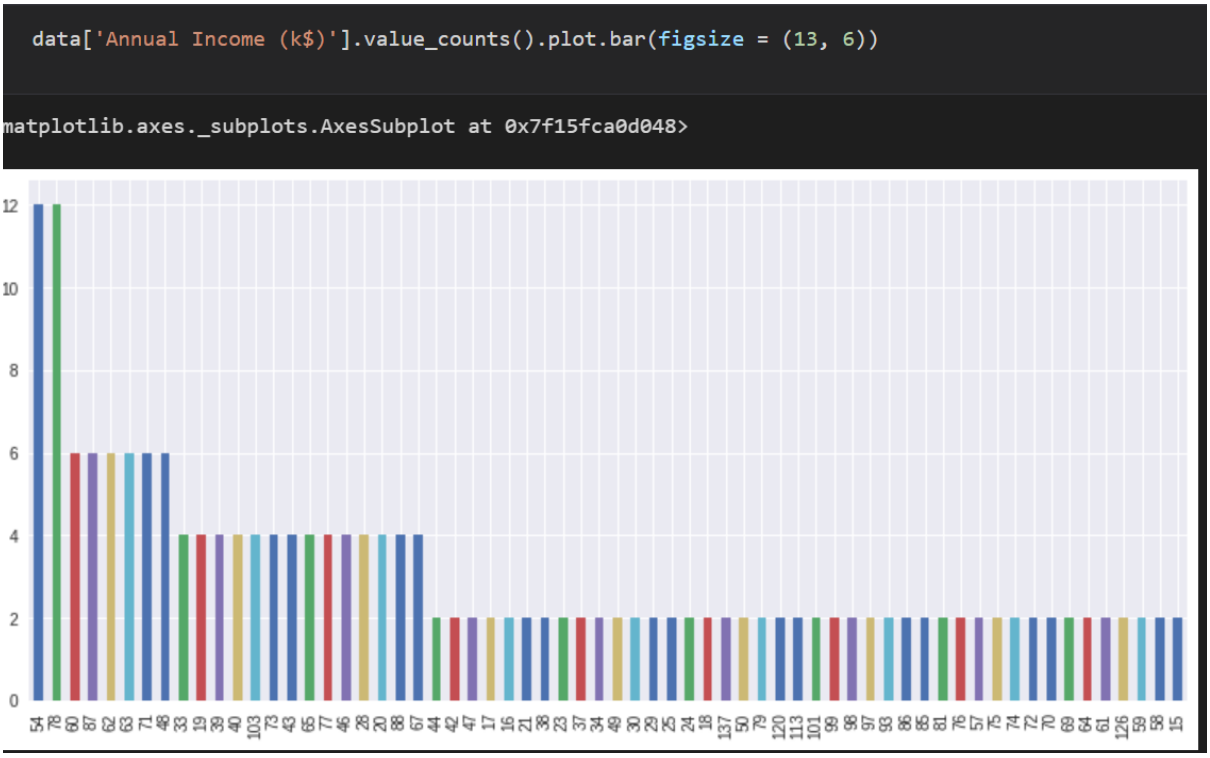
```
sns.set(style = 'dark')
sns.distplot(data['Age'])
plt.title('Distribution of Age', fontsize = 20)
plt.xlabel('Range of Age')
plt.ylabel('Count')
plt.show()
```





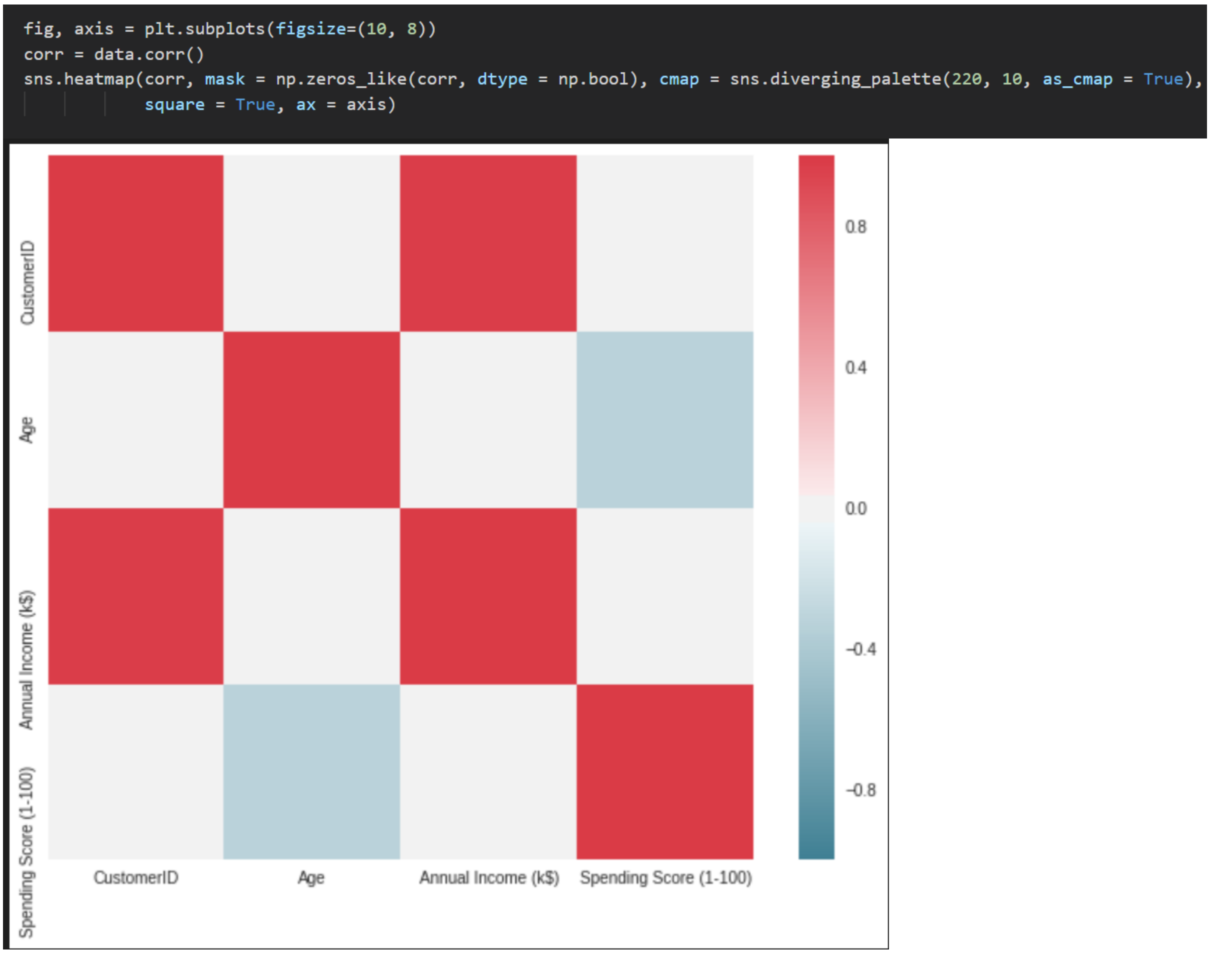
```
labels = ['Female', 'Male']
size = [112, 88]
colors = ['lightgreen', 'orange']
explode = [0, 0.1]

plt.rcParams['figure.figsize'] = (7, 7)
plt.pie(size, colors = colors, explode = explode, labels = labels, shadow = True, autopct = '%.2f%%')
plt.title('A pie chart Representing the Gender')
plt.axis('off')
plt.legend()
plt.show()
```



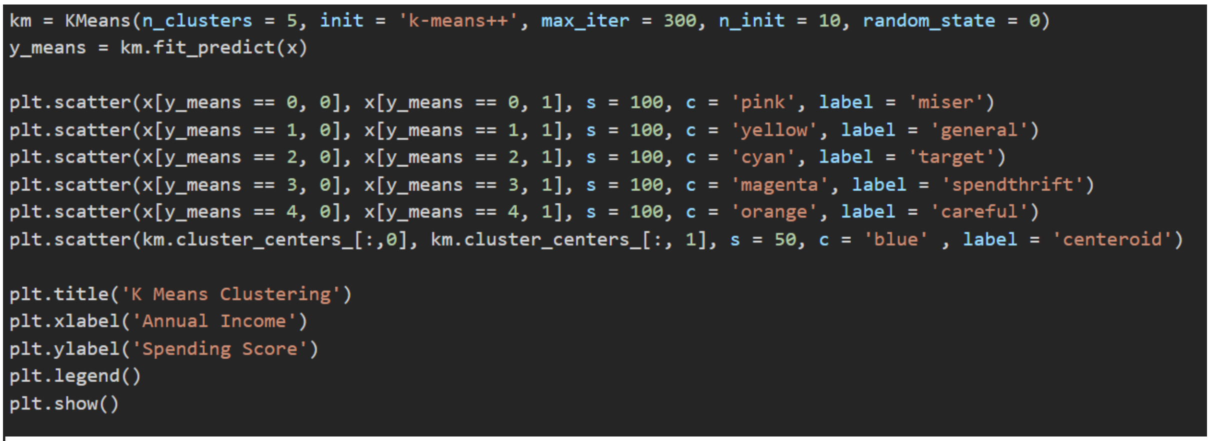


```
data['Annual Income (k$)'].value_counts().plot.bar(figsize = (13, 6))
```

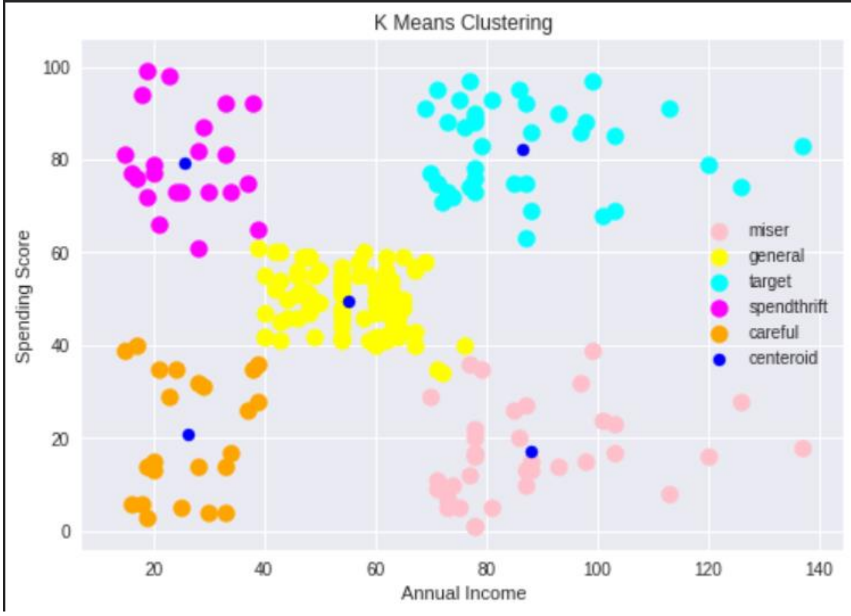
```
matplotlib.axes._subplots.AxesSubplot at 0x7f15fca0d048>
```

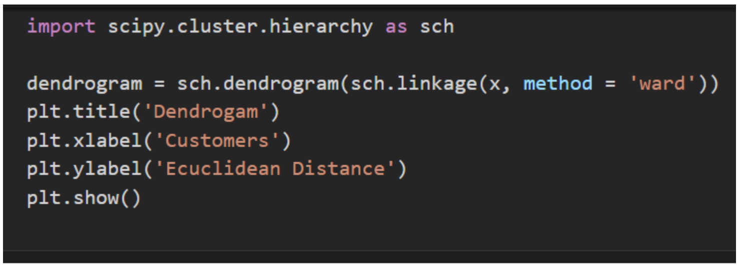


```
fig, axis = plt.subplots(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool), cmap = sns.diverging_palette(220, 10, as_cmap = True),
            square = True, ax = axis)
```

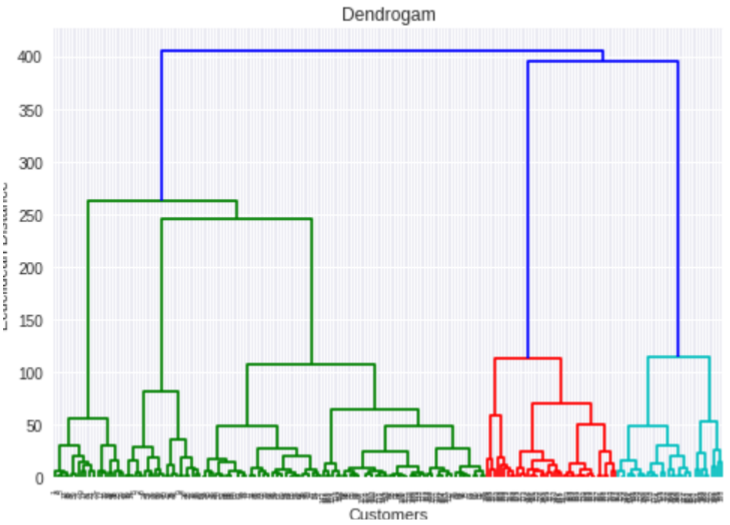


```
km = KMeans(n_clusters = 5, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
y_means = km.fit_predict(x)

plt.scatter(x[y_means == 0, 0], x[y_means == 0, 1], s = 100, c = 'pink', label = 'miser')
plt.scatter(x[y_means == 1, 0], x[y_means == 1, 1], s = 100, c = 'yellow', label = 'general')
plt.scatter(x[y_means == 2, 0], x[y_means == 2, 1], s = 100, c = 'cyan', label = 'target')
plt.scatter(x[y_means == 3, 0], x[y_means == 3, 1], s = 100, c = 'magenta', label = 'spendthrift')
plt.scatter(x[y_means == 4, 0], x[y_means == 4, 1], s = 100, c = 'orange', label = 'careful')
plt.scatter(km.cluster_centers_[:,0], km.cluster_centers_[:, 1], s = 50, c = 'blue' , label = 'centeroid')

plt.title('K Means Clustering')
plt.xlabel('Annual Income')
plt.ylabel('Spending Score')
plt.legend()
plt.show()
```





```
import scipy.cluster.hierarchy as sch

dendrogram = sch.dendrogram(sch.linkage(x, method = 'ward'))
plt.title('Dendrogam')
plt.xlabel('Customers')
plt.ylabel('Ecuclidean Distance')
plt.show()
```



Experiment-7

Aim: WAP To Recognizing Alphabets using SVM

Theory:

Support Vector Machine or SVM is one of the most popular Supervised Learning algorithms, which is used for Classification as well as Regression problems. However, primarily, it is used for Classification problems in Machine Learning.

The goal of the SVM algorithm is to create the best line or decision boundary that can segregate n-dimensional space into classes so that we can easily put the new data point in the correct category in the future. This best decision boundary is called a hyperplane.

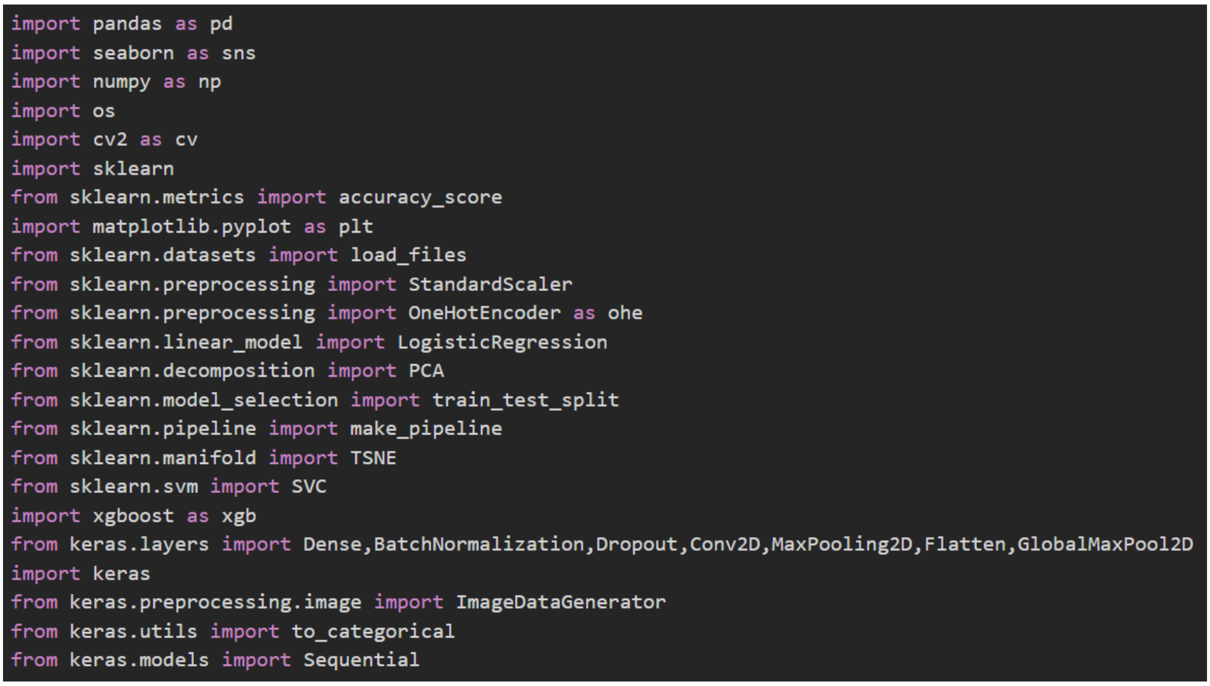
SVM chooses the extreme points/vectors that help in creating the hyperplane. These extreme cases are called as support vectors, and hence algorithm is termed as Support Vector Machine.

SVM can be of two types:

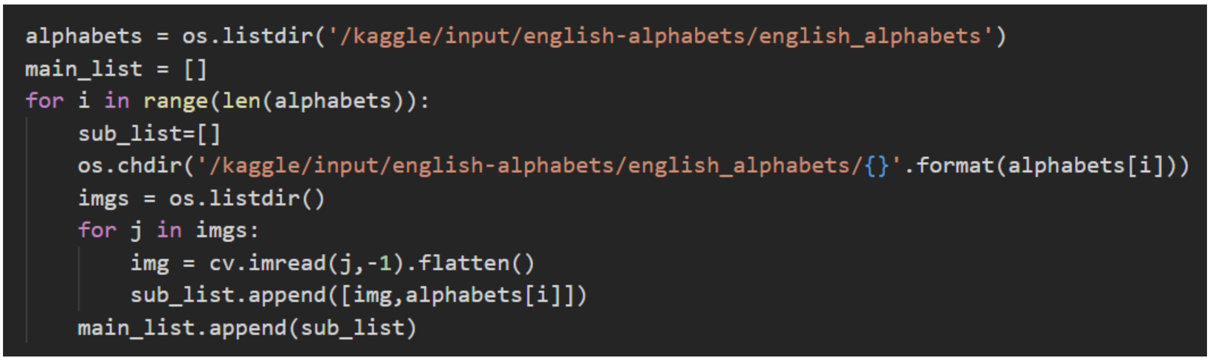
Linear SVM: Linear SVM is used for linearly separable data, which means if a dataset can be classified into two classes by using a single straight line, then such data is termed as linearly separable data, and classifier is used called as Linear SVM classifier.

Non-linear SVM: Non-Linear SVM is used for non-linearly separated data, which means if a dataset cannot be classified by using a straight line, then such data is termed as non-linear data and classifier used is called as Non-linear SVM classifier.

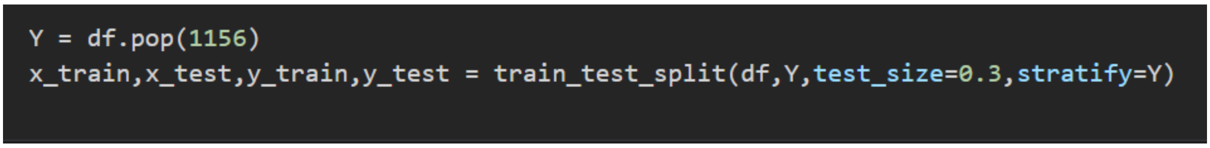
Implementation:



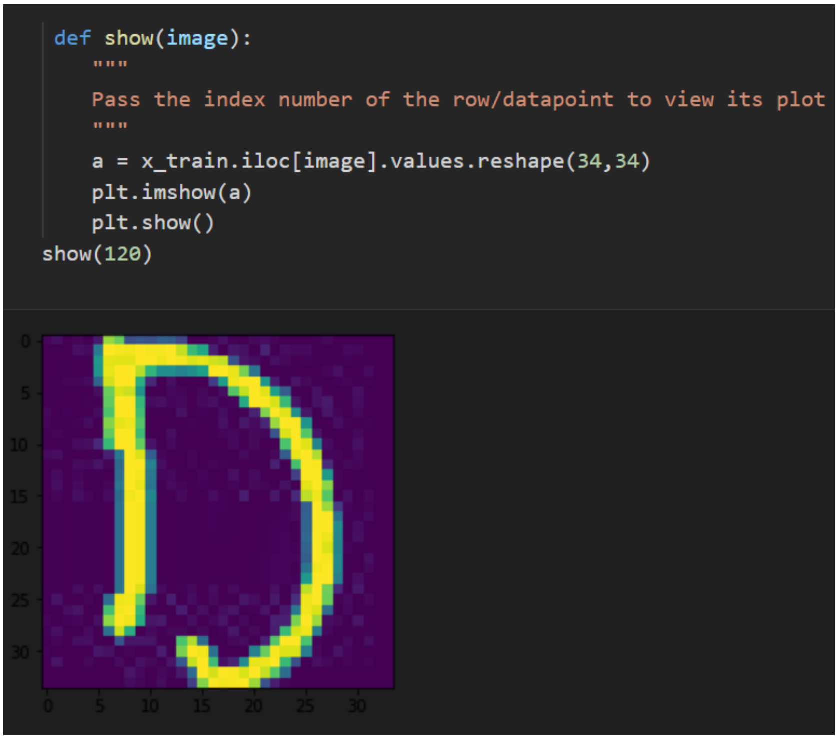
```python
import pandas as pd
import seaborn as sns
import numpy as np
import os
import cv2 as cv
import sklearn
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from sklearn.datasets import load_files
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder as ohe
from sklearn.linear_model import LogisticRegression
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.pipeline import make_pipeline
from sklearn.manifold import TSNE
from sklearn.svm import SVC
import xgboost as xgb
from keras.layers import Dense,BatchNormalization,Dropout,Conv2D,MaxPooling2D,Flatten,GlobalMaxPool2D
import keras
from keras.preprocessing.image import ImageDataGenerator
from keras.utils import to_categorical
from keras.models import Sequential
```



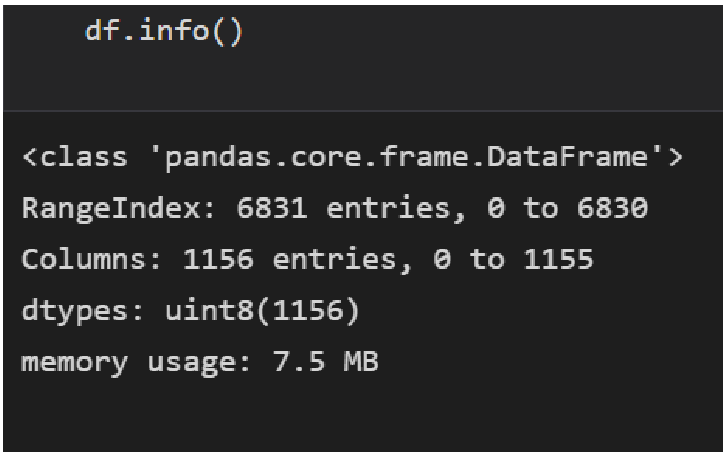
```
alphabets = os.listdir('/kaggle/input/english-alphabets/english_alphabets')
main_list = []
for i in range(len(alphabets)):
    sub_list=[]
    os.chdir('/kaggle/input/english-alphabets/english_alphabets/{}'.format(alphabets[i]))
    imgs = os.listdir()
    for j in imgs:
        img = cv.imread(j,-1).flatten()
        sub_list.append([img,alphabets[i]])
    main_list.append(sub_list)
```



```
Y = df.pop(1156)
x_train,x_test,y_train,y_test = train_test_split(df,Y,test_size=0.3,stratify=Y)
```

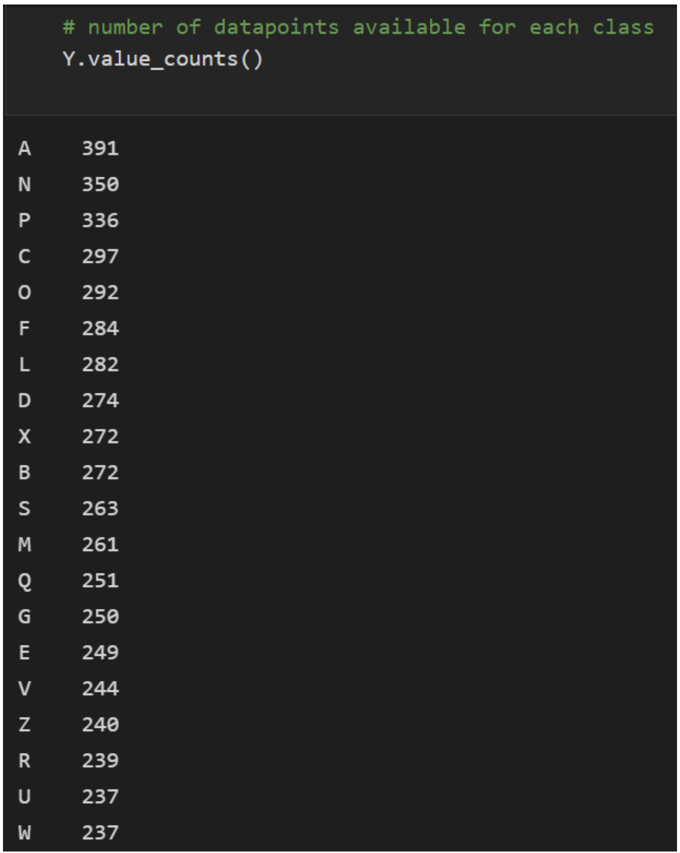


```
def show(image):
    """
    Pass the index number of the row/datapoint to view its plot
    """
    a = x_train.iloc[image].values.reshape(34,34)
    plt.imshow(a)
    plt.show()
show(120)
```



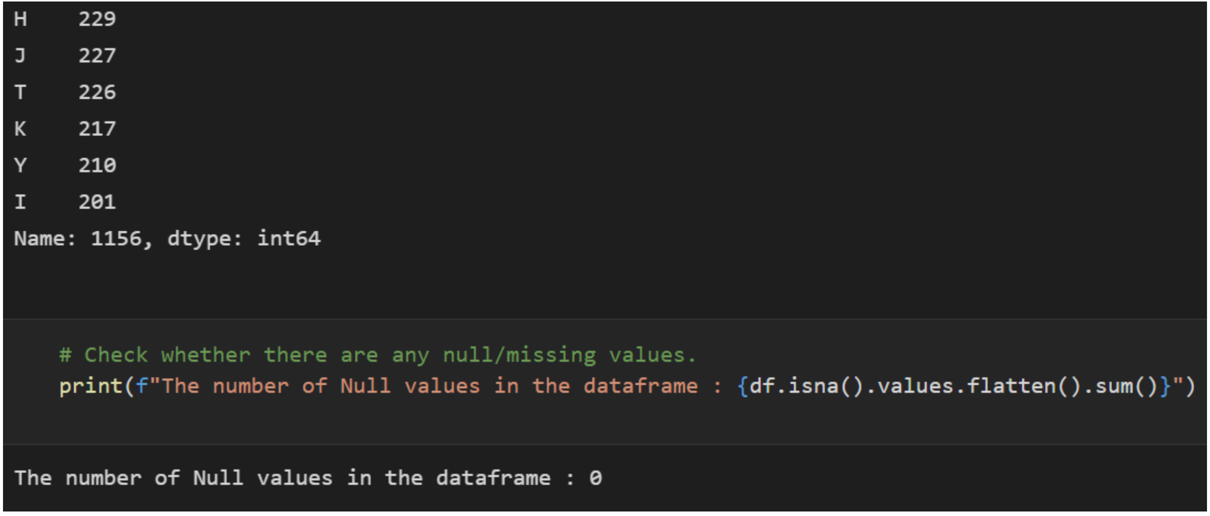
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6831 entries, 0 to 6830
Columns: 1156 entries, 0 to 1155
dtypes: uint8(1156)
memory usage: 7.5 MB
```



```
# number of datapoints available for each class
Y.value_counts()
```

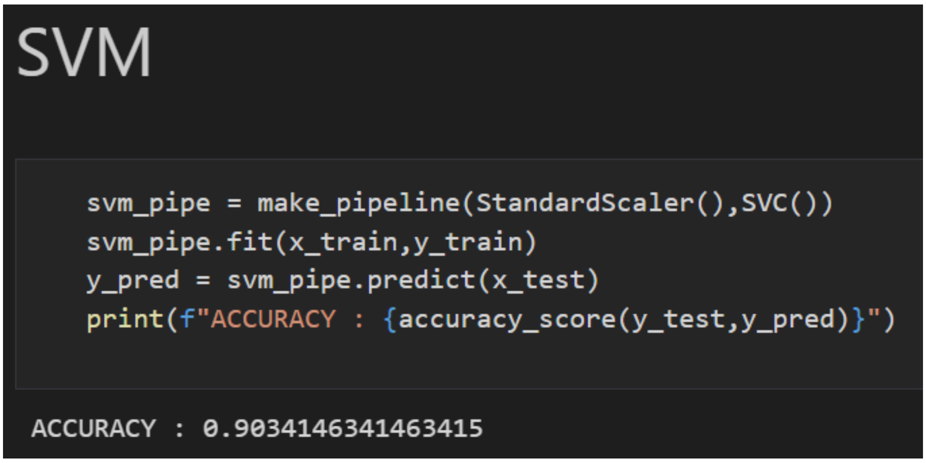
```
A    391
N    350
P    336
C    297
O    292
F    284
L    282
D    274
X    272
B    272
S    263
M    261
Q    251
G    250
E    249
V    244
Z    240
R    239
U    237
W    237
```



```
H    229
J    227
T    226
K    217
Y    210
I    201
Name: 1156, dtype: int64
```

```
# Check whether there are any null/missing values.
print(f"The number of Null values in the dataframe : {df.isna().values.flatten().sum()}")
```

```
The number of Null values in the dataframe : 0
```



SVM

```
svm_pipe = make_pipeline(StandardScaler(),SVC())
svm_pipe.fit(x_train,y_train)
y_pred = svm_pipe.predict(x_test)
print(f"ACCURACY : {accuracy_score(y_test,y_pred)}")
```

```
ACCURACY : 0.9034146341463415
```

Experiment-8

Aim: WAP To implement K Means Algorithm on Given Data Set.

Theory:

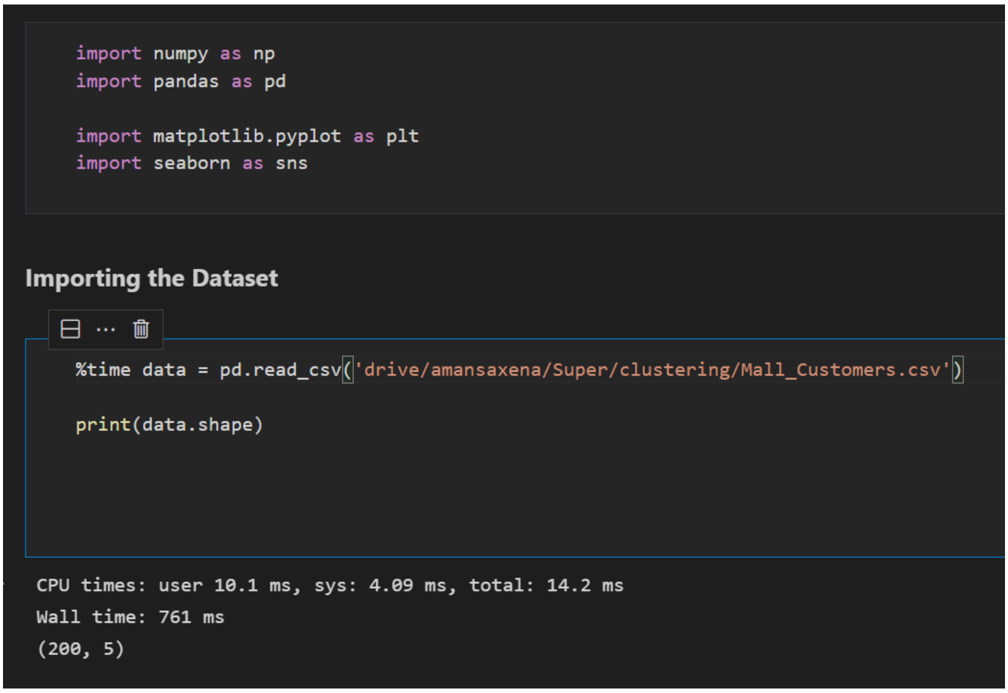
K-Means Clustering is an unsupervised learning algorithm that is used to solve the clustering problems in machine learning or data science. In this topic, we will learn what is K-means clustering algorithm, how the algorithm works, along with the Python implementation of k-means clustering. K-Means Clustering is an Unsupervised Learning algorithm, which groups the unlabelled dataset into different clusters. Here K defines the number of pre-defined clusters that need to be created in the process, as if K=2, there will be two clusters, and for K=3, there will be three clusters, and so on. It allows us to cluster the data into different groups and a convenient way to discover the categories of groups in the unlabelled dataset on its own without the need for any training.

It is a centroid-based algorithm, where each cluster is associated with a centroid. The main aim of this algorithm is to minimize the sum of distances between the data point and their corresponding clusters. The algorithm takes the unlabelled dataset as input, divides the dataset into k-number of clusters, and repeats the process until it does not find the best clusters. The value of k should be predetermined in this algorithm.

The k-means clustering algorithm mainly performs two tasks:

- Determines the best value for K centre points or centroids by an iterative process.
- Assigns each data point to its closest k-centre. Those data points which are near to the k-centre, create a cluster.

Implementation:

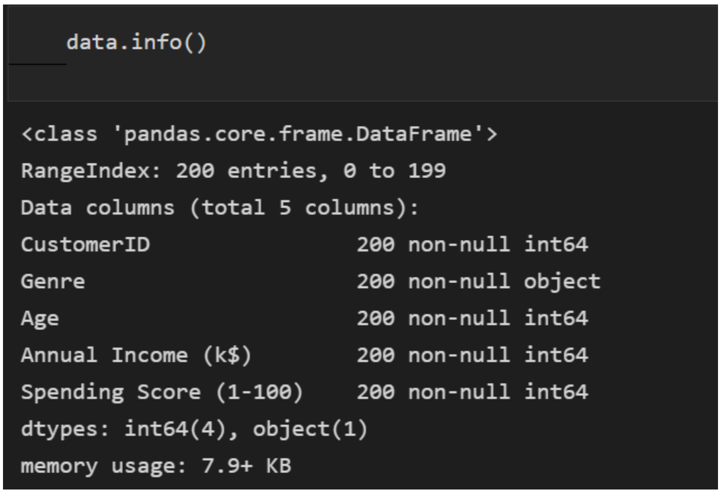


```
import numpy as np
import pandas as pd

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

Importing the Dataset

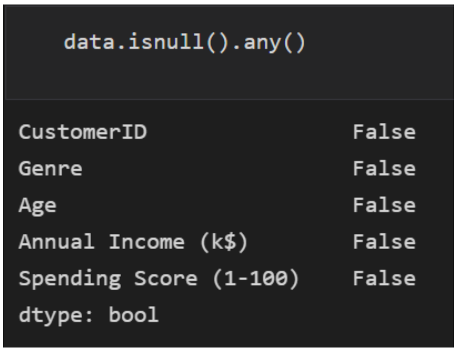
```
%time data = pd.read_csv('drive/amansaxena/Super/clustering/Mall_Customers.csv')

print(data.shape)
```

```
CPU times: user 10.1 ms, sys: 4.09 ms, total: 14.2 ms
Wall time: 761 ms
(200, 5)
```



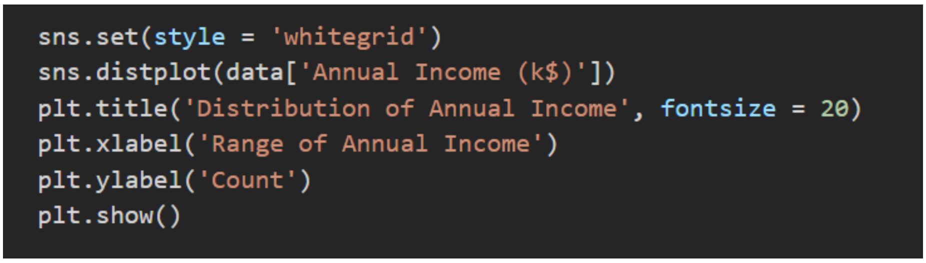
```
data.info()
```

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

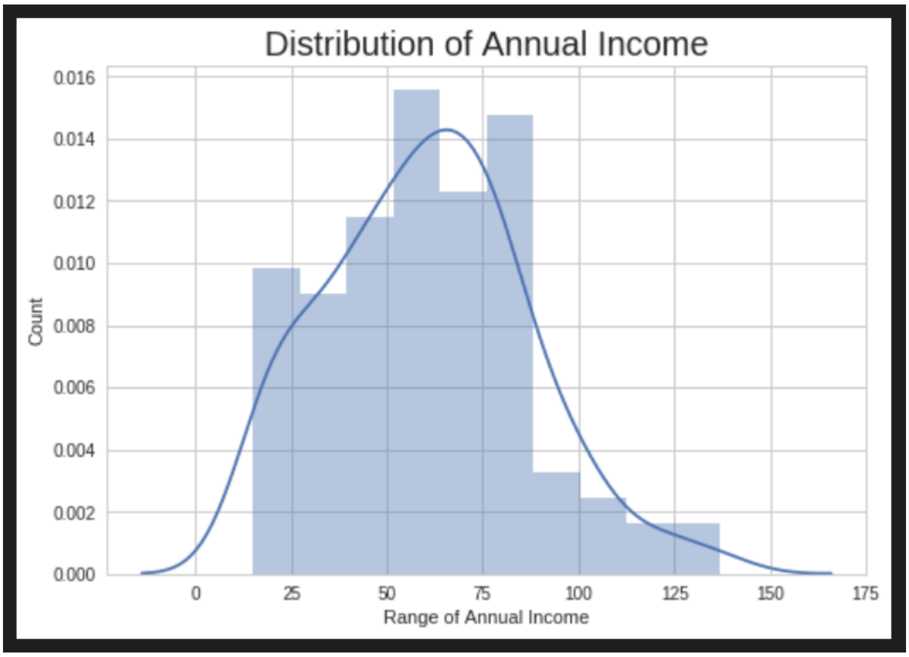


```
data.isnull().any()
```

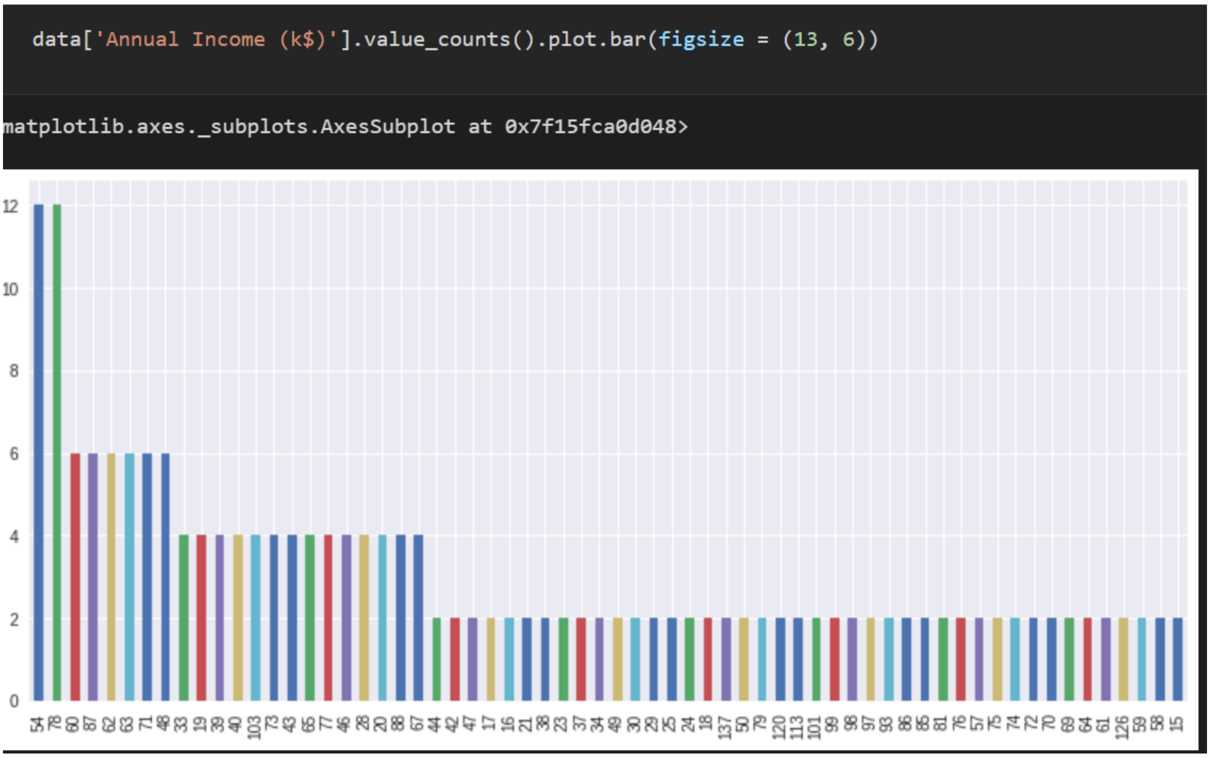
```
CustomerID                False
Genre                     False
Age                       False
Annual Income (k$)        False
Spending Score (1-100)    False
dtype: bool
```



```
sns.set(style = 'whitegrid')
sns.distplot(data['Annual Income (k$)'])
plt.title('Distribution of Annual Income', fontsize = 20)
plt.xlabel('Range of Annual Income')
plt.ylabel('Count')
plt.show()
```

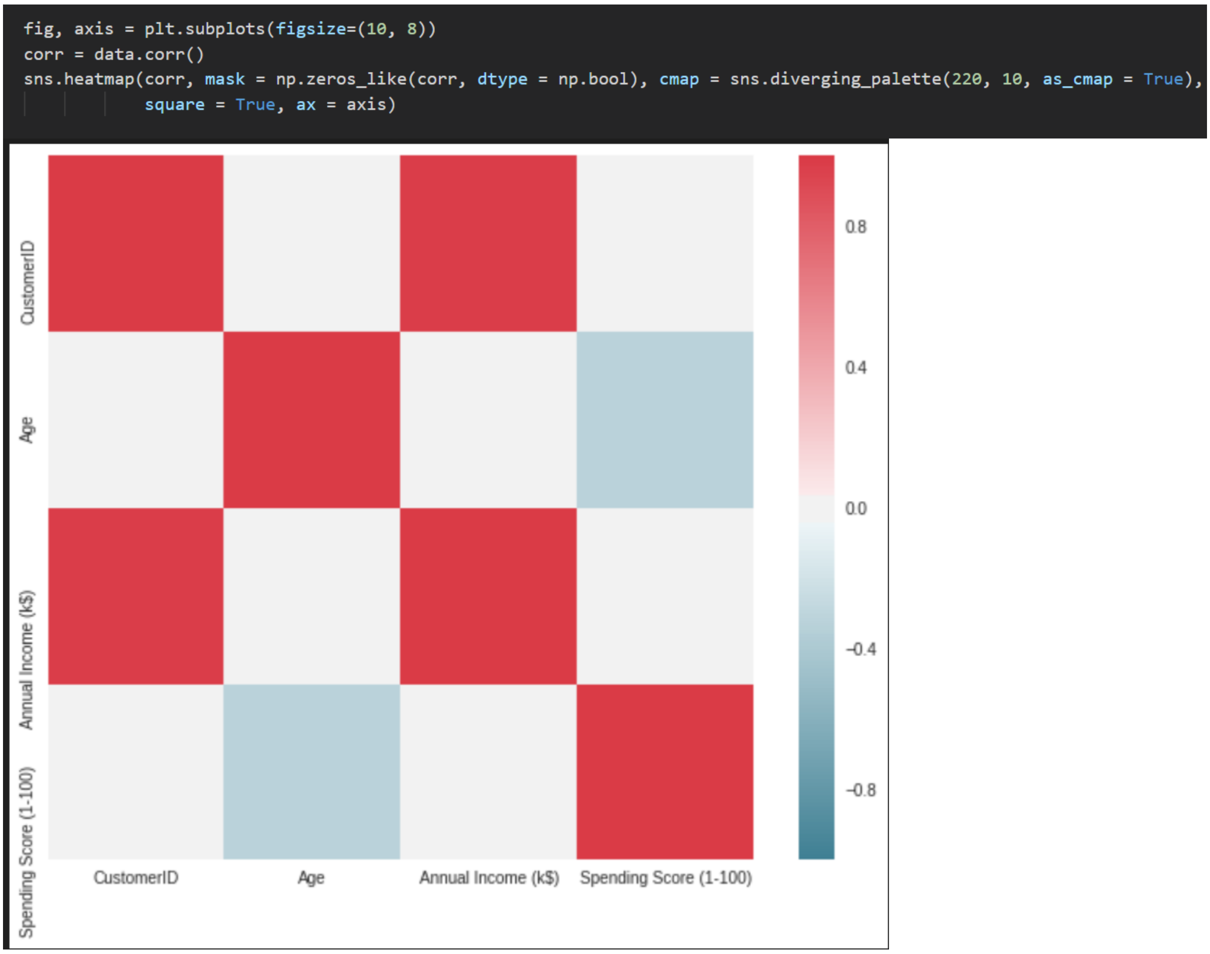


Distribution of Annual Income

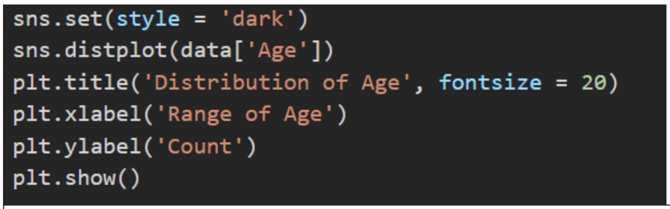


```
data['Annual Income (k$)'].value_counts().plot.bar(figsize = (13, 6))
```

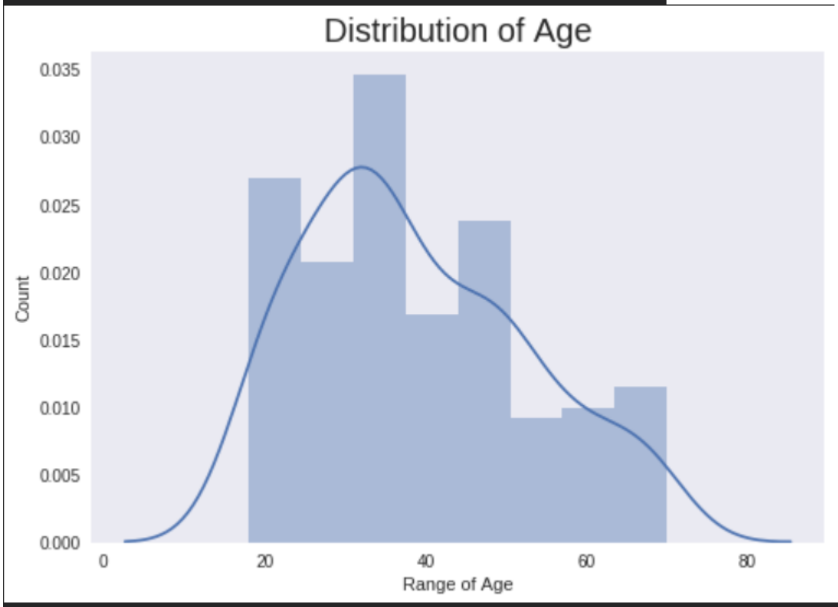
```
matplotlib.axes._subplots.AxesSubplot at 0x7f15fca0d048>
```

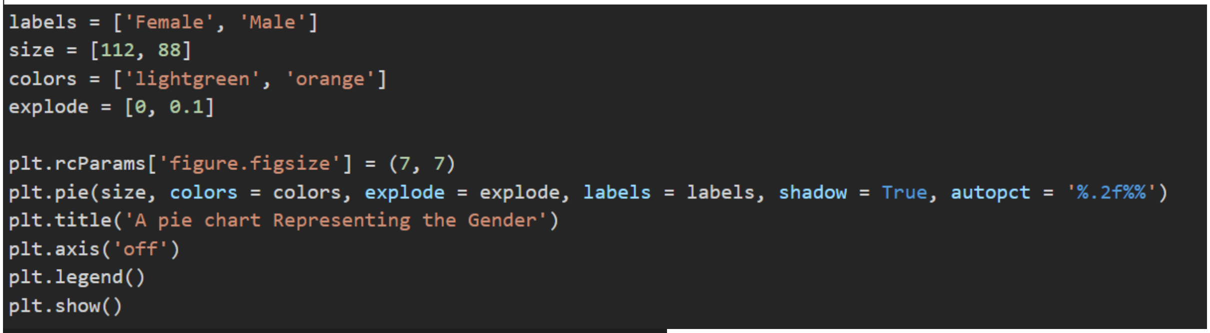


```
fig, axis = plt.subplots(figsize=(10, 8))
corr = data.corr()
sns.heatmap(corr, mask = np.zeros_like(corr, dtype = np.bool), cmap = sns.diverging_palette(220, 10, as_cmap = True),
            square = True, ax = axis)
```

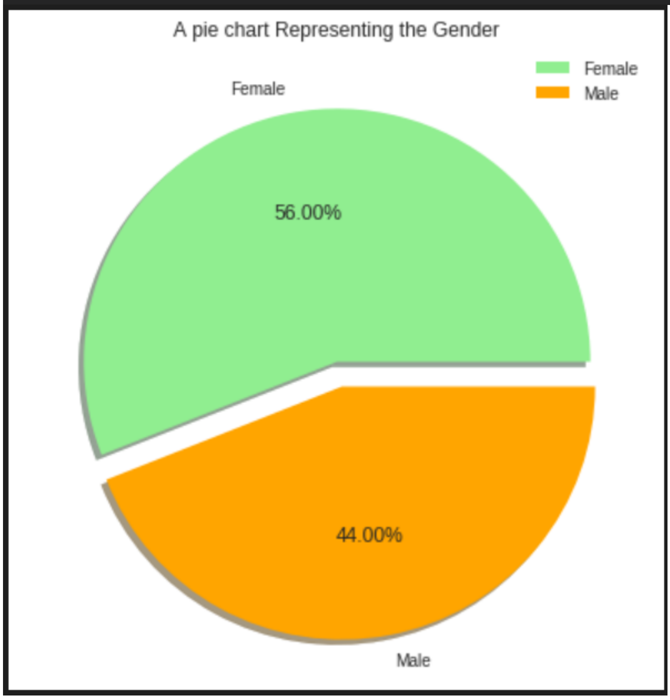


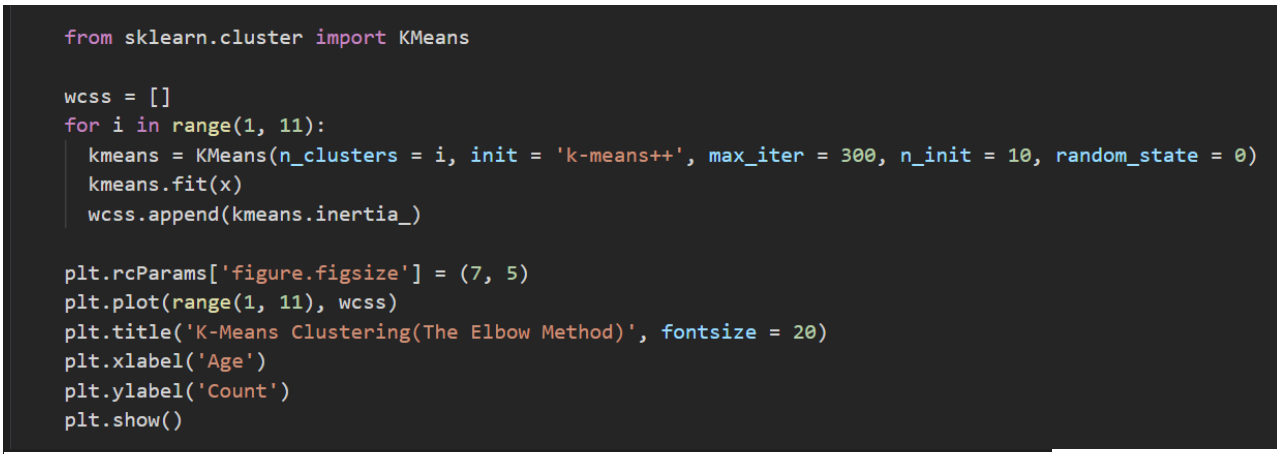
```
sns.set(style = 'dark')
sns.distplot(data['Age'])
plt.title('Distribution of Age', fontsize = 20)
plt.xlabel('Range of Age')
plt.ylabel('Count')
plt.show()
```



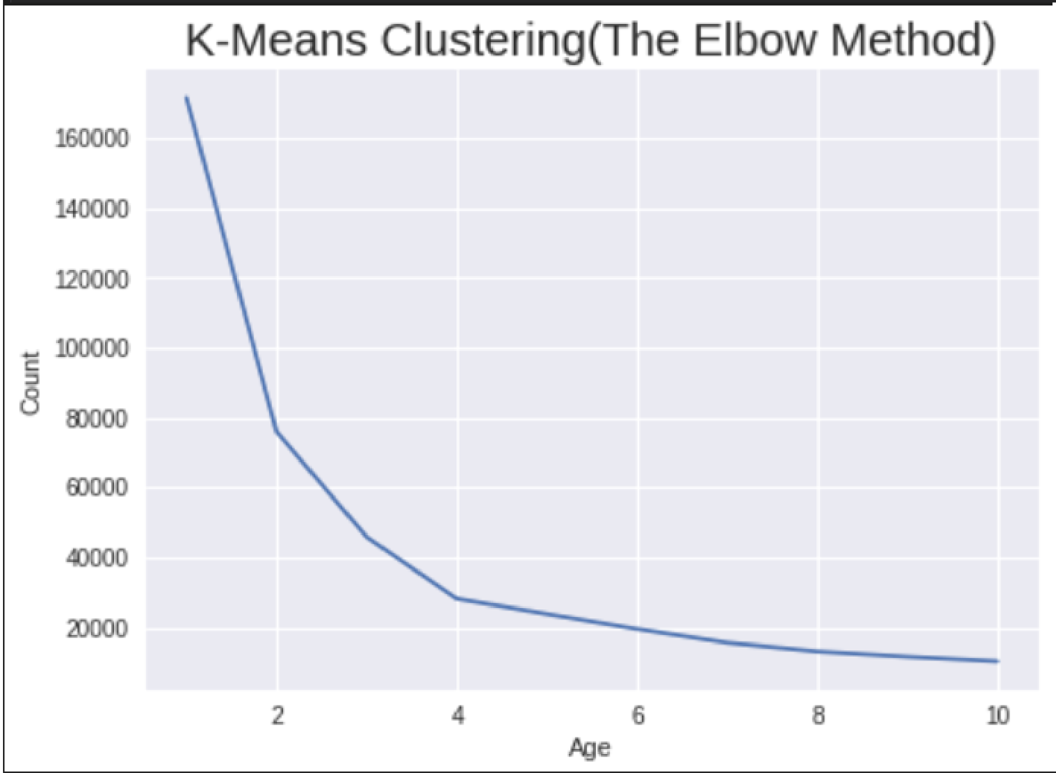


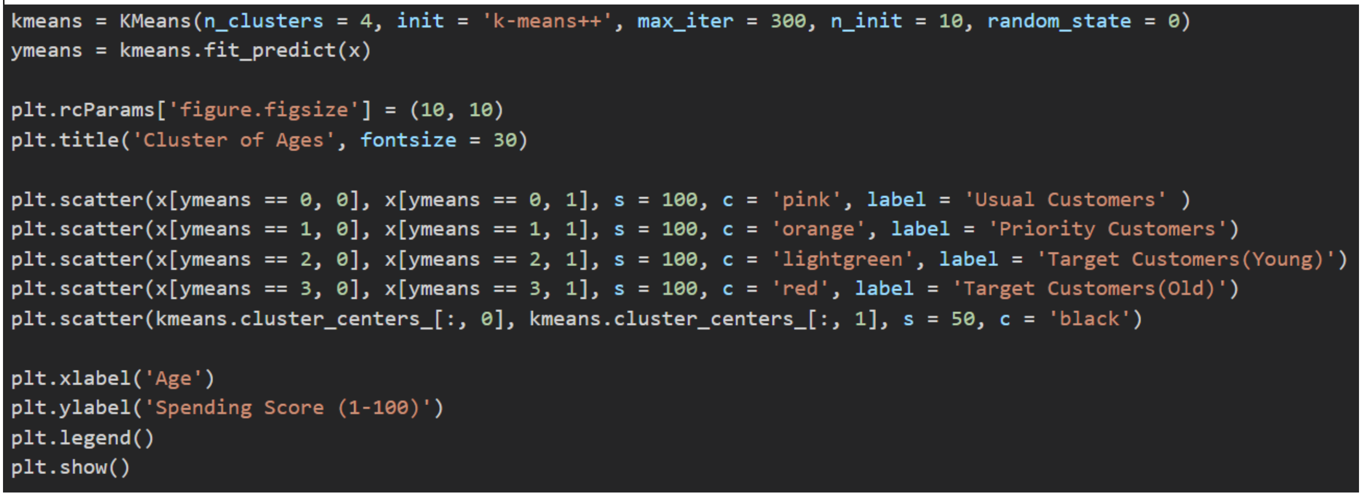
```
labels = ['Female', 'Male']
size = [112, 88]
colors = ['lightgreen', 'orange']
explode = [0, 0.1]

plt.rcParams['figure.figsize'] = (7, 7)
plt.pie(size, colors = colors, explode = explode, labels = labels, shadow = True, autopct = '%.2f%%')
plt.title('A pie chart Representing the Gender')
plt.axis('off')
plt.legend()
plt.show()
```



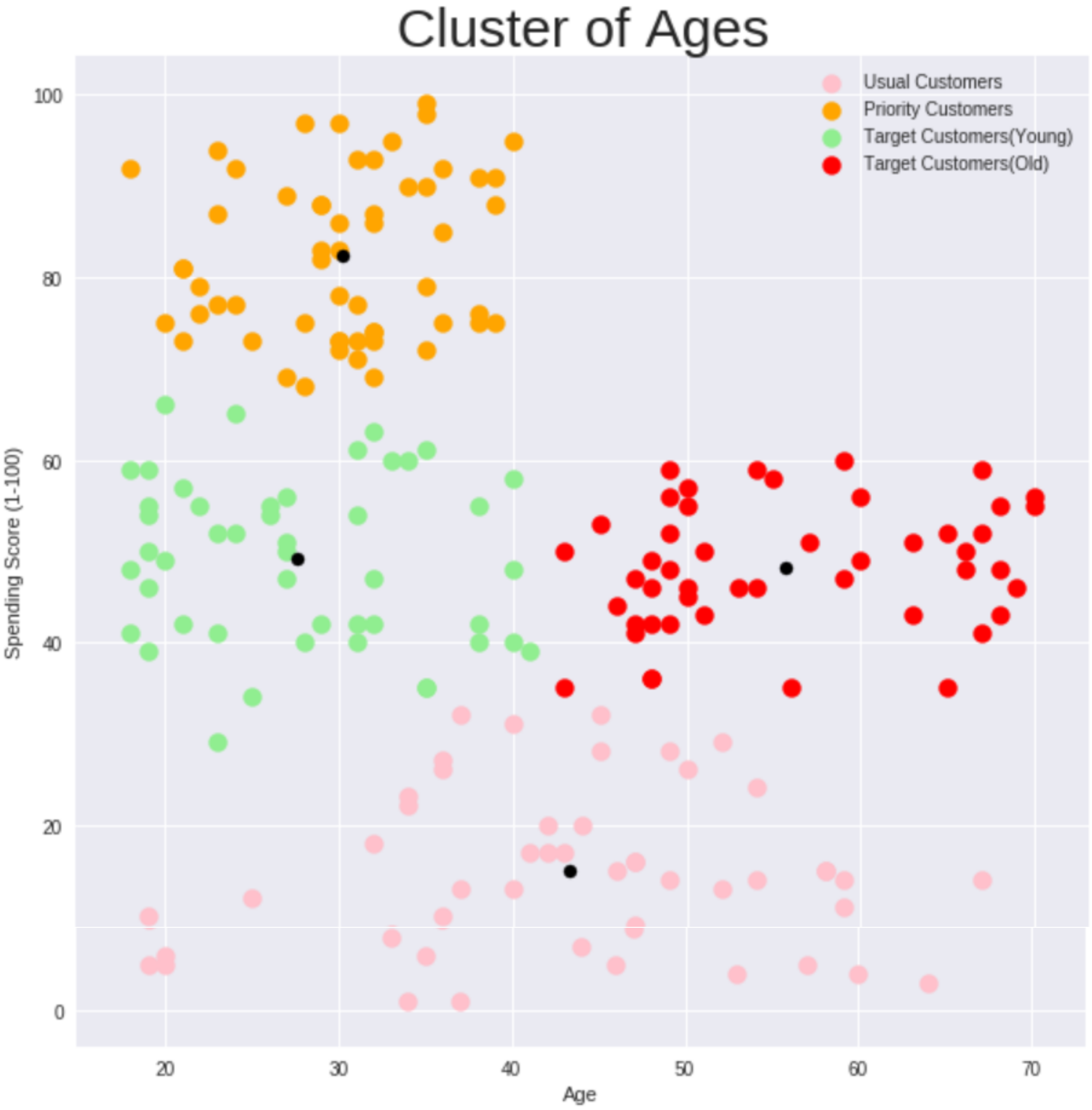


```
from sklearn.cluster import KMeans

wcss = []
for i in range(1, 11):
  kmeans = KMeans(n_clusters = i, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
  kmeans.fit(x)
  wcss.append(kmeans.inertia_)

plt.rcParams['figure.figsize'] = (7, 5)
plt.plot(range(1, 11), wcss)
plt.title('K-Means Clustering(The Elbow Method)', fontsize = 20)
plt.xlabel('Age')
plt.ylabel('Count')
plt.show()
```





```
kmeans = KMeans(n_clusters = 4, init = 'k-means++', max_iter = 300, n_init = 10, random_state = 0)
ymeans = kmeans.fit_predict(x)

plt.rcParams['figure.figsize'] = (10, 10)
plt.title('Cluster of Ages', fontsize = 30)

plt.scatter(x[ymeans == 0, 0], x[ymeans == 0, 1], s = 100, c = 'pink', label = 'Usual Customers' )
plt.scatter(x[ymeans == 1, 0], x[ymeans == 1, 1], s = 100, c = 'orange', label = 'Priority Customers')
plt.scatter(x[ymeans == 2, 0], x[ymeans == 2, 1], s = 100, c = 'lightgreen', label = 'Target Customers(Young)')
plt.scatter(x[ymeans == 3, 0], x[ymeans == 3, 1], s = 100, c = 'red', label = 'Target Customers(Old)')
plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 50, c = 'black')

plt.xlabel('Age')
plt.ylabel('Spending Score (1-100)')
plt.legend()
plt.show()
```


Cluster of Ages
Usual Customers
Priority Customers
Target Customers(Young)
Target Customers(Old)
Spending Score (1-100)
100
80
60
40
20
0
20
30
40
50
60
70
Age

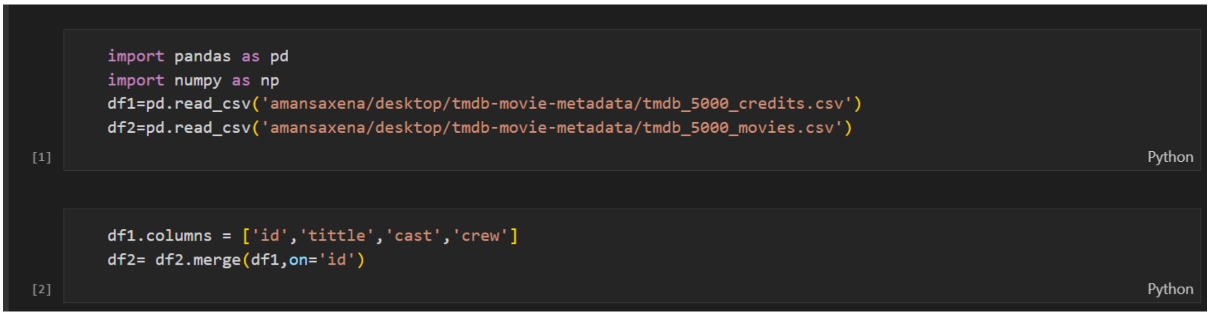
Experiment-9

Aim: WAP To Make a Movie Recommendation System

Theory:

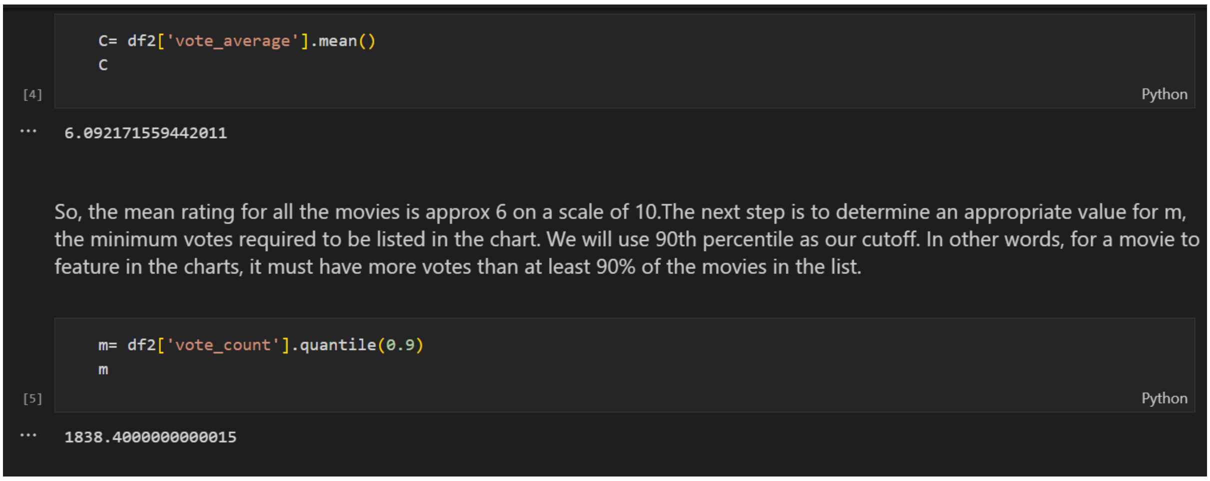
A movie recommendation system, or a movie recommender system, is an ML-based approach to filtering or predicting the users' film preferences based on their past choices and behaviour. It is an advanced filtration mechanism that predicts the possible movie choices of the concerned user and their preferences towards a domain-specific item, aka movie. The basic concept behind a movie recommendation system is quite simple. There are two main elements in every recommender system: users and items. The system generates movie predictions for its users, while items are the movies themselves. The primary goal of movie recommendation systems is to filter and predict only those movies that a corresponding user is most likely to want to watch. The ML algorithms for these recommendation systems use the data about this user from the system's database. This data is used to predict the future behaviour of the user concerned based on the information from the past.

Implementation:



```python
import pandas as pd
import numpy as np
df1=pd.read_csv('amansaxena/desktop/tmdb-movie-metadata/tmdb_5000_credits.csv')
df2=pd.read_csv('amansaxena/desktop/tmdb-movie-metadata/tmdb_5000_movies.csv')
```

```python
df1.columns = ['id','tittle','cast','crew']
df2= df2.merge(df1,on='id')
```



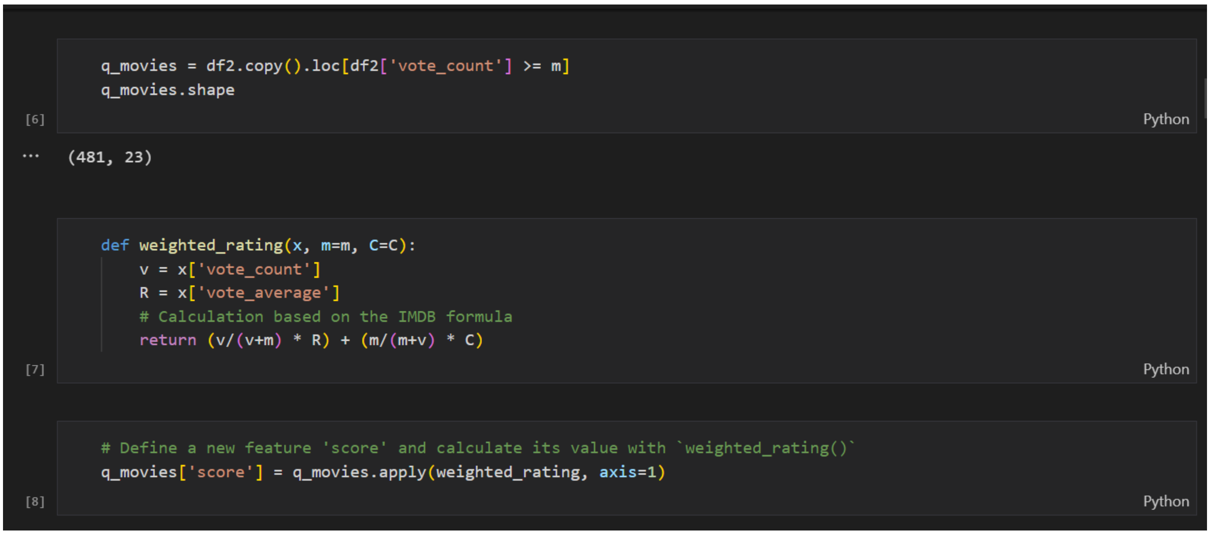
```python
C= df2['vote_average'].mean()
C
```

```
6.092171559442011
```

So, the mean rating for all the movies is approx 6 on a scale of 10.The next step is to determine an appropriate value for m, the minimum votes required to be listed in the chart. We will use 90th percentile as our cutoff. In other words, for a movie to feature in the charts, it must have more votes than at least 90% of the movies in the list.

```python
m= df2['vote_count'].quantile(0.9)
m
```

```
1838.4000000000015
```

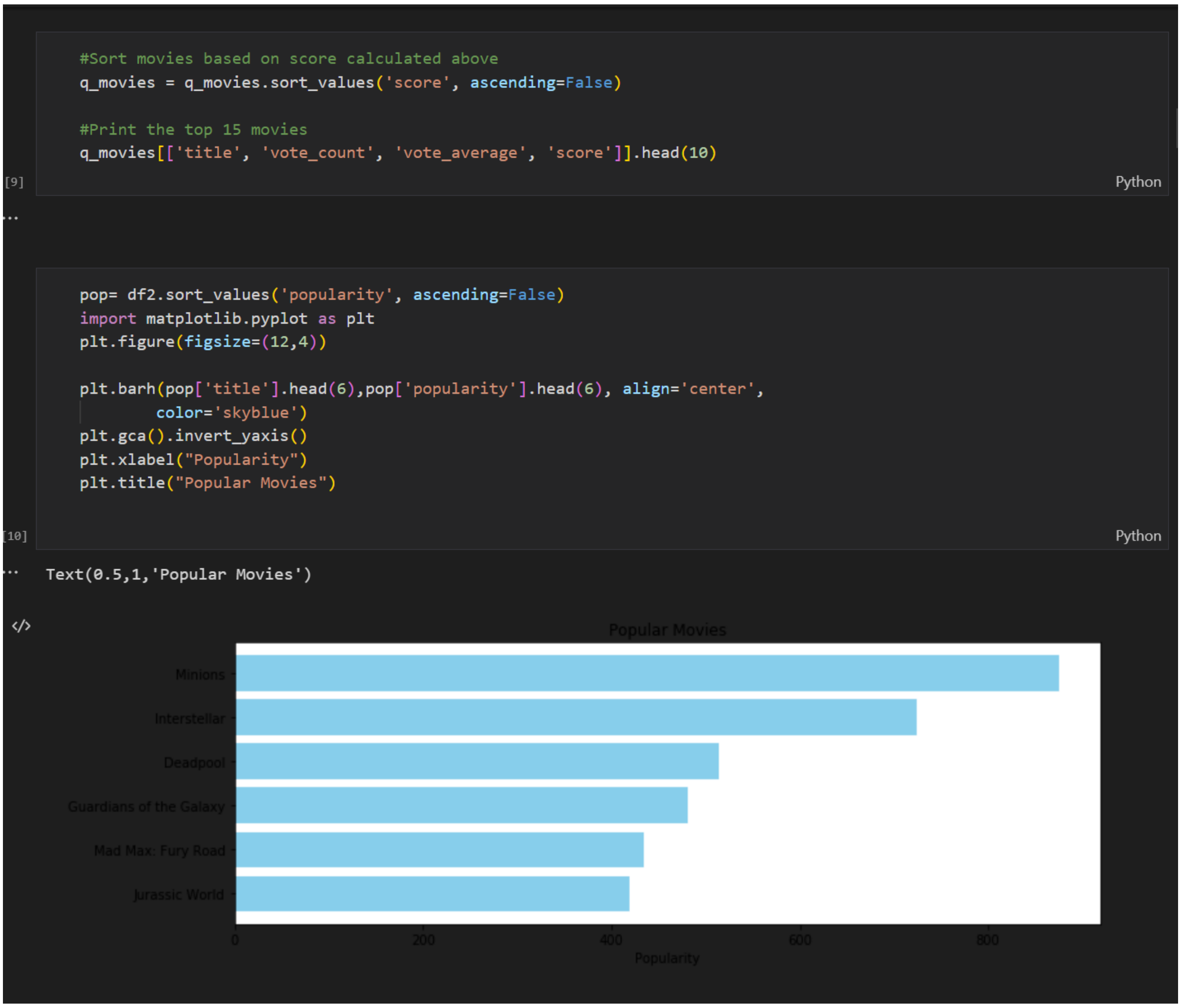


```python
q_movies = df2.copy().loc[df2['vote_count'] >= m]
q_movies.shape
```

```
(481, 23)
```

```python
def weighted_rating(x, m=m, C=C):
    v = x['vote_count']
    R = x['vote_average']
    # Calculation based on the IMDB formula
    return (v/(v+m) * R) + (m/(m+v) * C)
```

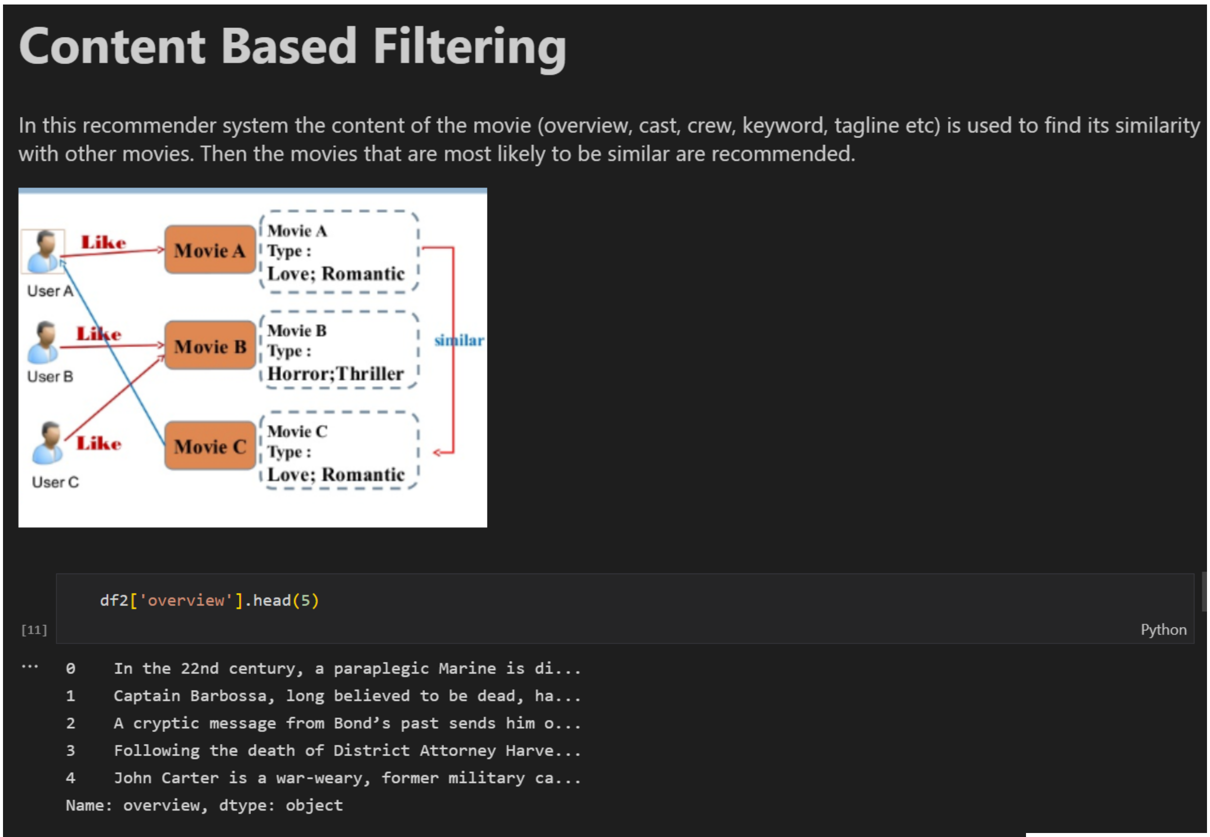
```python
# Define a new feature 'score' and calculate its value with `weighted_rating()`
q_movies['score'] = q_movies.apply(weighted_rating, axis=1)
```



```python
#Sort movies based on score calculated above
q_movies = q_movies.sort_values('score', ascending=False)

#Print the top 15 movies
q_movies[['title', 'vote_count', 'vote_average', 'score']].head(10)
```

```python
pop= df2.sort_values('popularity', ascending=False)
import matplotlib.pyplot as plt
plt.figure(figsize=(12,4))

plt.barh(pop['title'].head(6),pop['popularity'].head(6), align='center',
        color='skyblue')
plt.gca().invert_yaxis()
plt.xlabel("Popularity")
plt.title("Popular Movies")
```

```
Text(0.5,1,'Popular Movies')
```

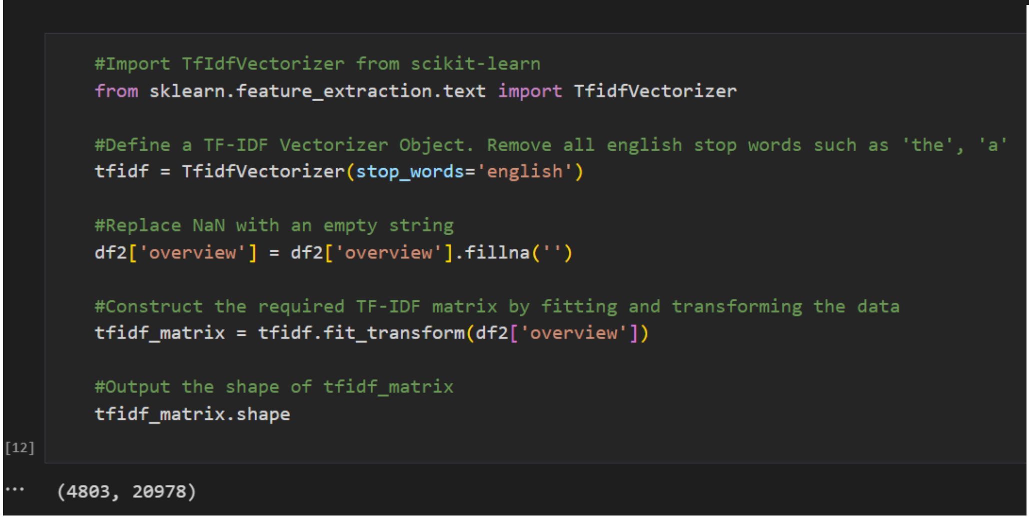


Content Based Filtering

In this recommender system the content of the movie (overview, cast, crew, keyword, tagline etc) is used to find its similarity with other movies. Then the movies that are most likely to be similar are recommended.

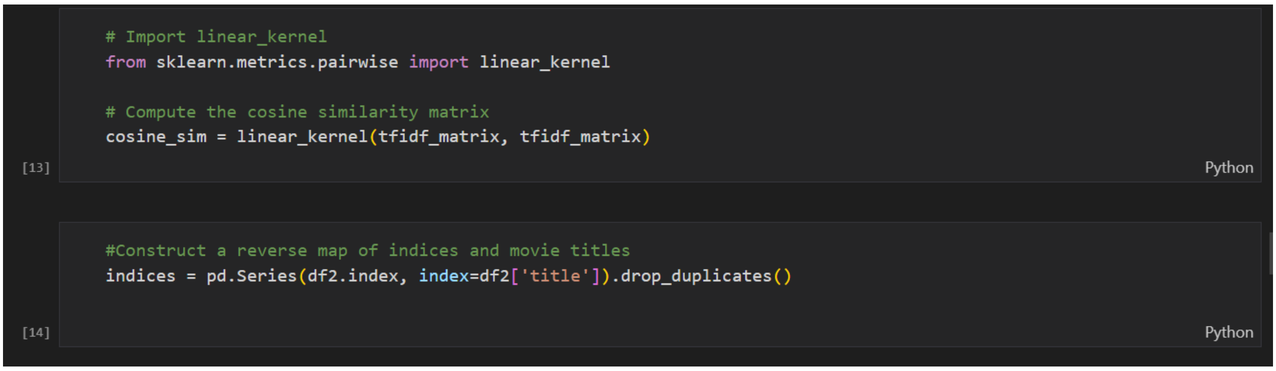
```
df2['overview'].head(5)
```

```
0    In the 22nd century, a paraplegic Marine is di...
1    Captain Barbossa, long believed to be dead, ha...
2    A cryptic message from Bond’s past sends him o...
3    Following the death of District Attorney Harve...
4    John Carter is a war-weary, former military ca...
Name: overview, dtype: object
```



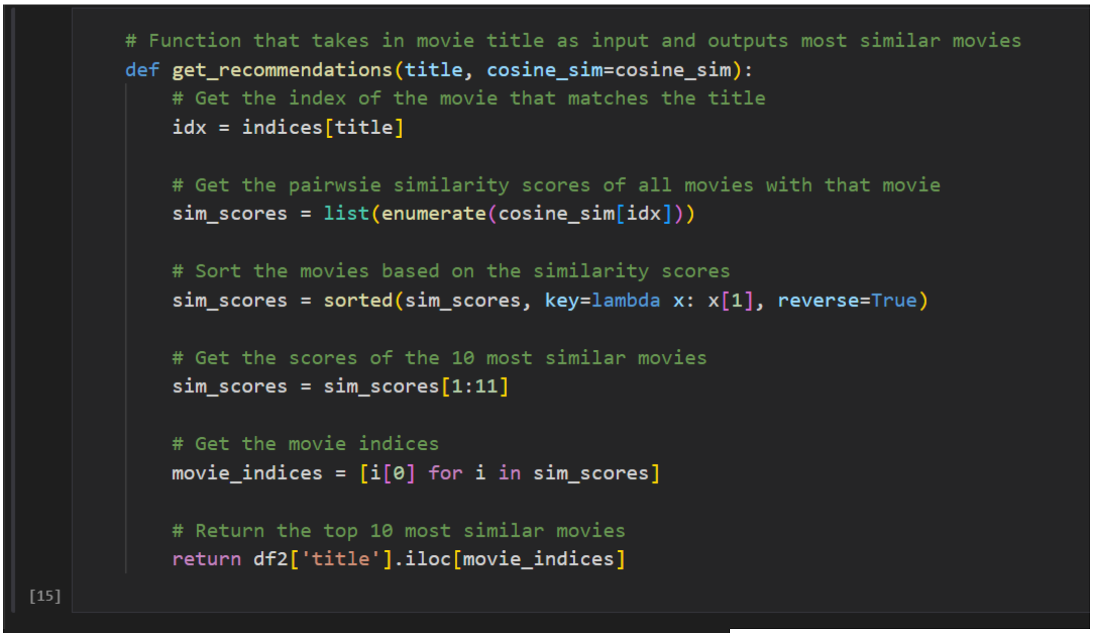
```
#Import TfIdfVectorizer from scikit-learn
from sklearn.feature_extraction.text import TfidfVectorizer

#Define a TF-IDF Vectorizer Object. Remove all english stop words such as 'the', 'a'
tfidf = TfidfVectorizer(stop_words='english')

#Replace NaN with an empty string
df2['overview'] = df2['overview'].fillna('')

#Construct the required TF-IDF matrix by fitting and transforming the data
tfidf_matrix = tfidf.fit_transform(df2['overview'])

#Output the shape of tfidf_matrix
tfidf_matrix.shape
```

```
(4803, 20978)
```



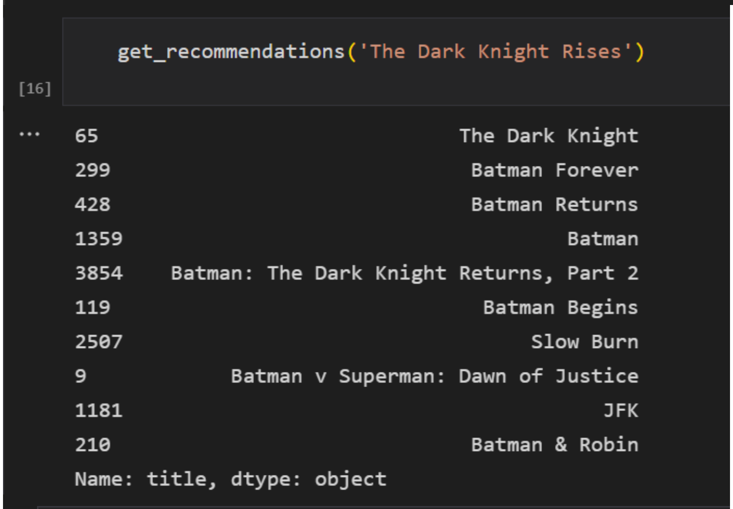
```
# Import linear_kernel
from sklearn.metrics.pairwise import linear_kernel

# Compute the cosine similarity matrix
cosine_sim = linear_kernel(tfidf_matrix, tfidf_matrix)
```

```
#Construct a reverse map of indices and movie titles
indices = pd.Series(df2.index, index=df2['title']).drop_duplicates()
```



```
# Function that takes in movie title as input and outputs most similar movies
def get_recommendations(title, cosine_sim=cosine_sim):
    # Get the index of the movie that matches the title
    idx = indices[title]

    # Get the pairwsie similarity scores of all movies with that movie
    sim_scores = list(enumerate(cosine_sim[idx]))

    # Sort the movies based on the similarity scores
    sim_scores = sorted(sim_scores, key=lambda x: x[1], reverse=True)

    # Get the scores of the 10 most similar movies
    sim_scores = sim_scores[1:11]

    # Get the movie indices
    movie_indices = [i[0] for i in sim_scores]

    # Return the top 10 most similar movies
    return df2['title'].iloc[movie_indices]
```

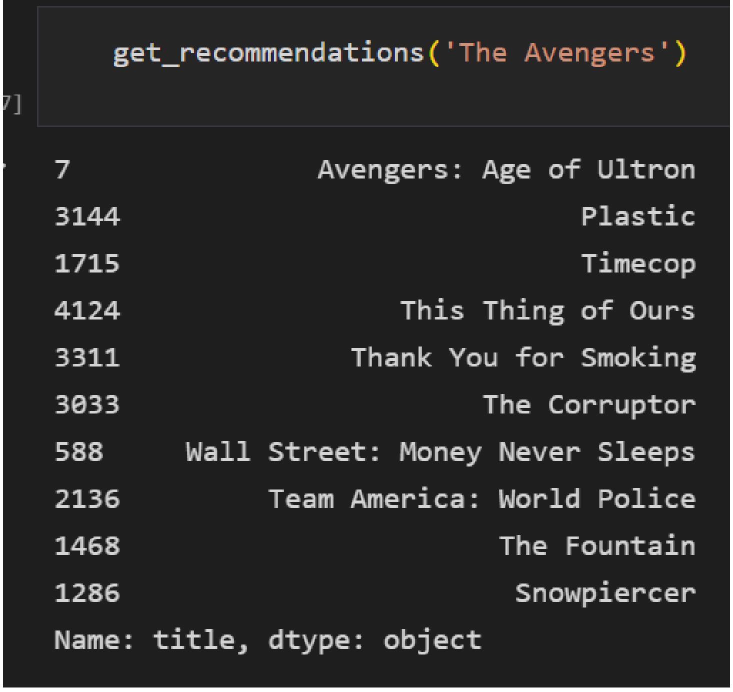
[15]



```
get_recommendations('The Dark Knight Rises')
```

[16]

```
65                               The Dark Knight
299                               Batman Forever
428                               Batman Returns
1359                                      Batman
3854    Batman: The Dark Knight Returns, Part 2
119                                Batman Begins
2507                                   Slow Burn
9            Batman v Superman: Dawn of Justice
1181                                         JFK
210                               Batman & Robin
Name: title, dtype: object
```



```
get_recommendations('The Avengers')
```

7]

```
7               Avengers: Age of Ultron
3144                            Plastic
1715                            Timecop
4124                 This Thing of Ours
3311              Thank You for Smoking
3033                      The Corruptor
588     Wall Street: Money Never Sleeps
2136         Team America: World Police
1468                       The Fountain
1286                        Snowpiercer
Name: title, dtype: object
```

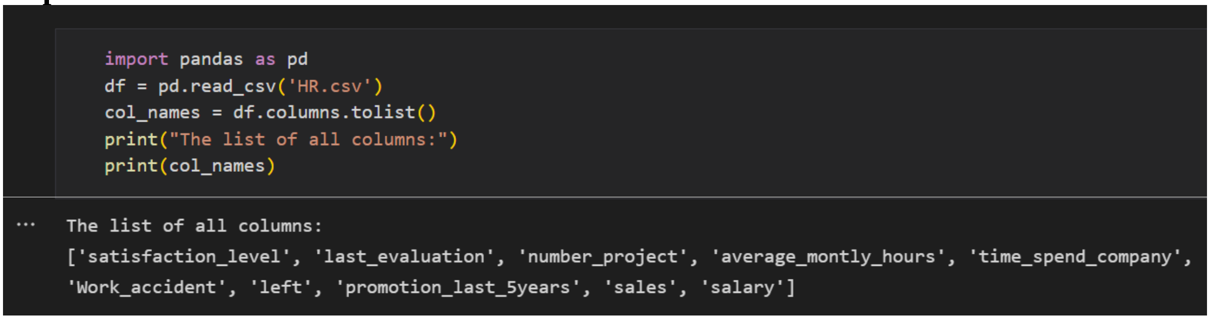
Experiment-10

Aim: WAP To Develop a prediction mechanism to predict which employee can go on leave in a company in near future.

Theory:

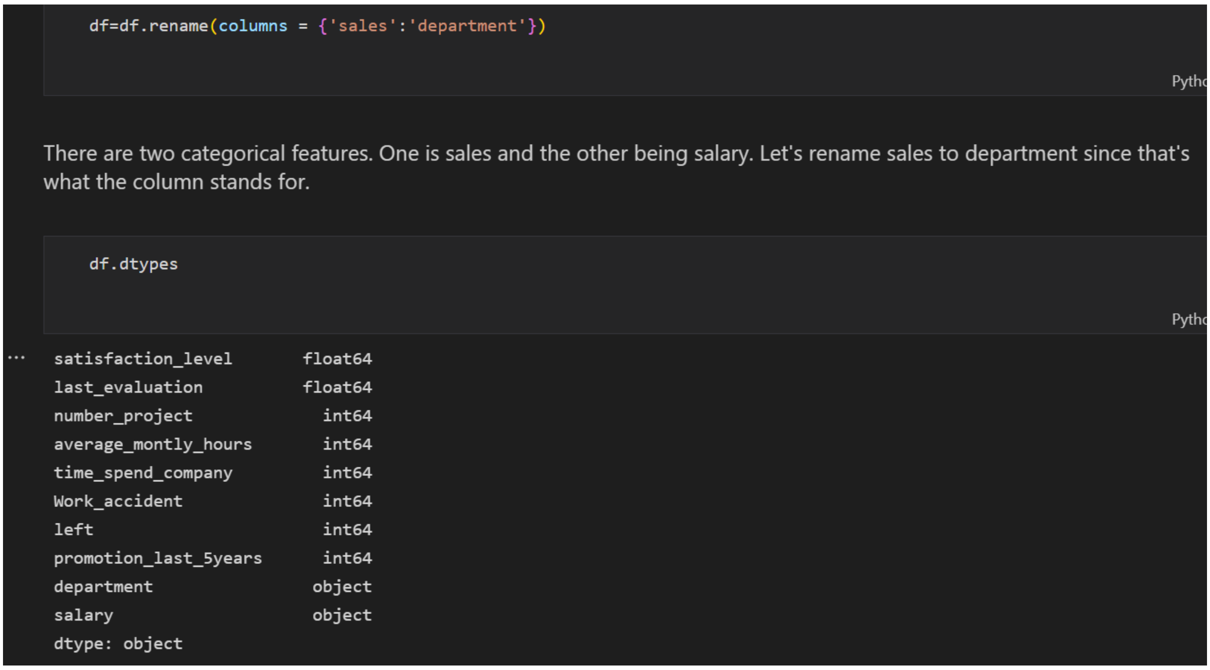
Prediction in machine learning refers to the output of an algorithm after it has been trained on a historical dataset and applied to new data when forecasting the likelihood of a particular outcome. Predicting supports the development of critical thinking skills by requiring students to draw upon their prior knowledge and experiences as well as observations to anticipate what might happen. The ability to make logical predictions supports the development of the ability to formulate hypotheses. Prediction mechanism to predict which employee can go on leave in a company in near future will not only help the human resource department to analysis that which employee will not be present and when but will also help the managerial department or the project manager to distribute the work accordingly among the people or the project team members which have a crucial role to play for the submission of the project on time as per the requirement of the stakeholders and the users of the project output.

Implementation:



```
import pandas as pd
df = pd.read_csv('HR.csv')
col_names = df.columns.tolist()
print("The list of all columns:")
print(col_names)
```

```
... The list of all columns:
    ['satisfaction_level', 'last_evaluation', 'number_project', 'average_montly_hours', 'time_spend_company',
    'Work_accident', 'left', 'promotion_last_5years', 'sales', 'salary']
```

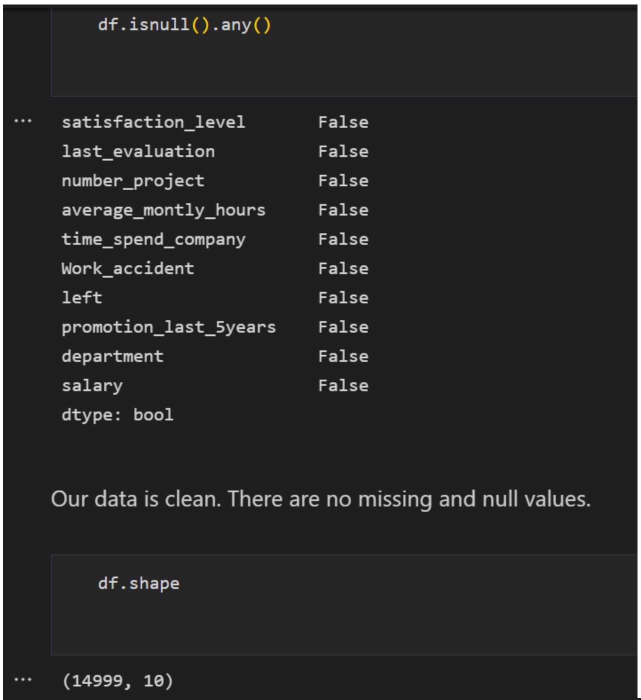


```
df=df.rename(columns = {'sales':'department'})
```

There are two categorical features. One is sales and the other being salary. Let's rename sales to department since that's what the column stands for.

```
df.dtypes
```

```
... satisfaction_level       float64
    last_evaluation          float64
    number_project             int64
    average_montly_hours       int64
    time_spend_company         int64
    Work_accident              int64
    left                       int64
    promotion_last_5years      int64
    department                object
    salary                    object
    dtype: object
```



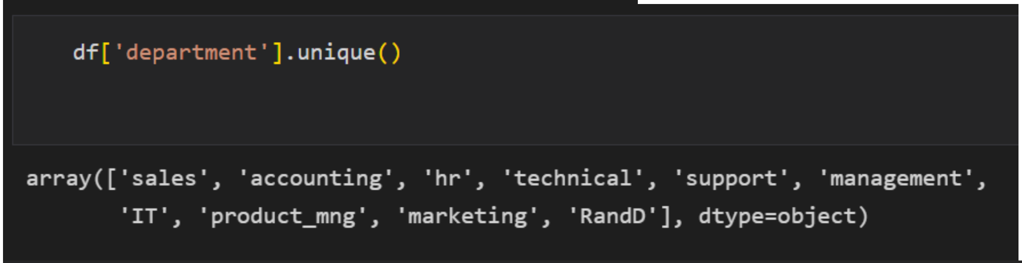
```
df.isnull().any()
```

```
satisfaction_level       False
last_evaluation          False
number_project           False
average_montly_hours     False
time_spend_company       False
Work_accident            False
left                     False
promotion_last_5years    False
department               False
salary                   False
dtype: bool
```

Our data is clean. There are no missing and null values.

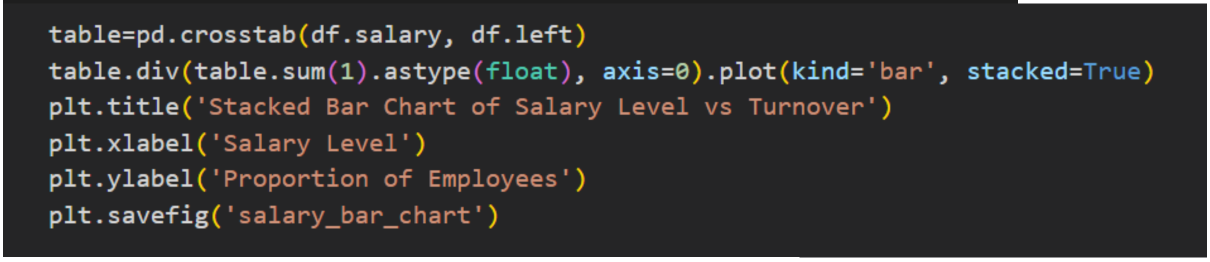
```
df.shape
```

```
(14999, 10)
```

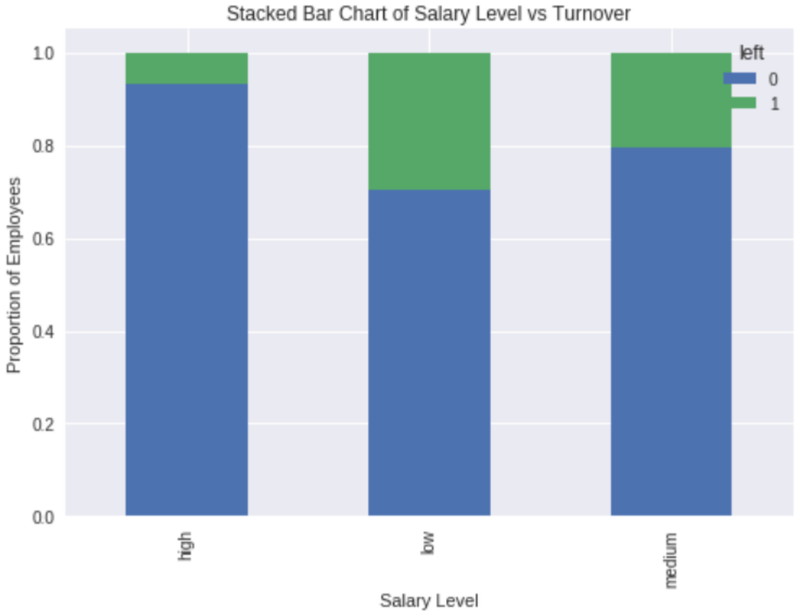


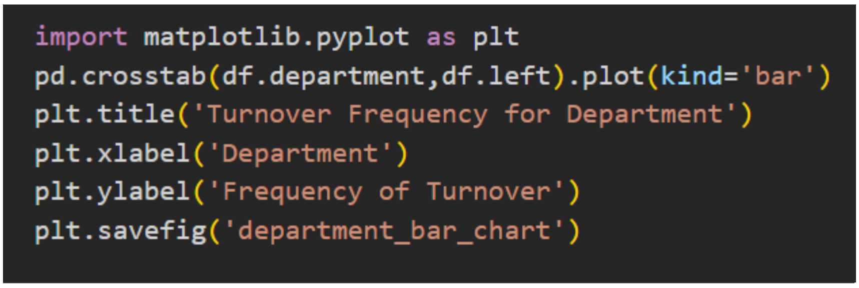
```
df['department'].unique()
```

```
array(['sales', 'accounting', 'hr', 'technical', 'support', 'management',
       'IT', 'product_mng', 'marketing', 'RandD'], dtype=object)
```

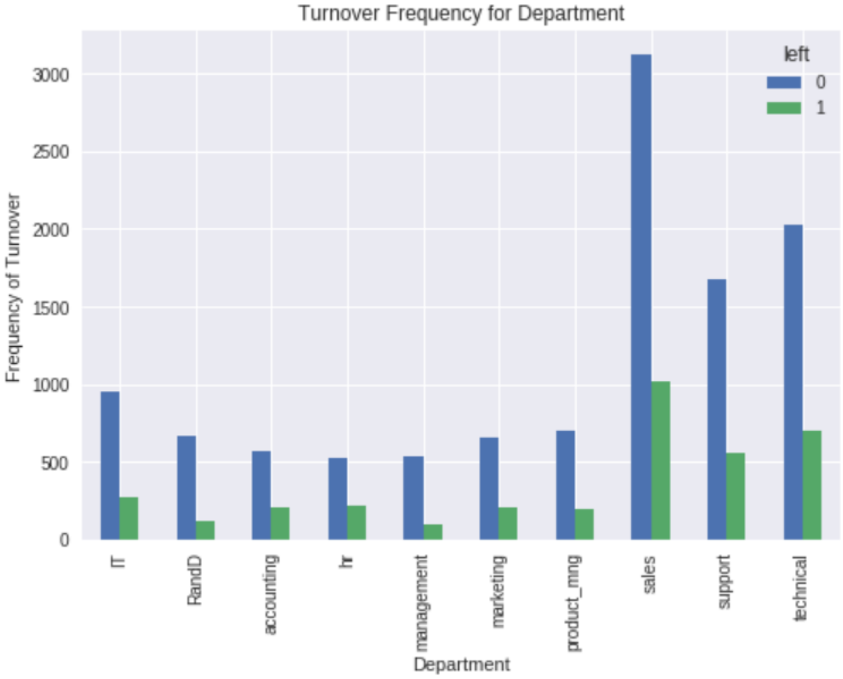


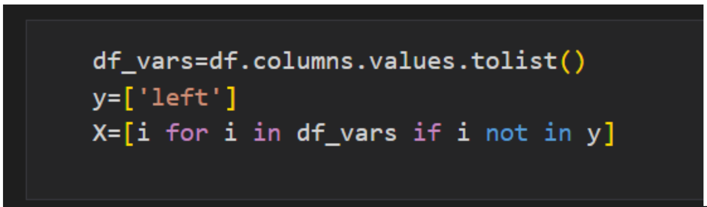
```
table=pd.crosstab(df.salary, df.left)
table.div(table.sum(1).astype(float), axis=0).plot(kind='bar', stacked=True)
plt.title('Stacked Bar Chart of Salary Level vs Turnover')
plt.xlabel('Salary Level')
plt.ylabel('Proportion of Employees')
plt.savefig('salary_bar_chart')
```



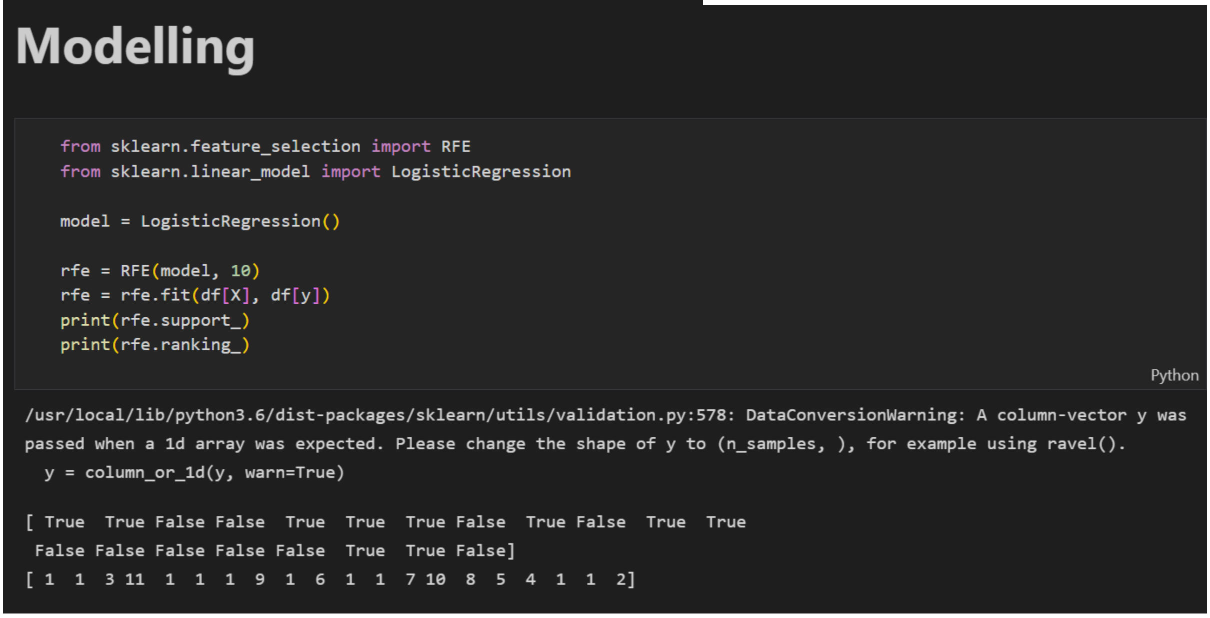


```
import matplotlib.pyplot as plt
pd.crosstab(df.department,df.left).plot(kind='bar')
plt.title('Turnover Frequency for Department')
plt.xlabel('Department')
plt.ylabel('Frequency of Turnover')
plt.savefig('department_bar_chart')
```





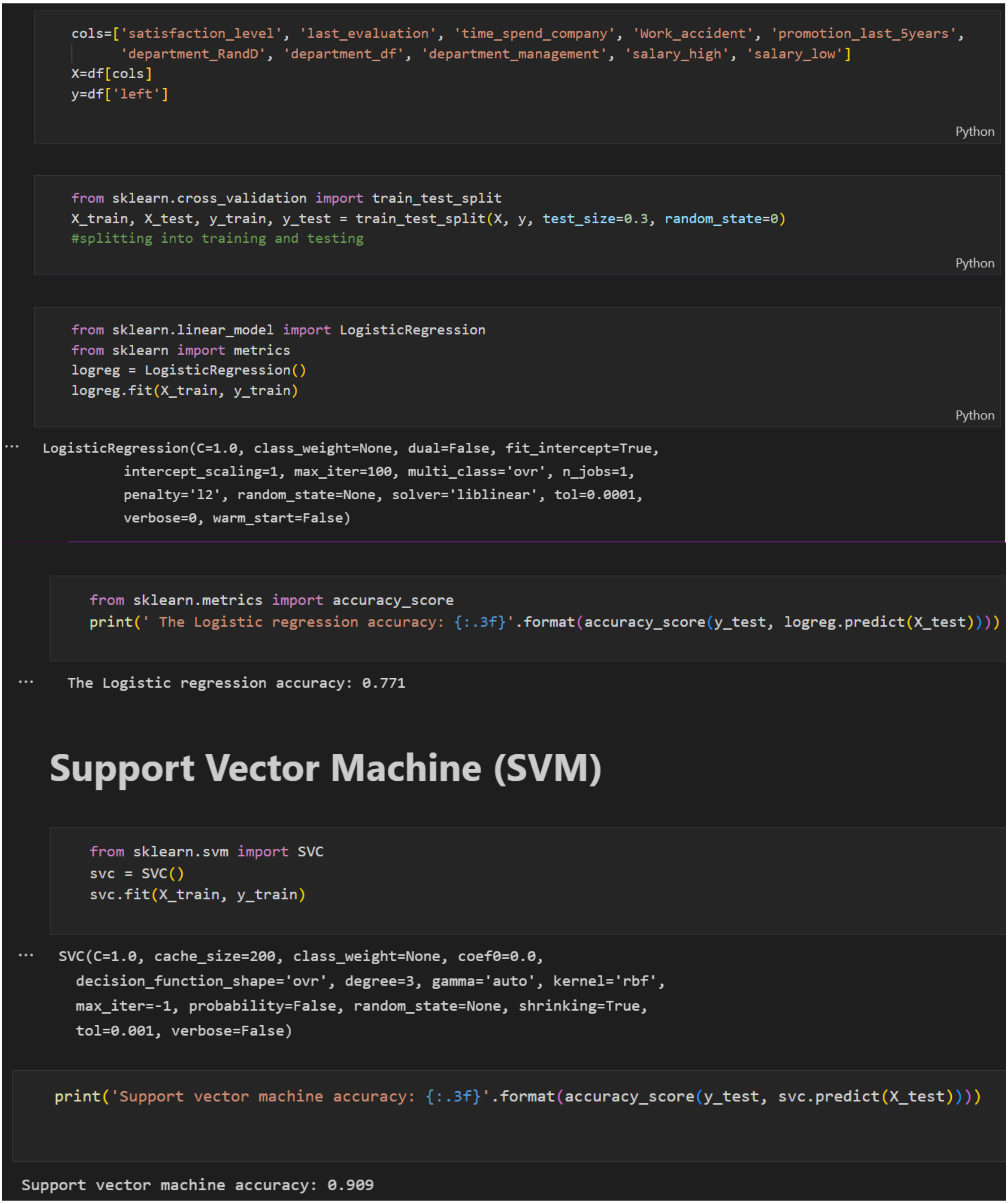
```
df_vars=df.columns.values.tolist()
y=['left']
X=[i for i in df_vars if i not in y]
```



Modelling

```
from sklearn.feature_selection import RFE
from sklearn.linear_model import LogisticRegression

model = LogisticRegression()

rfe = RFE(model, 10)
rfe = rfe.fit(df[X], df[y])
print(rfe.support_)
print(rfe.ranking_)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/utils/validation.py:578: DataConversionWarning: A column-vector y was
passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using ravel().
  y = column_or_1d(y, warn=True)

[ True  True False False  True  True  True False  True False  True  True
 False False False False False  True  True False]
[ 1  1  3 11  1  1  1  9  1  6  1  1  7 10  8  5  4  1  1  2]
```



```python
cols=['satisfaction_level', 'last_evaluation', 'time_spend_company', 'Work_accident', 'promotion_last_5years',
      'department_RandD', 'department_df', 'department_management', 'salary_high', 'salary_low']
X=df[cols]
y=df['left']
```

```python
from sklearn.cross_validation import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=0)
#splitting into training and testing
```

```python
from sklearn.linear_model import LogisticRegression
from sklearn import metrics
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='ovr', n_jobs=1,
          penalty='l2', random_state=None, solver='liblinear', tol=0.0001,
          verbose=0, warm_start=False)
```

```python
from sklearn.metrics import accuracy_score
print(' The Logistic regression accuracy: {:.3f}'.format(accuracy_score(y_test, logreg.predict(X_test))))
```

```
The Logistic regression accuracy: 0.771
```

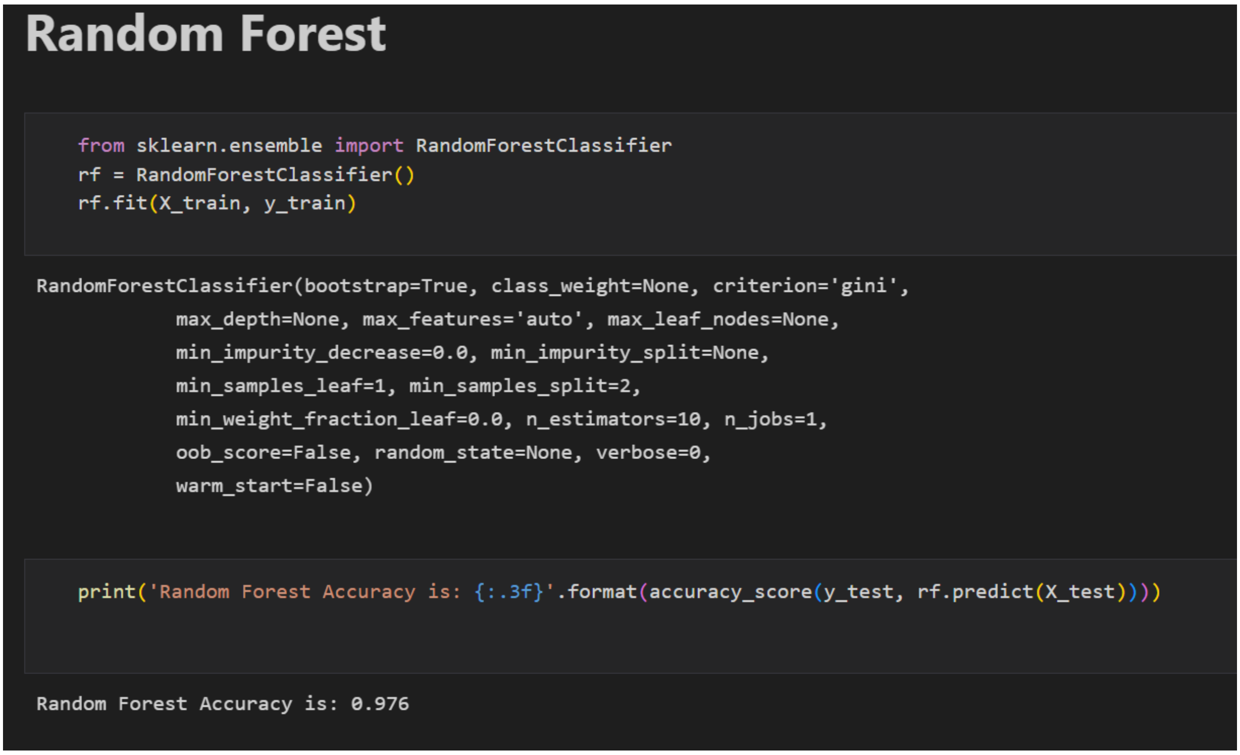
Support Vector Machine (SVM)

```python
from sklearn.svm import SVC
svc = SVC()
svc.fit(X_train, y_train)
```

```
SVC(C=1.0, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto', kernel='rbf',
  max_iter=-1, probability=False, random_state=None, shrinking=True,
  tol=0.001, verbose=False)
```

```python
print('Support vector machine accuracy: {:.3f}'.format(accuracy_score(y_test, svc.predict(X_test))))
```

```
Support vector machine accuracy: 0.909
```



Random Forest

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier()
rf.fit(X_train, y_train)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=None, max_features='auto', max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, n_estimators=10, n_jobs=1,
            oob_score=False, random_state=None, verbose=0,
            warm_start=False)
```

```
print('Random Forest Accuracy is: {:.3f}'.format(accuracy_score(y_test, rf.predict(X_test))))
```

```
Random Forest Accuracy is: 0.976
```