AHSANULLAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

Ahsanullah University of Science and Technology

Department of Computer Science and Engineering

Course No: CSE4108

Course Title: Artificial Intelligence Lab

Assignment No: 04

Submitted by

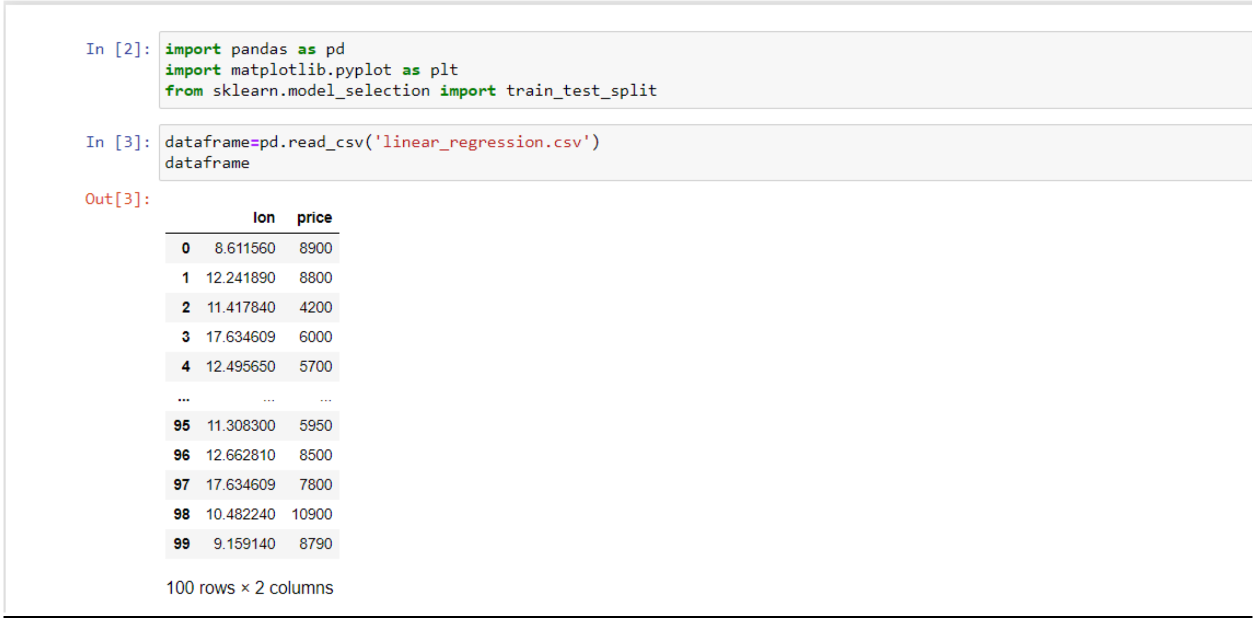
Name: Rizeya Rabbi Reyad

ID: 160104082

Lab Group: B2

3)Implement Linear Regression without using Scikit-learn.

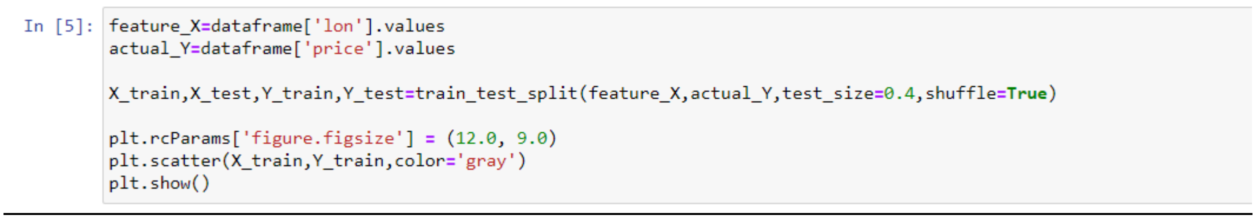
Python:



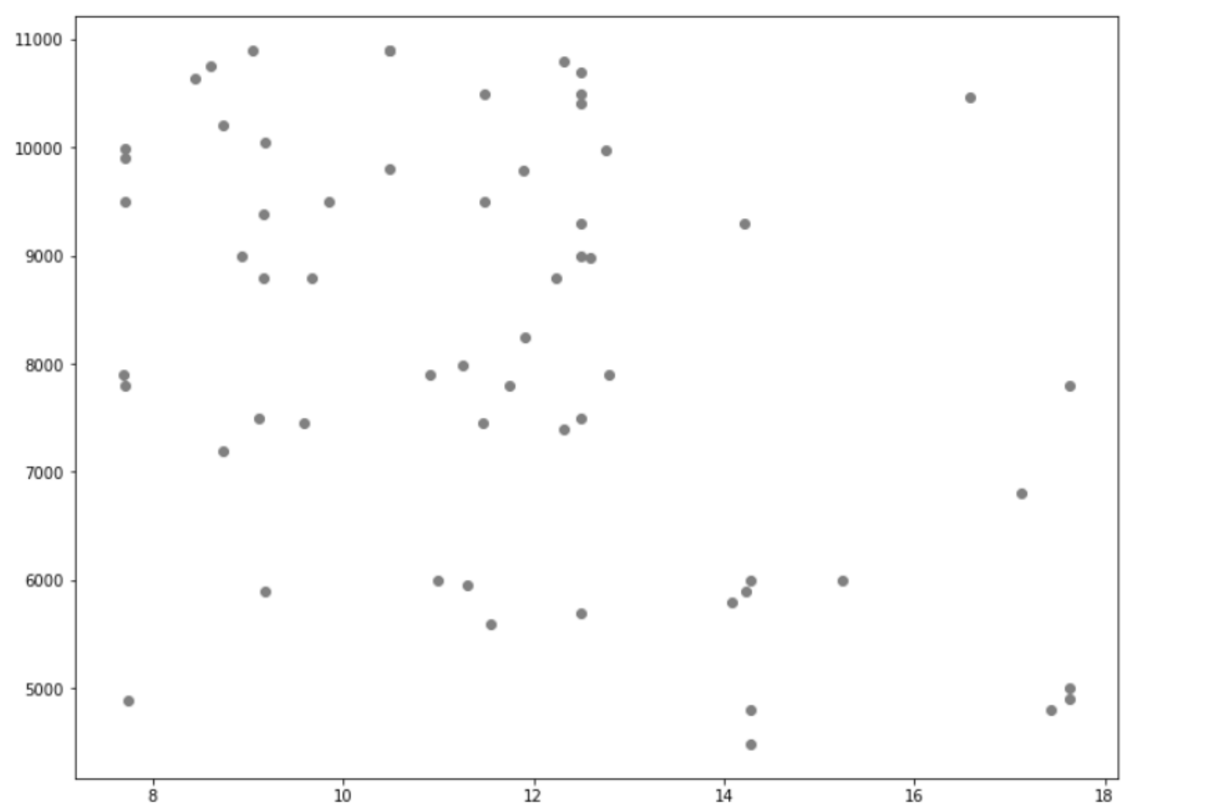
```
In [2]: import pandas as pd
        import matplotlib.pyplot as plt
        from sklearn.model_selection import train_test_split

In [3]: dataframe=pd.read_csv('linear_regression.csv')
        dataframe

Out[3]:
```

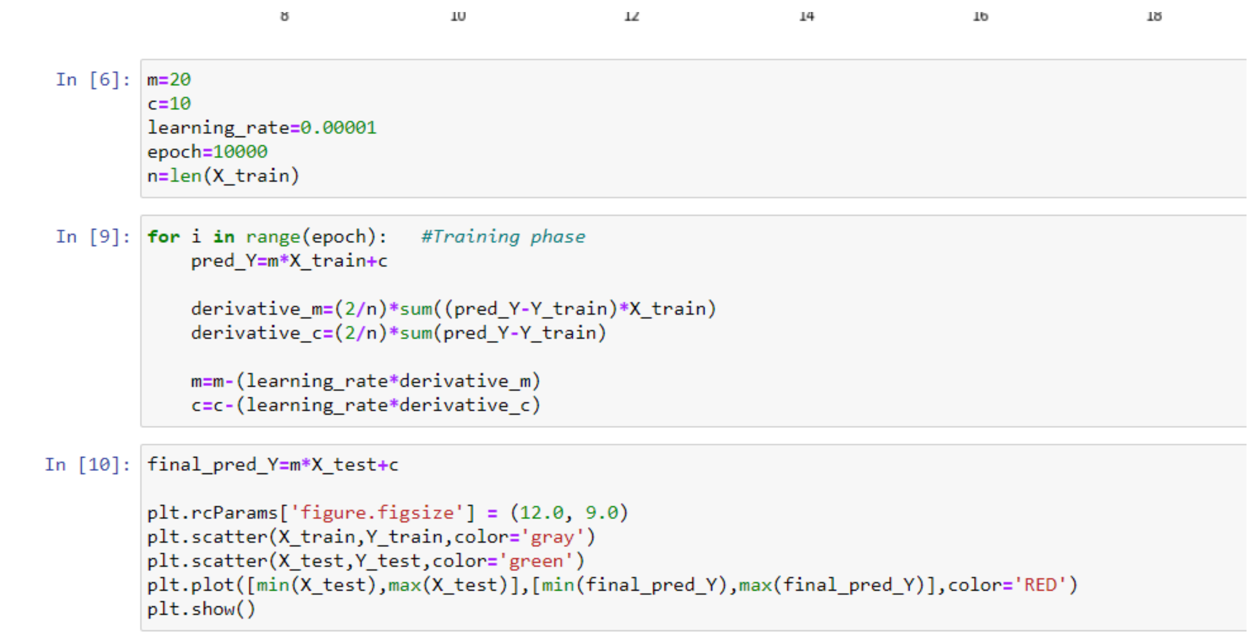
	lon	price
0	8.611560	8900
1	12.241890	8800
2	11.417840	4200
3	17.634609	6000
4	12.495650	5700
...	...	...
95	11.308300	5950
96	12.662810	8500
97	17.634609	7800
98	10.482240	10900
99	9.159140	8790

100 rows × 2 columns



```
In [5]: feature_X=dataframe['lon'].values
        actual_Y=dataframe['price'].values

        X_train,X_test,Y_train,Y_test=train_test_split(feature_X,actual_Y,test_size=0.4,shuffle=True)

        plt.rcParams['figure.figsize'] = (12.0, 9.0)
        plt.scatter(X_train,Y_train,color='gray')
        plt.show()
```

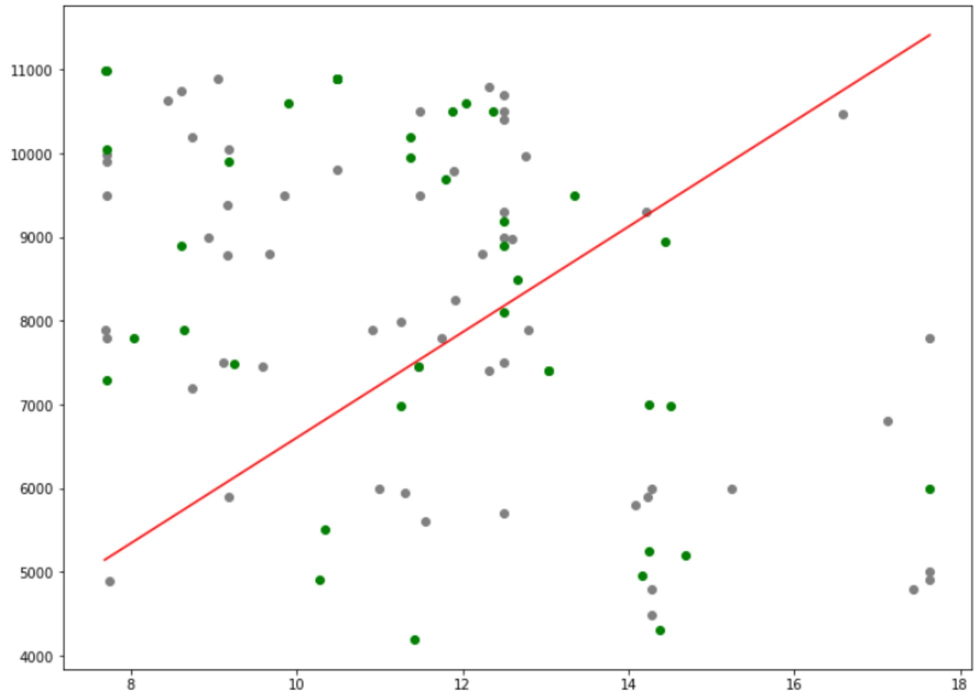




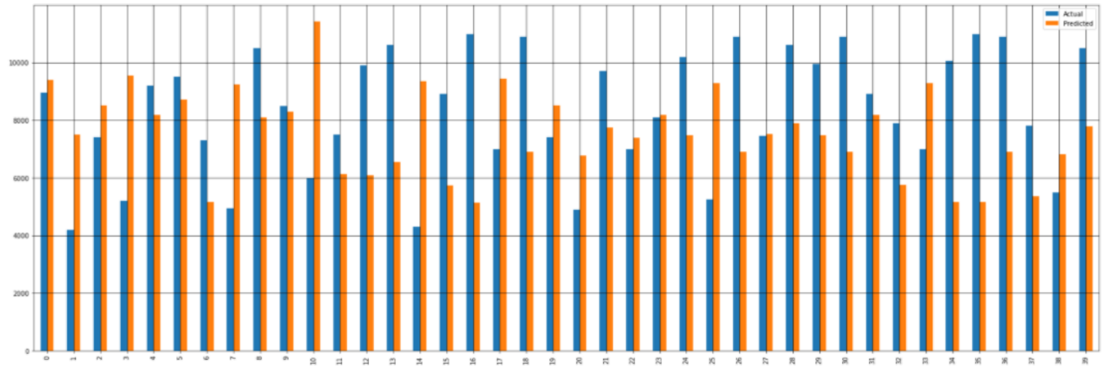
In [6]:
```
m=20
c=10
learning_rate=0.00001
epoch=10000
n=len(X_train)
```

In [9]:
```
for i in range(epoch):    #Training phase
    pred_Y=m*X_train+c

    derivative_m=(2/n)*sum((pred_Y-Y_train)*X_train)
    derivative_c=(2/n)*sum(pred_Y-Y_train)

    m=m-(learning_rate*derivative_m)
    c=c-(learning_rate*derivative_c)
```

In [10]:
```
final_pred_Y=m*X_test+c

plt.rcParams['figure.figsize'] = (12.0, 9.0)
plt.scatter(X_train,Y_train,color='gray')
plt.scatter(X_test,Y_test,color='green')
plt.plot([min(X_test),max(X_test)],[min(final_pred_Y),max(final_pred_Y)],color='RED')
plt.show()
```



In [13]:
```
plt.rcParams['figure.figsize'] = (30.0, 10.0)
dataframe=pd.DataFrame({'Actual': Y_test.flatten(), 'Predicted': final_pred_Y.flatten()})
dataframe.plot(kind='bar')
plt.grid(which='major',color='black')
plt.grid(which='minor',color='green')
plt.show()
```



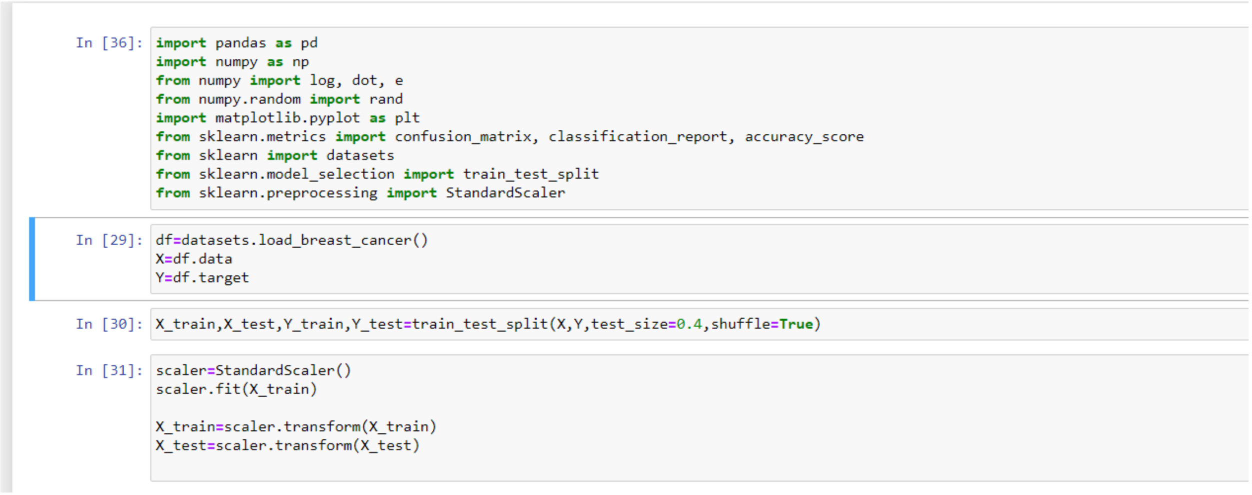
In [14]:
```
from sklearn import metrics
import numpy as np

print('Mean Absolute Error:   ',metrics.mean_absolute_error(Y_test,final_pred_Y))
print('Mean Squared Error:   ',metrics.mean_squared_error(Y_test,final_pred_Y))
print('Root Mean Squared Error:   ',np.sqrt(metrics.mean_squared_error(Y_test,final_pred_Y)))
```

```
Mean Absolute Error:    2676.669825299016
Mean Squared Error:    9826514.918216094
Root Mean Squared Error:    3134.7272478185555
```

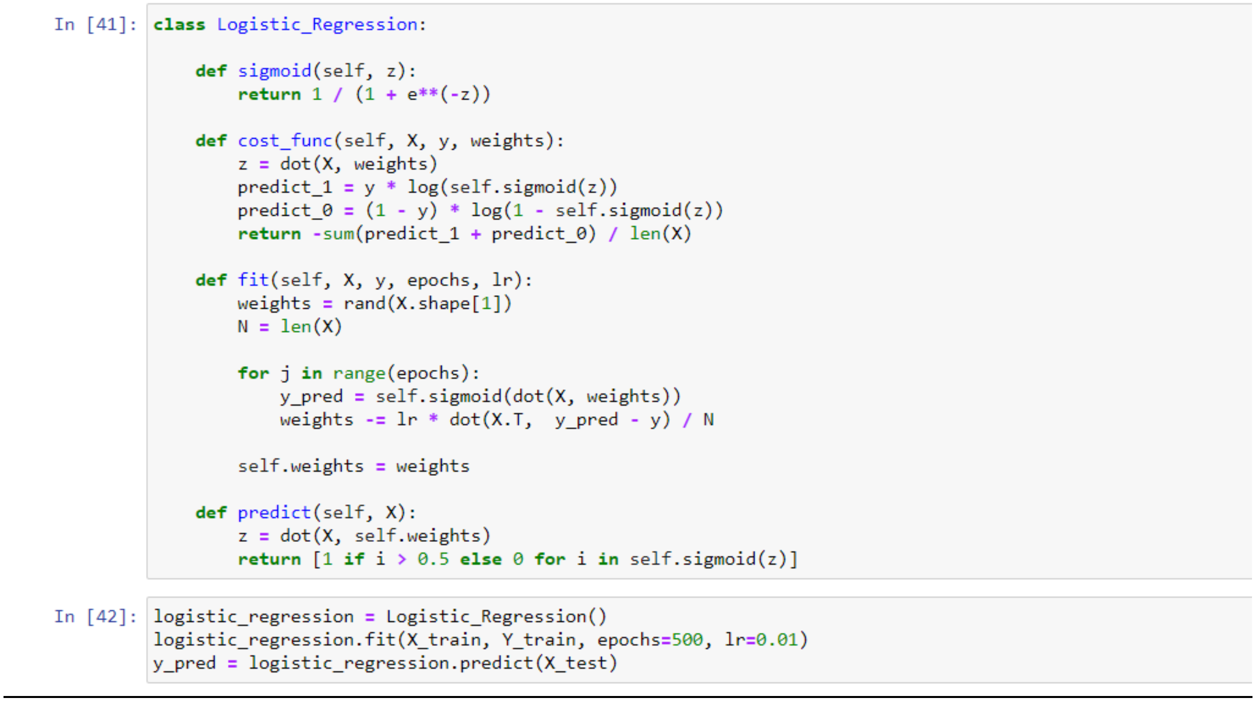
4)Implement Logistic Regression from scratch without using Scikit-learn. Run it against a dataset

Of choice (any dataset with over 1000 samples). Run the same algorithm with the help of Scikit-learn. Compare your implementation with Scikit-lean's one.

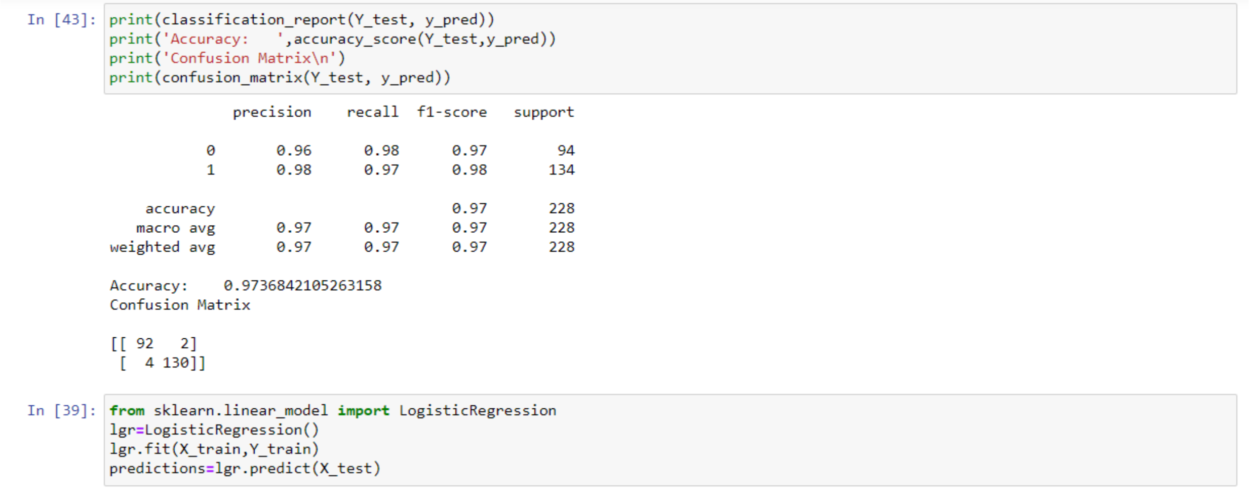
Python:



```
In [36]: import pandas as pd
         import numpy as np
         from numpy import log, dot, e
         from numpy.random import rand
         import matplotlib.pyplot as plt
         from sklearn.metrics import confusion_matrix, classification_report, accuracy_score
         from sklearn import datasets
         from sklearn.model_selection import train_test_split
         from sklearn.preprocessing import StandardScaler

In [29]: df=datasets.load_breast_cancer()
         X=df.data
         Y=df.target

In [30]: X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.4,shuffle=True)

In [31]: scaler=StandardScaler()
         scaler.fit(X_train)

         X_train=scaler.transform(X_train)
         X_test=scaler.transform(X_test)
```

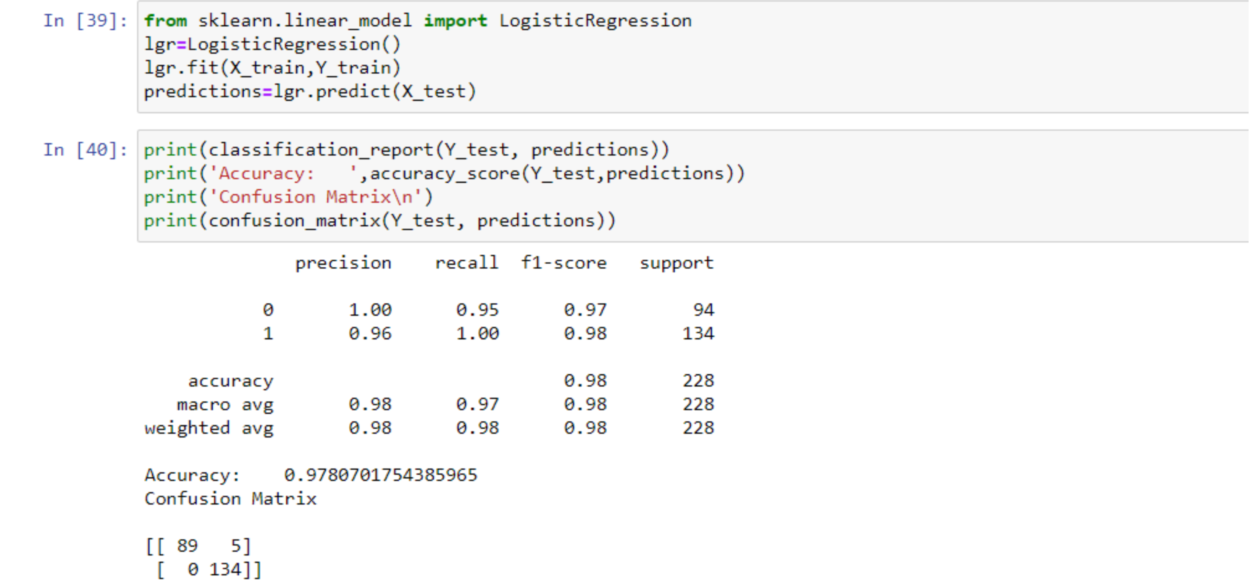


```
In [41]: class Logistic_Regression:

             def sigmoid(self, z):
                 return 1 / (1 + e**(-z))

             def cost_func(self, X, y, weights):
                 z = dot(X, weights)
                 predict_1 = y * log(self.sigmoid(z))
                 predict_0 = (1 - y) * log(1 - self.sigmoid(z))
                 return -sum(predict_1 + predict_0) / len(X)

             def fit(self, X, y, epochs, lr):
                 weights = rand(X.shape[1])
                 N = len(X)

                 for j in range(epochs):
                     y_pred = self.sigmoid(dot(X, weights))
                     weights -= lr * dot(X.T,  y_pred - y) / N

                 self.weights = weights

             def predict(self, X):
                 z = dot(X, self.weights)
                 return [1 if i > 0.5 else 0 for i in self.sigmoid(z)]

In [42]: logistic_regression = Logistic_Regression()
         logistic_regression.fit(X_train, Y_train, epochs=500, lr=0.01)
         y_pred = logistic_regression.predict(X_test)
```



```
In [43]: print(classification_report(Y_test, y_pred))
         print('Accuracy:   ',accuracy_score(Y_test,y_pred))
         print('Confusion Matrix\n')
         print(confusion_matrix(Y_test, y_pred))

              precision    recall  f1-score   support

           0       0.96      0.98      0.97        94
           1       0.98      0.97      0.98       134

    accuracy                           0.97       228
   macro avg       0.97      0.97      0.97       228
weighted avg       0.97      0.97      0.97       228

Accuracy:    0.9736842105263158
Confusion Matrix

[[ 92   2]
 [  4 130]]
```

```
In [39]: from sklearn.linear_model import LogisticRegression
         lgr=LogisticRegression()
         lgr.fit(X_train,Y_train)
         predictions=lgr.predict(X_test)
```



```
In [39]: from sklearn.linear_model import LogisticRegression
         lgr=LogisticRegression()
         lgr.fit(X_train,Y_train)
         predictions=lgr.predict(X_test)
```

```
In [40]: print(classification_report(Y_test, predictions))
         print('Accuracy:   ',accuracy_score(Y_test,predictions))
         print('Confusion Matrix\n')
         print(confusion_matrix(Y_test, predictions))

              precision    recall  f1-score   support

           0       1.00      0.95      0.97        94
           1       0.96      1.00      0.98       134

    accuracy                           0.98       228
   macro avg       0.98      0.97      0.98       228
weighted avg       0.98      0.98      0.98       228

Accuracy:    0.9780701754385965
Confusion Matrix

[[ 89   5]
 [  0 134]]
```

From above ,we get better accuracy result by using Scikit-learn in linear regression. With using Scikit-learn we get accuracy .97807017... and without using Scikit-learn we get accuracy .97368421... .

5)Make a dataset by yourself which should have enough samples and attributes and write documentation of it. Do classification or regression on it. If you want to do a classification task, implement at least five models. If you want to do regression, similarly at least five models need

To be implemented. For each model get at least three performance metric scores.Implemetation of cross validation is a must.(Name your dataset as Dataset_StudentId)

In [1]:

```
import pandas as pd
import numpy as np
from numpy.random import seed
from numpy.random import rand
from numpy.random import randint
```

In [2]:

```
# User ID
ID=[]*300
for i in range(300):
    ID.append(i+1)
ID=np.array(ID)

#Year
seed(5)
year=randint(2010,2020,300)


#User Gender
seed(10)
Gender=randint(0,2,300)

Gender = Gender.astype('str')
for i in range(300):
    if Gender[i]=='0':
        Gender[i]='Male'
    else:
        Gender[i]='Female'
Gender

#User Age
seed(5)
Age=randint(18,25,300)

Age

#User Clicked On Ad
seed(9)
Current_Student=randint(0,2,300)

Current_Student =Current_Student.astype('str')
for i in range(300):
    if Current_Student[i]=='0':
        Current_Student[i]='No'
    else:
        Current_Student[i]='Yes'
Current_Student
```

```
Out[2]: array(['No', 'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes',
               'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'Yes',
               'Yes', 'No', 'No', 'No', 'No', 'Yes', 'No', 'No', 'No', 'Yes',
               'No', 'No', 'No', 'No', 'No', 'No', 'No', 'No', 'Yes', 'No', 'Yes',
               'Yes', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes',
               'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No',
               'Yes', 'No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No',
               'No', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No',
               'No', 'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No',
               'Yes', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'No',
               'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No',
               'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'No', 'No', 'No',
               'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes',
               'No', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No',
               'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No',
               'No', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'No',
               'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No',
               'Yes', 'Yes', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'No',
               'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'Yes', 'No',
               'No', 'No', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'Yes',
               'No', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes',
               'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'Yes', 'No',
               'No', 'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'No', 'Yes',
               'Yes', 'Yes', 'No', 'No', 'No', 'No', 'Yes', 'No', 'Yes', 'Yes',
               'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes',
               'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes',
               'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes', 'Yes', 'No',
               'No', 'Yes', 'Yes', 'Yes', 'No', 'No', 'No', 'Yes', 'No', 'No',
               'Yes', 'No', 'No', 'Yes', 'No', 'Yes', 'No', 'No', 'No', 'Yes',
               'No', 'Yes', 'No', 'Yes', 'Yes', 'No', 'No', 'Yes', 'Yes'],
              dtype='<U11')
```

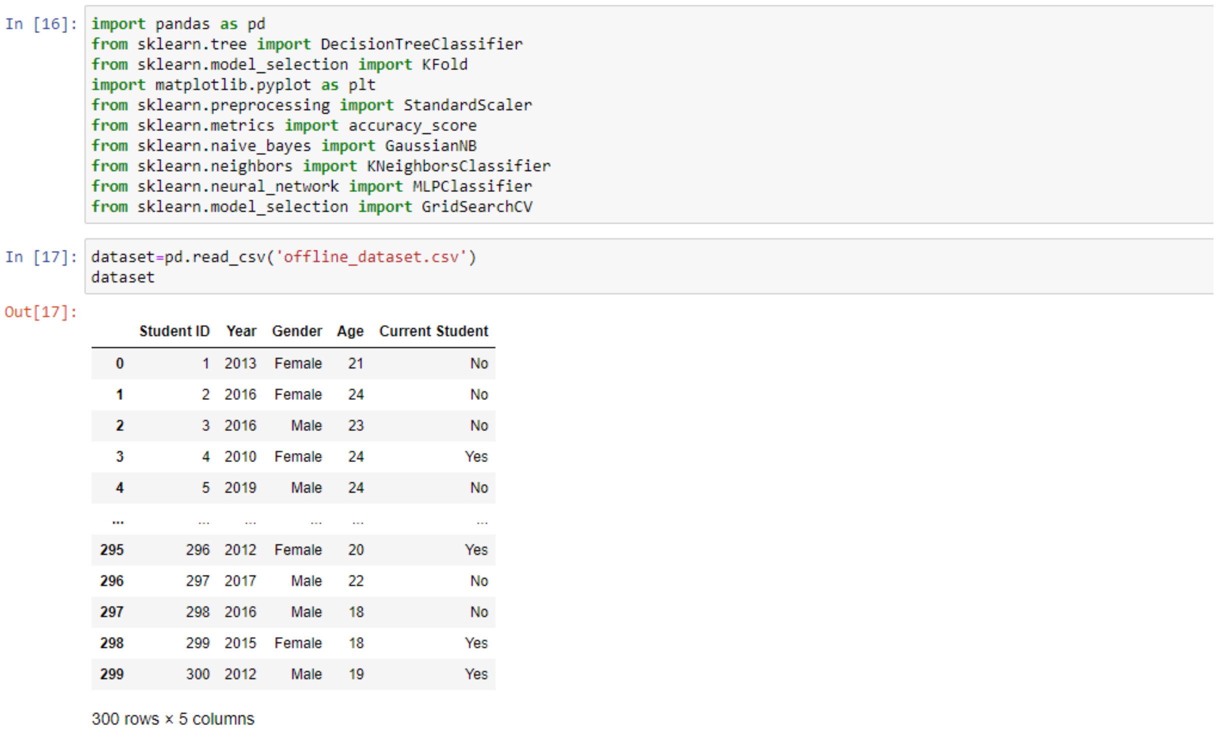
```
In [3]: df=pd.DataFrame({'Student ID':ID,'Year':year,'Gender':Gender,'Age':Age,'Current Student':Current_Student})
        df
```

Out[3]:

	Student ID	Year	Gender	Age	Current Student
0	1	2013	Female	21	No
1	2	2016	Female	24	No
2	3	2016	Male	23	No
3	4	2010	Female	24	Yes
4	5	2019	Male	24	No
...	...	...	...	...	...
295	296	2012	Female	20	Yes
296	297	2017	Male	22	No
297	298	2016	Male	18	No
298	299	2015	Female	18	Yes
299	300	2012	Male	19	Yes

300 rows × 5 columns

```
In [4]: df.to_csv(r'C:\Users\rakesh\offline_dataset.csv',index=False,header=True)
```



In [16]:

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import KFold
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import accuracy_score
from sklearn.naive_bayes import GaussianNB
from sklearn.neighbors import KNeighborsClassifier
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
```

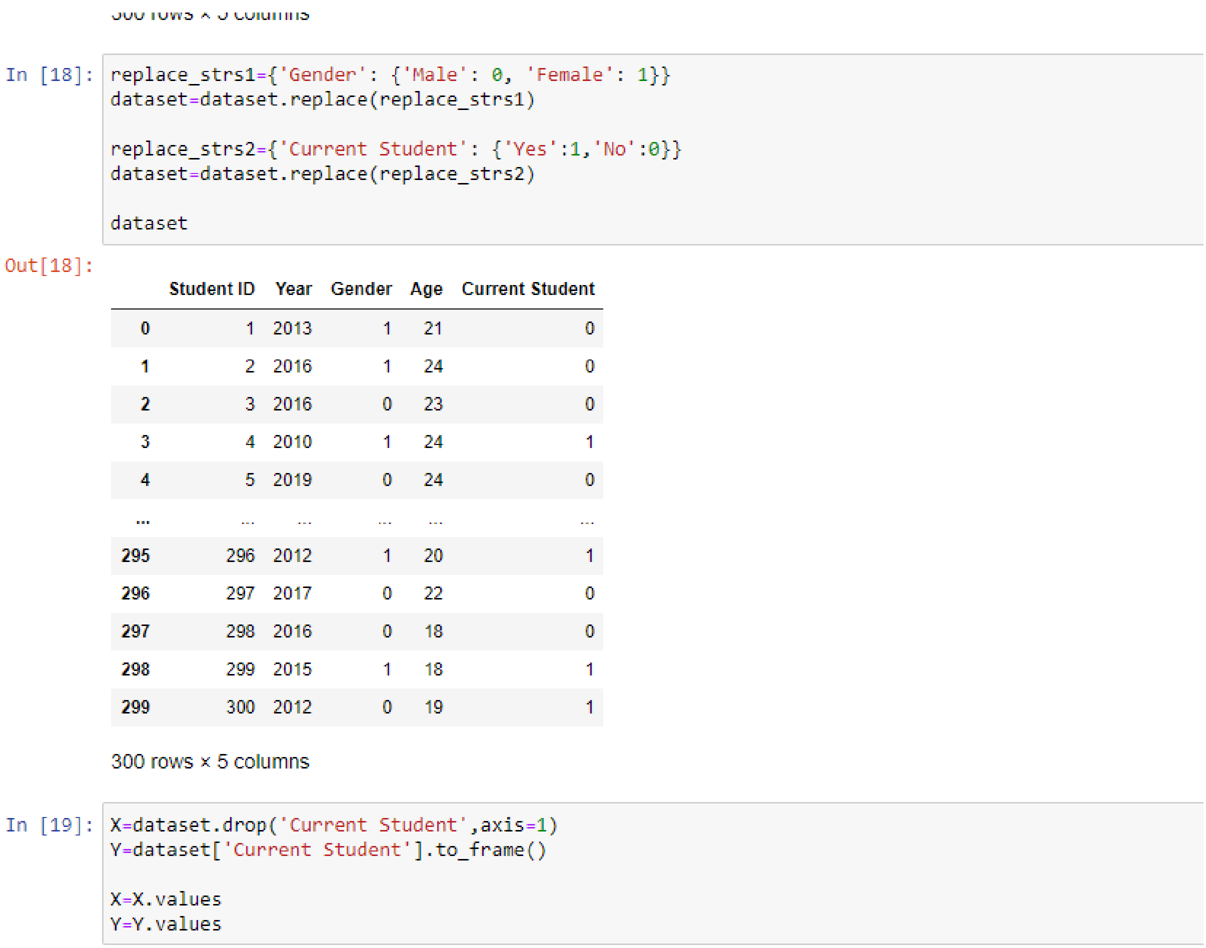
In [17]:

```
dataset=pd.read_csv('offline_dataset.csv')
dataset
```

Out[17]:

	Student ID	Year	Gender	Age	Current Student
0	1	2013	Female	21	No
1	2	2016	Female	24	No
2	3	2016	Male	23	No
3	4	2010	Female	24	Yes
4	5	2019	Male	24	No
...	...	...	...	...	...
295	296	2012	Female	20	Yes
296	297	2017	Male	22	No
297	298	2016	Male	18	No
298	299	2015	Female	18	Yes
299	300	2012	Male	19	Yes

300 rows × 5 columns



300 rows × 5 columns

In [18]:

```
replace_strs1={'Gender': {'Male': 0, 'Female': 1}}
dataset=dataset.replace(replace_strs1)

replace_strs2={'Current Student': {'Yes':1,'No':0}}
dataset=dataset.replace(replace_strs2)

dataset
```

Out[18]:

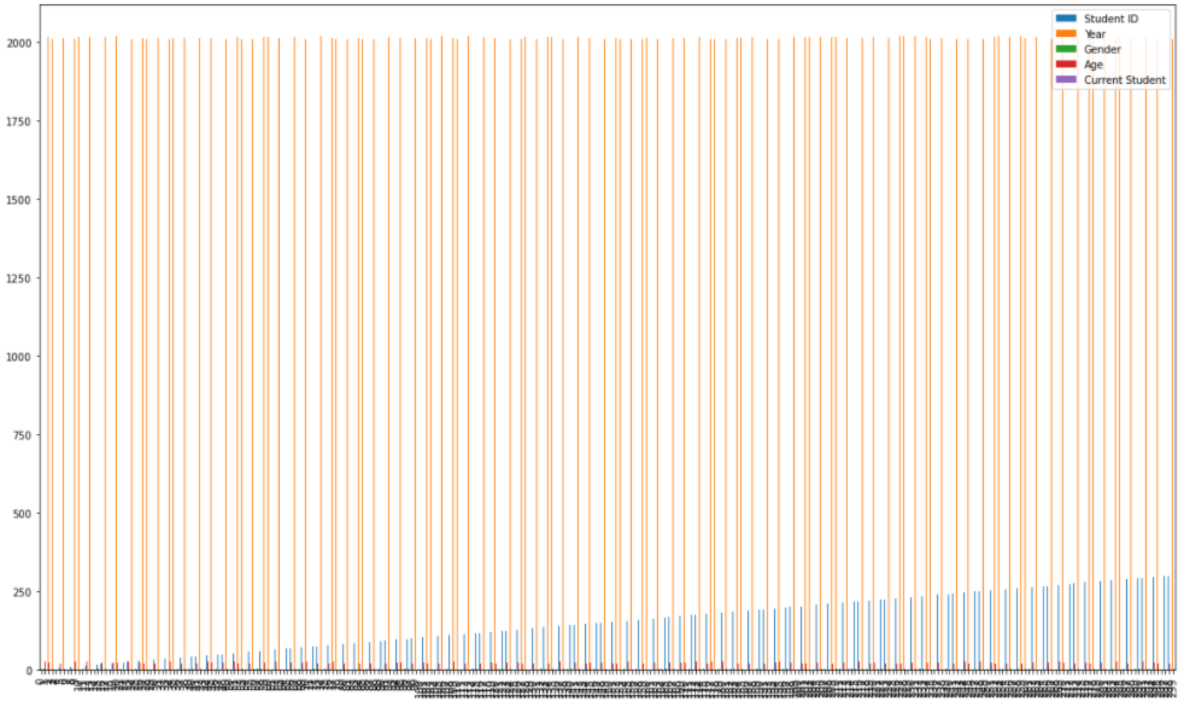
	Student ID	Year	Gender	Age	Current Student
0	1	2013	1	21	0
1	2	2016	1	24	0
2	3	2016	0	23	0
3	4	2010	1	24	1
4	5	2019	0	24	0
...	...	...	...	...	...
295	296	2012	1	20	1
296	297	2017	0	22	0
297	298	2016	0	18	0
298	299	2015	1	18	1
299	300	2012	0	19	1

300 rows × 5 columns

In [19]:

```
X=dataset.drop('Current Student',axis=1)
Y=dataset['Current Student'].to_frame()

X=X.values
Y=Y.values
```

In [6]:
```
plt.rcParams['figure.figsize'] = (20.0, 12.0)
dataset.plot(kind='bar')
plt.show()
```



Decision Tree Classifier

In [20]:
```
kf=KFold(n_splits=30)


sum=0

for train_index,test_index in kf.split(X):
    x_train,x_test=X[train_index],X[test_index]
    y_train,y_test=Y[train_index],Y[test_index]

    scaler=StandardScaler()
    scaler.fit(x_train)

    x_train=scaler.transform(x_train)
    x_test=scaler.transform(x_test)

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

    predictions=model.predict(x_test)
    acc=accuracy_score(y_test,predictions)

    sum+=acc

print('Accuracy_score:   ',sum/30)
```

```
Accuracy_score:    0.49
```

Naive Bayes Classifier

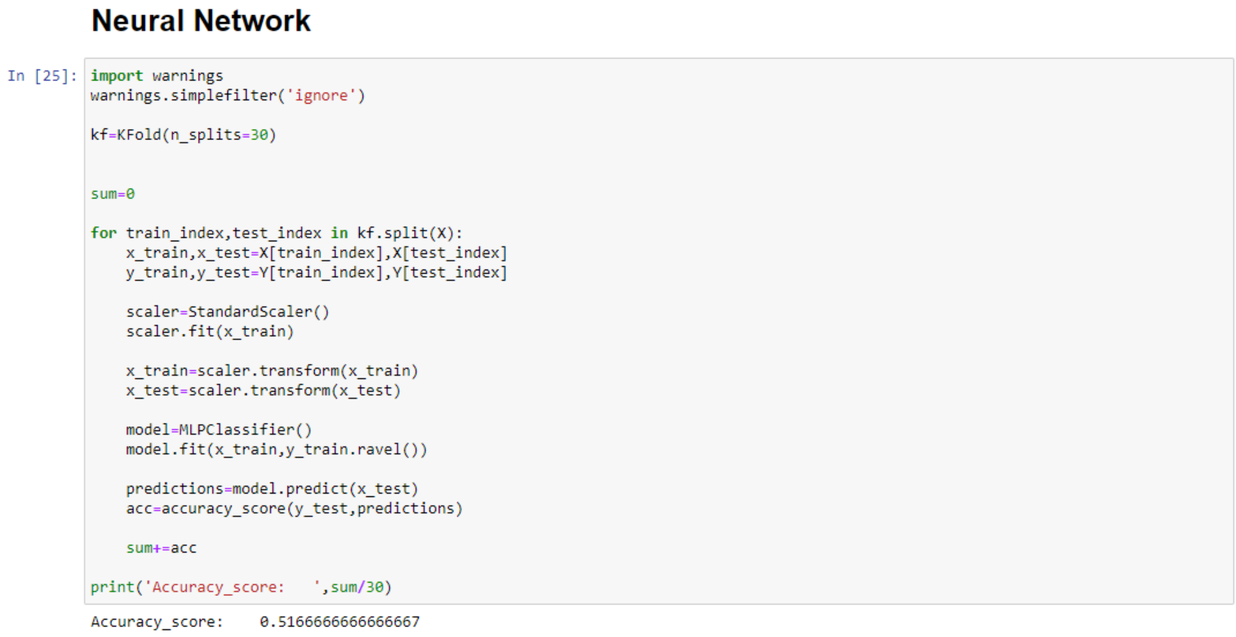
```
In [21]: kf=KFold(n_splits=30)


         sum=0

         for train_index,test_index in kf.split(X):
             x_train,x_test=X[train_index],X[test_index]
             y_train,y_test=Y[train_index],Y[test_index]

             scaler=StandardScaler()
             scaler.fit(x_train)

             x_train=scaler.transform(x_train)
             x_test=scaler.transform(x_test)

             model=GaussianNB()
             model.fit(x_train,y_train.ravel())

             predictions=model.predict(x_test)
             acc=accuracy_score(y_test,predictions)

             sum+=acc

         print('Accuracy_score:   ',sum/30)
```

```
Accuracy_score:    0.5333333333333333
```

KNN

```
In [22]: kf=KFold(n_splits=30)


         sum=0

         for train_index,test_index in kf.split(X):
             x_train,x_test=X[train_index],X[test_index]
             y_train,y_test=Y[train_index],Y[test_index]

             scaler=StandardScaler()
             scaler.fit(x_train)

             x_train=scaler.transform(x_train)
             x_test=scaler.transform(x_test)

             model=KNeighborsClassifier()
             model.fit(x_train,y_train.ravel())

             predictions=model.predict(x_test)
             acc=accuracy_score(y_test,predictions)

             sum+=acc

         print('Accuracy_score:   ',sum/30)
```

```
Accuracy_score:    0.5233333333333333
```



Neural Network

In [25]:

```
import warnings
warnings.simplefilter('ignore')

kf=KFold(n_splits=30)


sum=0

for train_index,test_index in kf.split(X):
    x_train,x_test=X[train_index],X[test_index]
    y_train,y_test=Y[train_index],Y[test_index]

    scaler=StandardScaler()
    scaler.fit(x_train)

    x_train=scaler.transform(x_train)
    x_test=scaler.transform(x_test)

    model=MLPClassifier()
    model.fit(x_train,y_train.ravel())

    predictions=model.predict(x_test)
    acc=accuracy_score(y_test,predictions)

    sum+=acc

print('Accuracy_score:   ',sum/30)
```

```
Accuracy_score:    0.5166666666666667
```

Explanation:

From above models we predicted different accuracy rate for different models. We got a slightly higher accuracy rate by using Naïve Bayes Classifier than other models. We also used these models to know in which model the dataset performs better. From Naïve Bayes Classifier we got accuracy score .533333333.