Q6. Take a regression dataset from Kaggle and implement linear regression.

AIM
To implement linear regression model on housing data from Kaggle.

SOFTWARE USED
Jupyter Platform - Python Programming Language

PROGRAM CODE

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

data = pd.read_csv("kc_house_data.csv")

data.head()
data.describe()
train1 = data.drop(['id', 'price'],axis=1)

train1.head()

from sklearn.linear_model import LinearRegression

reg = LinearRegression()

labels = data['price']
conv_dates = [1 if values == 2014 else 0 for values in data.date ]
data['date'] = conv_dates
train1 = data.drop(['id', 'price'],axis=1)

from sklearn.model_selection import train_test_split

x_train , x_test , y_train , y_test = train_test_split(train1 , labels , test_size = 0.10,random_state =2)

reg.fit(x_train,y_train)

reg.score(x_test,y_test)
```

OUTPUT

jupyter Linear_Regression_housesales Last Checkpoint: a minute ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

In [35]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import mpl_toolkits
%matplotlib inline
```

In [36]:
```
data = pd.read_csv("kc_house_data.csv")
```

In [37]:
```
data.head()
```

Out[37]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	...	grade	sqf
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1.0	0	0	...	7	
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2.0	0	0	...	7	
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1.0	0	0	...	6	
3	2487200875	20141209T000000	604000.0	4	3.00	1960	5000	1.0	0	0	...	7	
4	1954400510	20150218T000000	510000.0	3	2.00	1680	8080	1.0	0	0	...	8	

5 rows × 21 columns

In [38]:
```
data.describe()
```

Out[38]:

id price bedrooms bathrooms sqft_living sqft_lot floors waterfront

jupyter Linear_Regression_housesales Last Checkpoint: a minute ago (autosaved) Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

In [38]:
```
data.describe()
```

Out[38]:

	id	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	
count	2.161300e+04	2.161300e+04	21613.000000	21613.000000	21613.000000	2.161300e+04	21613.000000	21613.000000	2161
mean	4.580302e+09	5.400881e+05	3.370842	2.114757	2079.899736	1.510697e+04	1.494309	0.007542	
std	2.876566e+09	3.671272e+05	0.930062	0.770163	918.440897	4.142051e+04	0.539989	0.086517	
min	1.000102e+06	7.500000e+04	0.000000	0.000000	290.000000	5.200000e+02	1.000000	0.000000	
25%	2.123049e+09	3.219500e+05	3.000000	1.750000	1427.000000	5.040000e+03	1.000000	0.000000	
50%	3.904930e+09	4.500000e+05	3.000000	2.250000	1910.000000	7.618000e+03	1.500000	0.000000	
75%	7.308900e+09	6.450000e+05	4.000000	2.500000	2550.000000	1.068800e+04	2.000000	0.000000	
max	9.900000e+09	7.700000e+06	33.000000	8.000000	13540.000000	1.651359e+06	3.500000	1.000000	

In [47]:
```
train1 = data.drop(['id', 'price'],axis=1)
```

In [48]:
```
train1.head()
```

Out[48]:

	date	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	condition	grade	sqft_above	sqft_ba
0	20141013T000000	3	1.00	1180	5650	1.0	0	0	3	7	1180	
1	20141209T000000	3	2.25	2570	7242	2.0	0	0	3	7	2170	
2	20150225T000000	2	1.00	770	10000	1.0	0	0	3	6	770	
3	20141209T000000	4	3.00	1960	5000	1.0	0	0	5	7	1050	
4	20150218T000000	3	2.00	1680	8080	1.0	0	0	3	8	1680	

jupyter Linear_Regression_housesales Last Checkpoint: a minute ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help | Not Trusted | Python 3 (ipykernel)

Run | Code

3	20141209T000000	4	3.00	1960	5000	1.0	0	0	5	7	1050
4	20150218T000000	3	2.00	1680	8080	1.0	0	0	3	8	1680

In [53]:
```
from sklearn.linear_model import LinearRegression
```

In [54]:
```
reg = LinearRegression()
```

In [55]:
```
labels = data['price']
conv_dates = [1 if values == 2014 else 0 for values in data.date ]
data['date'] = conv_dates
train1 = data.drop(['id', 'price'],axis=1)
```

In [56]:
```
from sklearn.model_selection import train_test_split
```

In [57]:
```
x_train , x_test , y_train , y_test = train_test_split(train1 , labels , test_size = 0.10,random_state
```

In [58]:
```
reg.fit(x_train,y_train)
```

Out[58]:
```
LinearRegression()
```

In [68]:
```
reg.score(x_test,y_test)
```

Out[68]:
```
0.7320342760357641
```

DISCUSSION and CONCLUSION

The linear regression model has been applied and executed successfully on housing data.

Q7. Take a classification dataset from Kaggle and classify the data into output classes. Also evaluate the classifier efficiency using various evaluation measures.

AIM

Implement a classification problem on a Kaggle dataset using logistic regression.

SOFTWARE USED

Jupyter Platform - Python Programming Language

PROGRAM CODE

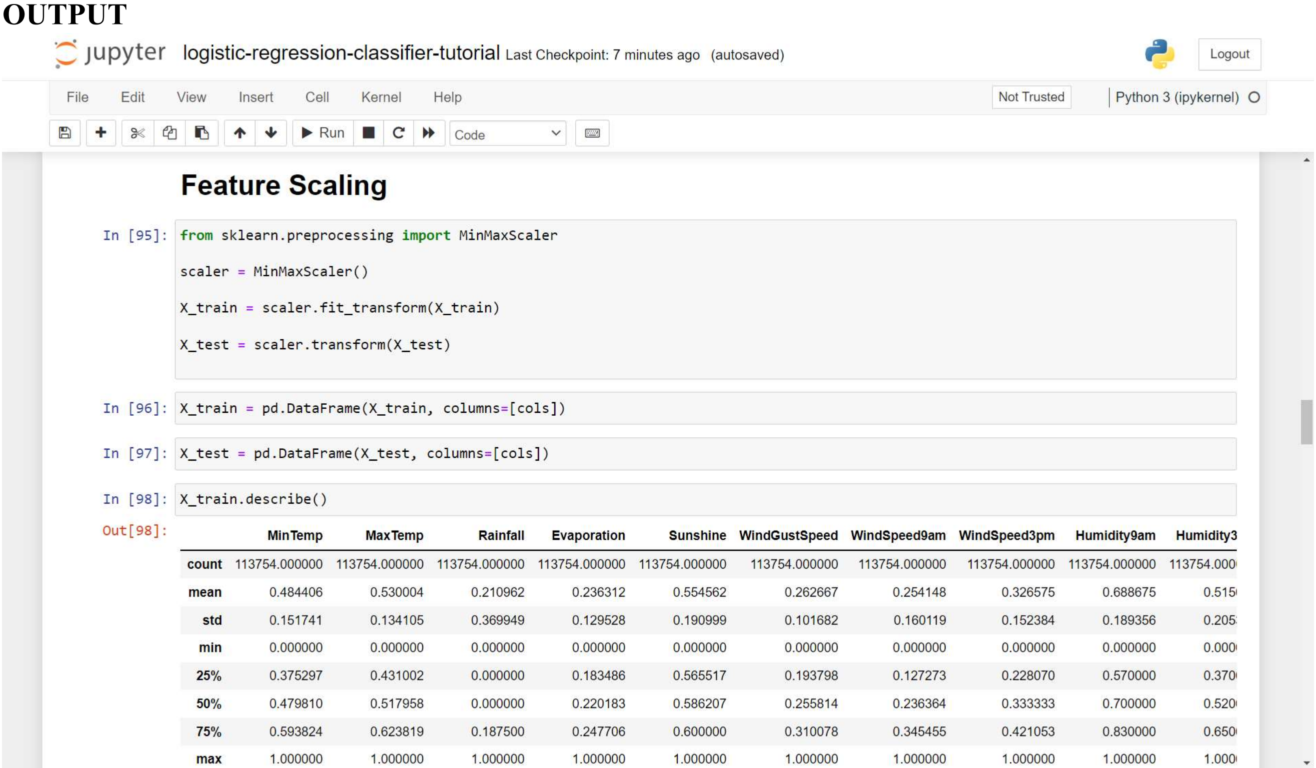
```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
    for filename in filenames:
        print(os.path.join(dirname, filename))

data = '/kaggle/input/weather-dataset-rattle-package/weatherAUS.csv'
```

```
df = pd.read_csv(data)

# preview the dataset

df.head()

col_names = df.columns

col_names

X = df.drop(['RainTomorrow'], axis=1)

y = df['RainTomorrow']

# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)

# check the shape of X_train and X_test

X_train.shape, X_test.shape
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)

X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])

X_train.describe()

# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


# fit the model
logreg.fit(X_train, y_train)


y_pred_test = logreg.predict(X_test)

y_pred_test

# probability of getting output as 0 - no rain
```

```
logreg.predict_proba(X_test)[:,0]

# probability of getting output as 1 - rain

logreg.predict_proba(X_test)[:,1]

from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))

y_pred_train = logreg.predict(X_train)

y_pred_train

print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))

# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])


from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))


# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))

# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))

# print precision score
```

```
precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))


recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))



true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))


false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))

specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

OUTPUT



	MinTemp	MaxTemp	Rainfall	Evaporation	Sunshine	WindGustSpeed	WindSpeed9am	WindSpeed3pm	Humidity9am	Humidity3
count	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000000	113754.000
mean	0.484406	0.530004	0.210962	0.236312	0.554562	0.262667	0.254148	0.326575	0.688675	0.515
std	0.151741	0.134105	0.369949	0.129528	0.190999	0.101682	0.160119	0.152384	0.189356	0.205
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.375297	0.431002	0.000000	0.183486	0.565517	0.193798	0.127273	0.228070	0.570000	0.370
50%	0.479810	0.517958	0.000000	0.220183	0.586207	0.255814	0.236364	0.333333	0.700000	0.520
75%	0.593824	0.623819	0.187500	0.247706	0.600000	0.310078	0.345455	0.421053	0.830000	0.650
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

50%	0.479810	0.517958	0.000000	0.220183	0.586207	0.255814	0.236364	0.333333	0.700000	0.520
75%	0.593824	0.623819	0.187500	0.247706	0.600000	0.310078	0.345455	0.421053	0.830000	0.650
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 118 columns

```
We now have `X_train` dataset ready to be fed into the Logistic Regression classifier. I will do it as follows.
```

Model training

In [99]:

```
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression


# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)


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

Out[99]:

```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=0, solver='liblinear', tol=0.0001, verbose=0,
                   warm_start=False)
```


random_state=0, solver='liblinear', tol=0.0001, verbose=0,
warm_start=False)

Predict results

In [100]:

```
y_pred_test = logreg.predict(X_test)

y_pred_test
```

Out[100]:

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

predict_proba method

predict_proba method gives the probabilities for the target variable(0 and 1) in this case, in array form.

`0 is for probability of no rain` and `1 is for probability of rain.`

In [101]:

```
# probability of getting output as 0 - no rain

logreg.predict_proba(X_test)[:,0]
```

Out[101]:

```
array([0.91382428, 0.83565645, 0.82033915, ..., 0.97674285, 0.79855098,
       0.30734161])
```

In [102]:

```
# probability of getting output as 1 - rain

logreg.predict_proba(X_test)[:,1]
```

Out[102]:

```
array([0.08617572, 0.16434355, 0.17966085, ..., 0.02325715, 0.20144902,
       0.69265839])
```

jupyter logistic-regression-classifier-tutorial Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

```
logreg.predict_proba(X_test)[:,1]
```

Out[102]:
```
array([0.08617572, 0.16434355, 0.17966085, ..., 0.02325715, 0.20144902,
       0.69265839])
```

Check accuracy score

In [103]:
```
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'. format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.8502
```

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

Compare the train-set and test-set accuracy

In [104]:
```
y_pred_train = logreg.predict(X_train)

y_pred_train
```

Out[104]:
```
array(['No', 'No', 'No', ..., 'No', 'No', 'No'], dtype=object)
```

In [105]:
```
print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train)))
```

```
Training-set accuracy score: 0.8476
```

Confusion matrix

jupyter logistic-regression-classifier-tutorial Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

Training-set accuracy score: 0.8476

Confusion matrix

In [113]:
```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) = ', cm[1,1])

print('\nFalse Positives(FP) = ', cm[0,1])

print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[20892  1175]
 [ 3086  3286]]

True Positives(TP) =  20892

True Negatives(TN) =  3286

False Positives(FP) =  1175

False Negatives(FN) =  3086
```

```
True Negatives(TN) =  5286

False Positives(FP) =  1175

False Negatives(FN) =  3086
```

Classification Efficiency

Classification Report

In [115]:
```
from sklearn.metrics import classification_report

print(classification_report(y_test, y_pred_test))
```

```
              precision    recall  f1-score   support

          No       0.87      0.95      0.91     22067
         Yes       0.74      0.52      0.61      6372

    accuracy                           0.85     28439
   macro avg       0.80      0.73      0.76     28439
weighted avg       0.84      0.85      0.84     28439
```

Classification accuracy

In [116]:
```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
```

jupyter logistic-regression-classifier-tutorial Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help

Not Trusted

Python 3 (ipykernel)

Run Code

In [116]:
```
TP = cm[0,0]
TN = cm[1,1]
FP = cm[0,1]
FN = cm[1,0]
```

In [117]:
```
# print classification accuracy

classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

```
Classification accuracy : 0.8502
```

Classification error

In [118]:
```
# print classification error

classification_error = (FP + FN) / float(TP + TN + FP + FN)

print('Classification error : {0:0.4f}'.format(classification_error))
```

```
Classification error : 0.1498
```

Precision

In [119]:
```
# print precision score
```

jupyter logistic-regression-classifier-tutorial Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

Precision

In [119]:
```
# print precision score

precision = TP / float(TP + FP)


print('Precision : {0:0.4f}'.format(precision))
```

```
Precision : 0.9468
```

Recall

In [120]:
```
recall = TP / float(TP + FN)

print('Recall or Sensitivity : {0:0.4f}'.format(recall))
```

```
Recall or Sensitivity : 0.8713
```

True Positive Rate

True Positive Rate is synonymous with **Recall**.

In [121]:
```
true_positive_rate = TP / float(TP + FN)


print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))
```

jupyter logistic-regression-classifier-tutorial Last Checkpoint: 8 minutes ago (autosaved)

Logout

File Edit View Insert Cell Kernel Help Not Trusted Python 3 (ipykernel)

Run Code

print('True Positive Rate : {0:0.4f}'.format(true_positive_rate))

```
True Positive Rate : 0.8713
```

False Positive Rate

In [122]:
```
false_positive_rate = FP / float(FP + TN)


print('False Positive Rate : {0:0.4f}'.format(false_positive_rate))
```

```
False Positive Rate : 0.2634
```

Specificity

In [123]:
```
specificity = TN / (TN + FP)

print('Specificity : {0:0.4f}'.format(specificity))
```

```
Specificity : 0.7366
```

DISCUSSION and CONCLUSION

The logistic regression model has been applied and executed successfully on the classification problem over the weather dataset of Australia from Kaggle.