# Weekly Returns (1990_2010)

**March 28, 2023**

**Program #01:**

This data set contains 1 089 weekly returns for 21 years, from the beginning of 1990 to the end of 2010.

a) Produce some numerical and graphical summaries of the Weekly data. Do there appear to be any patterns?

(b) Use the full data set to perform a logistic regression with Direction as the response and the five lag variables plus Volume as predictors. Use the summary function to print the results. Do any of the predictors appear to be statistically significant? If so, which ones?

(c) Compute the confusion matrix and overall fraction of correct predictions. Explain what the confusion matrix is telling you about the types of mistakes made by logistic regression.
(d) Now fit the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor. Compute the confusion matrix and the overall fraction of correct predictions for the held out data (that is, the data from 2009 and 2010).

CODE:

[1]:
```
#loading the required packages

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

[2]:
```
#loading the data set
```

```python
data = pd.read_csv("Weekly.csv")
```

[3]:
```python
#creating a shallow copy
weekly = data.copy()
weekly
```

[3]:

| | Year | Lag1 | Lag2 | Lag3 | Lag4 | Lag5 | Volume | Today | Direction |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1990 | 0.816 | 1.572 | -3.936 | -0.229 | -3.484 | 0.154976 | -0.270 | Down |
| 1 | 1990 | -0.270 | 0.816 | 1.572 | -3.936 | -0.229 | 0.148574 | -2.576 | Down |
| 2 | 1990 | -2.576 | -0.270 | 0.816 | 1.572 | -3.936 | 0.159837 | 3.514 | Up |
| 3 | 1990 | 3.514 | -2.576 | -0.270 | 0.816 | 1.572 | 0.161630 | 0.712 | Up |
| 4 | 1990 | 0.712 | 3.514 | -2.576 | -0.270 | 0.816 | 0.153728 | 1.178 | Up |
| … | … | … | … | … | … | … | … | … | … |
| 1084 | 2010 | -0.861 | 0.043 | -2.173 | 3.599 | 0.015 | 3.205160 | 2.969 | Up |
| 1085 | 2010 | 2.969 | -0.861 | 0.043 | -2.173 | 3.599 | 4.242568 | 1.281 | Up |
| 1086 | 2010 | 1.281 | 2.969 | -0.861 | 0.043 | -2.173 | 4.835082 | 0.283 | Up |
| 1087 | 2010 | 0.283 | 1.281 | 2.969 | -0.861 | 0.043 | 4.454044 | 1.034 | Up |
| 1088 | 2010 | 1.034 | 0.283 | 1.281 | 2.969 | -0.861 | 2.707105 | 0.069 | Up |

[1089 rows x 9 columns]

[5]:
```python
#checking for null values
weekly.isnull().sum()
```

[5]:
```
Year         0
Lag1         0
Lag2         0
Lag3         0
Lag4         0
Lag5         0
Volume       0
Today        0
Direction    0
```

```
dtype: int64
```

[4]:
```
#checking data types
weekly.dtypes
```

[4]:
```
Year        int64
Lag1      float64
Lag2      float64
Lag3      float64
Lag4      float64
Lag5      float64
```

```
Volume       float64
Today        float64
Direction     object
dtype: object
```

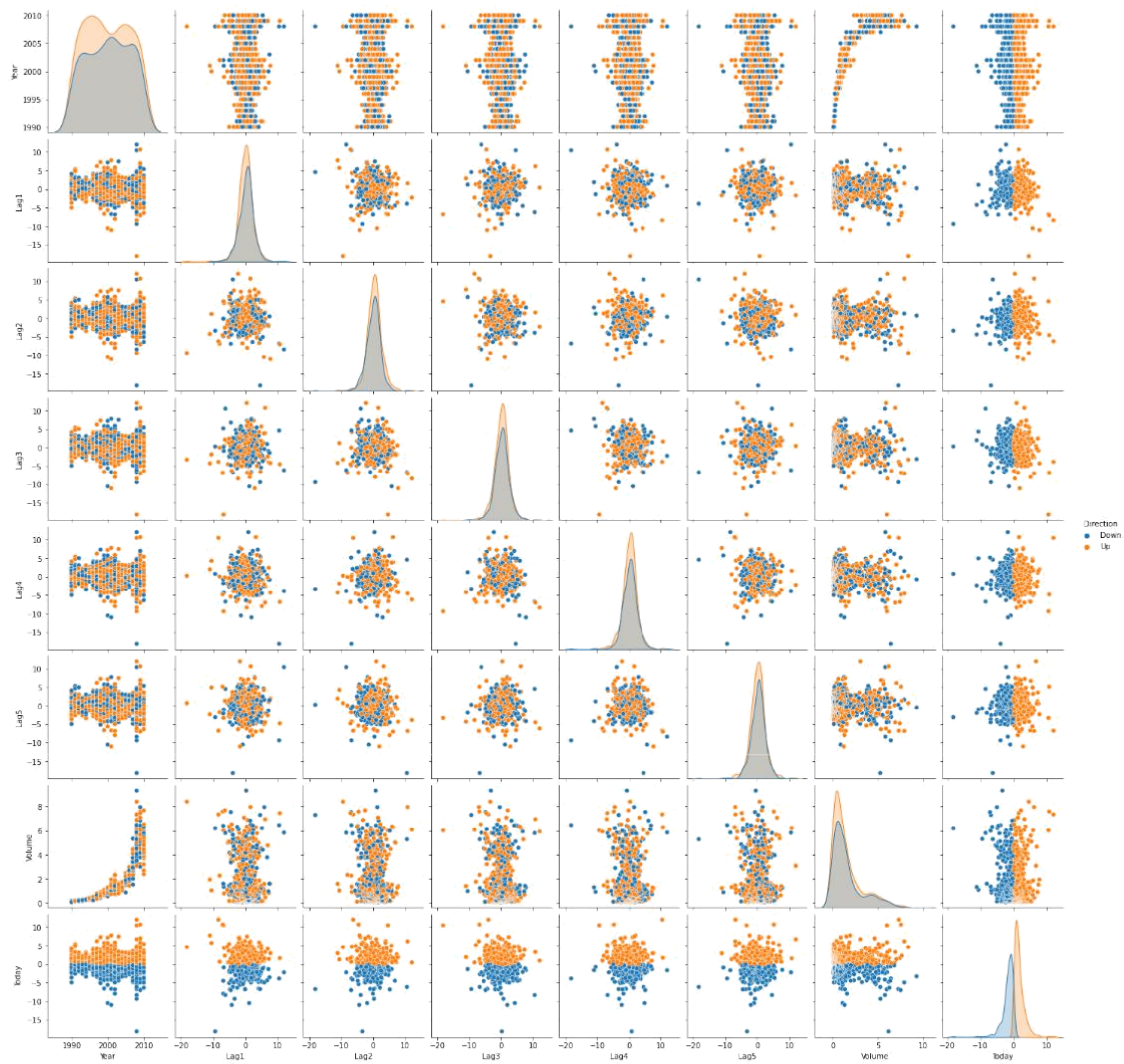
Numerical Analysis of the weekly data

```
data.describe()
```

| | Year | Lag1 | Lag2 | Lag3 | Lag4 | Lag5 | Volume | Today |
|---|---|---|---|---|---|---|---|---|
| **count** | 1089.000000 | 1089.000000 | 1089.000000 | 1089.000000 | 1089.000000 | 1089.000000 | 1089.000000 | 1089.000000 |
| **mean** | 2000.048669 | 0.150585 | 0.151079 | 0.147205 | 0.145818 | 0.139893 | 1.574618 | 0.149899 |
| **std** | 6.033182 | 2.357013 | 2.357254 | 2.360502 | 2.360279 | 2.361285 | 1.686636 | 2.356927 |
| **min** | 1990.000000 | -18.195000 | -18.195000 | -18.195000 | -18.195000 | -18.195000 | 0.087465 | -18.195000 |
| **25%** | 1995.000000 | -1.154000 | -1.154000 | -1.158000 | -1.158000 | -1.166000 | 0.332022 | -1.154000 |
| **50%** | 2000.000000 | 0.241000 | 0.241000 | 0.241000 | 0.238000 | 0.234000 | 1.002680 | 0.241000 |
| **75%** | 2005.000000 | 1.405000 | 1.409000 | 1.409000 | 1.409000 | 1.405000 | 2.053727 | 1.405000 |
| **max** | 2010.000000 | 12.026000 | 12.026000 | 12.026000 | 12.026000 | 12.026000 | 9.328214 | 12.026000 |

[6]:
```
sns.pairplot(weekly, hue = "Direction")
```

[6]: <seaborn.axisgrid.PairGrid at 0x7f2b44c00670>


Year
Lag1
Lag2
Lag3
Lag4
Lag5
Volume
Today
Direction
Down
Up

From the above pairplot, it can be observed that all the predictors have some degree of association with the response variable.The lag variables all have the same type of correlation.

The variable today is distinctly split between values having direction up and down. This phenomenon is not observed in the variable today.

Fitting Logistic Regression Using Whole Data Set

[7]:
```
from sklearn.preprocessing import
LabelEncoder enc = LabelEncoder()
```

```
weekly["Direction"] = enc.fit_transform(weekly["Direction"])
```

[8]:
```
X = weekly.loc[:, weekly.columns != "Direction"]

y= weekly["Direction"]
```

[9]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y,
 test_size = 0.3, ↪random_state = 6)
```

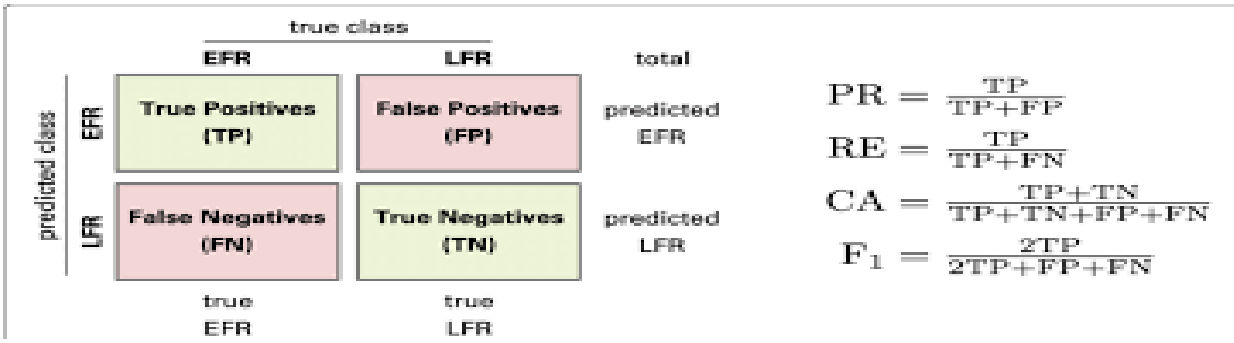
[10]:
```
from sklearn.linear_model import LogisticRegression
reg_model1 = LogisticRegression()
```

[11]:
```
reg_model1.fit(X_train,y_train)
```

[11]: LogisticRegression()

[13]:
```
#summary of the logistic model

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
y_pred = reg_model1.predict(X_test)
print("The accuracy of the model is", accuracy_score(y_test, y_pred).round(2))

print("The f1 score of the model is", f1_score(y_test,y_pred).round(2))

print("The precision score of the model is", precision_score(y_test,
y_pred).

 ↪round(2))

print("The recall score of the model is", recall_score(y_test, y_pred).round(2))
```

```
The accuracy of the model is 0.82

The f1 score of the model is 0.99

The precision score of the model is 0.97

The recall score of the model is 1
//
```




[24]:
```
#computing confusion matrix

from sklearn.metrics import confusion_matrix
cfm = confusion_matrix(y_test, y_pred)

print("The confusion matrix for the logistic model is\n", cfm)
```

```
The confusion matrix for the logistic
 model is [[146 3]

 0  0 178]]
```

From the confusion matrix, it can be observed that the logistic model correctly predicted 146 true values and 178 false values. Only 3 true values were incorrectly predicted.

[22]:
```
#computing fraction of correct predictions

frac_correct = (cfm[0][0]+cfm[1][1])/(cfm[0][0]+cfm[0][1]+cfm[1][0]+cfm[1][1])
print("The fraction of correct predictions of the model is", frac_correct.

 ↪round(2))
```

```
The fraction of correct predictions of the model is 0.99
```

Fitting the logistic regression model using a training data period from 1990 to 2008, with Lag2 as the only predictor.

[58]:
```
weekly_training = weekly[weekly["Year"]<=2008]
weekly_testing = weekly[weekly["Year"]>2008]
```

[68]:
```
X1_train = np.array(weekly_training["Lag2"]).reshape(-1,1)

y1_train = np.array(weekly_training["Direction"]).reshape(-1,1)
```

[69]:
```
X1_test = np.array(weekly_testing["Lag2"]).reshape(-1,1)

y1_test = np.array(weekly_testing["Direction"]).reshape(-1,1)
```

[62]:
```
reg_model2 = LogisticRegression()
```

[70]:
```
reg_model2.fit(X1_train, y1_train)
```

```
/usr/local/lib/python3.9/dist-
packages/sklearn/utils/validation.py:1143: 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)
```

[70]:
```
LogisticRegression()
```

[71]:
```
#summary of the logistic model

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
y1_pred = reg_model2.predict(X1_test)

print("The accuracy of the model is", accuracy_score(y1_test, y1_pred).round(2))

print("The f1 score of the model is",
f1_score(y1_test,y1_pred).round(2))

print("The precision score of the model is", precision_score(y1_test, y1_pred).

 ↪round(2))

print("The recall score of the modle is", recall_score(y1_test,

 y1_pred). ↪round(2))
```

```
The accuracy of the model is 0.62
```

```
The f1 score of the model is 0.74

The precision score of the model is 0.62

The recall score of the modle is 0.92
```

```
[74]: #computing confusion matrix

from sklearn.metrics import confusion_matrix
cfm1 = confusion_matrix(y1_test, y1_pred)
print("The confusion matrix for the logistic model is\n", cfm1)
```

```
The confusion matrix for the logistic model is

 [[ 9 34]

 [ 5 56]]
```

From the confusion matrix, it can be observed that the logistic model correctly predicted 9 true values and 56 false values. Only 9 true values were correctly predicted.

```
[75]: #computing fraction of correct predictions
frac_correct = (cfm1[0][0]+cfm1[1][1])/
 ↪(cfm1[0][0]+cfm1[0][1]+cfm1[1][0]+cfm1[1][1])

print("The fraction of correct predictions of the model is",
 frac_correct. ↪round(2))
```

```
The fraction of correct predictions of the model is 0.62 Which means 62 percent of the
variability in Y is explained by  the predictor variables. This is  a good fit.
```

# Auto MPG Logistic Regression

**Program #02:**

You will develop a model to predict whether a given car gets high or low gas mileage.

(a) Create a binary variable, mpg01, that contains a 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can compute the median using the median() function. Note you may find it helpful to use the data.frame() function to create a single data set containing both mpg01 and the other Auto variables.

(b) Explore the data graphically in order to investigate the association between mpg01 and the other features. Which of the other features seem most likely to be useful in predicting mpg01? Scatter plots and box plots may be useful tools to answer this question. Describe your findings.

(c) Split the data into a training set and a test set.

(d) Perform logistic regression on the training data in order to predict mpg01 using the variables that seemed most associated with mpg01 in (b). What is the test error of the model obtaine

[2]:
```
import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns
```

[3]:
```
auto_original = pd.read_csv("/content/auto-mpg.csv")
```

[4]:
```
#making a shallow copy of the data

auto = auto_original.copy()

auto.head()
```

[4]:
```
    mpg  cylinders  displacement horsepower  weight  acceleration  model year  \
0  18.0          8         307.0        130    3504          12.0          70
1  15.0          8         350.0        165    3693          11.5          70
2  18.0          8         318.0        150    3436          11.0          70
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70

   origin                   car name
0       1  chevrolet chevelle malibu
1       1          buick skylark 320
2       1         plymouth satellite
3       1              amc rebel sst
4       1                ford torino
```

Data pre-processing

[5]:
```
#checking data types
auto.dtypes
```

[5]:
```
mpg             float64
cylinders         int64
displacement    float64
horsepower       object
weight            int64
acceleration    float64
model year        int64
origin            int64
```

```
car name        object
dtype: object
```

[6]:
```
#checking levels of horsepower
auto["horsepower"].unique()
```

[6]:
```
array(['130', '165', '150', '140', '198', '220', '215', '225', '190',
       '170', '160', '95', '97', '85', '88', '46', '87', '90', '113',
       '200', '210', '193', '?', '100', '105', '175', '153', '180', '110',
       '72', '86', '70', '76', '65', '69', '60', '80', '54', '208', '155',
       '112', '92', '145', '137', '158', '167', '94', '107', '230', '49',
       '75', '91', '122', '67', '83', '78', '52', '61', '93', '148',
       '129', '96', '71', '98', '115', '53', '81', '79', '120', '152',
       '102', '108', '68', '58', '149', '89', '63', '48', '66', '139',
       '103', '125', '133', '138', '135', '142', '77', '62', '132', '84',
       '64', '74', '116', '82'], dtype=object)
```

[7]:
```
#removing garbage values
auto = auto.replace("?", np.nan)
```

[8]:
```
#checking for missing data
total=auto.isnull().sum().sort_values(ascending=False)
percent=((auto.isnull().sum()/auto.isnull().count())*100).
 ↪sort_values(ascending=False)
missing_values=pd.concat([total, percent], axis=1, keys=["Total
 Missing Data",␣↪"Percentage of Missing Data"])
missing_values
```

[8]:

| | Total Missing Data | Percentage of Missing Data |
|---|---|---|
| horsepower | 6 | 1.507538 |
| mpg | 0 | 0.000000 |
| cylinders | 0 | 0.000000 |

```
displacement          0          0.000000
weight                0          0.000000
acceleration          0          0.000000
model year            0          0.000000
origin                0          0.000000
car name              0          0.000000
```

[9]:
```
#removing missing data

auto = auto.dropna()

auto.head()
```

[9]:
```
    mpg   cylinders  displacement  horsepower  weight  acceleration  model year  \
0  18.0           8         307.0         130    3504          12.0          70
1  15.0           8         350.0         165    3693          11.5          70
2  18.0           8         318.0         150    3436          11.0          70
```

```
3  16.0          8         304.0        150    3433          12.0          70
4  17.0          8         302.0        140    3449          10.5          70

   origin                    car name
0       1    chevrolet chevelle malibu
1       1            buick skylark 320
2       1           plymouth satellite
3       1                amc rebel sst
4       1                  ford torino
```

[10]:

```
#changing data typr to int
auto = auto.astype({"horsepower":int})
```

Regression Model

[11]:

```
#calcualting the median
from statistics import median
med = median(auto["mpg"])
```

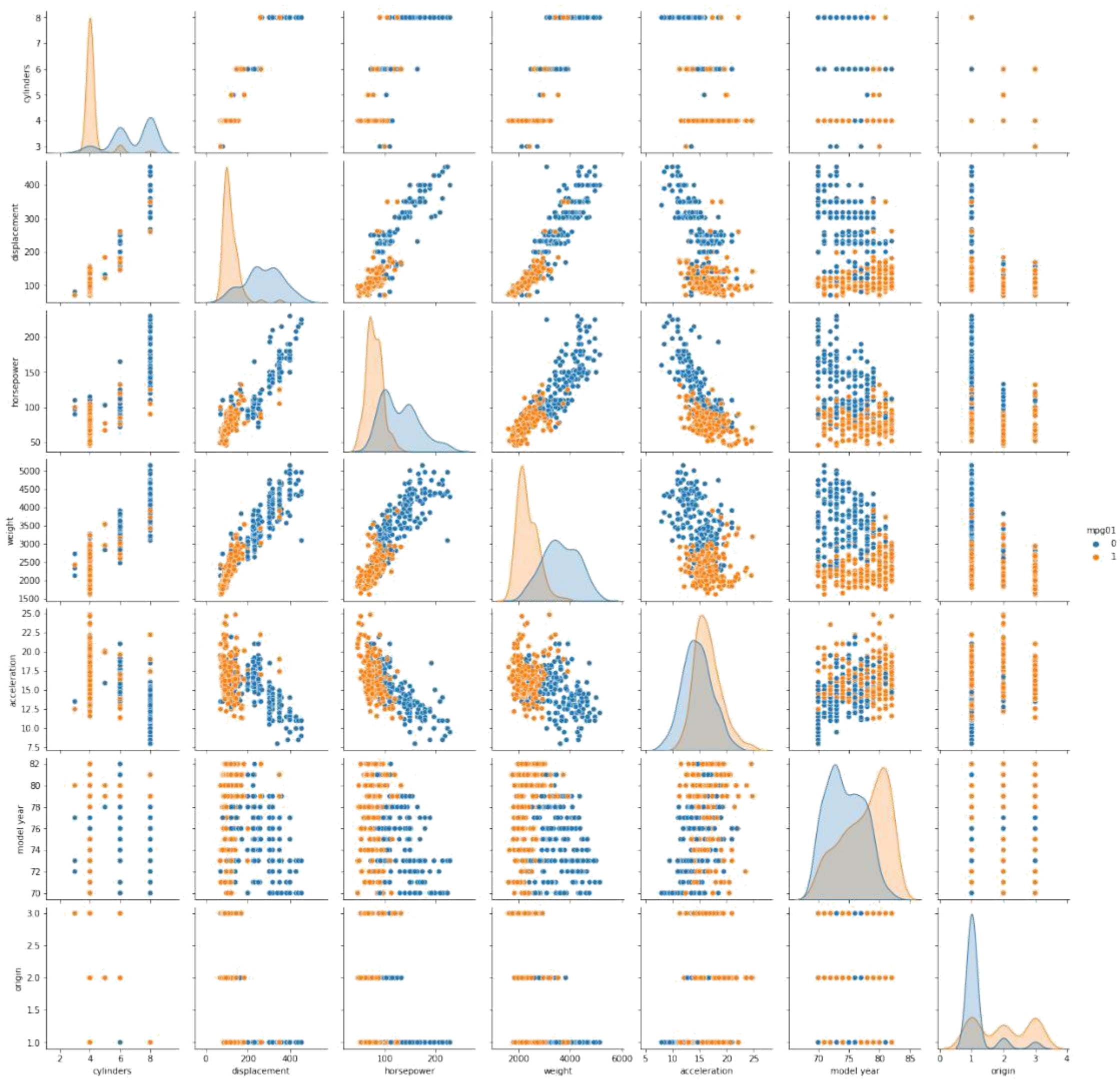
[24]:

```
#formulation of binary variable
mpg01 = []
for row in auto["mpg"]:
  if row <= med:
    mpg01.append(0)
  else:
    mpg01.append(1)

mpg01 = np.array(mpg01).reshape(-1,1)
```

[39]:
```
#adding binary variable to data frame

auto["mpg01"]=mpg01

auto = auto.drop("mpg", axis = 1)

auto = auto.drop("car name", axis = 1)
```

[40]:
```
#graphical exploration

sns.pairplot(auto, hue = "mpg01")
```

[40]: <seaborn.axisgrid.PairGrid at 0x7fc1a2a41a30>


cylinders
displacement
horsepower
weight
acceleration
model year
origin
mpg01
0
1

From the above plot, we can observe that the variables displacement, horsepower, weight and accelaeration have a higher degree of association when mpg is taken as the response variable. Hence we use these variables to formulate the model.

```
[50]: #removing unnesessary predictors
auto = auto.drop("origin", axis =
1) auto = auto.drop("model year",
axis = 1) auto =
auto.drop("cylinders", axis = 1)
```

```
[51]: #splitting the data into predictors and response
variables X = auto.loc[:,auto.columns != "mpg01"]

y = auto["mpg01"]
```

X contains all the predictor variables, and y contains the response variable for the auto dataset. This separation of predictor and response variables is necessary for building and evaluating predictive models.

[53]:
```
#splitting the data into training and testing set
from sklearn.model_selection import train_test_split

X_train, X_test, y_train,y_test = train_test_split(X,y,
 test_size = 0.25, ↪random_state = 6)
```

[37]:
```
 #model formulation

from sklearn.linear_model import LogisticRegression
reg_model = LogisticRegression()
```

[54]:
```
 #fitting the model

reg_model.fit(X_train, y_train)
```

[54]: LogisticRegression()

[55]:
```
 #summary of the logistic model

from sklearn.metrics import accuracy_score,precision_score,recall_score,f1_score
y_pred = reg_model.predict(X_test)
print("The accuracy of the model is", accuracy_score(y_test, y_pred).round(2))

print("The f1 score of the model is", f1_score(y_test,y_pred).round(2))

print("The precision score of the model is", precision_score(y_test, y_pred).
 ↪round(2))

print("The recall score of the modle is", recall_score(y_test, y_pred).round(2))
```

```
The accuracy of the model is 0.86

The f1 score of the model is 0.85

The precision score of the model is 0.82

The recall score of the modle is 0.89
```

[56]:
```
 #calculating the test error
```

```
test_error = print("The test error of the model is", 1-
 accuracy_score(y_test, ␣ ↪y_pred).round(2))
```

```
The test error of the model is 0.14
```

The accuracy score measures the proportion of correctly classified instances out of all instances in the test set. In this case, the accuracy of the model is 0.86, which means that 86% of the instances in the test set were correctly classified.

The f1 score is the harmonic mean of precision and recall, and it provides a balanced measure between the two. The f1 score in this case is 0.85, which is high and indicates that the model has a good balance between precision and recall.

The precision score measures the proportion of true positives (correctly predicted positives) out of all predicted positives. The precision score in this case is 0.82, which means that 82% of the predicted positive instances were true positives.

The recall score measures the proportion of true positives out of all actual positives. The recall score in this case is 0.89, which means that 89% of the actual positive instances were correctly identified by the model.

Finally, the code calculates the test error, which is the proportion of incorrectly classified instances in the test set. In this case, the test error is 0.14, which means that 14% of the instances in the test set were incorrectly classified.