INDIAN INSTITUTE OF TECHNOLOGY ROPAR

Foundation of Data Science

Guided By:
Dr. Arun Kumar

Submitted By:
Akansha Sukhadeve(2020CHB1038)
Shailja Nigam(2020CSB1124)
Shreya Kamble(2020CEB1028)

Auto Data

Problem Statement: Do exploratory data analysis on the Auto data. Using the ridge and lasso regression techniques on the the data and predict the mpg. Compare your results.

Ridge and Lasso Regression:

Ridge and Lasso Regression are regularization techniques used to prevent overfitting in linear regression models.

➢ **Ridge Regression:**

- Adds a regularization term proportional to the square of the coefficients to the linear regression cost function.
- The regularization term helps to shrink the coefficients, making them more robust cost function.

➢ **Lasso Regression:**

- Lasso Regression introduces adds a regularization term, which includes the absolute values of the coefficients.
- Lasso not only prevents overfitting but also performs feature selection by driving some coefficients to exactly zero. This can lead to a more interpretable and sparse model.

Comparison and Results:

➢ **Model Training:**

- Split the dataset into training and testing sets to assess the model's generalization performance.

- Train Ridge and Lasso Regression models on the training set, adjusting the regularization parameter based on cross-validation.

➢ **Evaluation:**

- Evaluate model performance on the testing set using metrics like Mean Squared Error(MSE), R-squared, or other relevant measures.
- Compare the results of Ridge and Lasso models to understand their respective strengths and weaknesses in predicting the target variable(mpg in the case).

➔ **Ridge regression**

It performs L2 regularization, i.e., adds penalty equivalent to square of the magnitude of coefficients.

Minimization objective = LS Obj + α *(Sum of square of coefficients)

➔ **Lasso Regression:**

Performs L1 regularization, i.e., adds penalty equivalent to absolute value of the magnitude of coefficients.

Note that here ‘LS Obj’ refers to ‘least squares objective’, i.e., the linear regression objective without regularization.

We can see that when alpha is greater in these cases accuracy of ridge regression id better than lasso regression and when alpha becomes close to zero accuracy of ridge and lasso regression becomes equal.

1} alpha = 0.1

```
from sklearn.linear_model import Lasso
reg = Lasso(alpha = 0.1)
reg.fit(X_train,y_train)
y_pred = reg.predict(X_test)
print("regressor score lasso test\t", reg.score(X_test, y_test))
print("regressor score lasso training\t", reg.score(X_train, y_train))
```

```
regressor score lasso test       0.8070750530795664
regressor score lasso training   0.8199743294365871
```

```
[24] from sklearn.linear_model import Ridge
     regr = Ridge(alpha = 0.1)
     regr.fit(X_train,y_train)
     yr_pred = regr.predict(X_test)
     print("regressor score ridge test\t", regr.score(X_test, y_test))
     print("regressor score ridge train\t", regr.score(X_train, y_train))
```

```
regressor score ridge test       0.8066732364614188
regressor score ridge train      0.8211588091231565
```

2} alpha = 5

```
[15] from sklearn.linear_model import Lasso
     reg = Lasso(alpha = 5)
     reg.fit(X_train,y_train)
     y_pred = reg.predict(X_test)
     print("regressor score lasso test\t", reg.score(X_test, y_test))
     print("regressor score lasso training\t", reg.score(X_train, y_train))
```

```
regressor score lasso test       0.7779511974731892
regressor score lasso training   0.7755251026018444
```

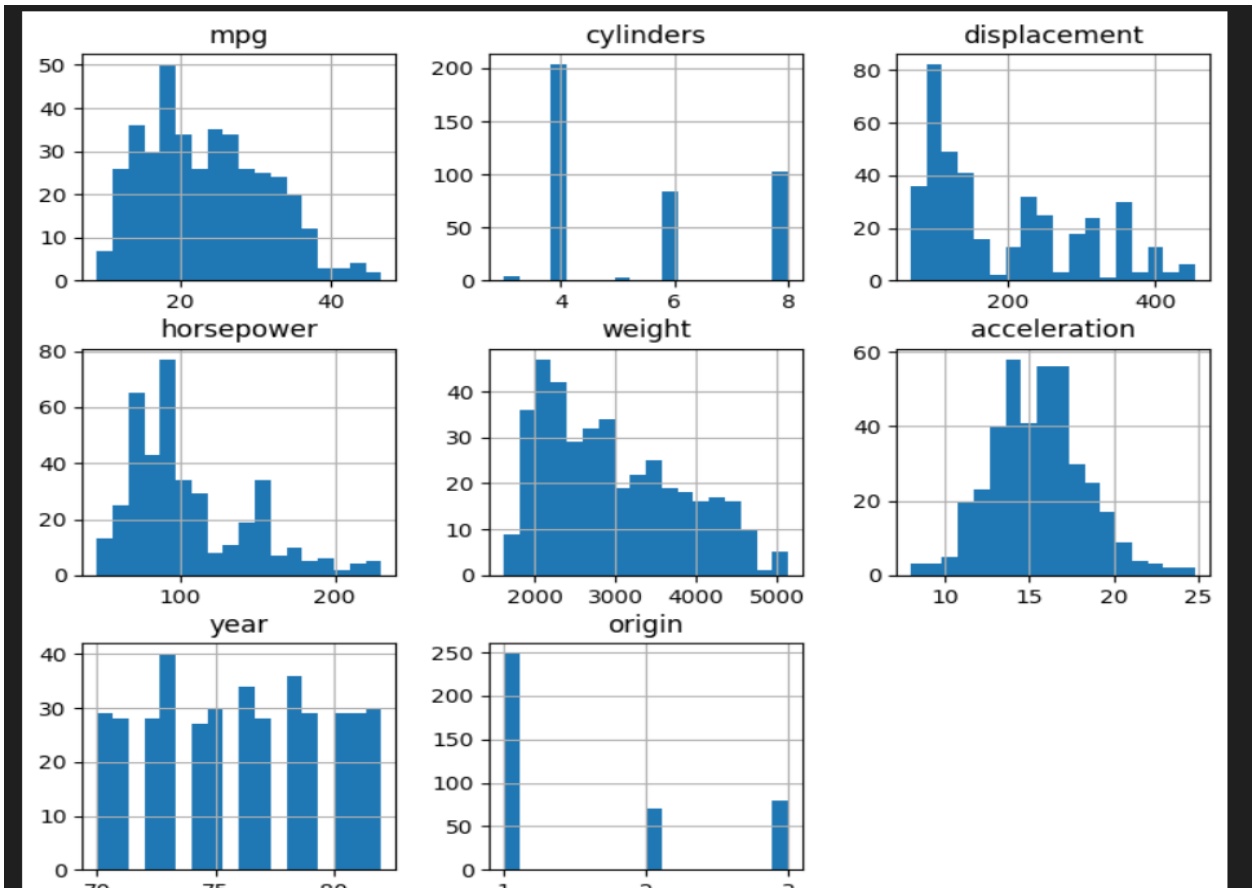
```
[18] from sklearn.linear_model import Ridge
     regr = Ridge(alpha = 5)
     regr.fit(X_train,y_train)
     yr_pred = regr.predict(X_test)
     print("regressor score ridge test\t", regr.score(X_test, y_test))
     print("regressor score ridge train\t", regr.score(X_train, y_train))
```

```
regressor score ridge test       0.8070290753188015
regressor score ridge train      0.8211270067471811
```

3} alpha = 10

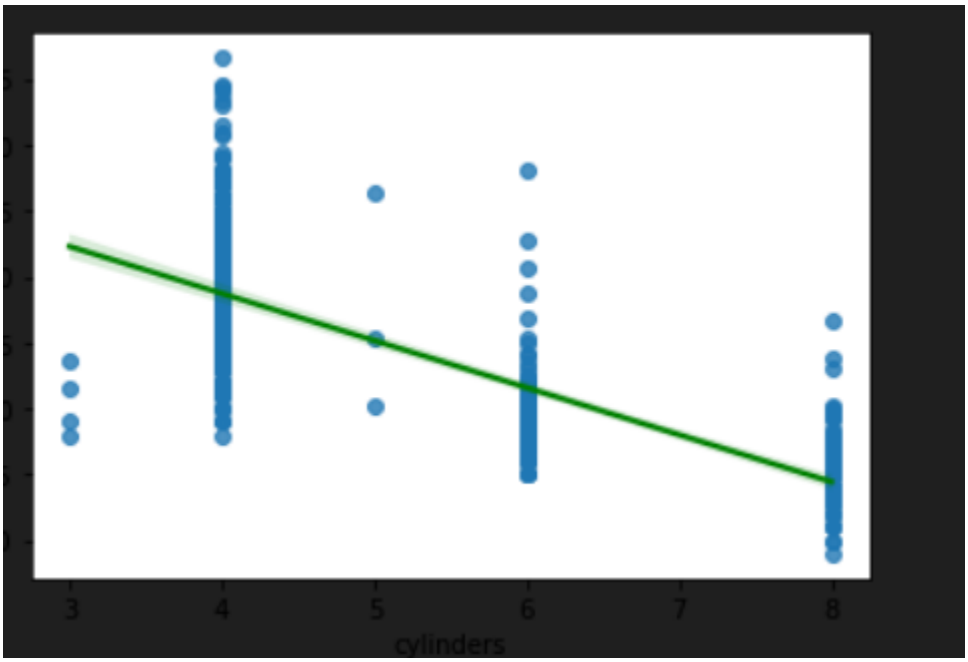
```
[22] from sklearn.linear_model import Lasso
     reg = Lasso(alpha = 10)
     reg.fit(X_train,y_train)
     y_pred = reg.predict(X_test)
     print("regressor score lasso test\t", reg.score(X_test, y_test))
     print("regressor score lasso training\t", reg.score(X_train, y_train))

     regressor score lasso test       0.7017707936204175
     regressor score lasso training   0.6939881143144223

     from sklearn.linear_model import Ridge
     regr = Ridge(alpha = 10)
     regr.fit(X_train,y_train)
     yr_pred = regr.predict(X_test)
     print("regressor score ridge test\t", regr.score(X_test, y_test))
     print("regressor score ridge train\t", regr.score(X_train, y_train))

     regressor score ridge test       0.8072890655521598
     regressor score ridge train      0.8210437669700963
```

Histogram:

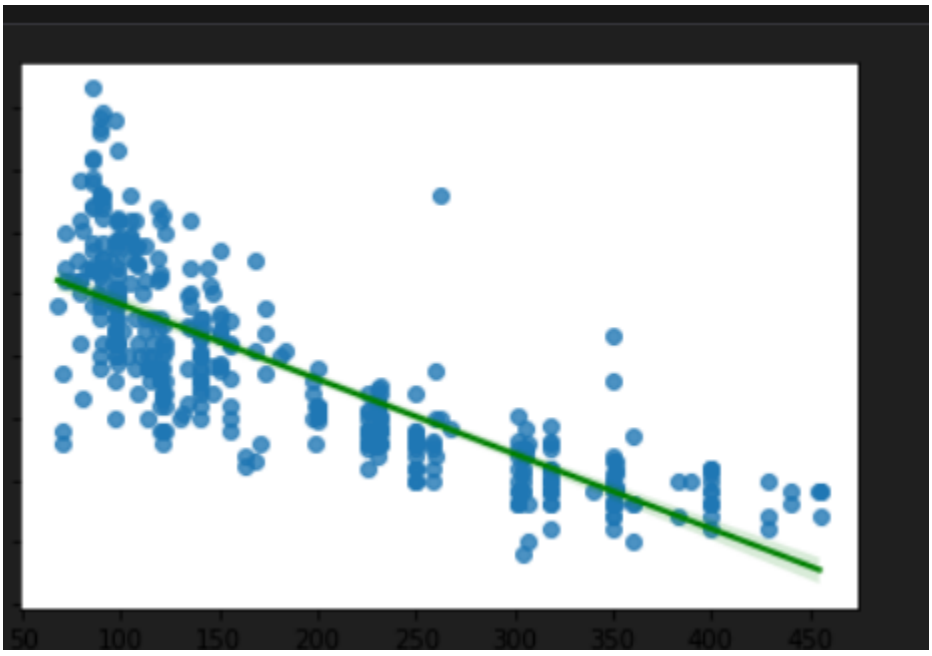


Regression Plots:

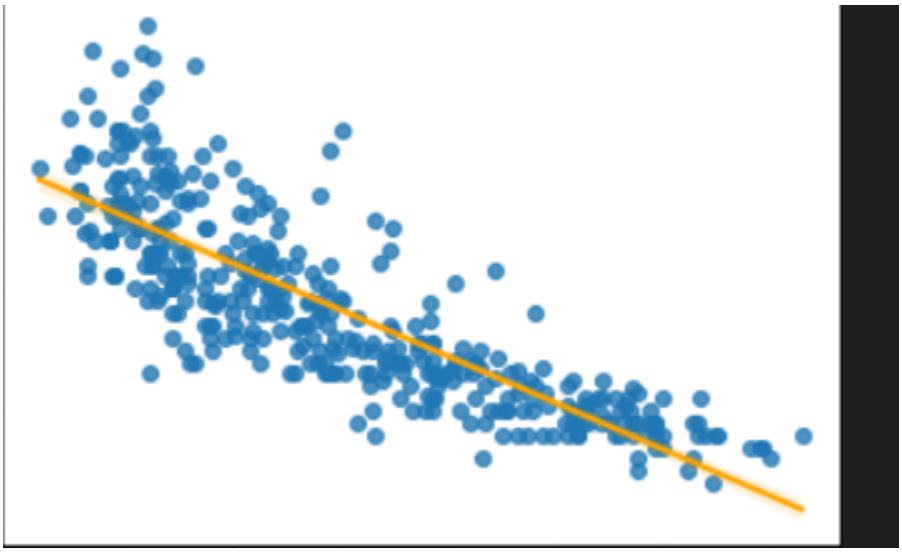
1. **X =’cylinders’, Y = ‘mpg’**



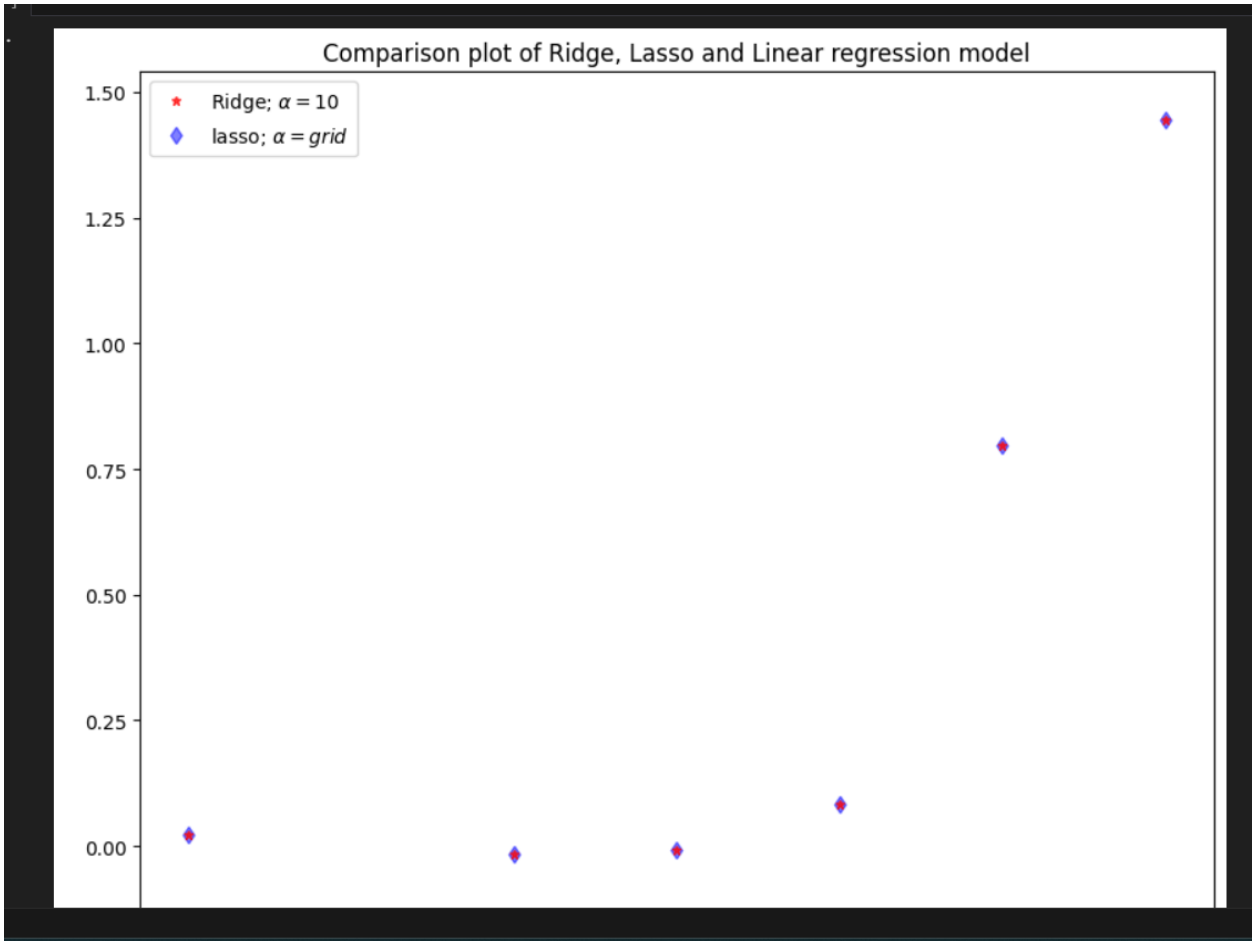
2. **X= ‘Displacement’ , Y = ‘mpg’**



3. **X= ‘weight’ , Y = ‘mpg’**



Comparison Plot of Ridge , Lasso and Linear Regression:



Conclusion:

In Summary, an exhaustive exploratory data analysis establishes the groundwork for well-informed decisions regarding modeling. By virtue of their regularization properties, Ridge and Lasso regression provide methods to prevent overfitting and enhance the robustness of models; the selection between the two is contingent on the particular attributes of the dataset and the objectives of the modeling process.

Both ridge regression and lasso regression have a similar cost function. However, lasso takes the magnitude and ridge regression of the square of the coefficients.

Result:

Results from a ridge or lasso model with an alpha value of 0 will resemble those from a regression model. The severity of the penalty increases with the alpha value.

Boston Data

Problem Statement: Exploratory data analysis on the BOSTON data. Use linear regression and LDA to predict whether a given suburb has a crime rate above or below the median. Compare the findings from different methods.

1. Exploratory data Analysis

1. Inserting libraries

```
[71] ##IMPORTING THE LIBRARIES
     import numpy as np
     import pandas as pd
     import matplotlib.pyplot as plt
     import seaborn as sns
     %matplotlib inline
```

We use the Pandas library to import the data. Here we have a CSV file for the BOSTON data. So, we have used the command **"read_csv"** to read the CSV file. If the data is contained in an Excel file, we would have used the command **"read_xlsx".** We have saved the BOSTON data as **"boston_data".**

```
[13] #Reading the data
     boston_data = pd.read_csv('/content/Boston.csv')
     boston_data.head() #Viewing 1st 5 rows of the data
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

boston_data.head() command is used to print the first 5 entries of the data. boston_data.tail() command is used to print the last 5 entries of the data.

We can also specify the number of entries to be displayed by specifying in the brackets "(_)".

2. Libraries in Python

A python library is a collection of related modules. It contains codes that can be used multiple times in a program. It makes Python Programming easier for a programmer. Python libraries play a very vital role in the field of Machine Learning, Data Science, Data Visualization, etc.

a. Pandas: Pandas is primarily used for data analysis, data manipulation, cleaning of data and it is one of the most commonly used Python libraries. It supports operations like sorting, Re-indexing, iteration, concatenation, visualization etc.
b. Numpy: Numpy stands for "Numerical Python". Numpy is the one of the most commonly used libraries. It supports large matrices and multi-dimensional data. It contains built-in functions that make the computations easier. It uses array oriented computations which makes the computations easier.
c. Matplotlib: Matplotlib library is used to plot numerical data. We can design charts, graphs, pie charts, scatter plots, histogram, error charts, etc.
d. Seaborn: Seaborn library is used for creating different visualizations. Seaborn is among the reliable sources. This Python library is derived from Matplotlib and closely integrated with Pandas data structures.
e. Scikit-learn: Scikit-learn library is used for a variety of data science applications such as classification, regression, clustering, model selection, k-means, etc.

3. Summary of Data

We have BOSTON data which contains the following columns.

a. Structure of Data.

-Number of Cases

The dataset contains a total of 506 cases.

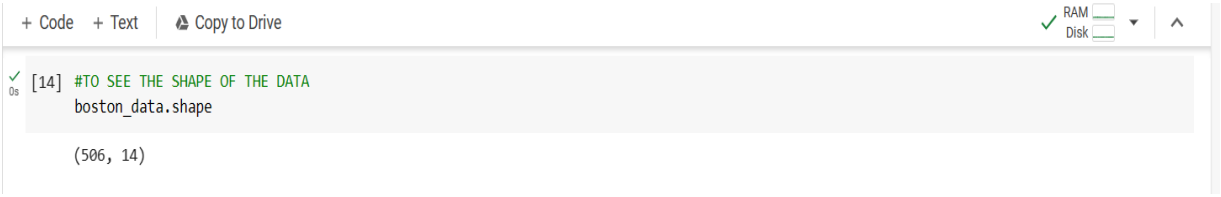
-Order

The order of the cases is mysterious.

-Variables

There are 14 attributes in each case of the dataset. They are:
crim - per capita crime rate by town
zn-proportion of residential land zoned for lots over 25,000 sq.ft.
indus - proportion of non-retail business acres per town.
chas - Charles River dummy variable (1 if tract bounds river; 0 otherwise)
nox- nitric oxides concentration (parts per 10 million)
rm - average number of rooms per dwelling
age - proportion of owner-occupied units built prior to 1940
dis - weighted distances to five Boston employment centers
rad- index of accessibility to radial highways
tax- full-value property-tax rate per $10,000
ptratio - pupil-teacher ratio by town
black- is the proportion of blacks by town
lstat- % lower status of the population
medv- Median value of owner-occupied homes in $1000's

data.shape command is used to check the number of rows and columns in the data. As we can see, there are 506 rows and 14 columns in the Boston Dataset



boston_data.info() command is used to get some information about the data. We use this command to check if there are null values in the data. Also, it gives information about the number of Non-Null entities in the data. It provides information about the data type of columns.

```
[17] #Getting the information about the data
     boston_data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 506 entries, 0 to 505
Data columns (total 14 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   crim     506 non-null    float64
 1   zn       506 non-null    float64
 2   indus    506 non-null    float64
 3   chas     506 non-null    int64
 4   nox      506 non-null    float64
 5   rm       506 non-null    float64
 6   age      506 non-null    float64
 7   dis      506 non-null    float64
 8   rad      506 non-null    int64
 9   tax      506 non-null    int64
 10  ptratio  506 non-null    float64
 11  black    506 non-null    float64
 12  lstat    506 non-null    float64
 13  medv     506 non-null    float64
dtypes: float64(11), int64(3)
memory usage: 55.5 KB
```

Here, we observe that all the columns are in numerical features.

b. Describing data:

describe() is used to view some basic statistical details like percentile, mean, std etc. of a data frame or a series of numeric values.

```
[18] boston_data.describe()# to get the statistical infor about the data
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv
count	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000	506.000000
mean	3.613524	11.363636	11.136779	0.069170	0.554695	6.284634	68.574901	3.795043	9.549407	408.237154	18.455534	356.674032	12.653063	22.532806
std	8.601545	23.322453	6.860353	0.253994	0.115878	0.702617	28.148861	2.105710	8.707259	168.537116	2.164946	91.294864	7.141062	9.197104
min	0.006320	0.000000	0.460000	0.000000	0.385000	3.561000	2.900000	1.129600	1.000000	187.000000	12.600000	0.320000	1.730000	5.000000
25%	0.082045	0.000000	5.190000	0.000000	0.449000	5.885500	45.025000	2.100175	4.000000	279.000000	17.400000	375.377500	6.950000	17.025000
50%	0.256510	0.000000	9.690000	0.000000	0.538000	6.208500	77.500000	3.207450	5.000000	330.000000	19.050000	391.440000	11.360000	21.200000
75%	3.677083	12.500000	18.100000	0.000000	0.624000	6.623500	94.075000	5.188425	24.000000	666.000000	20.200000	396.225000	16.955000	25.000000
max	88.976200	100.000000	27.740000	1.000000	0.871000	8.780000	100.000000	12.126500	24.000000	711.000000	22.000000	396.900000	37.970000	50.000000

To start, we create a binary variable, **crim_med**, which is **“0**” if CRIM contains a value above the median and **"1"** otherwise.

```
[22] #creating a seperate column in the dataframe which has value which is above and below median
     crim_med = (boston_data["crim"] > boston_data["crim"].median()).map({False: 0, True: 1})
     boston_data["crim_med"] = crim_med
     boston_data.head(10)
```

	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	black	lstat	medv	crim_med
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0	0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6	0
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7	0
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4	0
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2	0
5	0.02985	0.0	2.18	0	0.458	6.430	58.7	6.0622	3	222	18.7	394.12	5.21	28.7	0
6	0.08829	12.5	7.87	0	0.524	6.012	66.6	5.5605	5	311	15.2	395.60	12.43	22.9	0
7	0.14455	12.5	7.87	0	0.524	6.172	96.1	5.9505	5	311	15.2	396.90	19.15	27.1	0
8	0.21124	12.5	7.87	0	0.524	5.631	100.0	6.0821	5	311	15.2	386.63	29.93	16.5	0
9	0.17004	12.5	7.87	0	0.524	6.004	85.9	6.5921	5	311	15.2	386.71	17.10	18.9	0

Next we explore the data both numerically, by looking at the matrix of correlations, and graphically, by looking at boxplots to investigate the association between crim_med and the other features. First, we look at the matrix of correlations.

From above, We can observe that there are some moderate to moderately strong correlations between crime rate and the other variables:

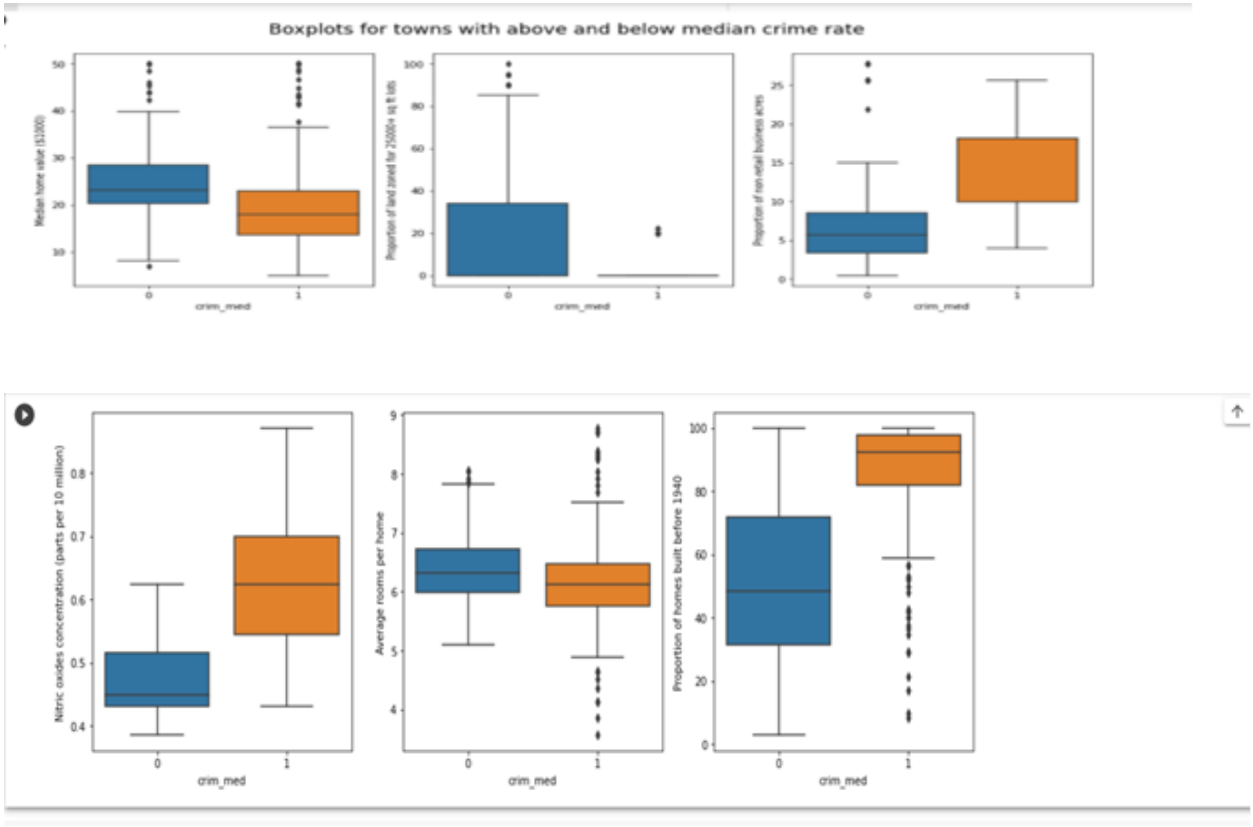
- The most correlated variables are **'rad'** (correlation of 0.626), a measure of accessibility to radial highways, and '**tax**' (correlation of 0.583), the property tax rate in USD per $10,000. In fact, all of the variables aside from **'chas'**, an indicator variable with the value of 1 if a town borders the Charles River, have correlation values with a magnitude of at least 0.2.
- We will further plot the box plot and discuss more the correlation and decide which all should be the predictors and which all should be dropped from the table.

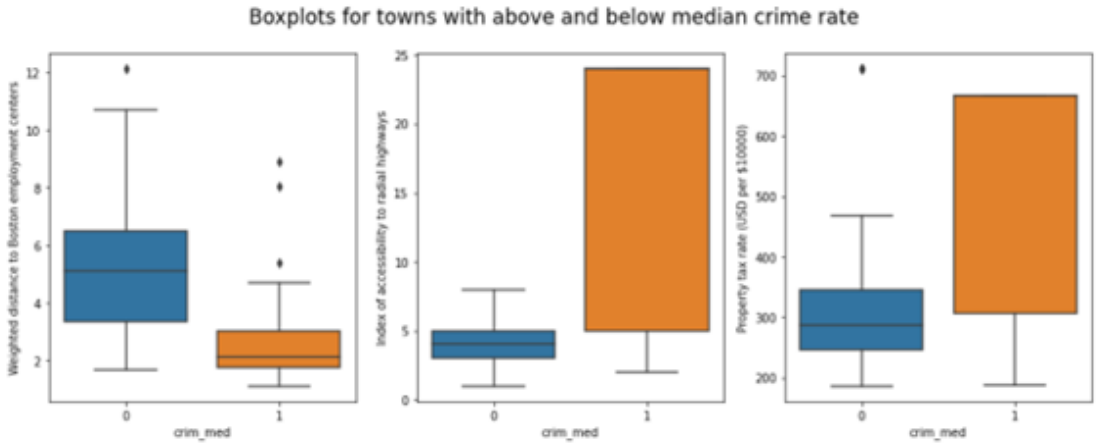
4. Plotting the data

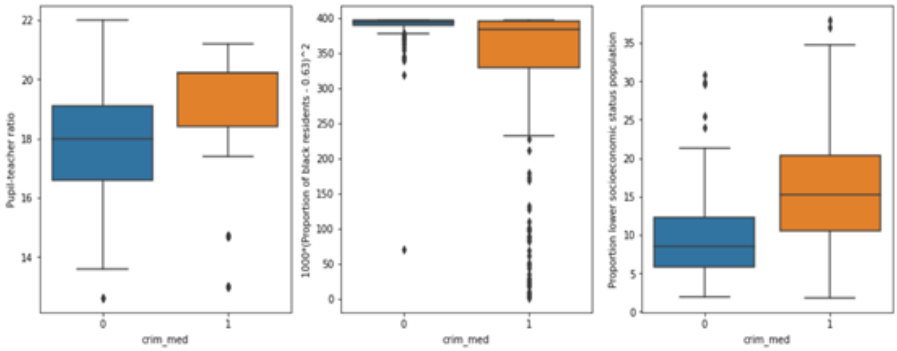
i. Box and Whiskers Plot

Box Plot is also known as Whisker plot is created to display the summary of the set of data values having properties like minimum, first quartile, median, third quartile and maximum. In the box plot,

a box is created from the first quartile to the third quartile, a vertical line is also there which goes through the box at the median. Here x-axis denotes the data to be plotted while the y-axis shows the frequency distribution.







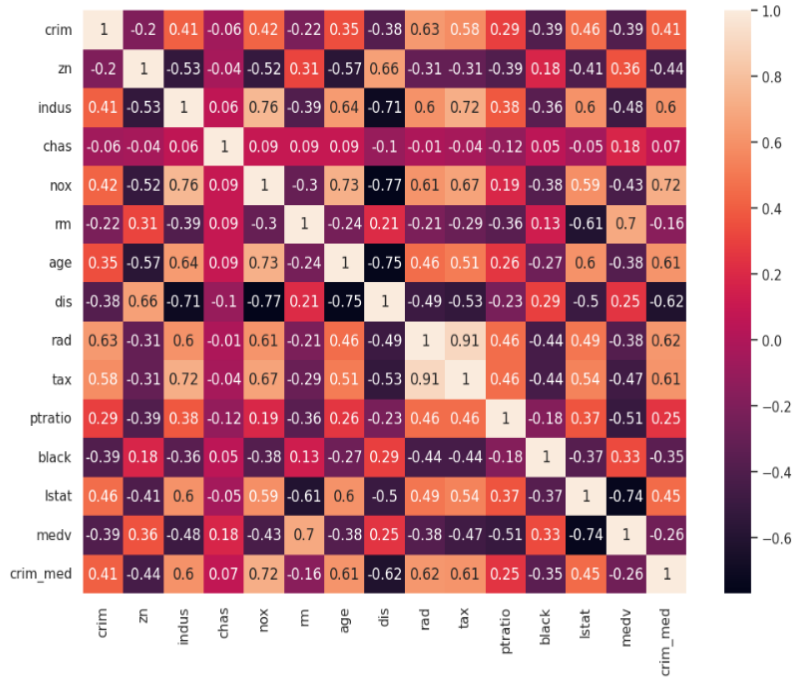
I. The ***twelve boxplots*** we produced provide further evidence to suggest that the predictors which are likely to be most useful in predicting crim_med are those having a correlation magnitude **of at least 0.3** with crim (**medv, indus, nox, age, dis, rad, tax, black, and lstat**). Those are the variables that show the most separation in distributions between the suburbs with above-median crime rates and those with below-median crime rates. In particular, for all of those variables except cmedv, it appears that the medians for the suburbs with above-median crime rates are well clear of the upper or lower quartile (depending on the particular variable) of the values for suburbs with below-median crime rates. Therefore, we will proceed further and split the data as test and train and then fix the predictors and delete the features which have multicollinearity.We can further see the heatmap:

II. **Heatmap**

Heatmap is defined as a graphical representation of data using colors to visualize the value of the matrix. In this, to represent more common values or higher activities brighter colors basically reddish colors are used and to represent less common or activity values, darker colors are preferred. Heatmap is also defined by the name of the shading matrix. Heatmaps in Seaborn can be plotted by using the ***seaborn.heatmap()*** function.

```
[29] #PLOTTING THE HEATMAP
     sns.set(rc={'figure.figsize':(11.7,8.27)})
     correlation_matrix=boston_data.corr().round(2)
     sns.heatmap(data=correlation_matrix,annot=True)
```

<Axes: >



5. Cleaning the data

Removing the correlated variables

As we observed in the heatmap, some columns have high correlation. So, we shall remove those columns. Multicollinearity occurs when there are two or more independent variables in a multiple regression model, which have a high correlation among themselves. When some features are highly correlated, we might have difficulty in distinguishing between their individual effects on the dependent variable. Multicollinearity can be detected using various techniques, one such technique being the **Variance Inflation Factor(VIF)**.

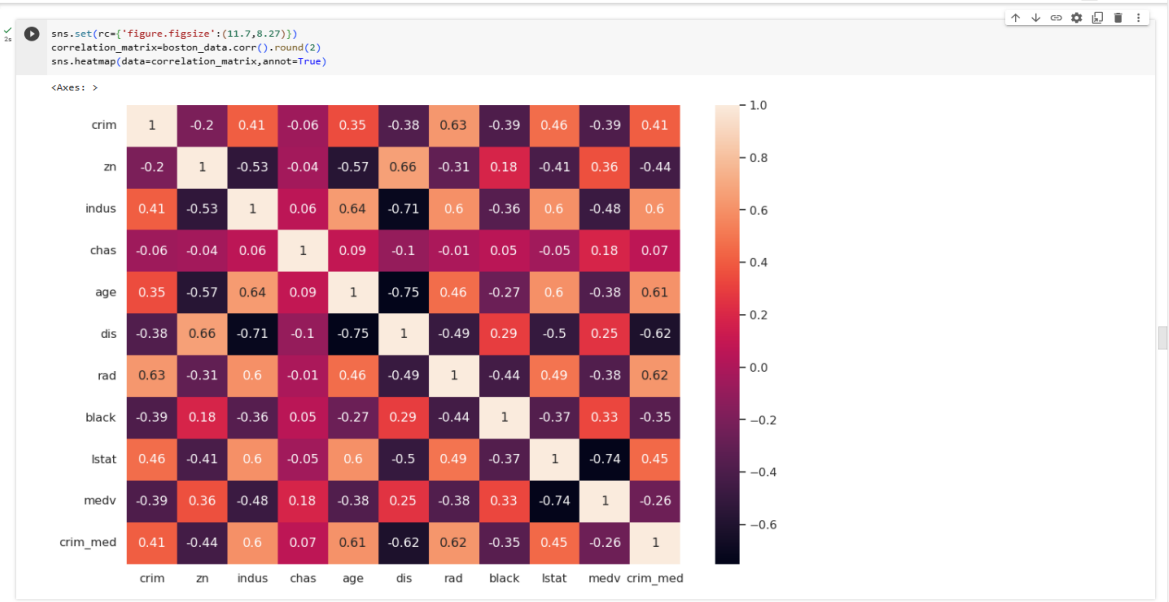
```
[30] from statsmodels.stats.outliers_influence import variance_inflation_factor
     # VIF dataframe
     vif_data = pd.DataFrame()
     vif_data["feature"] = boston_data.columns

     # calculating VIF for each feature
     vif_data["VIF"] = [variance_inflation_factor(boston_data.values, i)
                        for i in range(len(boston_data.columns))]

     print(vif_data)#TO SEE THE HIGHLY CORRELATED VALUES
```

```
      feature         VIF
0        crim    2.131428
1          zn    2.920967
2       indus   14.513748
3        chas    1.176340
4         nox   80.070400
5          rm  136.796242
6         age   21.903488
7         dis   15.462682
8         rad   16.256148
9         tax   62.190801
10    ptratio   87.691565
11      black   21.577664
12      lstat   12.627871
13       medv   24.689191
14   crim_med    4.915128
```

From this we can see that the columns ["rm","ptratio","nox","tax"])are highly correlated. So, we remove the columns ["rm","ptratio","nox","tax"]).Now, we again plot the co-relation matrix using the command “data.corr()”and observe that there are no more columns with high correlation.

```
[33] boston_data.corr()["crim"]#AFTER DELETING THE HIGHLY CORRELATED COLUMNS SEEING THE CORRELATION AGAIN
```

```
crim        1.000000
zn         -0.200469
indus       0.406583
chas       -0.055892
age         0.352734
dis        -0.379670
rad         0.625505
black      -0.385064
lstat       0.455621
medv       -0.388305
crim_med    0.409395
Name: crim, dtype: float64
```



Before training any models, we split our data in to a training set and a test set using the 75%-25% split.

6. Splitting of data into training and testing sets

Train_test_split() command is used to split the data into training and testing sets. We apply the machine learning model to the training set and test the model's accuracy by applying it on the testing set. Now that we have split the data into a training set and a test set, we fit **linear regression** models and **LDA models** to predict whether or not a given suburb has an above-median crime rate.

Let's frist divide the data into X and Y :

```
[38] #Divide the data into X and Y
     X = boston_data.drop(columns = ["crim_med"])
     Y = boston_data["crim_med"]
     print(X.shape)
     print(Y)

     (506, 10)
     0      0
     1      0
     2      0
     3      0
     4      0
           ..
     501    0
     502    0
     503    0
     504    0
     505    0
     Name: crim_med, Length: 506, dtype: int64
```

Now let's split the data into training and testing:

```
[39] # splitting the data 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 = 4)
```

7. Linear Regression

Linear regression is a basic and commonly used type of predictive analysis. The overall idea of regression is to examine two things:

(1) Predictor variables do a good job in predicting an outcome (dependent) variable

(2) Variables in that are significant predictors of the outcome variable

Given a data set of $\{y_i, x_{i1}, x_{i2}, \ldots x_{ip}\}^n_{i=1}$ n statistical units, a linear regression model assumes that the relationship between the dependent variable y and the p-vector of regressors x is linear.

Thus the model takes the form:

$$Y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_p x_{ip} + \varepsilon_i = x_i^T + \varepsilon_i$$

$$i = 1,2,3,\ldots,n$$

```
[40] # Linear Regression
     from sklearn.linear_model import LinearRegression

     model_reg = LinearRegression()
     model_reg = model_reg.fit(X_train,Y_train)
     Y_pred = model_reg.predict(X_test)
     Y_train_pred = model_reg.predict(X_train)
```

Here, we fit the model on training set and name it as “model_reg”. Then we test the model accuracy by applying the model on testing data and store it as “Y_pred”. Also, we test the model for training data and name it as “Y_train_pred”.

```
[41] for i in range(len(Y_pred)):
         if(Y_pred[i] < 0.5):
             Y_pred[i] = 0
         else:
             Y_pred[i] = 1
```

Since, the response variable is categorical, it takes the values only 0 and 1 but, by applying Linear regression model, we got the floating values between 0 and 1 so, we categorize the data by assigning the value 0 to the values less than 0.5 and 1 to the values greater than 0.5.
We perform the same technique on both the predicted outcomes i.e. “Y_pred” and “Y_train_pred”.

```
[44] # Now we do the same for Y_train
     for i in range(len(Y_train_pred)):
         if(Y_train_pred[i] < 0.5):
             Y_train_pred[i] = 0
         else:
             Y_train_pred[i] = 1
```

Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known.

```
[43] from sklearn.metrics import confusion_matrix
     cm = confusion_matrix(Y_test,Y_pred)
     print(cm)
     #CALCULATING THE ACCURACY OF LINEAR REGRESSION TESTING DATA
     accuracy = (cm[0][0]+cm[1][1])/(cm[0][0]+cm[1][1] + cm[0][1]+cm[1][0])
     print(accuracy)
     from sklearn import metrics
     cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm, display_labels = [False, True])

     cm_display.plot()
     plt.show()


     [[45  8]
      [11 38]]
     0.8137254901960784
```

Here, we observe that from true outcomes, 63 are predicted to be true whereas 45 are predicted as accurate and 11 are predicted wrongly. And from 46 false outcomes, 38 are predicted as correctly and 6 are predicted as wrongly. Thus, we can say that accuracy of the model is

Accuracy = (Total number of correct predictions)/(Total number of predictions)

= cm[0][0]+cm[1][1]/(cm[0][0]+cm[1][1]+ cm[0][1]+cm[1][0])

=45+38 /(45+38+8+11)

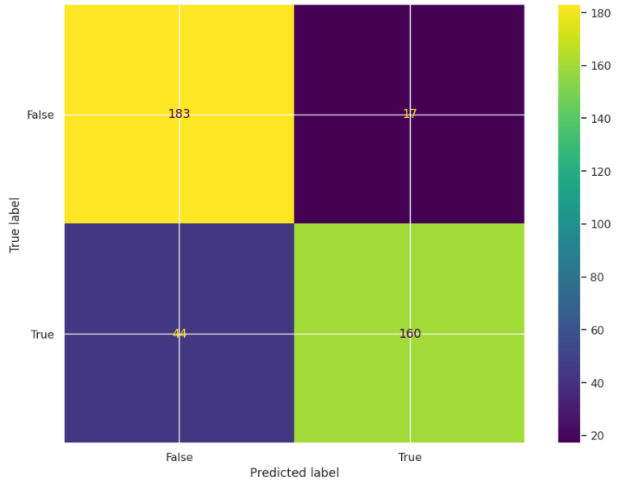
=83/102

=0.8137

Thus, the accuracy of the model is 81.37%

Now, we shall plot a confusion matrix for the training data

```
[45] cm_train = confusion_matrix(Y_train, Y_train_pred)
     print(cm_train)

     cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm_train, display_labels = [False, True])

     cm_display.plot()
     plt.show()

     [[183  17]
      [ 44 160]]
```



Here, we observe that for the training data, from 227 True values, 183 are predicted correctly whereas, 44 are predicted wrongly. From 117 False values, 180 are predicted correctly and 17 are predicted wrongly.

```
[46] #CALCULATING THE ACCURACY OF LINEAR REGRESSION TRAINING DATA
     accuracy = (cm_train[0][0]+cm_train[1][1])/(cm_train[0][0]+cm_train[1][1] + cm_train[0][1]+cm_train[1][0])
     print(accuracy)

     0.849009900990099
```

Here, we have calculated the accuracy for the training set. This comes out to be 84.90%

Regression coefficient:

```
[47] reg_coeff = model_reg.coef_
     reg_intercept = model_reg.intercept_
     print(reg_coeff)
     print(reg_intercept)

     [ 0.00037981 -0.00094907  0.01280105 -0.00412964  0.00525867 -0.01168553
       0.0195593  -0.00036874  0.00392519  0.00824843]
     -0.23925016148496625
```

We use the command “model_reg.coef_” to calculate the regression coefficients. These are the values for β_1,+ β_2 , ... , β_p. “model_reg.intercept_” gives the value for β_0. Thus forms the Linear Regression model.

Calculating R^2 score and Adjusted R^2 score:

```
from sklearn.metrics import mean_squared_error, r2_score
print("R2 score : %.2f" % r2_score(Y_test,Y_pred))
print("Mean squared error: %.2f" % mean_squared_error(Y_test,Y_pred))
```

8. Linear Discriminant Analysis

Linear Discriminant Analysis is a dimensionality reduction technique that is commonly used for supervised classification problems. It is used for modeling differences in groups i.e. separating two or more classes. It is used to project the features in higher dimension space into a lower dimension space.
Here, we normalize the training and testing data using fit_transform()

```
[48] # Linear Discriminant Analysis
     from sklearn.preprocessing import StandardScaler
     sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.fit_transform(X_test)
```

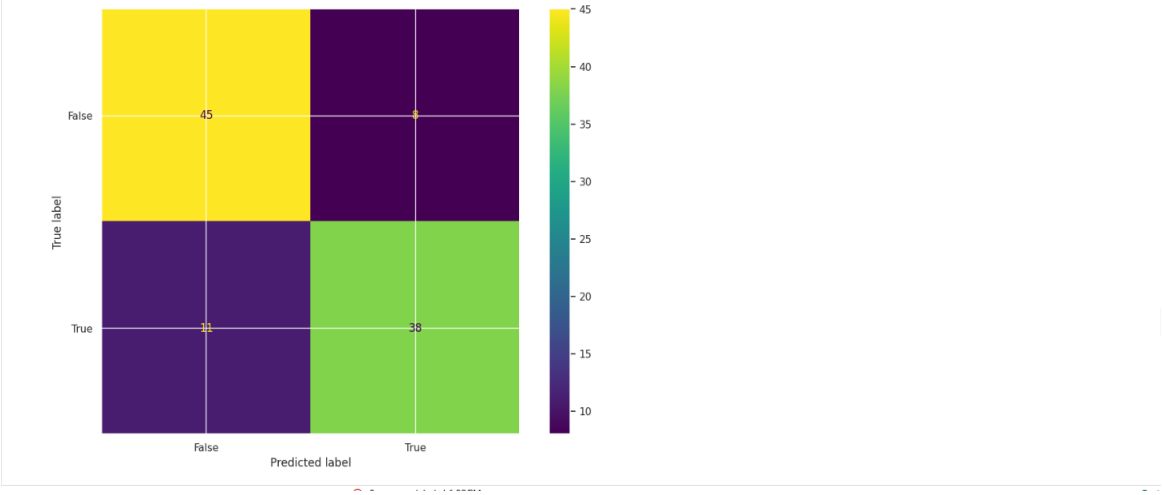
Fitting the Linear Discriminant model as LDA.

```
[49] #Fitting the LDA model
     from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
     model_LDA = LDA()
     model_LDA = model_LDA.fit(X_train,Y_train)
     Y_pred_LDA = model_LDA.predict(X_test)
     print(Y_pred_LDA)

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

Confusion Matrix:

```
#CONFUSION MATRIX FOR LDA MODEL
cm_LDA = confusion_matrix(Y_test,Y_pred_LDA)
print(cm_LDA)
from sklearn import metrics
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm_LDA, display_labels = [False, True])

cm_display.plot()
plt.show()
```

```
[[45  8]
 [11 38]]
```



Here, we observe that from true outcomes, 63 are predicted to be true whereas 45 are predicted as accurate and 11 are predicted wrongly. And from 46 false outcomes, 38 are predicted as correctly and 6 are predicted as wrongly. Thus, we can say that accuracy of the model is

Accuracy = (Total number of correct predictions)/(Total number of predictions)

= cm[0][0]+cm[1][1]/

cm[0][0]+cm[1][1]+ cm[0][1]+cm[1][0]

=45+38/(45+38+8+11)

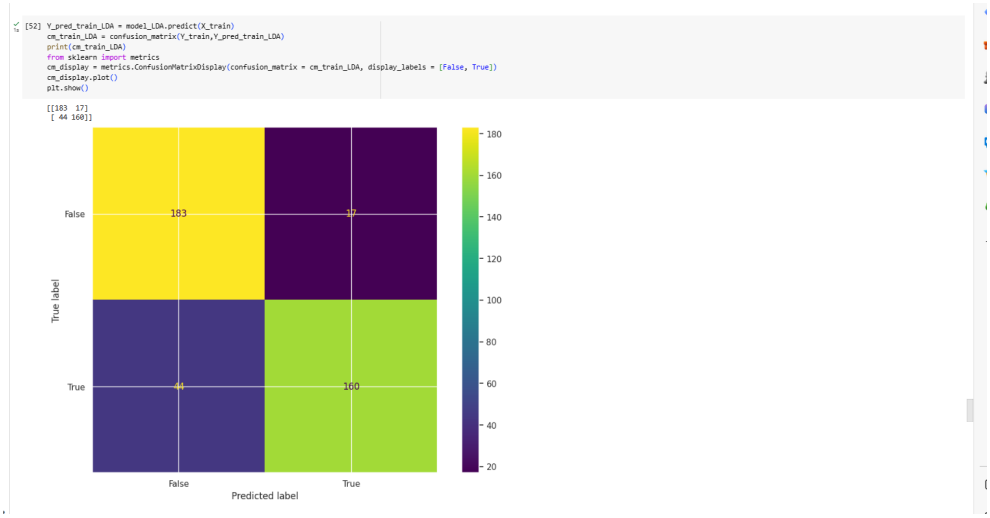
=83/102

=0.8137

Thus, the accuracy of the model is 81.37%

Now, we shall plot a confusion matrix for the training data

```
[52] Y_pred_train_LDA = model_LDA.predict(X_train)
     cm_train_LDA = confusion_matrix(Y_train,Y_pred_train_LDA)
     print(cm_train_LDA)
     from sklearn import metrics
     cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = cm_train_LDA, display_labels = [False, True])
     cm_display.plot()
     plt.show()

     [[183  17]
      [ 44 160]]
```



Here, we have calculated the accuracy for the training set. This comes out to be 84.90%

```
[53] #CALCULATING THE ACCURACY OF LDA TRAINING DATA
     accuracy_LDA = (cm_train_LDA[0][0]+cm_train_LDA[1][1])/(cm_train_LDA[0][0]+cm_train_LDA[1][1] + cm_train_LDA[0][1]+cm_train_LDA[1][0])
     print(accuracy_LDA)

     0.849009900990099
```

9. Conclusions

- ➢ Using **Linear Regression Model**, we get an accuracy of **81.37%**
- ➢ Using **Linear Discriminant Analysis**, we get an accuracy of **81.37%**
- ➢ Clearly, We can see that both the models give exactly the same accuracy so we can use any one of them for the Boston data.

Insurance Data

Dataset and Problem Statement:

To create a classification model for insurance data, predicting responses based on 13 parameters like age, city, region, insurance type, accommodation type, health, marital status, and policy features. We'll use two methods, QDA and KNN, to build the model, make predictions, compare their accuracies, and create confusion matrices.

Tasks Accomplished in Our Work

1.How the Data Look Like?

There are 13 parameters, and the response variable is the output. Before proceeding forward, we eliminate rows with missing data, such as NaN values.

ID	City_Code	Region_Code	Accomodation_Type	Reco_Insurance_Type	Upper_Age	Lower_Age	Is_Spouse	Health Indicator	Holding_Policy_Duration	Holding_Policy_Type	Reco_Policy_Cat
1	C3	3213	Rented	Individual	36	36	No	X1	14+	3.0	22
2	C5	1117	Owned	Joint	75	22	No	X2	NaN	NaN	22
3	C5	3732	Owned	Individual	32	32	No	NaN	1	1.0	19
4	C24	4378	Owned	Joint	52	48	No	X1	14+	3.0	19
5	C8	2190	Rented	Individual	44	44	No	X2	3	1.0	16
...	...	...	...	...	...	...	...	...	...	...	...
50878	C4	845	Rented	Individual	22	22	No	X3	NaN	NaN	18
50879	C5	4188	Rented	Individual	27	27	No	X3	7	3.0	4
50880	C1	442	Rented	Individual	63	63	No	X2	14+	1.0	12
50881	C1	4	Owned	Joint	71	49	No	X2	2	2.0	16
50882	C3	3866	Rented	Individual	24	24	No	X3	2	3.0	18

Table No1

2.Data Description: The data.describe() function is used to generate descriptive statistics of a dataset. It provides a summary that includes measures of central tendency, dispersion, and shape of the distribution of a dataset's values. This includes the count, mean, standard deviation, minimum, 25th percentile (Q1), median (50th percentile or Q2), 75th percentile (Q3), and maximum values for each numerical column in the dataset.

	ID	Region_Code	Upper_Age	Lower_Age	Holding_Policy_Type	Reco_Policy_Cat	Reco_Policy_Premium	Response
count	23548.000000	23548.000000	23548.000000	23548.000000	23548.000000	23548.000000	23548.000000	23548.000000
mean	25545.658060	1731.704051	48.864192	46.365381	2.437574	15.207364	15409.000161	0.242059
std	14656.637655	1435.834257	16.021466	16.578403	1.025915	6.326014	6416.327319	0.428339
min	1.000000	1.000000	21.000000	16.000000	1.000000	1.000000	3216.000000	0.000000
25%	12837.500000	516.000000	35.000000	32.000000	1.000000	12.000000	10704.000000	0.000000
50%	25617.500000	1381.000000	49.000000	46.000000	3.000000	17.000000	14580.000000	0.000000
75%	38167.000000	2672.000000	62.000000	60.000000	3.000000	20.000000	19140.000000	0.000000
max	50882.000000	6194.000000	75.000000	75.000000	4.000000	22.000000	43350.400000	1.000000

Table No 2

3.Correlation Between Parameters:

The code ‘sns.heatmap(data.corr())’ is used to create a heatmap in the Seaborn library, which visualizes the correlation matrix of a dataset. This heatmap provides a color-coded representation of the correlation coefficients between pairs of variables in the dataset. In the figure we can see the dark color indicates a strong negative correlation, strong positive correlation, while shades of white represent weaker or no correlation.

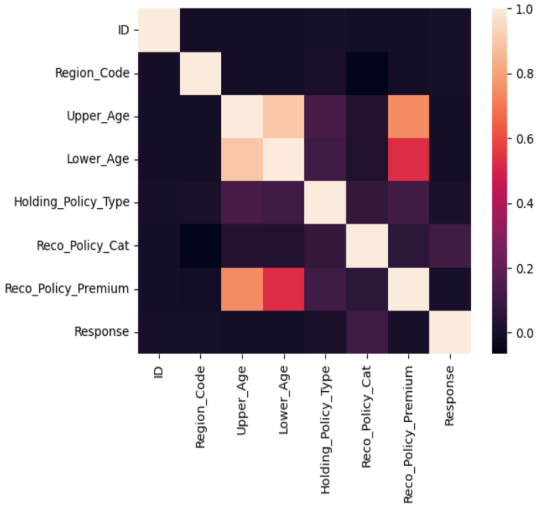


Figure 1

4. Histogram of Each Feature:

We got each plot representing each attribute and its corresponding histogram.

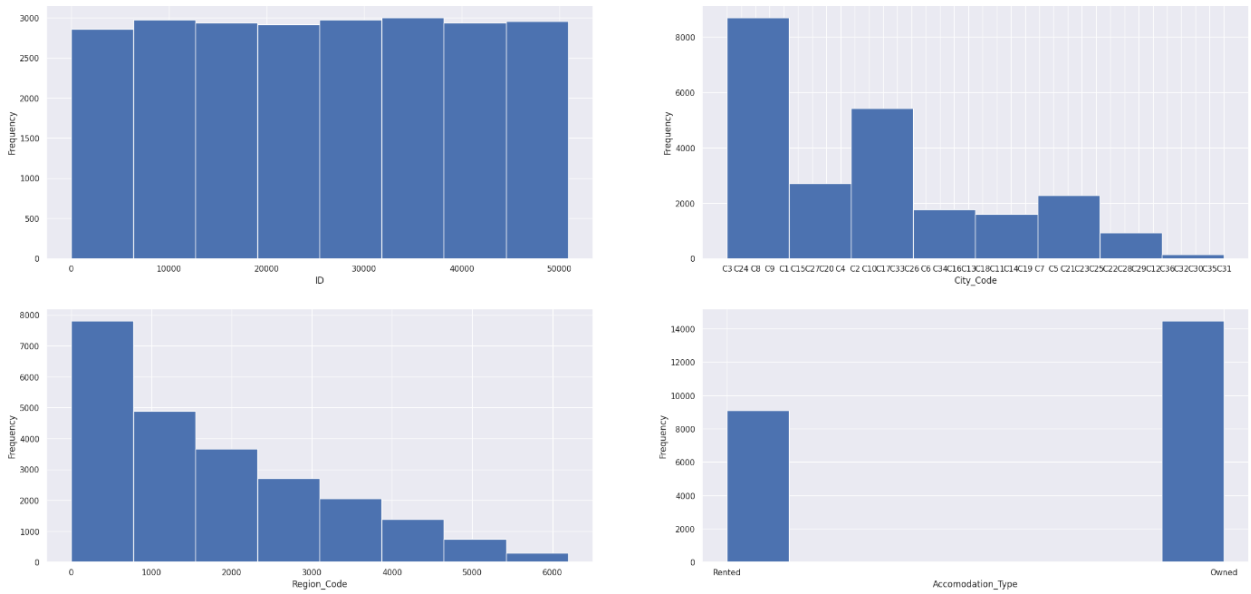


Figure 2: (Frequency VS ID, City Code, Region Code, Accomodation Type)

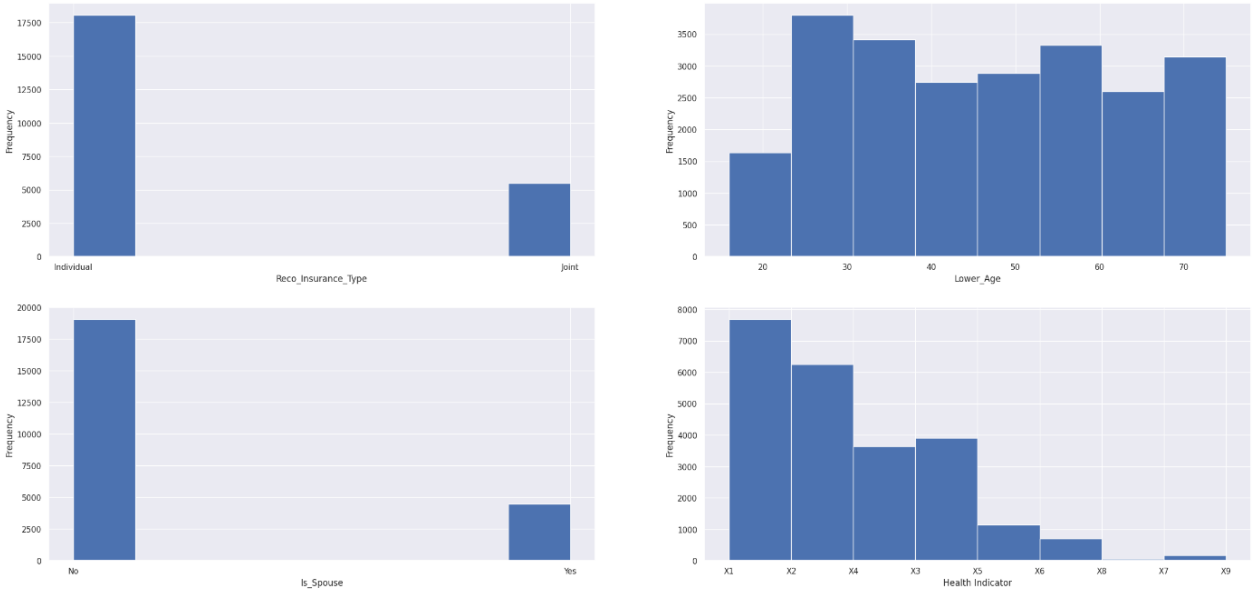


Figure 3: (Frequency VS Reco_insurance, Lower age, Is_spouse, Health Indicator)

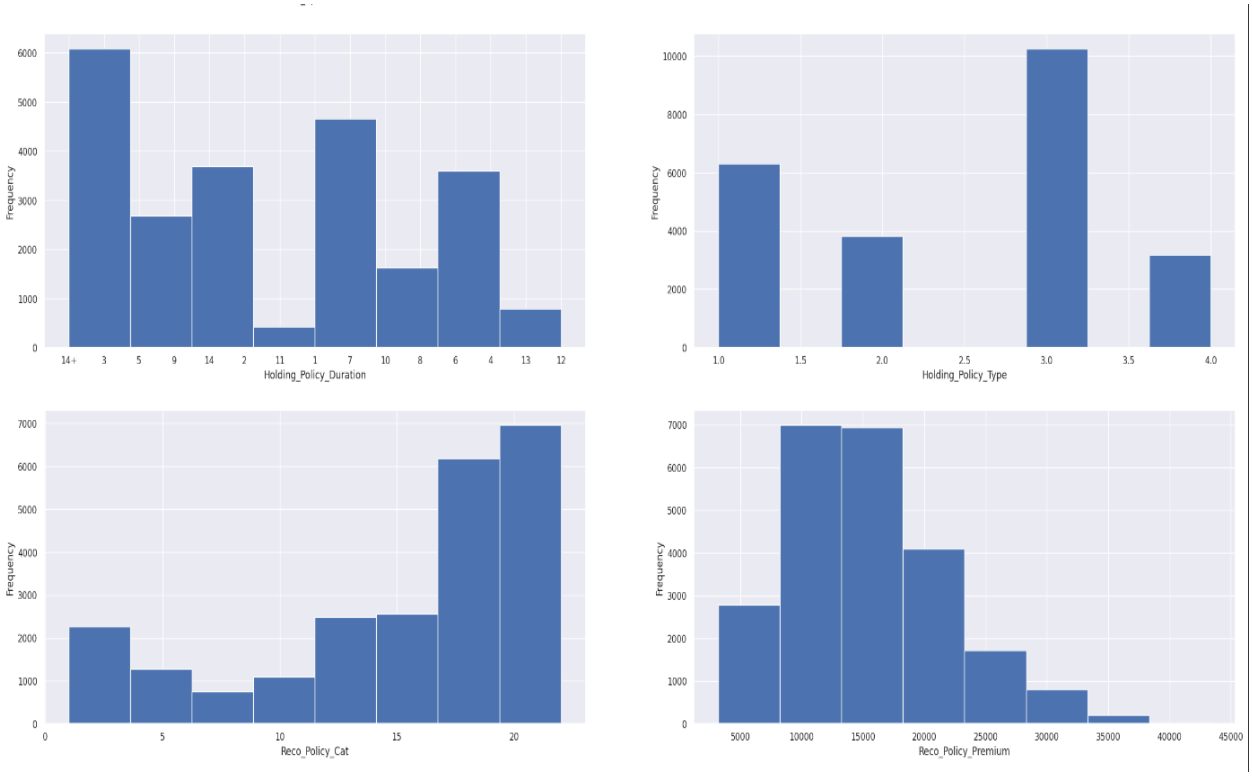


Figure 4: (Frequency VS Holding Policy Duration, Holding policy Type, Reco_Policy Cat, Perco_Policy_Premium)

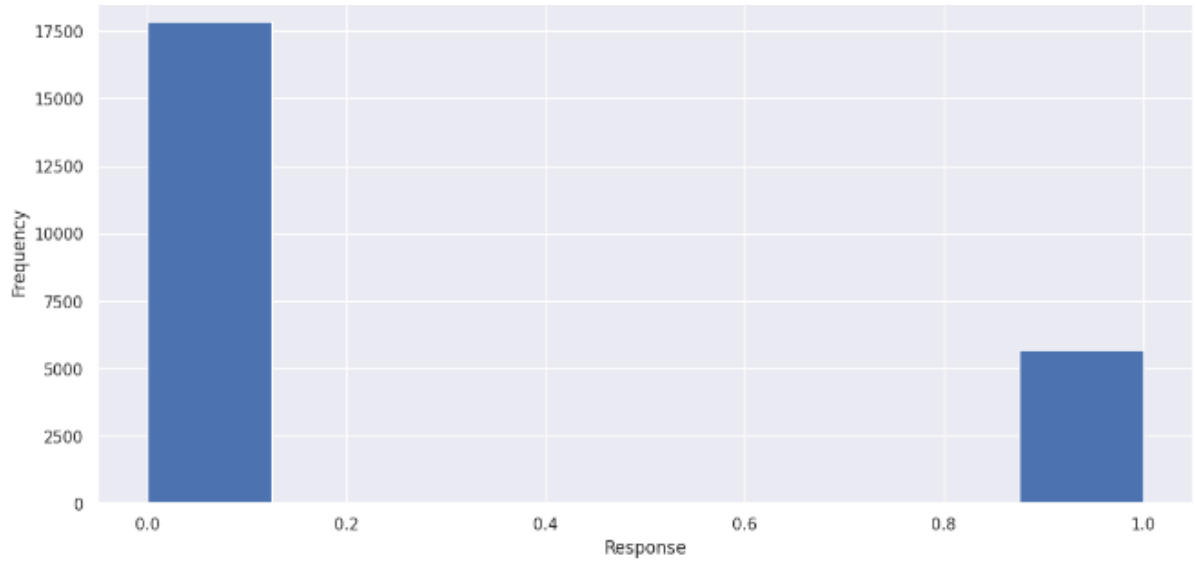


Figure 5: (Frequency VS Response)

5. Data Transformation:

1. After removing rows with missing values, we proceed to eliminate columns with significant correlation. For instance, we exclude "Upper_Age" due to its strong correlation with "Lower Age," resulting in an observed improvement in accuracy.
2. Now, we split the data into input X and output Y and encode the categorical variables so that they can be computed, using label encoder.
3. Then, we divide the dataset into training and testing sets, and then apply feature scaling using StandardScaler() to standardize the variance.
4. Then, we utilize the training data to train the models and employ the testing data for predictions, assessing accuracy in the process.

6. Quadratic Discriminant Analysis (QDA):

$$\delta_k(x) = \log \pi_k - \tfrac{1}{2}\mu_k^T \mathbf{\Sigma}_k^{-1}\mu_k + x^T \mathbf{\Sigma}_k^{-1}\mu_k - \tfrac{1}{2}x^T \Sigma_k^{-1} x - \tfrac{1}{2}\log|\Sigma_k|$$

The above formula gives the quadratic discriminant function for each class, and the formula below gives the covariance matrix for the class y.

- πk refers to P(Y=k), the prior probability
- $\boldsymbol{\mu}$k refers to the mean of the examples of category k
- $\mathbf{\Sigma}$k refers to the covariance matrix of class k
- We calculate the values of the QDA function for each class, depending on which we predict which class the input x belongs to

$$\Sigma_y = \frac{1}{N_y - 1} \sum_{y_i = y} (x_i - \mu_y)(x_i - \mu_y)^T$$

Confusion Matrix for QDA: This is the confusion matrix for the QDA, and we got a prediction accuracy of 74.32%

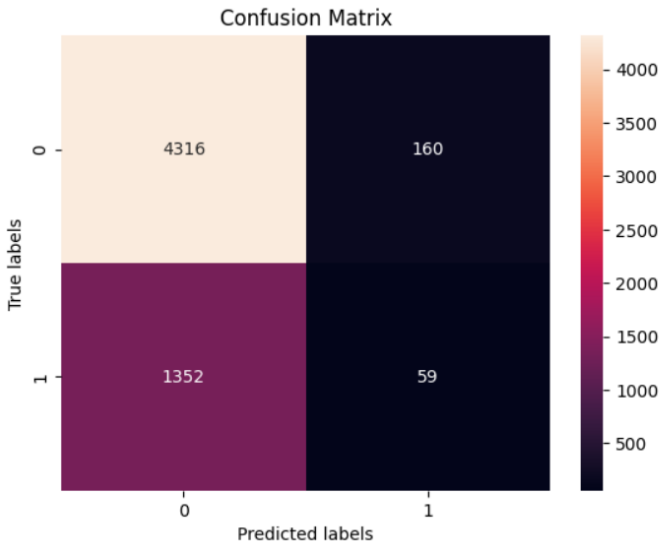


Figure 2

7. K-nearest neighbors (KNN):

In this algorithm, we compute the distance between specific data points to determine the class to which the observation belongs. We take the k nearest observation i.e. we pick the k nearest neighbors of our corresponding observation. Within these K nearest neighbors, we examine the class in which a greater number of these K neighbors are situated. We predict the same class for our observation. K can be varied in algorithm along with how we measure distance(p). Upon trying for each value of k and p, maximum accuracy was obtained for k=14, p=2.

Confusion Matrix for KNN: Confusion matrix for KNN model with k=14, p=2. Accuracy of 75.5% was obtained.

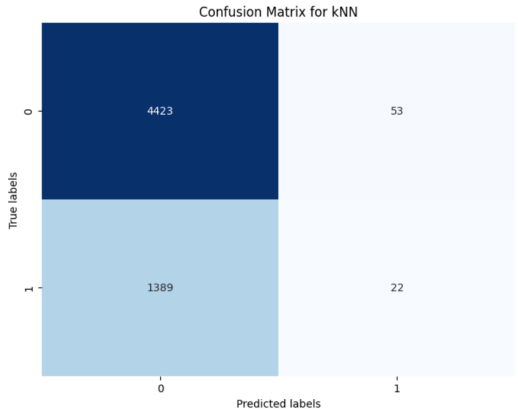
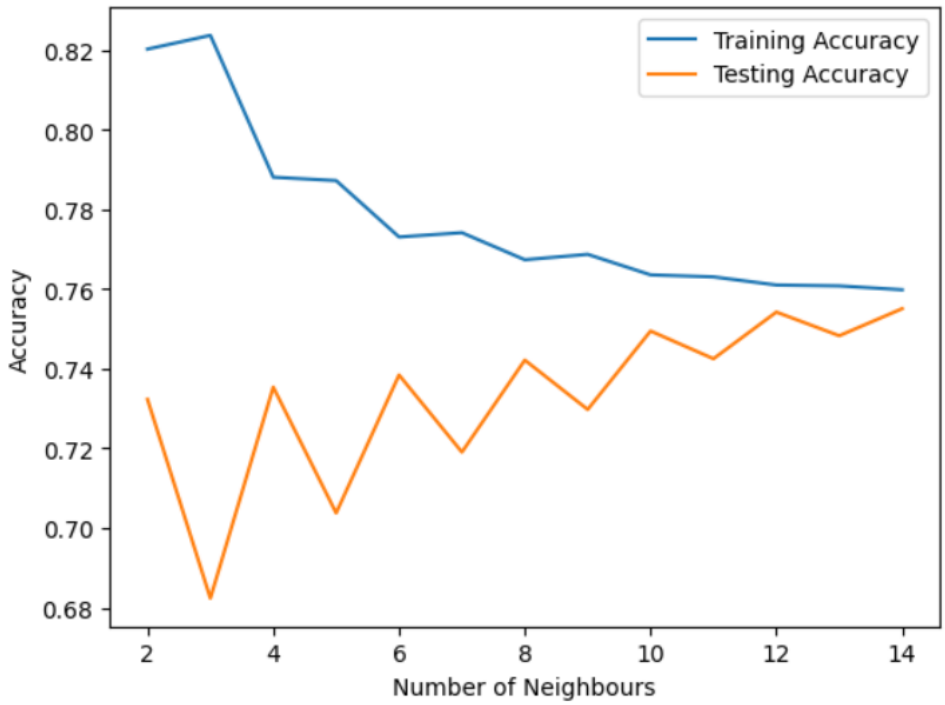


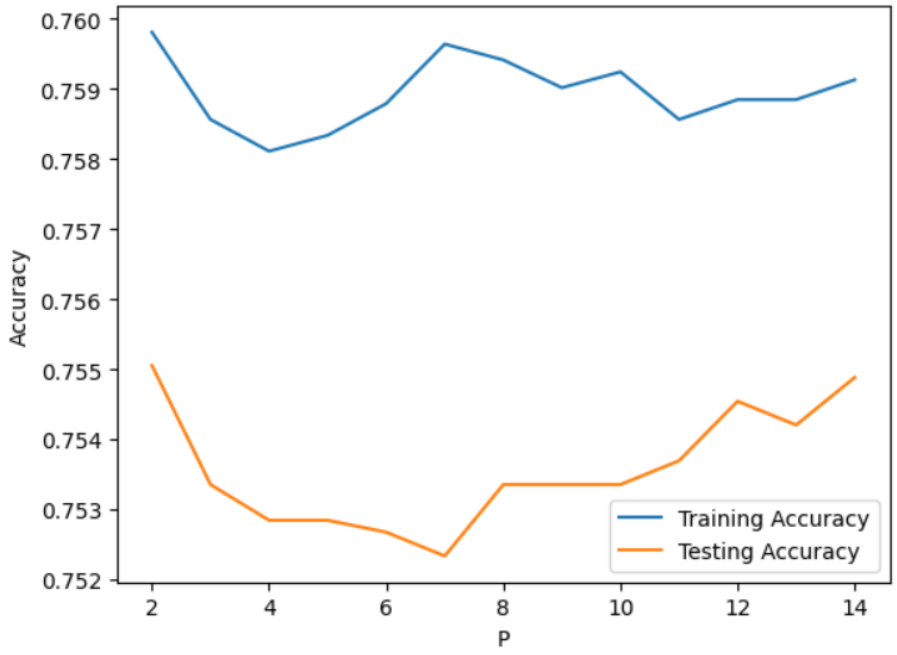
Figure 3

8.Selecting best value of K and P



Graph 1 :Variation of accuracy with K

Graph 1 illustrates how the testing accuracy changes with the variation in the number of neighbors (k). As the number of neighbors increases, the testing accuracy demonstrates an upward trend.



Graph 2: Variation of accuracy with P

The accuracy changes with different values of P. The testing accuracy is highest when p is equal to 2, but it also shows an increase for larger values of P.

9.Conclusion

- QDA and KNN were used on the provided insurance dataset to predict the response variable.
- QDA yielded an accuracy of 74.32%.
- In the case of KNN, various combinations of k and p were tested, and the highest accuracy (75.5%) was achieved with k=14 and p=2.
- Thus, based on the accuracies, the KNN model was better at predicting with test data set