MA515 Foundation To Data Science Project Assignment

Aryan Bansal (2020EEB1162)

Submitted To: Dr. Arun Kumar

Contents

AIM

The aim of this project is to do exploratory data analysis on the data. Use linear regression and QDA classification techniques to predict the sales for CH or MM. And compare the methods.

Exploratory Data Analysis

Exploratory Data Analysis (EDA) is an approach to analyzing data sets to summarize their main characteristics.

We first start by finding the shape of the data. The data is 18-parameter data with 1070 data points. Our objective is to predict the value of the Purchase Parameter(Column 1) whether it belongs to the CH class or the MM class.

Checking the Shape and Decription of data

```
[4] data.shape

    (1070, 18)
```

Checking the Datatype we find that Purchase and Store& are strings and the rest are integer or float.

Checking the Datayypes of various Parameters/Columns

```
data.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1070 entries, 0 to 1069
Data columns (total 18 columns):
 #   Column          Non-Null Count  Dtype
---  ------          --------------  -----
 0   Purchase        1070 non-null   object
 1   WeekofPurchase  1070 non-null   int64
 2   StoreID         1070 non-null   int64
 3   PriceCH         1070 non-null   float64
 4   PriceMM         1070 non-null   float64
 5   DiscCH          1070 non-null   float64
 6   DiscMM          1070 non-null   float64
 7   SpecialCH       1070 non-null   int64
 8   SpecialMM       1070 non-null   int64
 9   LoyalCH         1070 non-null   float64
 10  SalePriceMM     1070 non-null   float64
 11  SalePriceCH     1070 non-null   float64
 12  PriceDiff       1070 non-null   float64
 13  Store7          1070 non-null   object
 14  PctDiscMM       1070 non-null   float64
 15  PctDiscCH       1070 non-null   float64
 16  ListPriceDiff   1070 non-null   float64
 17  STORE           1070 non-null   int64
dtypes: float64(11), int64(5), object(2)
memory usage: 150.6+ KB
```

So we will convert those 2 into numerical values for Analysis.

I map CH to 1 and MM to 0 in the Purchase Column.

YES to 1 and NO to 0 in the Store7 column and then convert all the data types in the DATA to float.

Changing CH =1 and MA = 0 in Purchase & YES=1 and NO=0 in Store7

```
[9] data['Purchase'] = data['Purchase'].apply(lambda x: 1 if x == 'CH' else 0)

[10] data['Store7'] = data['Store7'].apply(lambda x: 1 if x == 'Yes' else 0)
```

I check for NULL values and if any NULL value is there, I will remove that datapoint. In the given data, there is no NULL value, hence no need to drop any data point.

Checking if Null values are there and if present , removing them

```
data.isnull().sum()
```

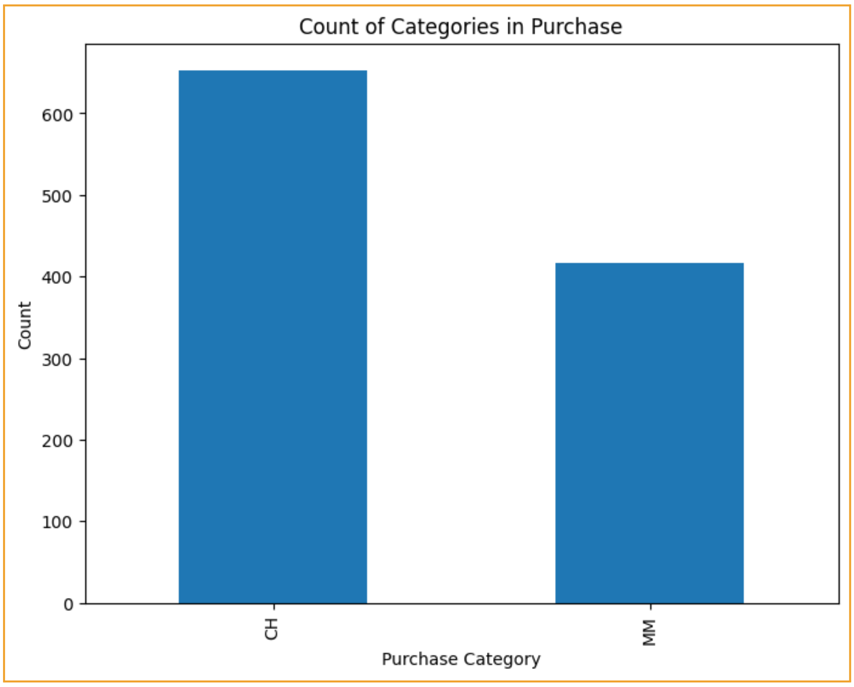
```
Purchase         0
WeekofPurchase   0
StoreID          0
PriceCH          0
PriceMM          0
DiscCH           0
DiscMM           0
SpecialCH        0
SpecialMM        0
LoyalCH          0
SalePriceMM      0
SalePriceCH      0
PriceDiff        0
Store7           0
PctDiscMM        0
PctDiscCH        0
ListPriceDiff    0
STORE            0
dtype: int64
```

I used the describe() function in Pandas to provide descriptive statistics of the dataframe.

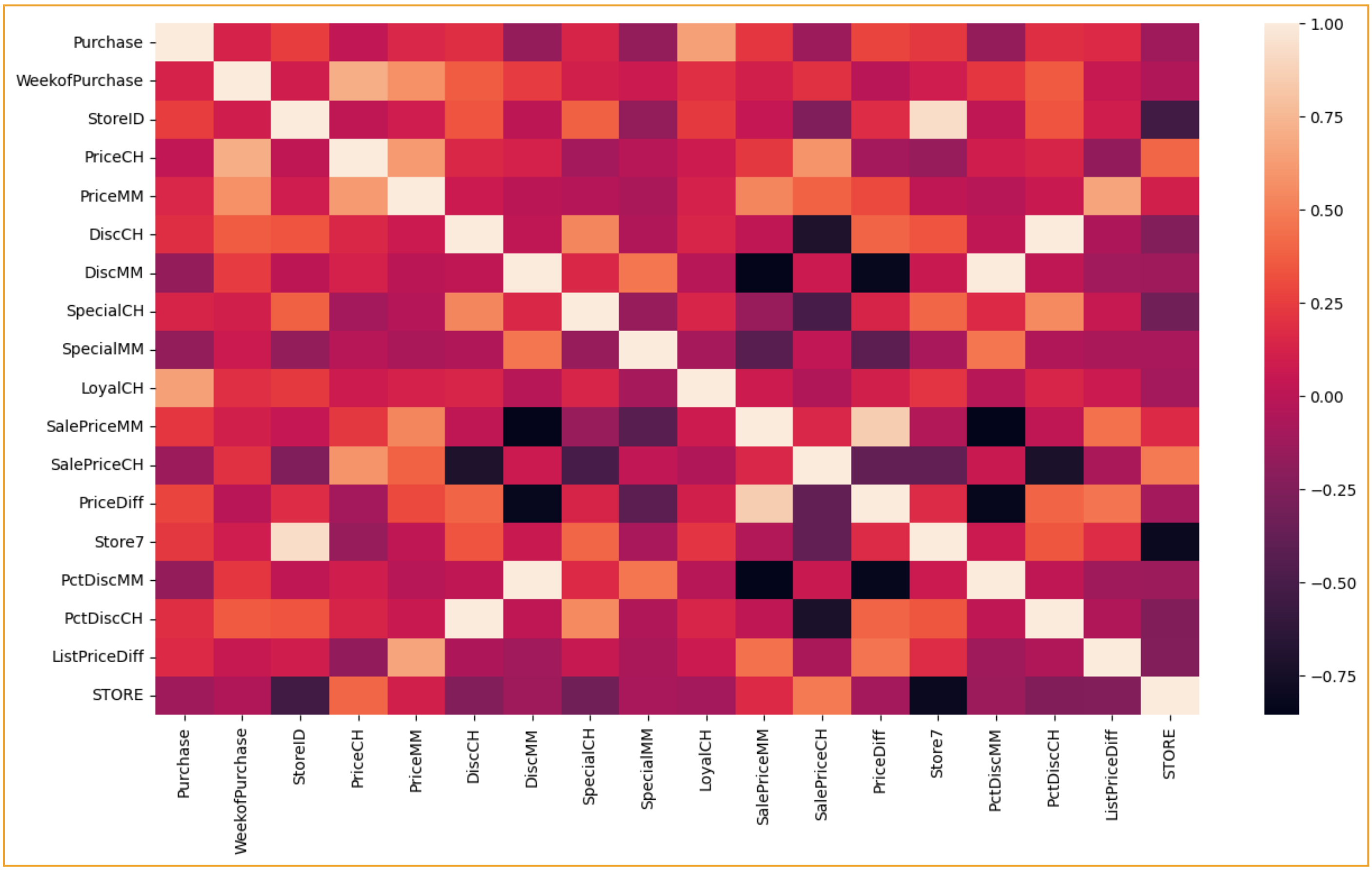
```
[5] data.describe()
```

	WeekofPurchase	StoreID	PriceCH	PriceMM	DiscCH	DiscMM	SpecialCH	SpecialMM	LoyalCH	SalePriceMM	SalePriceCH	PriceDiff	PctDiscMM	PctDiscCH	ListPriceDiff	STORE
count	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000	1070.000000
mean	254.381308	3.959813	1.867421	2.085411	0.051860	0.123364	0.147664	0.161682	0.565782	1.962047	1.815561	0.146486	0.059298	0.027314	0.217991	1.630841
std	15.558286	2.308984	0.101970	0.134386	0.117474	0.213834	0.354932	0.368331	0.307843	0.252697	0.143384	0.271563	0.101760	0.062232	0.107535	1.430387
min	227.000000	1.000000	1.690000	1.690000	0.000000	0.000000	0.000000	0.000000	0.000011	1.190000	1.390000	-0.670000	0.000000	0.000000	0.000000	0.000000
25%	240.000000	2.000000	1.790000	1.990000	0.000000	0.000000	0.000000	0.000000	0.325257	1.690000	1.750000	0.000000	0.000000	0.000000	0.140000	0.000000
50%	257.000000	3.000000	1.860000	2.090000	0.000000	0.000000	0.000000	0.000000	0.600000	2.090000	1.860000	0.230000	0.000000	0.000000	0.240000	2.000000
75%	268.000000	7.000000	1.990000	2.180000	0.000000	0.230000	0.000000	0.000000	0.850873	2.130000	1.890000	0.320000	0.112676	0.000000	0.300000	3.000000
max	278.000000	7.000000	2.090000	2.290000	0.500000	0.800000	1.000000	1.000000	0.999947	2.290000	2.090000	0.640000	0.402010	0.252688	0.440000	4.000000

As this is a categorization problem, I checked for the count of various categories in Purchase and plotted them.



Then I plotted the Heatmap of the Correlation Matrix of data.



As we can see from the heatmap there is the correlation between various parameters we can't use the data as it is because then it will produce spurious results. Multicollinearity can lead to unstable and unreliable coefficient estimates, making it harder to interpret the results and draw meaningful conclusions from the model. Hence we need to handle the problem of MultiColinearity.

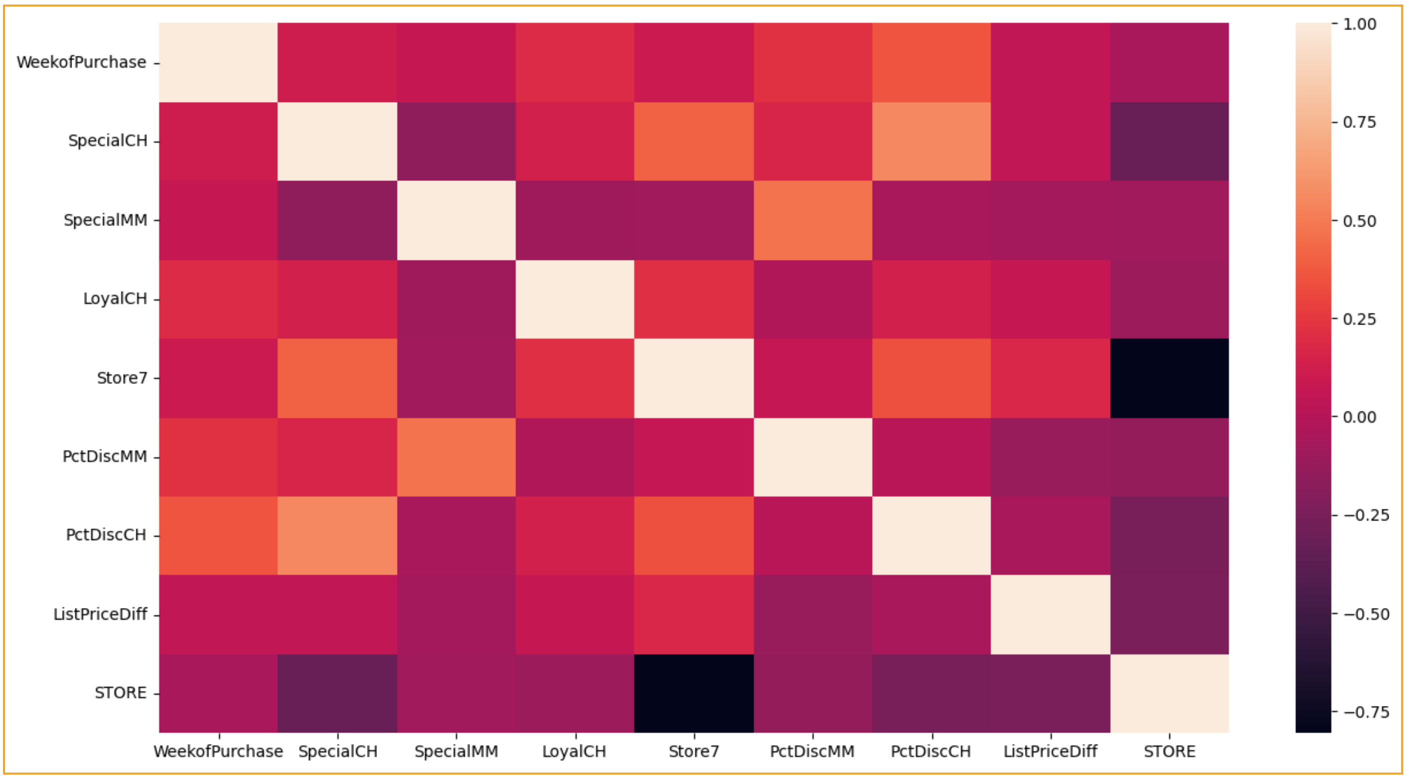
I solved it using the Variance Inflation Factor (VIF). VIF determines the strength of the correlation between the independent variables. It is predicted by taking a variable and regressing it against every other variable. From the list of variables, we select the variables with high VIF as collinear variables and drop those variables to get data without multi-colinearity.

```
def calculate_vif_(X, thresh):
    X = X.assign(const=1)  # faster than add_constant from statsmodels
    variables = list(range(X.shape[1]))
    dropped = True
    while dropped:
        dropped = False
        vif = [variance_inflation_factor(X.iloc[:, variables].values, ix)
               for ix in range(X.iloc[:, variables].shape[1])]
        vif = vif[:-1]  # don't let the constant be removed in the loop.
        maxloc = vif.index(max(vif))
        if max(vif) > thresh:
            print('dropping \'' + X.iloc[:, variables].columns[maxloc] +
                  '\' at index: ' + str(maxloc))
            del variables[maxloc]
            dropped = True

    print('Remaining variables:')
    print(X.columns[variables[:-1]])
    return X.iloc[:, variables[:-1]]
```

```
[84] X = calculate_vif_(X,7)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/stats/outliers_influence.py:198: RuntimeWarning: divide by zero encountered in double_scalars
  vif = 1. / (1. - r_squared_i)
dropping 'StoreID' at index: 1
dropping 'PriceCH' at index: 1
dropping 'PriceMM' at index: 1
dropping 'DiscCH' at index: 1
dropping 'SalePriceMM' at index: 5
dropping 'PriceDiff' at index: 6
dropping 'DiscMM' at index: 1
dropping 'SalePriceCH' at index: 4
Remaining variables:
Index(['WeekofPurchase', 'SpecialCH', 'SpecialMM', 'LoyalCH', 'Store7',
       'PctDiscMM', 'PctDiscCH', 'ListPriceDiff', 'STORE'],
      dtype='object')
```

From that, we get 'WeekofPurchase', 'SpecialCH', 'SpecialMM', 'LoyalCH', 'Store7', 'PctDiscMM', 'PctDiscCH', 'ListPriceDiff' and 'STORE' as the remaining variables. We again make a heat map to check for Multicollinearity.

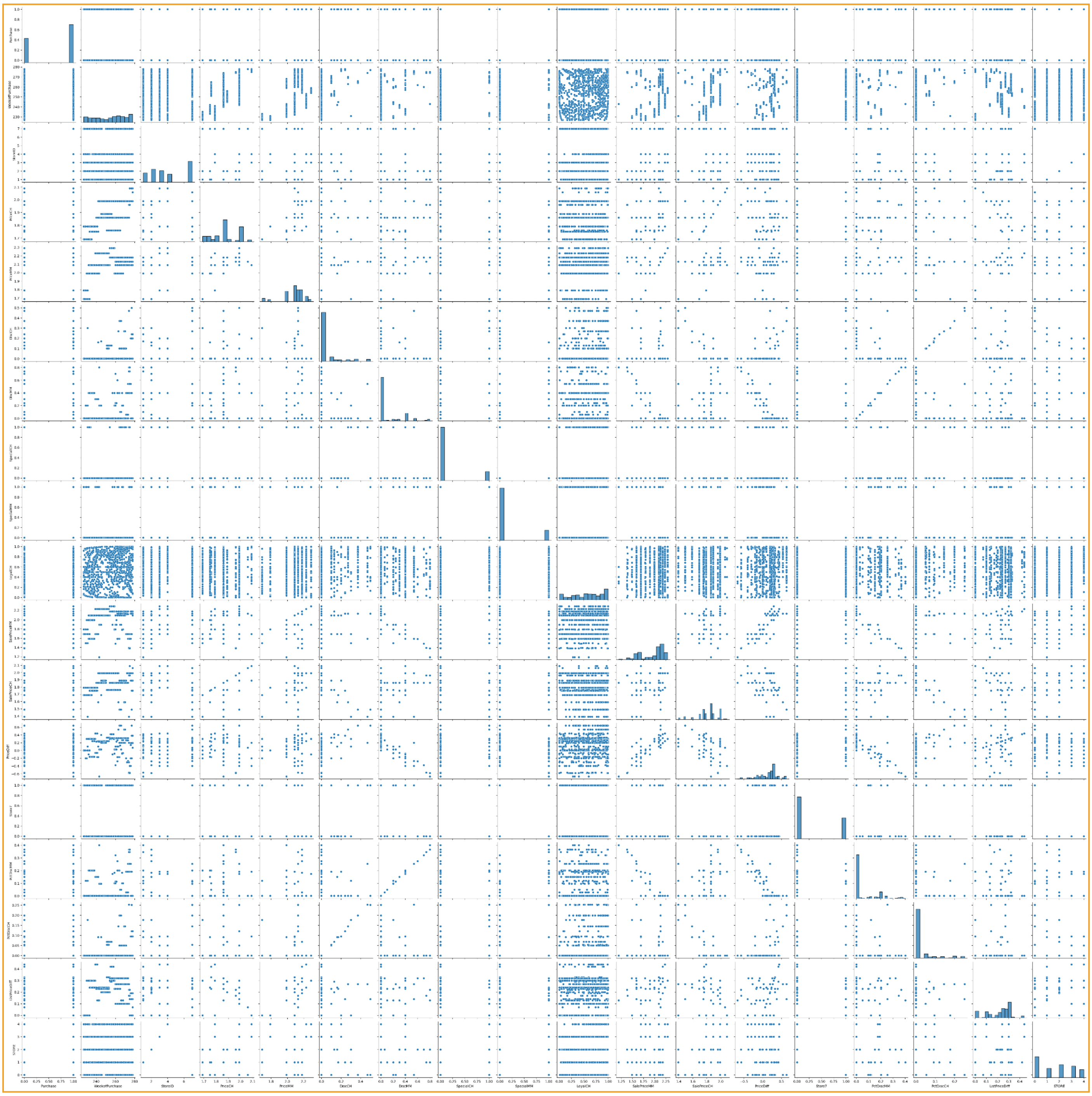


I also checked the measures of central tendency in the new input data.

Checking Measures of centeral Tendency on the new Input data

```
print(X.mean())
print(X.var())
print(X.skew())
print(X.kurtosis())
```

```
WeekofPurchase    254.381308
SpecialCH           0.147664
SpecialMM           0.161682
LoyalCH             0.565782
Store7              0.332710
PctDiscMM           0.059298
PctDiscCH           0.027314
ListPriceDiff       0.217991
STORE               1.630841
dtype: float64
WeekofPurchase    242.060268
SpecialCH           0.125977
SpecialMM           0.135668
LoyalCH             0.094767
Store7              0.222222
PctDiscMM           0.010355
PctDiscCH           0.003873
ListPriceDiff       0.011564
STORE               2.046008
dtype: float64
WeekofPurchase   -0.210990
SpecialCH         1.989092
SpecialMM         1.840470
LoyalCH          -0.278889
Store7            0.711080
PctDiscMM         1.539491
PctDiscCH         2.422300
ListPriceDiff    -0.645208
STORE             0.249493
dtype: float64
WeekofPurchase   -1.274927
SpecialCH         1.960147
SpecialMM         1.389925
LoyalCH          -1.059837
Store7           -1.497167
```



I used pairplots as a powerful visualization tool in my exploratory data analysis to gain insights into the relationships between different variables in the dataset. Pairplots enabled me to examine scatter plots for numerical variables and histograms for individual variables at the same time, giving me a comprehensive overview of the data distribution and potential patterns. I was able to discern trends and patterns specific to different groups within the data by color-coding the plots based on specific categorical variables.

Making Linear Regression and Analysis

I split the data into testing and training data with testing data having 321 data points and the rest in training data. After fitting the linear regression model on training data, I tested it on testing data for various thresholds from o to 1 with a stepsize of 0.05 and found the threshold to be 0.5 which gave the best accuracy. for that threshold, I tested the prediction on both training and testing data and got an accuracy as shown.

Threshold Value for which it has maximum Acccuracy

```
[47] threshold_test

     0.5
```

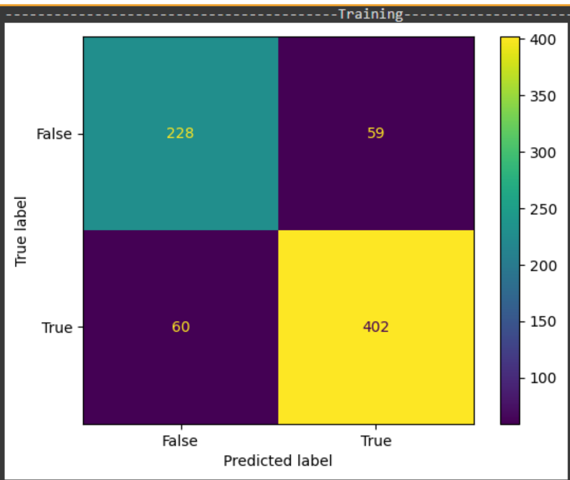
Accuracy Score for Training and testing data

```
[48] print("Accuracy of Linear Regression Model on Testing Data is = ",accuracy_score(test_Y, best_pred_test))
     print("Accuracy of Linear Regression Model on Training Data is =" , accuracy_score(train_Y, best_pred_train))

     Accuracy of Linear Regression Model on Testing Data is =  0.8099688473520249
     Accuracy of Linear Regression Model on Training Data is = 0.8411214953271028
```

Confusion Matrix and Accuracy

I Built the confusion matrix for both training and testing testing data with a threshold = 0.5.

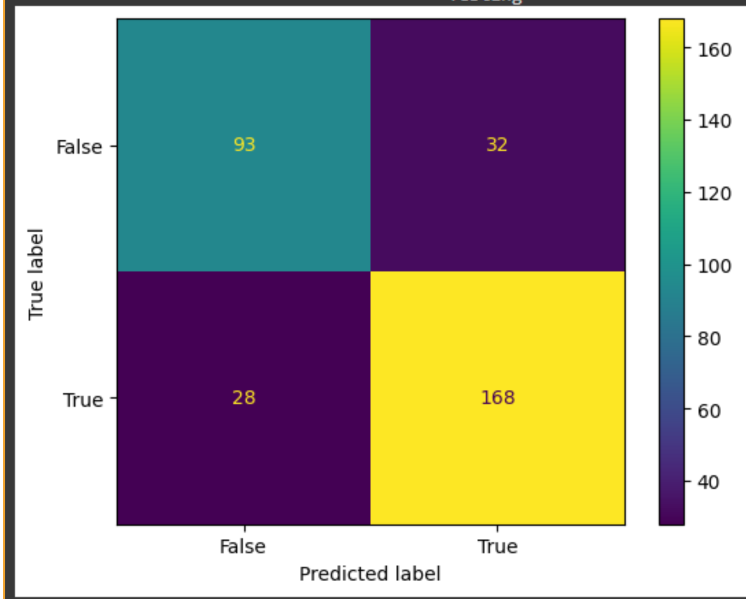
```
------------------------------------Training------------------------------------
```




```
Accuracy of Linear Regression Model on Training Data is : 0.8411214953271028

 Classification report on Training Data is :
               precision    recall  f1-score   support

           0       0.79      0.79      0.79       287
           1       0.87      0.87      0.87       462

    accuracy                           0.84       749
   macro avg       0.83      0.83      0.83       749
weighted avg       0.84      0.84      0.84       749
```

For Training data, I got an accuracy of 84.1121 % and for Testing data I got an accuracy of 81.3084 %.

```
----------------------------------------Testing---------------------------------------------
```

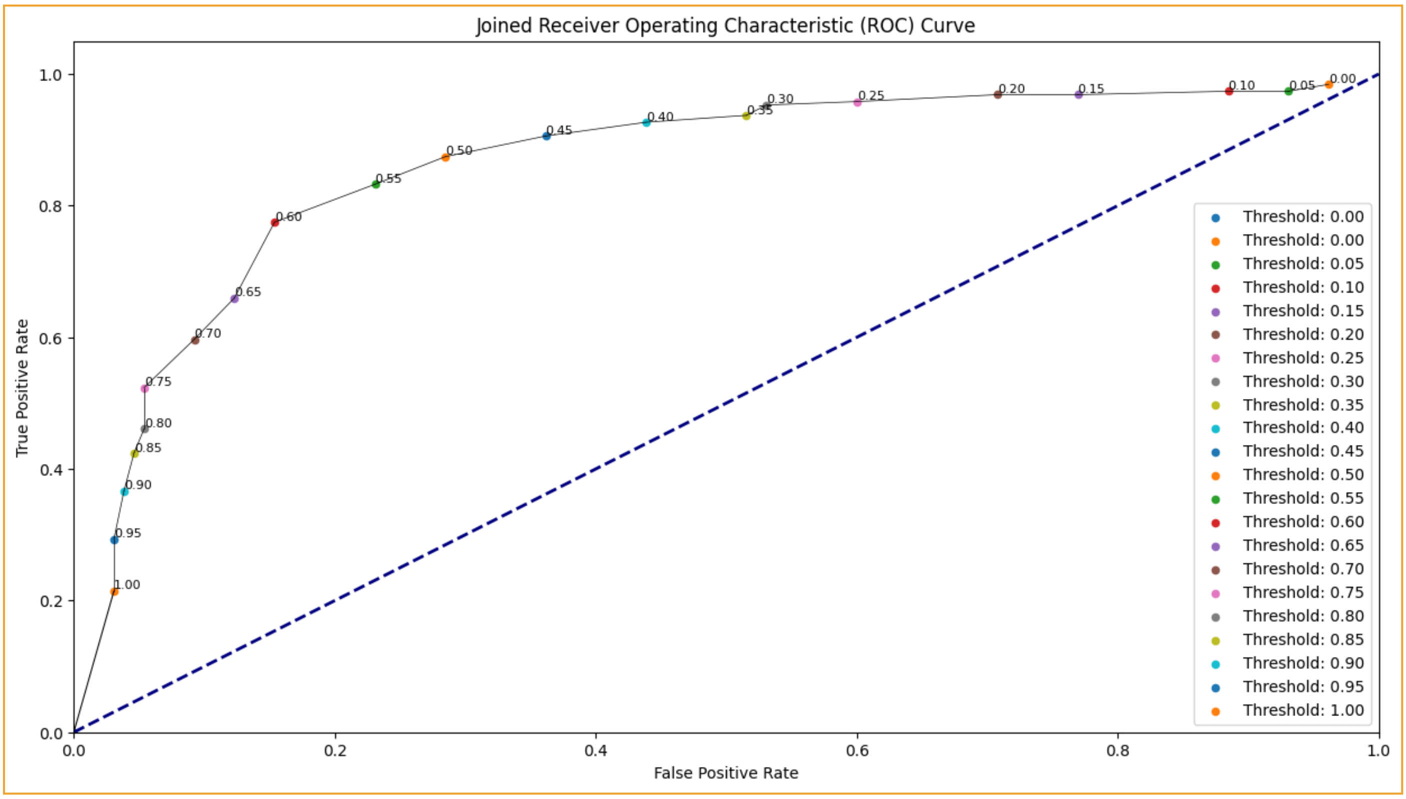



```
Accuracy of  Model on Testing Data is : 0.8130841121495327

 Classification report on Testing Data is :
               precision    recall  f1-score   support

           0       0.77      0.74      0.76       125
           1       0.84      0.86      0.85       196

    accuracy                           0.81       321
   macro avg       0.80      0.80      0.80       321
weighted avg       0.81      0.81      0.81       321
```

Built an ROC Curve for various Threshold Values

Making Receiver Operating Characteristic for Various Thresholld Values

```python
def get_roc_points(model, x_test, y_test):
    threshold_values = np.arange(0, 1.05, 0.05)
    roc_points = []

    for threshold in threshold_values:
        predictions = predict_with_threshold(model, x_test, threshold)
        fpr, tpr, _ = roc_curve(y_test, predictions)
        roc_points.append((fpr, tpr, threshold))

    return roc_points

def plot_roc_curve(roc_points):
    roc_points.sort(key=lambda x: x[2])  # Sort by threshold
    plt.figure(figsize=(15,8))
    plt.scatter(0, 0, label='Threshold: 0.00', s=20)
    for i in range(len(roc_points) - 1):
        fpr1, tpr1, threshold1 = roc_points[i]
        fpr2, tpr2, threshold2 = roc_points[i + 1]
        plt.plot([fpr1[1], fpr2[1]], [tpr1[1], tpr2[1]], color='black', lw=0.5)
    fpr2, tpr2, threshold2 = roc_points[len(roc_points)-1]
    plt.plot([0, fpr2[1]], [0, tpr2[1]], color='black', lw=0.75)
    for fpr, tpr, threshold in roc_points:
        plt.scatter(fpr[1], tpr[1], label=f'Threshold: {threshold:.2f}', s=20)
        plt.text(fpr[1], tpr[1], f'{threshold:.2f}', fontsize=8, ha='left', va='bottom', color='black')

    auc_values = [auc(fpr, tpr) for fpr, tpr, _ in roc_points]
    mean_auc = np.mean(auc_values)
    plt.text(0.5, 0.05, f'Mean AUC = {mean_auc:.2f}', ha='center', va='center', fontsize=12, color='black')
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Joined Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc="lower right")
    plt.show()
```


Joined Receiver Operating Characteristic (ROC) Curve
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
1.00
0.95
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.45
0.40
0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
Threshold: 0.00
Threshold: 0.00
Threshold: 0.05
Threshold: 0.10
Threshold: 0.15
Threshold: 0.20
Threshold: 0.25
Threshold: 0.30
Threshold: 0.35
Threshold: 0.40
Threshold: 0.45
Threshold: 0.50
Threshold: 0.55
Threshold: 0.60
Threshold: 0.65
Threshold: 0.70
Threshold: 0.75
Threshold: 0.80
Threshold: 0.85
Threshold: 0.90
Threshold: 0.95
Threshold: 1.00

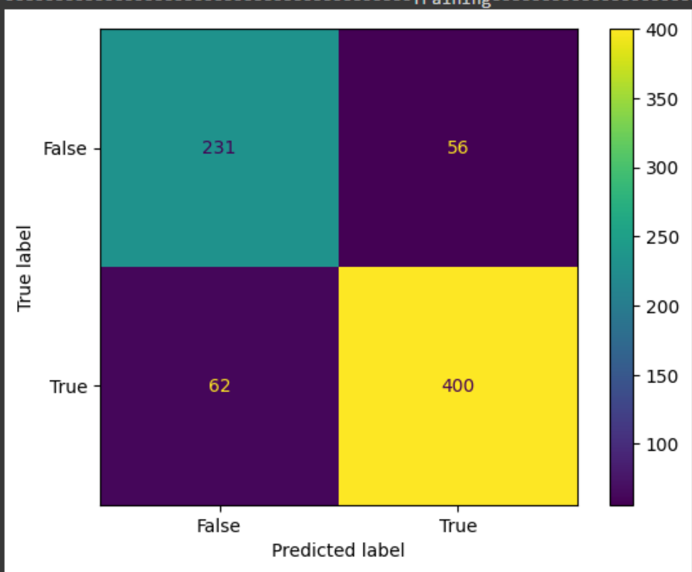
Training QDA and Analysis

I trained QDA model and evaluated the predictive accuracy for both training and testing data. I made the confusion Matrix and ROC curve as well.

Training QDA

```python
def train_qda_model(train_X, train_Y):
    model_qda = QDA_MODEL()
    train_X = train_X.astype(float)
    train_Y = train_Y.astype(int)
    model_qda.fit(train_X, train_Y)
    return model_qda

def evaluate_qda_model(model, test_X, test_Y):
    predictions_qda_test = model.predict(test_X)
    predictions_qda_train = model.predict(train_X)

    confusion_matrix(train_Y , predictions_qda_train, "Training")
    confusion_matrix(test_Y , predictions_qda_test, "Testing")
    return predictions_qda_test ,  predictions_qda_train

model_qda = train_qda_model(train_X, train_Y)
predicted_qda_test ,predictions_qda_train =  evaluate_qda_model(model_qda, test_X, test_Y)
```

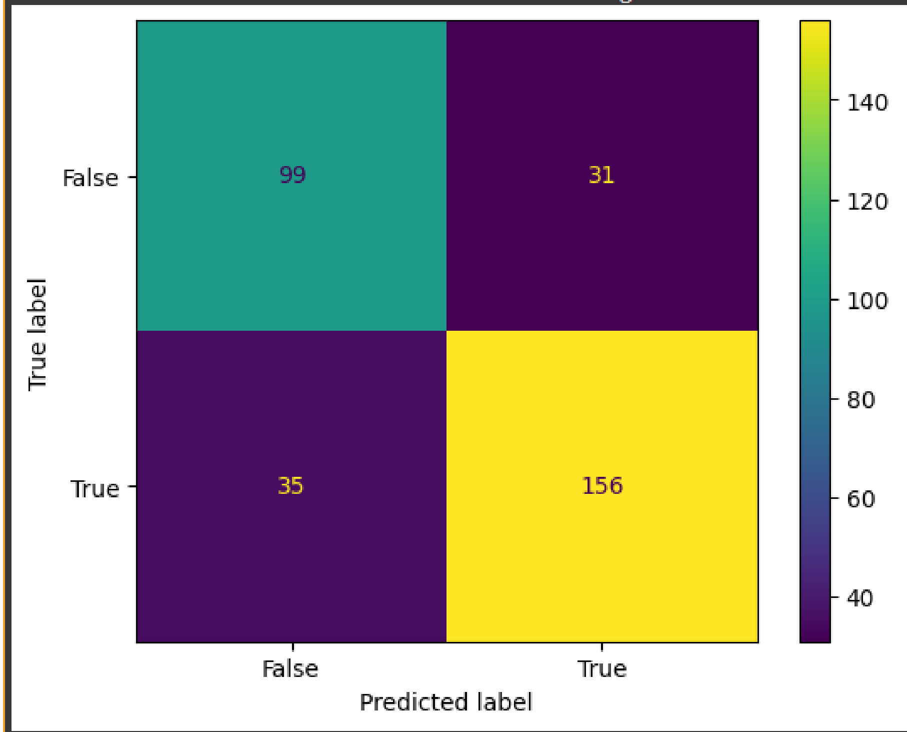
```
--------------------------------------Training--------------------------------------
```




```
Accuracy of  Model on Training Data is : 0.842456608811749

 Classification report on Training Data is :
               precision    recall  f1-score   support

           0       0.79      0.80      0.80       287
           1       0.88      0.87      0.87       462

    accuracy                           0.84       749
   macro avg       0.83      0.84      0.83       749
weighted avg       0.84      0.84      0.84       749
```

QDA gave a predictive accuracy of 84.245 % on training data and 79.439% on Testing data.

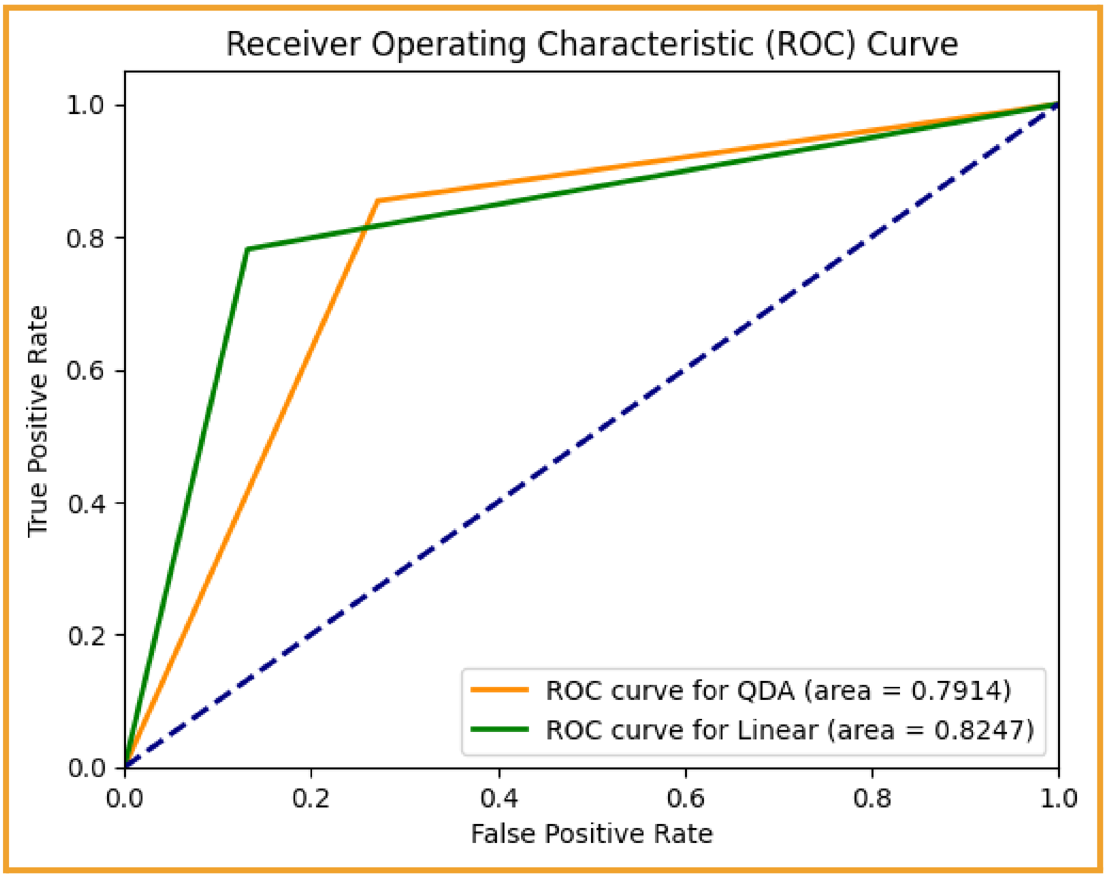
```
---------------------------------------Testing-------------------------------
```




```
Accuracy of  Model on Testing Data is : 0.794392523364486

 Classification report on Testing Data is :
               precision    recall  f1-score   support

           0       0.74      0.76      0.75       130
           1       0.83      0.82      0.83       191

    accuracy                           0.79       321
   macro avg       0.79      0.79      0.79       321
weighted avg       0.80      0.79      0.79       321
```

Plotting ROC curve for Both QDA and for best Threshold in Linear Regression

I plotted the ROC curve for QDA and Linear regression for the best Threshold. It is observed the AUC of linear regression is more than the AUC of QDA. Also, the Test accuracy of Linear Regression is more than the test accuracy of QDA.



Conclusion

From this, we can say that the Linear Regression appears to be a slightly better model for classification in this case. QDA also performed worse than LDA, since it fit a more flexible classifier than necessary.. The added flexibility of QDA would result in higher estimation variance due to the need for estimating some additional parameters. In relatively small samples, this cost will outweigh the lower model bias due to QDA offering a better approximation to the Data Generating Process than Linear Boundary.

I tested it for various test data sizes and various iterations of fitting, still, the result of LDA having better testing accuracy remained the same with a higher AUC. Hence We can conclude that Linear regression performed better in this as compared to QDA.

References

- Stats Stack Exchange - Removing Multicollinearity Here
- Stats Stack Exchange - Linear V/S Quadratic Boundary Here
- An Introduction to Statistical Learning (2013) p. 152-153