FLIP ROBO

Micro Credit Defaulter Project

Submitted by:

Rahul Singh

ACKNOWLEDGMENT

Thanks for giving me the opportunity to work in FlipRobo Technologies as Intern and would like to express my gratitude to Data Trained Institute as well for trained me in Data Science Domain. This helps me to do my projects well and understand the concepts.

Dataset – FlipRobo Tech

Resources used – Google, GitHub, Blogs for conceptual referring.

INTRODUCTION

- **Business Problem Framing**
  A Microfinance Institution (MFI) is an organization that offers financial services to low-income populations.
  MFS becomes very useful when targeting especially the unbanked poor families living in remote areas with not much sources of income.
  It is widely accepted as a poverty-reduction tool, representing $70 billion in outstanding loans and a global outreach of 200 million clients.
  The client wants some predictions that could help them in further investment and improvement in selection of customers for the credit.

- **Conceptual Background of the Domain Problem**
  Many microfinance institutions (MFI), experts and donors are supporting the idea

of using mobile financial services (MFS) which they feel are more convenient and efficient, and cost saving, than the traditional high-touch model used since long for the purpose of delivering microfinance services.
They are collaborating with an MFI to provide micro-credit on mobile balances to be paid back in 5 days. The Consumer is believed to be defaulter if he deviates from the path of paying back the loaned amount within the time duration of 5 days. For the loan amount of 5 (in Indonesian Rupiah), payback amount should be 6 (in Indonesian Rupiah), while, for the loan amount of 10 (in Indonesian Rupiah), the payback amount should be 12 (in Indonesian Rupiah).
This problem contains data of customers who is defaulter / Non – defaulters and has the main account and data account recharge and total amount of sum amount and its frequency. So, we need to predict for each loan transaction, whether the customer will

be paying back the loaned amount within 5 days of insurance of loan.

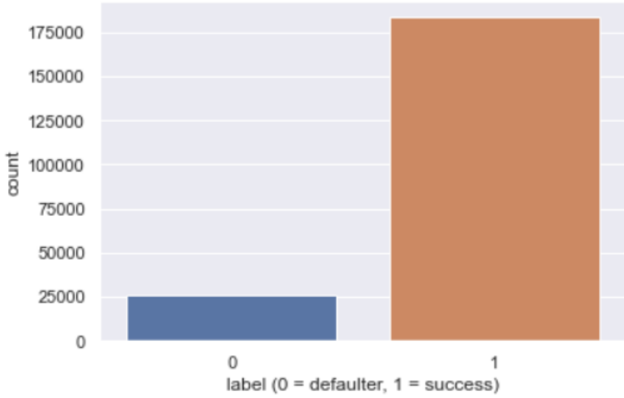
- **Motivation for the Problem Undertaken**
  This will help the client to get help on their future investment on telecom industry and that will improve the importance of communication in a person’s life, thus, focusing on providing their services and products to low-income families and poor customers that can help them in the need of hour.

Analytical Problem Framing

- **Mathematical/ Analytical Modeling of the Problem**
  In this case, Label ‘1’ indicates that the loan has been payed i.e., Non- defaulter, while Label ‘0’ indicates that the loan has not been payed i.e., defaulter. In the provided *dataset*, our target variable "label" is a *categorical* with two categories: " defaulter " and " Non- defaulter ". Therefore, we will be handling this modelling problem as classification.

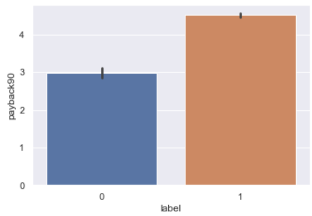
- **Data Sources and their formats**
  We can see that our target variable has more non-defaulters (paying loan on time) than defaulters (not paying loan on time)

```
9]: # We can see that most of the customer will be paying back the loaned amount within 5 days of i
#In this case, Label '1' indicates that the loan has been payed i.e. Non- defaulter, while,
#Label '0' indicates that the loan has not been payed i.e. defaulter.
#let's visualize the count of label using Seaborn
sns.set_theme()
sns.countplot(df['label'])
plt.xlabel('label (0 = defaulter, 1 = success)')
plt.show()
```



Most of the customers (non-defaulters) are paying back their loan by 3-5 days,

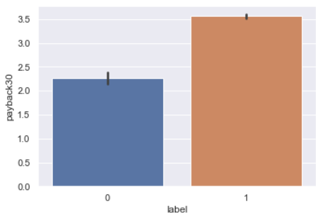
```
In [20]: #let's visualize the count of payback90 using Seaborn
sns.set_theme()
sns.barplot(x = df['label'], y = df['payback90'])
plt.show()
```



Average pay back time of potential defaulter in 90 days is 3 days.

Average pay back time of repayer in 90 days is greater than 4 days.

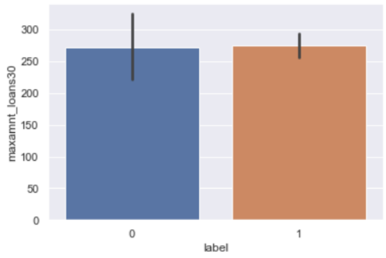
```
In [21]: #let's visualize the count of payback30 using seaborn
sns.set_theme()
sns.barplot(x = df['label'], y = df['payback30'])
plt.show()
```



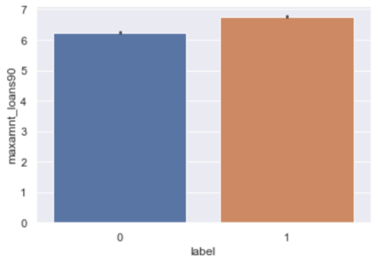
Maximum amount of loans is taken by defaulters in <30 days and there are 2

options 5rs and 10rs which customer needs to payback as 6rs and 12rs.

```
sns.set_theme()
sns.barplot(x = df['label'], y = df['maxamnt_loans30'])
plt.show()
```




```
In [23]: sns.set_theme()
sns.barplot(x = df['label'], y = df['maxamnt_loans90'])
plt.show()
```



- **Data Preprocessing Done**
  Replacing some of the 0 values to mean, median as it is having 0 values more and customer who got loan has to payback In 30 days and 90 days and frequency of main account and data account recharged and count of data account and main account of recharged.

If account got recharged and customers needs to payback the loan within their 30days and 90days.
Also, we have outliers as well and tried applying Z-score method, we are losing >10% data, So I am removing skewness by using power transform method.

- **Data Inputs- Logic- Output Relationships**

  Our target variable is label which indicates the customer is a defaulter who is not paying back or non-defaulter who is paying back the loan with some features like how often they are recharging, and daily amount spend by customer from main account and average main account balance and number of loans taken by user and maximum amount of loan taken by user in last 30 days or 90 days.

- **Hardware and Software Requirements and Tools Used**

  1. **Pandas is open-source library tool which provides high performance data analysis tool by its powerful data structures. It**

helps to shorten the procedure of handling the data with extensive set of features.

2. **Numpy is most used package for scientific computing for multi-dimensional array of objects.**
3. **Other than this, as a pre-processing steps, I imported standard scaler for scaling the data.**
4. **I imported f1 score, classification report, confusion matrix, roc curve in terms of metrics to calculate the model score.**

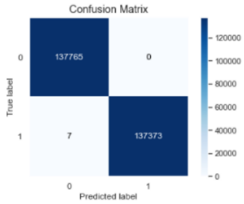
Model/s Development and Evaluation

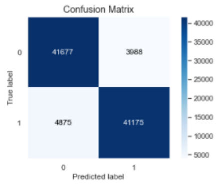
- **Testing of Identified Approaches (Algorithms)**
  I have used Decision tree algorithm, Random Forest, Ada Boost and Gradient Boost algorithm to calculate the score of the model.

- **Run and Evaluate selected models Decision Tree model which has score – 90.34% and CV score – 88.29%**

In [48]:
```
#train result
DecisionTree = DecisionTreeClassifier()
DecisionTree.fit(x_train, y_train)
y_pred =DecisionTree .predict(x_train)
accuracy = classification_report(y_train, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_train, y_pred))

#test result
DecisionTree = DecisionTreeClassifier()
DecisionTree.fit(x_train, y_train)
y_pred =DecisionTree .predict(x_test)
accuracy = classification_report(y_test, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    137765
           1       1.00      1.00      1.00    137380

    accuracy                           1.00    275145
   macro avg       1.00      1.00      1.00    275145
weighted avg       1.00      1.00      1.00    275145

AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0       0.90      0.91      0.90     45665
           1       0.91      0.89      0.90     46050

    accuracy                           0.90     91715
   macro avg       0.90      0.90      0.90     91715
weighted avg       0.90      0.90      0.90     91715

AxesSubplot(0.125,0.125;0.62x0.755)
```





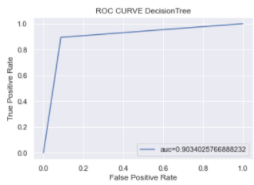
In [49]:
```
print("Training accuracy::",DecisionTree.score(x_train,y_train))
print("Test accuracy::",DecisionTree.score(x_test,y_test))
```

```
Training accuracy:: 0.99997455886896
Test accuracy:: 0.9033636809682167
```

In [50]:
```
print(cross_val_score(DecisionTree,x,y,cv=5).mean())
```

```
0.8829296848090911
```

In [51]:
```
#roc_curve plot to check the socre of Decisiontree
fpr, tpr, _= roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
plt.plot(fpr, tpr, label="auc="+str(auc_score))
plt.box(True)
plt.title('ROC CURVE DecisionTree')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.grid(True)
plt.show()
print('The Score for the ROC Curve is : {}%'.format(round(auc_score,4)*100))
```

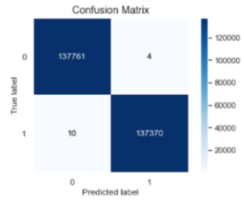


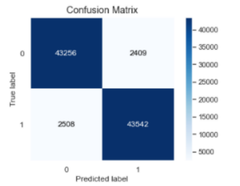

```
The Score for the ROC Curve is : 90.34%
```

Random Forest model has score- 94.64% and CV score – 91.9%

In [52]:
```
#train result
RFC = RandomForestClassifier()
RFC.fit(x_train, y_train)
y_pred =RFC .predict(x_train)
accuracy = classification_report(y_train, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_train, y_pred))


#test result
RFC = RandomForestClassifier()
RFC.fit(x_train, y_train)
y_pred =RFC .predict(x_test)
accuracy = classification_report(y_test, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00    137765
           1       1.00      1.00      1.00    137380

    accuracy                           1.00    275145
   macro avg       1.00      1.00      1.00    275145
weighted avg       1.00      1.00      1.00    275145

AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     45665
           1       0.95      0.95      0.95     46050

    accuracy                           0.95     91715
   macro avg       0.95      0.95      0.95     91715
weighted avg       0.95      0.95      0.95     91715

AxesSubplot(0.125,0.125;0.62x0.755)
```





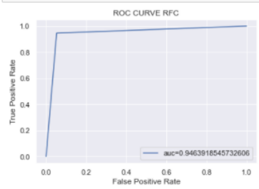
In [53]:
```
print("Training accuracy::",RFC.score(x_train,y_train))
print("Test accuracy::",RFC.score(x_test,y_test))
```

```
Training accuracy:: 0.9999418488433371
Test accuracy:: 0.9463882680041432
```

In [54]:
```
print(cross_val_score(RFC,x,y,cv=5).mean())
```

```
0.9197297643546923
```

In [55]:
```
#roc_curve plot to check the socre of RFC
fpr, tpr, _= roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
plt.plot(fpr, tpr, label="auc="+str(auc_score))
plt.box(True)
plt.title('ROC CURVE RFC')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.grid(True)
plt.show()
print('The Score for the ROC Curve is : {}%'.format(round(auc_score,4)*100))
```

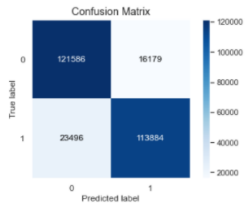


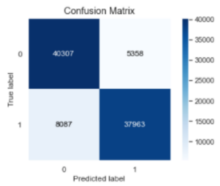

```
The Score for the ROC Curve is : 94.64%
```

Ada boost has score – 85.35% and CV score is 90%

In [56]:
```
#train result
adb = AdaBoostClassifier()
adb.fit(x_train, y_train)
y_pred =adb .predict(x_train)
accuracy = classification_report(y_train, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_train, y_pred))


#test result
adb = AdaBoostClassifier()
adb.fit(x_train, y_train)
y_pred =adb .predict(x_test)
accuracy = classification_report(y_test, y_pred)
print(accuracy)
print(skplt.metrics.plot_confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.84      0.88      0.86    137765
           1       0.88      0.83      0.85    137380

    accuracy                           0.86    275145
   macro avg       0.86      0.86      0.86    275145
weighted avg       0.86      0.86      0.86    275145

AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0       0.83      0.88      0.86     45665
           1       0.88      0.82      0.85     46050

    accuracy                           0.85     91715
   macro avg       0.85      0.85      0.85     91715
weighted avg       0.85      0.85      0.85     91715

AxesSubplot(0.125,0.125;0.62x0.755)
```





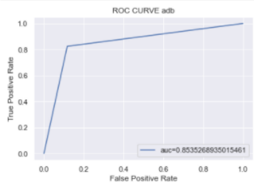
In [57]:
```
print("Training accuracy::",adb.score(x_train,y_train))
print("Test accuracy::",adb.score(x_test,y_test))
```

```
Training accuracy:: 0.855803303712588
Test accuracy:: 0.8534045685002454
```

In [58]:
```
print(cross_val_score(adb,x,y,cv=10).mean())
```

```
0.909376306849906
```

In [59]:
```
#roc_curve plot to check the socre of AdaBoostClassifier
fpr, tpr, _= roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
plt.plot(fpr, tpr, label="auc="+str(auc_score))
plt.box(True)
plt.title('ROC CURVE adb')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.grid(True)
plt.show()
print('The Score for the ROC Curve is : {}%'.format(round(auc_score,4)*100))
```

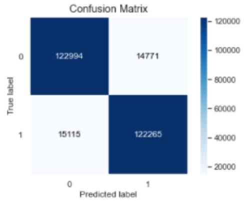


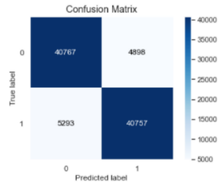

```
The Score for the ROC Curve is : 85.35000000000001%
```

Gradient Boost model -88.8% and cv score – 91.7%

In [60]:
```python
#train result
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
y_pred =gbc .predict(x_train)
accuracy = classification_report(y_train, y_pred)
print(accuracy)
print( skplt.metrics.plot_confusion_matrix(y_train, y_pred))


#test result
gbc = GradientBoostingClassifier()
gbc.fit(x_train, y_train)
y_pred =gbc .predict(x_test)
accuracy = classification_report(y_test, y_pred)
print(accuracy)
print( skplt.metrics.plot_confusion_matrix(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.89      0.89      0.89    137765
           1       0.89      0.89      0.89    137380

    accuracy                           0.89    275145
   macro avg       0.89      0.89      0.89    275145
weighted avg       0.89      0.89      0.89    275145

AxesSubplot(0.125,0.125;0.62x0.755)
              precision    recall  f1-score   support

           0       0.89      0.89      0.89     45665
           1       0.89      0.89      0.89     46050

    accuracy                           0.89     91715
   macro avg       0.89      0.89      0.89     91715
weighted avg       0.89      0.89      0.89     91715

AxesSubplot(0.125,0.125;0.62x0.755)
```





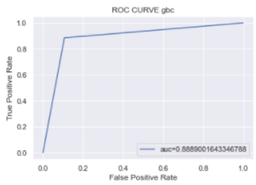
In [61]:
```python
print("Training accuracy::",gbc.score(x_train,y_train))
print("Test accuracy::",gbc.score(x_test,y_test))
```

```
Training accuracy:: 0.8913809082483781
Test accuracy:: 0.888884042959167
```

In [62]:
```python
print(cross_val_score(gbc,x,y,cv=5).mean())
```

```
0.9179739680283596
```

In [63]:
```python
#roc_curve plot to check the socre of GradientBoostClassifier
fpr, tpr, _= roc_curve(y_test, y_pred)
auc_score = roc_auc_score(y_test, y_pred)
plt.plot(fpr, tpr, label="auc="+str(auc_score))
plt.box(True)
plt.title('ROC CURVE gbc')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.legend(loc=4)
plt.grid(True)
plt.show()
print('The Score for the ROC Curve is : {}%'.format(round(auc_score,4)*100))
```




```
The Score for the ROC Curve is : 88.89%
```

- **Key Metrics for success in solving problem under consideration**

Used F1 Score for calculating the accuracy score as the target variables classes are im-balanced and accuracy score metric won't give correct results as it may take classes with more count.

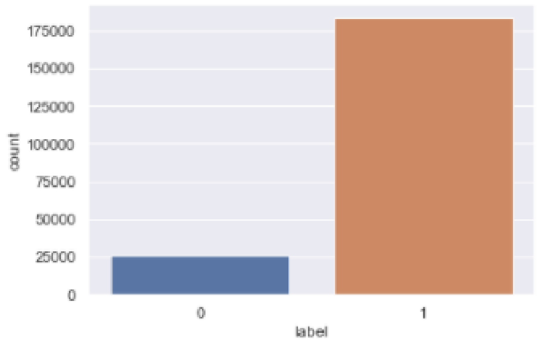
Classification report will display the overview of accuracy, precision, recall, f1 score , support and weighted average.

Confusion matrix for calculating true positive and true negative.

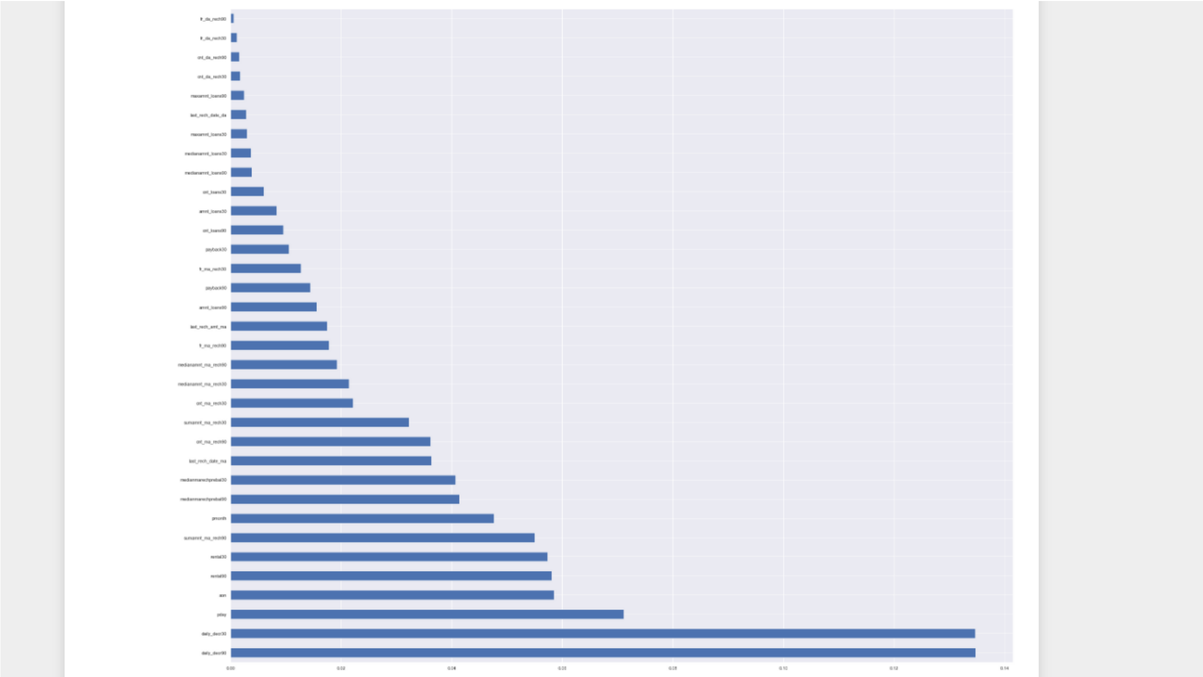
- **Visualizations**

  Target variable plot where it shows the classes are im-balanced.

In [42]:
```
sns.countplot(Y)
plt.show()
```

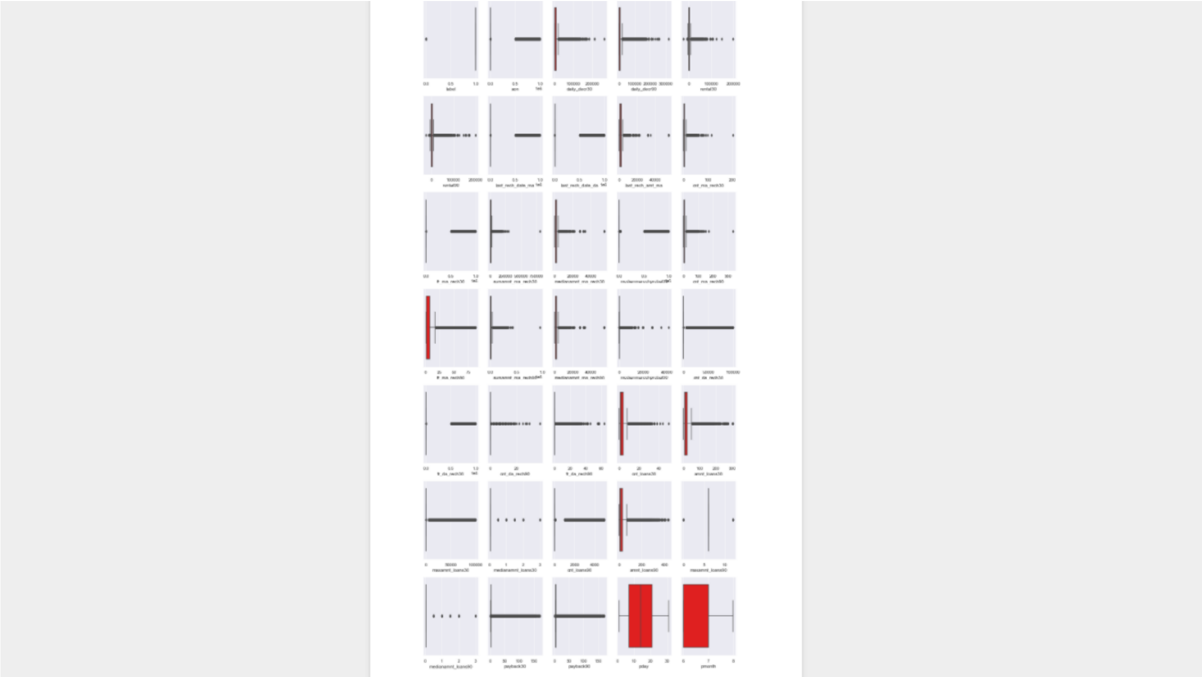


Feature Importance,



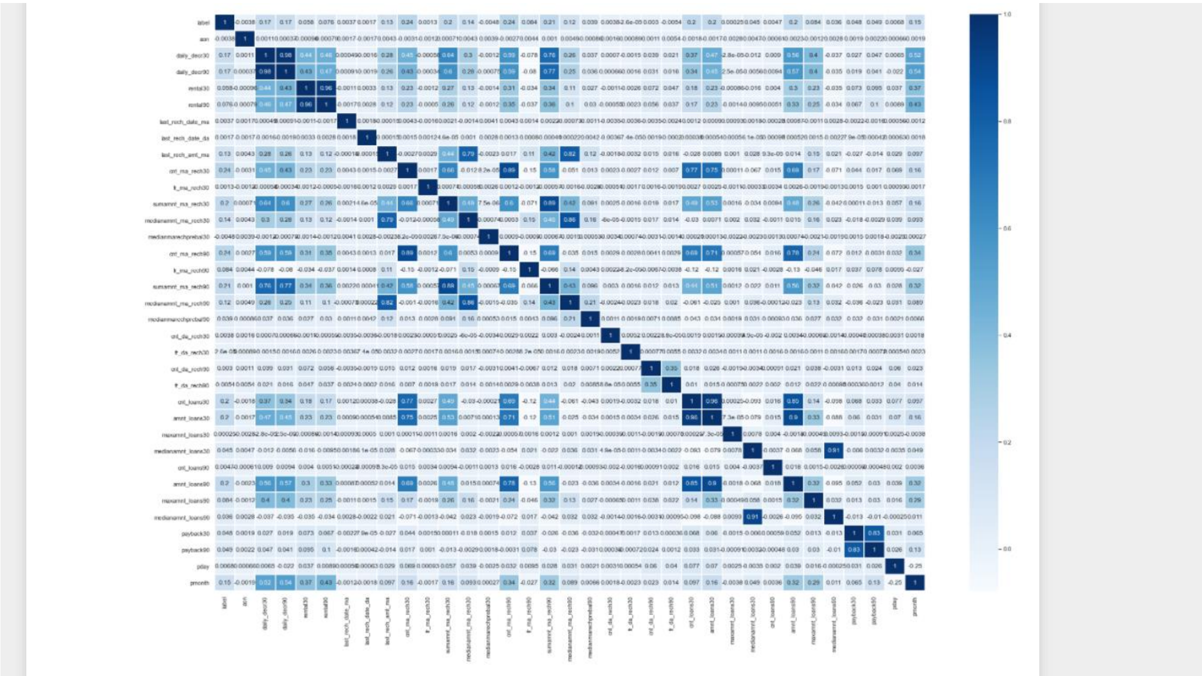
Histogram plots of columns,

Box plot



- **Interpretation of the Results**

  Correlation matrix after dropping less importance features and high skewed data,

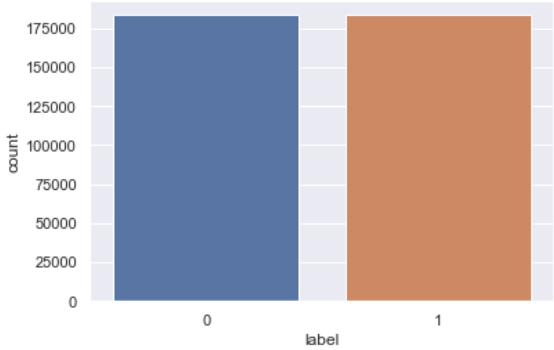


After using SMOTE() technique for balancing the im-balanced class,

```
In [43]: #using SMOTE() technique to balance the classes,

         from sklearn.utils import resample
         from imblearn.over_sampling import SMOTE

         sm = SMOTE()
         x_over,y_over = sm.fit_resample(X,Y)
```

```
In [44]: sns.countplot(y_over)
```

```
Out[44]: <AxesSubplot:xlabel='label', ylabel='count'>
```

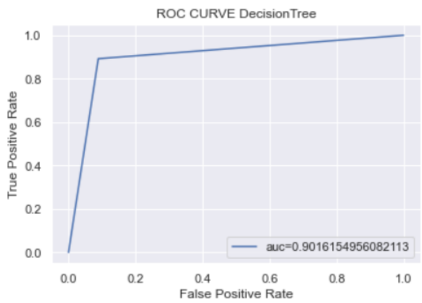


Final model accuracy Decision tree score – 90.16%

Roc curve of final model,

Model has improved the accuracy from 89% to 91%

```
In [68]: #Roc Curve for final model,
         fpr, tpr, _= roc_curve(y_test, y_pred)
         auc_score = roc_auc_score(y_test, y_pred)
         plt.plot(fpr, tpr, label="auc="+str(auc_score))
         plt.box(True)
         plt.title('ROC CURVE DecisionTree')
         plt.xlabel('False Positive Rate')
         plt.ylabel('True Positive Rate')
         plt.legend(loc=4)
         plt.grid(True)
         plt.show()
         print('The Score for the ROC Curve is : {}%'.format(round(auc_score,4)*100))
```




```
The Score for the ROC Curve is : 90.16%
```

CONCLUSION

- **Key Findings and Conclusions of the Study We can tell that target variable is im-balanced and need to balance that and data loss is more actually and need to handle that as well as we can’t lose >8% of data.**

Dealing with huge dataset has taken a lot of time for running each algorithm and hyper parameter has taken more time to train the data and it was a nice experience that I have learnt so many things by worked on this project.