FLIP ROBO

MICRO CREDIT LOAN DEFAULTER PROJECT

Submitted by:

Dipesh Ramesh Limaje

Problem Statement:

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. The Microfinance services (MFS) provided by MFI are Group Loans, Agricultural Loans, Individual Business Loans and so on.

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. Though, the MFI industry is primarily focusing on low income families and are very useful in such areas, the implementation of MFS has been uneven with both significant challenges and successes.

Today, microfinance is widely accepted as a poverty-reduction tool, representing $70 billion in outstanding loans and a global outreach of 200 million clients.

We are working with one such client that is in Telecom Industry. They are a fixed wireless telecommunications network provider. They have launched various products and have developed its business and organization based on the budget operator model, offering better products at Lower Prices to all value conscious customers through a strategy of disruptive innovation that focuses on the subscriber.

They understand the importance of communication and how it affects 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.

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).

The sample data is provided to us from our client database. It is hereby given to you for this exercise. In order to improve the selection of customers for the credit, the client wants some predictions that could help them in further investment and improvement in selection of customers.

Conceptual Background of the Domain Problem

The domain problem for housing projects is a complex one and involves a range of factors. The most fundamental issue is the lack of affordable housing in many cities and regions, especially those with high demand for housing due to population growth. This is compounded by the fact that the cost of construction often outpaces inflation, making it difficult for low-income households to find affordable housing in desirable areas. Additionally, local zoning regulations, building codes, and other restrictions can further limit the available options.
In order to address this issue, housing projects must focus on creating model that predict house sale price. The final step in the domain problem is predicting the sale price of the housing project. This is a complex task that involves analyzing a variety of factors, such as the location, the quality of the construction, the amenities offered, and the condition of the property. Additionally, the sale price can be affected by the market conditions, the availability of comparable properties, and the overall demand for housing in the area. By understanding these factors and using predictive analytics, it is possible to determine the likely sale price of a housing project.

Review of Literature

There is a significant amount of literature on the use of microcredit loans as a tool for poverty reduction and economic development. Research in this area has generally focused on evaluating the effectiveness of microcredit loan programs in achieving these goals, as well as identifying factors that are associated with successful repayment of loans.

One study by Khandker, Khalily and Khan (1998) found that microcredit programs led to an increase in household income and consumption, and improved living standards for the poor. Similarly, a study by Banerjee, Duflo and Kinnan (2015) found that microcredit programs led to increased business investments and profits for microentrepreneurs in India.

Other studies have looked at the factors associated with successful repayment of microcredit loans. For example, a study by Hashemi, Schuler and Riley (1996) found that social networks play a significant role in loan repayment, as borrowers are more likely to repay their loans if they have a supportive network of friends and family.

Additionally, research has shown that microcredit loan programs are most effective when they are accompanied by other forms of support, such as financial education, business training, and access to markets. This finding has been supported by a study by Gine, Karlan and Zinman (2010) which found that microcredit programs that provide business training and financial education have a higher repayment rate than those that do not.

In a data science perspective, these studies suggest that predictive modeling techniques can be used to identify individuals who are most likely to repay their loans and targeting these individuals for microcredit loan programs. Additionally, data analysis can be used to identify patterns and trends in loan data that can be used to improve the design and implementation of microcredit loan programs.

Motivation for the Problem Undertaken

There are several motivations for undertaking a problem on microcredit loan use case in data science. One of the main motivations is to improve the effectiveness of microcredit loan programs in reducing poverty and promoting economic development. By analyzing loan data, it is possible to identify patterns and trends that can be used to improve the design and implementation of these programs, such as identifying factors that are associated with successful repayment of loans.

Another motivation is to use data science and machine learning techniques to improve the credit scoring and risk assessment process of microcredit loan providers. This will enable them to identify the best candidates for microcredit loans, thus minimizing the risk of loan defaults and maximizing the impact of the loan programs on poverty reduction and economic development.

Moreover, an additional motivation is to use data science to improve the operations and management of microcredit loan providers. By analyzing data on loan performance, providers can identify areas of their operations that need improvement, as well as identify best practices that can be adopted by other providers.

Additionally, understanding the micro-loan market would be beneficial for researchers and policymakers to understand the impact of the microcredit loan program and make decisions on future programs.

In summary, the main motivations for undertaking a problem on microcredit loan use case in data science are to improve the effectiveness and efficiency of microcredit loan programs and to understand the micro-loan market.

Mathematical/ Analytical Modeling of the Problem

In the context of microcredit loan use case, data science can be used to develop mathematical and analytical models to identify default risk and predict loan default. This can be achieved through various techniques, including but not limited to:

- Logistic Regression: Logistic regression is a statistical method that can be used to model the relationship between a binary dependent variable (e.g., loan default) and one or more independent variables (e.g., income, credit score, etc.). It can be used to identify factors that are associated with loan default and to predict the probability of loan default for individual borrowers.
- Decision Trees: Decision tree algorithms can be used to identify patterns in the data and make predictions about loan default. This method can be used to identify the most important factors that are associated with loan default, and to create a model that can be used to predict the probability of loan default for individual borrowers.
- Random Forest: Random Forest is an ensemble learning method that combines multiple decision trees to improve the accuracy of predictions. It can be used to identify the most important factors that are associated with loan default and to create a model that can be used to predict the probability of loan default for individual borrowers.
- SVM: SVC is another ensemble learning method that can be used to improve the accuracy of predictions. It creates a series of decision trees, where each tree is built to correct the errors of the previous tree.

Once these models are built, they can be used to identify high-risk borrowers, and target them for interventions such as financial education or credit counseling in order to reduce the risk of loan default.

Data Sources and their formats

Data sources for microcredit loan data science projects can come from a variety of sources, including financial institutions, government agencies, and non-profit organizations. The format of the data can vary, but common formats include CSV, Excel, and SQL databases. It may also include unstructured data such as PDFs or scanned documents which would need to be preprocessed. Additionally, data may be collected through surveys or other forms of direct data collection. The specifics of the data source and format will depend on the specific project and goals of the analysis.

Data Description

1. Label: Flag indicating whether the user paid back the credit amount within 5 days of issuing the loan{1:success, 0:failure}

2. Msisdn: mobile number of user

3. Aon: age on cellular network in days

4. daily_decr30: Daily amount spent from main account, averaged over last 30 days (in Indonesian Rupiah)

5. daily_decr90: Daily amount spent from main account, averaged over last 90 days (in Indonesian Rupiah)

6. rental30: Average main account balance over last 30 days

7. rental90: Average main account balance over last 90 days

8. last_rech_date_ma: Number of days till last recharge of main account

9. last_rech_date_da: Number of days till last recharge of data account

10. last_rech_amt_ma: Amount of last recharge of main account (in Indonesian Rupiah)

11. cnt_ma_rech30: Number of times main account got recharged in last 30 days

12. fr_ma_rech30: Frequency of main account recharged in last 30 days

13. sumamnt_ma_rech30: Total amount of recharge in main account over last 30 days (in Indonesian Rupiah)

14. medianamnt_ma_rech30: Median of amount of recharges done in main account over last 30 days at user level (in Indonesian Rupiah)

15. medianmarechprebal30: Median of main account balance just before recharge in last 30 days at user level (in Indonesian Rupiah)

16. cnt_ma_rech90: Number of times main account got recharged in last 90 days

17. fr_ma_rech90: Frequency of main account recharged in last 90 days

18. sumamnt_ma_rech90: Total amount of recharge in main account over last 90 days (in Indonasian Rupiah)

19. medianamnt_ma_rech90: Median of amount of recharges done in main account over last 90 days at user level (in Indonasian Rupiah)

20. medianmarechprebal90: Median of main account balance just before recharge in last 90 days at user level (in Indonasian Rupiah)

21. cnt_da_rech30: Number of times data account got recharged in last 30 days

22. fr_da_rech30: Frequency of data account recharged in last 30 days

23. cnt_da_rech90: Number of times data account got recharged in last 90 days

24. fr_da_rech90: Frequency of data account recharged in last 90 days

25. cnt_loans30: Number of loans taken by user in last 30 days

26. amnt_loans30: Total amount of loans taken by user in last 30 days

27. maxamnt_loans30: maximum amount of loan taken by the user in last 30 days

28. medianamnt_loans30: Median of amounts of loan taken by the user in last 30 days

29. cnt_loans90: Number of loans taken by user in last 90 days

30. amnt_loans90: Total amount of loans taken by user in last 90 days

31. maxamnt_loans90: maximum amount of loan taken by the user in last 90 days

32. medianamnt_loans90: Median of amounts of loan taken by the user in last 90 days

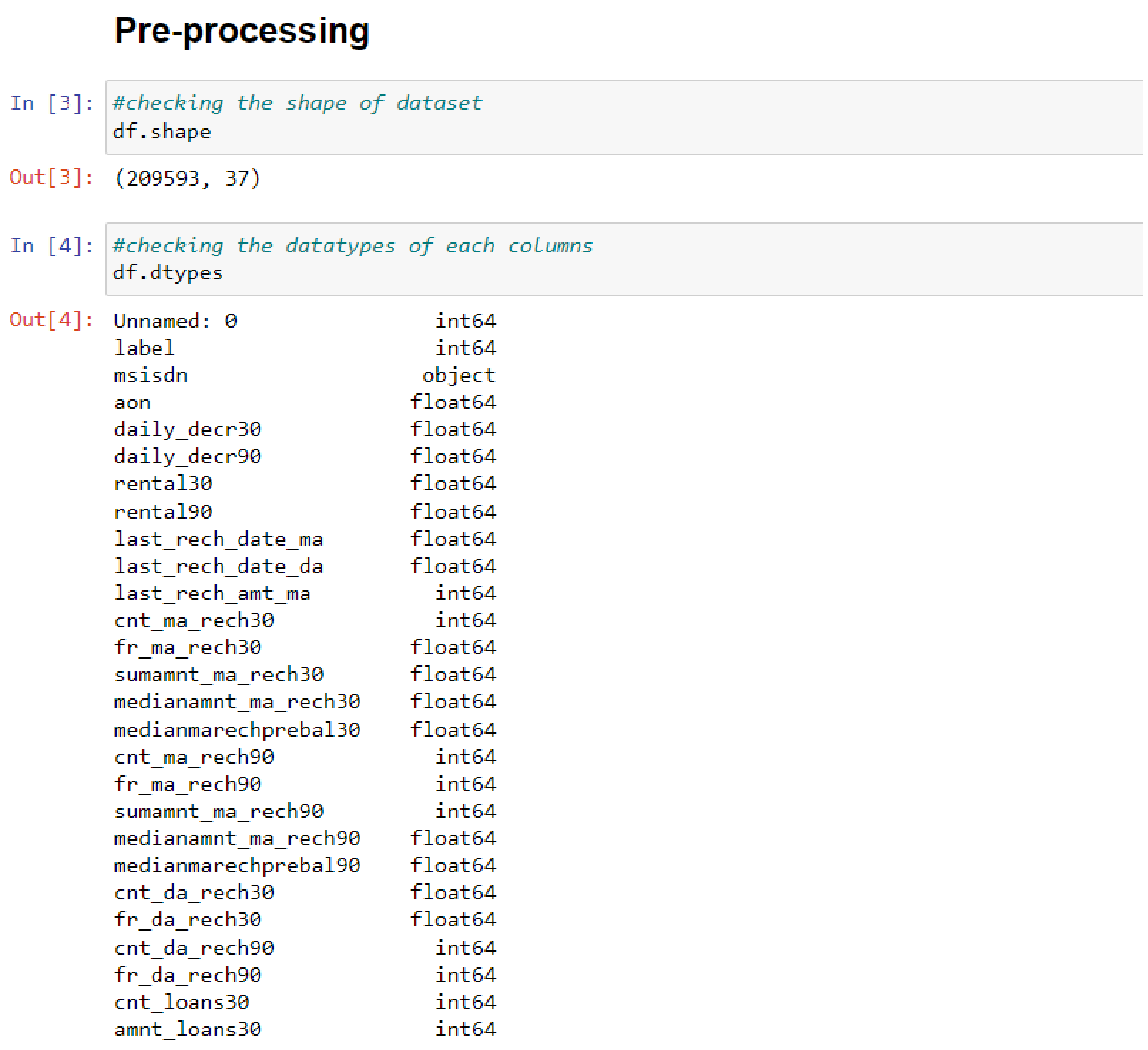
33. payback30: Average payback time in days over last 30 days

34. payback90: Average payback time in days over last 90 days

35. pcircle: telecom circle

36. pdate: date

Data Pre-processing Done



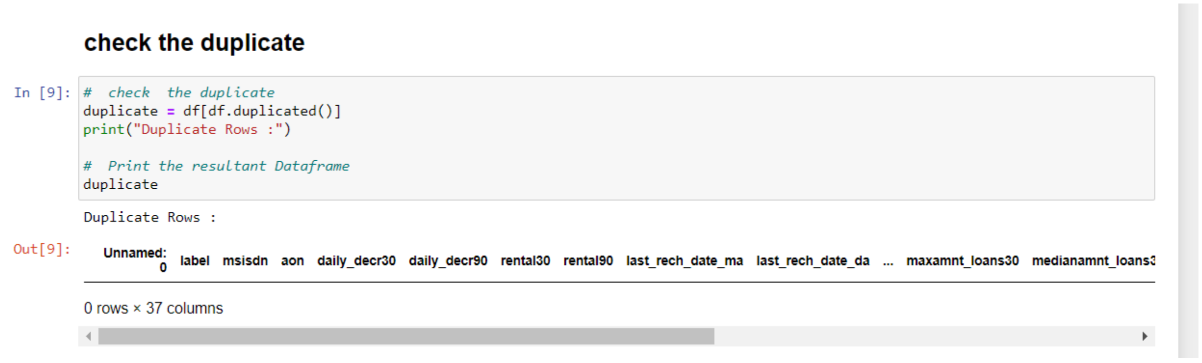
Pre-processing

```
In [3]: #checking the shape of dataset
        df.shape

Out[3]: (209593, 37)

In [4]: #checking the datatypes of each columns
        df.dtypes

Out[4]: Unnamed: 0                int64
        label                     int64
        msisdn                   object
        aon                     float64
        daily_decr30            float64
        daily_decr90            float64
        rental30                float64
        rental90                float64
        last_rech_date_ma       float64
        last_rech_date_da       float64
        last_rech_amt_ma          int64
        cnt_ma_rech30             int64
        fr_ma_rech30            float64
        sumamnt_ma_rech30       float64
        medianamnt_ma_rech30    float64
        medianmarechprebal30    float64
        cnt_ma_rech90             int64
        fr_ma_rech90              int64
        sumamnt_ma_rech90         int64
        medianamnt_ma_rech90    float64
        medianmarechprebal90    float64
        cnt_da_rech30           float64
        fr_da_rech30            float64
        cnt_da_rech90             int64
        fr_da_rech90              int64
        cnt_loans30               int64
        amnt_loans30              int64
```

Pre-processing I have done

1. Check for null values in the dataset
2. Verify the shape of the dataset
3. Compare column names with the data description provided.

4] Then I have check for the duplicates and there was no duplicates were present in dataset.



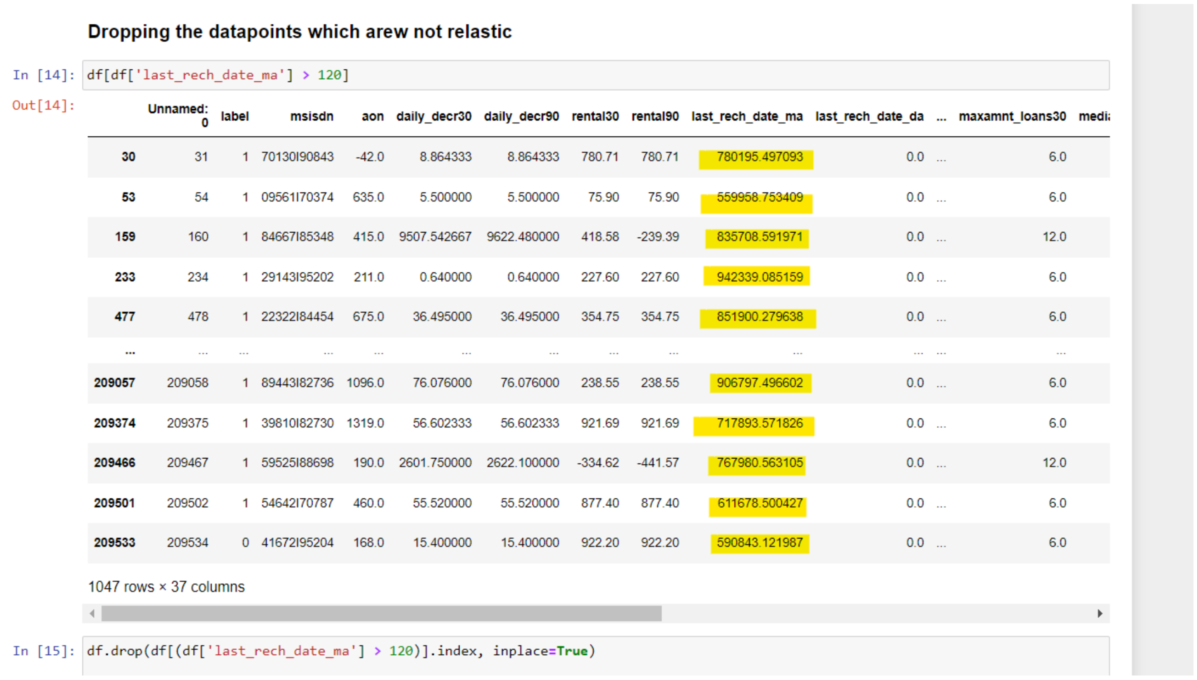
check the duplicate

```
In [9]: # check  the duplicate
        duplicate = df[df.duplicated()]
        print("Duplicate Rows :")

        #  Print the resultant Dataframe
        duplicate

        Duplicate Rows :

Out[9]:
```

Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	medianamnt_loans3

0 rows × 37 columns

5] Then I have checked the Data columns wise and then I realise that there are some non-realistic data present in datatset so I dropped the datapoints which are non-realistic.



Dropping the datapoints which arew not relastic

```
In [14]: df[df['last_rech_date_ma'] > 120]

Out[14]:
```

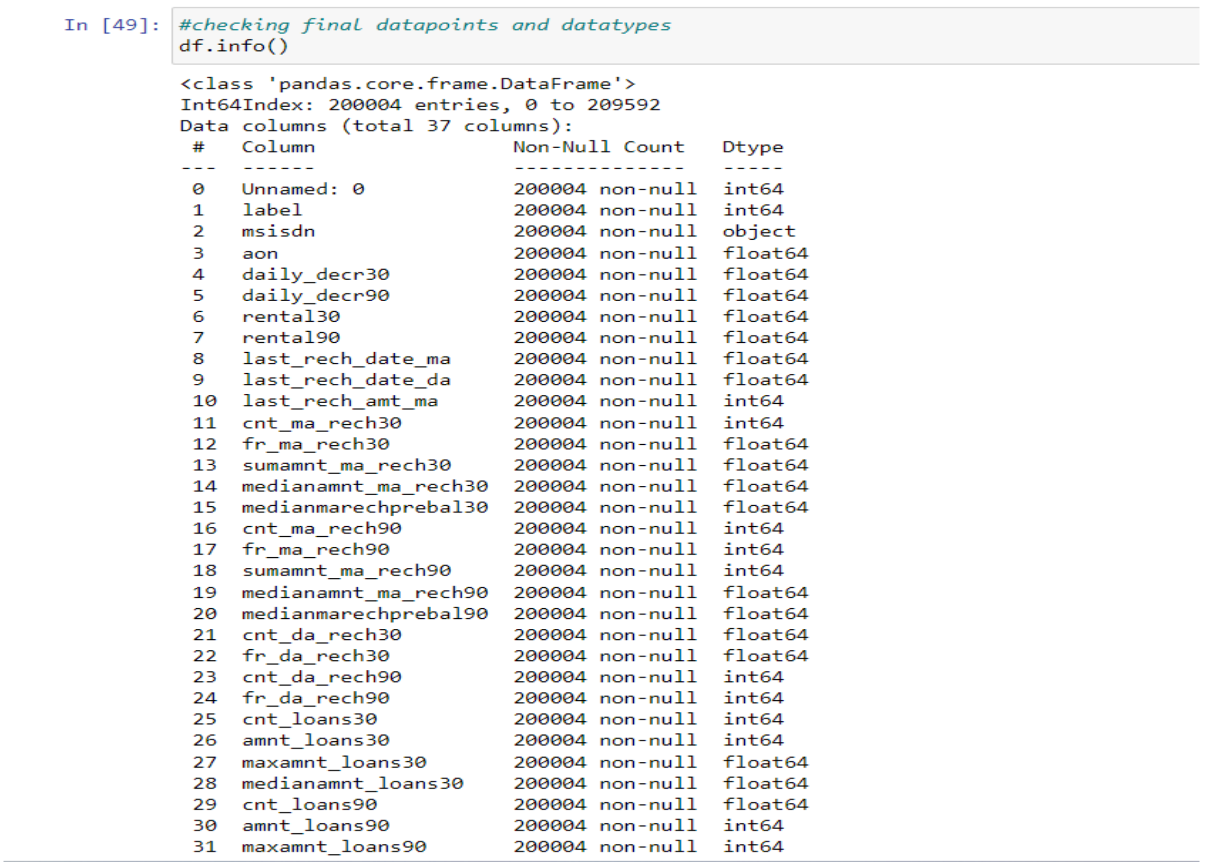
	Unnamed: 0	label	msisdn	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	...	maxamnt_loans30	medi
30	31	1	70130I90843	-42.0	8.864333	8.864333	780.71	780.71	780195.497093	0.0	...	6.0	
53	54	1	09561I70374	635.0	5.500000	5.500000	75.90	75.90	559958.753409	0.0	...	6.0	
159	160	1	84667I85348	415.0	9507.542667	9622.480000	418.58	-239.39	835708.591971	0.0	...	12.0	
233	234	1	29143I95202	211.0	0.640000	0.640000	227.60	227.60	942339.085159	0.0	...	6.0	
477	478	1	22322I84454	675.0	36.495000	36.495000	354.75	354.75	851900.279638	0.0	...	6.0	
...	...	...	...	...	...	...	...	...	...	...	...	...	
209057	209058	1	89443I82736	1096.0	76.076000	76.076000	238.55	238.55	906797.496602	0.0	...	6.0	
209374	209375	1	39810I82730	1319.0	56.602333	56.602333	921.69	921.69	717893.571826	0.0	...	6.0	
209466	209467	1	59525I88698	190.0	2601.750000	2622.100000	-334.62	-441.57	767980.563105	0.0	...	12.0	
209501	209502	1	54642I70787	460.0	55.520000	55.520000	877.40	877.40	611678.500427	0.0	...	6.0	
209533	209534	0	41672I95204	168.0	15.400000	15.400000	922.20	922.20	590843.121987	0.0	...	6.0	

1047 rows × 37 columns

```
In [15]: df.drop(df[(df['last_rech_date_ma'] > 120)].index, inplace=True)
```

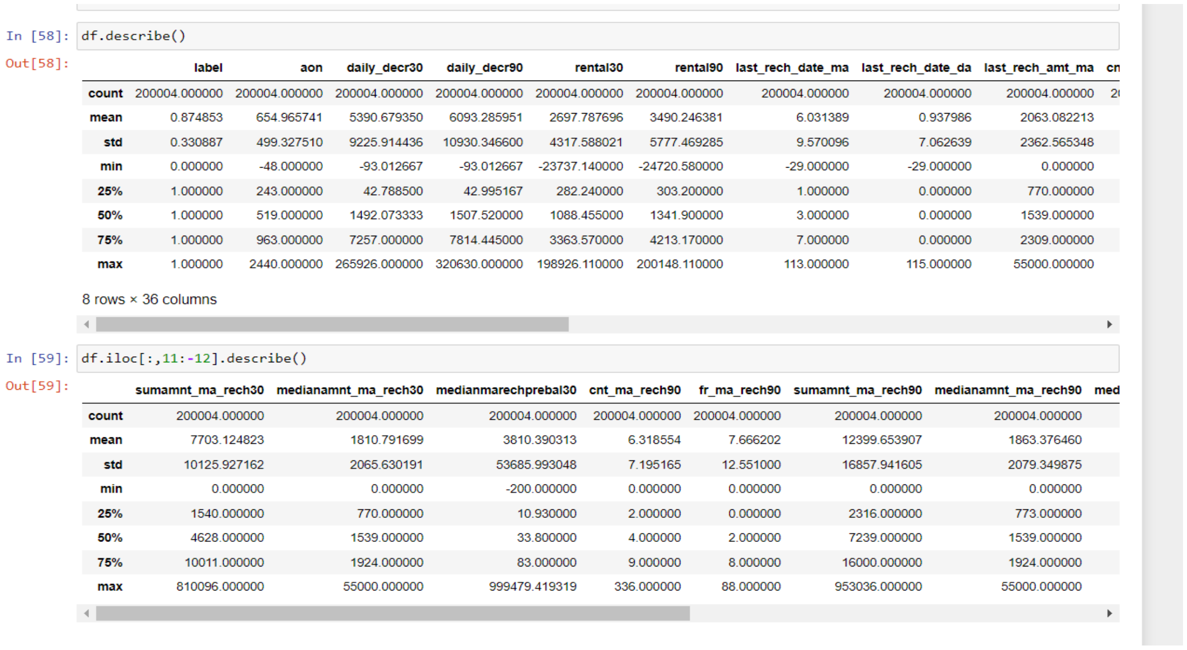
Then I followed same steps for this particular columns (last_rech_date_ma,last_rech_date_da,fr_ma_rech30,cnt_da_rech30,fr_da_rech30,maxamnt_loans30,cnt_loans90,aon)

6] After treating all un-realistic datapoints and dropping all unwanted columns finally adding some columns and treating all datatypes our dataset's final shape is 200004 Rows and 36 Columns.



```
In [49]: #checking final datapoints and datatypes
         df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 200004 entries, 0 to 209592
Data columns (total 37 columns):
 #   Column                Non-Null Count   Dtype
---  ------                --------------   -----
 0   Unnamed: 0            200004 non-null  int64
 1   label                 200004 non-null  int64
 2   msisdn                200004 non-null  object
 3   aon                   200004 non-null  float64
 4   daily_decr30          200004 non-null  float64
 5   daily_decr90          200004 non-null  float64
 6   rental30              200004 non-null  float64
 7   rental90              200004 non-null  float64
 8   last_rech_date_ma     200004 non-null  float64
 9   last_rech_date_da     200004 non-null  float64
 10  last_rech_amt_ma      200004 non-null  int64
 11  cnt_ma_rech30         200004 non-null  int64
 12  fr_ma_rech30          200004 non-null  float64
 13  sumamnt_ma_rech30     200004 non-null  float64
 14  medianamnt_ma_rech30  200004 non-null  float64
 15  medianmarechprebal30  200004 non-null  float64
 16  cnt_ma_rech90         200004 non-null  int64
 17  fr_ma_rech90          200004 non-null  int64
 18  sumamnt_ma_rech90     200004 non-null  int64
 19  medianamnt_ma_rech90  200004 non-null  float64
 20  medianmarechprebal90  200004 non-null  float64
 21  cnt_da_rech30         200004 non-null  float64
 22  fr_da_rech30          200004 non-null  float64
 23  cnt_da_rech90         200004 non-null  int64
 24  fr_da_rech90          200004 non-null  int64
 25  cnt_loans30           200004 non-null  int64
 26  amnt_loans30          200004 non-null  int64
 27  maxamnt_loans30       200004 non-null  float64
 28  medianamnt_loans30    200004 non-null  float64
 29  cnt_loans90           200004 non-null  float64
 30  amnt_loans90          200004 non-null  int64
 31  maxamnt_loans90       200004 non-null  int64
```

7] After that I have to Describe the dataset to observe the numerical values and write the Observations.



```
In [58]: df.describe()
```

Out[58]:

	label	aon	daily_decr30	daily_decr90	rental30	rental90	last_rech_date_ma	last_rech_date_da	last_rech_amt_ma	cn
count	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	2
mean	0.874853	654.965741	5390.679350	6093.285951	2697.787696	3490.246381	6.031389	0.937986	2063.082213	
std	0.330887	499.327510	9225.914436	10930.346600	4317.588021	5777.469285	9.570096	7.062639	2362.565348	
min	0.000000	-48.000000	-93.012667	-93.012667	-23737.140000	-24720.580000	-29.000000	-29.000000	0.000000	
25%	1.000000	243.000000	42.788500	42.995167	282.240000	303.200000	1.000000	0.000000	770.000000	
50%	1.000000	519.000000	1492.073333	1507.520000	1088.455000	1341.900000	3.000000	0.000000	1539.000000	
75%	1.000000	963.000000	7257.000000	7814.445000	3363.570000	4213.170000	7.000000	0.000000	2309.000000	
max	1.000000	2440.000000	265926.000000	320630.000000	198926.110000	200148.110000	113.000000	115.000000	55000.000000	

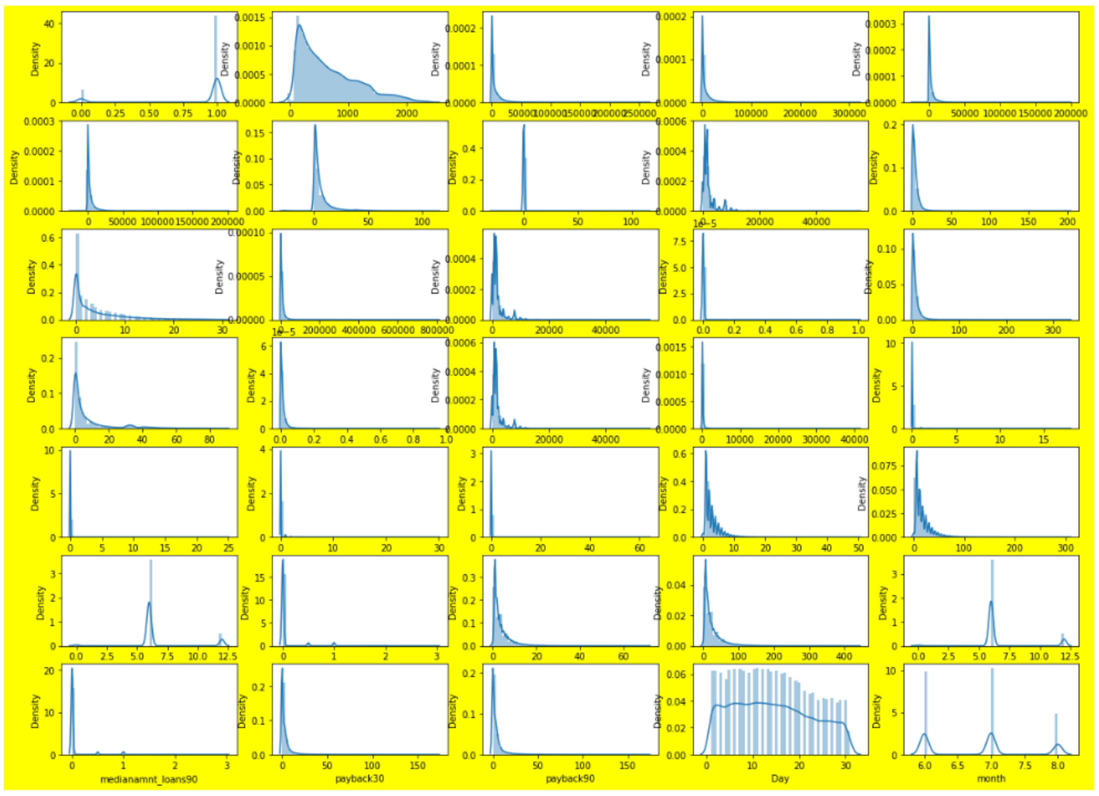
8 rows × 36 columns

```
In [59]: df.iloc[:,11:-12].describe()
```

Out[59]:

	sumamnt_ma_rech30	medianamnt_ma_rech30	medianmarechprebal30	cnt_ma_rech90	fr_ma_rech90	sumamnt_ma_rech90	medianamnt_ma_rech90	med
count	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	200004.000000	
mean	7703.124823	1810.791699	3810.390313	6.318554	7.666202	12399.653907	1863.376460	
std	10125.927162	2065.630191	53685.993048	7.195165	12.551000	16857.941605	2079.349875	
min	0.000000	0.000000	-200.000000	0.000000	0.000000	0.000000	0.000000	
25%	1540.000000	770.000000	10.930000	2.000000	0.000000	2316.000000	773.000000	
50%	4628.000000	1539.000000	33.800000	4.000000	2.000000	7239.000000	1539.000000	
75%	10011.000000	1924.000000	83.000000	9.000000	8.000000	16000.000000	1924.000000	
max	810096.000000	55000.000000	999479.419319	336.000000	88.000000	953036.000000	55000.000000	

Data Inputs- Logic- Output Relationships



To observe the relationship between Feature and label so I created this Regression plot to observe which features are positively co-related and which features are negatively co-related.

State the set of assumptions (if any) related to the problem under consideration

1] For this particular problem I have dropped the unwanted columns which I feel like it is unwanted.

2]Identify and remove unrealistic data points by dropping any data points with a value greater than a certain threshold for each column.

3] For this particular problem I have assumed that the Maximum VIF should be 10, if any of the features has a VIF which is greater than 10 we should drop that feature.

Hardware and Software Requirements and Tools Used

1. **Hardware Requirements**: -Computer with minimum 8 GB RAM -High-speed internet connection -High-end graphics card -External storage device

2. **Software Requirements**: -Python programming language -TensorFlow -Keras -Scikit-Learn -Pandas -Matplotlib -Seaborn

3. **Tools Used**: -Jupyter Notebook -Google Collab -Tableau -Power BI

4. **Model For Micro Credit Project**: -Logistic regression -Random Forest -Decision Tree – KNN

Model/s Development and Evaluation

Identification of possible problem-solving approaches (methods)

There are several problem-solving approaches that can be applied to a micro credit loan use case in data science, including:

1. **Predictive modeling**: This approach involves using historical loan data to build a model that can predict the likelihood of loan default. Various machine learning algorithms such as Logistic Regression, Random Forest, Decision Tree etc. can be used for this purpose.
2. **Clustering**: This approach involves grouping similar customers together based on their characteristics and behaviour. Clustering algorithms such as k-means, hierarchical clustering, and density-based clustering can be used to identify segments of customers with similar characteristics.
3. **Anomaly detection:** This approach involves identifying abnormal or unusual patterns in the loan data that may indicate potential fraud or other issues. Algorithms such as Isolation Forest, Local Outlier Factor (LOF) can be used for this purpose.
4. **Decision Trees:** This approach involves using tree-based algorithms to identify the factors that influence the loan approval or rejection. These algorithms can be used to identify the most important factors to consider when making lending decisions and can help to identify which customers are most likely to default on their loans.
5. **Natural Language Processing**: This approach involves using techniques from NLP to extract insights from unstructured data such as customer reviews, complaints, and feedback. This can help to identify patterns and trends that may not be immediately obvious from the structured loan data.
6. **Network Analysis**: this approach involves identifying the relationship between different entities (e.g. borrowers, lenders, guarantors) in the loan data. This can help to identify potential fraud or other issues by identifying unusual patterns in the relationships between borrowers, lenders, and guarantors.

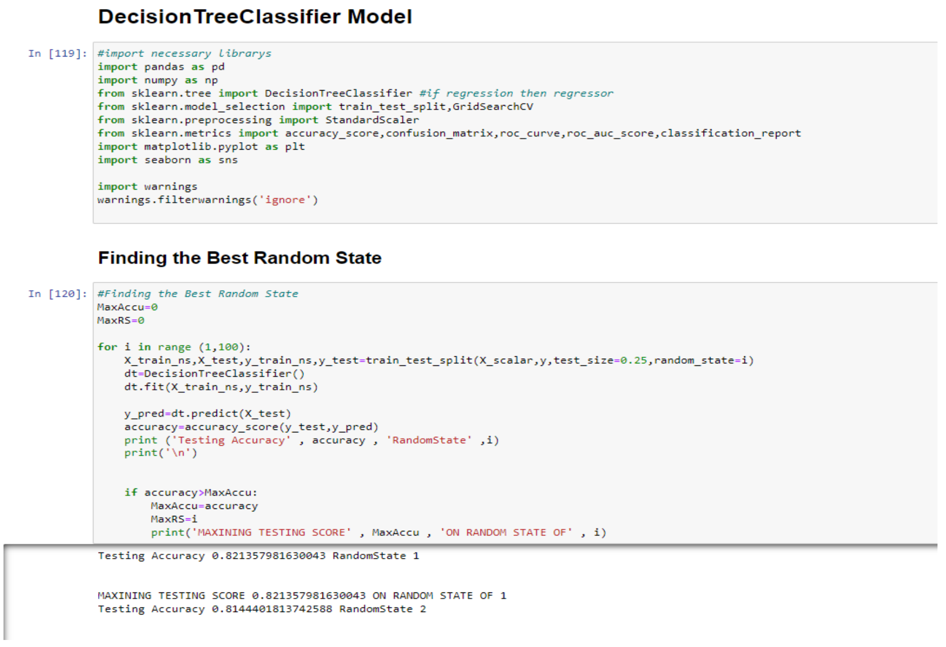
These are some of the common methods, you can use one or more based on the requirements.

Testing of Identified Approaches (Algorithms)

- ➢ LR (Logistic Regression Model)
- ➢ DT (Decision Tree Classifier Model)
- ➢ RF (Random Forest Classifier Model)
- ➢ KNN (K-Nearest Neighbours Model)

Run and Evaluate selected models

1st Model I have Created is the Decision Tree Model



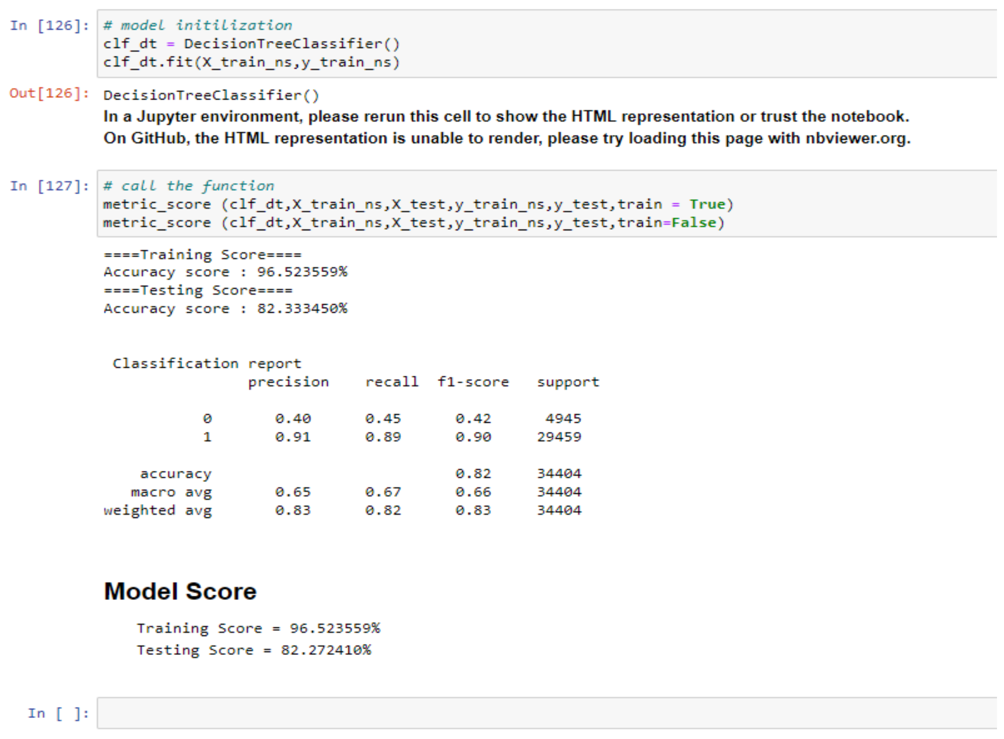
DecisionTreeClassifier Model

```
In [119]: #import necessary librarys
          import pandas as pd
          import numpy as np
          from sklearn.tree import DecisionTreeClassifier #if regression then regressor
          from sklearn.model_selection import train_test_split,GridSearchCV
          from sklearn.preprocessing import StandardScaler
          from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score,classification_report
          import matplotlib.pyplot as plt
          import seaborn as sns

          import warnings
          warnings.filterwarnings('ignore')
```

Finding the Best Random State

```
In [120]: #Finding the Best Random State
          MaxAccu=0
          MaxRS=0

          for i in range (1,100):
              X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=i)
              dt=DecisionTreeClassifier()
              dt.fit(X_train_ns,y_train_ns)

              y_pred=dt.predict(X_test)
              accuracy=accuracy_score(y_test,y_pred)
              print ('Testing Accuracy' , accuracy , 'RandomState' ,i)
              print('\n')


              if accuracy>MaxAccu:
                  MaxAccu=accuracy
                  MaxRS=i
                  print('MAXINING TESTING SCORE' , MaxAccu , 'ON RANDOM STATE OF' , i)
```

```
Testing Accuracy 0.821357981630043 RandomState 1


MAXINING TESTING SCORE 0.821357981630043 ON RANDOM STATE OF 1
Testing Accuracy 0.8144401813742588 RandomState 2
```



```
In [126]: # model initilization
          clf_dt = DecisionTreeClassifier()
          clf_dt.fit(X_train_ns,y_train_ns)

Out[126]: DecisionTreeClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [127]: # call the function
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 96.523559%
          ====Testing Score====
          Accuracy score : 82.333450%


           Classification report
                        precision    recall  f1-score   support

                     0       0.40      0.45      0.42      4945
                     1       0.91      0.89      0.90     29459

              accuracy                           0.82     34404
             macro avg       0.65      0.67      0.66     34404
          weighted avg       0.83      0.82      0.83     34404
```

Model Score

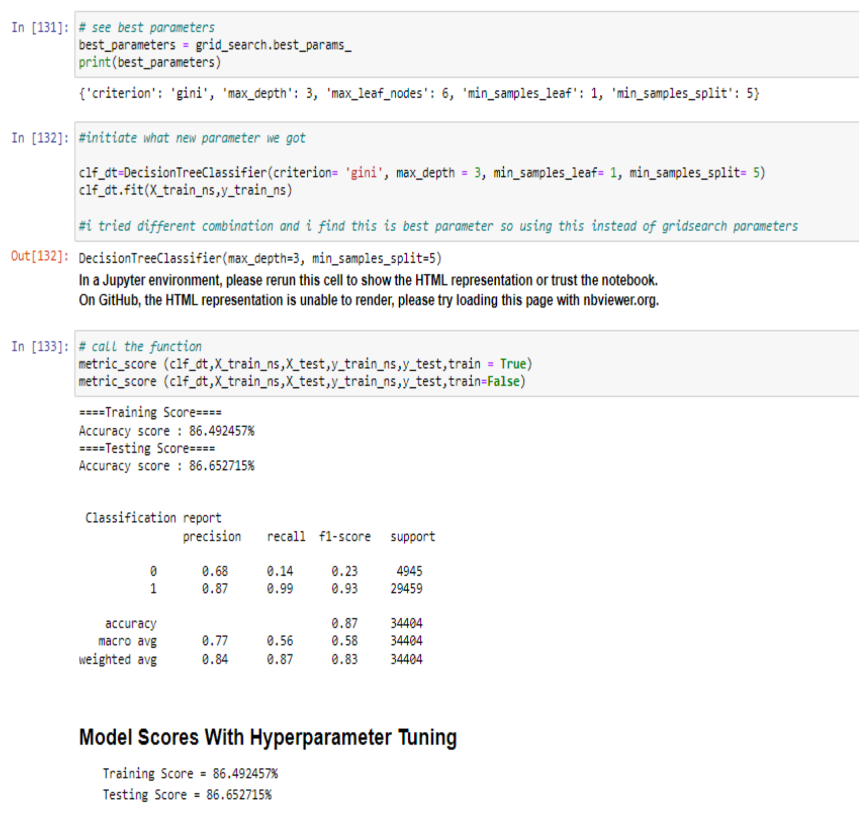
```
   Training Score = 96.523559%
   Testing Score = 82.272410%
```

```
In [ ]:
```

DT (Decision Tree Model) Score are

Training score = 96.523559%

Testing Score = 82.272410%



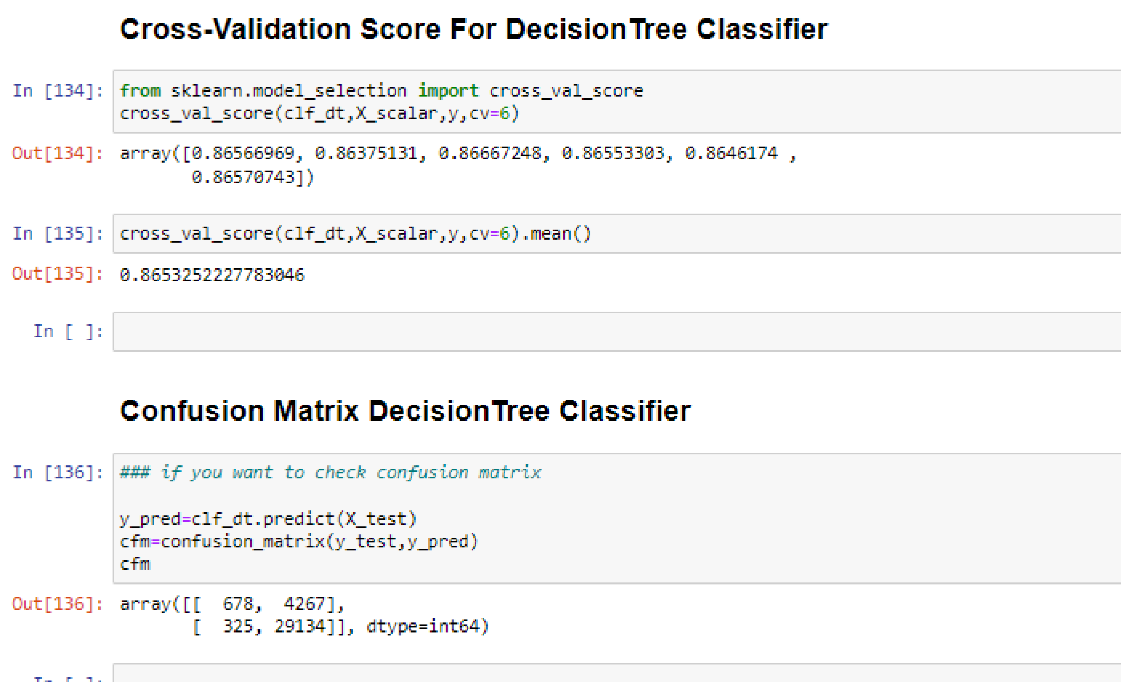
```
In [131]: # see best parameters
          best_parameters = grid_search.best_params_
          print(best_parameters)

          {'criterion': 'gini', 'max_depth': 3, 'max_leaf_nodes': 6, 'min_samples_leaf': 1, 'min_samples_split': 5}

In [132]: #initiate what new parameter we got

          clf_dt=DecisionTreeClassifier(criterion= 'gini', max_depth = 3, min_samples_leaf= 1, min_samples_split= 5)
          clf_dt.fit(X_train_ns,y_train_ns)

          #i tried different combination and i find this is best parameter so using this instead of gridsearch parameters

Out[132]: DecisionTreeClassifier(max_depth=3, min_samples_split=5)
          In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
          On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [133]: # call the function
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_dt,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 86.492457%
          ====Testing Score====
          Accuracy score : 86.652715%


           Classification report
                        precision    recall  f1-score   support

                     0       0.68      0.14      0.23      4945
                     1       0.87      0.99      0.93     29459

              accuracy                           0.87     34404
             macro avg       0.77      0.56      0.58     34404
          weighted avg       0.84      0.87      0.83     34404
```

Model Scores With Hyperparameter Tuning

```
Training Score = 86.492457%
Testing Score = 86.652715%
```

Model Score after Hyperparameter Tuning

Training Score = 86.492457%
Testing Score = 86.652715%



Cross-Validation Score For DecisionTree Classifier

```
In [134]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf_dt,X_scalar,y,cv=6)

Out[134]: array([0.86566969, 0.86375131, 0.86667248, 0.86553303, 0.8646174 ,
                 0.86570743])

In [135]: cross_val_score(clf_dt,X_scalar,y,cv=6).mean()

Out[135]: 0.8653252227783046

In [ ]:
```

Confusion Matrix DecisionTree Classifier

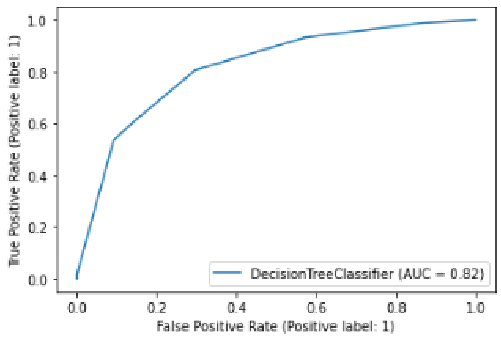
```
In [136]: ### if you want to check confusion matrix

          y_pred=clf_dt.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[136]: array([[  678,  4267],
                 [  325, 29134]], dtype=int64)
```

Cross Validation score for Decision Tree¶

Cross Validation Score at cv = 6 = 86.53252227783046%

AUC-ROC Curve for Training Data

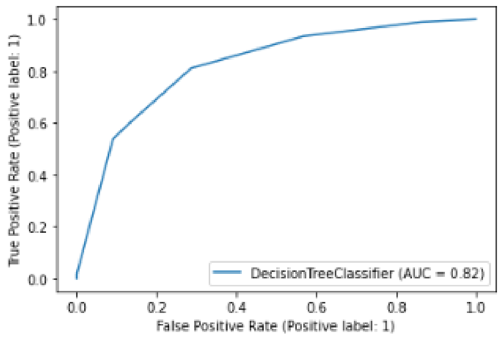
```
In [207]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(clf_dt, X_train_ns,y_train_ns)

Out[207]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x202693b8a00>
```



```
In [ ]:
```

AUC-ROC Curve for Testing Data

```
In [146]: from sklearn.metrics import RocCurveDisplay

In [149]: RocCurveDisplay.from_estimator(clf_dt, X_test, y_test)

Out[149]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x203aec67a00>
```

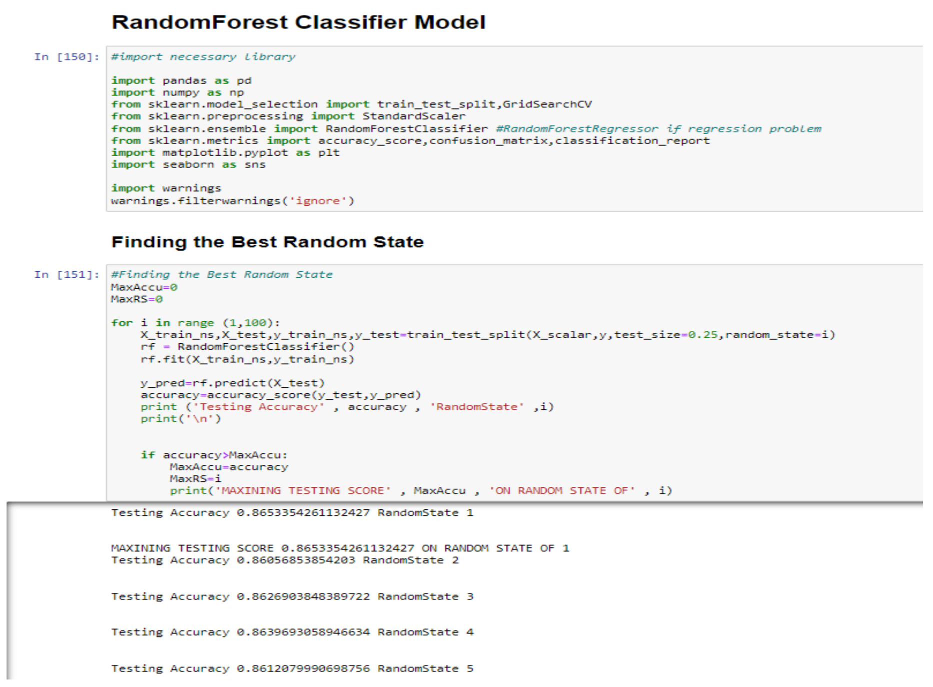


AUC-ROC Curve

For Training data, it cover 0.82% Area

For Testing data, it cover 0.82% Area

<u>**2nd Model I have Created is Random Forest Classifier Model**</u>

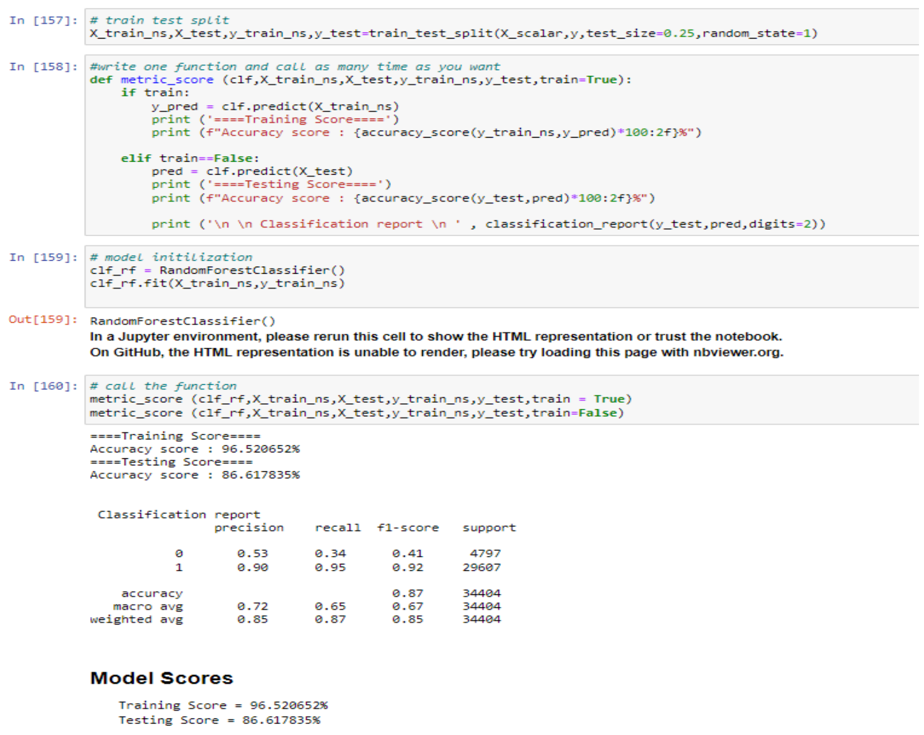


RandomForest Classifier Model

```
In [150]: #import necessary library

          import pandas as pd
          import numpy as np
          from sklearn.model_selection import train_test_split,GridSearchCV
          from sklearn.preprocessing import StandardScaler
          from sklearn.ensemble import RandomForestClassifier #RandomForestRegressor if regression problem
          from sklearn.metrics import accuracy_score,confusion_matrix,classification_report
          import matplotlib.pyplot as plt
          import seaborn as sns

          import warnings
          warnings.filterwarnings('ignore')
```

Finding the Best Random State

```
In [151]: #Finding the Best Random State
          MaxAccu=0
          MaxRS=0

          for i in range (1,100):
              X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=i)
              rf = RandomForestClassifier()
              rf.fit(X_train_ns,y_train_ns)

              y_pred=rf.predict(X_test)
              accuracy=accuracy_score(y_test,y_pred)
              print ('Testing Accuracy' , accuracy , 'RandomState' ,i)
              print('\n')


              if accuracy>MaxAccu:
                  MaxAccu=accuracy
                  MaxRS=i
                  print('MAXINING TESTING SCORE' , MaxAccu , 'ON RANDOM STATE OF' , i)
          Testing Accuracy 0.8653354261132427 RandomState 1


          MAXINING TESTING SCORE 0.8653354261132427 ON RANDOM STATE OF 1
          Testing Accuracy 0.86056853854203 RandomState 2


          Testing Accuracy 0.8626903848389722 RandomState 3


          Testing Accuracy 0.8639693058946634 RandomState 4


          Testing Accuracy 0.8612079990698756 RandomState 5
```



```
In [157]: # train test split
          X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=1)

In [158]: #write one function and call as many time as you want
          def metric_score (clf,X_train_ns,X_test,y_train_ns,y_test,train=True):
              if train:
                  y_pred = clf.predict(X_train_ns)
                  print ('====Training Score====')
                  print (f"Accuracy score : {accuracy_score(y_train_ns,y_pred)*100:2f}%")

              elif train==False:
                  pred = clf.predict(X_test)
                  print ('====Testing Score====')
                  print (f"Accuracy score : {accuracy_score(y_test,pred)*100:2f}%")

                  print ('\n \n Classification report \n ' , classification_report(y_test,pred,digits=2))

In [159]: # model initilization
          clf_rf = RandomForestClassifier()
          clf_rf.fit(X_train_ns,y_train_ns)

Out[159]: RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [160]: # call the function
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 96.520652%
          ====Testing Score====
          Accuracy score : 86.617835%


           Classification report
                         precision    recall  f1-score   support

                     0       0.53      0.34      0.41      4797
                     1       0.90      0.95      0.92     29607

              accuracy                           0.87     34404
             macro avg       0.72      0.65      0.67     34404
          weighted avg       0.85      0.87      0.85     34404
```

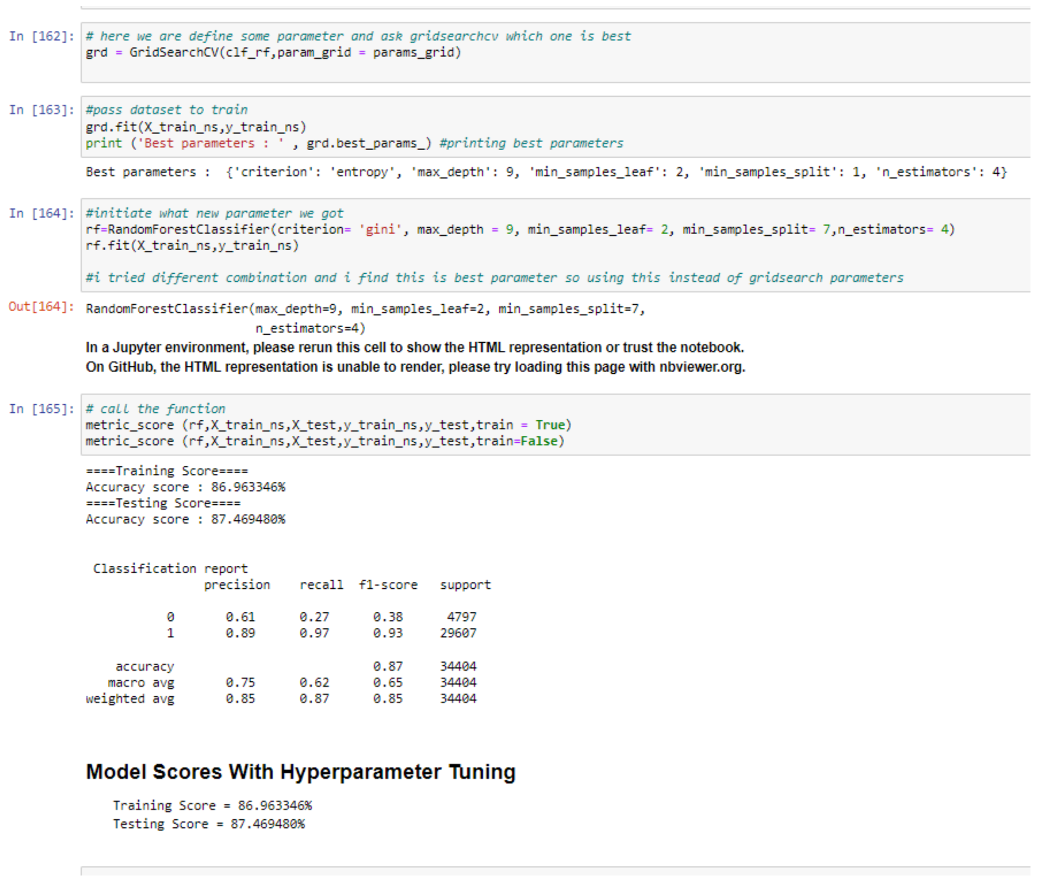
Model Scores

```
    Training Score = 96.520652%
    Testing Score = 86.617835%
```

RF (Random Forest Classifier Model) Score are

Training Score = 96.520652%

Testing Score = 86.617835%



```
In [162]: # here we are define some parameter and ask gridsearchcv which one is best
          grd = GridSearchCV(clf_rf,param_grid = params_grid)

In [163]: #pass dataset to train
          grd.fit(X_train_ns,y_train_ns)
          print ('Best parameters : ' , grd.best_params_) #printing best parameters

          Best parameters :  {'criterion': 'entropy', 'max_depth': 9, 'min_samples_leaf': 2, 'min_samples_split': 1, 'n_estimators': 4}

In [164]: #initiate what new parameter we got
          rf=RandomForestClassifier(criterion= 'gini', max_depth = 9, min_samples_leaf= 2, min_samples_split= 7,n_estimators= 4)
          rf.fit(X_train_ns,y_train_ns)

          #i tried different combination and i find this is best parameter so using this instead of gridsearch parameters

Out[164]: RandomForestClassifier(max_depth=9, min_samples_leaf=2, min_samples_split=7,
                                 n_estimators=4)
          In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
          On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [165]: # call the function
          metric_score (rf,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (rf,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 86.963346%
          ====Testing Score====
          Accuracy score : 87.469480%


           Classification report
                         precision    recall  f1-score   support

                     0       0.61      0.27      0.38      4797
                     1       0.89      0.97      0.93     29607

              accuracy                           0.87     34404
             macro avg       0.75      0.62      0.65     34404
          weighted avg       0.85      0.87      0.85     34404
```

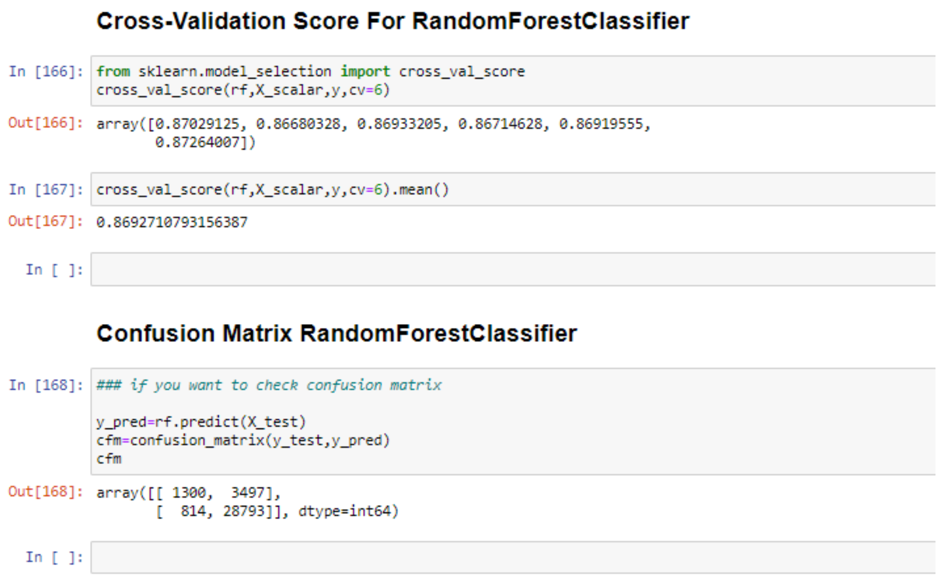
Model Scores With Hyperparameter Tuning

```
Training Score = 86.963346%
Testing Score = 87.469480%
```

Model Score after Hyperparameter Tuning

Training Score = 86.963346%

Testing Score = 87.469480%



Cross-Validation Score For RandomForestClassifier

```
In [166]: from sklearn.model_selection import cross_val_score
          cross_val_score(rf,X_scalar,y,cv=6)

Out[166]: array([0.87029125, 0.86680328, 0.86933205, 0.86714628, 0.86919555,
                 0.87264007])

In [167]: cross_val_score(rf,X_scalar,y,cv=6).mean()

Out[167]: 0.8692710793156387

  In [ ]:
```

Confusion Matrix RandomForestClassifier

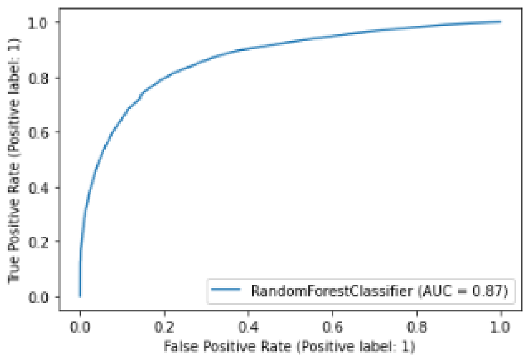
```
In [168]: ### if you want to check confusion matrix

          y_pred=rf.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[168]: array([[ 1300,  3497],
                 [  814, 28793]], dtype=int64)

  In [ ]:
```

Cross Validation score for Random Forest Classifier Model

Cross Validation Score at cv = 6 = 86.92710793156387%

AUC-ROC Curve for Training Data

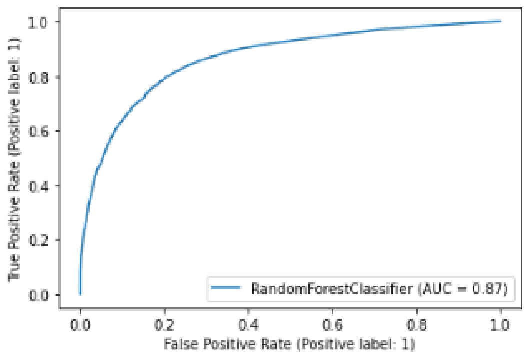
```
In [206]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(rf, X_train_ns,y_train_ns)

Out[206]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x20269004700>
```



```
In [ ]:
```

AUC-ROC Curve for Testing Data

```
In [201]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(rf, X_test, y_test)

Out[201]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x20258b0ffa0>
```



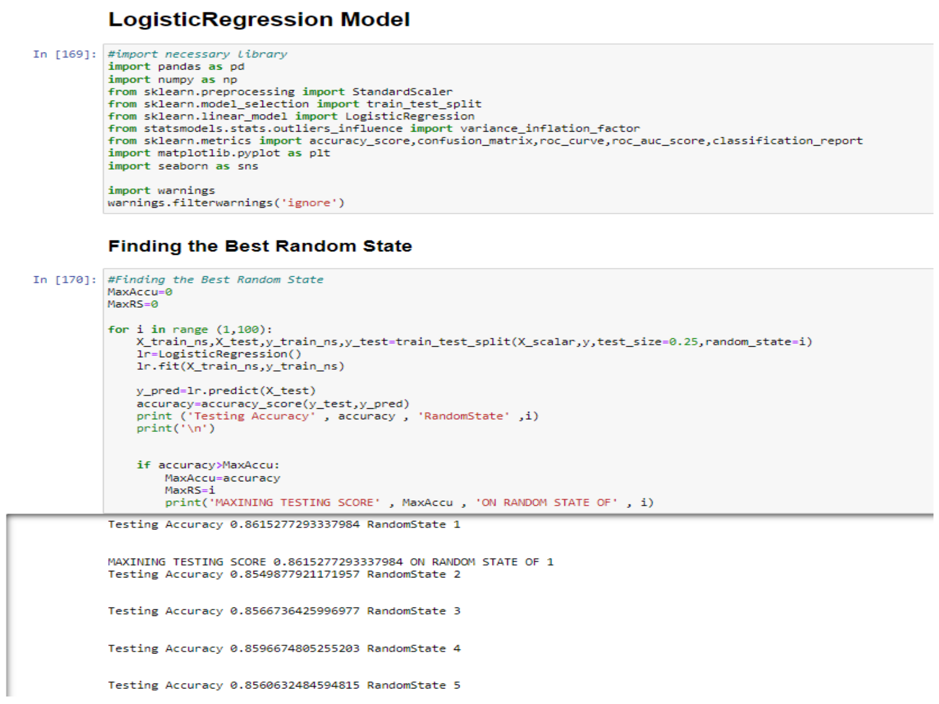
```
In [ ]:
```

<u>AUC-ROC Curve</u>

For Training data, it cover 0.87% Area

For Testing data, it cover 0.87% Area

3rd Model I have Created Logistic Regression Model



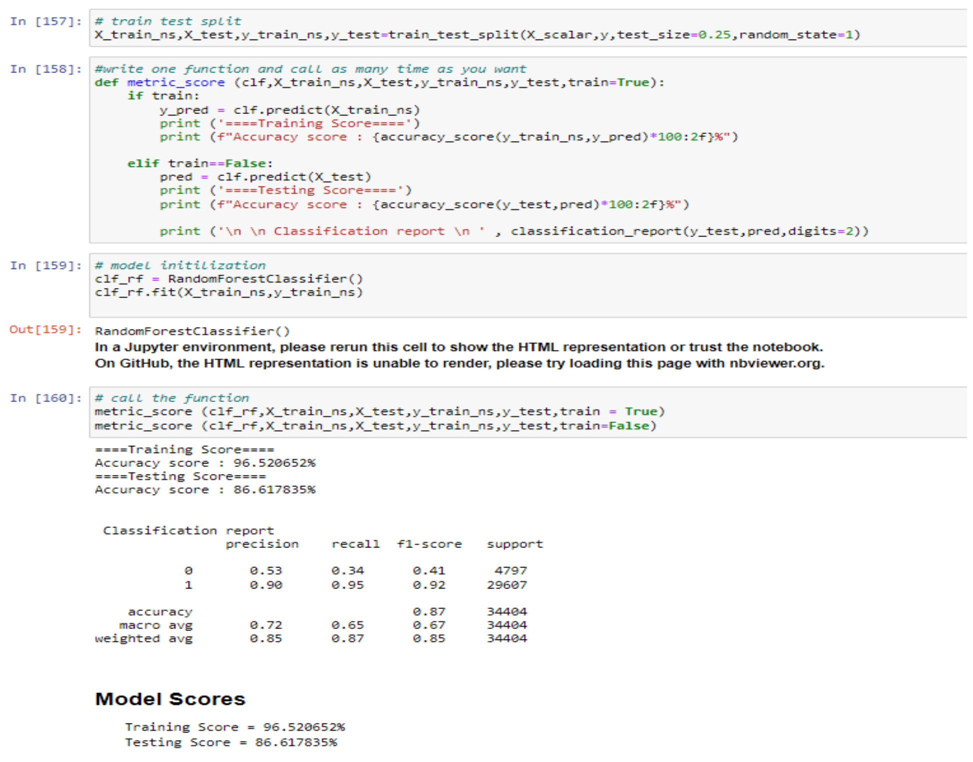
LogisticRegression Model

```
In [169]: #import necessary library
          import pandas as pd
          import numpy as np
          from sklearn.preprocessing import StandardScaler
          from sklearn.model_selection import train_test_split
          from sklearn.linear_model import LogisticRegression
          from statsmodels.stats.outliers_influence import variance_inflation_factor
          from sklearn.metrics import accuracy_score,confusion_matrix,roc_curve,roc_auc_score,classification_report
          import matplotlib.pyplot as plt
          import seaborn as sns

          import warnings
          warnings.filterwarnings('ignore')
```

Finding the Best Random State

```
In [170]: #Finding the Best Random State
          MaxAccu=0
          MaxRS=0

          for i in range (1,100):
              X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=i)
              lr=LogisticRegression()
              lr.fit(X_train_ns,y_train_ns)

              y_pred=lr.predict(X_test)
              accuracy=accuracy_score(y_test,y_pred)
              print ('Testing Accuracy' , accuracy , 'RandomState' ,i)
              print('\n')


              if accuracy>MaxAccu:
                  MaxAccu=accuracy
                  MaxRS=i
                  print('MAXINING TESTING SCORE' , MaxAccu , 'ON RANDOM STATE OF' , i)
```

```
Testing Accuracy 0.8615277293337984 RandomState 1


MAXINING TESTING SCORE 0.8615277293337984 ON RANDOM STATE OF 1
Testing Accuracy 0.8549877921171957 RandomState 2


Testing Accuracy 0.8566736425996977 RandomState 3


Testing Accuracy 0.8596674805255203 RandomState 4


Testing Accuracy 0.8560632484594815 RandomState 5
```



```
In [157]: # train test split
          X_train_ns,X_test,y_train_ns,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=1)

In [158]: #write one function and call as many time as you want
          def metric_score (clf,X_train_ns,X_test,y_train_ns,y_test,train=True):
              if train:
                  y_pred = clf.predict(X_train_ns)
                  print ('====Training Score====')
                  print (f"Accuracy score : {accuracy_score(y_train_ns,y_pred)*100:2f}%")

              elif train==False:
                  pred = clf.predict(X_test)
                  print ('====Testing Score====')
                  print (f"Accuracy score : {accuracy_score(y_test,pred)*100:2f}%")

                  print ('\n \n Classification report \n ' , classification_report(y_test,pred,digits=2))

In [159]: # model initilization
          clf_rf = RandomForestClassifier()
          clf_rf.fit(X_train_ns,y_train_ns)

Out[159]: RandomForestClassifier()
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [160]: # call the function
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_rf,X_train_ns,X_test,y_train_ns,y_test,train=False)

====Training Score====
Accuracy score : 96.520652%
====Testing Score====
Accuracy score : 86.617835%


 Classification report
               precision    recall  f1-score   support

           0       0.53      0.34      0.41      4797
           1       0.90      0.95      0.92     29607

    accuracy                           0.87     34404
   macro avg       0.72      0.65      0.67     34404
weighted avg       0.85      0.87      0.85     34404
```

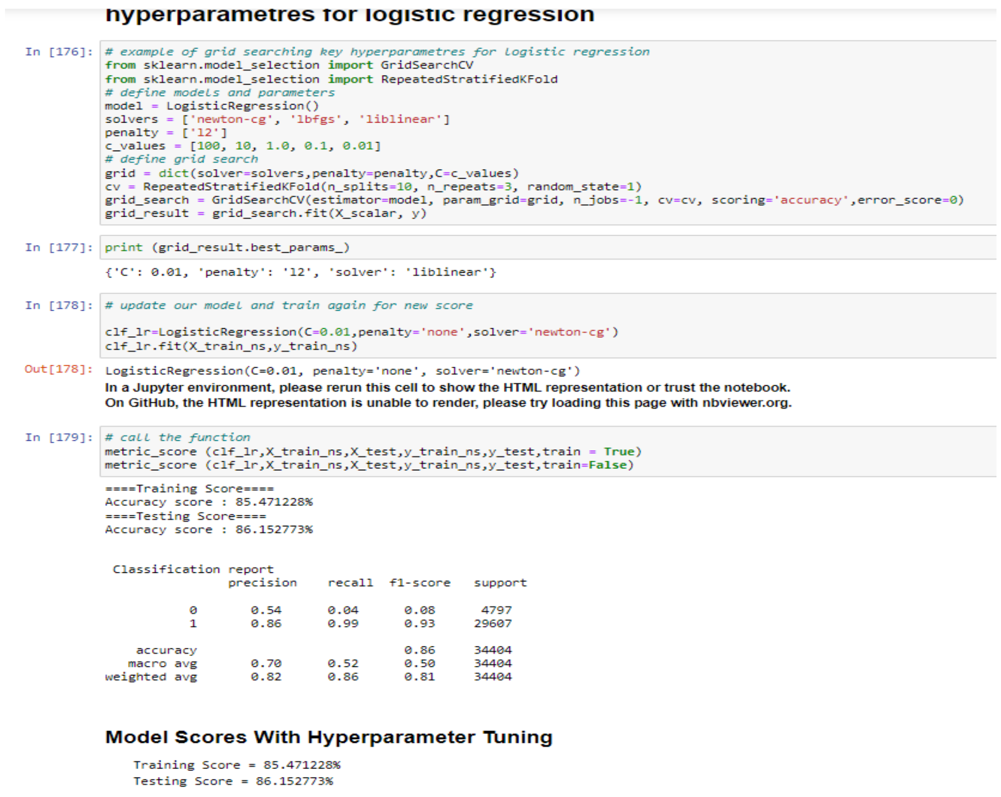
Model Scores

```
    Training Score = 96.520652%
    Testing Score = 86.617835%
```

LR (Logistic Regression Model) Score are

Training Score = 85.471228%

Testing Score = 86.152773%



hyperparametres for logistic regression

```
In [176]: # example of grid searching key hyperparametres for logistic regression
          from sklearn.model_selection import GridSearchCV
          from sklearn.model_selection import RepeatedStratifiedKFold
          # define models and parameters
          model = LogisticRegression()
          solvers = ['newton-cg', 'lbfgs', 'liblinear']
          penalty = ['l2']
          c_values = [100, 10, 1.0, 0.1, 0.01]
          # define grid search
          grid = dict(solver=solvers,penalty=penalty,C=c_values)
          cv = RepeatedStratifiedKFold(n_splits=10, n_repeats=3, random_state=1)
          grid_search = GridSearchCV(estimator=model, param_grid=grid, n_jobs=-1, cv=cv, scoring='accuracy',error_score=0)
          grid_result = grid_search.fit(X_scalar, y)

In [177]: print (grid_result.best_params_)

          {'C': 0.01, 'penalty': 'l2', 'solver': 'liblinear'}

In [178]: # update our model and train again for new score

          clf_lr=LogisticRegression(C=0.01,penalty='none',solver='newton-cg')
          clf_lr.fit(X_train_ns,y_train_ns)

Out[178]: LogisticRegression(C=0.01, penalty='none', solver='newton-cg')
          In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
          On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [179]: # call the function
          metric_score (clf_lr,X_train_ns,X_test,y_train_ns,y_test,train = True)
          metric_score (clf_lr,X_train_ns,X_test,y_train_ns,y_test,train=False)

          ====Training Score====
          Accuracy score : 85.471228%
          ====Testing Score====
          Accuracy score : 86.152773%


           Classification report
                         precision    recall  f1-score   support

                     0       0.54      0.04      0.08      4797
                     1       0.86      0.99      0.93     29607

              accuracy                           0.86     34404
             macro avg       0.70      0.52      0.50     34404
          weighted avg       0.82      0.86      0.81     34404
```

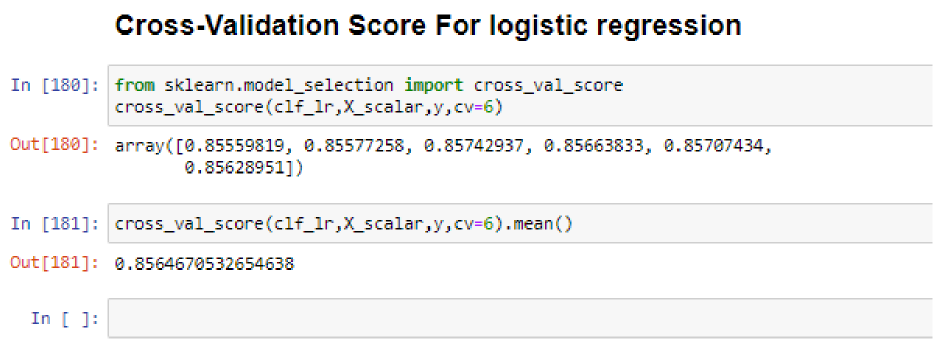
Model Scores With Hyperparameter Tuning

```
    Training Score = 85.471228%
    Testing Score = 86.152773%
```

Model Score after Hyperparameter Tuning

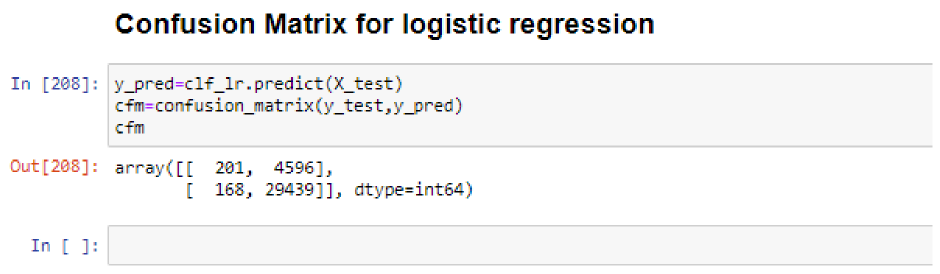
Training Score = 85.471228%

Testing Score = 86.152773%



Cross-Validation Score For logistic regression

```
In [180]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf_lr,X_scalar,y,cv=6)

Out[180]: array([0.85559819, 0.85577258, 0.85742937, 0.85663833, 0.85707434,
                 0.85628951])

In [181]: cross_val_score(clf_lr,X_scalar,y,cv=6).mean()

Out[181]: 0.8564670532654638

In [ ]:
```



Confusion Matrix for logistic regression

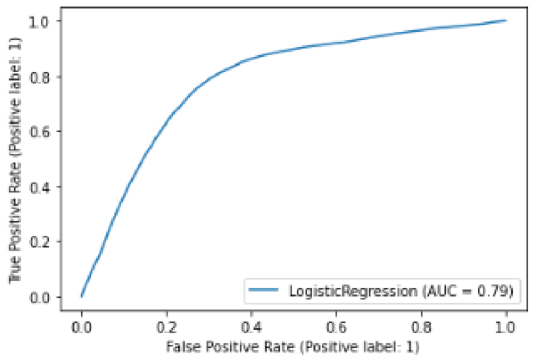
```
In [208]: y_pred=clf_lr.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[208]: array([[  201,  4596],
                 [  168, 29439]], dtype=int64)

In [ ]:
```

Cross Validation score for Decision Tree¶

Cross Validation Score at cv = 6 = 85.64670532654638%

AUC-ROC Curve for Training Data

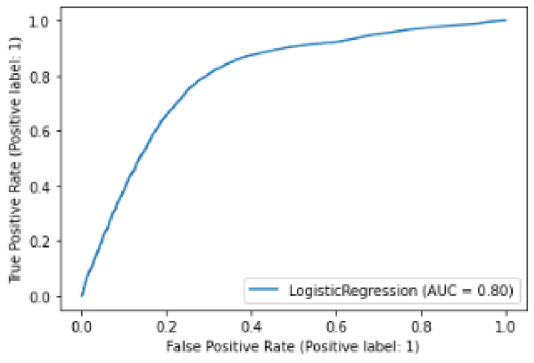
```
In [205]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(clf_lr, X_train_ns,y_train_ns)

Out[205]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x20268c5fee0>
```



```
In [ ]:
```

AUC-ROC Curve for Testing Data

```
In [202]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(clf_lr, X_test, y_test)

Out[202]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2038fcc5bb0>
```



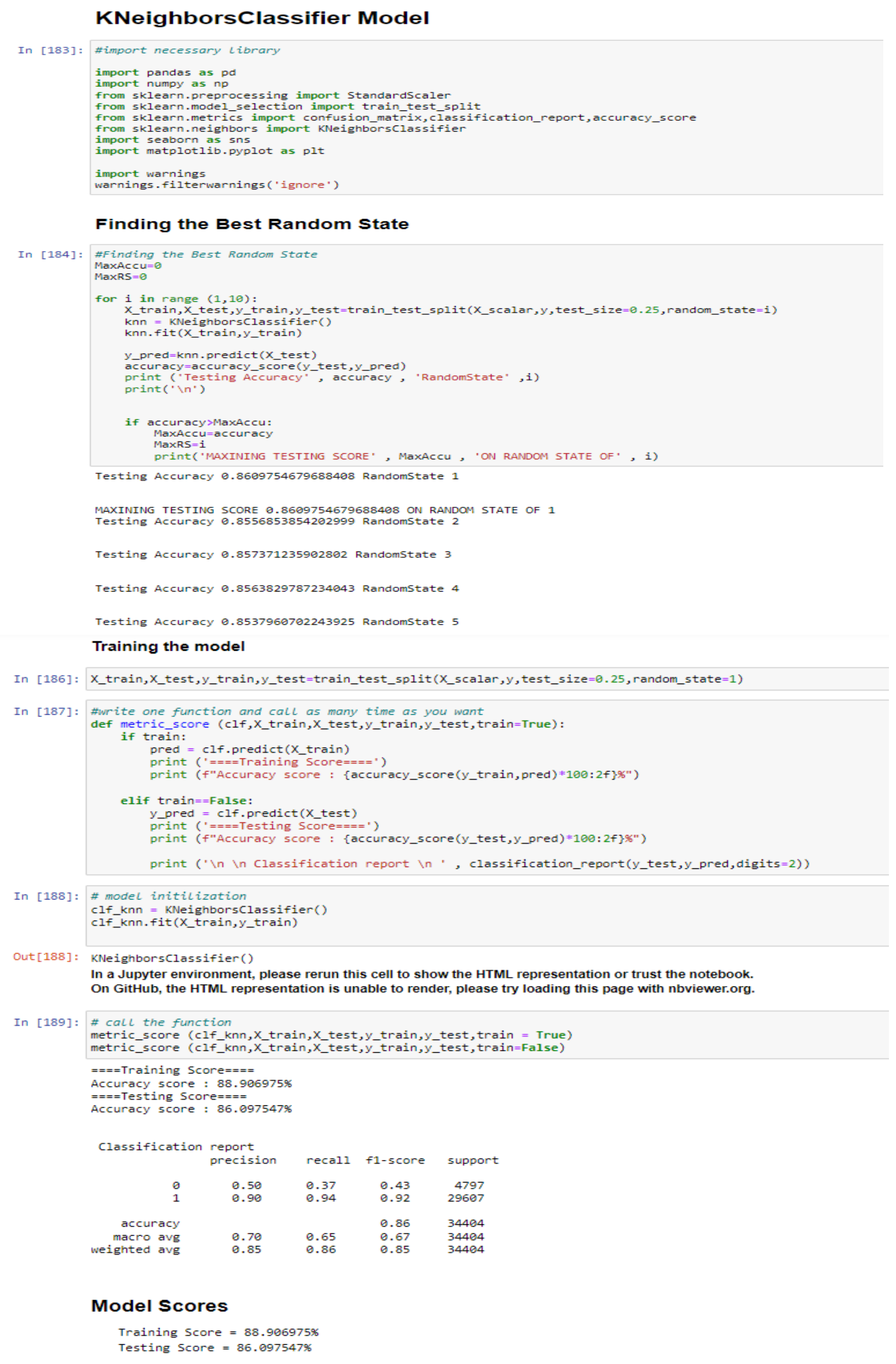
```
In [ ]:
```

AUC-ROC Curve

For Training data, it cover 0.79% Area

For Testing data, it cover 0.80% Area

4th Model I have Created is KNN Model



KNeighborsClassifier Model

```
In [183]: #import necessary library

          import pandas as pd
          import numpy as np
          from sklearn.preprocessing import StandardScaler
          from sklearn.model_selection import train_test_split
          from sklearn.metrics import confusion_matrix,classification_report,accuracy_score
          from sklearn.neighbors import KNeighborsClassifier
          import seaborn as sns
          import matplotlib.pyplot as plt

          import warnings
          warnings.filterwarnings('ignore')
```

Finding the Best Random State

```
In [184]: #Finding the Best Random State
          MaxAccu=0
          MaxRS=0

          for i in range (1,10):
              X_train,X_test,y_train,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=i)
              knn = KNeighborsClassifier()
              knn.fit(X_train,y_train)

              y_pred=knn.predict(X_test)
              accuracy=accuracy_score(y_test,y_pred)
              print ('Testing Accuracy' , accuracy , 'RandomState' ,i)
              print('\n')


              if accuracy>MaxAccu:
                  MaxAccu=accuracy
                  MaxRS=i
                  print('MAXINING TESTING SCORE' , MaxAccu , 'ON RANDOM STATE OF' , i)
          Testing Accuracy 0.8609754679688408 RandomState 1


          MAXINING TESTING SCORE 0.8609754679688408 ON RANDOM STATE OF 1
          Testing Accuracy 0.8556853854202999 RandomState 2


          Testing Accuracy 0.857371235902802 RandomState 3


          Testing Accuracy 0.8563829787234043 RandomState 4


          Testing Accuracy 0.8537960702243925 RandomState 5
```

Training the model

```
In [186]: X_train,X_test,y_train,y_test=train_test_split(X_scalar,y,test_size=0.25,random_state=1)

In [187]: #write one function and call as many time as you want
          def metric_score (clf,X_train,X_test,y_train,y_test,train=True):
              if train:
                  pred = clf.predict(X_train)
                  print ('====Training Score====')
                  print (f"Accuracy score : {accuracy_score(y_train,pred)*100:2f}%")

              elif train==False:
                  y_pred = clf.predict(X_test)
                  print ('====Testing Score====')
                  print (f"Accuracy score : {accuracy_score(y_test,y_pred)*100:2f}%")

                  print ('\n \n Classification report \n ' , classification_report(y_test,y_pred,digits=2))

In [188]: # model initilization
          clf_knn = KNeighborsClassifier()
          clf_knn.fit(X_train,y_train)

Out[188]: KNeighborsClassifier()
```

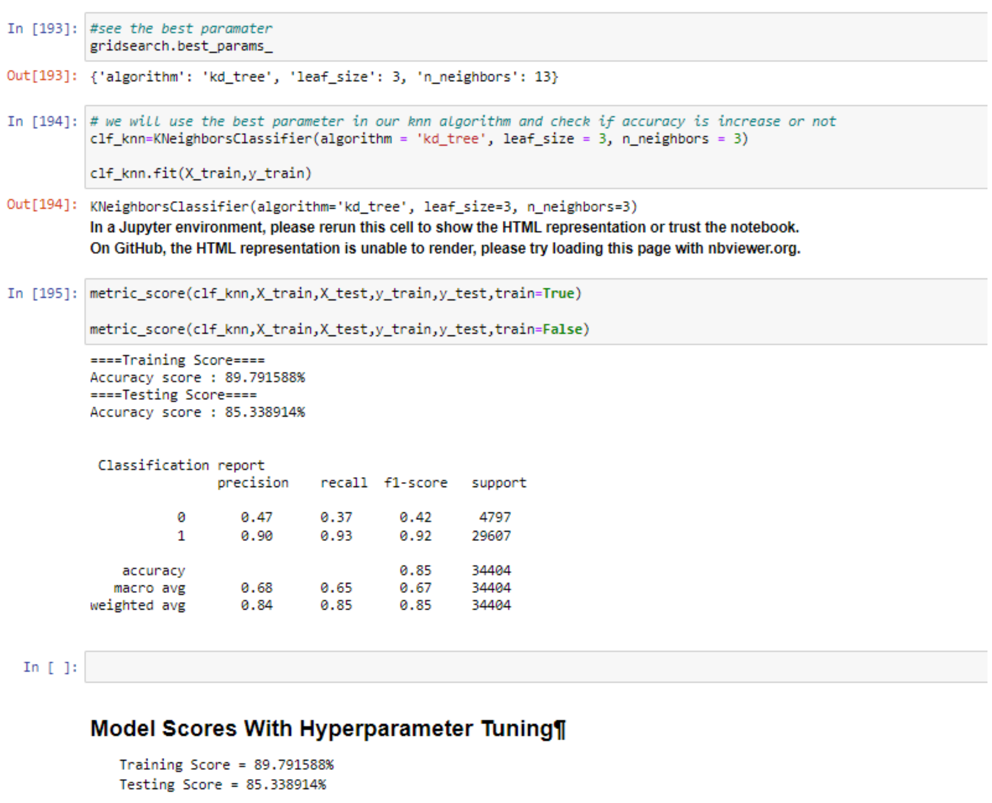
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
In [189]: # call the function
          metric_score (clf_knn,X_train,X_test,y_train,y_test,train = True)
          metric_score (clf_knn,X_train,X_test,y_train,y_test,train=False)

          ====Training Score====
          Accuracy score : 88.906975%
          ====Testing Score====
          Accuracy score : 86.097547%


           Classification report
                         precision    recall  f1-score   support

                     0       0.50      0.37      0.43      4797
                     1       0.90      0.94      0.92     29607

              accuracy                           0.86     34404
             macro avg       0.70      0.65      0.67     34404
          weighted avg       0.85      0.86      0.85     34404
```

Model Scores

```
    Training Score = 88.906975%
    Testing Score = 86.097547%
```

KNN (**KNeighborsClassifier**) Score are
Training Score = 88.906975%
Testing Score = 86.097547%



```
In [193]: #see the best paramater
          gridsearch.best_params_

Out[193]: {'algorithm': 'kd_tree', 'leaf_size': 3, 'n_neighbors': 13}

In [194]: # we will use the best parameter in our knn algorithm and check if accuracy is increase or not
          clf_knn=KNeighborsClassifier(algorithm = 'kd_tree', leaf_size = 3, n_neighbors = 3)

          clf_knn.fit(X_train,y_train)

Out[194]: KNeighborsClassifier(algorithm='kd_tree', leaf_size=3, n_neighbors=3)
          In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
          On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [195]: metric_score(clf_knn,X_train,X_test,y_train,y_test,train=True)

          metric_score(clf_knn,X_train,X_test,y_train,y_test,train=False)

          ====Training Score====
          Accuracy score : 89.791588%
          ====Testing Score====
          Accuracy score : 85.338914%


           Classification report
                         precision    recall  f1-score   support

                      0       0.47      0.37      0.42      4797
                      1       0.90      0.93      0.92     29607

               accuracy                           0.85     34404
              macro avg       0.68      0.65      0.67     34404
           weighted avg       0.84      0.85      0.85     34404


In [ ]:
```

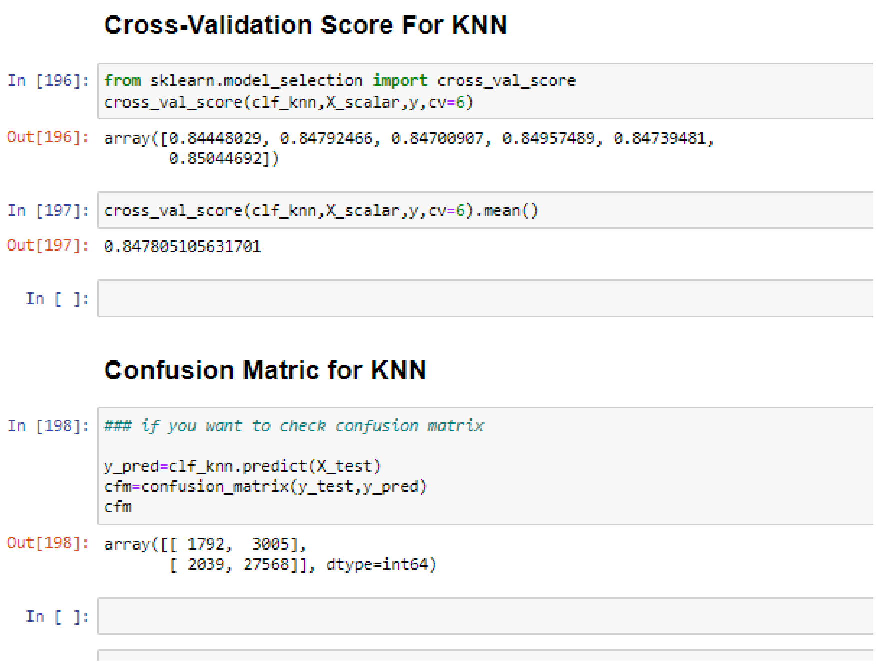
Model Scores With Hyperparameter Tuning¶

```
    Training Score = 89.791588%
    Testing Score = 85.338914%
```

Model Score after Hyperparameter Tuning

Training Score = 89.791588%

Testing Score = 85.338914%



Cross-Validation Score For KNN

```
In [196]: from sklearn.model_selection import cross_val_score
          cross_val_score(clf_knn,X_scalar,y,cv=6)

Out[196]: array([0.84448029, 0.84792466, 0.84700907, 0.84957489, 0.84739481,
                 0.85044692])

In [197]: cross_val_score(clf_knn,X_scalar,y,cv=6).mean()

Out[197]: 0.847805105631701

In [ ]:
```

Confusion Matric for KNN

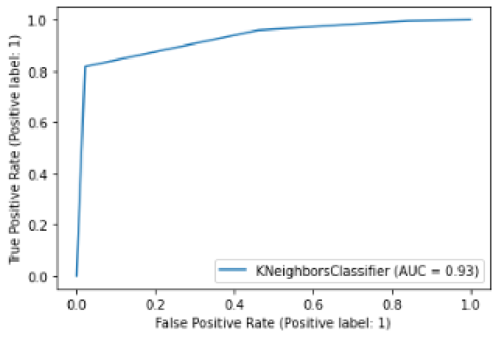
```
In [198]: ### if you want to check confusion matrix

          y_pred=clf_knn.predict(X_test)
          cfm=confusion_matrix(y_test,y_pred)
          cfm

Out[198]: array([[ 1792,  3005],
                 [ 2039, 27568]], dtype=int64)

In [ ]:
```

Cross Validation score for Decision Tree

Cross Validation Score at cv = 6 = 85.64670532654638%

AUC-ROC Curve for Training Data

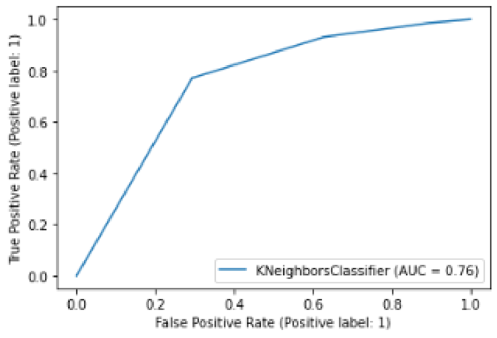
```
In [204]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(clf_knn, X_train_ns,y_train_ns)

Out[204]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2026a4a9940>
```



```
In [ ]:
```

AUC-ROC Curve for Testing Data

```
In [203]: from sklearn.metrics import RocCurveDisplay
          RocCurveDisplay.from_estimator(clf_knn, X_test, y_test)

Out[203]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x2038fcfdbe0>
```



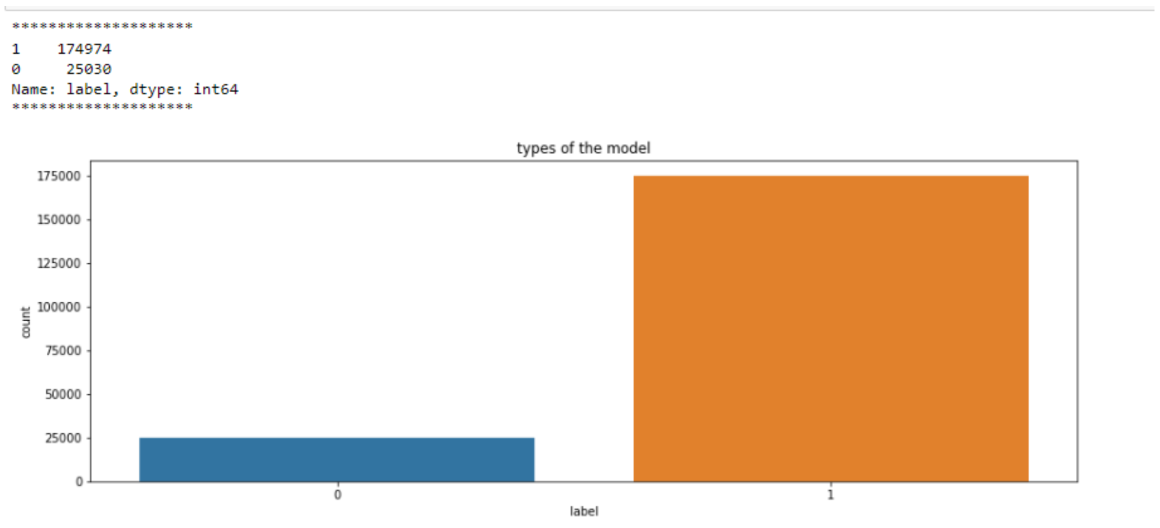
```
In [ ]:
```

AUC-ROC Curve

For Training data, it cover 0.93% Area

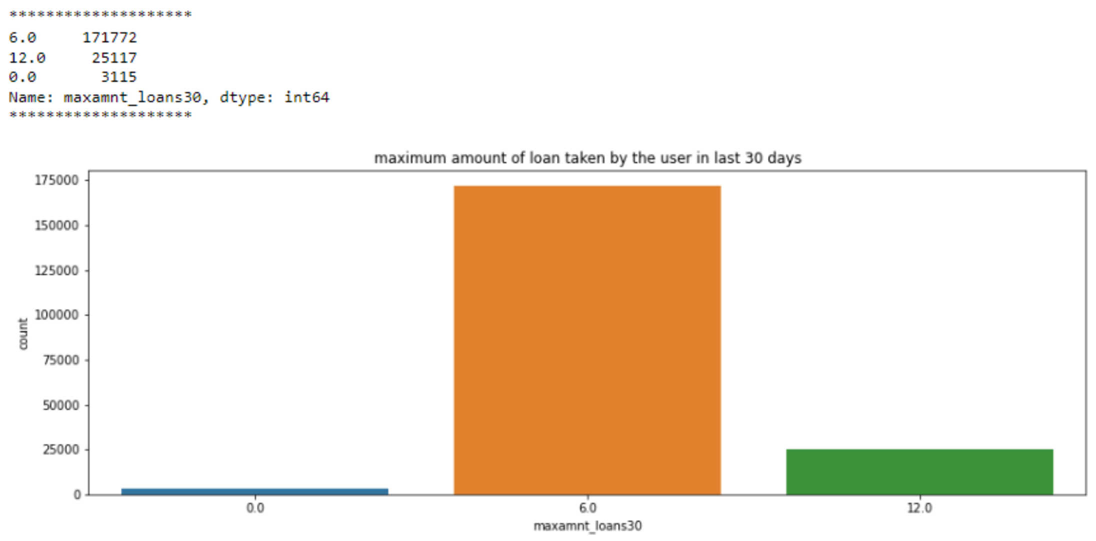
For Testing data, it cover 0.76% Area

Visualizations and EDA



```
********************
1    174974
0     25030
Name: label, dtype: int64
********************
```

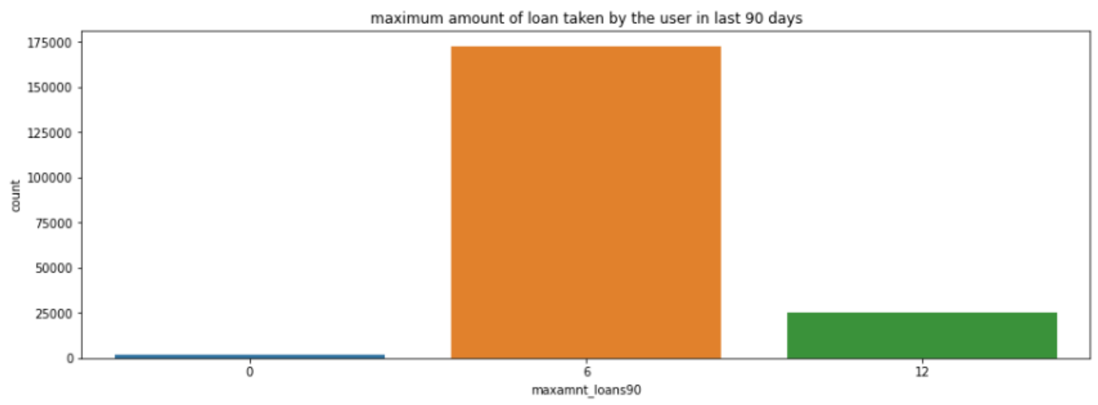
The bar graph shows that the majority of people (176,707) have successfully repaid their loans, while only 25,314 are in default.



```
********************
6.0     171772
12.0     25117
0.0       3115
Name: maxamnt_loans30, dtype: int64
********************
```

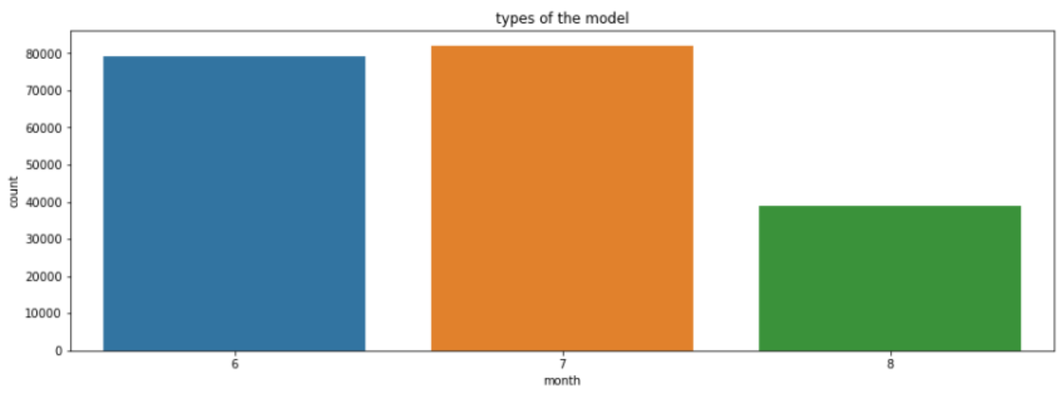
The bar graph indicates that the most common loan amount taken was 5, to be repaid as 6, with 173507 individuals taking out this loan. Additionally, 25360 people took out a loan of 10, to be repaid as 12. Lastly, there are 3154 individuals who did not take out a loan.

```
********************
6     172590
12     25470
0       1944
Name: maxamnt_loans90, dtype: int64
********************
```

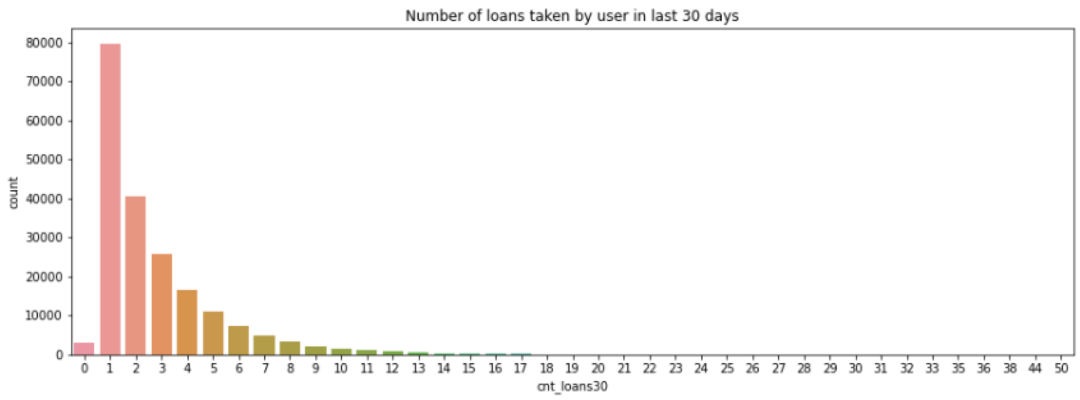


According to the bar graph, the most frequently borrowed loan amount was 5, with a repayment rate of 6. This loan amount was taken out by 173507 individuals. There were also 25360 individuals who borrowed 10 with a repayment rate of 12. Lastly, the graph shows that 3154 people did not borrow any loan at all.

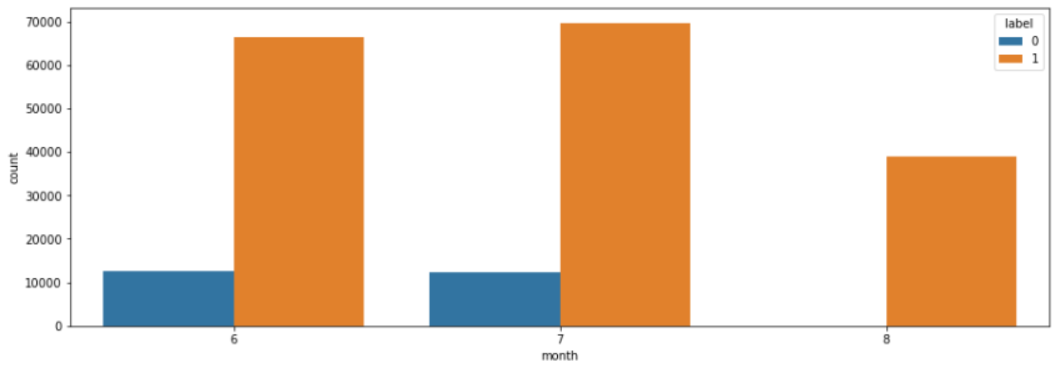
```
********************
7    82039
6    79067
8    38898
Name: month, dtype: int64
********************
```



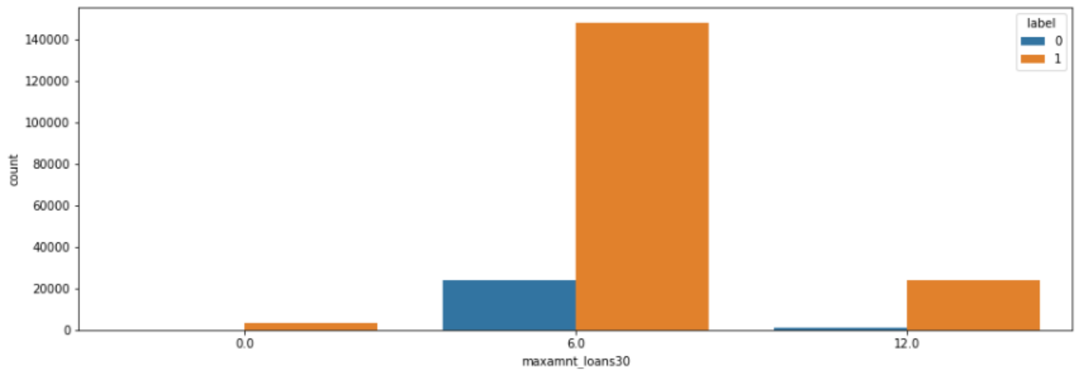
The bar graph shows that the highest amount of loans were issued in the month of July with 82858, followed by June with 79890, and the least amount was issued in August with 39273.



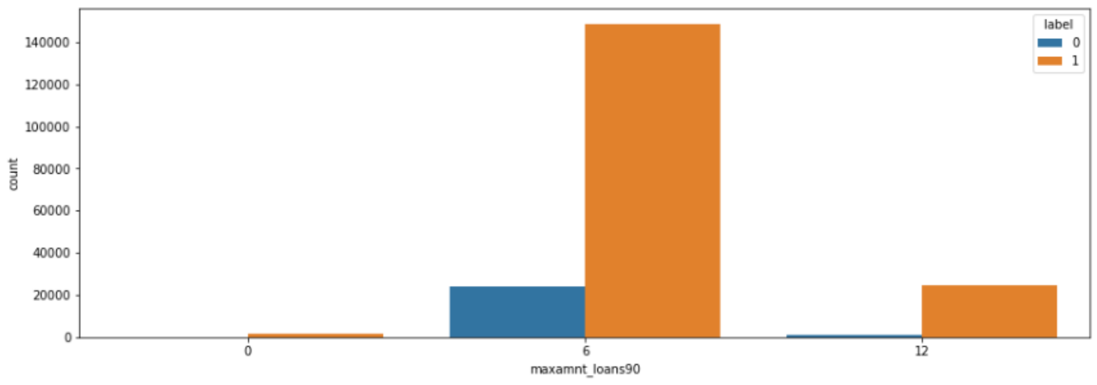
The bar chart reveals that, based on the dataset, most people have only taken one loan within the past 30 days.



The graph illustrates that in July (month 7), the highest number of loans were taken and 84.92% of those loans were successfully repaid. June (month 6) had the second highest number of loans taken with 83.98% repayment success rate. Lastly, August (month 8) had the least number of loans taken but had a 100% repayment success rate.



The bar chart above displays the maximum amount of loans taken by users in the last 30 days versus the repayment amounts. It can be seen that when the loan amount is 5 and the repayment amount is 6, 86.00% (171772) of individuals successfully repay their loan within 30 days. Similarly, when the loan amount is 10 and the repayment amount is 12, 96.07% (25117) of individuals successfully repay their loan within 30 days. Additionally, the chart indicates that 3115 people did not take any loan.



The bar chart above illustrates the maximum amount of loans taken by users in the last 90 days in relation to the repayment amounts. It can be seen that when the loan amount is 5 and the repayment amount is 6, 86.06% (172590) of users successfully repay their loans within 90 days. On the other hand, when the loan amount is 10 and the repayment amount is 12, 96.12% (25470) of users successfully repay their loans within 90 days. Additionally, the chart also indicates that 1944 people did not take any loan during this period.

CONCLUSION

Key Findings and Conclusions of the Study

After tuning the hyperparameters, the Random Forest Classifier has been determined to be the best model for this problem, as it has a training score of 87% and a testing score of 87% according to the AUC-ROC curve.

Learning Outcomes of the Study in respect of Data Science

1) First we identify null values and there were no null values present in dataset

2) Then I identified duplicates and there were no duplicates

3) Performed EDA and wrote all observations for each graph

4) Then i dropped unnecessary columns

5) Then also plotted the Distribution plot and regression plot

6) Then plotted boxplot to remove outliers

7) Then treated outliers with the Z-score method

8) Then scaled data and Also check for VIF

9) Then find the co-relation between feature and label by the CORR method

10) Then selected the top features by busing Selectkbest technique

11) Then created 4 models that is Logistic Regressor, Random Forest Regressor ,linear Regressor model, KNN Modelwith hyperparameter tuning for all 4 models and also Cross-validations

12) At last I selected the best model according to their CV score and Training(R2) and testing score(R2)

Limitations of this work and Scope for Future Work

The limitations of this work on the microcredit defaulter model include a lack of comprehensive and diverse data on the individuals and businesses who have defaulted on their loans. This can lead to a lack of understanding of the specific factors that contribute to loan defaults and an over-reliance on certain variables or features. Additionally, there may be limitations in the analysis and modeling techniques used, such as a lack of robustness and interpretability of the model.

Scope for future work in this area could include the collection of more diverse and comprehensive data on loan defaulters, as well as the development of more advanced machine learning models to better predict loan defaults. Additionally, research could be conducted to better understand the factors that contribute to loan defaults, such as economic and social factors, in order to develop more tailored lending strategies. Another area that can be explored is using unstructured data such as text data from customer interactions for risk scoring. Additionally, interpretability methods such as LIME, SHAP etc can be used to understand the factors that are most important in the model.