Configuration Manual

MSc Research Project
Data Analytics

Tushar Patil
Student ID: X19199988

School of Computing
National College of Ireland

Supervisor: Bharathi Chakravarthi

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Tushar Patil
Student ID:	X19199988
Programme:	Data Analytics
Year:	2020-2021
Module:	MSc Research Project
Supervisor:	Bharathi Chakravarthi
Submission Due Date:	16/08/2021
Project Title:	Configuration Manual
Word Count:	1247
Page Count:	21

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Tushar Patil
Date:	16th August 2021

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Contents

Configuration Manual

Tushar Patil
X19199988

1 Introduction

The goal of this paper is to provide a quick overview of the processes required in putting this project into action. The main objective of this research was to test the efficiency of the proposed approach of using CT-GAN for the data augmentation method to handle class imbalance issues in credit card fraud prediction tasks. We have implemented the proposed architecture with help of 3 different classifiers and tested the performance of the same for both unbalanced and with balanced data. The subsequent sections of this handbook discuss the tools and strategies that were utilized to achieve the defined goals.

2 System Specification

The system configuration on which research work has been carried out is mentioned below.

- **Operating System:** Windows 10 Home
- **System Type:** 64 bit
- **Installed Memory (RAM):** 16 GB
- **Hard Drive:** 500 GB SSD
- **Processor:** Intel® Core™ i5-9300H CPU @ 2.40GHz
- **GPU:** GeForce GTX 1650 - 4GB

3 Tools and Technologies

The Python programming language was utilized to complete this project, with Google Colaboratory serving as the coding platform for developing and processing our code. Google Colaboratory is a free cloud platform provided by Google. It is free for usage and provides free access to GPU/TPU for python coding.

- Python 3.7.11

4 Environmental Setup

As mentioned above, we have used Google Colaboratory for the development of this research, which does not require any environmental set-up on a local machine. The following steps could be followed to configure and run the coding files associated with this research. Step 2 and 3 are alternatives to each other, and either one should be followed. After Initialization of the session, the runtime type should be changes to GPU for faster processing. [1]

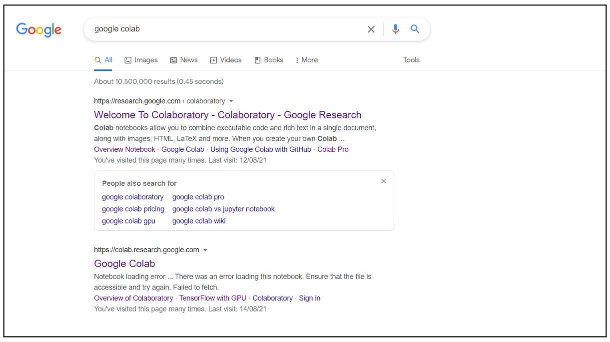


Figure 1: Step 1: Google search for 'google colab'

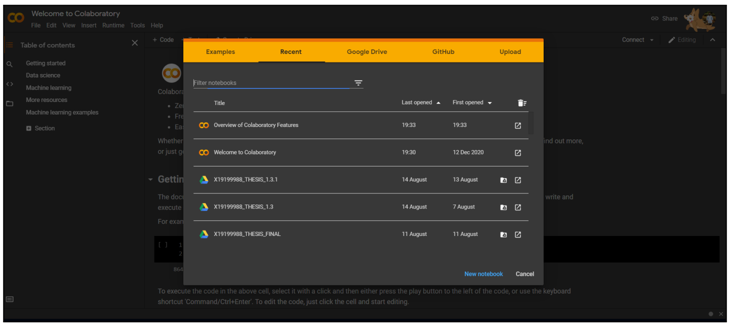


Figure 2: Step 2: Open an existing Jupyter Notebook or create a new one.

Figure 3: Step 3: Upload an existing Jupyter notebook from the local system.

[1] https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index#recent=true

5 Data Selection and Collection

The dataset utilized in this research has been fetched from the open-source dataset platform Kaggle which is a public repository. The source URL of the data is given below. Also, the features of the selected dataset are discussed below.[2]

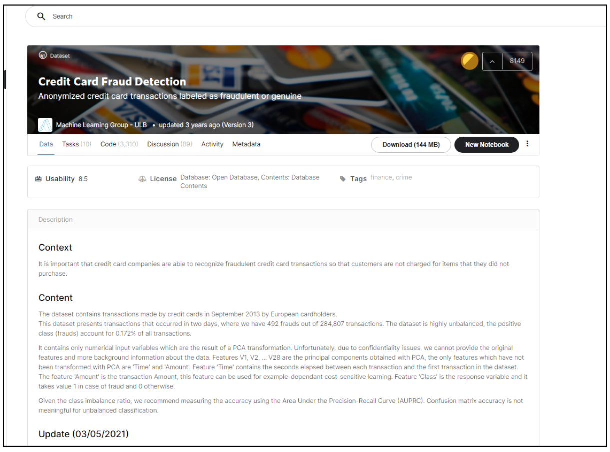
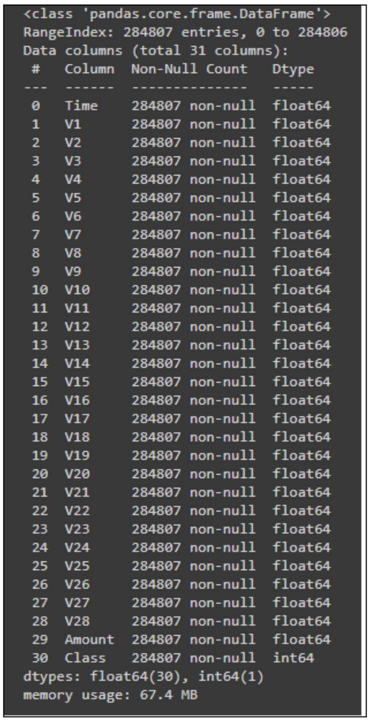


Figure 4: kaggle Dataset



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 284807 entries, 0 to 284806
Data columns (total 31 columns):
 #   Column  Non-Null Count   Dtype
---  ------  --------------   -----
 0   Time    284807 non-null  float64
 1   V1      284807 non-null  float64
 2   V2      284807 non-null  float64
 3   V3      284807 non-null  float64
 4   V4      284807 non-null  float64
 5   V5      284807 non-null  float64
 6   V6      284807 non-null  float64
 7   V7      284807 non-null  float64
 8   V8      284807 non-null  float64
 9   V9      284807 non-null  float64
 10  V10     284807 non-null  float64
 11  V11     284807 non-null  float64
 12  V12     284807 non-null  float64
 13  V13     284807 non-null  float64
 14  V14     284807 non-null  float64
 15  V15     284807 non-null  float64
 16  V16     284807 non-null  float64
 17  V17     284807 non-null  float64
 18  V18     284807 non-null  float64
 19  V19     284807 non-null  float64
 20  V20     284807 non-null  float64
 21  V21     284807 non-null  float64
 22  V22     284807 non-null  float64
 23  V23     284807 non-null  float64
 24  V24     284807 non-null  float64
 25  V25     284807 non-null  float64
 26  V26     284807 non-null  float64
 27  V27     284807 non-null  float64
 28  V28     284807 non-null  float64
 29  Amount  284807 non-null  float64
 30  Class   284807 non-null  int64
dtypes: float64(30), int64(1)
memory usage: 67.4 MB
```

Figure 5: Data Description

The selected dataset has 31 columns, out of which 28 columns are resulting components of PCA transformation. PCA transformation is carried out for hiding sensitive customer information. Time and amount are only non-transformed features present in data. We have a binary target variable-'Class'.

[2]https://www.kaggle.com/mlg-ulb/creditcardfraud

6 Implementation

6.1 Installing and Importing Required Packages

In this section of code, we have installed all the required packages for this research. All the packages are installed and imported in the coding environment using 'pip'. Figure:- 6,7

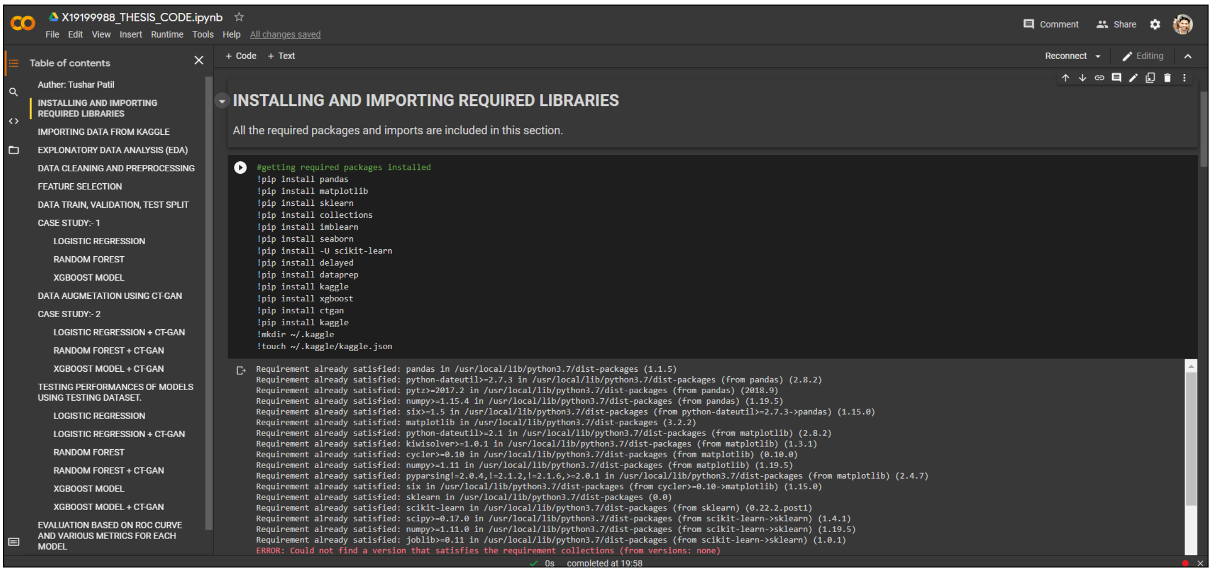


Figure 6: Installing required packages

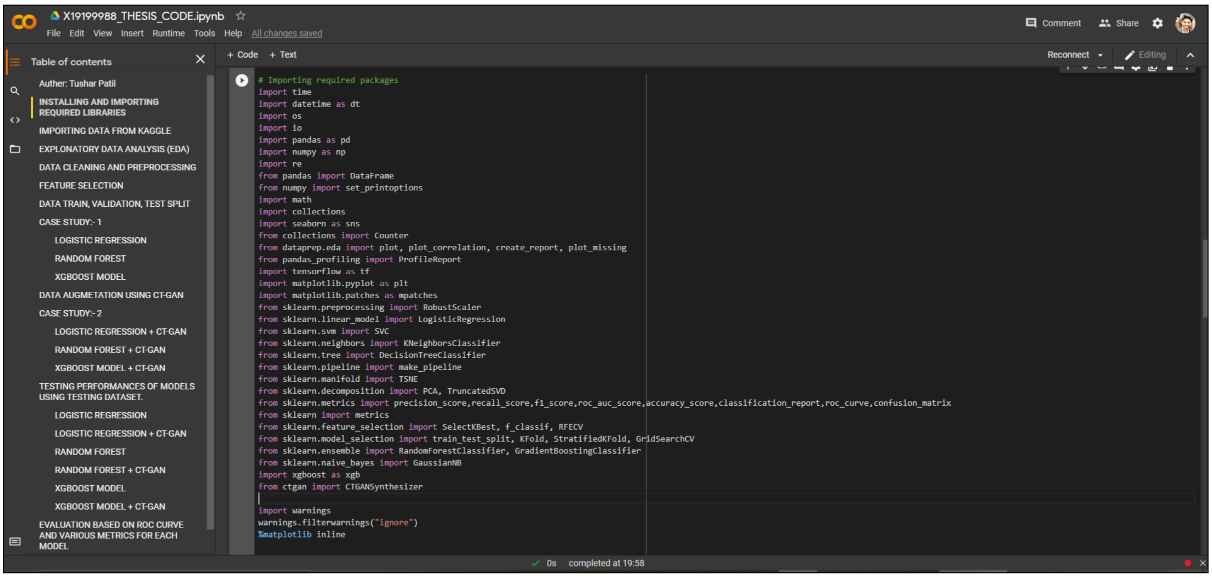


Figure 7: Importing installed packages

6.2 Importing selected Data from Kaggle

For getting our data, we have used kaggle API to remove the dependency of uploading the data into code each time. The code for the same is as given below. Figure:- 8

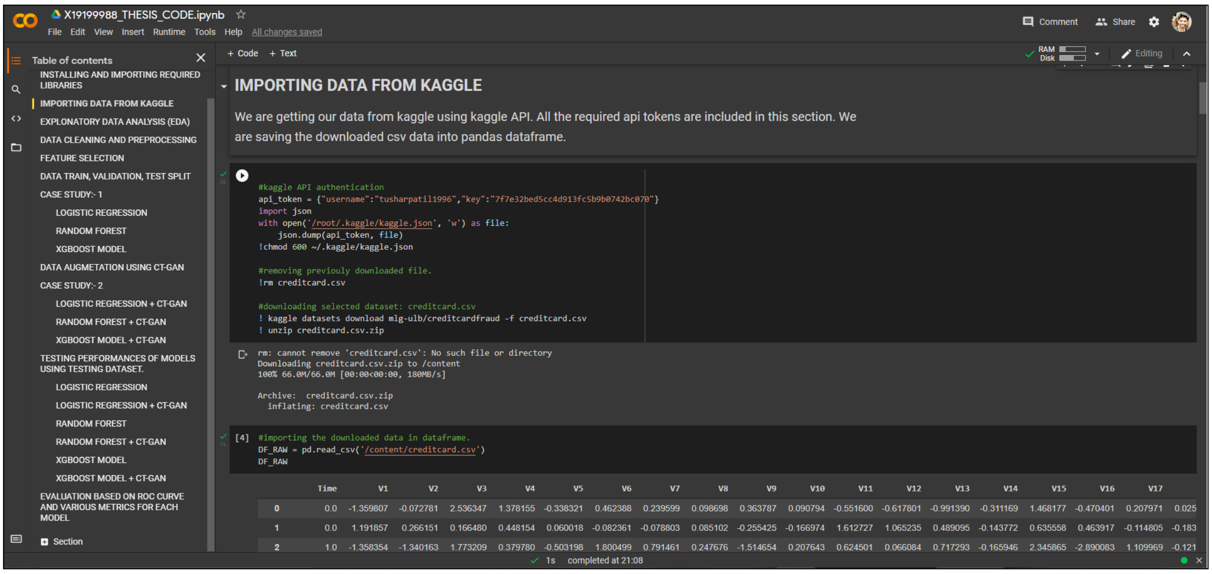


Figure 8: Importing dataset from kaggle

6.3 Exploratory Data Analysis

It is important to understand the patterns and nature of data before going ahead with any pre-processing step. We have done exploratory data analysis using visualizations in this section. The code for the same is as following. Figure:- 9

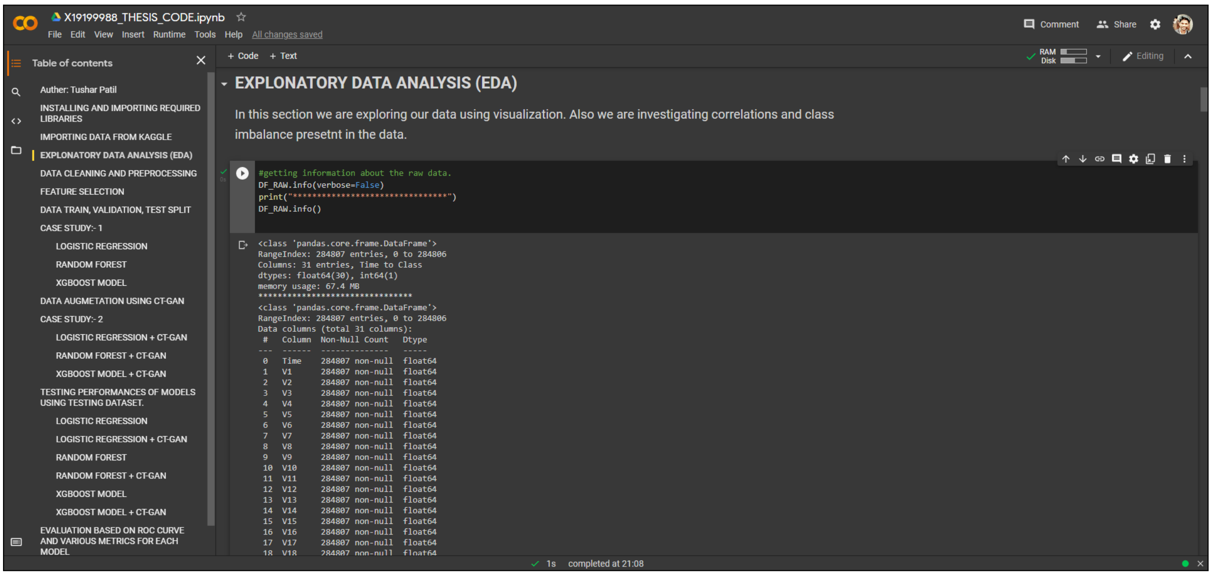


Figure 9: Understanding Features in dataset

To understand the distribution of the target variable, we have used a bar graph from **Matplotlib** package. The following code snippet gets the plot of target variable distribution. Figure:- 10, 11

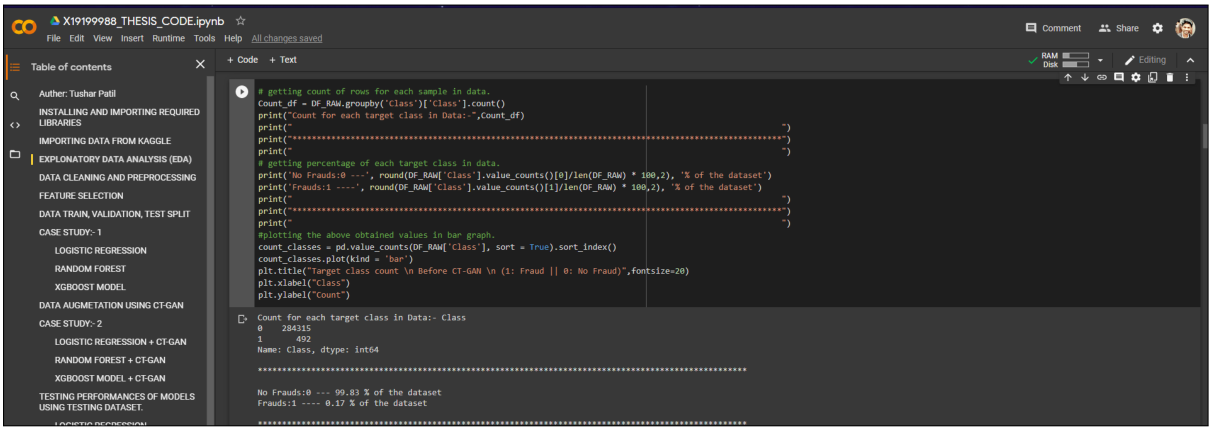


Figure 10: Getting distribution of target feature

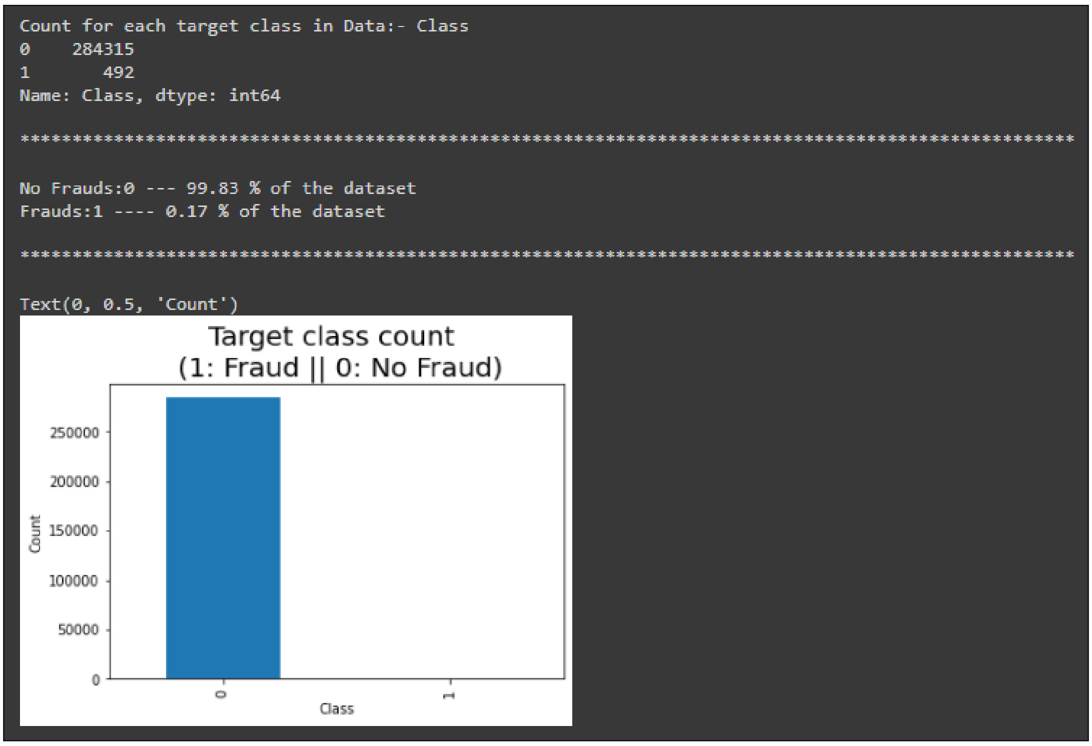


Figure 11: distribution of target feature

To understand the correlation between features of data, we are using correlation matrix. We have also plotted correlation matrix for subsample of original data to understand the correlations between feature more accurately. Figure:- 12, 13,14

To understand the distribution and nature of all the features, we have used the **dataprep** package from **Pandas** to get an automatic report for our data. Figure:- 15

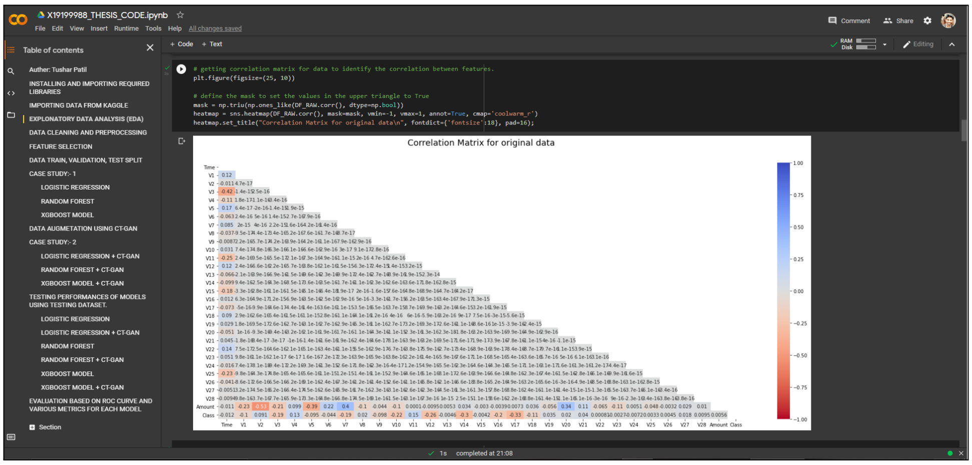


Figure 12: Correlation matrix for original dataset

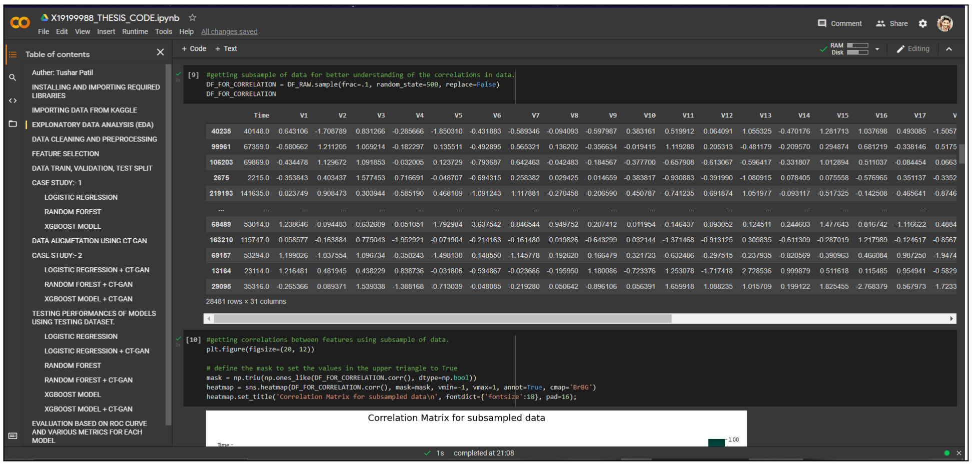


Figure 13: Getting subsample of data

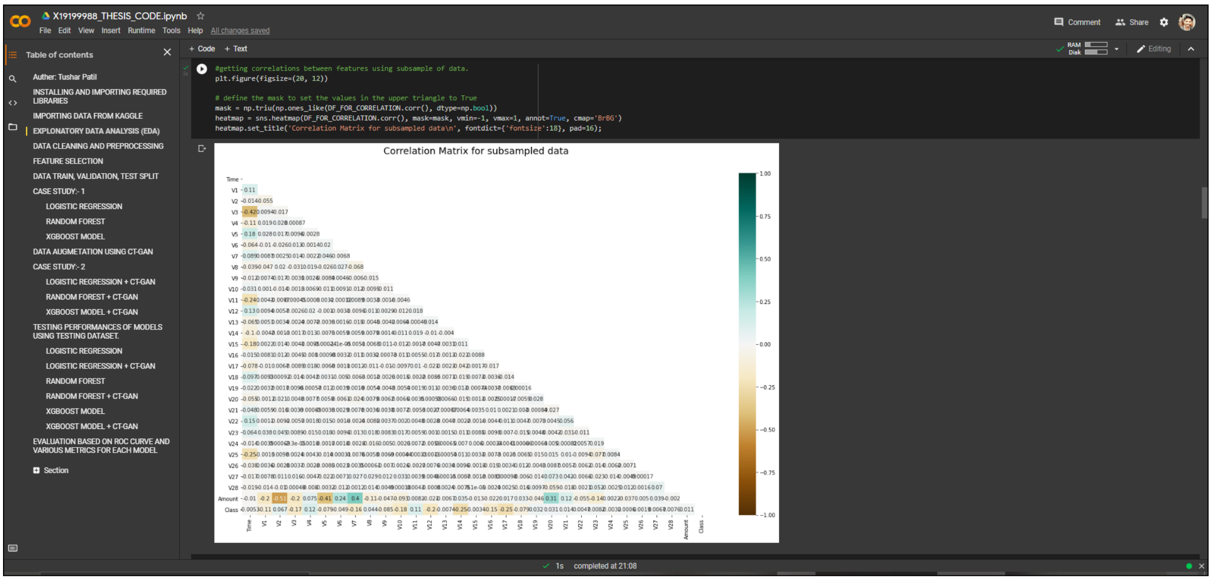


Figure 14: Correlation matrix for subsample of data

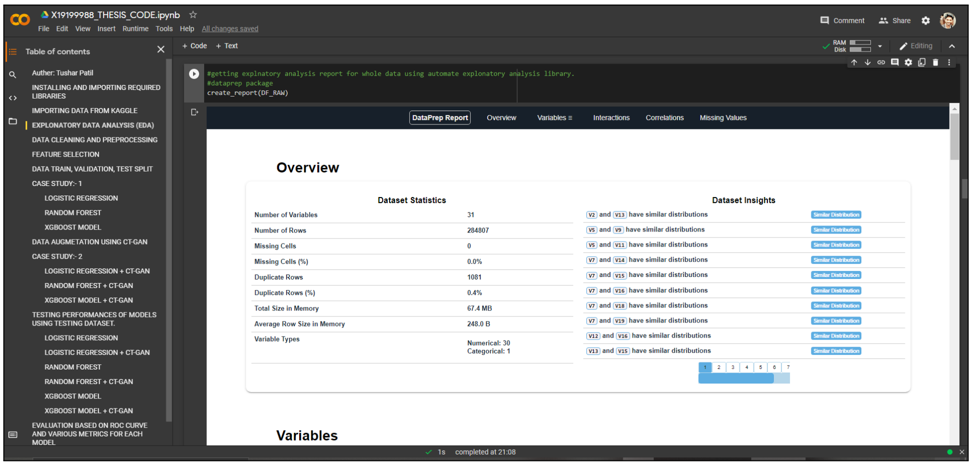


Figure 15: Auto explanatory analysis

6.4 Data Cleaning and Preprocessing

Data cleaning is one of the most important stages in the data analysis process. We are checking for any missing or duplicate values in our data and removing the same from the data. Also, we are scaling our Time and Amount features to keep them at the same scale as other variables. Figure:- 16

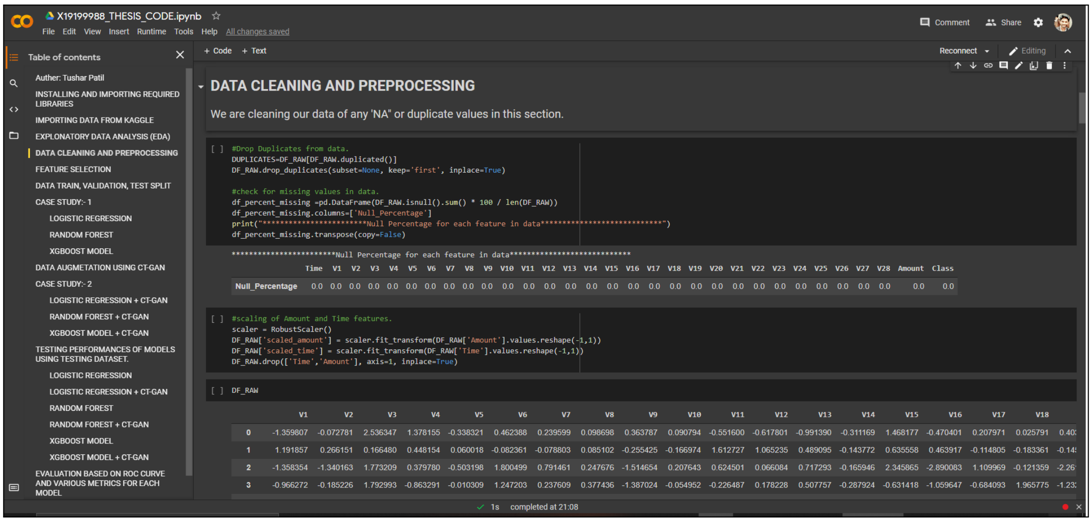


Figure 16: Data cleaning and preprocessing

6.5 Feature Selection

To get rid out non-required features from our data, we are using **SelectKBest** feature selection technique from **Scikit-learn** package for getting most relevant features for our further analysis. Below, given code explains the steps involved for getting the top 14 best features from our dataset. Figure:- 17, 18, 19

To verify our selection of features, we are getting correlation between remaining features. Figure:- 20

X19199988_THESIS_CODE.ipynb

File Edit View Insert Runtime Tools Help All changes saved

Table of contents

- Auther: Tushar Patil
- INSTALLING AND IMPORTING REQUIRED LIBRARIES
- IMPORTING DATA FROM KAGGLE
- EXPLORATORY DATA ANALYSIS (EDA)
- DATA CLEANING AND PREPROCESSING
- FEATURE SELECTION
- DATA TRAIN, VALIDATION, TEST SPLIT
- CASE STUDY:- 1
  - LOGISTIC REGRESSION
  - RANDOM FOREST
  - XGBOOST MODEL
- DATA AUGMETATION USING CT-GAN
- CASE STUDY:- 2
  - LOGISTIC REGRESSION + CT-GAN
  - RANDOM FOREST + CT-GAN
  - XGBOOST MODEL + CT-GAN
- TESTING PERFORMANCES OF MODELS USING TESTING DATASET.
  - LOGISTIC REGRESSION
  - LOGISTIC REGRESSION + CT-GAN
  - RANDOM FOREST
  - RANDOM FOREST + CT-GAN
  - XGBOOST MODEL
  - XGBOOST MODEL + CT-GAN
- EVALUATION BASED ON ROC CURVE AND VARIOUS METRICS FOR EACH MODEL

FEATURE SELECTION

In this section we are using KBestClass model for detection and selection of best predictors from our data. We are removing other features from data for further analysis as they are redundant towards our target variable.

```
#defining function for feature selection using KBestClass
def Feature_selection_model_KBest(datafile, response_variable):
    X = datafile.drop(response_variable, axis=1)
    target = datafile[response_variable]

    # Univariate Selection using KBestClass feature extraction
    test = SelectKBest(score_func=f_classif, k=14)
    fit = test.fit(X, target)

    # summarize scores
    set_printoptions(precision=3)
    features = fit.transform(X)

    # #plotting the above obtained values in bar graph.
    X_indices = np.arange(X.shape[-1])
    scores = fit.scores_
    scores /= scores.max()
    plt.bar(X_indices, scores, width=.5)
    plt.title("K best Features",fontsize=20)
    plt.xlabel("Feature")
    plt.ylabel("Feature importance score")

    # fetch selected columns
    selected_features = fit.scores_.argsort()[-14:][::-1]
    return X.iloc[:, selected_features].copy()
```

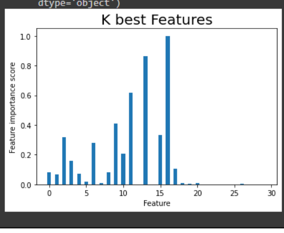
```
[ ] #calling the function for feature selection.
    Features_Kbest = Feature_selection_model_KBest(DF_RAW,'Class')
    print(Features_Kbest.columns)
```

✓ 1s completed at 21:08

Figure 17: Defining a function for selection of K best Features using SelectKBest method

```
#calling the function for feature selection.
Features_Kbest = Feature_selection_model_KBest(DF_RAW,'Class')
print(Features_Kbest.columns)
```

```
Index(['V17', 'V14', 'V12', 'V10', 'V16', 'V3', 'V7', 'V11', 'V4', 'V18', 'V1',
       'V9', 'V5', 'V2'],
      dtype='object')
```



(a) Scores for all features

```
[ ] #updating the features list with our target variable.
    best_features=list(Features_Kbest.columns.values)
    best_features.extend(['Class'])
    best_features
```

```
['V17',
 'V14',
 'V12',
 'V10',
 'V16',
 'V3',
 'V7',
 'V11',
 'V4',
 'V18',
 'V1',
 'V9',
 'V5',
 'V2',
 'Class']
```

(b) List of selected features

Figure 18: feature selection: scores and features

```
#keeping the above selected features and our target variable for further analysis. Droping other features from dataframe.
DF_RAW.drop(DF_RAW.columns.difference(best_features), 1, inplace=True)
DF_CLEANED=DF_RAW
DF_CLEANED
```

	V1	V2	V3	V4	V5	V7	V9	V10	V11	V12	V14	V16	V17	V18	Class
0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.239599	0.363787	0.090794	-0.551600	-0.617801	-0.311169	-0.470401	0.207971	0.025791	0
1	1.191857	0.266151	0.166480	0.448154	0.060018	-0.078803	-0.255425	-0.166974	1.612727	1.065235	-0.143772	0.463917	-0.114805	-0.183361	0
2	-1.358354	-1.340163	1.773209	0.379780	-0.503198	0.791461	-1.514654	0.207643	0.624501	0.066084	-0.165946	-2.890083	1.109969	-0.121359	0
3	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	0.237609	-1.387024	-0.054952	-0.226487	0.178228	-0.287924	-1.059647	-0.684093	1.965775	0
4	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.592941	0.817739	0.753074	-0.822843	0.538196	-1.119670	-0.451449	-0.237033	-0.038195	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
284802	-11.881118	10.071785	-9.834783	-2.066656	-5.364473	-4.918215	1.914428	4.356170	-1.593105	2.711941	4.626942	1.107641	1.991691	0.510632	0
284803	-0.732789	-0.055080	2.035030	-0.738589	0.868229	0.024330	0.584800	-0.975926	-0.150189	0.915802	-0.675143	-0.711757	-0.025693	-1.221179	0
284804	1.919565	-0.301254	-3.249640	-0.557828	2.630515	-0.296827	0.432454	-0.484782	0.411614	0.063119	-0.510602	0.140716	0.313502	0.395652	0
284805	-0.240440	0.530483	0.702510	0.689799	-0.377961	-0.686180	0.392087	-0.399126	-1.933849	-0.962886	0.449624	-0.608577	0.509928	1.113981	0
284806	-0.533413	-0.189733	0.703337	-0.506271	-0.012546	1.577006	0.486180	-0.915427	-1.040458	-0.031513	-0.084316	-0.302620	-0.660377	0.167430	0

283726 rows × 15 columns

Figure 19: Reduced dataframe after removal of redundant features

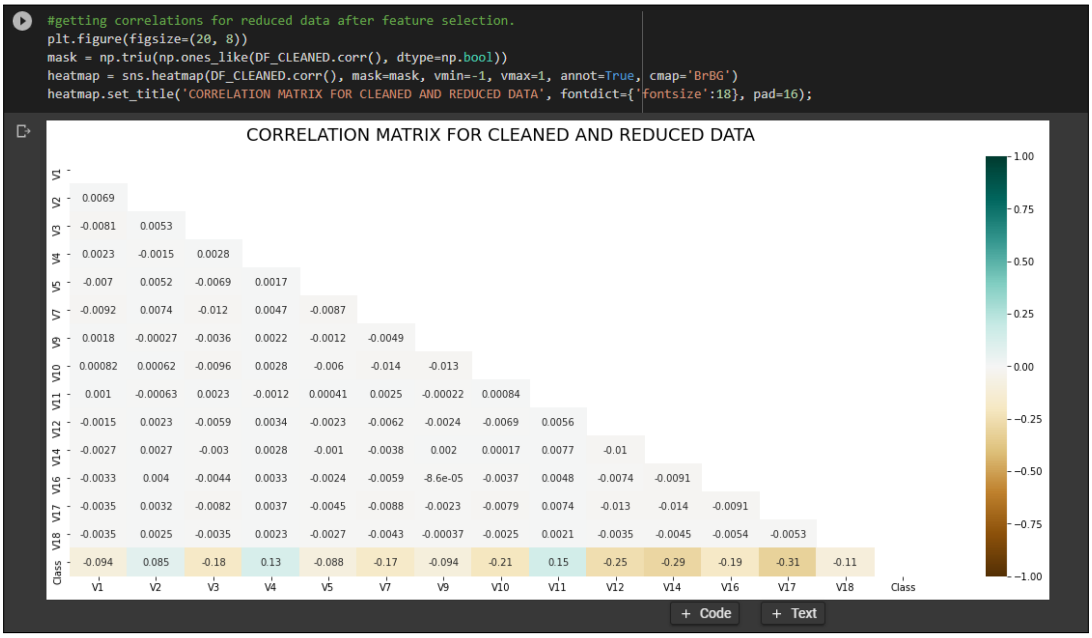


Figure 20: Correlation between reduced dataframe

6.6 Data Split: Train, validation and Test

Once our data is ready for application of machine learning models, we are devising our data into train, validation and test subsets for further usage. Figure:- 21

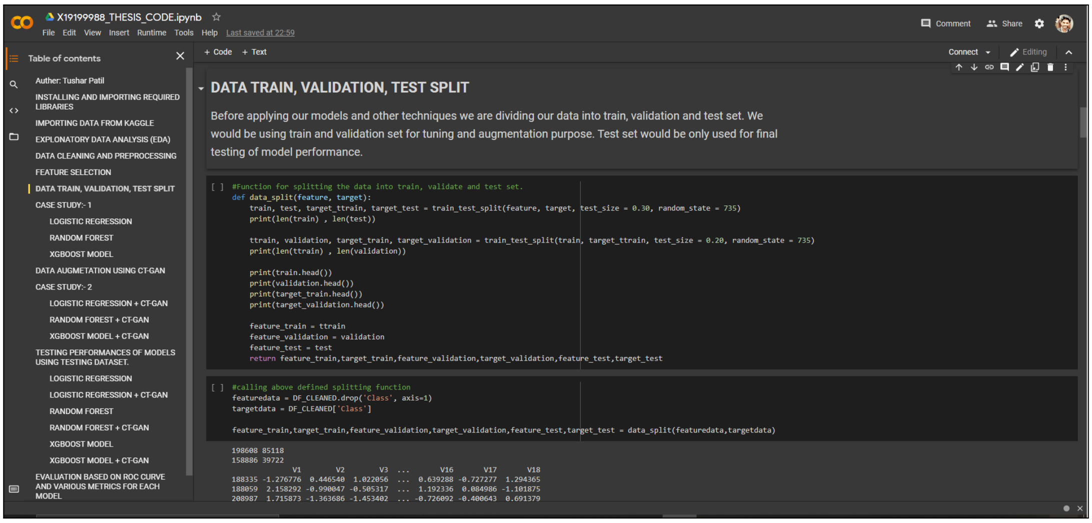


Figure 21: Splitting data into train, validation and test set

6.7 Case Study 1: Training Models on Unbalanced Data

Modelling is the most crucial part of our research work. For testing our proposed methodology, we are training our selected models on unbalanced data and verifying their performance using a validation dataset.

6.7.1 Training Logistic Regression

In this section, we have trained our Logistic regression model on unbalanced data.

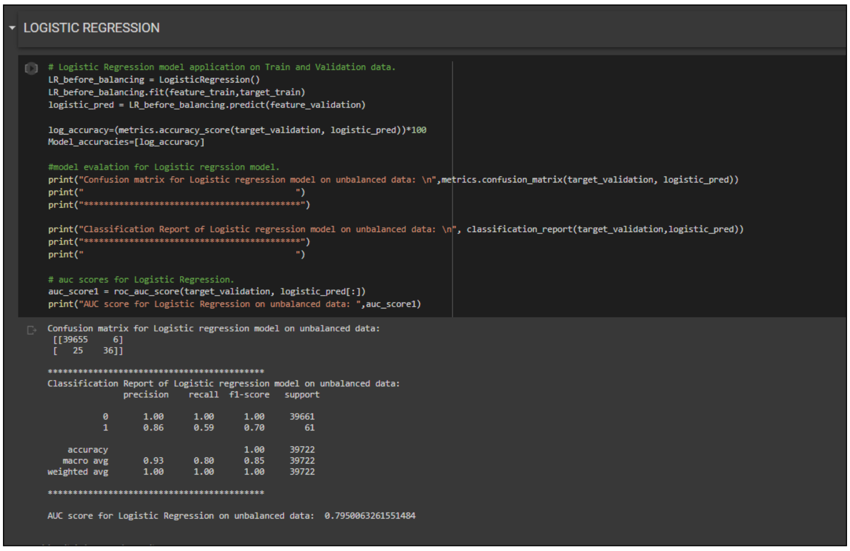


Figure 22: Training of logistic regression model on unbalanced data

6.7.2 Training Random Forest

In this section, we have trained our Random forest model on unbalanced data.

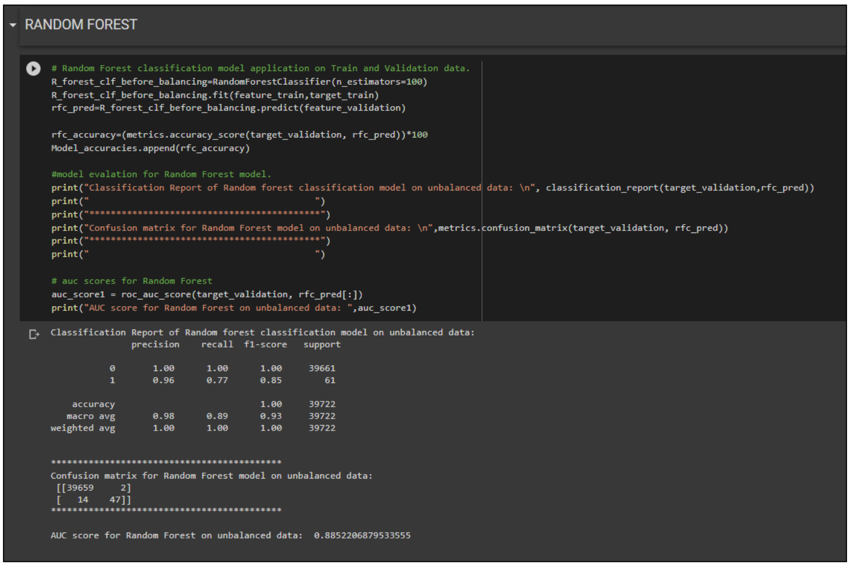
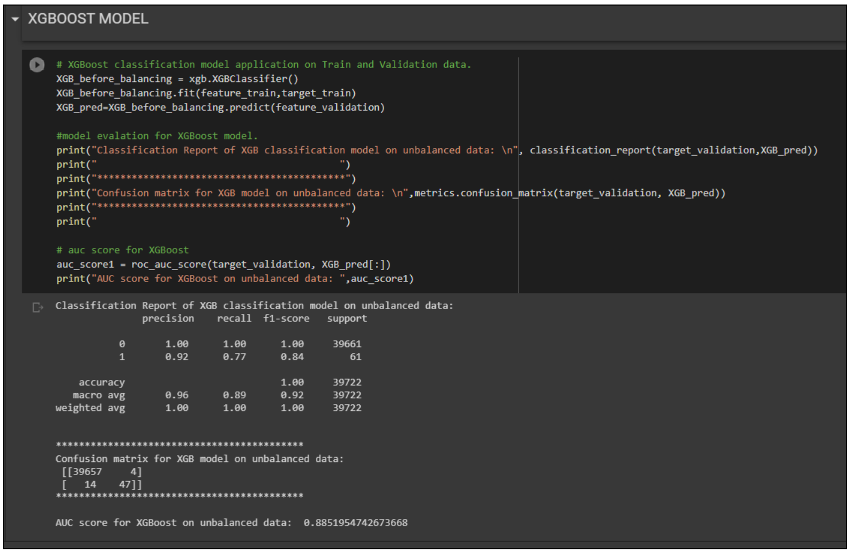


Figure 23: Training of Random forest model on unbalanced data

6.7.3 Training XGBoost Model

In this section, we have trained our XGBoost model on unbalanced data.



XGBOOST MODEL

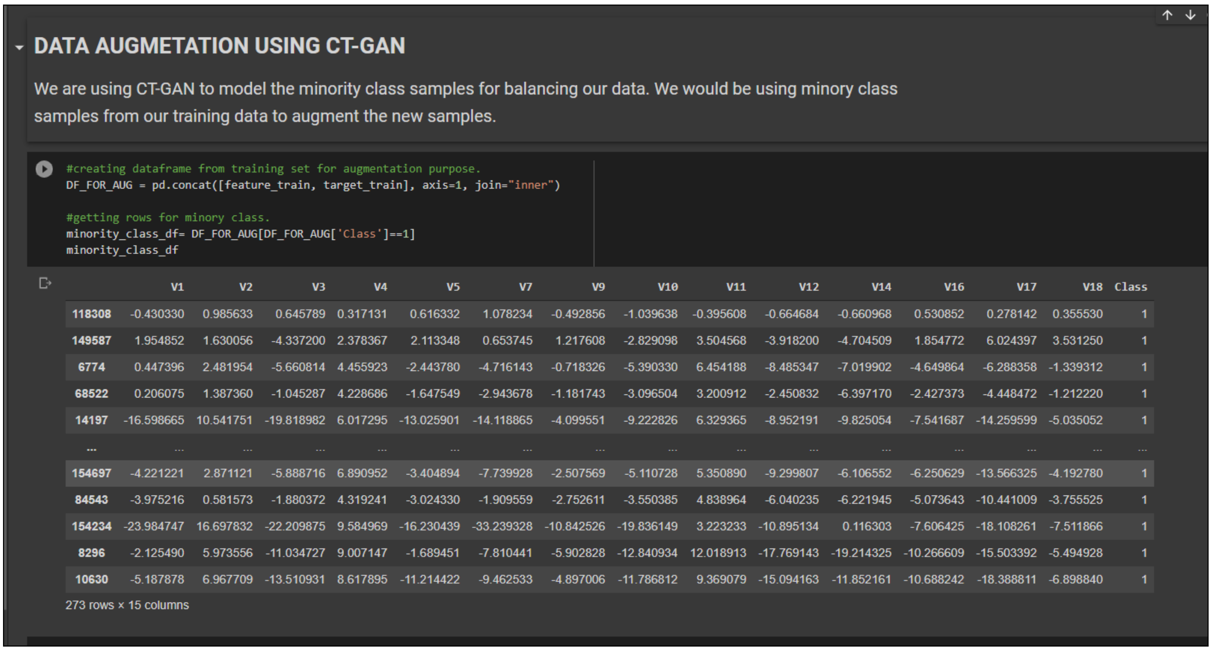
```
# XGBoost classification model application on Train and Validation data.
XGB_before_balancing = xgb.XGBClassifier()
XGB_before_balancing.fit(feature_train,target_train)
XGB_pred=XGB_before_balancing.predict(feature_validation)

#model evalation for XGBoost model.
print("Classification Report of XGB classification model on unbalanced data: \n", classification_report(target_validation,XGB_pred))
print("                                        ")
print("******************************************")
print("Confusion matrix for XGB model on unbalanced data: \n",metrics.confusion_matrix(target_validation, XGB_pred))
print("******************************************")
print("                                        ")

# auc score for XGBoost
auc_score1 = roc_auc_score(target_validation, XGB_pred[:])
print("AUC score for XGBoost on unbalanced data: ",auc_score1)
```

```
Classification Report of XGB classification model on unbalanced data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     39661
           1       0.92      0.77      0.84        61

    accuracy                           1.00     39722
   macro avg       0.96      0.89      0.92     39722
weighted avg       1.00      1.00      1.00     39722


******************************************
Confusion matrix for XGB model on unbalanced data:
 [[39657     4]
 [   14    47]]
******************************************

AUC score for XGBoost on unbalanced data:  0.8851954742673668
```

Figure 24: Training of XGBoost model on unbalanced data

6.8 Data Augmentation and Balancing Using CT-GAN

Data augmentation using CT-GAN is the most crucial part of our research project. We are using CTGAN package to generate synthetic samples of data and use for to balance our training data set. The code used for implementation of CT-GAN is explained in this section. Figure:- **??**,**??**,**??**



DATA AUGMETATION USING CT-GAN

We are using CT-GAN to model the minority class samples for balancing our data. We would be using minory class samples from our training data to augment the new samples.

```
#creating dataframe from training set for augmentation purpose.
DF_FOR_AUG = pd.concat([feature_train, target_train], axis=1, join="inner")

#getting rows for minory class.
minority_class_df= DF_FOR_AUG[DF_FOR_AUG['Class']==1]
minority_class_df
```

	V1	V2	V3	V4	V5	V7	V9	V10	V11	V12	V14	V16	V17	V18	Class
118308	-0.430330	0.985633	0.645789	0.317131	0.616332	1.078234	-0.492856	-1.039638	-0.395608	-0.664684	-0.660968	0.530852	0.278142	0.355530	1
149587	1.954852	1.630056	-4.337200	2.378367	2.113348	0.653745	1.217608	-2.829098	3.504568	-3.918200	-4.704509	1.854772	6.024397	3.531250	1
6774	0.447396	2.481954	-5.660814	4.455923	-2.443780	-4.716143	-0.718326	-5.390330	6.454188	-8.485347	-7.019902	-4.649864	-6.288358	-1.339312	1
68522	0.206075	1.387360	-1.045287	4.228686	-1.647549	-2.943678	-1.181743	-3.096504	3.200912	-2.450832	-6.397170	-2.427373	-4.448472	-1.212220	1
14197	-16.598665	10.541751	-19.818982	6.017295	-13.025901	-14.118865	-4.099551	-9.222826	6.329365	-8.952191	-9.825054	-7.541687	-14.259599	-5.035052	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
154697	-4.221221	2.871121	-5.888716	6.890952	-3.404894	-7.739928	-2.507569	-5.110728	5.350890	-9.299807	-6.106552	-6.250629	-13.566325	-4.192780	1
84543	-3.975216	0.581573	-1.880372	4.319241	-3.024330	-1.909559	-2.752611	-3.550385	4.838964	-6.040235	-6.221945	-5.073643	-10.441009	-3.755525	1
154234	-23.984747	16.697832	-22.209875	9.584969	-16.230439	-33.239328	-10.842526	-19.836149	3.223233	-10.895134	0.116303	-7.606425	-18.108261	-7.511866	1
8296	-2.125490	5.973556	-11.034727	9.007147	-1.689451	-7.810441	-5.902828	-12.840934	12.018913	-17.769143	-19.214325	-10.266609	-15.503392	-5.494928	1
10630	-5.187878	6.967709	-13.510931	8.617895	-11.214422	-9.462533	-4.897006	-11.786812	9.369079	-15.094163	-11.852161	-10.688242	-18.388811	-6.898840	1

273 rows × 15 columns

Figure 25: Isolation of minority class samples from data for data augmentation

```
[ ] #inializing the CTGANSynthesizer function of CT-GAN package and provide the required input and tuning parameters for training of CT-GAN.
    ctgan = CTGANSynthesizer()
    ctgan.fit(minority_class_df, column_names, epochs=150)
```

```
[ ] # generating new samples using our trained CT-GAN model.
    minority_class_df_new=pd.DataFrame()
    minority_class_augmented_df=minority_class_df
    i = 1
    while i < 10:
      augmented_df = ctgan.sample(30000)
      concat_df=[minority_class_augmented_df,augmented_df]
      minority_class_augmented_df=pd.concat(concat_df)
      i += 1
```

```
    # Merging newly generated samples into original training dataset for balancing the same.
    concat_df=[DF_RAW,minority_class_augmented_df]
    balanced_df_GAN=pd.concat(concat_df)
    balanced_df_GAN
```

Figure 26: Using CT-GAN for generation of synthetic minority class samples and merging them to original data

```
[ ] #getting count for each target class in new balanced dataset.
    Count_df = balanced_df_GAN.groupby('Class')['Class'].count()
    print("Count for each class in Data:-",Count_df)
    print("                                                                                          ")
    print("******************************************************************************************")
    print("                                                                                          ")

    # getting percentage of each target class in balanced data.
    print('No Frauds:0 ---', round(balanced_df_GAN['Class'].value_counts()[0]/len(balanced_df_GAN) * 100,2), '% of the dataset')
    print('Frauds:1 ----', round(balanced_df_GAN['Class'].value_counts()[1]/len(balanced_df_GAN) * 100,2), '% of the dataset')
    print("                                                                                          ")
    print("******************************************************************************************")
    print("                                                                                          ")

    #plotting the above obtained values in bar graph.
    count_classes = pd.value_counts(balanced_df_GAN['Class'], sort = True).sort_index()
    count_classes.plot(kind = 'bar')
    plt.title("Target class count \n After CT-GAN \n (1: Fraud || 0: No Fraud)",fontsize=20)
    plt.xlabel("Class")
    plt.ylabel("Count")

    Count for each class in Data:- Class
    0    283253
    1    270746
    Name: Class, dtype: int64

    ******************************************************************************************

    No Frauds:0 --- 51.13 % of the dataset
    Frauds:1 ---- 48.87 % of the dataset

    ******************************************************************************************
```

Figure 27: Balanced training data after merging augmented data

6.9 Case Study 2: Training Models on balanced Data

To verify the efficiency of our proposed approach to sole class imbalance using CT-GAN we would be training another set of classifiers on balanced test data and validating their performances on a validation dataset. The code used for training our models on balanced data is as follows. Figure:- 28,29,30,31

Creating feature and target sets for training our models.

CASE STUDY:- 2

APPLYING MACHINE LERANING MODELS:- TRAINED ON BALANCED DATA

We are training our classifiers on newly balanced dataframe and validating their performance on validation set. Validation set is used to fine tune the performance of models.

```
[ ] # seperating our predictors and target variable for model application.
    X_train = balanced_df_GAN.drop(['Class'], inplace=False, axis=1)
    y_train = balanced_df_GAN['Class']
```

Figure 28: Creating feature and target set for training

6.9.1 Training Logistic Regression + CT-GAN

In this section, we have trained our Logistic regression model on balanced data.

LOGISTIC REGRESSION + CT-GAN

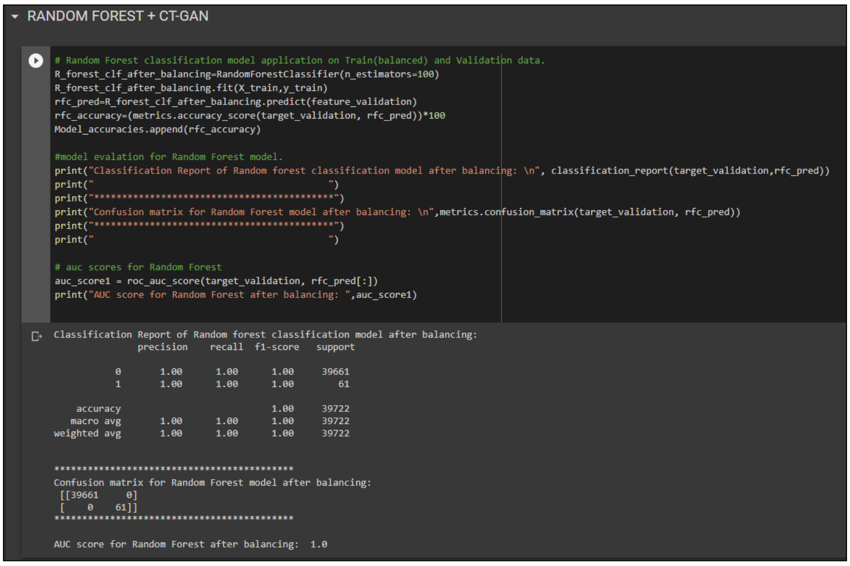
```
# Logistic Regression model application on Train(balanced) and Validation data.
LR_after_balancing = LogisticRegression()
LR_after_balancing.fit(X_train,y_train)
logistic_pred = LR_after_balancing.predict(feature_validation)

log_accuracy=(metrics.accuracy_score(target_validation, logistic_pred))*100
Model_accuracies=[log_accuracy]

#model evalation for Logistic regrssion model.
print("Confusion matrix for Logistic regression model after balancing: \n",metrics.confusion_matrix(target_validation, logistic_pred))
print("                                          ")
print("*******************************************")
print("Classification Report of Logistic regression model after balancing: \n", classification_report(target_validation,logistic_pred))
print("*******************************************")
print("                                          ")
# auc scores for Logistic Regression.
auc_score1 = roc_auc_score(target_validation, logistic_pred[:])
print("AUC score for Logistic Regression after balancing: ",auc_score1)
```

```
Confusion matrix for Logistic regression model after balancing:
 [[39646    15]
 [   11    50]]

*******************************************
Classification Report of Logistic regression model after balancing:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     39661
           1       0.77      0.82      0.79        61

    accuracy                           1.00     39722
   macro avg       0.88      0.91      0.90     39722
weighted avg       1.00      1.00      1.00     39722

*******************************************

AUC score for Logistic Regression after balancing:  0.9096469629288547
```

Figure 29: Training of logistic regression model on balanced data

6.9.2 Training Random Forest + CT-GAN

In this section, we have trained our Random Forest model on balanced data.



RANDOM FOREST + CT-GAN

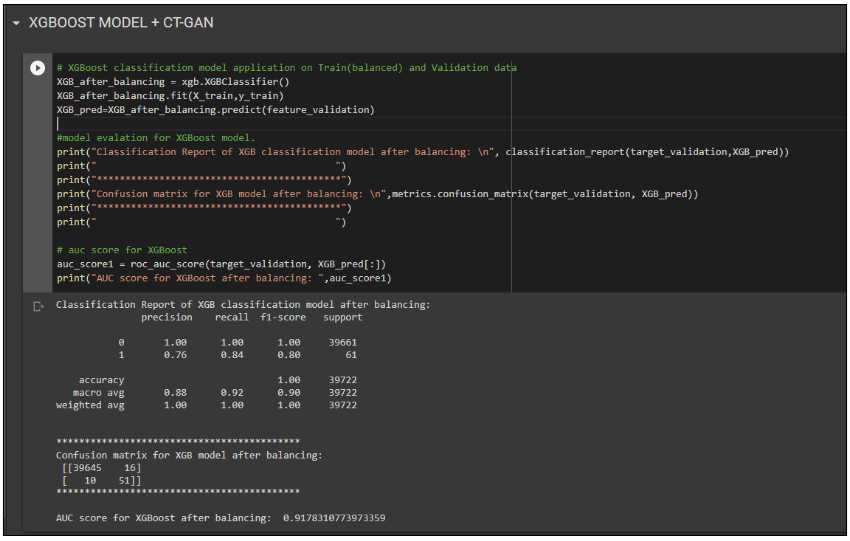
```python
# Random Forest classification model application on Train(balanced) and Validation data.
R_forest_clf_after_balancing=RandomForestClassifier(n_estimators=100)
R_forest_clf_after_balancing.fit(X_train,y_train)
rfc_pred=R_forest_clf_after_balancing.predict(feature_validation)
rfc_accuracy=(metrics.accuracy_score(target_validation, rfc_pred))*100
Model_accuracies.append(rfc_accuracy)

#model evalation for Random Forest model.
print("Classification Report of Random forest classification model after balancing: \n", classification_report(target_validation,rfc_pred))
print("                                        ")
print("******************************************")
print("Confusion matrix for Random Forest model after balancing: \n",metrics.confusion_matrix(target_validation, rfc_pred))
print("******************************************")
print("                                        ")

# auc scores for Random Forest
auc_score1 = roc_auc_score(target_validation, rfc_pred[:])
print("AUC score for Random Forest after balancing: ",auc_score1)
```

```
Classification Report of Random forest classification model after balancing:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39661
           1       1.00      1.00      1.00        61

    accuracy                           1.00     39722
   macro avg       1.00      1.00      1.00     39722
weighted avg       1.00      1.00      1.00     39722


******************************************
Confusion matrix for Random Forest model after balancing:
 [[39661     0]
 [    0    61]]
******************************************

AUC score for Random Forest after balancing:  1.0
```

Figure 30: Training of Random Forest model on balanced data

6.9.3 Training XGBoost Model + CT-GAN

In this section, we have trained our XGBoost Model on balanced data.



XGBOOST MODEL + CT-GAN

```python
# XGBoost classification model application on Train(balanced) and Validation data
XGB_after_balancing = xgb.XGBClassifier()
XGB_after_balancing.fit(X_train,y_train)
XGB_pred=XGB_after_balancing.predict(feature_validation)

#model evalation for XGBoost model.
print("Classification Report of XGB classification model after balancing: \n", classification_report(target_validation,XGB_pred))
print("                                        ")
print("******************************************")
print("Confusion matrix for XGB model after balancing: \n",metrics.confusion_matrix(target_validation, XGB_pred))
print("******************************************")
print("                                        ")

# auc score for XGBoost
auc_score1 = roc_auc_score(target_validation, XGB_pred[:])
print("AUC score for XGBoost after balancing: ",auc_score1)
```

```
Classification Report of XGB classification model after balancing:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     39661
           1       0.76      0.84      0.80        61

    accuracy                           1.00     39722
   macro avg       0.88      0.92      0.90     39722
weighted avg       1.00      1.00      1.00     39722


******************************************
Confusion matrix for XGB model after balancing:
 [[39645    16]
 [   10    51]]
******************************************

AUC score for XGBoost after balancing:  0.9178310773973359
```

Figure 31: Training of XGBoost Model on balanced data

6.10 Testing Performance of All Models on Test Dataset

Once we have obtained both set of models: Trained on unbalanced data, trained on balanced data, we are testing their performance on test set of data. We have collected the performance parameters into a single dataframe for further visualization and summarized results.

TESTING PERFORMANCES OF MODELS USING TESTING DATASET.

```
#DEFINING DATAFRAME FOR COLLECTING THE RESULTS OF EACH MODEL FOR EAVALUATION AND COMPARISON PURPOSE.
MODEL_PERFOROMANCE_DF = pd.DataFrame(columns=['MODEL_CASE','MODEL_ABB','PRECISION','RECALL','F1_SCORE','AUC_SCORE','GM','FP_RATE','TP_RATE'])
```

```
def get_model_performance_parameters(targe_set, pred_variable,model_name,model_abb):

  #DEFINING MODEL_PERFROMANCE_DF AS GLOBAL VARIABLE.
  global MODEL_PERFOROMANCE_DF
  #GETTING MODEL SCORES
  FP_RATE,TP_RATE,_ = roc_curve(targe_set,  pred_variable)
  AUC_SCORE= roc_auc_score(targe_set, pred_variable[:])
  F1_SCORE= f1_score(targe_set, pred_variable)
  PRECISION= precision_score(targe_set, pred_variable)
  RECALL=recall_score(targe_set, pred_variable)
  GM=math.sqrt(PRECISION*RECALL)

  #rounding values into percentage.
  PRECISION=PRECISION*100
  RECALL=RECALL*100
  F1_SCORE=F1_SCORE*100
  AUC_SCORE=AUC_SCORE*100
  GM=GM*100
  MODEL_CASE= model_name
  MODEL_ABB=model_abb


  #COLLECTING MODEL PERFORMANCE VALUES.
  MODEL_PERFOROMANCE_DF=MODEL_PERFOROMANCE_DF.append({'MODEL_CASE':MODEL_CASE,'MODEL_ABB':model_abb,'PRECISION':PRECISION,'RECALL':RECALL,'GM':GM,'F1_SCORE':F1_SCORE,
                                          'FP_RATE':FP_RATE, 'TP_RATE':TP_RATE, 'AUC_SCORE':AUC_SCORE}, ignore_index=True)

  print(model_name +" parameters recorded")
```

Figure 32: Defining function and dataframe to collect the performance parameters for all models

6.10.1 Testing Logistic Regression

In this section, we are testing our Logistic regression model using a test dataset.

LOGISTIC REGRESSION

```
print("********Logistic Regression performance on testing data after trained on Unbalanced data.******")
print("\n")
unbalanced_logistic_pred_test = LR_before_balancing.predict(feature_test)

#model evalation for Logistic regrssion model trained on unbalanced data tested on testing data.
print("Confusion matrix for Logistic regression model trained on unbalanced data: \n",metrics.confusion_matrix(target_test, unbalanced_logistic_pred_test))
print("                                            ")
print("********************************************")

print("Classification Report of Logistic regression model trained on unbalanced data: \n", classification_report(target_test,unbalanced_logistic_pred_test))
print("********************************************")
print("                                            ")

#RECORDING EVALUATION METRICS FOR LR
get_model_performance_parameters(target_test,unbalanced_logistic_pred_test,'LOGISTIC_REGRESSION','LR')
```

```
********Logistic Regression performance on testing data after trained on Unbalanced data.******


Confusion matrix for Logistic regression model trained on unbalanced data:
 [[84967    12]
 [   60    79]]

********************************************
Classification Report of Logistic regression model trained on unbalanced data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       0.87      0.57      0.69       139

    accuracy                           1.00     85118
   macro avg       0.93      0.78      0.84     85118
weighted avg       1.00      1.00      1.00     85118

********************************************

LOGISTIC_REGRESSION parameters recorded
```

Figure 33: Testing logistic regression model on test data.

6.10.2 Testing Logistic Regression + CT-GAN

In this section, we are testing our Logistic regression + CT-GAN model using a test dataset.

LOGISTIC REGRESSION + CT-GAN

```python
print("**********Logistic Regression performance on testing data after trained on balanced data.************")
print("\n")
balanced_logistic_pred_test = LR_after_balancing.predict(feature_test)

#model evalation for Logistic regrssion model trained on unbalanced data tested on testing data.
print("Confusion matrix for Logistic regression model trained on balanced data: \n",metrics.confusion_matrix(target_test, balanced_logistic_pred_test))
print("                                          ")
print("*******************************************")

print("Classification Report of Logistic regression model trained on balanced data: \n", classification_report(target_test,balanced_logistic_pred_test))
print("*******************************************")
print("                                          ")

#RECORDING EVALUATION METRICS FOR LR +CT-GAN
get_model_performance_parameters(target_test,balanced_logistic_pred_test,'LOGISTIC_REGRESSION + CT-GAN','LR+CT-GAN')
```

```
**********Logistic Regression performance on testing data after trained on balanced data.************


Confusion matrix for Logistic regression model trained on balanced data:
 [[84952    27]
 [   26   113]]

*******************************************
Classification Report of Logistic regression model trained on balanced data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       0.81      0.81      0.81       139

    accuracy                           1.00     85118
   macro avg       0.90      0.91      0.90     85118
weighted avg       1.00      1.00      1.00     85118

*******************************************

LOGISTIC_REGRESSION + CT-GAN parameters recorded
```

Figure 34: Testing logistic regression + CT-GAN model on test data.

6.10.3 Testing Random Forest

In this section, we are testing our Random Forest model using a test dataset.

RANDOM FOREST

```python
print("********Random Forest model performance on testing data after trained on Unbalanced data.******")
print("\n")
unbalanced_rfc_pred=R_forest_clf_before_balancing.predict(feature_test)

#model evalation for Random Forest model.

print("Confusion matrix for Random Forest model trained on Unbalanced data: \n",metrics.confusion_matrix(target_test, unbalanced_rfc_pred))
print("*******************************************")
print("                                          ")
print("Classification Report of Random forest classification model trained on Unbalanced data: \n", classification_report(target_test,unbalanced_rfc_pred))
print("                                          ")
print("*******************************************")

#RECORDING EVALUATION METRICS FOR RF
get_model_performance_parameters(target_test,unbalanced_rfc_pred,'RANDOM_FOREST','RF')
```

```
********Random Forest model performance on testing data after trained on Unbalanced data.******


Confusion matrix for Random Forest model trained on Unbalanced data:
 [[84977     2]
 [   36   103]]
*******************************************

Classification Report of Random forest classification model trained on Unbalanced data:
              precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       0.98      0.74      0.84       139

    accuracy                           1.00     85118
   macro avg       0.99      0.87      0.92     85118
weighted avg       1.00      1.00      1.00     85118


*******************************************
RANDOM_FOREST parameters recorded
```

Figure 35: Testing Random Forest model on test data.

6.10.4 Testing Random Forest + CT-GAN

In this section, we are testing our Random Forest + CT-GAN model using a test dataset.

```
RANDOM FOREST + CT-GAN

print("********Random Forest model performance on testing data after trained on balanced data.******")
print("\n")
balanced_rfc_pred=R_forest_clf_after_balancing.predict(feature_test)

#model evalation for Random Forest model.

print("Confusion matrix for Random Forest model trained on balanced data: \n",metrics.confusion_matrix(target_test, balanced_rfc_pred))
print("*******************************************")
print("                                          ")

print("Classification Report of Random forest classification model trained on balanced data: \n", classification_report(target_test,balanced_rfc_pred))
print("                                          ")
print("*******************************************")

#RECORDING EVALUATION METRICS FOR RF+CTGAN
get_model_performance_parameters(target_test,balanced_rfc_pred,'RANDOM_FOREST + CT-GAN','RF+CT-GAN')
```

```
********Random Forest model performance on testing data after trained on balanced data.******


Confusion matrix for Random Forest model trained on balanced data:
 [[84979     0]
 [    0   139]]
*******************************************

Classification Report of Random forest classification model trained on balanced data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       1.00      1.00      1.00       139

    accuracy                           1.00     85118
   macro avg       1.00      1.00      1.00     85118
weighted avg       1.00      1.00      1.00     85118


*******************************************
RANDOM_FOREST + CT-GAN parameters recorded
```

Figure 36: Testing Random Forest + CT-GAN model on test data.

6.10.5 Testing XGBoost Model

In this section, we are testing our XGBoost model using a test dataset.

```
XGBOOST MODEL

print("**********XGBoost model performance on testing data after trained on unbalanced data.************")
print("\n")

UNBALANCED_XGB_pred=XGB_before_balancing.predict(feature_test)

#model evalation for XGBoost model trained on unbalanced data, tested on testing data
print("Confusion matrix for XGBoost model trained on unbalanced data: \n",metrics.confusion_matrix(target_test, UNBALANCED_XGB_pred))
print("*******************************************")
print("                                          ")
print("Classification Report of XGBoost classification model trained on unbalanced data: \n", classification_report(target_test,UNBALANCED_XGB_pred))
print("                                          ")
print("*******************************************")

#RECORDING EVALUATION METRICS FOR XGBOOST
get_model_performance_parameters(target_test,UNBALANCED_XGB_pred,'XGBOOST','XGB')
```

```
**********XGBoost model performance on testing data after trained on unbalanced data.************


Confusion matrix for XGBoost model trained on unbalanced data:
 [[84972     7]
 [   39   100]]
*******************************************

Classification Report of XGBoost classification model trained on unbalanced data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       0.93      0.72      0.81       139

    accuracy                           1.00     85118
   macro avg       0.97      0.86      0.91     85118
weighted avg       1.00      1.00      1.00     85118


*******************************************
XGBOOST parameters recorded
```

Figure 37: Testing XGBoost model on test data.

6.10.6 Testing XGBoost Model + CT-GAN

In this section, we are testing our XGBoost + CT-GAN model using a test dataset.

XGBOOST MODEL + CT-GAN

```
print("**********XGBoost model performance on testing data after trained on balanced data.************")
print("\n")

BALANCED_XGB_pred=XGB_after_balancing.predict(feature_test)

#model evalation for XGBoost model trained on unbalanced data, tested on testing data
print("Confusion matrix for XGBoost model trained on balanced data: \n",metrics.confusion_matrix(target_test, BALANCED_XGB_pred))
print("*******************************************")
print("                                           ")
print("Classification Report of XGBoost classification model trained on balanced data: \n", classification_report(target_test,BALANCED_XGB_pred))
print("                                           ")
print("*******************************************")

#RECORDING EVALUATION METRICS FOR XGBOOST+CT-GAN
get_model_performance_parameters(target_test,BALANCED_XGB_pred,'XGBOOST + CT-GAN','XGB+CT-GAN')
```

```
**********XGBoost model performance on testing data after trained on balanced data.************


Confusion matrix for XGBoost model trained on balanced data:
 [[84956    23]
 [   22   117]]
*******************************************

Classification Report of XGBoost classification model trained on balanced data:
               precision    recall  f1-score   support

           0       1.00      1.00      1.00     84979
           1       0.84      0.84      0.84       139

    accuracy                           1.00     85118
   macro avg       0.92      0.92      0.92     85118
weighted avg       1.00      1.00      1.00     85118


*******************************************
XGBOOST + CT-GAN parameters recorded
```

Figure 38: Testing XGBoost + CT-GAN model on test data.

6.11 Evaluation Using Visualization

In this section, we have summarized the various evaluation metrics calculated for each mode after testing on the test dataset. Figure:- 39 shows the summarised metrics in a single dataframe.

EVALUATION BASED ON ROC CURVE AND VARIOUS METRICS FOR EACH MODEL

```
[ ] MODEL_PERFOROMANCE_DF
```

	MODEL_CASE	MODEL_ABB	PRECISION	RECALL	F1_SCORE	AUC_SCORE	GM	FP_RATE	TP_RATE
0	LOGISTIC_REGRESSION	LR	86.813187	56.834532	68.695652	78.410206	70.242344	[0.0, 0.00014121135810023653, 1.0]	[0.0, 0.5683453237410072, 1.0]
1	LOGISTIC_REGRESSION + CT-GAN	LR+CT-GAN	80.714286	81.294964	81.003584	90.631596	81.004105	[0.0, 0.0003177255557255322, 1.0]	[0.0, 0.8129496402877698, 1.0]
2	RANDOM_FOREST	RF	98.095238	74.100719	84.426230	87.049183	85.258007	[0.0, 2.353522635003942e-05, 1.0]	[0.0, 0.7410071942446043, 1.0]
3	RANDOM_FOREST + CT-GAN	RF+CT-GAN	100.000000	100.000000	100.000000	100.000000	100.000000	[0.0, 0.0, 1.0]	[0.0, 1.0, 1.0]
4	XGBOOST	XGB	93.457944	71.942446	81.300813	85.967104	81.997519	[0.0, 8.237329222513797e-05, 1.0]	[0.0, 0.7194244604316546, 1.0]
5	XGBOOST + CT-GAN	XGB+CT-GAN	83.571429	84.172662	83.870968	92.072798	83.871506	[0.0, 0.00027065510302545337, 1.0]	[0.0, 0.841726618705036, 1.0]

Figure 39: Summarized performance parameters

By using all the performance parameter collected into single dataframe we have used bar graph to carry out comparative analysis. The code for bar plot is as Figure:- 40

```python
# evaluation metrics plot for comparison.
ax = MODEL_PERFOROMANCE_DF.plot(x='MODEL_ABB',kind='bar',title='EVALUATION METRICS FOR CLASSIFIERS',stacked=False,figsize=(25,10),colormap='copper', )
ax.set_xlabel("CLASSIFIERS",fontsize = 20)
ax.set_ylabel("SCORE VALUES [0-100]",fontsize = 20)
ax.grid(color = 'black', linestyle = '--', linewidth = 0.5)
```

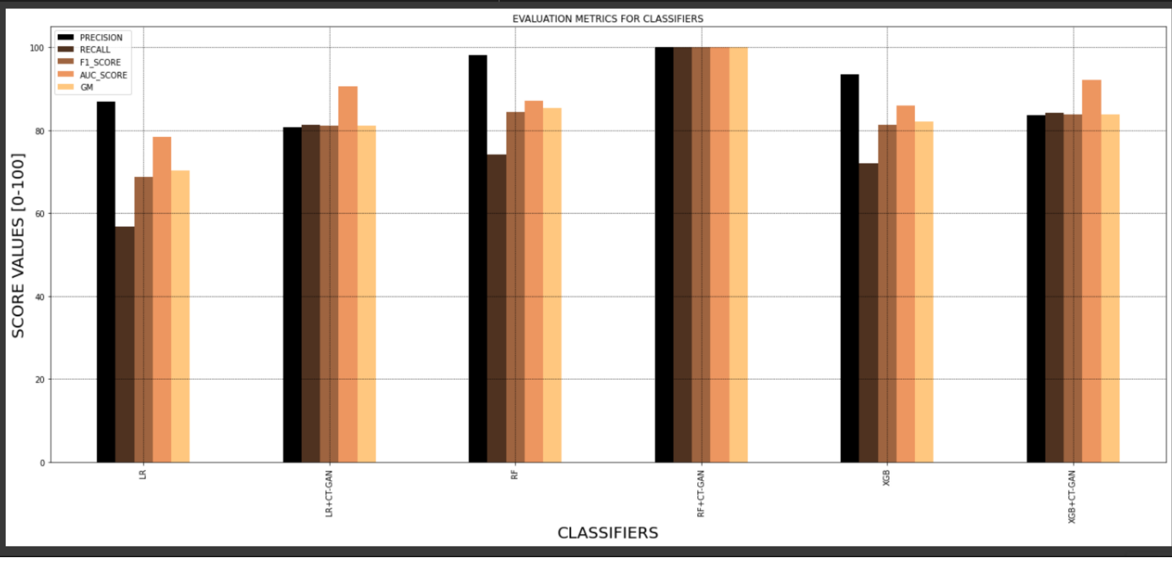


Figure 40: Code for plotting bar graph using values for all models.

By using all the TP and FP parameters collected into dataframe we have plotted a AUC-ROC curve for further comparative analysis. The code for AUC-ROC is as Figure:- 41, 42

```python
# Set name of the MODEL_CASE as index labels
MODEL_PERFOROMANCE_DF.set_index('MODEL_CASE', inplace=True)

fig = plt.figure(figsize=(15,8))

for i in MODEL_PERFOROMANCE_DF.index:
    plt.plot(MODEL_PERFOROMANCE_DF.loc[i]['FP_RATE'],
             MODEL_PERFOROMANCE_DF.loc[i]['TP_RATE'],
             label="{}, AUC={:.3f}".format(i, MODEL_PERFOROMANCE_DF.loc[i]['AUC_SCORE']))

plt.plot([0,1], [0,1], color='orange', linestyle='--')

plt.xticks(np.arange(0.0, 1.1, step=0.1))
plt.xlabel("Flase Positive Rate", fontsize=20)

plt.yticks(np.arange(0.0, 1.1, step=0.1))
plt.ylabel("True Positive Rate", fontsize=20)

plt.title('AUC-ROC Curve Analysis', fontweight='bold', fontsize=20)
plt.legend(prop={'size':15}, loc='lower right')

plt.show()
```

Figure 41: AUC-ROC curve graph for all models.

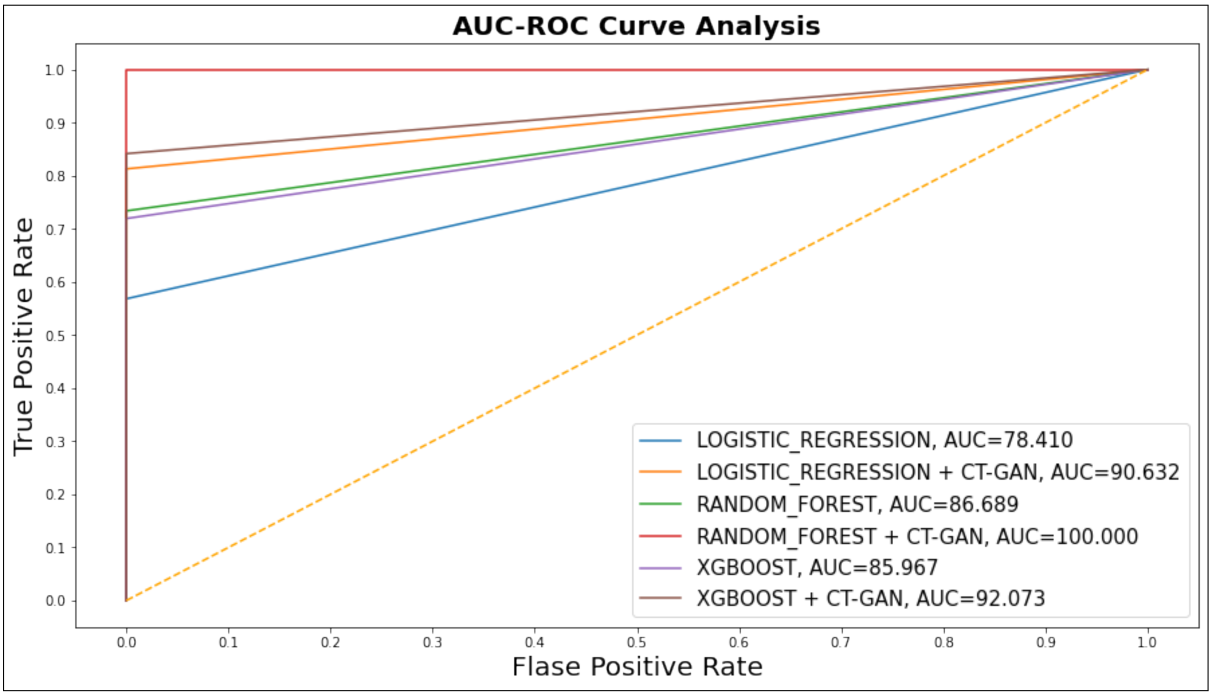


Figure 42: AUC-ROC curve

7 Conclusion

The whole implementation procedure of this project has been outlined in a succinct, thorough, and sequential way using the information presented in the preceding parts. The needed packages have been indicated wherever they were used. All the codes are commented and divided into sections for better readability.