Configuration Manual

MSc Research Project
Data Analytics

Pratiksha Arvind Chate
Student ID: x20150377

School of Computing
National College of Ireland

Supervisor: Prof. Christian Horn

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Pratiksha Arvind Chate
Student ID:	x20150377
Programme:	Data Analytics
Year:	2021
Module:	MSc Research Project
Supervisor:	Prof. Christian Horn
Submission Due Date:	31/01/2022
Project Title:	Configuration Manual
Word Count:	1534
Page Count:	15

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:	Pratiksha Arvind Chate
Date:	31st January 2022

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):	

Configuration Manual

Pratiksha Arvind Chate
x20150377

1 Introduction

This configuration manual provides a high-level overview of the hardware and software requirements for replicating the study. This handbook will be valuable in gaining a decent understanding of the prerequisites, starting with setting up the execution environment for implementing the research.

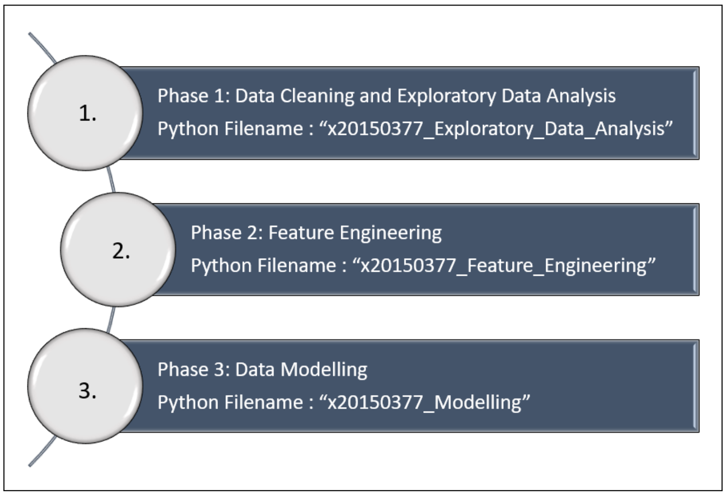


Figure 1: Phases of executing the Jupyter notebook files

The figure Figure 1 depicts the three phases to execute the files created. The sequence of the notebooks is Data Cleaning and Exploratory Data Analysis ("x20150377_Exploratory_Data_Analysis"), Feature Engineering ("x20150377_Feature_Engineering") and Data Modelling ("x20150377_Modelling").

2 System Configuration

2.1 Hardware Requirements

The hardware specifications on which this research is implemented is given as follows:

- **Windows Edition:** Windows 10 Home Single Language
- **Processor:** Intel(R) Core(TM) i7-10750H CPU @ 2.60GHz 2.59 GHz
- **Installed RAM:** 8.00 GB (7.80 GB usable)

- **System Type:** 64-bit operating system, x64-based processor
- **Pen and Touch** No pen or touch input is available for this display

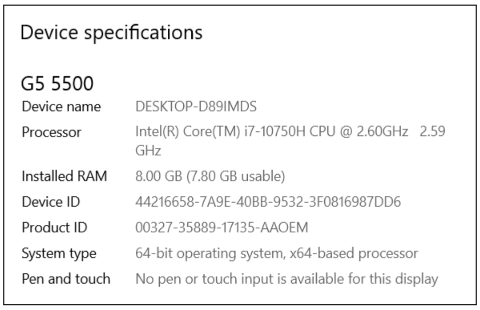


Figure 2: Device Specifications

2.2 Software Requirements

The software requirements to implement this research are stated below:

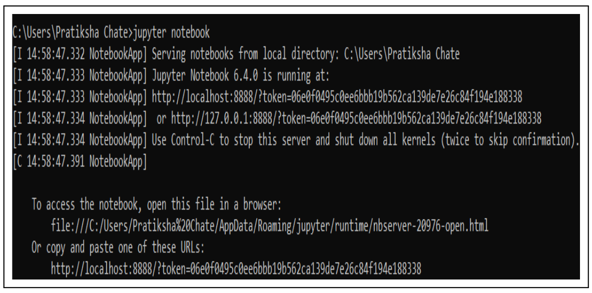
- **Programming Language:** Python (version - 3.9.5)
- **IDE:** Jupyter Notebook

3 Project Implementation

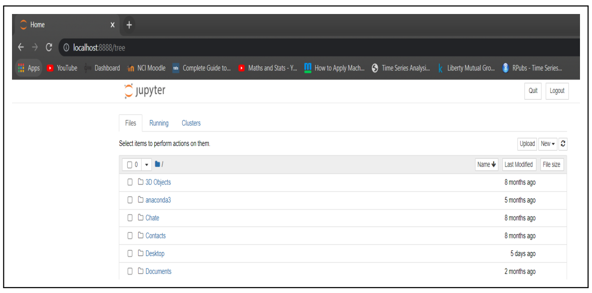
This section concentrates upon the steps involved to execute the research model implemented.

3.1 Programming Environment Setup

The Jupyter Notebook is launched from the command prompt in order to start the execution environment for its implementation.



(a) Launch Jupyter Notebook



(b) Jupyter Notebook home page

Figure 3: Execution environment

The Figure 3a shows the command to start the jupyter notebook through command prompt. As the jupyter notebook is launched, a new tab with home page is opened in the browser as shown in Figure 3b.

3.2 Data Collection

The dataset which the research is based upon is a fairly descriptive sales and order data of an e-commerce giant based out of Brazil. The dataset incorporates particulars of 100k customer orders placed in Brazil between the year 2016 and the year 2018. The attributes of this data facilitate observation of details from several viewpoints. The data is gathered from Kaggle[1] and is present in the form of CSV files that are further subdivided into numerous distinct datasets for easier interpretation and organization.

The new Python 3 file is created (with .ipynb extension) to extract the collected data and explore it to draw meaningful insights required for further processing.

3.3 Python Libraries

The libraries used such as pandas[2], Numpy[3], matplotlib[4], seaborn[5] are described in the table 1 below.

Library	Version
pandas	1.2.4
Numpy	1.19.5
matplotlib	3.4.2
seaborn	0.11.1
datetime	

Table 1: Python Libraries used for Data Analysis

These libraries can be installed using pip command in the Jupyter Notebook. Example: The Numpy library can be used by the command:
!pip install numpy.

Figure 4: Fetch the collected data to jupyter notebook

Once all the standard packages are in place, the collected data is fetched into the python notebook to perform data cleaning operations and exploratory data analyis as shown in the Figure 5.

[1]https://www.kaggle.com/olistbr/brazilian-ecommerce
[2]https://pandas.pydata.org/
[3]https://numpy.org/
[4]https://matplotlib.org/
[5]https://seaborn.pydata.org/

3.4 Data Merging (One-to-One Mapping)

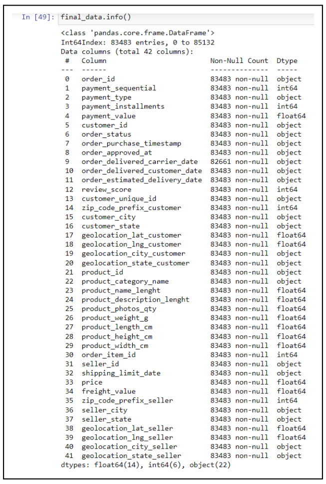
Few discrepancies were observed in the data causing cartesian product which were handled keeping in mind the below assumptions.
Assumptions:

- Order ID will be unique across all the transaction tables so that the resultant data is based on one-to-one mapping.
- One review ID can be tagged to just one order ID.
- For analysis, the data will be constrained to contain unique order IDs with one order item and one payment record.

```
In [1]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        %matplotlib inline
        from matplotlib.gridspec import GridSpec
        from datetime import datetime

In [2]: import os
        os.chdir(r"C:\Users\Pratiksha Chate\Desktop\Books\SEM III\Thesis\Dataset")

        Reading Data

In [3]: data = pd.read_csv("olist_customers_dataset.csv")
        geo_data = pd.read_csv("olist_geolocation_dataset.csv")
        order_itemdata = pd.read_csv("olist_order_items_dataset.csv")
        pay_data = pd.read_csv("olist_order_payments_dataset.csv")
        rev_data = pd.read_csv("olist_order_reviews_dataset.csv")
        orders = pd.read_csv("olist_orders_dataset.csv")
        order_prddata = pd.read_csv("olist_products_dataset.csv")
        order_selldata = pd.read_csv("olist_sellers_dataset.csv")
        order_prd_catdata = pd.read_csv("product_category_name_translation.csv")
```

Figure 5: Fetch the collected data to jupyter notebook

Data cleaning operations such as treating null values, deduplication, etc. were performed.



```
In [49]: final_data.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 83483 entries, 0 to 85132
Data columns (total 42 columns):
 #   Column                         Non-Null Count  Dtype
---  ------                         --------------  -----
 0   order_id                       83483 non-null  object
 1   payment_sequential             83483 non-null  int64
 2   payment_type                   83483 non-null  object
 3   payment_installments           83483 non-null  int64
 4   payment_value                  83483 non-null  float64
 5   customer_id                    83483 non-null  object
 6   order_status                   83483 non-null  object
 7   order_purchase_timestamp       83483 non-null  object
 8   order_approved_at              83483 non-null  object
 9   order_delivered_carrier_date   82661 non-null  object
 10  order_delivered_customer_date  83483 non-null  object
 11  order_estimated_delivery_date  83483 non-null  object
 12  review_score                   83483 non-null  int64
 13  customer_unique_id             83483 non-null  object
 14  zip_code_prefix_customer       83483 non-null  int64
 15  customer_city                  83483 non-null  object
 16  customer_state                 83483 non-null  object
 17  geolocation_lat_customer       83483 non-null  float64
 18  geolocation_lng_customer       83483 non-null  float64
 19  geolocation_city_customer      83483 non-null  object
 20  geolocation_state_customer     83483 non-null  object
 21  product_id                     83483 non-null  object
 22  product_category_name          83483 non-null  object
 23  product_name_lenght            83483 non-null  float64
 24  product_description_lenght     83483 non-null  float64
 25  product_photos_qty             83483 non-null  float64
 26  product_weight_g               83483 non-null  float64
 27  product_length_cm              83483 non-null  float64
 28  product_height_cm              83483 non-null  float64
 29  product_width_cm               83483 non-null  float64
 30  order_item_id                  83483 non-null  int64
 31  seller_id                      83483 non-null  object
 32  shipping_limit_date            83483 non-null  object
 33  price                          83483 non-null  float64
 34  freight_value                  83483 non-null  float64
 35  zip_code_prefix_seller         83483 non-null  int64
 36  seller_city                    83483 non-null  object
 37  seller_state                   83483 non-null  object
 38  geolocation_lat_seller         83483 non-null  float64
 39  geolocation_lng_seller         83483 non-null  float64
 40  geolocation_city_seller        83483 non-null  object
 41  geolocation_state_seller       83483 non-null  object
dtypes: float64(14), int64(6), object(22)
```

Figure 6: Final Data for EDA

The final set of data obtained contains 42 variables and no missing values as shown in Figure 6

The detailed exploratory data analysis on the cleaned data is performed with respect to the positive and negative reviews and some meaningful insights were discovered.

3.5 Customer Segmentation

The customer segmentation based on quantile method is implemented to segment the customers into their respective groups. The Figure 7, Figure 8, Figure 9 depict the customer segmentation implemented.

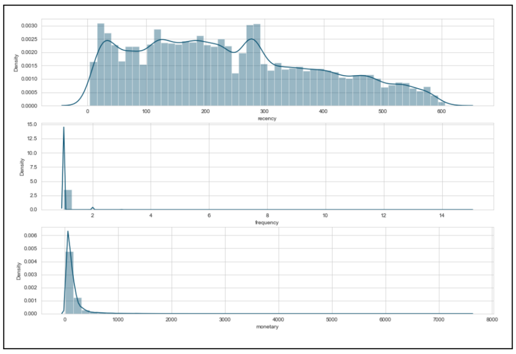


Figure 7: Distribution Plot for Recency, Frequency and Monetary value

```
In [146]: # Define rfm_level function
def rfm_level(df):
    if df['RFM_Score_s'] >= 9:
        return 'Can\'t Loose Them'
    elif ((df['RFM_Score_s'] >= 8) and (df['RFM_Score_s'] < 9)):
        return 'Champions'
    elif ((df['RFM_Score_s'] >= 7) and (df['RFM_Score_s'] < 8)):
        return 'Loyal'
    elif ((df['RFM_Score_s'] >= 6) and (df['RFM_Score_s'] < 7)):
        return 'Potential'
    elif ((df['RFM_Score_s'] >= 5) and (df['RFM_Score_s'] < 6)):
        return 'Promising'
    elif ((df['RFM_Score_s'] >= 4) and (df['RFM_Score_s'] < 5)):
        return 'Needs Attention'
    else:
        return 'Require Activation'
# Create a new variable RFM_Level
rfm['RFM_Level'] = rfm.apply(rfm_level, axis=1)
# Print the header with top 5 rows to the console
rfm.head()
```

Out[146]:

customer_unique_id	recency	frequency	monetary	f_quartile	r_quartile	m_quartile	RFM_Score	RFM_Score_s	RFM_Level
0000366f3b9a7992bf8c76cfdf3221e2	115	1	141.90	1	4	3	413	8	Champions
0000b849f77a49e4a4ce2b2a4ca5be3f	118	1	27.19	1	3	1	311	5	Promising
0000f46a3911fa3c0805444483337064	541	1	86.22	1	1	2	112	4	Needs Attention
0000f6ccb0745a6a4b88665a16c9f078	325	1	43.62	1	2	1	211	4	Needs Attention
0004aac84e0df4da2b147fca70cf8255	292	1	196.89	1	2	4	214	7	Loyal

Figure 8: Customer segmentation

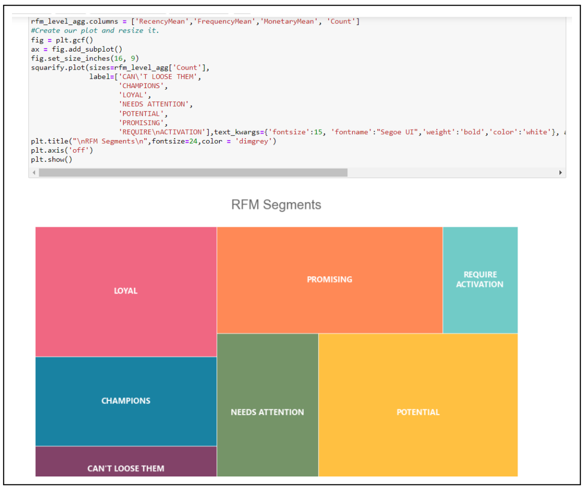
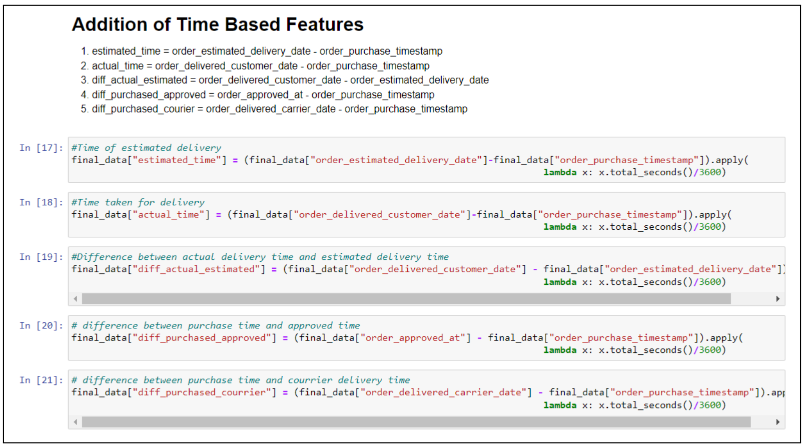


Figure 9: Customer Segmentation Diagram

3.6 Feature Engineering

This part of the research is considered to be significant as feature engineering is known to enhance the performance of the machine learning models applied. This is the second phase of this research. It is implemented by executing the "x20150377_Feature_Engineering" upon the successful execution of "x20150377_Exploratory_Data_Analysis". The set of libraries required to execute the feature engineering file are same as mentioned in the Table 1

To achieve the desired objective, new time-based and distance based features have been added and the existing attributes were not highly correlated with the target variable.



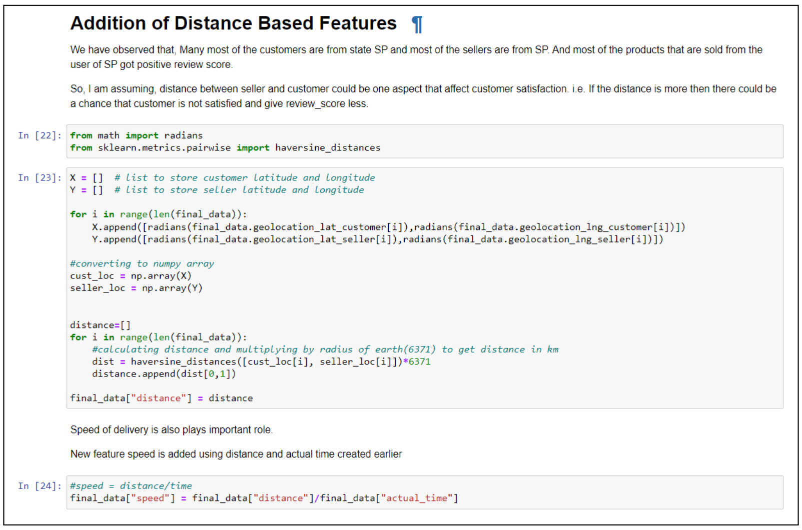
Addition of Time Based Features

1. estimated_time = order_estimated_delivery_date - order_purchase_timestamp
2. actual_time = order_delivered_customer_date - order_purchase_timestamp
3. diff_actual_estimated = order_delivered_customer_date - order_estimated_delivery_date
4. diff_purchased_approved = order_approved_at - order_purchase_timestamp
5. diff_purchased_courier = order_delivered_carrier_date - order_purchase_timestamp

```
In [17]: #Time of estimated delivery
final_data["estimated_time"] = (final_data["order_estimated_delivery_date"]-final_data["order_purchase_timestamp"]).apply(
                                                            lambda x: x.total_seconds()/3600)

In [18]: #Time taken for delivery
final_data["actual_time"] = (final_data["order_delivered_customer_date"]-final_data["order_purchase_timestamp"]).apply(
                                                            lambda x: x.total_seconds()/3600)

In [19]: #Difference between actual delivery time and estimated delivery time
final_data["diff_actual_estimated"] = (final_data["order_delivered_customer_date"] - final_data["order_estimated_delivery_date"])
                                                            lambda x: x.total_seconds()/3600)

In [20]: # difference between purchase time and approved time
final_data["diff_purchased_approved"] = (final_data["order_approved_at"] - final_data["order_purchase_timestamp"]).apply(
                                                            lambda x: x.total_seconds()/3600)

In [21]: # difference between purchase time and courrier delivery time
final_data["diff_purchased_courrier"] = (final_data["order_delivered_carrier_date"] - final_data["order_purchase_timestamp"]).app
                                                            lambda x: x.total_seconds()/3600)
```

Figure 10: Time-based features

Time-based features such as "estimated time", "actual delivery time", "difference between actual and estimated delivery time", "difference between purchased and order approved time", and "difference between purchased and shipped time" have been created as demonstrated in Figure 10 from the existing features to check if these are correlated with the target variable.



Addition of Distance Based Features ¶

We have observed that, Many most of the customers are from state SP and most of the sellers are from SP. And most of the products that are sold from the user of SP got positive review score.

So, I am assuming, distance between seller and customer could be one aspect that affect customer satisfaction. i.e. If the distance is more then there could be a chance that customer is not satisfied and give review_score less.

```
In [22]: from math import radians
from sklearn.metrics.pairwise import haversine_distances

In [23]: X = []  # list to store customer latitude and longitude
Y = []  # list to store seller latitude and longitude

for i in range(len(final_data)):
    X.append([radians(final_data.geolocation_lat_customer[i]),radians(final_data.geolocation_lng_customer[i])])
    Y.append([radians(final_data.geolocation_lat_seller[i]),radians(final_data.geolocation_lng_seller[i])])

#converting to numpy array
cust_loc = np.array(X)
seller_loc = np.array(Y)


distance=[]
for i in range(len(final_data)):
    #calculating distance and multiplying by radius of earth(6371) to get distance in km
    dist = haversine_distances([cust_loc[i], seller_loc[i]])*6371
    distance.append(dist[0,1])

final_data["distance"] = distance
```

Speed of delivery is also plays important role.

New feature speed is added using distance and actual time created earlier

```
In [24]: #speed = distance/time
final_data["speed"] = final_data["distance"]/final_data["actual_time"]
```

Figure 11: Distance-based features

Similarly, the distance-based features such as "distance" between customer and seller and "speed" of delivery were created as depicted in Figure 11.

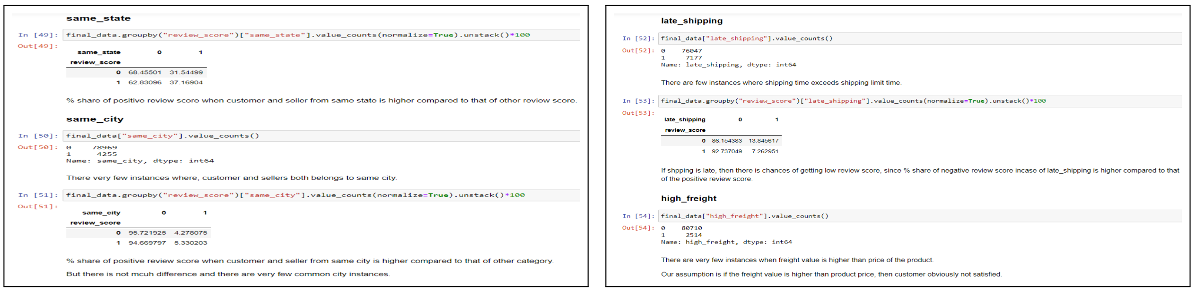
The newly created attributes were analysed with kdeplot and box plot to check if they are correlated with the target variable. The analysis of "difference between actual and estimated delivery time" is presented as an example in the figure below.

(a) kdeplot

(b) Box plot

Figure 12: Analysis of difference between actual and estimated delivery time

This newly derived attribute is found to be useful for predicting the review score as for lower values, probability density of positive review scores is highly peaked (refer Figure 12a). The same is conveyed through the box plot in Figure 12b. Thus, the likelihood of receiving a positive review score is high if the product is delivered before the estimated delivery date. Also, business is likely to receive a lower review score if the delivery exceeds the estimated delivery time.



(a) Same city and same state

(b) Late shipping and high freight

Figure 13: Binary features

Some new binary features such as "same city", "same state", "late shipping" and "high freight" were derived (refer Figure 13) to check if the customers are sellers are from the same city, same state or if the product is shipped late or high freight value is paid by the customers respectively.

3.7 Data Modelling

This is the final phase of this research where data oversampling, and modelling is performed and the implemented models are evaluated. It is implemented by executing the "x20150377_Modelling" upon the successful execution of "x20150377_Feature_Engineering".

The packages imported such as pickle[6], sklearn[7], imblearn[8], and lightgbm[9] for executing this file are presented in the Table 2 along with the versions used.

Library	**Version**
pandas	1.2.4
Numpy	1.19.5
matplotlib	3.4.2
seaborn	0.11.1
datetime	
pickle	4.0
scikit-learn (sklearn)	0.24.2
imblearn	0.8.0
lightgbm	3.2.1

Table 2: Python Libraries used for Data Modelling

```
In [3]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import datetime
        import pickle

        from sklearn.model_selection import train_test_split
        from sklearn.model_selection import StratifiedKFold
        from sklearn.feature_extraction.text import CountVectorizer
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import confusion_matrix,f1_score,accuracy_score
        from sklearn.preprocessing import LabelEncoder
        from imblearn.over_sampling import SMOTE
        from imblearn.over_sampling import RandomOverSampler

        from lightgbm import LGBMClassifier
        from sklearn.model_selection import RandomizedSearchCV,GridSearchCV
        from sklearn.tree import DecisionTreeClassifier
        from sklearn.ensemble import RandomForestClassifier, AdaBoostClassifier

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

Figure 14: Packages for Data Modelling

The modules and sub-packages from the libraries mentioned in the Table 2 that are used for data modelling are depicted in the Figure 14. These packages are the prerequisites for the execution of machine learning models implemented.

3.8 Data Preparation

The categorical variables such as "payment type", "order status", "product category name", and "RFM Level" are transformed in the numeric values to train the model using LabelEncoder as shown in Figure 15.

[6] `https://docs.python.org/3/library/pickle.html`
[7] `https://scikit-learn.org/stable/`
[8] `https://imbalanced-learn.org/stable/`
[9] `https://lightgbm.readthedocs.io/en/latest/Python-Intro.html`

```
In [19]: for colname, coltype in final_data.dtypes.iteritems():
         #        print(colname, coltype)
                 if (coltype == 'object'):
                     print(colname)
                     #Create an instance of label encoder
                     labelencoder = LabelEncoder()
                     #Assign a numerical value and storing it in another column
                     final_data[colname] = labelencoder.fit_transform(final_data[colname]) # '0:Can\'t loose them','1: Champions','2:Loyal

         payment_type
         order_status
         product_category_name_english
         RFM_Level
```

Figure 15: Data Preparation for modelling

3.9 Data Oversampling

The data is highly imbalanced (79% posiive and 21% negative). This might yeild poor results.

```
In [24]: random=RandomOverSampler(random_state=0)
         X_random, y_random = random.fit_resample(X,y)

In [25]: oversampler=SMOTE(random_state=0)
         X_smote, y_smote = oversampler.fit_resample(X,y)

In [26]: print("Positive Reviews in SMOTE",X_smote[y_smote==1].shape , "Positive Reviews in SMOTE",X_smote[X_smote.review_score==1].shape

         Positive Reviews in SMOTE (65941, 40) Positive Reviews in SMOTE (65941, 40)

In [27]: print("Positive Reviews in Random Sample",X_random[y_random==1].shape , "Positive Reviews in Random Sample",X_random[X_random.rev

         Positive Reviews in Random Sample (65941, 40) Positive Reviews in Random Sample (65941, 40)
```

Figure 16: Oversample data using SMOTE and Random Sampling

For this purpose, Synthetic Minority Oversampling Technique (SMOTE) and Random Sampling is used as shown in Figure 16. The distribution of features when randomly oversampled was similar to the original distribution of features as observed in the original dataset. However, with SMOTE oversampling, the distribution is slightly deviated as compared to that of the original data. Therefore, randomly oversampled data is used for training the models.

3.10 Stratified Train-Test Split

The randomly oversampled data is used for splitting it into the training(80%) and testing(20%) datasets as shown in Figure 17.

```
In [32]: # train test split
         from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y,random_state=25)


         print(" X_train","   y_train")
         print(X_train.shape,y_train.shape)
         print('-'*15)
         print("X_test","    y_test")
         print(X_test.shape,y_test.shape)

          X_train    y_train
         (105505, 39) (105505,)
         ---------------
         X_test     y_test
         (26377, 39) (26377,)
```

Figure 17: Train-Test Split

3.11 Classification Models

3.11.1 Random Forest Model

In this study, the Random Forest Classifier is used for the binary classification of reviews as positive or negative. The hyperparameter tuning of this model is implemented in Figure 18, and Figure 19. The best parameters of the model obtained are depicted in Figure 20. The model is then trained with these best parameters as shown in Figure 21.

```
In [36]: estimators = [5,10,25,50,100,150]
train_scores = []
test_scores = []
for i in estimators:
    clf = RandomForestClassifier(criterion='gini',
            max_depth=None,min_samples_leaf=1, min_samples_split=2,
            n_estimators=i, random_state=25,verbose=0)
    clf.fit(X_train,y_train)
    train_sc = f1_score(y_train,clf.predict(X_train),average='macro')
    test_sc = f1_score(y_test,clf.predict(X_test),average='macro')
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    train_ac = accuracy_score(y_train,clf.predict(X_train))
    test_ac = accuracy_score(y_test,clf.predict(X_test))
    print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc,'Train Accuracy',train_ac, 'Test Accuracy',test_ac)
plt.plot(estimators,train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
plt.plot(estimators,test_scores,label='Test Score',color='#005777',linewidth=2)
plt.xlabel('Estimators')
plt.ylabel('Score')
plt.legend()
plt.title('Estimators vs score')
plt.grid()
```

```
Estimators =  5 Train Score 0.9923888967017664 test Score 0.9167470414926302 Train Accuracy 0.9923889863039667 Test Accuracy 0.
9169352087045533
Estimators =  10 Train Score 0.997118607098022 test Score 0.9275734946928391 Train Accuracy 0.9971186199706175 Test Accuracy 0.
9277400765818705
Estimators =  25 Train Score 0.999649305719982 test Score 0.9615896760279714 Train Accuracy 0.999649305720108 Test Accuracy 0.9
615953292641316
Estimators =  50 Train Score 0.9999905217762192 test Score 0.9638655491284874 Train Accuracy 0.9999905217762192 Test Accuracy
0.9638700382909353
Estimators =  100 Train Score 1.0 test Score 0.9665214939993008 Train Accuracy 1.0 Test Accuracy 0.9665238654888729
Estimators =  150 Train Score 1.0 test Score 0.9670147653979506 Train Accuracy 1.0 Test Accuracy 0.967016719111347
```

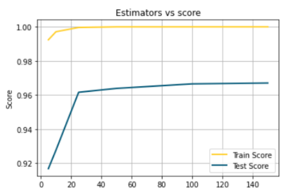


Figure 18: n$_{e}stimatorsforRandomForestModel$

```
In [37]: depths = [None,3,9,11,15,20]
train_scores = []
test_scores = []
for i in depths:
    clf = RandomForestClassifier(criterion='gini',
            max_depth=i, min_samples_leaf=1, min_samples_split=2,
            n_estimators=100,random_state=25,verbose=0)
    clf.fit(X_train,y_train)
    train_sc = f1_score(y_train,clf.predict(X_train),average='macro')
    test_sc = f1_score(y_test,clf.predict(X_test),average='macro')
    train_ac = accuracy_score(y_train,clf.predict(X_train))
    test_ac = accuracy_score(y_test,clf.predict(X_test))
    test_scores.append(test_sc)
    train_scores.append(train_sc)
    print('depth = ',i,'Train Score',train_sc,'test Score',test_sc,'Train Accuracy',train_ac, 'Test Accuracy',test_ac)
plt.plot([None,3,9,11,15,20],train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
plt.plot([None,3,9,11,15,20],test_scores,label='Test Score',color='#005777',linewidth=2)
plt.xlabel('Depth')
plt.ylabel('Score')
plt.title('Depth vs score')
plt.legend()
plt.grid()
plt.show()
```

```
depth =  None Train Score 1.0 test Score 0.9665214939993008 Train Accuracy 1.0 Test Accuracy 0.9665238654888729
depth =  3 Train Score 0.6247391799897997 test Score 0.6284136680936194 Train Accuracy 0.6496185014928203 Test Accuracy 0.65242
44607044015
depth =  9 Train Score 0.6757168936965388 test Score 0.6645246895368409 Train Accuracy 0.6900241694706412 Test Accuracy 0.67877
31735982105
depth =  11 Train Score 0.7217353336778716 test Score 0.6955134827303466 Train Accuracy 0.7325340031278138 Test Accuracy 0.7065
246237252152
depth =  15 Train Score 0.8743658543318396 test Score 0.8172595446113062 Train Accuracy 0.8757499644566609 Test Accuracy 0.8194
260150889032
depth =  20 Train Score 0.9875253864028753 test Score 0.9410769498772896 Train Accuracy 0.9875266575043837 Test Accuracy 0.9410
850362057853
```

Figure 19: Depth for Random Forest Model

```
In [38]: from sklearn.metrics import f1_score
         from sklearn.ensemble import RandomForestClassifier
         from sklearn.model_selection import RandomizedSearchCV
         from scipy.stats import uniform


         param_dist = {"n_estimators": [50,100,120,150],
                       "max_depth": [None, 1,2,5,10],
                       "min_samples_split": [2,4,6,8],
                       "min_samples_leaf": [1,2,3],
                       "criterion" : ['gini','entropy']}

         clf = RandomForestClassifier(random_state=25,n_jobs=-1)

         rf_random = RandomizedSearchCV(clf, param_distributions=param_dist,
                                        n_iter=5,cv=10,scoring='f1_macro',random_state=25,return_train_score=True )

         rf_random.fit(X_train,y_train)
         print('mean test scores',rf_random.cv_results_['mean_test_score'])
         print('mean train scores',rf_random.cv_results_['mean_train_score'])

         mean test scores [0.94298645 0.67424276 0.66790159 0.63373945 0.62073768]
         mean train scores [0.99077242 0.69629855 0.68728359 0.63509862 0.62090063]

In [39]: # printing best parameters and score
         print("Best Parameters: ",rf_random.best_params_)
         print("Best Score: ",rf_random.best_score_)

         Best Parameters:  {'n_estimators': 100, 'min_samples_split': 4, 'min_samples_leaf': 3, 'max_depth': None, 'criterion': 'gini'}
         Best Score:  0.9429864452299995
```

Figure 20: Best Parameters for Random Forest Model

```
In [41]: # Fitting the model on best parameters
         rf_classifier = RandomForestClassifier(max_depth = None, min_samples_leaf = 3, min_samples_split = 4, n_estimators = 100,criteric
                                      n_jobs=-1)
         rf_classifier.fit(X_train,y_train)

         y_train_pred_rf = rf_classifier.predict(X_train)
         y_test_pred_rf = rf_classifier.predict(X_test)

         # printing train and test scores
         print('Train f1 score',f1_score(y_train,y_train_pred_rf,average='macro'))
         print('Test f1 score',f1_score(y_test,y_test_pred_rf,average='macro'))

         # printing train and test scores Accuracy
         print('Train Accuracy',accuracy_score(y_train,y_train_pred_rf))
         print('Test Accuracy',accuracy_score(y_test,y_test_pred_rf))

         Train f1 score 0.9915454139902318
         Test f1 score 0.9502974566020448
         Train Accuracy 0.9915454243874698
         Test Accuracy 0.9502976077643401
```

Figure 21: Random Forest training with best parameters

3.11.2 Light Gradient Boosting Model (LGBM)

The LGBM Classifier is also used for the binary classification of reviews as positive or negative. The hyperparameter tuning of this model is implemented in Figure 22. The best parameters of the model obtained are depicted in Figure 23. The model is then trained with these best parameters as shown in Figure 24.

```
In [48]: # Variation of score with estimators used in LGBM with other parameters set to default value
         # estimators = [1,3,5,10,50,100,250,500,1000]
         estimators = [300,320,350, 400,430,450,500,530,550,600,650,700,800,900,1000,1050,1100,1200,1500]
         train_scores = []
         test_scores = []
         for i in estimators:
             clf_lgbm = LGBMClassifier(n_estimators=i,random_state=25)
             clf_lgbm.fit(X_train,y_train)
             train_sc = f1_score(y_train,clf_lgbm.predict(X_train),average='macro')
             test_sc = f1_score(y_test,clf_lgbm.predict(X_test),average='macro')
             test_scores.append(test_sc)
             train_scores.append(train_sc)
             print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc)
         plt.plot(estimators,train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
         plt.plot(estimators,test_scores,label='Test Score',color='#005777',linewidth=2)
         plt.xlabel('Estimators')
         plt.ylabel('Score')
         plt.legend()
         plt.title('Estimators vs score')
         plt.grid()

         Estimators =  300 Train Score 0.7912744815015101 test Score 0.7423389579234455
         Estimators =  320 Train Score 0.7990425289297217 test Score 0.7474203495212701
         Estimators =  350 Train Score 0.8083650467184074 test Score 0.7528506191877763
         Estimators =  400 Train Score 0.8215193708166163 test Score 0.7630329167683226
         Estimators =  430 Train Score 0.8296791631608378 test Score 0.7679731640134624
         Estimators =  450 Train Score 0.8371795796149228 test Score 0.7739262817939265
         Estimators =  500 Train Score 0.8487119834180357 test Score 0.7845279352994268
         Estimators =  530 Train Score 0.8558716601731 test Score 0.7897093630116163
         Estimators =  550 Train Score 0.8605119478607667 test Score 0.7937501039110744
         Estimators =  600 Train Score 0.8698707486538857 test Score 0.799289517361891
         Estimators =  650 Train Score 0.8803379148486967 test Score 0.8065740791703282
         Estimators =  700 Train Score 0.8876819180270017 test Score 0.8126491333793622
         Estimators =  800 Train Score 0.9019868225339427 test Score 0.8255710014735521
         Estimators =  900 Train Score 0.915270354145691 test Score 0.8378575417719616
         Estimators =  1000 Train Score 0.924795166260594 test Score 0.8470163480720149
         Estimators =  1050 Train Score 0.9293584739781553 test Score 0.8500013939152729
         Estimators =  1100 Train Score 0.9342042266305637 test Score 0.856020921492171
         Estimators =  1200 Train Score 0.9428303523138124 test Score 0.8638713642559436
         Estimators =  1500 Train Score 0.9606479638364469 test Score 0.8833692495946253
```

Figure 22: n_estimators for LGBM

```
In [49]: x_cfl_lgbm=LGBMClassifier(random_state=25,n_jobs=-1)

         prams={
             'learning_rate':[0.01,0.03,0.05,0.1,0.15,0.2],
              'n_estimators':[300,320,350, 400,430,450,500,530,550,600,650,700,800,900,1000,1050,1100,1200,1500],
              'max_depth':[1,3,5,10,15,20],
             'colsample_bytree':[0.1,0.3,0.5,1],
             'subsample':[0.1,0.3,0.5,1]
         }
         random_cfl1_lgbm=RandomizedSearchCV(x_cfl_lgbm,param_distributions=prams,verbose=10,n_jobs=-1,random_state=25,scoring='f1_macro',
                                     return_train_score=True)
         random_cfl1_lgbm.fit(X_train,y_train)

         print('mean test scores',random_cfl1_lgbm.cv_results_['mean_test_score'])
         print('mean train scores',random_cfl1_lgbm.cv_results_['mean_train_score'])

         Fitting 5 folds for each of 10 candidates, totalling 50 fits
         mean test scores [0.70341997 0.70043838 0.84159933 0.65886863 0.66218285 0.70701284
          0.76093105 0.71387439 0.88527419 0.69806863]
         mean train scores [0.73879175 0.73528977 0.93661215 0.66857174 0.67426252 0.74640178
          0.82875618 0.75768447 0.97607918 0.73141795]

In [50]: # printing best parameters and score
         print("Best Parameters: ",random_cfl1_lgbm.best_params_)
         print("Best Score: ",random_cfl1_lgbm.best_score_)

         Best Parameters:  {'subsample': 0.3, 'n_estimators': 1500, 'max_depth': 10, 'learning_rate': 0.15, 'colsample_bytree': 0.3}
         Best Score:  0.8852741862145302
```

Figure 23: Best parameters for LGBM

```
In [52]: # Fitting the model on best parameters
         lgbm = LGBMClassifier(n_estimators=1500, max_depth=10,subsample=0.3,learning_rate=0.15,colsample_bytree=0.3,random_state=25)
         lgbm.fit(X_train,y_train)


         y_train_pred_lgbm = lgbm.predict(X_train)
         y_test_pred_lgbm = lgbm.predict(X_test)

         # printing train and test scores
         print('Train f1 score',f1_score(y_train,y_train_pred_lgbm,average='macro'))
         print('Test f1 score',f1_score(y_test,y_test_pred_lgbm,average='macro'))

         # printing train and test scores Accuracy
         print('Train Accuracy',accuracy_score(y_train,y_train_pred_lgbm))
         print('Test Accuracy',accuracy_score(y_test,y_test_pred_lgbm))

         Train f1 score 0.9711335204325451
         Test f1 score 0.8980544057997748
         Train Accuracy 0.9711388085872708
         Test Accuracy 0.8980551237820829
```

Figure 24: LGBM training with best parameters

3.11.3 AdaBoost Model

The AdaBoost Classifier is the final model used for the binary classification of reviews. The hyperparameter tuning of this model is implemented in Figure 25. The best parameters of the model obtained are depicted in Figure 26. The model is then trained with these best parameters as shown in Figure 27.

```
In [55]: estimators = [25,50,100,200,250,300,500,750,1000]
         train_scores = []
         test_scores = []
         for i in estimators:
             clf = AdaBoostClassifier(learning_rate = 1,base_estimator = None,n_estimators=i, random_state=25)
             clf.fit(X_train,y_train)
             train_sc = f1_score(y_train,clf.predict(X_train),average='macro')
             test_sc = f1_score(y_test,clf.predict(X_test),average='macro')
             test_scores.append(test_sc)
             train_scores.append(train_sc)
             train_ac = accuracy_score(y_train,clf.predict(X_train))
             test_ac = accuracy_score(y_test,clf.predict(X_test))
             print('Estimators = ',i,'Train Score',train_sc,'test Score',test_sc,'Train Accuracy',train_ac, 'Test Accuracy',test_ac)
         plt.plot(estimators,train_scores,label='Train Score',color='#ffcf2b',linewidth=2)
         plt.plot(estimators,test_scores,label='Test Score',color='#005777',linewidth=2)
         plt.xlabel('Estimators')
         plt.ylabel('Score')
         plt.legend()
         plt.title('Estimators vs score')
         plt.grid()

         Estimators =  25 Train Score 0.6424395228548293 test Score 0.6440373787527929 Train Accuracy 0.6543007440405668 Test Accuracy
         0.6558744360617205
         Estimators =  50 Train Score 0.6466196143416666 test Score 0.6466317077443765 Train Accuracy 0.6574285578882517 Test Accuracy
         0.65765629146605
         Estimators =  100 Train Score 0.6510163582635617 test Score 0.6493572635690874 Train Accuracy 0.6618359319463533 Test Accuracy
         0.6604996777495545
         Estimators =  200 Train Score 0.6561082189894182 test Score 0.6527936927058613 Train Accuracy 0.6656366996824795 Test Accuracy
         0.6626227395079046
         Estimators =  250 Train Score 0.6586147225286824 test Score 0.6561336906383404 Train Accuracy 0.6677787782569546 Test Accuracy
         0.6655419494256359
         Estimators =  300 Train Score 0.6605089620572544 test Score 0.6569086306326491 Train Accuracy 0.6693521634045779 Test Accuracy
         0.6659968912309967
         Estimators =  500 Train Score 0.6657771497458014 test Score 0.6604927213262239 Train Accuracy 0.6738164068053647 Test Accuracy
         0.6688402775145013
         Estimators =  750 Train Score 0.6705922897608453 test Score 0.663379550307482 Train Accuracy 0.6779015212549169 Test Accuracy
         0.6710012510899648
         Estimators =  1000 Train Score 0.6742690326691729 test Score 0.6664153057847491 Train Accuracy 0.6811146391166295 Test Accuracy
         0.6736929901050157
```

Figure 25: n_estimators for AdaBoost Model

```
In [56]: # https://www.analyticsvidhya.com/blog/2016/03/complete-guide-parameter-tuning-xgboost-with-codes-python/
x_cfl_adab=AdaBoostClassifier(random_state=25)

prams={
    'learning_rate': [0.001,0.01,0.03,0.05,0.1,0.1,0.2],
     'n_estimators': [25,50,100,200,250,300,500,750,1000],
      'algorithm' : ['SAMME', 'SAMME.R']
}
random_cfl1_adab=RandomizedSearchCV(x_cfl_adab,param_distributions=prams,random_state=None,scoring='f1_macro',
                              return_train_score=True)
random_cfl1_adab.fit(X_train,y_train)

print('mean test scores',random_cfl1_adab.cv_results_['mean_test_score'])
print('mean train scores',random_cfl1_adab.cv_results_['mean_train_score'])

mean test scores [0.6311167  0.62064528 0.59020838 0.62064076 0.62105439 0.59020838
 0.62072349 0.60858948 0.62128435 0.63811522]
mean train scores [0.63212367 0.62076903 0.59026829 0.62074749 0.62131557 0.59026829
 0.62082348 0.60879959 0.62167616 0.63951851]

In [57]: # printing best parameters and score
print("Best Parameters: ",random_cfl1_adab.best_params_)
print("Best Score: ",random_cfl1_adab.best_score_)

Best Parameters:  {'n_estimators': 250, 'learning_rate': 0.1, 'algorithm': 'SAMME.R'}
Best Score:  0.638115218807908
```

Figure 26: Best Parameters for AdaBoost model

```
In [58]: # Fitting the model on best parameters
ada = AdaBoostClassifier(n_estimators=1000, learning_rate=0.1, algorithm='SAMME.R', random_state=None)
ada.fit(X_train,y_train)

y_train_pred_adab = ada.predict(X_train)
y_test_pred_adab = ada.predict(X_test)

# printing train and test scores
print('Train f1 score',f1_score(y_train,y_train_pred_adab,average='macro'))
print('Test f1 score',f1_score(y_test,y_test_pred_adab,average='macro'))

# printing train and test scores Accuracy
print('Train Accuracy',accuracy_score(y_train,y_train_pred_adab))
print('Test Accuracy',accuracy_score(y_test,y_test_pred_adab))

Train f1 score 0.6481024698871446
Test f1 score 0.6482296474736925
Train Accuracy 0.6613335860859675
Test Accuracy 0.6614474731773894
```

Figure 27: Training AdaBoost Model with Best parameters

3.12 Evaluation and Results

The function was defined to plot the confusion matrices for train and test data. The function is depicted in the Figure 28

```
In [46]: def confusion_matrices_plot(y_train, y_train_pred, y_test,y_test_pred):
    # representing confusion matric in heatmap format
    # https://seaborn.pydata.org/generated/seaborn.heatmap.html
    group_names = ['True Negative','False Positive','False Negative','True Positive']
    C1 = confusion_matrix(y_train, y_train_pred)
    C2 = confusion_matrix(y_test,y_test_pred)

    fig,ax = plt.subplots(1, 2, figsize=(15,5))
    group_counts = ["{0:0.0f}".format(value) for value in C1.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in C1.flatten()/np.sum(C1)]
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
    zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    ax1 = sns.heatmap(C1, annot=labels, fmt='', cmap='Blues', ax = ax[0])
    ax1.set_xlabel('Predicted labels');ax1.set_ylabel('True labels');
    ax1.set_title('Train Confusion Matrix');
    ax1.xaxis.set_ticklabels(['Negative', 'Positive']); ax1.yaxis.set_ticklabels(['Negative', 'Positive']);


    group_counts = ["{0:0.0f}".format(value) for value in C2.flatten()]
    group_percentages = ["{0:.2%}".format(value) for value in C2.flatten()/np.sum(C2)]
    # categories = ['Negative Reviews', 'Positive Reviews']
    labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in
    zip(group_names,group_counts,group_percentages)]
    labels = np.asarray(labels).reshape(2,2)
    ax2 = sns.heatmap(C2, annot=labels, fmt='', cmap='Blues', ax = ax[1])
    ax2.set_xlabel('Predicted labels');ax2.set_ylabel('True labels');
    ax2.set_title('Test Confusion Matrix');
    ax2.xaxis.set_ticklabels(['Negative', 'Positive']); ax2.yaxis.set_ticklabels(['Negative', 'Positive']);


    plt.show()
```

Figure 28: Function to plot confusion matrix

3.12.1 Random Forest Model

The classification report for the Random Forest classification model trained on best parameters is depicted in Figure 29 along with the confusion matrices in Figure 30. The results shows that the overall accuracy of the test data using Random Forest model is 95% and the F1-score for positive and negative review class is 0.95.

```
In [45]: from sklearn.metrics import classification_report
         print("*"*30,"Training Dataset","*"*30)
         print(classification_report(y_train,y_train_pred_rf))
         print("*"*30,"Test Dataset","*"*30)
         print(classification_report(y_test,y_test_pred_rf))

****************************** Training Dataset ******************************
              precision    recall  f1-score   support

           0       0.99      0.99      0.99     52752
           1       0.99      0.99      0.99     52753

    accuracy                           0.99    105505
   macro avg       0.99      0.99      0.99    105505
weighted avg       0.99      0.99      0.99    105505

****************************** Test Dataset ******************************
              precision    recall  f1-score   support

           0       0.95      0.95      0.95     13189
           1       0.95      0.95      0.95     13188

    accuracy                           0.95     26377
   macro avg       0.95      0.95      0.95     26377
weighted avg       0.95      0.95      0.95     26377
```

Figure 29: Classification Report for Random Forest Model

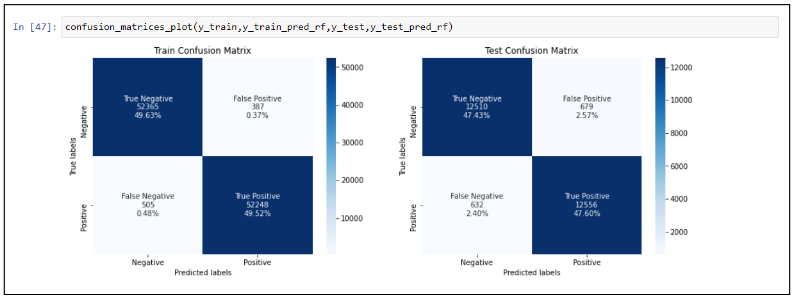


Figure 30: Confusion Matrices for Random Forest model

3.12.2 LGBM

The classification report for the LGBM classifier trained on best parameters is depicted in Figure 31 along with the confusion matrices in Figure 32. The results shows that the overall test accuracy for the LGBM model is 90% and the F1-score metric for both the positive and negative classes is 0.90.

```
****************************** Training Dataset ******************************
              precision    recall  f1-score   support

           0       0.98      0.96      0.97     52752
           1       0.96      0.98      0.97     52753

    accuracy                           0.97    105505
   macro avg       0.97      0.97      0.97    105505
weighted avg       0.97      0.97      0.97    105505

****************************** Test Dataset ******************************
              precision    recall  f1-score   support

           0       0.90      0.90      0.90     13189
           1       0.90      0.90      0.90     13188

    accuracy                           0.90     26377
   macro avg       0.90      0.90      0.90     26377
weighted avg       0.90      0.90      0.90     26377
```

Figure 31: Classification Report for LGBM

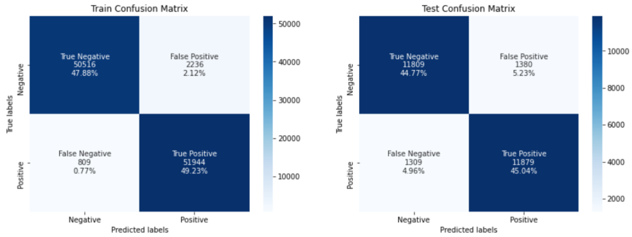


Figure 32: Confusion Matrices for LGBM

3.12.3 AdaBoost

The classification report for the AdaBoost classifier trained on best parameters is depicted in Figure 33 along with the confusion matrices in Figure 34. The results shows that the overall test accuracy for the AdaBoost classification model is 66% and the F1-score metric for negative review class is 0.58 and positive class is 0.72.

```
***************************** Training Dataset *****************************
              precision    recall  f1-score   support

           0       0.76      0.47      0.58     52752
           1       0.62      0.86      0.72     52753

    accuracy                           0.66    105505
   macro avg       0.69      0.66      0.65    105505
weighted avg       0.69      0.66      0.65    105505

***************************** Test Dataset *****************************
              precision    recall  f1-score   support

           0       0.76      0.47      0.58     13189
           1       0.62      0.86      0.72     13188

    accuracy                           0.66     26377
   macro avg       0.69      0.66      0.65     26377
weighted avg       0.69      0.66      0.65     26377
```

Figure 33: Classification Report for AdaBoost Model

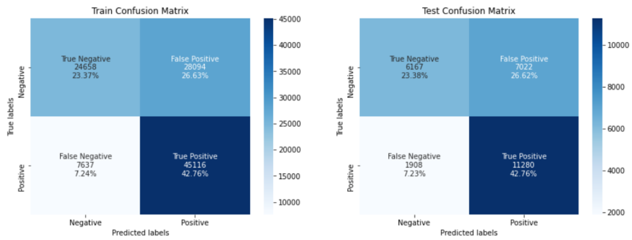


Figure 34: Confusion Matrices for AdaBoost Model