Capstone Project

Analysing E-Commerce Order Rejection

INTERIM REPORT

Mentored by:

Mr. Mohit Sahu

Submitted by: (Group – 8)

Atharva Sadanshive
Diwakar Kaul
Govind T
Vishnu Prasad Shetty K S
Vinutha R

Table of Contents

Overview

Industry Background and Key Objectives

Amazon is the world's largest and by far most visited E-Commerce platform in the world allowing hundreds of millions of customers to find an incredible range of products from a broad group of manufacturers and suppliers.

We are an in-house analytics team at this leading ecommerce player, tasked with better understanding order rejection by customers at various stages of delivery.

In particular, we have been assigned a dataset containing orders for women's clothing, a certain portion of which have been cancelled due to various reasons.

Business problem statement:

1) **Business Problem Understanding**
   In the hyper competitive online retail space, a continuing struggle for major players is to provide a superior shopping experience. E-Commerce players aim to achieve this partly by allowing easy product returns/order rejections by customers.

   However offering such a facility can be costly to the E-Commerce platform, as well as to the third party manufacturer/vendor which might be fulfilling and delivering the order.

   Such costly and time consuming product returns can hamper the long term profitability of the E-Commerce business and might even threaten the business' viability.

2) **Business Objective**
   Broadly speaking, recent developments in machine learning techniques and data mining has led to an interest of implementing these techniques in various fields. The e-commerce space is no different in this regard. Potentially, the implementation of machine learning techniques could lead greater overall profitability in the business.

   It is important for E-Commerce platforms to better understand the trends and key reasons involved in customers subsequently rejecting orders they have placed. This will not only help the business identify potential pain points in the shopping experience, but might also help in finding ways in which customer returns can be reduced, and this associated costs can be reduced. Ideally, both a reduction in order rejection volumes and an improvement in customer shopping experience can be achieved by implementing suggestions derived from such an analysis.

Data Dictionary

The Amazon online platform dataset consists of below mentioned attributes:

Category: Type of product.

Size: Size of the product.

Date: Date of the sale.

Status: Status of the sale.

Fulfilment: Method of fulfilment (Merchant, Amazon).

Style: Style of the product.

SKU: Stock Keeping Unit.

ASIN: Amazon Standard Identification Number.

Courier Status: Status of the courier.

Qty: Quantity of the product.

Amount: Amount of the sale.

B2B: Business to business sale.

Currency: The currency used for the sale.

Index: Used to maintain the data in the order.

Order ID: Order ID of the product ordered.

Sales Channel: It tells if purchased from amazon on not (Amazon.in, Non-Amazon).

Ship Service Level: It is based on the type of shipping method you choose and the number of delivery days it takes for your order to arrive (Standard, Expedited).

Ship City: City to which order has to be sent.

Ship State: State to which order has to be sent.

Ship Postal Code: Pin-code to which order has to be sent.

Ship Country: Country to which order has to be sent (India).

Promotion Ids: Promotion IDs in Amazon are unique codes used to identify promotions and discounts for products. These codes are typically used when setting up sales and promotions for products.

Unnamed: 22: Unwanted column.

Data Pre-processing

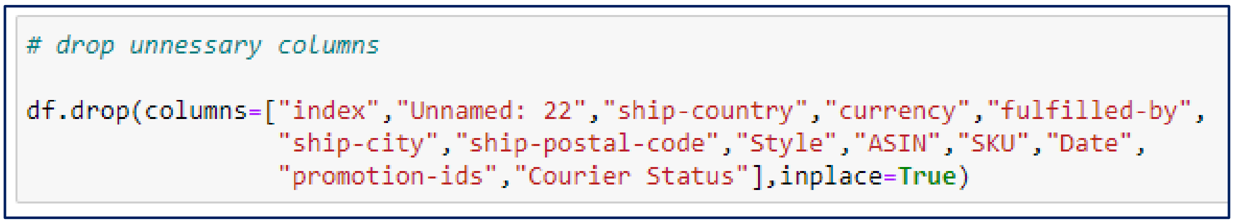
We begin by reading in the default csv file and getting a sense of the overall size and features we will be working with.

Our default data file includes:

Number of Columns: 24
Total Number of Records: 128975

Dropping irrelevant columns:

At this stage we drop columns that either have their contained information replicated in another column or simply contain data that will not be useful for our analysis.

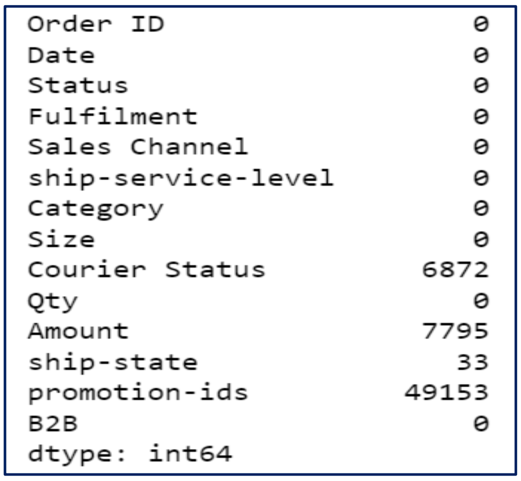


```python
# drop unnessary columns

df.drop(columns=["index","Unnamed: 22","ship-country","currency","fulfilled-by",
                  "ship-city","ship-postal-code","Style","ASIN","SKU","Date",
                  "promotion-ids","Courier Status"],inplace=True)
```

Imputation of missing values:

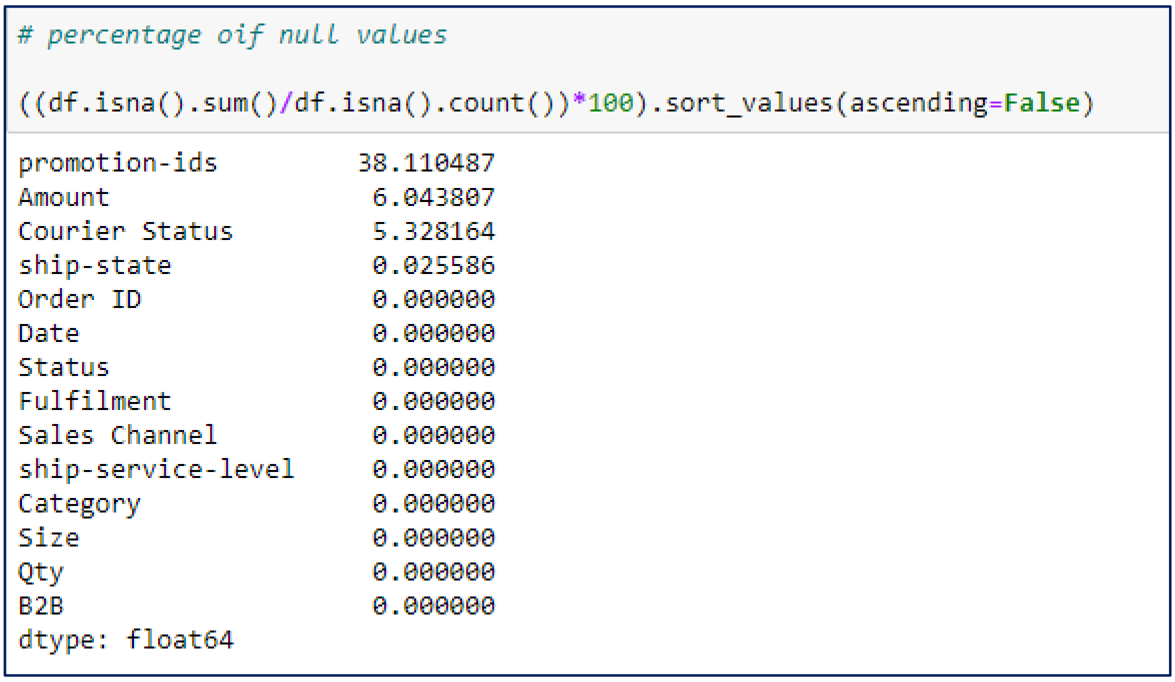
Our initial search helps us identify missing values. The columns 'Courier Status', 'Amount', 'ship-state' and 'promotion-ids' have data missing.

We will think about treatment of null values on a feature-by-feature basis.



```
Order ID                  0
Date                      0
Status                    0
Fulfilment                0
Sales Channel             0
ship-service-level        0
Category                  0
Size                      0
Courier Status         6872
Qty                       0
Amount                 7795
ship-state               33
promotion-ids         49153
B2B                       0
dtype: int64
```

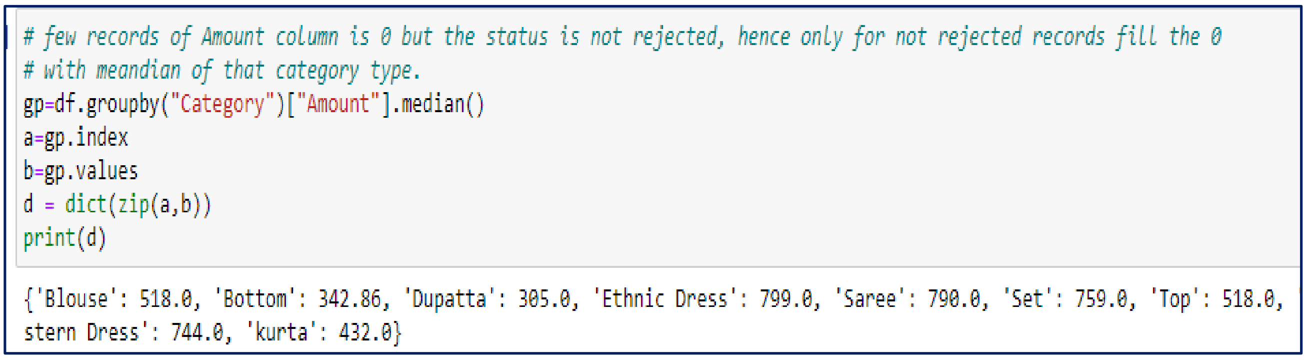
A quick check tells us that except for 'promotion-ids' column, the other features with missing values only have up to 6% of total as not present. Given the relatively smaller proportion, we shall proceed with imputing mean (for numeric columns) and median (for categorical columns) for missing values.



```
# percentage oif null values

((df.isna().sum()/df.isna().count())*100).sort_values(ascending=False)
```

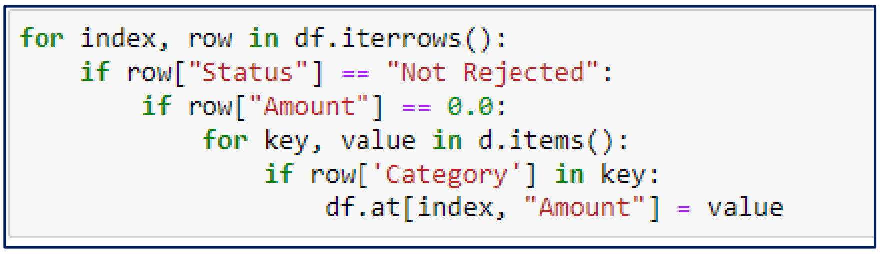
```
promotion-ids         38.110487
Amount                 6.043807
Courier Status         5.328164
ship-state             0.025586
Order ID               0.000000
Date                   0.000000
Status                 0.000000
Fulfilment             0.000000
Sales Channel          0.000000
ship-service-level     0.000000
Category               0.000000
Size                   0.000000
Qty                    0.000000
B2B                    0.000000
dtype: float64
```

However, specifically for the ‘Amount’ column, it would make sense to fill the missing values by category-type, in the ‘not-rejected’ segment. Essentially, we take the median for each clothing category and impute the null values in that segment accordingly.



```
# few records of Amount column is 0 but the status is not rejected, hence only for not rejected records fill the 0
# with meandian of that category type.
gp=df.groupby("Category")["Amount"].median()
a=gp.index
b=gp.values
d = dict(zip(a,b))
print(d)
```

```
{'Blouse': 518.0, 'Bottom': 342.86, 'Dupatta': 305.0, 'Ethnic Dress': 799.0, 'Saree': 790.0, 'Set': 759.0, 'Top': 518.0, '
stern Dress': 744.0, 'kurta': 432.0}
```

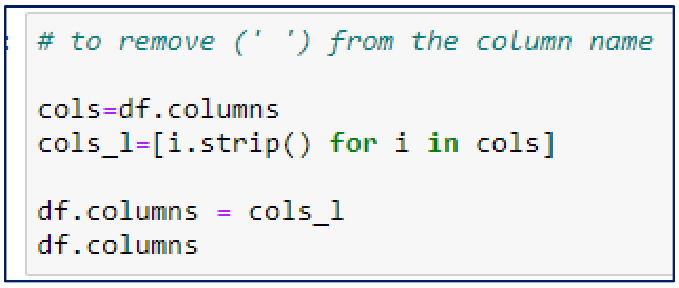


```
for index, row in df.iterrows():
    if row["Status"] == "Not Rejected":
        if row["Amount"] == 0.0:
            for key, value in d.items():
                if row['Category'] in key:
                    df.at[index, "Amount"] = value
```

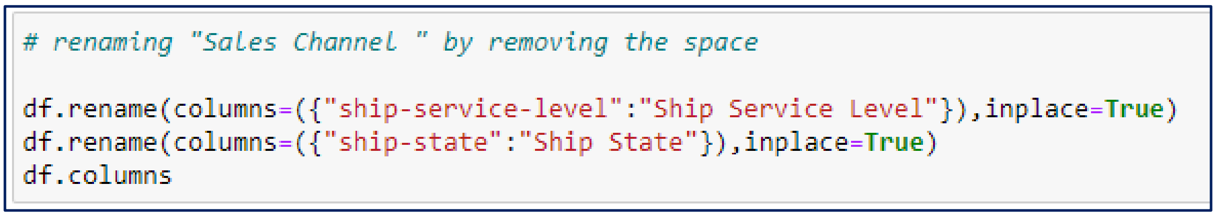
For the ‘promotion-ids’ column, since it is unclear whether the missing data points were part of the various promotions advertised to customers, we shall put those blanks under the ‘No Promotion’ segment.

Data cleaning/sanitization:

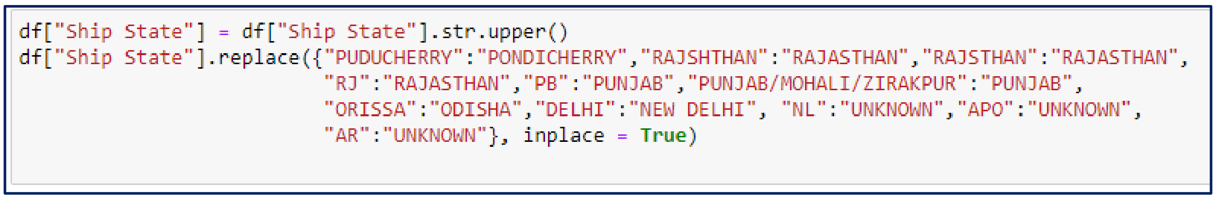
To make for easier reading, understanding and analysis, we have proceeded with renaming certain value descriptions and attempted other data cleaning processes including fixing grammatical errors in the original data.



```
# to remove (' ') from the column name

cols=df.columns
cols_l=[i.strip() for i in cols]

df.columns = cols_l
df.columns
```



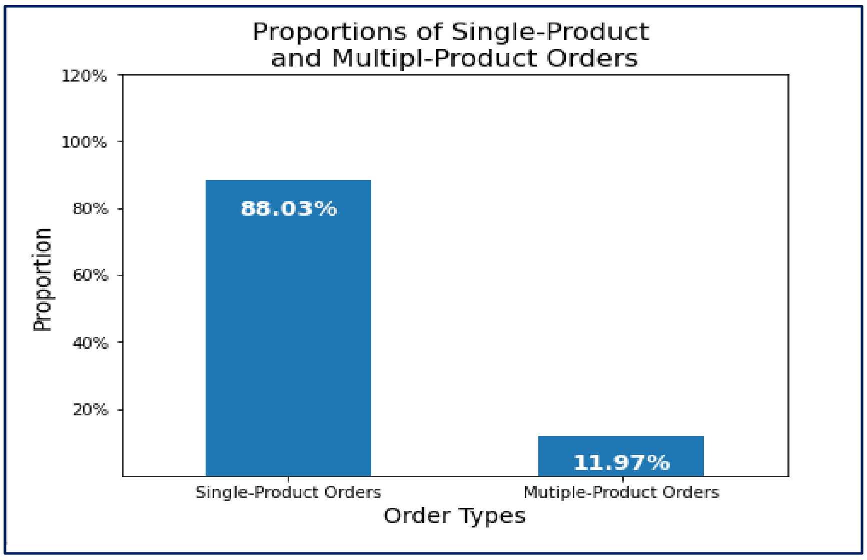
```
# renaming "Sales Channel " by removing the space

df.rename(columns=({"ship-service-level":"Ship Service Level"}),inplace=True)
df.rename(columns=({"ship-state":"Ship State"}),inplace=True)
df.columns
```



```
df["Ship State"] = df["Ship State"].str.upper()
df["Ship State"].replace({"PUDUCHERRY":"PONDICHERRY","RAJSHTHAN":"RAJASTHAN","RAJSTHAN":"RAJASTHAN",
                          "RJ":"RAJASTHAN","PB":"PUNJAB","PUNJAB/MOHALI/ZIRAKPUR":"PUNJAB",
                          "ORISSA":"ODISHA","DELHI":"NEW DELHI", "NL":"UNKNOWN","APO":"UNKNOWN",
                          "AR":"UNKNOWN"}, inplace = True)
```

Analysing the "orderid" column:

Here we attempt to understand the distribution between single and multiple orders within the 'orderid' column. Since multiple orders from a single id/customer may impact the overall probability of order rejection, we take a look at the proportion and make a decision on how to tackle orderids' that have placed multiple orders.



Above as we plot the proportion of single to multiple product orders, we see that about 88% are single-product orders. Hence with a large enough proportion of single-product orders, and in an attempt to avoid any misrepresentation from using multi-product orders, we drop the multi-product records.

Additionally, we now drop the 'orderid' column itself, as there is no further use for this column beyond the above stated segmentation.

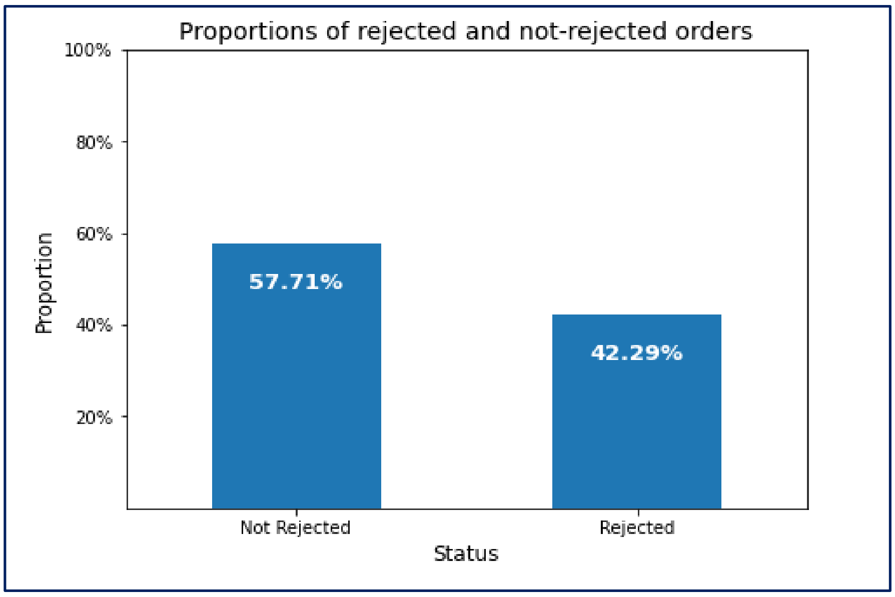
```
df = df[df["Order ID"].duplicated(keep = False) == False]

df.drop("Order ID", axis = 1, inplace = True)
```

Creating the target column:

Originally our target column 'Status' has multiple value names provided to signify whether an order had been rejected or not. We will work to classify them under a single value name. Additionally we shall remove records/rows where the order rejection information is either missing or is unclear.

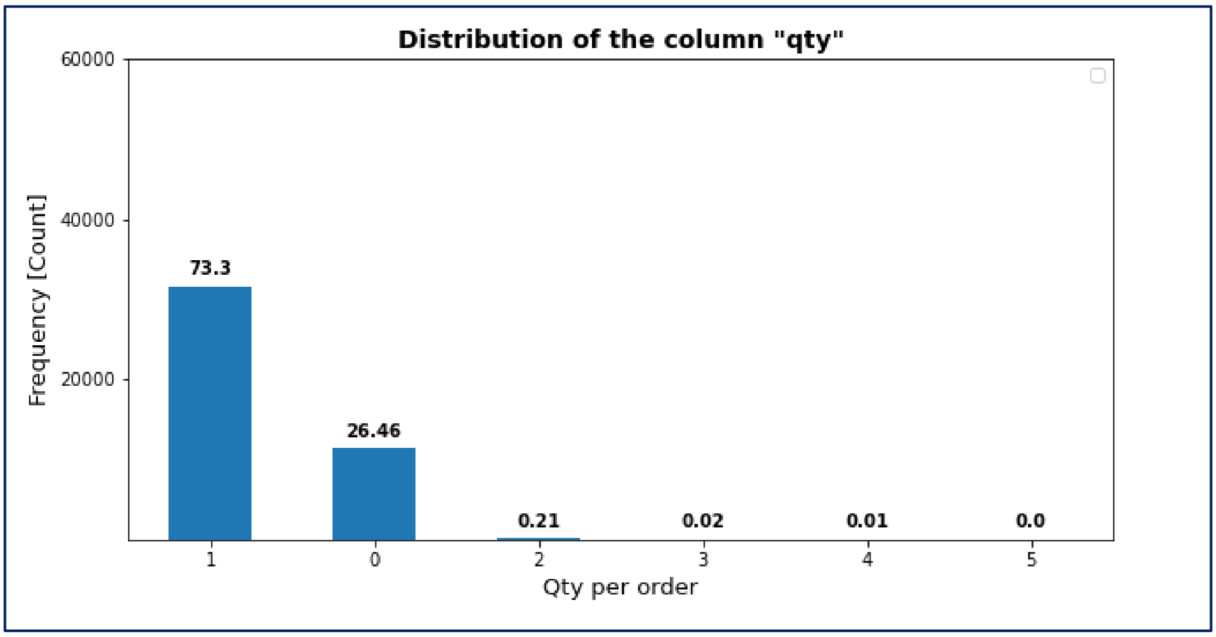
```
# drop the rows with unsure rejection status
known_value = ["Cancelled", 'Shipped - Returned to Seller','Shipped - Rejected by Buyer',
            'Shipped - Returning to Seller','Shipped - Delivered to Buyer']
df = df[df["Status"].isin(known_value)]

# create a col "rejected" where value 1 means rejected and 0 means not-rejected"
rejected = ["Cancelled", 'Shipped - Returned to Seller','Shipped - Rejected by Buyer',
            'Shipped - Returning to Seller']
df["Status"] = df["Status"].isin(rejected).astype(int)    # change the dtype to "int"
```

After renaming and reclassifying the values within the status column, we can see below that there is a slight class imbalance in the proportion of 'rejected' to 'not-rejected' segment. For now we shall work with the column as given, and may look to address the imbalance at a later stage.



Analysing feature "qty" - Quantity per Order:

Below we can see the proportion distribution for the quantity per order column. Given the highly skewed distribution of data here, and keeping in mind the adjustments to the data we made in the ‘orderid’ column, we can drop the ‘qty’ column in its entirety. We do not see any value to be derived from using the ‘qty’ column for now.

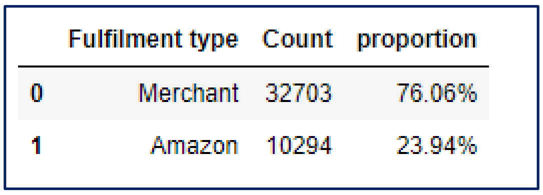


Analysing feature “Amount”:

While we can clearly see significant presence of outliers within the ‘Amount’ column, we will only be removing the far/extreme outlier values, and leaving the majority of the outlier values intact. This is to adjust for the impact of the very small number of extremely large values present in ‘Amount’ while simultaneously keeping in the other outlier values, which would give us an indication of how the order rejection propensity consists amongst the ‘luxury’ segment.

Analysing feature ‘Fulfilment’:

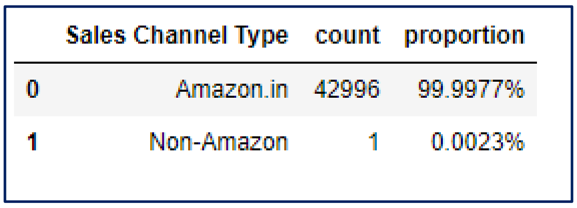
At first glance we see that delivery for the majority of the orders has been handled by third-party delivery services, while 24% of the order deliveries have been handled by Amazon itself. This shall give us good insights into how order rejection gets affected when handled by Amazon vs. third-parties, and hence we shall not make any adjustments to this column.



	Fulfilment type	Count	proportion
0	Merchant	32703	76.06%
1	Amazon	10294	23.94%

Analysing feature “Sales Channel”:

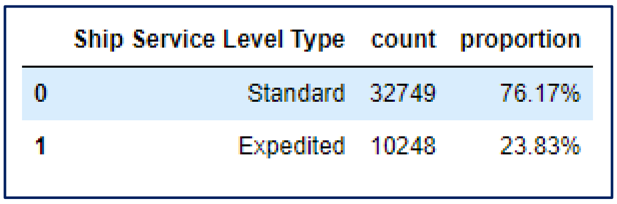
This column looks broadly irrelevant with 99.9977% of the orders being processed via the amazon.in itself, and as such we shall drop this column.



	Sales Channel Type	count	proportion
0	Amazon.in	42996	99.9977%
1	Non-Amazon	1	0.0023%

Analysing feature "Ship Service Level":

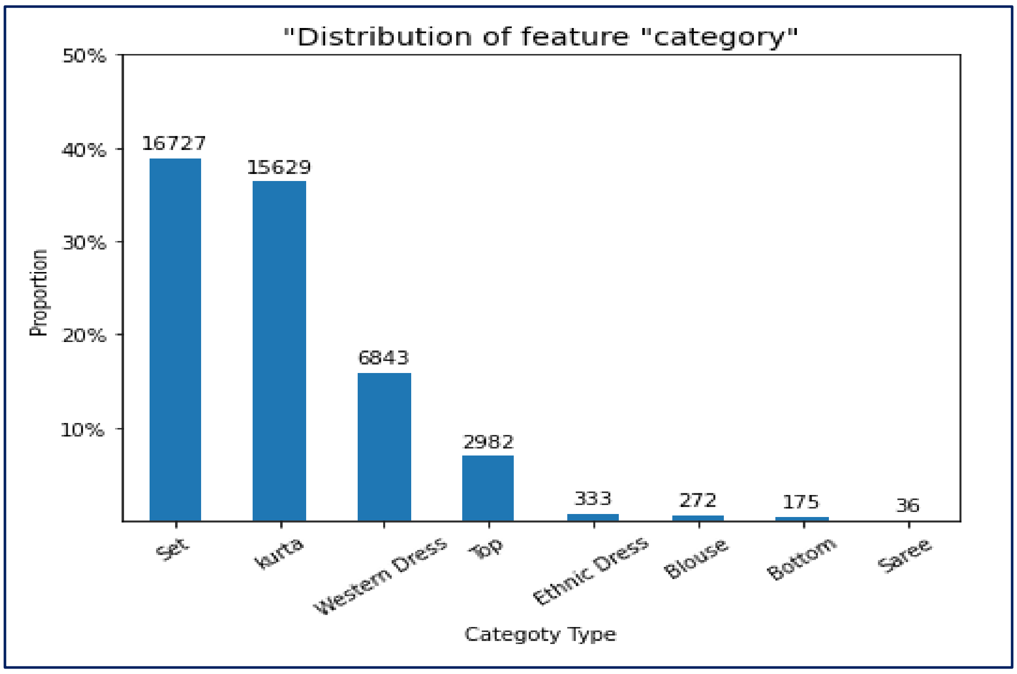
This column has valuable information about whether an order was shipped under ‘Standard’ or ‘Expedited’ process, and will indicate whether an order was more likely to be placed by customers under the ‘Prime’ program of Amazon, or those that were willing to pay more for faster delivery of their order. We will keep this column in its current form and use for further analysis.



	Ship Service Level Type	count	proportion
0	Standard	32749	76.17%
1	Expedited	10248	23.83%

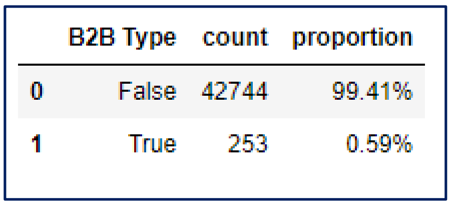
Analysing feature “Category”:

Visualizing the column below we can clearly marked segments within the women’s clothing category. Hence there is no requirement for further preparing this column and we shall use this column as given for further analysis.



Analysing feature “B2B”:

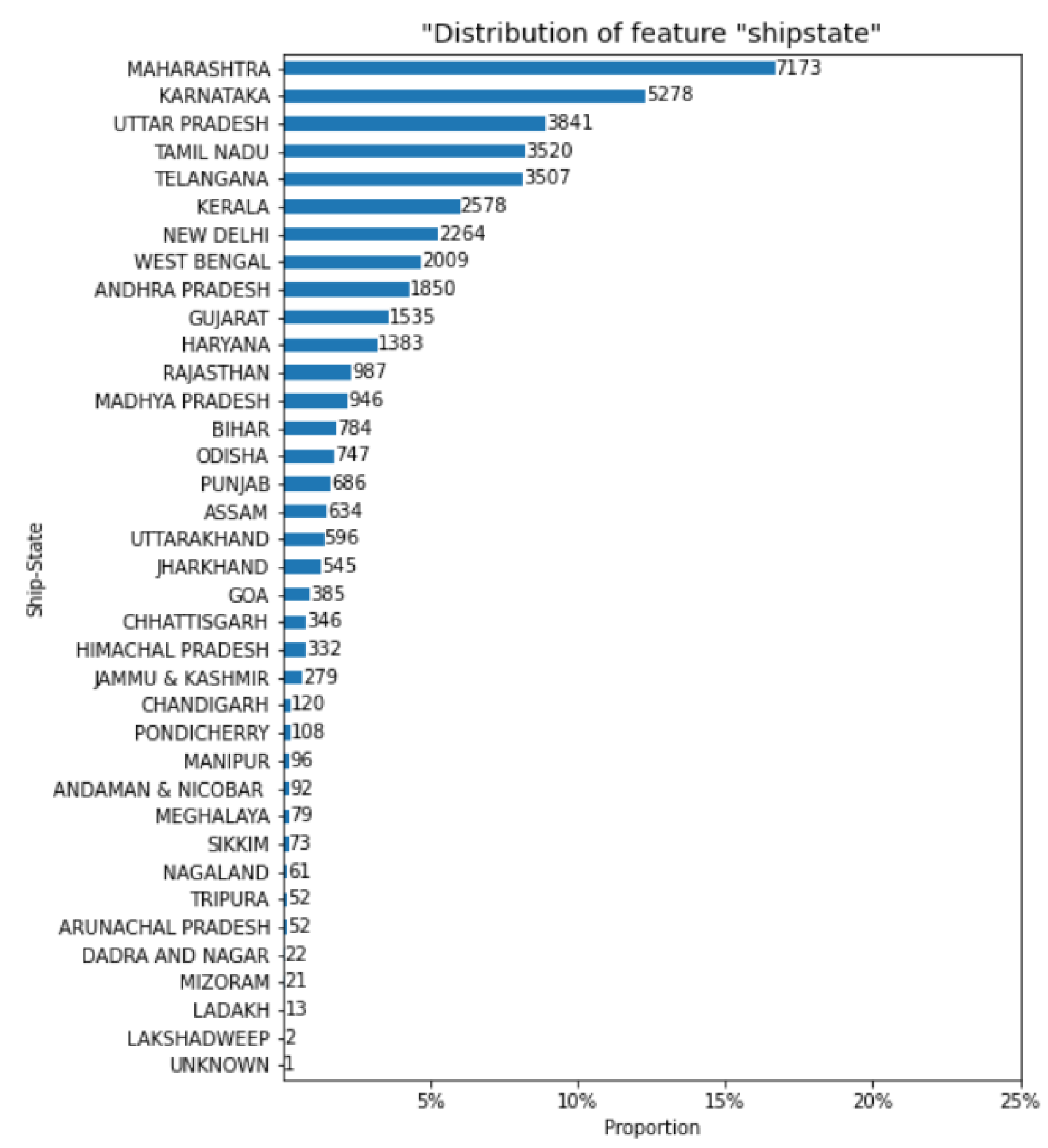
As seen below, only a very small portion of the orders are within the B2B segment, however for now we shall proceed with using this data point.



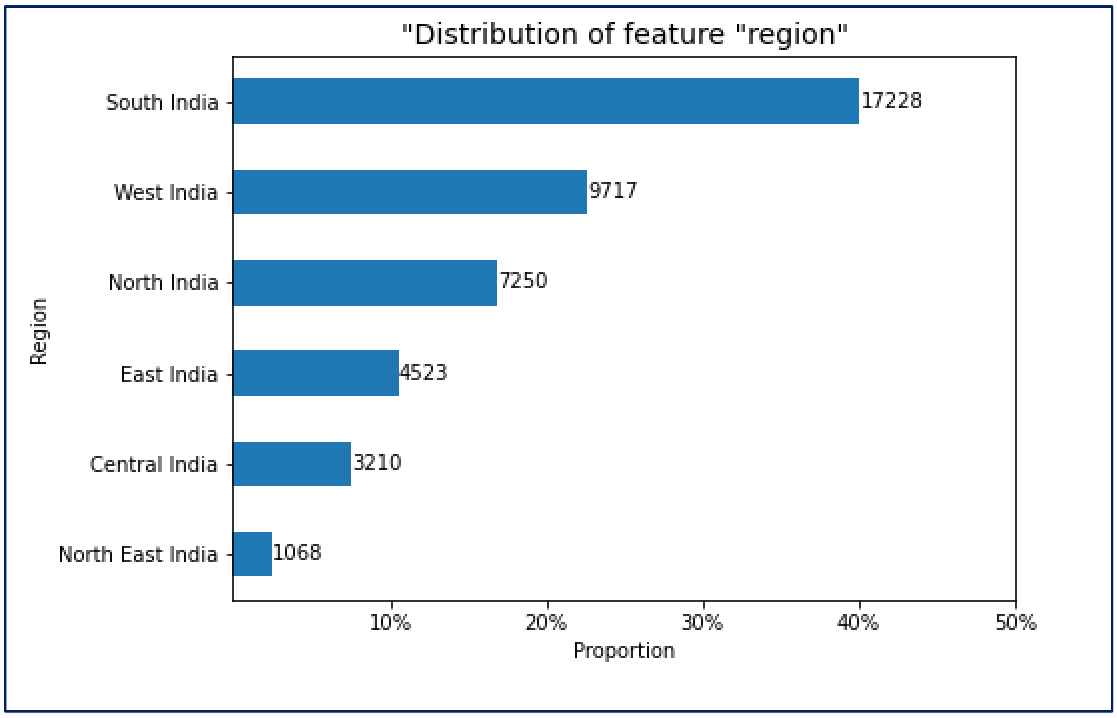
	B2B Type	count	proportion
0	False	42744	99.41%
1	True	253	0.59%

Analysing feature “Ship State”:

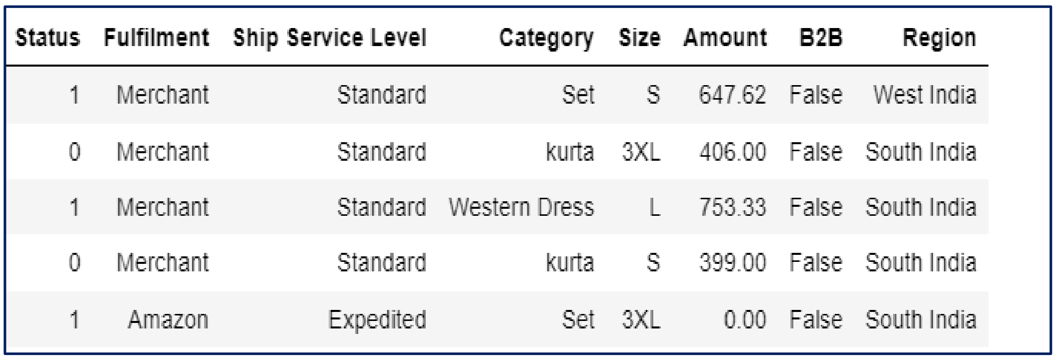
Rather than a state-wise analysis, where states such as ‘Maharashtra’ containing a significant proportion of the order may have otherwise biased the interpretation, we have attempted to segment the India-wide distribution of orders under various sub-regions in India.



The resultant distribution of data under sub-regions looks as is shown below. Each region now has a healthy proportion of data points, and we shall use this distribution to proceed with analysis. This new feature “Region” shall be used moving forward and we will drop the column “Ship State”



Data snapshot post pre-processing conducted so far:

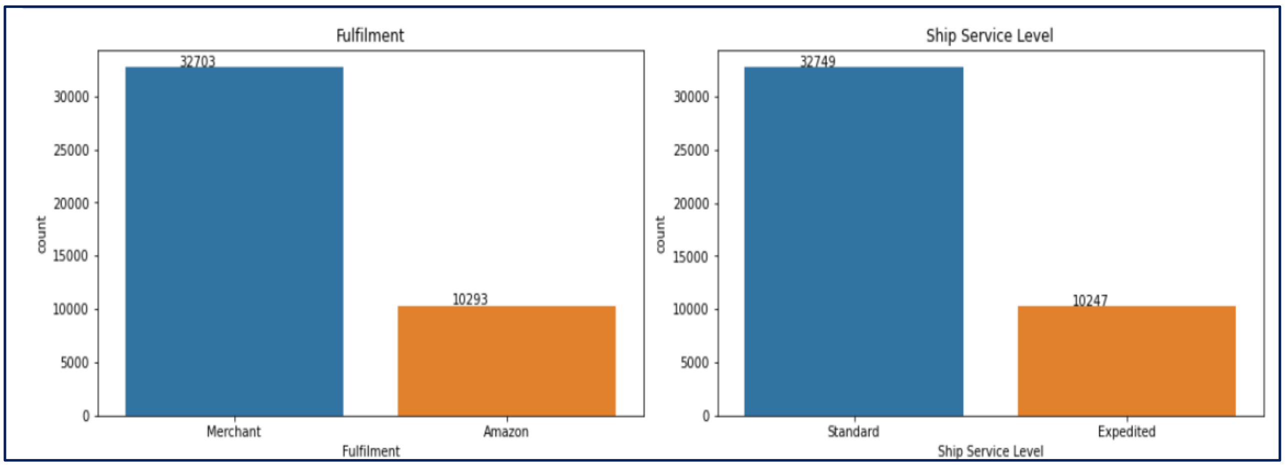


Status	Fulfilment	Ship Service Level	Category	Size	Amount	B2B	Region
1	Merchant	Standard	Set	S	647.62	False	West India
0	Merchant	Standard	kurta	3XL	406.00	False	South India
1	Merchant	Standard	Western Dress	L	753.33	False	South India
0	Merchant	Standard	kurta	S	399.00	False	South India
1	Amazon	Expedited	Set	3XL	0.00	False	South India

Initial Data Exploration - Analysing Relationships between the Variables

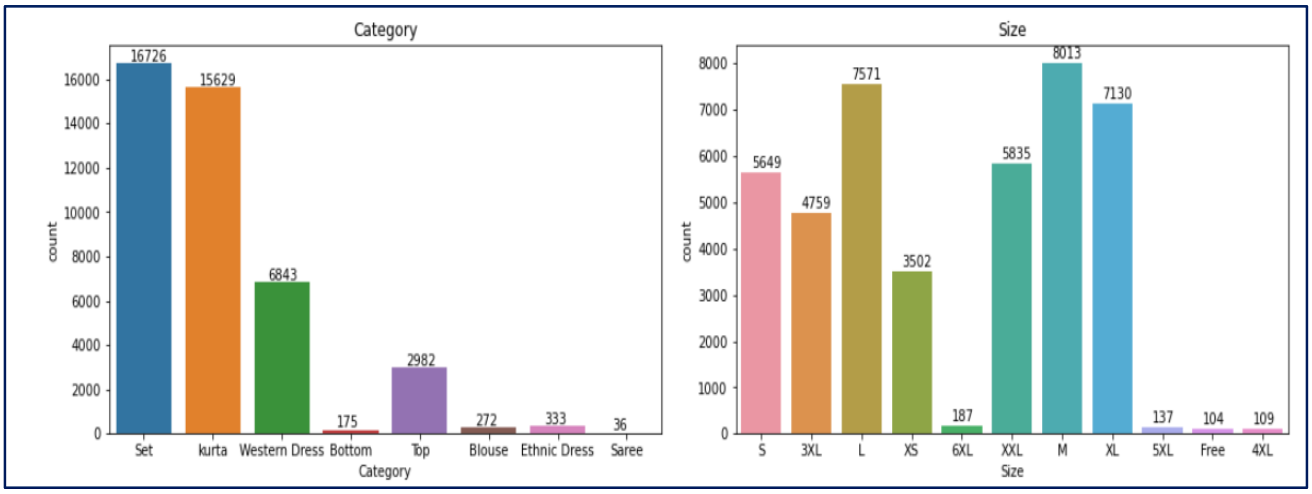
Univariate analysis

Let us begin by charting the updated categorical columns:

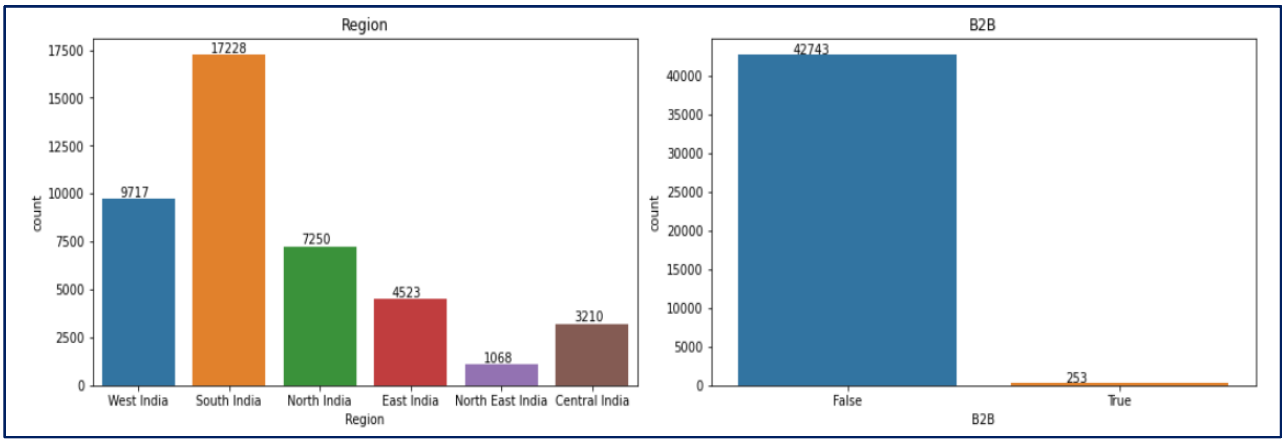


Regarding order fulfilment, an important point to highlight will be that Amazon will have more flexibility and can better guarantee implementation of any changes that might be suggested via this analysis. This is valuable because such an impact would not be possible if most suggestions were to be made to third party delivery services.

Next, visualising the 'Category' and 'Size' data points tells us that most of the orders placed have been either a set (combination of clothing types) or kurtas in particular; with a broad range of sizes being selected, all the way from small to extra-extra-large seeing robust demand.



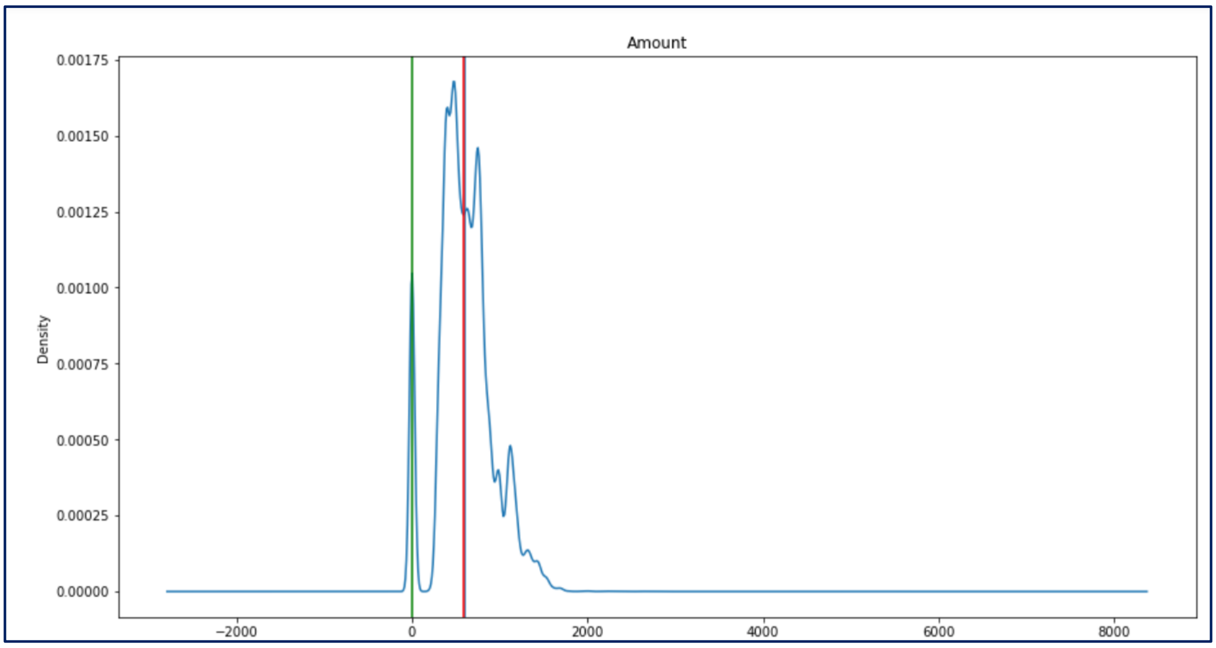
Finally, comparing B2B vs. B2B orders, and comparing by region, we recognise that a majority of the orders are from southern Indian locations and most had been shipped at the point of data collection.



Now, let us understand what the numerical columns are displaying:

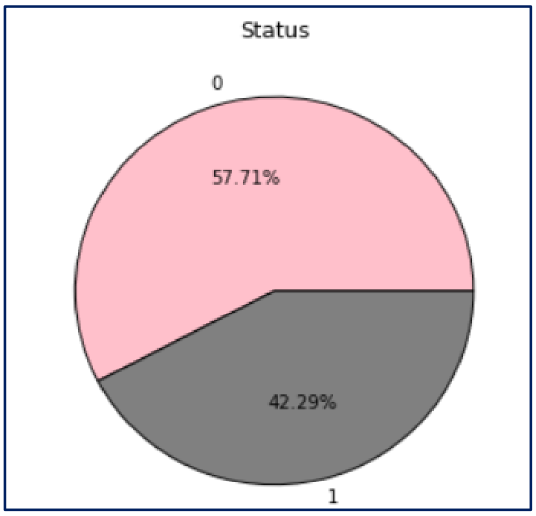
Most orders contain a single unit or a single set. These orders have an average value of Rs. 545.53, with Rs. 533.665 being the most common order value.

Additionally, as can be seen below, a smaller segment of the orders have a much higher price value; which can be seen as the “luxury” segment, and hence and it might make sense to focus on them particularly to solve pain points in order processing and delivery.



Displaying the target variable:

We reconfirm the fact that a majority of the orders are categorised under the ‘Not Rejected’ section, with about 42.29% of the orders being categorised as ‘Rejected’. We hope to see a reasonable reduction in the % share for ‘Rejected’ post implementation of suggestions made via this analysis.



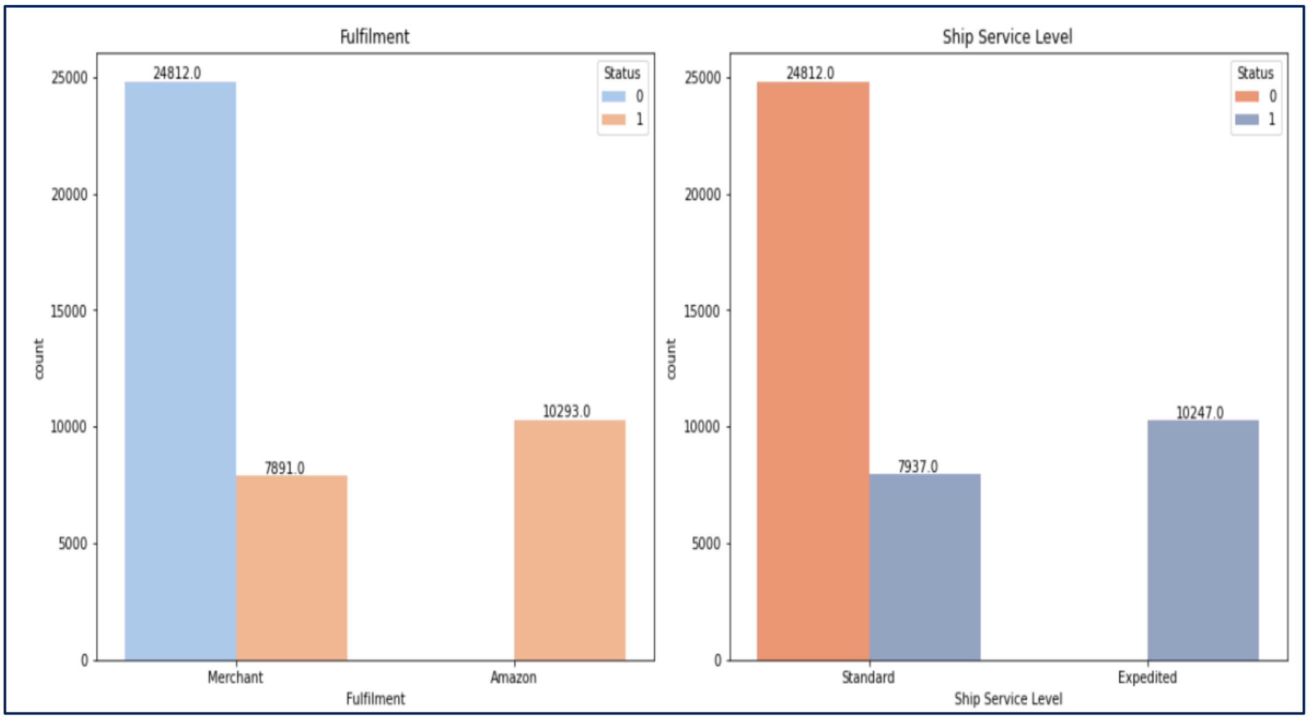
Bi-variate analysis

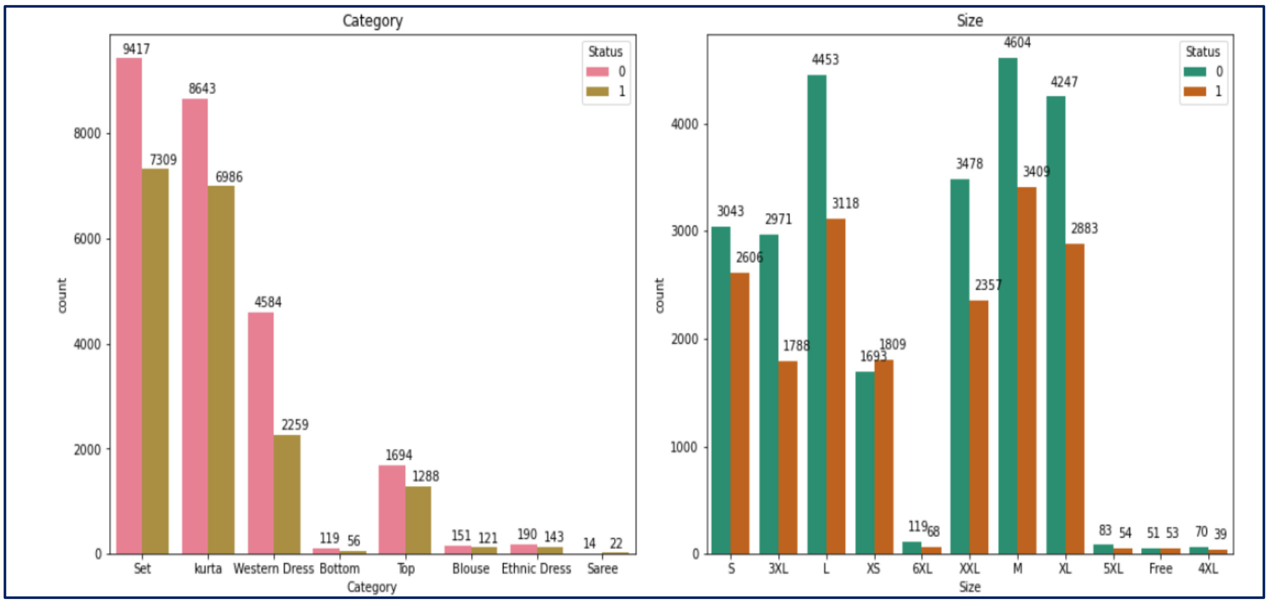
Here we will attempt to understand how the given columns are impacting each other on a one to one basis.

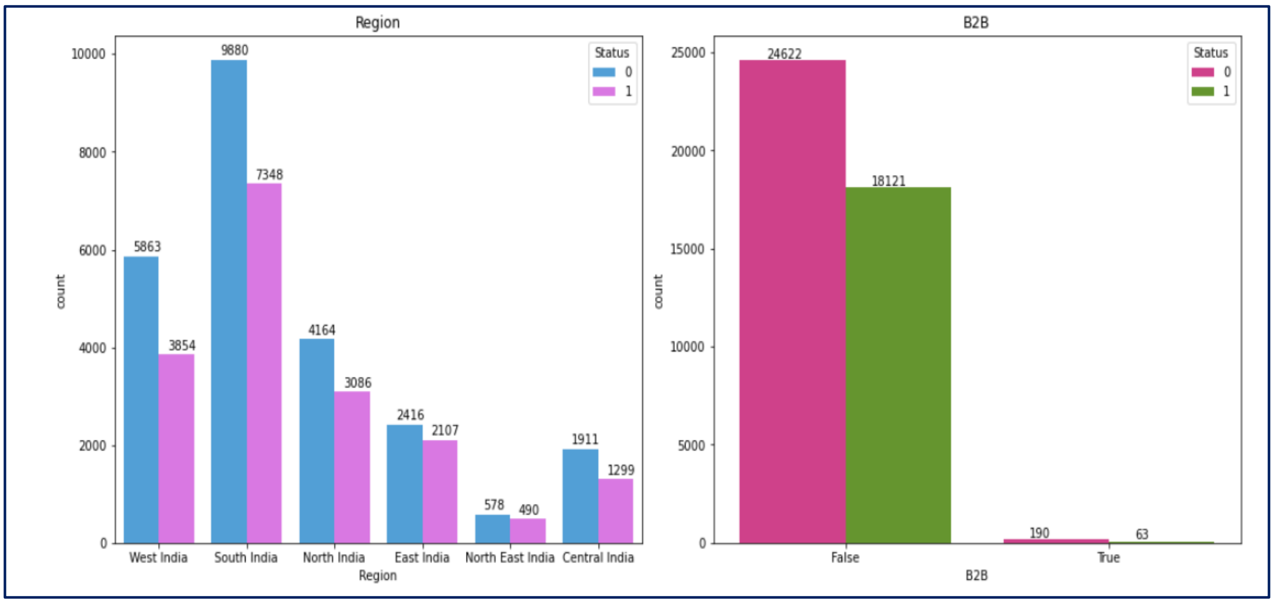
First, we visualise the relationship between our target column (Status) and features:

For quantity, larger proportion orders exist under ‘Not-Rejected’, with ‘Rejected’ having mostly a single unit per order. Hence an initial inference can be made that most rejections have smaller volume per order.

Another important point to note is the almost negligible order rejection rates when orders are under the ‘expedited’ segment and are fulfilled by Amazon. This suggests that the largest pain points to solve when trying to curtail order rejection rates, will be with orders delivered via third-parties.


Fulfilment
Status
0
1
24812.0
7891.0
10293.0
count
Merchant
Amazon
Fulfilment
Ship Service Level
Status
0
1
24812.0
7937.0
10247.0
count
Standard
Expedited
Ship Service Level


Category
Status
0
1
9417
7309
8643
6986
4584
2259
119
56
1694
1288
151
121
190
143
14
22
count
Set
kurta
Western Dress
Bottom
Top
Blouse
Ethnic Dress
Saree
Category
Size
Status
0
1
3043
2606
2971
1788
4453
3118
1693
1809
119
68
3478
2357
4604
3409
4247
2883
83
54
51
53
70
39
count
S
3XL
L
XS
6XL
XXL
M
XL
5XL
Free
4XL
Size

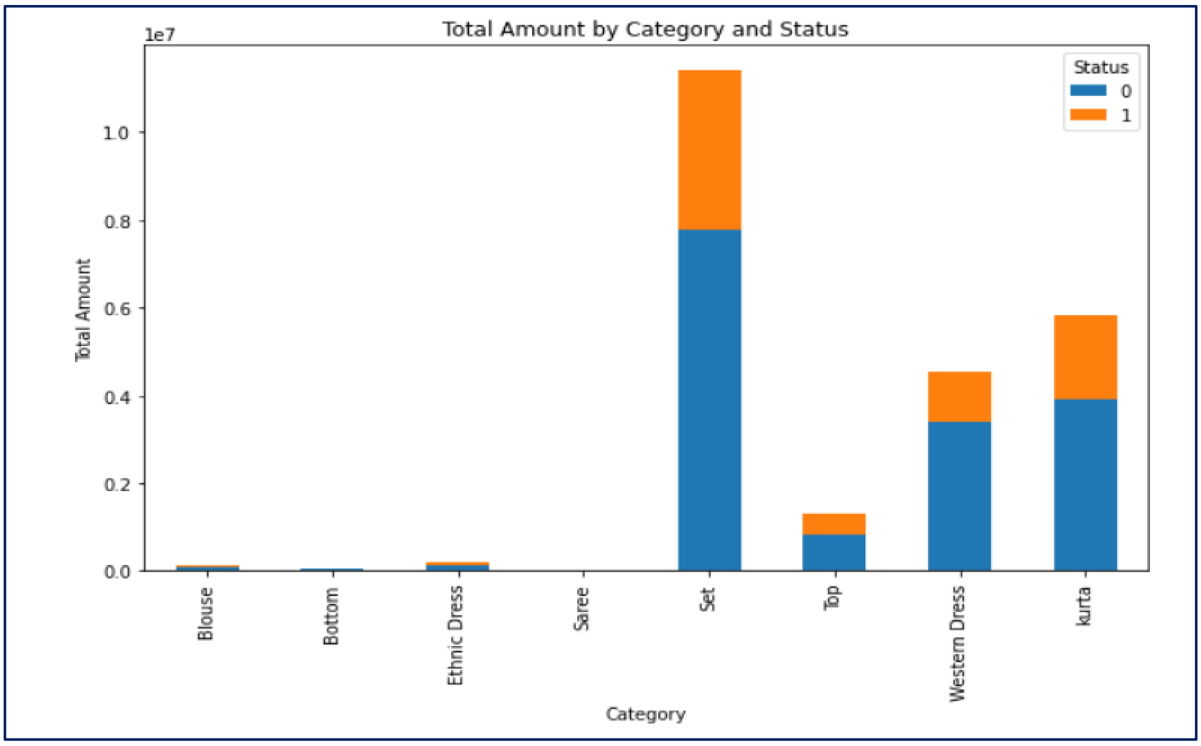


Multi-Variate Analysis

In this section we aim to visualise the relationship between our target variable and features, and amongst the features themselves, more broadly.

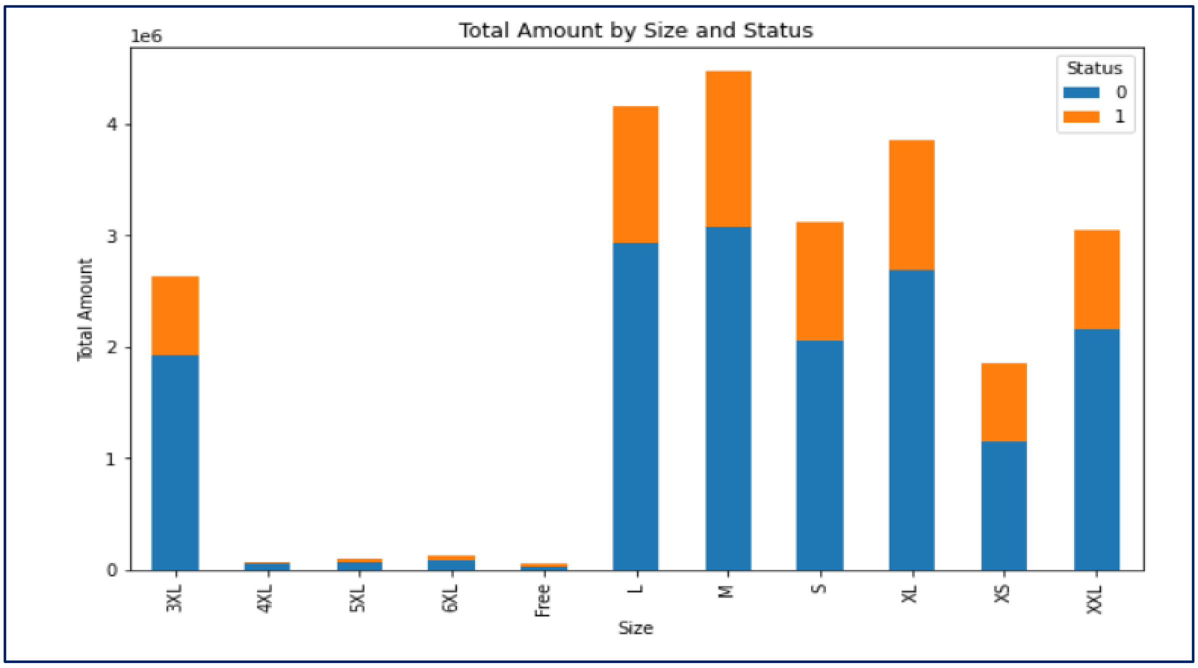
For instance, it might be useful to check how order rejection is affected when distributed by clothing-type/category and the average amount per order for the given clothing categories.

We begin by trying to understand how customer spend ('Amount') differs and whether we could spot any potential trends in rejection rates across various features.

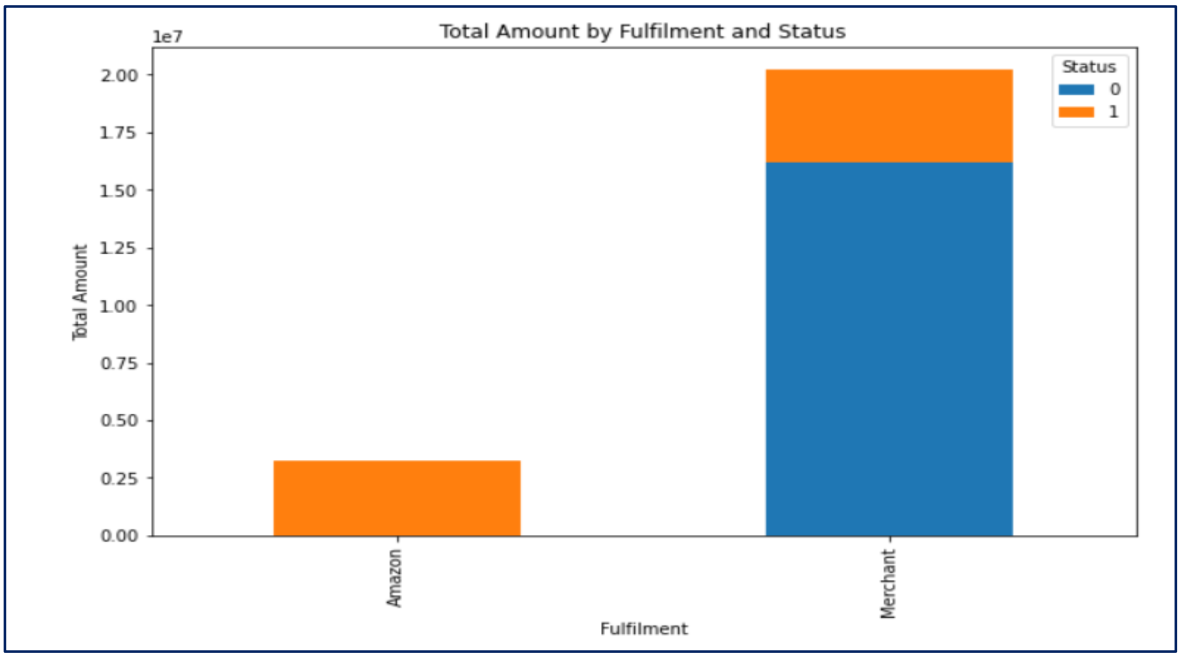


Above we can see that order rejections occur across clothing type/category and no one category has extremely large rejection rates.

Additionally, looking at how rejection rates occur across various clothing sizes, we are unable to pinpoint a single size with particularly large rejection rates.



Finally, a distribution of customer spend across orders delivered via Amazon vs third party merchants tells us that order rejection is primarily an issue with third party sellers, indicating that the pain points might be with the delivery process itself, rather than any specific issues with goods quality or the online amazon website shopping experience of the customer.



Model Building and Additional Data Treatments

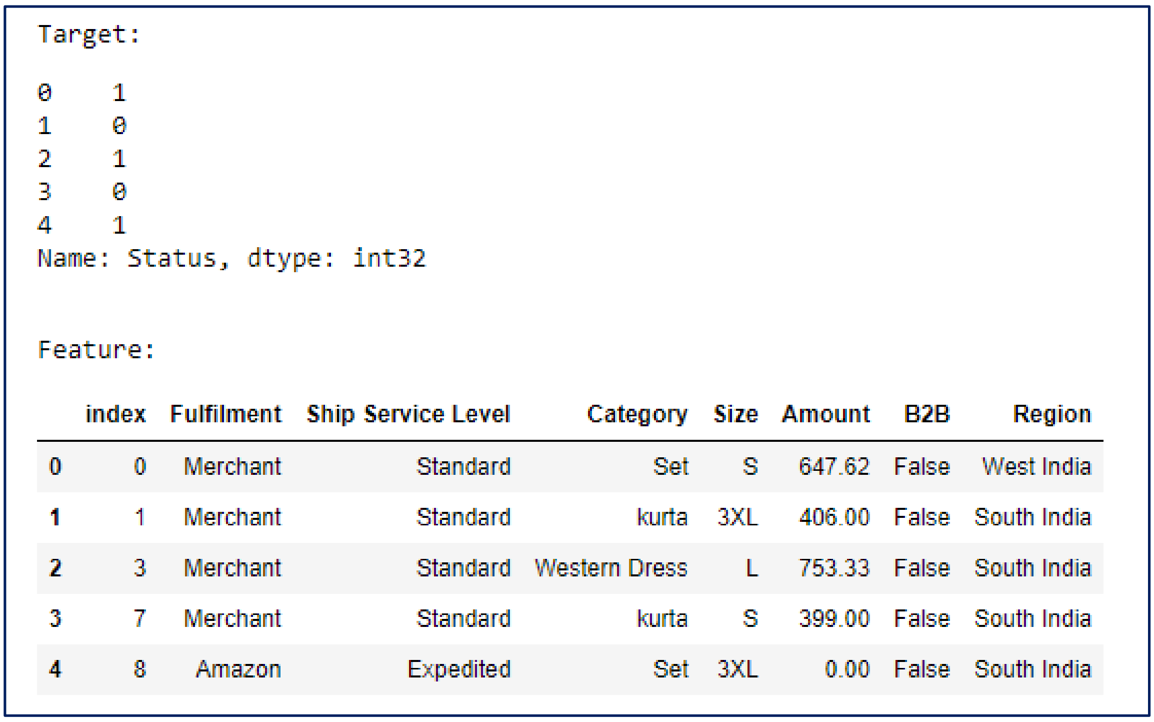
Classification Focussed Predictive Modelling:

As originally mentioned in our objective, our aim with this analysis is to try and understand trends in order rejections. Accordingly, we will build a model that will help identify which of the features are most likely to affect rejections and consequently be useful in predicting rejections in the future.

As a start, we will build a logistic regression based classifier. We shall use popular metrics such as precision and f1-score among other, to interpret the performance of our base model.

For our logistic model, we shall start with reclassifying our target variable ‘Status’ with values 0 (for ‘Not Rejected’) and 1(for ‘Rejected’).

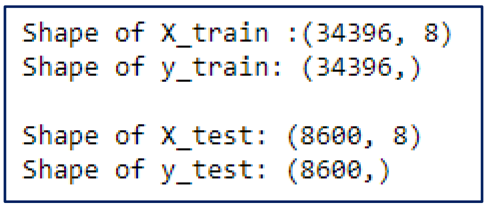
We begin by splitting the dataset into target, representing the ‘Status’ column, and features - representing the independent variables. A snapshot of both has be visualised below.



```
Target:

0    1
1    0
2    1
3    0
4    1
Name: Status, dtype: int32


Feature:
```

	index	Fulfilment	Ship Service Level	Category	Size	Amount	B2B	Region
0	0	Merchant	Standard	Set	S	647.62	False	West India
1	1	Merchant	Standard	kurta	3XL	406.00	False	South India
2	3	Merchant	Standard	Western Dress	L	753.33	False	South India
3	7	Merchant	Standard	kurta	S	399.00	False	South India
4	8	Amazon	Expedited	Set	3XL	0.00	False	South India

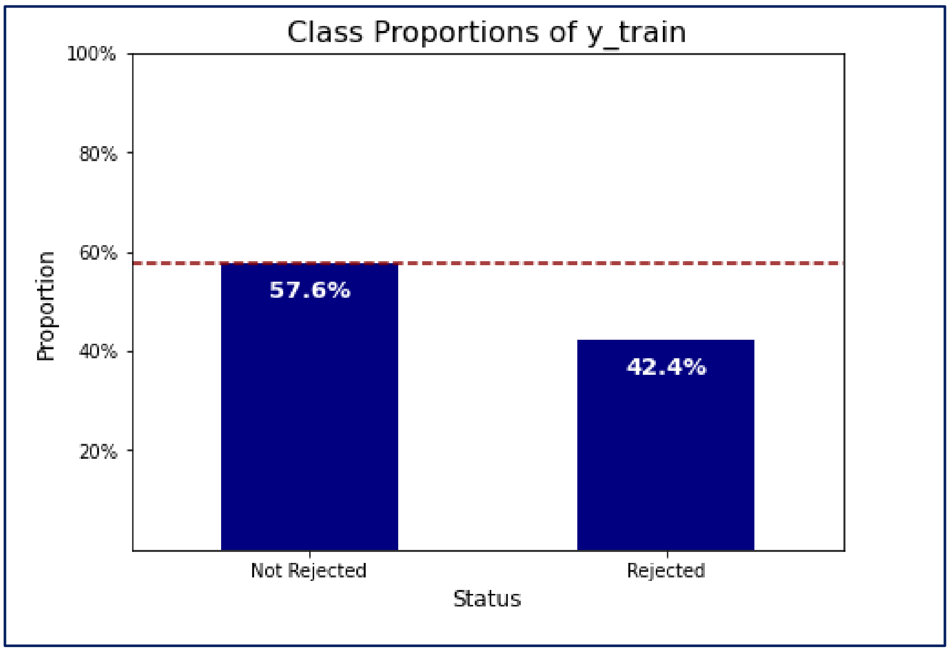
Next, we split the data into train and test segments, using the training segments to train our model and the test segments to assess the efficacy of the model. The resultant division of the data is as shown below, with ‘X_train’ and ‘X_test’ representing the features, and ‘y_train’ and ‘y_test’ representing the target variable.



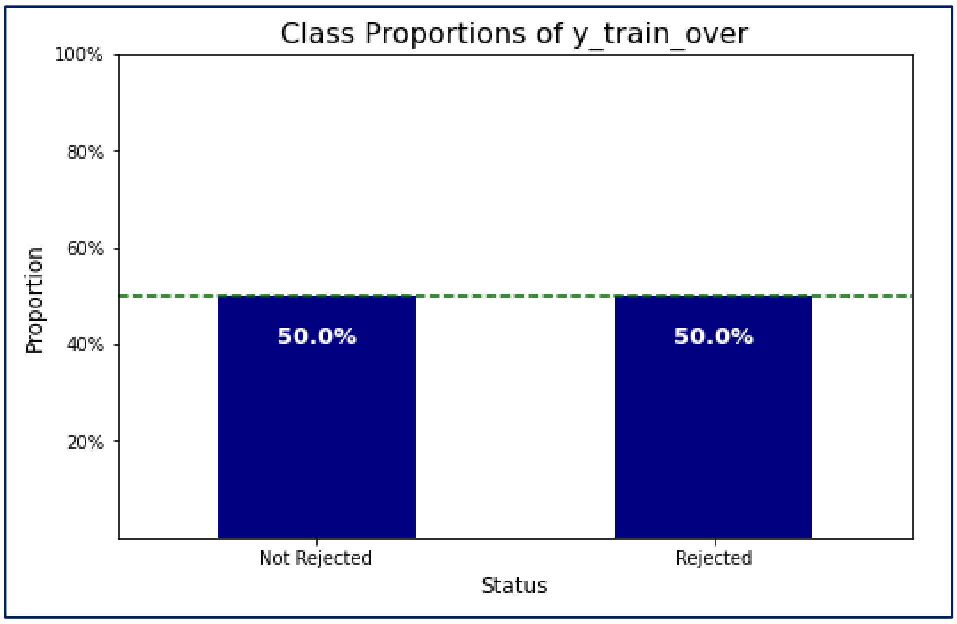
```
Shape of X_train :(34396, 8)
Shape of y_train: (34396,)

Shape of X_test: (8600, 8)
Shape of y_test: (8600,)
```

Target class imbalance considerations:

Below we can see the imbalance between proportions among the target variable values.



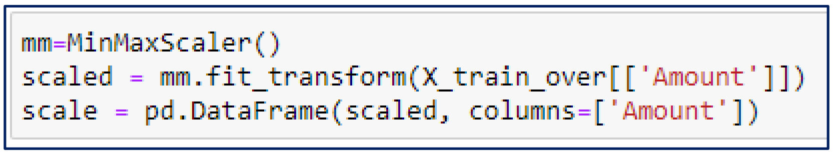
To address this imbalance we shall use the imblearn library. The resultant correction in distribution among the target classes is displayed below. We shall use this rebalanced set to move forward with our analysis.



Feature encoding and scaling:

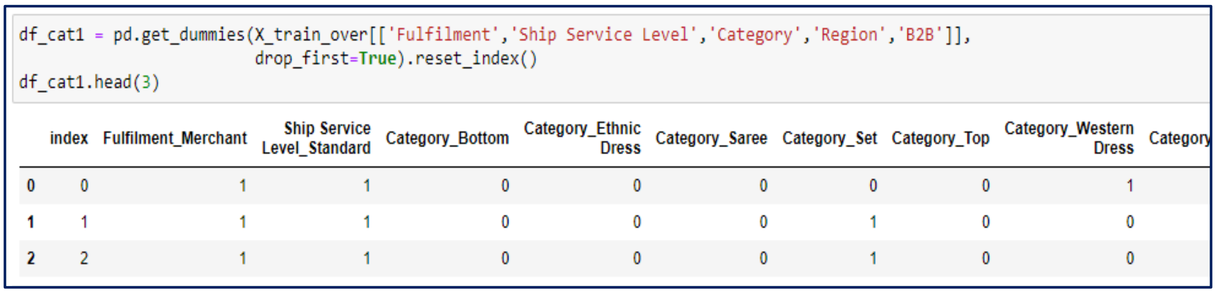
We shall scale the numeric features of our dataset, and encode the categorical features. Numeric features are scaled to bring them into the same base level; and categorical features are encoded so that their numeric representations can be used to perform further analysis and be fed into out machine learning model.

Scaling the numerical columns using the Standard Scalar function



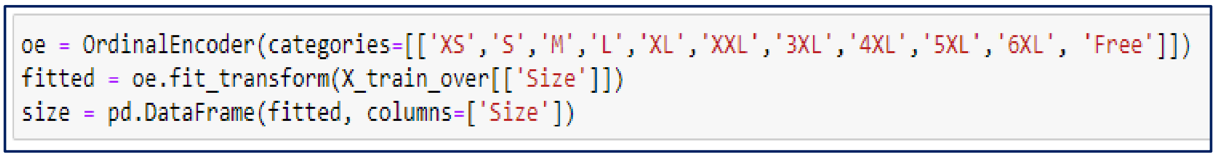
```
mm=MinMaxScaler()
scaled = mm.fit_transform(X_train_over[['Amount']])
scale = pd.DataFrame(scaled, columns=['Amount'])
```

Encoding the categorical columns:



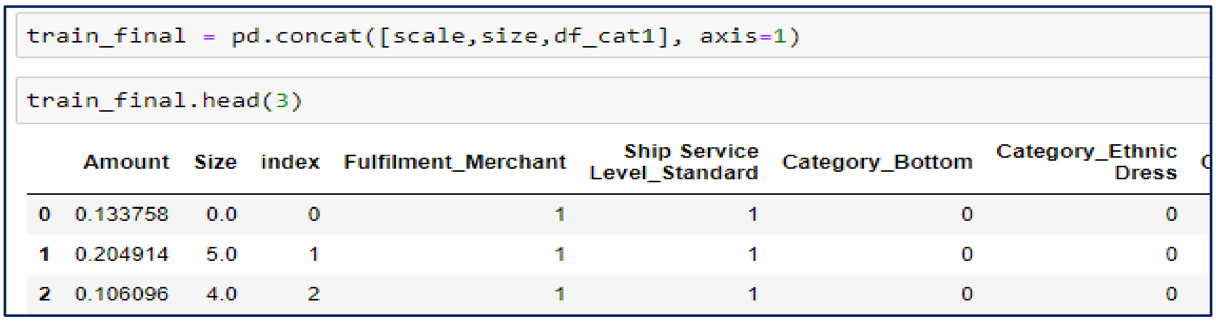
```
df_cat1 = pd.get_dummies(X_train_over[['Fulfilment','Ship Service Level','Category','Region','B2B']],
                         drop_first=True).reset_index()
df_cat1.head(3)
```

	index	Fulfilment_Merchant	Ship Service Level_Standard	Category_Bottom	Category_Ethnic Dress	Category_Saree	Category_Set	Category_Top	Category_Western Dress	Category
0	0	1	1	0	0	0	0	0	1	
1	1	1	1	0	0	0	1	0	0	
2	2	1	1	0	0	0	1	0	0	



```
oe = OrdinalEncoder(categories=[['XS','S','M','L','XL','XXL','3XL','4XL','5XL','6XL', 'Free']])
fitted = oe.fit_transform(X_train_over[['Size']])
size = pd.DataFrame(fitted, columns=['Size'])
```

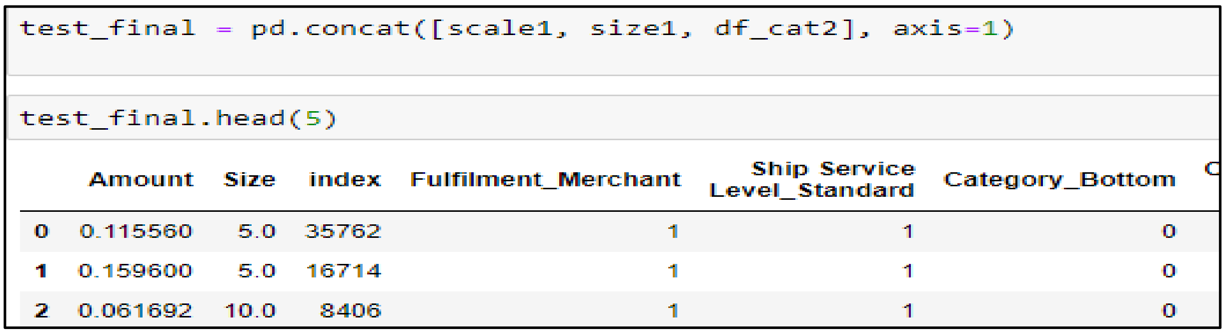
Concatenating the scaled numeric and encoded categorical columns to re-form our dataset. A snapshot of the re-formed train and test sets are displayed below.



```
train_final = pd.concat([scale,size,df_cat1], axis=1)
```

```
train_final.head(3)
```

	Amount	Size	index	Fulfilment_Merchant	Ship Service Level_Standard	Category_Bottom	Category_Ethnic Dress
0	0.133758	0.0	0	1	1	0	0
1	0.204914	5.0	1	1	1	0	0
2	0.106096	4.0	2	1	1	0	0



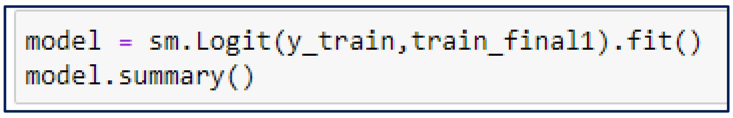
```
test_final = pd.concat([scale1, size1, df_cat2], axis=1)
```

```
test_final.head(5)
```

	Amount	Size	index	Fulfilment_Merchant	Ship Service Level_Standard	Category_Bottom
0	0.115560	5.0	35762	1	1	0
1	0.159600	5.0	16714	1	1	0
2	0.061692	10.0	8406	1	1	0

Given the above datasets for training and testing, we shall now proceed with fitting the data to our initial model of logistic regression.

Feeding the data in to train our model

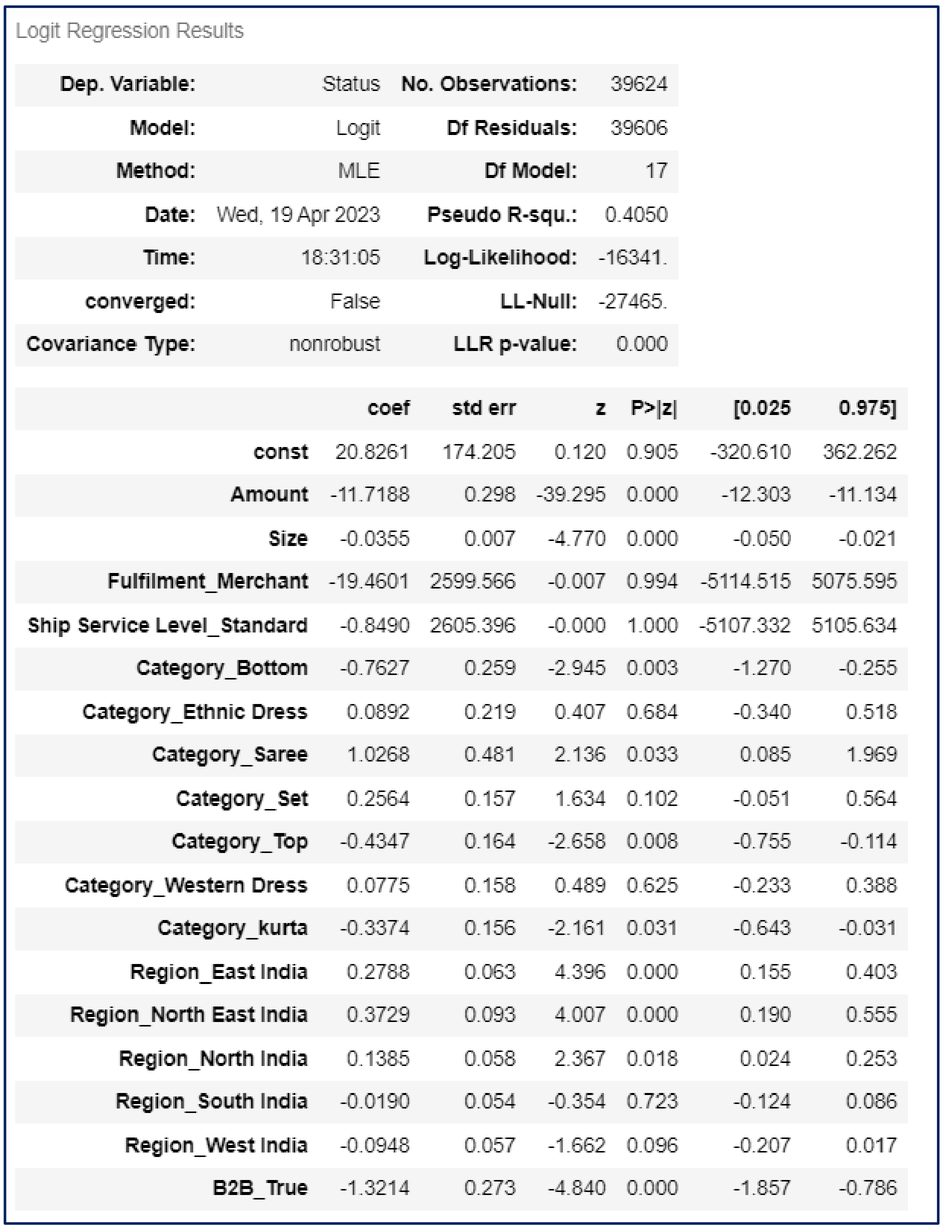


```
model = sm.Logit(y_train,train_final1).fit()
model.summary()
```

Analysis and Scoring:

Now that we have split our model in training and testing segments, and trained our model using the training segment, we shall use various methods to check performance.

First, we use the model.summary () function to output a table of various results from our Logistic Regression Classifier.

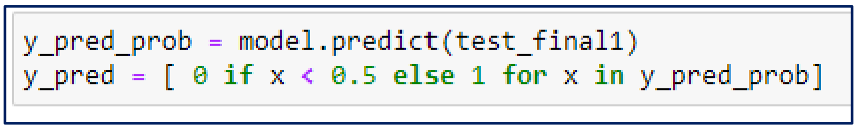


Logit Regression Results

Dep. Variable:	Status	**No. Observations:**	39624
Model:	Logit	**Df Residuals:**	39606
Method:	MLE	**Df Model:**	17
Date:	Wed, 19 Apr 2023	**Pseudo R-squ.:**	0.4050
Time:	18:31:05	**Log-Likelihood:**	-16341.
converged:	False	**LL-Null:**	-27465.
Covariance Type:	nonrobust	**LLR p-value:**	0.000

	coef	std err	z	P>|z|	[0.025	0.975]
const	20.8261	174.205	0.120	0.905	-320.610	362.262
Amount	-11.7188	0.298	-39.295	0.000	-12.303	-11.134
Size	-0.0355	0.007	-4.770	0.000	-0.050	-0.021
Fulfilment_Merchant	-19.4601	2599.566	-0.007	0.994	-5114.515	5075.595
Ship Service Level_Standard	-0.8490	2605.396	-0.000	1.000	-5107.332	5105.634
Category_Bottom	-0.7627	0.259	-2.945	0.003	-1.270	-0.255
Category_Ethnic Dress	0.0892	0.219	0.407	0.684	-0.340	0.518
Category_Saree	1.0268	0.481	2.136	0.033	0.085	1.969
Category_Set	0.2564	0.157	1.634	0.102	-0.051	0.564
Category_Top	-0.4347	0.164	-2.658	0.008	-0.755	-0.114
Category_Western Dress	0.0775	0.158	0.489	0.625	-0.233	0.388
Category_kurta	-0.3374	0.156	-2.161	0.031	-0.643	-0.031
Region_East India	0.2788	0.063	4.396	0.000	0.155	0.403
Region_North East India	0.3729	0.093	4.007	0.000	0.190	0.555
Region_North India	0.1385	0.058	2.367	0.018	0.024	0.253
Region_South India	-0.0190	0.054	-0.354	0.723	-0.124	0.086
Region_West India	-0.0948	0.057	-1.662	0.096	-0.207	0.017
B2B_True	-1.3214	0.273	-4.840	0.000	-1.857	-0.786

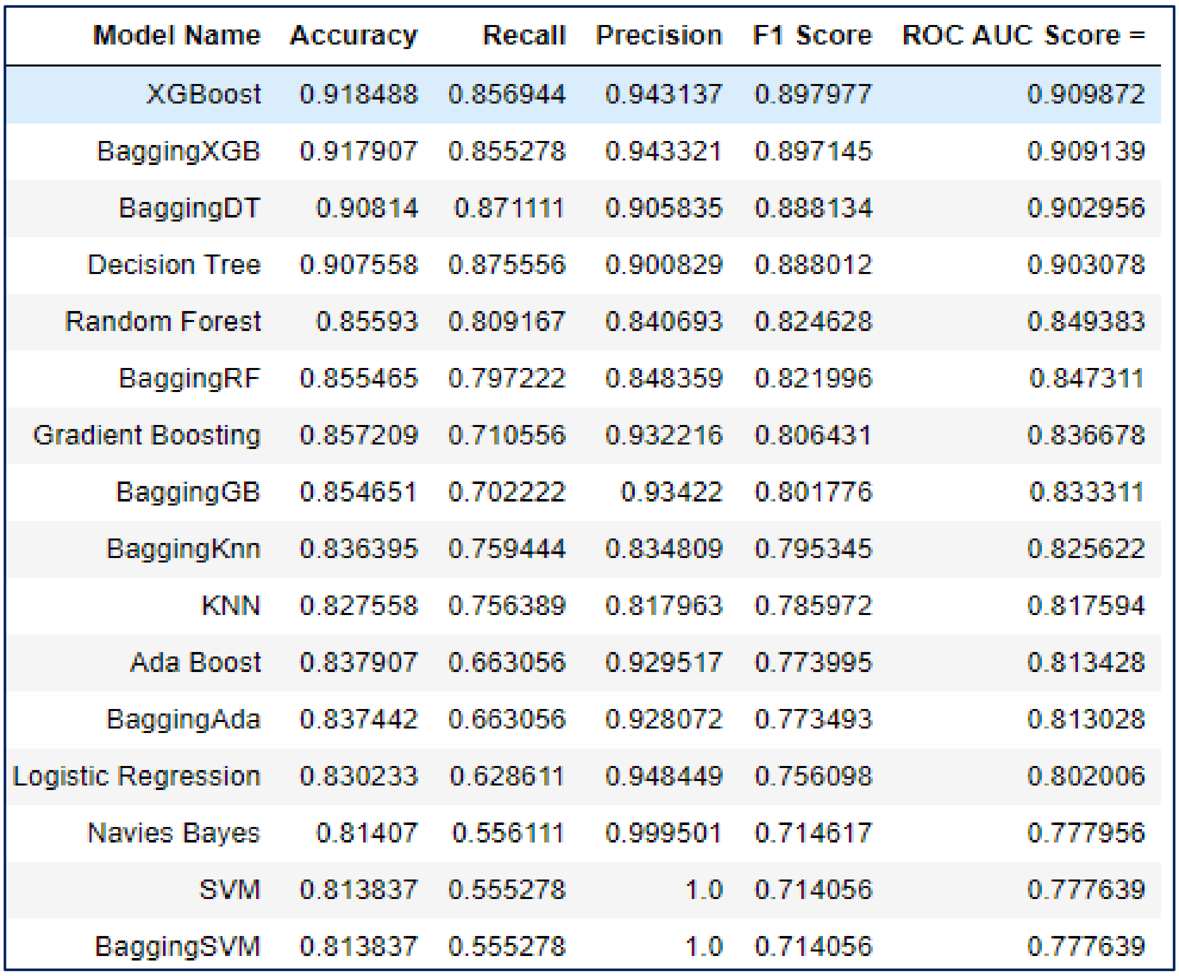
Next, given the classification nature of this model, we shall use a standard cut-off of 0.5 to put predictions into the ‘Rejected’/’Not Rejected’ buckets



```
y_pred_prob = model.predict(test_final1)
y_pred = [ 0 if x < 0.5 else 1 for x in y_pred_prob]
```

While the model performance above gives us a good starting point, there is certainly scope for improvement, and hence we shall now fit and score a broad range of models to our updated data set, and display the performance for each below.

First let us look at the performance of all the models employed, across a range of scores.



Model Name	Accuracy	Recall	Precision	F1 Score	ROC AUC Score =
XGBoost	0.918488	0.856944	0.943137	0.897977	0.909872
BaggingXGB	0.917907	0.855278	0.943321	0.897145	0.909139
BaggingDT	0.90814	0.871111	0.905835	0.888134	0.902956
Decision Tree	0.907558	0.875556	0.900829	0.888012	0.903078
Random Forest	0.85593	0.809167	0.840693	0.824628	0.849383
BaggingRF	0.855465	0.797222	0.848359	0.821996	0.847311
Gradient Boosting	0.857209	0.710556	0.932216	0.806431	0.836678
BaggingGB	0.854651	0.702222	0.93422	0.801776	0.833311
BaggingKnn	0.836395	0.759444	0.834809	0.795345	0.825622
KNN	0.827558	0.756389	0.817963	0.785972	0.817594
Ada Boost	0.837907	0.663056	0.929517	0.773995	0.813428
BaggingAda	0.837442	0.663056	0.928072	0.773493	0.813028
Logistic Regression	0.830233	0.628611	0.948449	0.756098	0.802006
Navies Bayes	0.81407	0.556111	0.999501	0.714617	0.777956
SVM	0.813837	0.555278	1.0	0.714056	0.777639
BaggingSVM	0.813837	0.555278	1.0	0.714056	0.777639

Let us now individually display the performance of the top 10 models to dive deeper into each model. Specifically we shall visualise the classification reports, confusion matrix and ROC curve for each of the models.

Classification Models and Scoring

Decision Tree:

A decision tree is a tree-based supervised learning method used to predict the output of a target variable. Supervised learning uses labelled data (data with known output variables) to make predictions with the help of regression and classification algorithms. Supervised learning algorithms act as a supervisor for training a model with a defined output variable. It learns from simple decision rules using the various data features. Decision trees in Python can be used to solve both classification and regression problems—they are frequently used in determining odds

Advantages of Using Decision Trees:

- Decision trees are simple to understand, interpret, and visualize
- They can effectively handle both numerical and categorical data
- They can determine the worst, best, and expected values for several scenarios
- Decision trees require little data preparation and data normalization
- They perform well, even if the actual model violates the assumptions

Decision Tree Applications:

- A decision tree is used to determine whether an applicant is likely to default on a loan.
- It can be used to determine the odds of an individual developing a specific disease.
- It can help ecommerce companies in predicting whether a consumer is likely to purchase a specific product.
- Decision trees can also be used to find customer churn rates.

Ensemble Methods:

The ensemble methods in machine learning combine the insights obtained from multiple learning models to facilitate accurate and improved decisions.

In learning models, noise, variance, and bias are the major sources of error. The ensemble methods in machine learning help minimize these error-causing factors, thereby ensuring the accuracy and stability of machine learning (ML) algorithms.

Ensemble methods in machine learning employ a set of models and take advantage of the blended output, which, compared to a solitary model, will most certainly be a superior option when it comes to prediction accuracy.

Bagging (Bootstrap Aggregating):

The primary goal of "bagging" or "bootstrap aggregating" ensemble method is to minimize variance errors in decision trees. The objective here is to randomly create samples of training datasets with replacement (subsets of the training data). The subsets are then used for training decision trees or models. Consequently, there is a combination of multiple models, which reduces variance, as the average prediction generated from different models is much more reliable and robust than a single model or a decision tree.

Boosting:

An iterative ensemble technique, "boosting," adjusts an observation's weight based on its last classification. In case observation is incorrectly classified, "boosting" increases the observation's weight, and vice versa. Boosting algorithms reduce bias errors and produce superior predictive models.

In the boosting ensemble method, data scientists train the first boosting algorithm on an entire dataset and then build subsequent algorithms by fitting residuals from the first boosting algorithm, thereby giving more weight to observations that the previous model predicted inaccurately.

Random Forest:

Random Forest is an ensemble machine learning technique capable of performing both regression and classification tasks using multiple decision trees and a statistical technique called bagging. Bagging along with boosting are two of the most popular ensemble techniques which aim to tackle high variance and high bias. A RF instead of just averaging the prediction of trees it uses two key concepts that give it the name random.

Random sampling of training observations:

Each tree in a random forest learns from a random sample of the training observations. The samples are drawn with replacement, known as bootstrapping, which means that some samples will be used multiple times in a single tree. The idea is that by training each tree on different samples, although each tree might have high variance with respect to a particular set of the training data, overall, the entire forest will have lower variance but not at the cost of increasing the bias. In Sklearn implementation of Random forest the sub-sample size of each tree is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True. If bootstrap=False each tree will use exactly the same dataset without any randomness.

Random Subsets of features for splitting nodes:

The other main concept in the random forest is that each tree sees only a subset of all the features when deciding to split a node. In Skearn this can be set by specifying max_features = sqrt(n_features) meaning that if there are 16 features, at each node in each tree, only 4 random features will be considered for splitting the node.

Why a Random Forest is better than a single decision tree?

The fundamental idea behind a random forest is to combine the predictions made by many decision trees into a single model. Individually, predictions made by decision trees may not be accurate but combined together, the predictions will be closer to the true value on average.

The objective of a machine learning model is to generalize well to new data it has never seen before. Over fitting occurs when a very flexible model (high capacity) memorizes the training data by fitting it closely. The problem is that the model learns not only the actual relationships in the training data but also any noise that is present. A flexible model is said to have high variance because the learned parameters (such as the structure of the decision tree) will vary considerably with the training data.

On the other hand, an inflexible model is said to have high bias because it makes assumptions about the training data (it's biased towards pre-conceived ideas of the data). An inflexible model may not have the capacity to fit even the training data and in both cases—high variance and high bias—the model is not able to generalize well to new data.

Bias – Variance Trade Off

The balance between creating a model that is so flexible it memorizes the training data versus an inflexible model that can't learn the training data is known as the bias-variance trade off and is a foundational concept in machine learning.

As I stated earlier decision tree is prone to overfitting as it can keep growing until it has exactly one leaf node for every single observation. If the maximum depth is set to 2 (making only a single split), the predictions are no longer 100% correct. We have reduced the variance of the decision tree but at the cost of increasing the bias.

That said a small change in the initial parameters of a decision tree can cause the model prediction to vary a lot, which qualifies it as an unstable model. That is the reason we apply Bagging on unstable models like Decision Tree to reduces variance (good) and increases bias (bad) by combining many trees into a single ensemble model known as the random forest.

Gradient Boosting:

GBM is a framework that provides an implementation of gradient boosted decision trees. It's created by the researchers and developers' team at Microsoft. Light GBM is known for its faster training speed, good accuracy with default parameters, parallel, and GPU learning, low memory footprint, and capability of handling large dataset.

GB Classifier on the test problem using repeated k-fold cross-validation and reports the mean accuracy. Then a single model is fit on all available data and a single prediction is made. Results may vary given the stochastic nature of the algorithm or evaluation procedure, or differences in numerical precision. Consider running the example a few times and compare the average outcome.

Running the example first reports the evaluation of the model using repeated k-fold cross-validation, then the result of making a single prediction with a model fit on the entire dataset.

Logistic Regression:

It is a classification algorithm in machine learning that uses one or more independent variables to determine an outcome. The outcome is measured with a dichotomous variable meaning it will have only two possible outcomes.

The goal of logistic regression is to find a best-fitting relationship between the dependent variable and a set of independent variables. It is better than other binary classification algorithms like nearest neighbour since it quantitatively explains the factors leading to classification.

Logistic regression is named for the function used at the core of the method, the logistic function. The logistic function, also called the sigmoid function was developed by statisticians to describe properties of population growth in ecology, rising quickly and maxing out at the carrying capacity of the environment. It's an S-shaped curve that can take any real-valued number and map it into a value between 0 and 1, but never exactly at those limits.

1 / (1 + e^-value), Where 'e' is the base of the natural logarithms (Euler's number or the EXP () function in your spreadsheet) and value is the actual numerical value that you want to transform. Below is a plot of the numbers between -5 and 5 transformed into the range 0 and 1 using the logistic function.

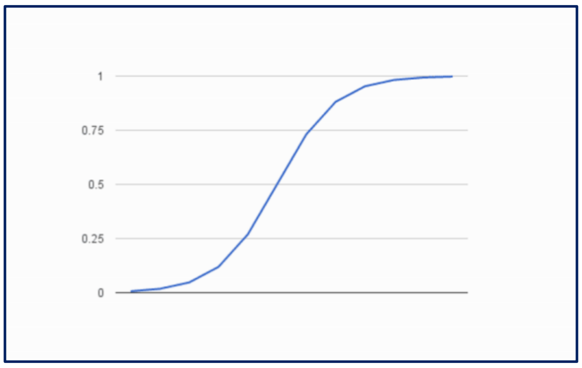


Fig. Logistic Function.

Naïve Bayes Classifier:

A Naive Bayes classifier is a probabilistic machine learning model that's used for classification task. The crux of the classifier is based on the Bayes theorem.

Bayes Theorem:

Using Bayes theorem, we can find the probability of A happening, given that B has occurred. Here, B is the evidence and A is the hypothesis. The assumption made here is that the predictors/features are independent. That is presence of one particular feature does not affect the other. Hence it is called naive.

Gaussian Naive Bayes:

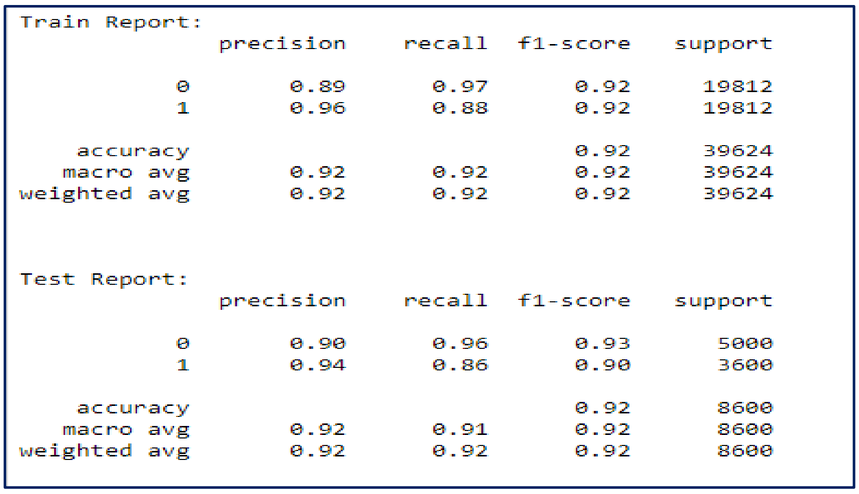
When the predictors take up a continuous value and are not discrete, we assume that these values are sampled from a gaussian distribution. Since the way the values are present in the dataset changes, the formula for conditional probability changes to,

Types of Learners in Classification:

Lazy Learners – Lazy learners simply store the training data and wait until a testing data appears. The classification is done using the most related data in the stored training data. They have more predicting time compared to eager learners. Ex. K-nearest neighbour, case-based reasoning.

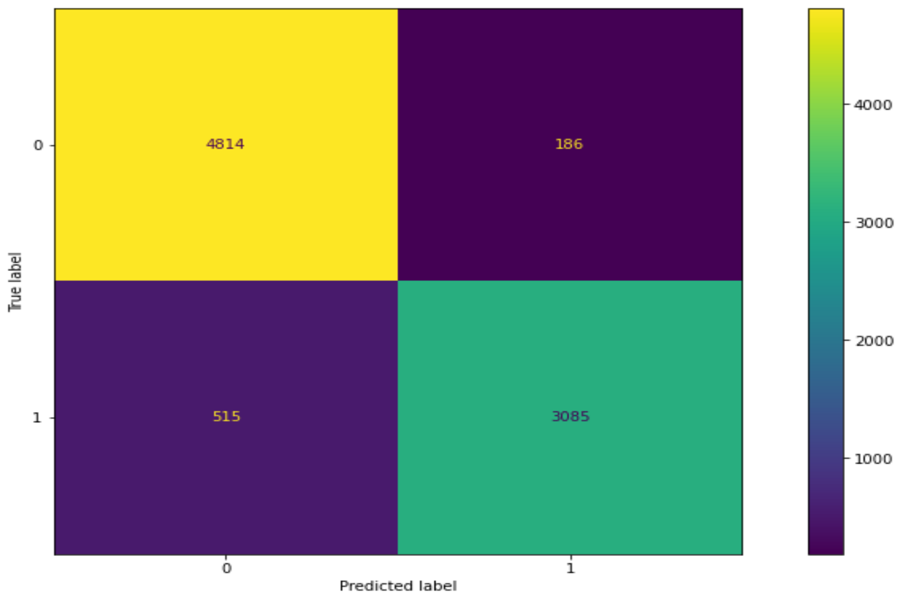
Eager Learners – Eager learners construct a classification model based on the given training data before getting data for predictions. It must be able to commit to a single hypothesis that will work for the entire space. Due to this, they take a lot of time in training and less time for a prediction. Ex. Decision Tree, Naive Bayes, Artificial Neural Networks.

XGBoost Classifier:

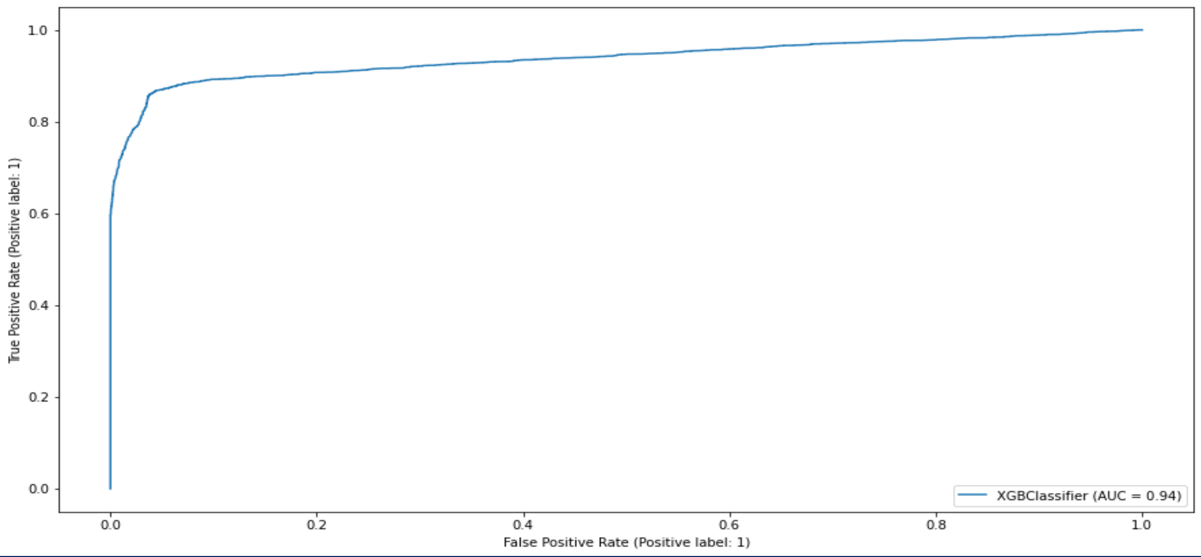


```
Train Report:
              precision    recall  f1-score   support

           0       0.89      0.97      0.92     19812
           1       0.96      0.88      0.92     19812

    accuracy                           0.92     39624
   macro avg       0.92      0.92      0.92     39624
weighted avg       0.92      0.92      0.92     39624


Test Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      5000
           1       0.94      0.86      0.90      3600

    accuracy                           0.92      8600
   macro avg       0.92      0.91      0.92      8600
weighted avg       0.92      0.92      0.92      8600
```

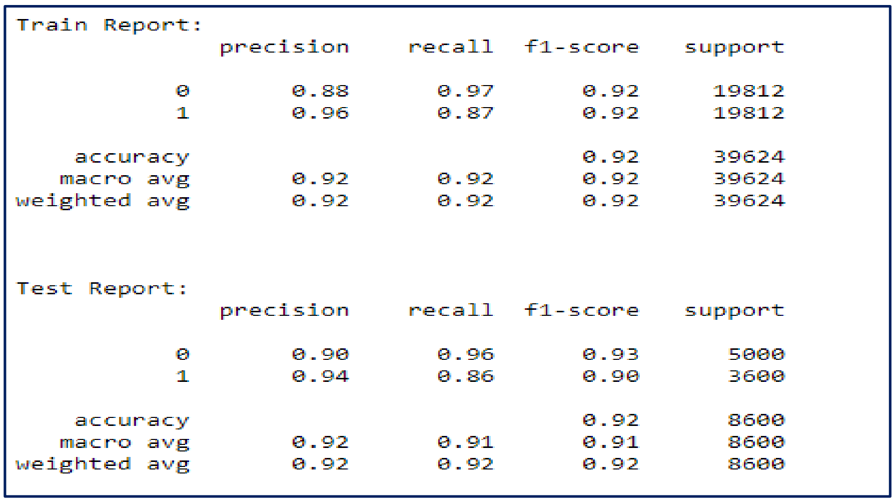
Confusion Matrix:



ROC Curve:

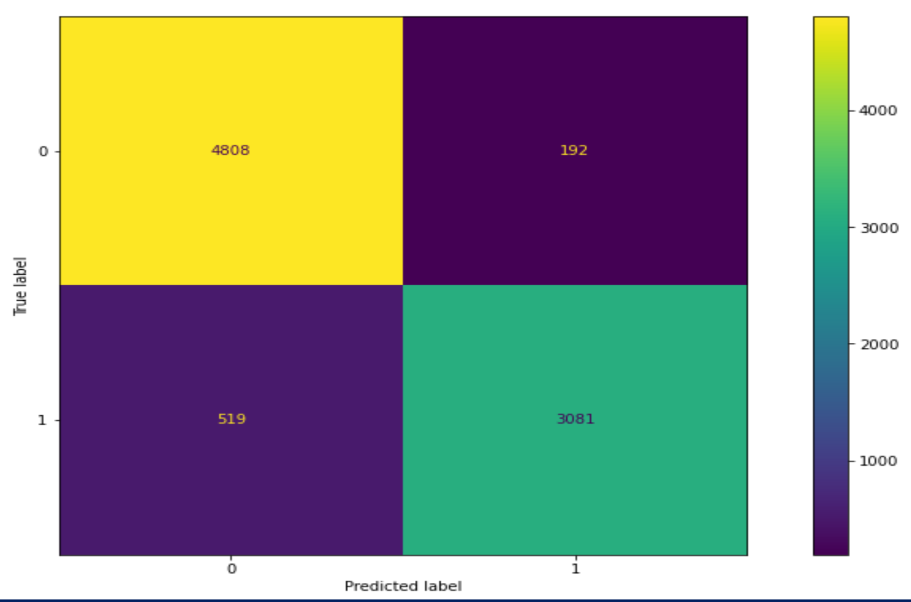


Bagging Classifier (with base_estimator – XGBoost Classifier):

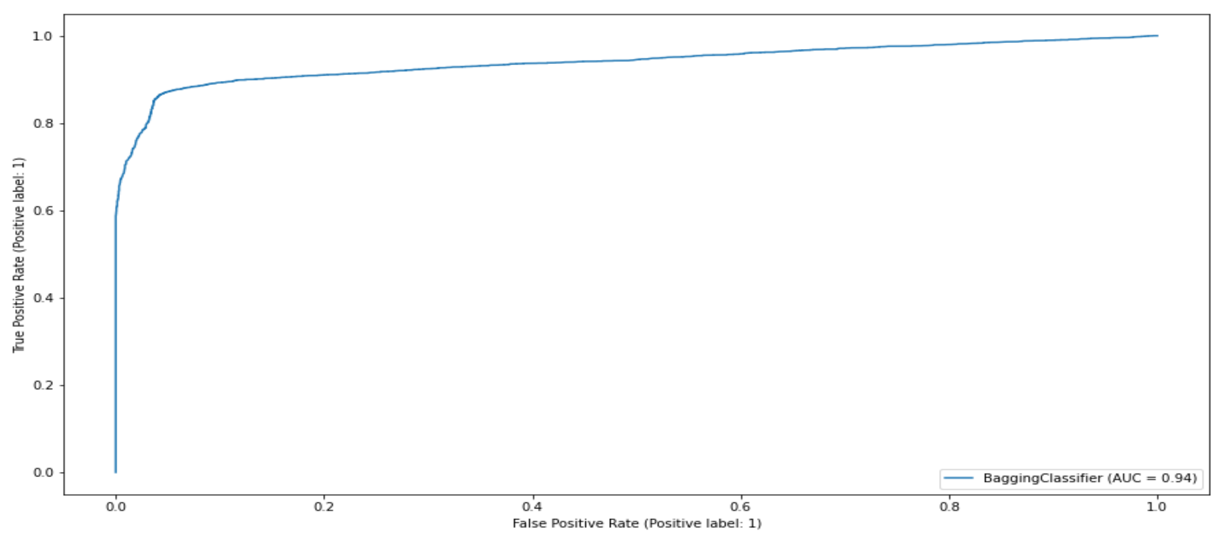


```
Train Report:
              precision    recall  f1-score   support

           0       0.88      0.97      0.92     19812
           1       0.96      0.87      0.92     19812

    accuracy                           0.92     39624
   macro avg       0.92      0.92      0.92     39624
weighted avg       0.92      0.92      0.92     39624


Test Report:
              precision    recall  f1-score   support

           0       0.90      0.96      0.93      5000
           1       0.94      0.86      0.90      3600

    accuracy                           0.92      8600
   macro avg       0.92      0.91      0.91      8600
weighted avg       0.92      0.92      0.92      8600
```

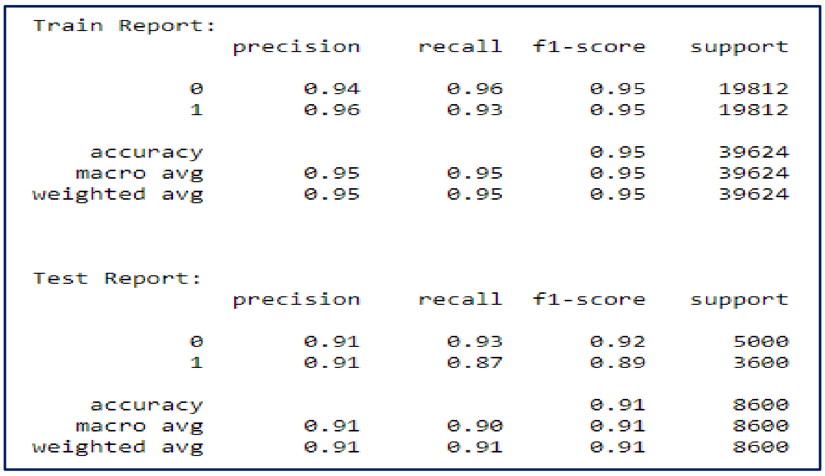
Confusion Matrix:



ROC Curve:

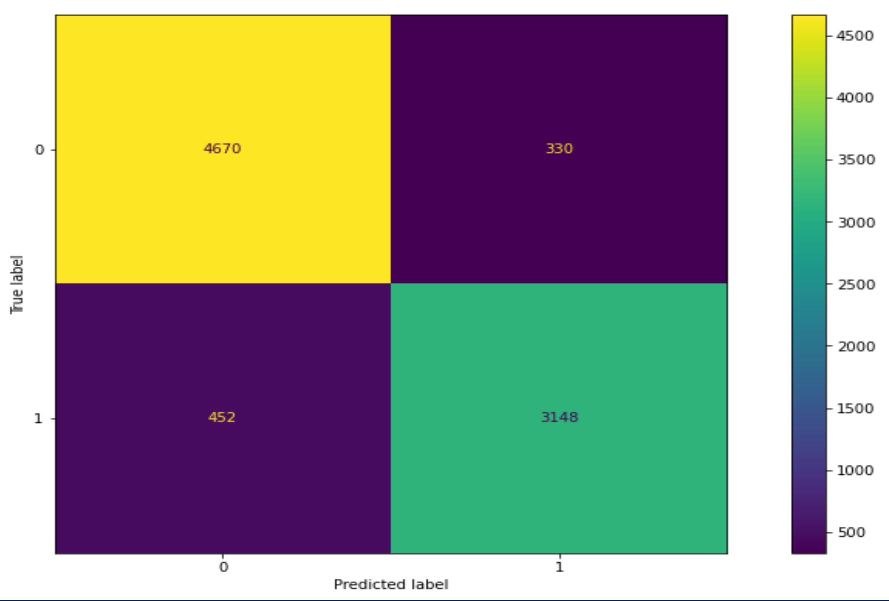


BaggingClassifier (base_estimator-DecisionTreeClassifier()):

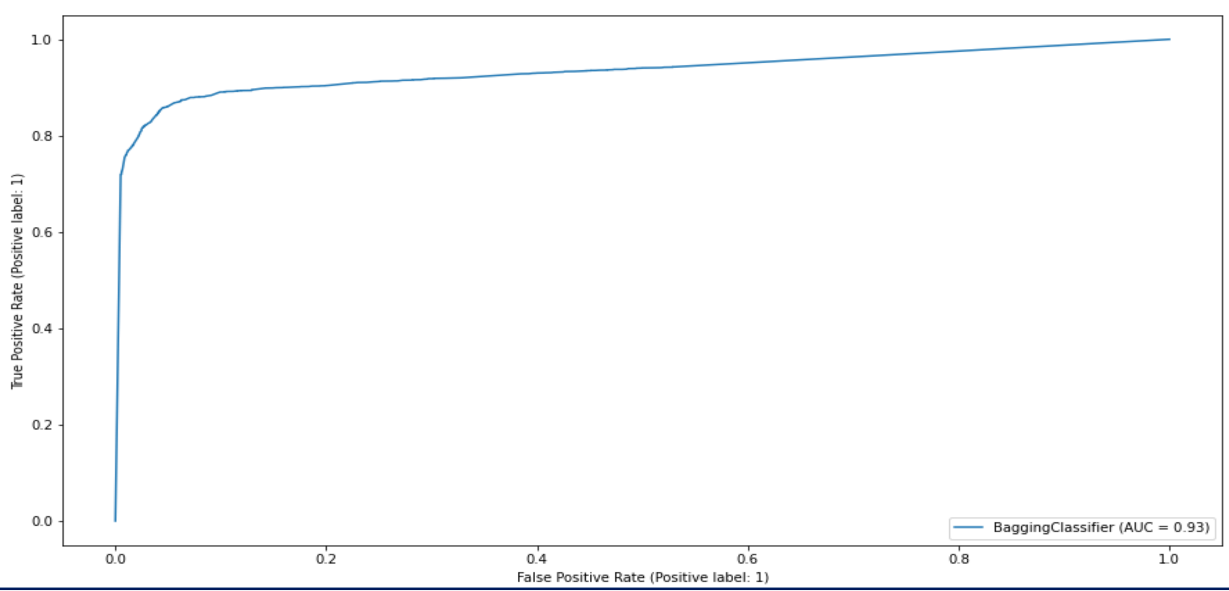


```
 Train Report:
               precision    recall  f1-score   support

           0       0.94      0.96      0.95     19812
           1       0.96      0.93      0.95     19812

    accuracy                           0.95     39624
   macro avg       0.95      0.95      0.95     39624
weighted avg       0.95      0.95      0.95     39624


 Test Report:
               precision    recall  f1-score   support

           0       0.91      0.93      0.92      5000
           1       0.91      0.87      0.89      3600

    accuracy                           0.91      8600
   macro avg       0.91      0.90      0.91      8600
weighted avg       0.91      0.91      0.91      8600
```

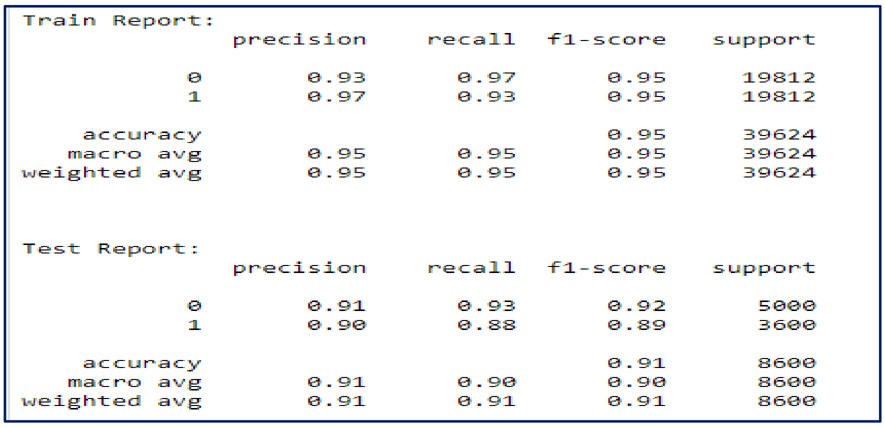
Confusion Matrix:



ROC Curve:

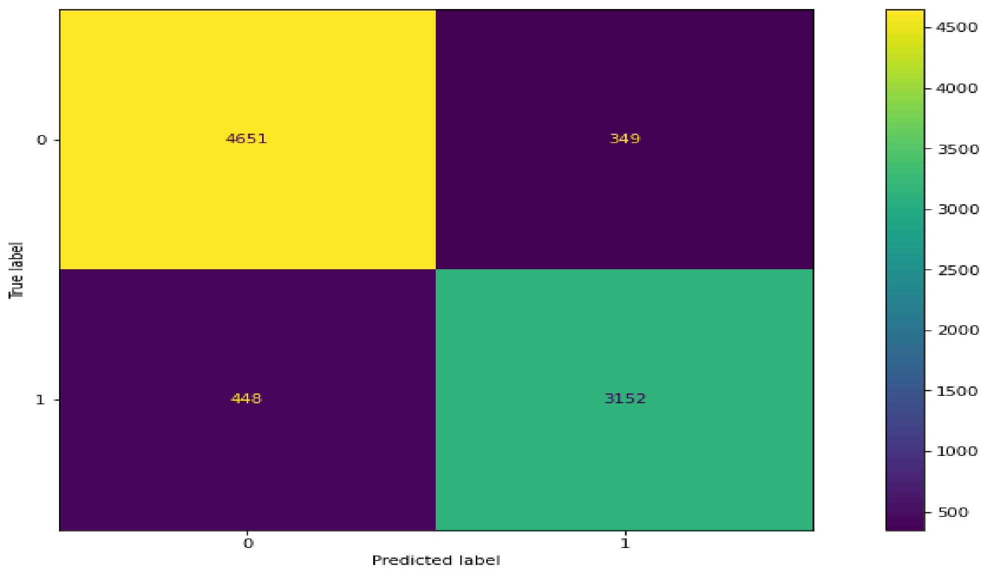


Decision Tree Classifier:

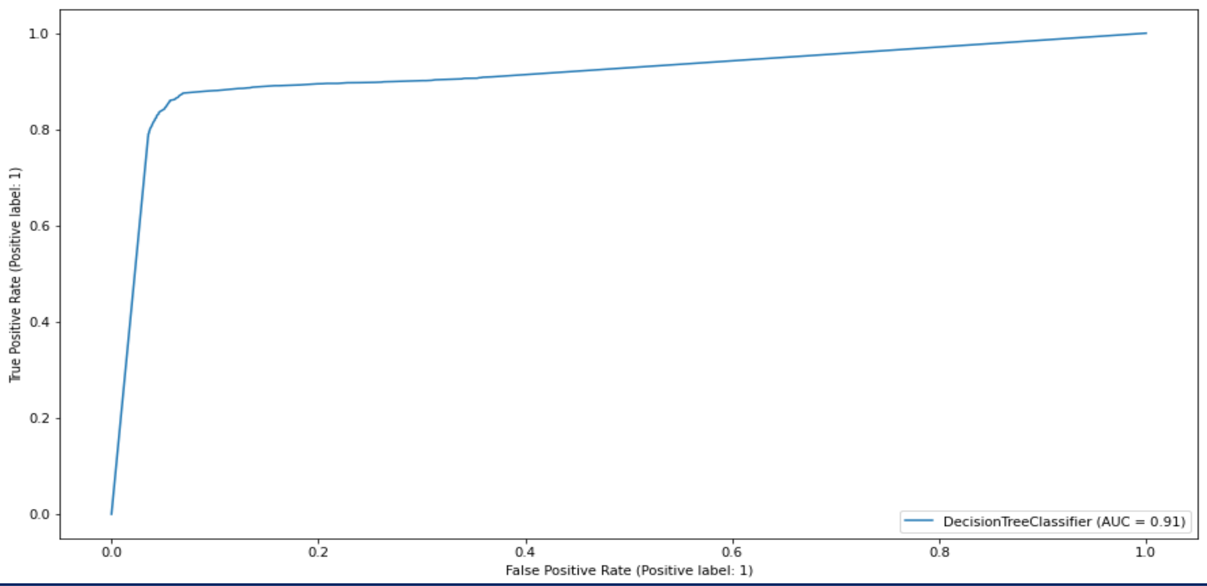


```
Train Report:
              precision    recall  f1-score   support

           0       0.93      0.97      0.95     19812
           1       0.97      0.93      0.95     19812

    accuracy                           0.95     39624
   macro avg       0.95      0.95      0.95     39624
weighted avg       0.95      0.95      0.95     39624


Test Report:
              precision    recall  f1-score   support

           0       0.91      0.93      0.92      5000
           1       0.90      0.88      0.89      3600

    accuracy                           0.91      8600
   macro avg       0.91      0.90      0.90      8600
weighted avg       0.91      0.91      0.91      8600
```

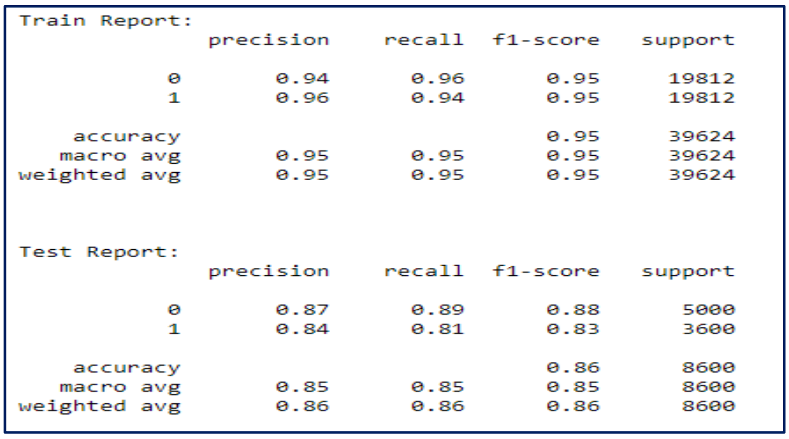
Confusion Matrix:



ROC Curve:

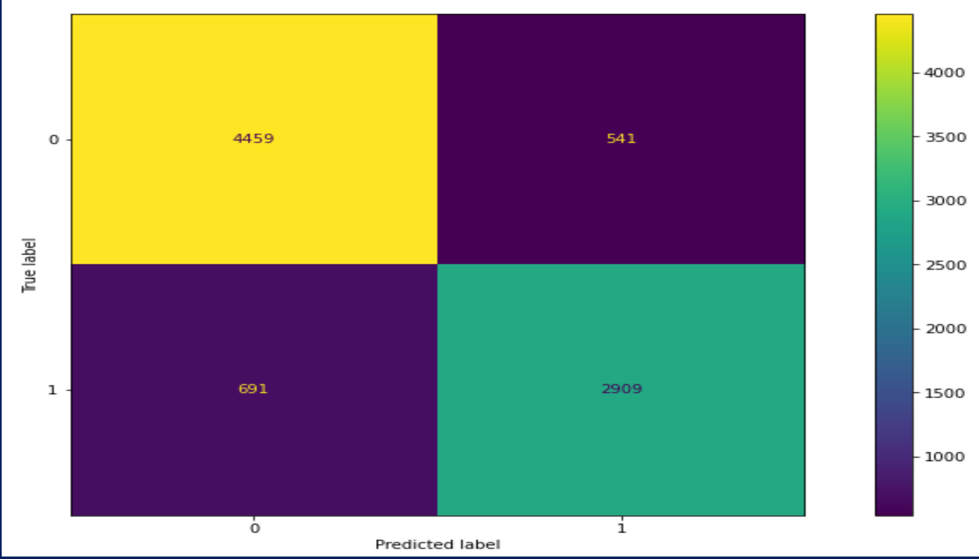


Random Forrest Classifier:

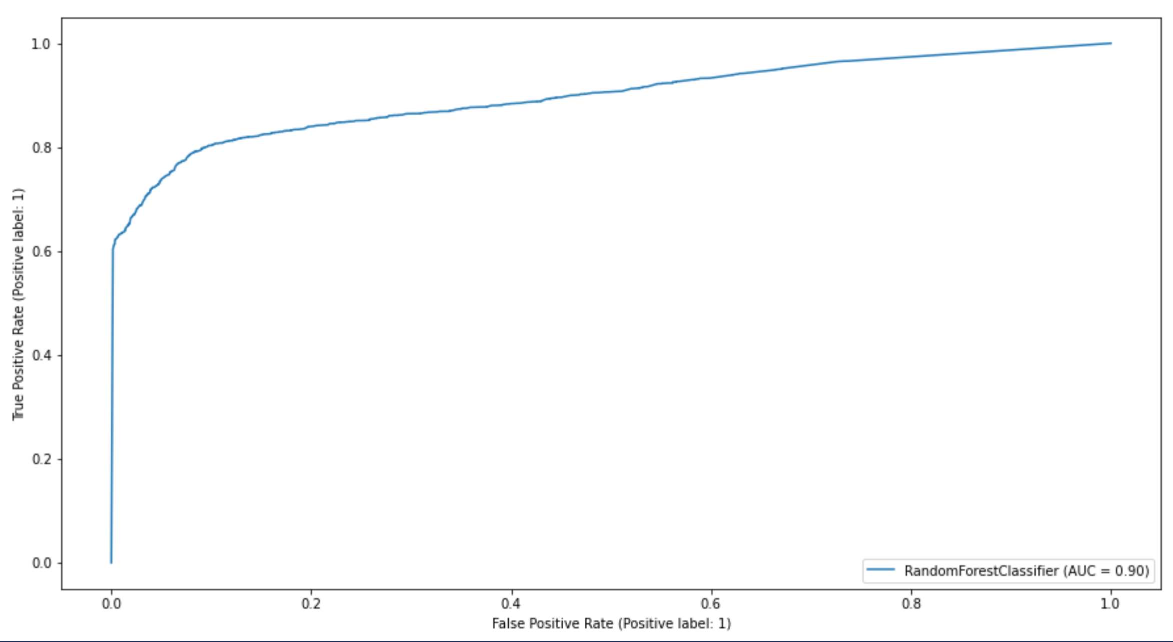


```
Train Report:
              precision    recall  f1-score   support

           0       0.94      0.96      0.95     19812
           1       0.96      0.94      0.95     19812

    accuracy                           0.95     39624
   macro avg       0.95      0.95      0.95     39624
weighted avg       0.95      0.95      0.95     39624


Test Report:
              precision    recall  f1-score   support

           0       0.87      0.89      0.88      5000
           1       0.84      0.81      0.83      3600

    accuracy                           0.86      8600
   macro avg       0.85      0.85      0.86      8600
weighted avg       0.86      0.86      0.86      8600
```

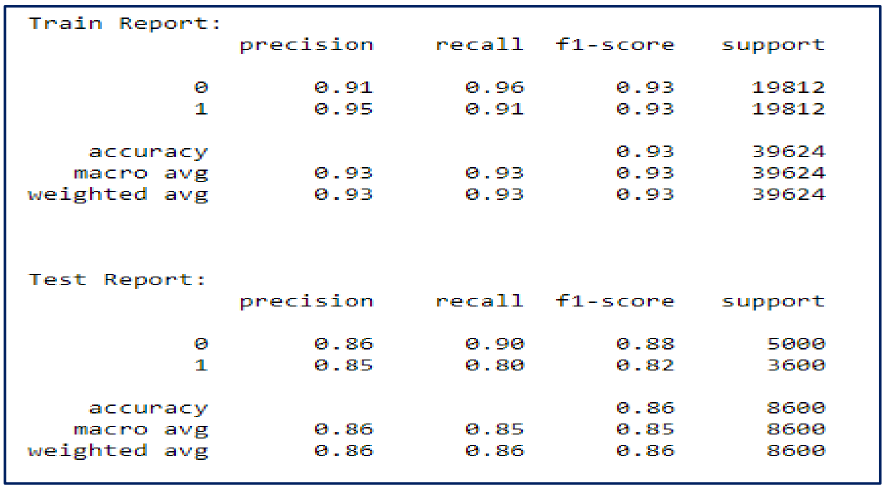
Confusion Matrix:



ROC Curve:

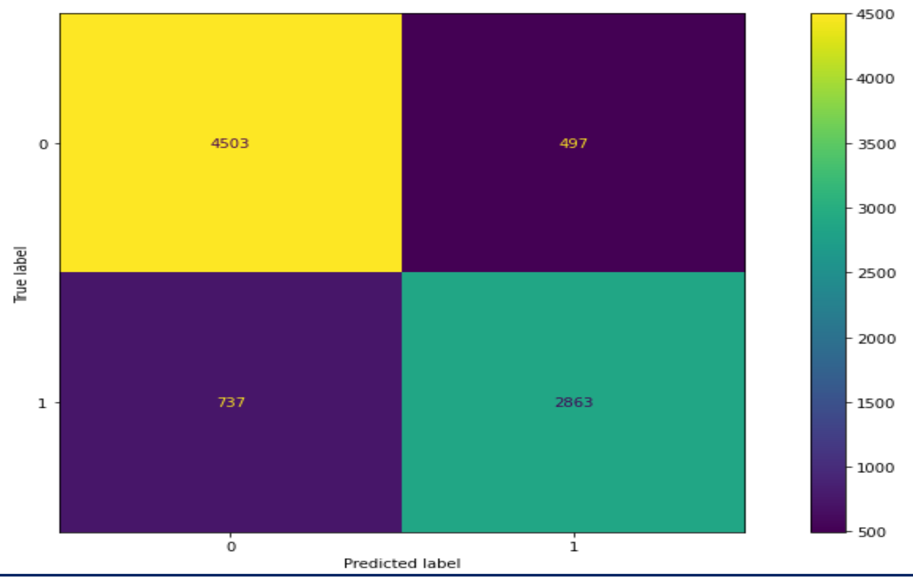


Bagging Classifier (base_estimator - RandomForestClassifier):

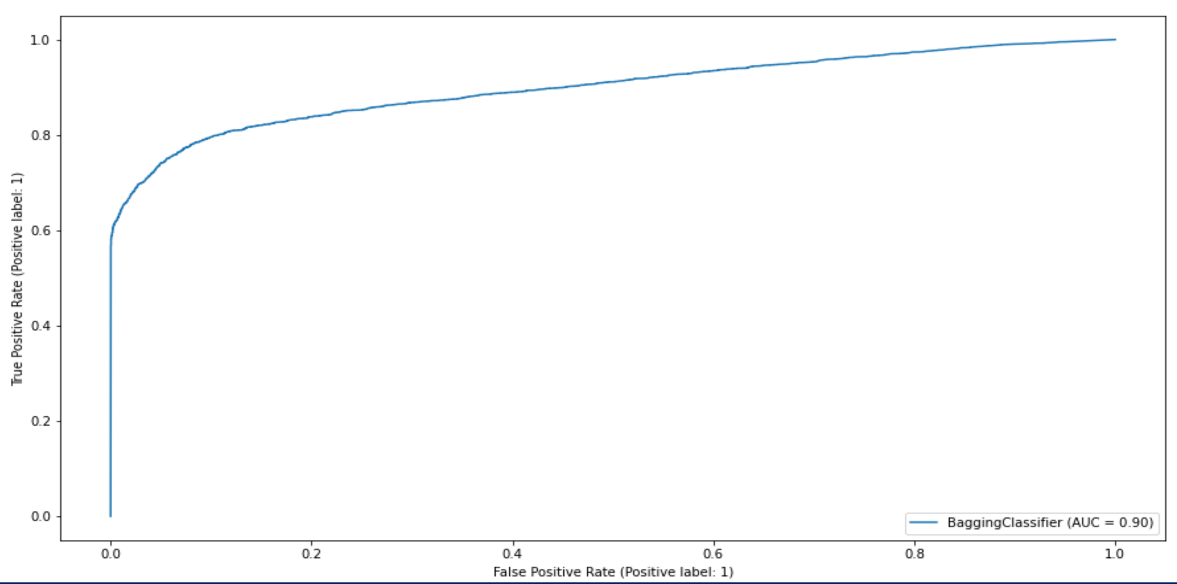


```
Train Report:
              precision    recall  f1-score   support

           0       0.91      0.96      0.93     19812
           1       0.95      0.91      0.93     19812

    accuracy                           0.93     39624
   macro avg       0.93      0.93      0.93     39624
weighted avg       0.93      0.93      0.93     39624


Test Report:
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      5000
           1       0.85      0.80      0.82      3600

    accuracy                           0.86      8600
   macro avg       0.86      0.85      0.85      8600
weighted avg       0.86      0.86      0.86      8600
```

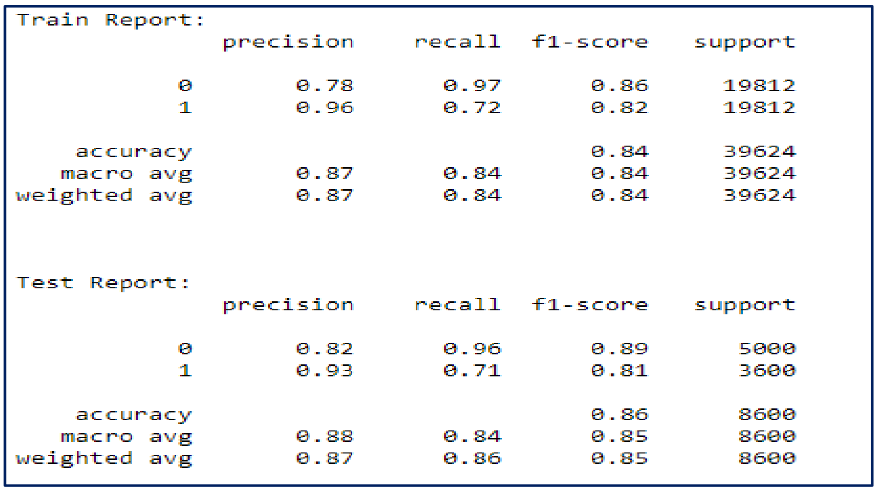
Confusion Matrix:



ROC Curve:

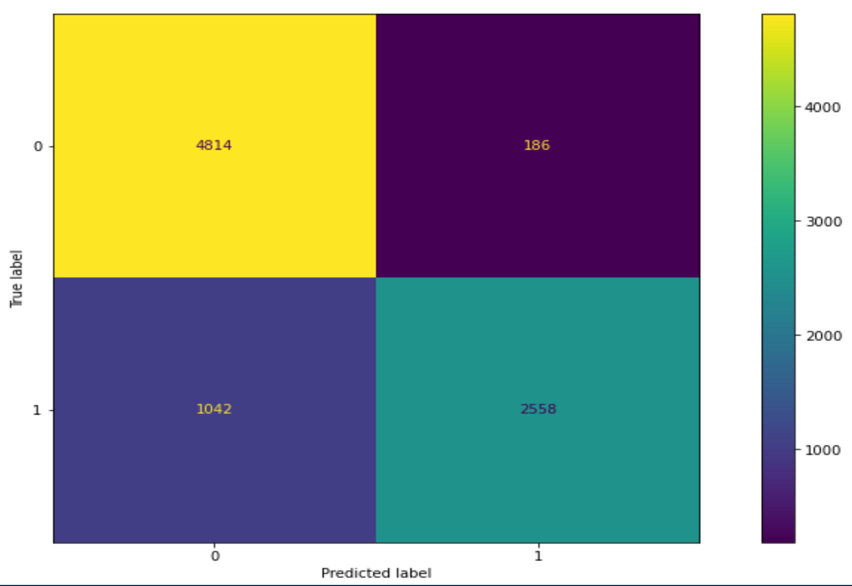


Gradient Boosting Classifier:

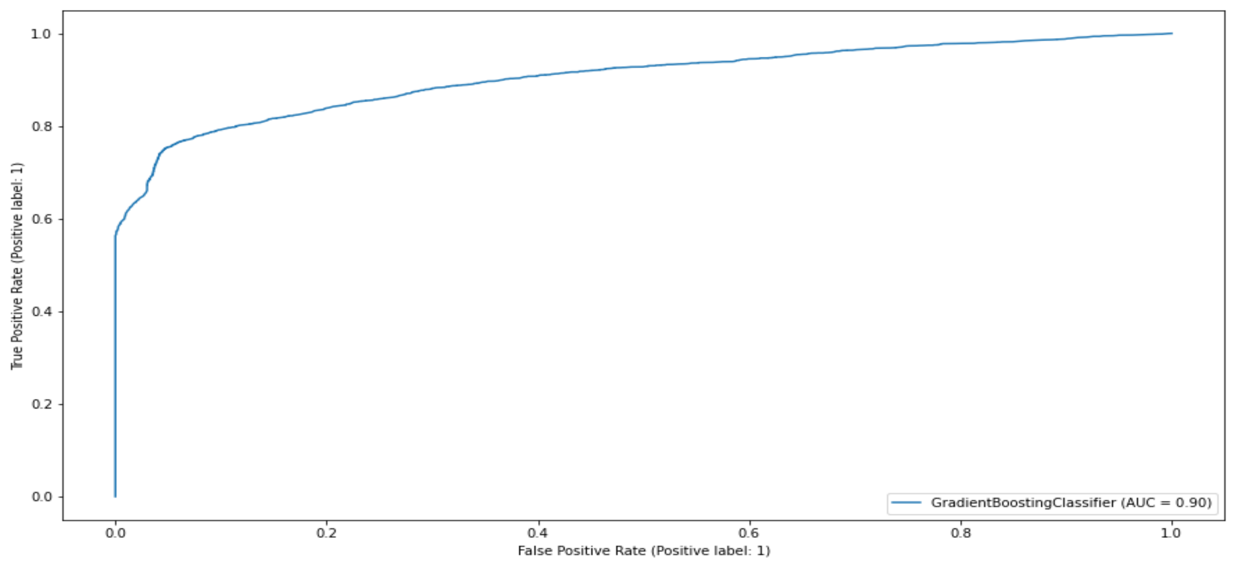


```
Train Report:
              precision    recall  f1-score   support

           0       0.78      0.97      0.86     19812
           1       0.96      0.72      0.82     19812

    accuracy                           0.84     39624
   macro avg       0.87      0.84      0.84     39624
weighted avg       0.87      0.84      0.84     39624


Test Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.89      5000
           1       0.93      0.71      0.81      3600

    accuracy                           0.86      8600
   macro avg       0.88      0.84      0.85      8600
weighted avg       0.87      0.86      0.85      8600
```

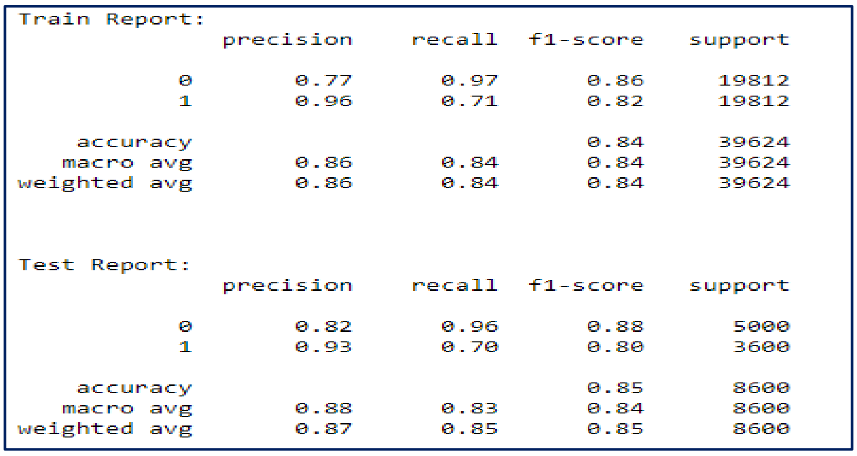
Confusion Matrix:



ROC Curve:

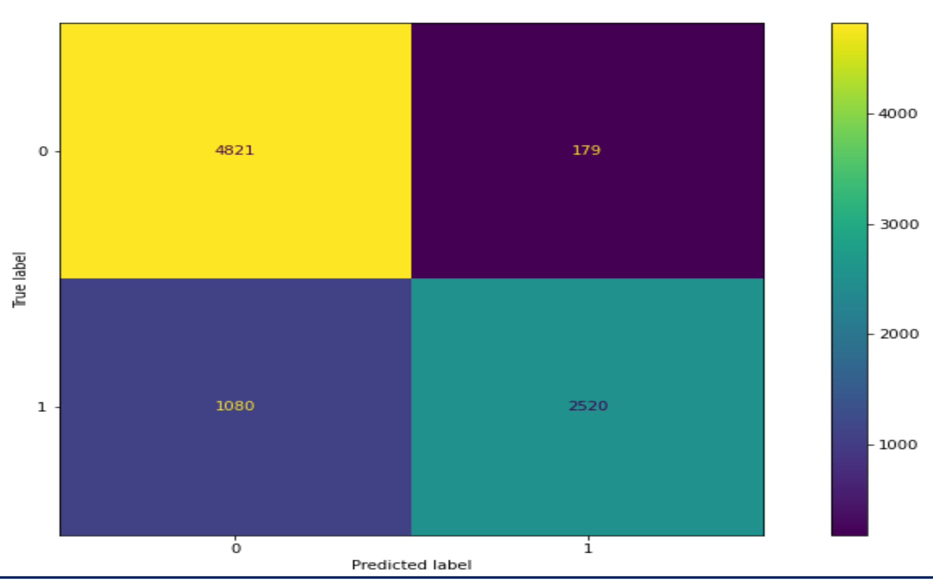


Bagging Classifier (base_estimator = GradientBoostingClassifier):

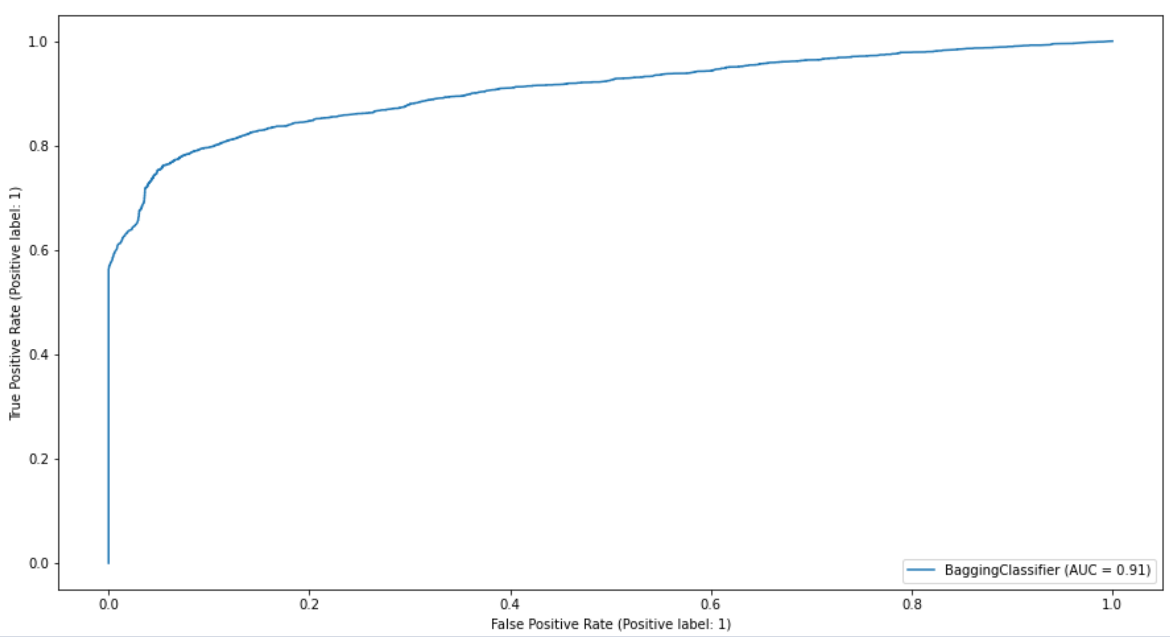


```
Train Report:
              precision    recall  f1-score   support

           0       0.77      0.97      0.86     19812
           1       0.96      0.71      0.82     19812

    accuracy                           0.84     39624
   macro avg       0.86      0.84      0.84     39624
weighted avg       0.86      0.84      0.84     39624


Test Report:
              precision    recall  f1-score   support

           0       0.82      0.96      0.88      5000
           1       0.93      0.70      0.80      3600

    accuracy                           0.85      8600
   macro avg       0.88      0.83      0.84      8600
weighted avg       0.87      0.85      0.85      8600
```

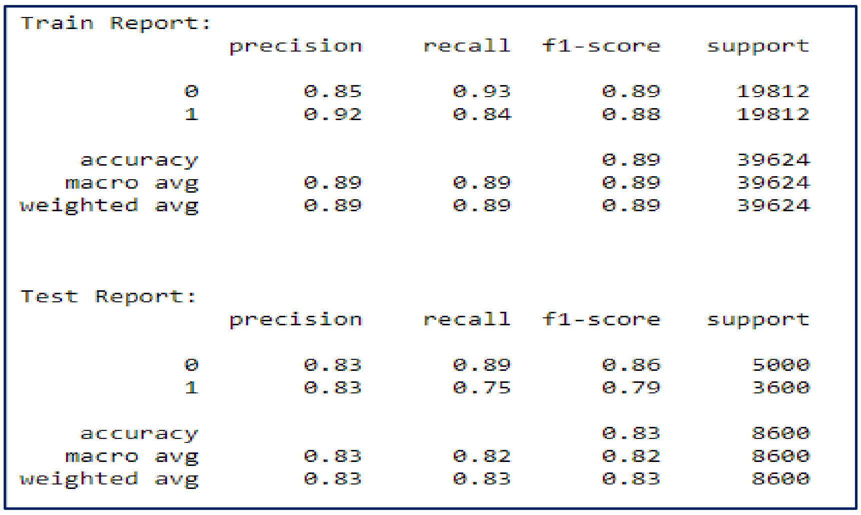
Confusion Matrix:



ROC Curve:

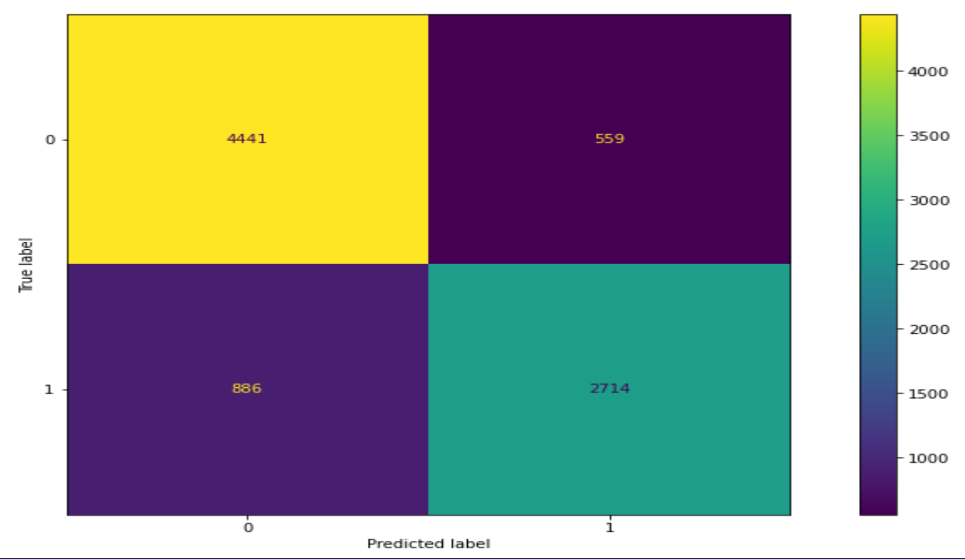


Bagging Classifier (base_estimator = KNeighborsClassifier):

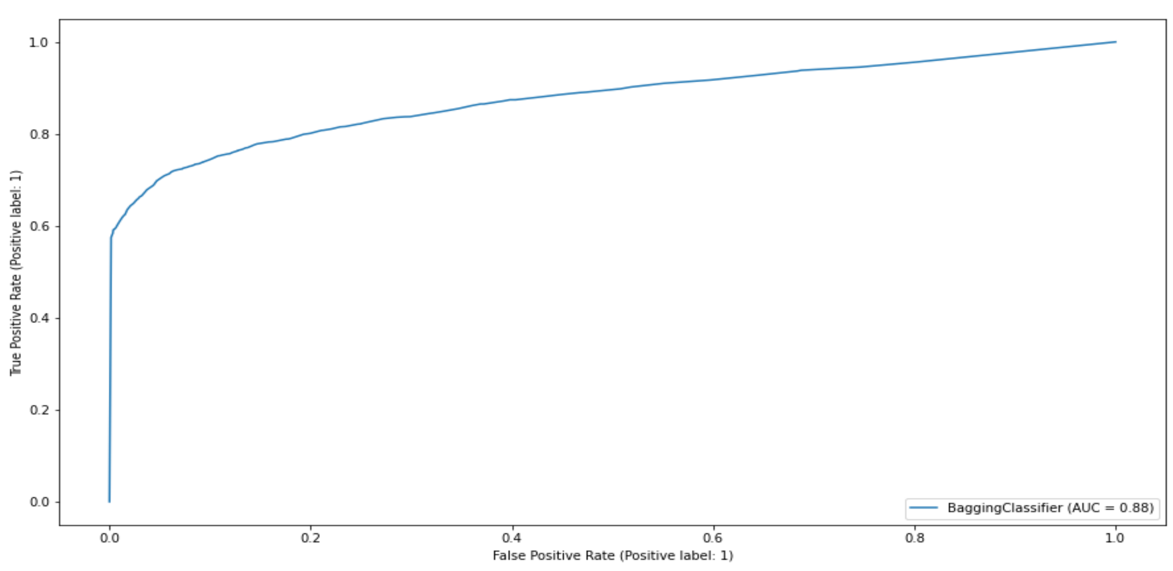


```
Train Report:
              precision    recall  f1-score   support

           0       0.85      0.93      0.89     19812
           1       0.92      0.84      0.88     19812

    accuracy                           0.89     39624
   macro avg       0.89      0.89      0.89     39624
weighted avg       0.89      0.89      0.89     39624


Test Report:
              precision    recall  f1-score   support

           0       0.83      0.89      0.86      5000
           1       0.83      0.75      0.79      3600

    accuracy                           0.83      8600
   macro avg       0.83      0.82      0.82      8600
weighted avg       0.83      0.83      0.83      8600
```

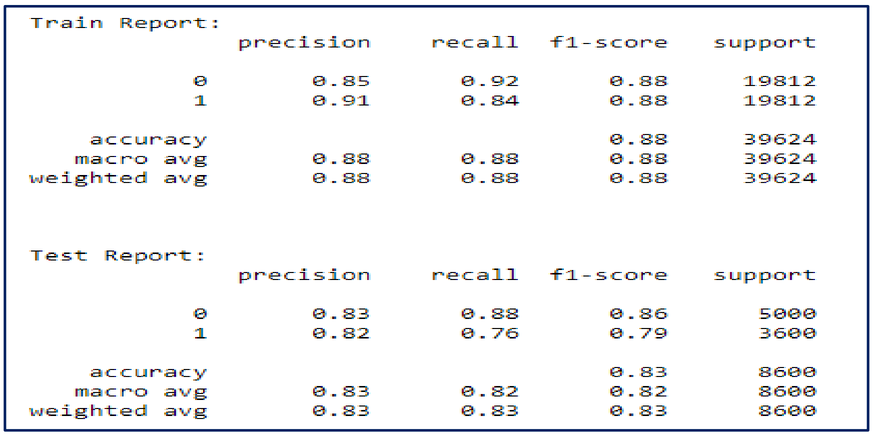
Confusion Matrix:



ROC Curve:

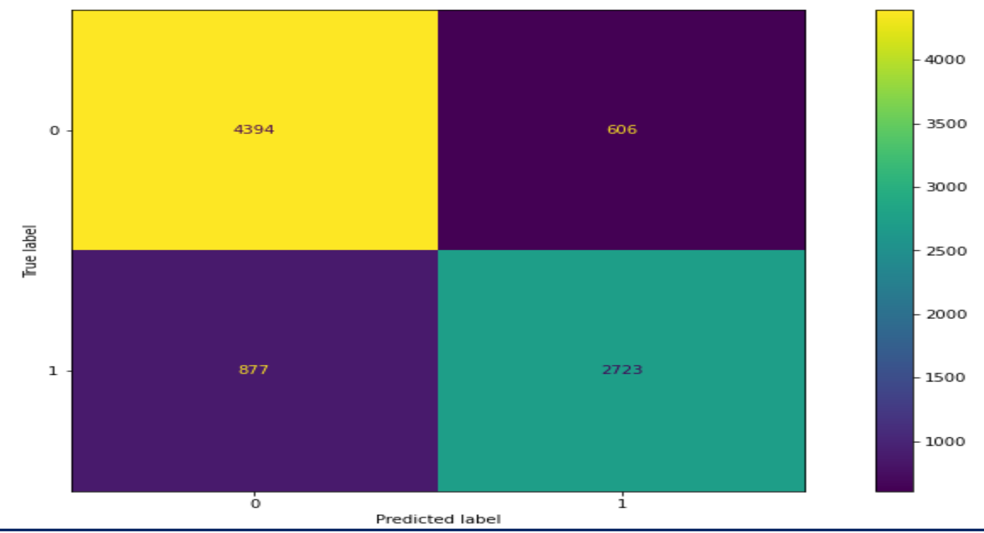


KNeighbors Classifier:

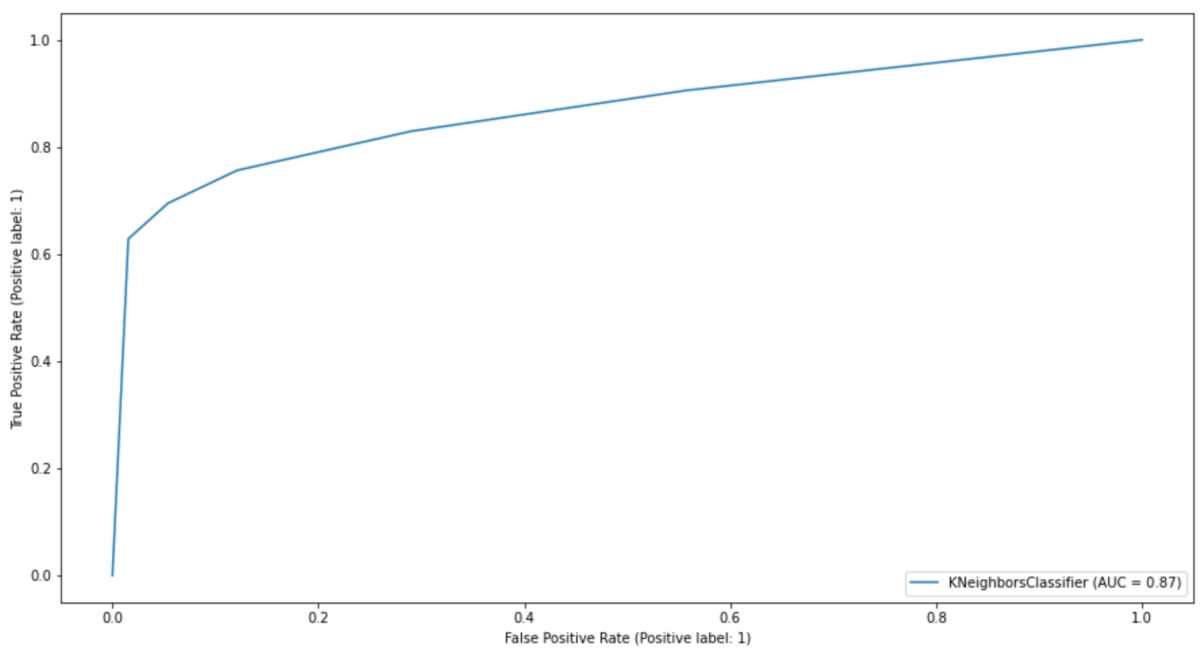


```
Train Report:
              precision    recall  f1-score   support

           0       0.85      0.92      0.88     19812
           1       0.91      0.84      0.88     19812

    accuracy                           0.88     39624
   macro avg       0.88      0.88      0.88     39624
weighted avg       0.88      0.88      0.88     39624


Test Report:
              precision    recall  f1-score   support

           0       0.83      0.88      0.86      5000
           1       0.82      0.76      0.79      3600

    accuracy                           0.83      8600
   macro avg       0.83      0.82      0.82      8600
weighted avg       0.83      0.83      0.83      8600
```

Confusion Matrix:



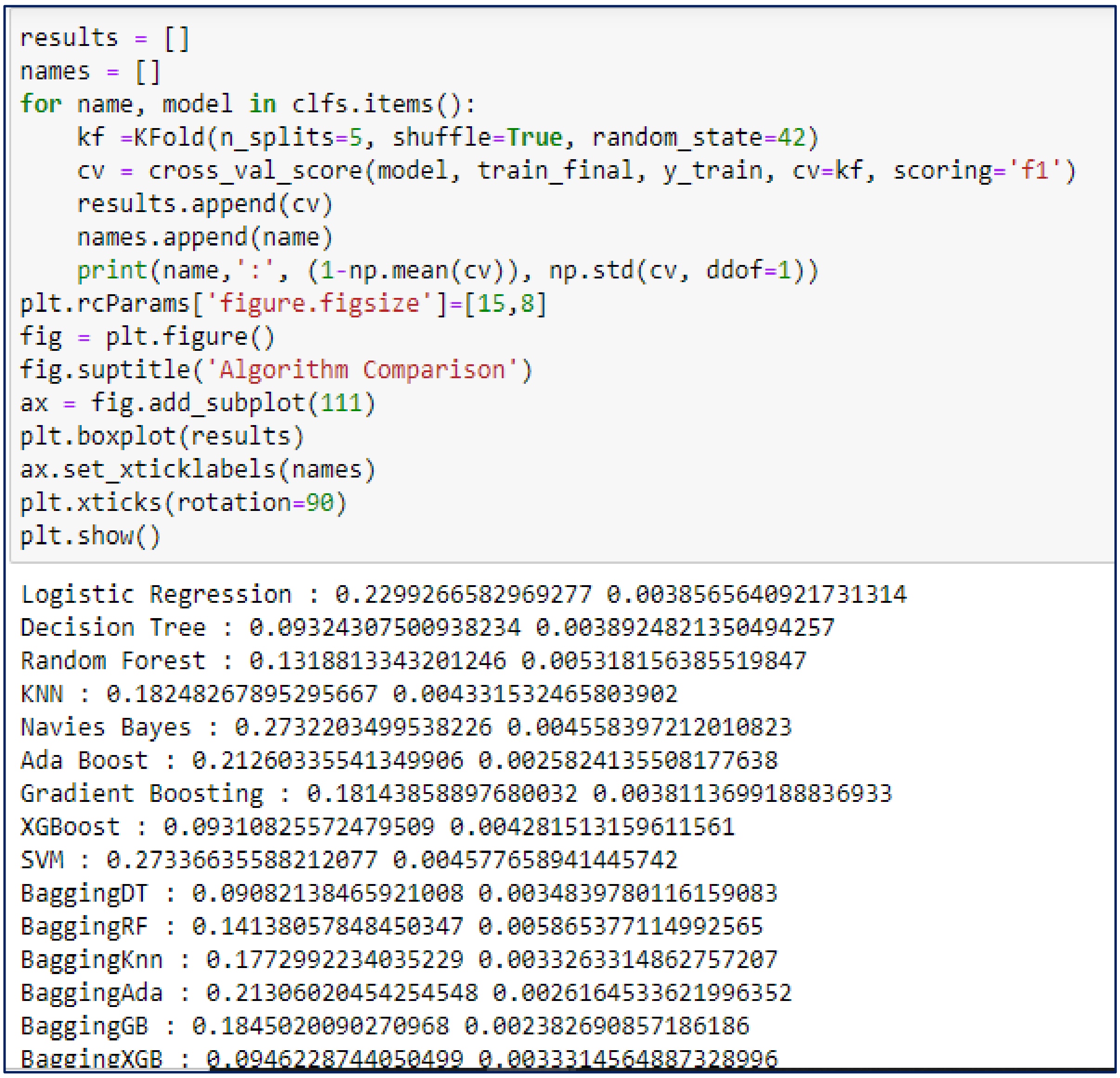
ROC Curve:



Error Rate Comparison of all Models

So far we have displayed various metrics for models that have given us the best performance, to get a better understanding of each model’s strengths and weaknesses.

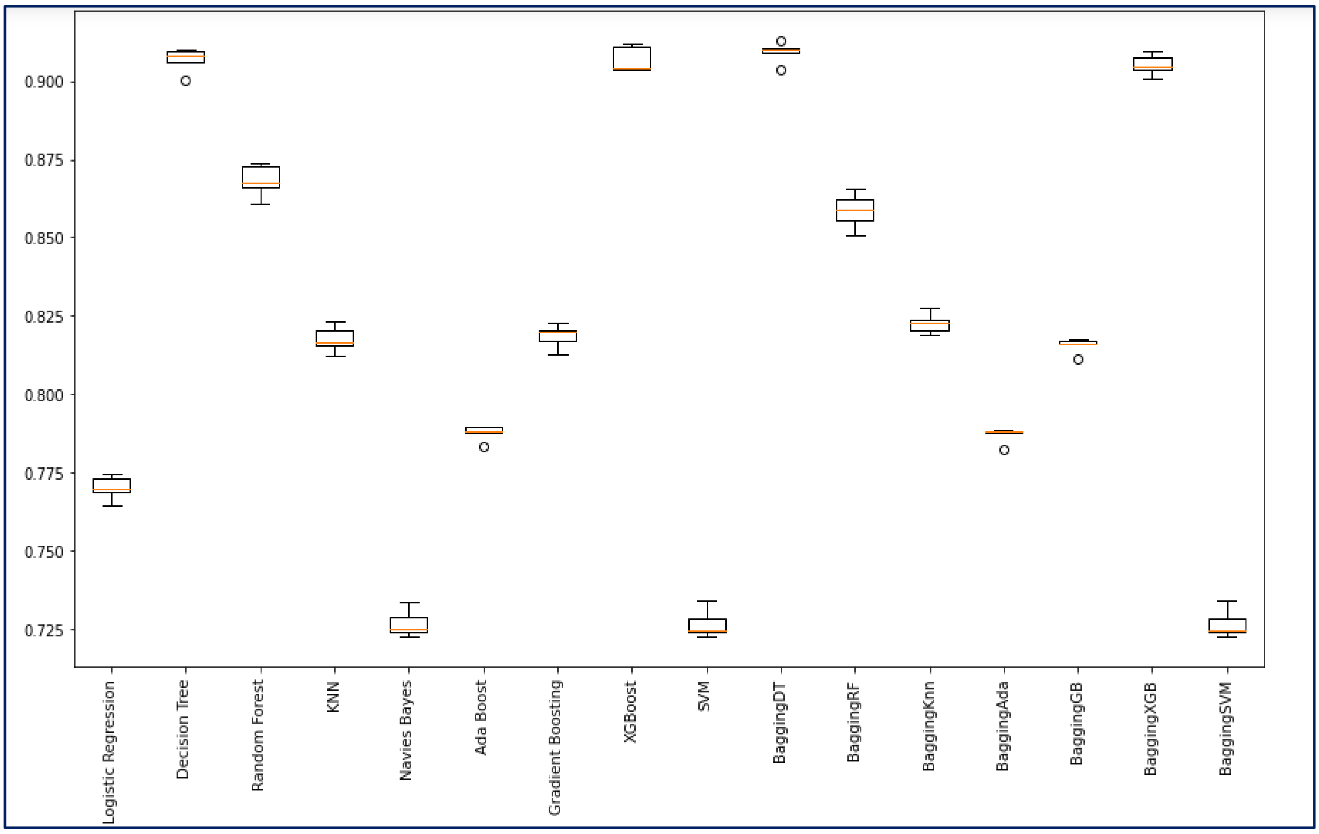
Let us now score and plot all the models together to get a sense of the model with the least error rate.



```python
results = []
names = []
for name, model in clfs.items():
    kf =KFold(n_splits=5, shuffle=True, random_state=42)
    cv = cross_val_score(model, train_final, y_train, cv=kf, scoring='f1')
    results.append(cv)
    names.append(name)
    print(name,':', (1-np.mean(cv)), np.std(cv, ddof=1))
plt.rcParams['figure.figsize']=[15,8]
fig = plt.figure()
fig.suptitle('Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.xticks(rotation=90)
plt.show()
```

```
Logistic Regression : 0.2299266582969277 0.0038565640921731314
Decision Tree : 0.09324307500938234 0.0038924821350494257
Random Forest : 0.1318813343201246 0.005318156385519847
KNN : 0.18248267895295667 0.004331532465803902
Navies Bayes : 0.2732203499538226 0.004558397212010823
Ada Boost : 0.21260335541349906 0.0025824135508177638
Gradient Boosting : 0.18143858897680032 0.0038113699188836933
XGBoost : 0.09310825572479509 0.004281513159611561
SVM : 0.27336635588212077 0.004577658941445742
BaggingDT : 0.09082138465921008 0.0034839780116159083
BaggingRF : 0.14138057848450347 0.005865377114992565
BaggingKnn : 0.1772992234035229 0.0033263314862757207
BaggingAda : 0.21306020454254548 0.0026164533621996352
BaggingGB : 0.1845020090270968 0.002382690857186186
BaggingXGB : 0.0946228744050499 0.0033314564887328996
```

Algorithm Comparison:



- The XGBoost Classifier has the highest median f1 score, which gives us our best model, considering that higher the value of f1 score better is the performance of the model.
- XGBoost also has a smaller box and shorter whiskers indicating a more consistent performance compared to other models.
- By both vanilla model as well as error rate, we got to know that XGBoost Classifier is the best model compared to all other models.
- Hence we can move forward with working to tune key parameters for the XGBoost model and then check whether that might improve performance of the model.

Hyper Parameter Tuning

Hyper parameter tuning for the XGBoost Classifier

Let us now GridSearchCV to identify best parameters to be employed with our selected model.

Cross Validation:

Cross-validation is a statistical method used to estimate the skill of machine learning models. It is commonly used in applied machine learning to compare and select a model for a given predictive modelling problem because it is easy to understand, easy to implement, and results in skill estimates that generally have a lower bias than other methods.

K-Fold Cross-Validation:

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation. When a specific value for k is chosen, it may be used in place of k in the reference to the model, such as k=10 becoming 10-fold cross-validation.

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

The general procedure is as follows:

- Shuffle the dataset randomly.
- Split the dataset into k groups
- For each unique group
- Take the group as a hold out or test data set
- Take the remaining groups as a training data set
- Fit a model on the training set and evaluate it on the test set
- Retain the evaluation score and discard the model
- Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold out set 1 time and used to train the model k-1 times.

```
param_grid = {
    'learning_rate': [0.2,0.18,0.22],
    'max_depth': [6,7,8],
    'n_estimators': [350,370,390,410,430,400],
    'gamma': [0]}

kf= KFold(n_splits=5, shuffle=True, random_state=42)
cv=GridSearchCV(estimator=xgb,param_grid=param_grid,cv=kf,n_jobs=-1,
                return_train_score=False,verbose=3,scoring='f1')
cv.fit(train_final,y_train)
```

Post hyper parameter tuning, we have identified the following values for each key parameter to be used as a part of fitting and training the model.

```
cv.best_params_

{'gamma': 0, 'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 400}
```

We shall now utilize the above stated parameters as part of the final model run and analysis.

Final Model performance post hyperparameter tuning:

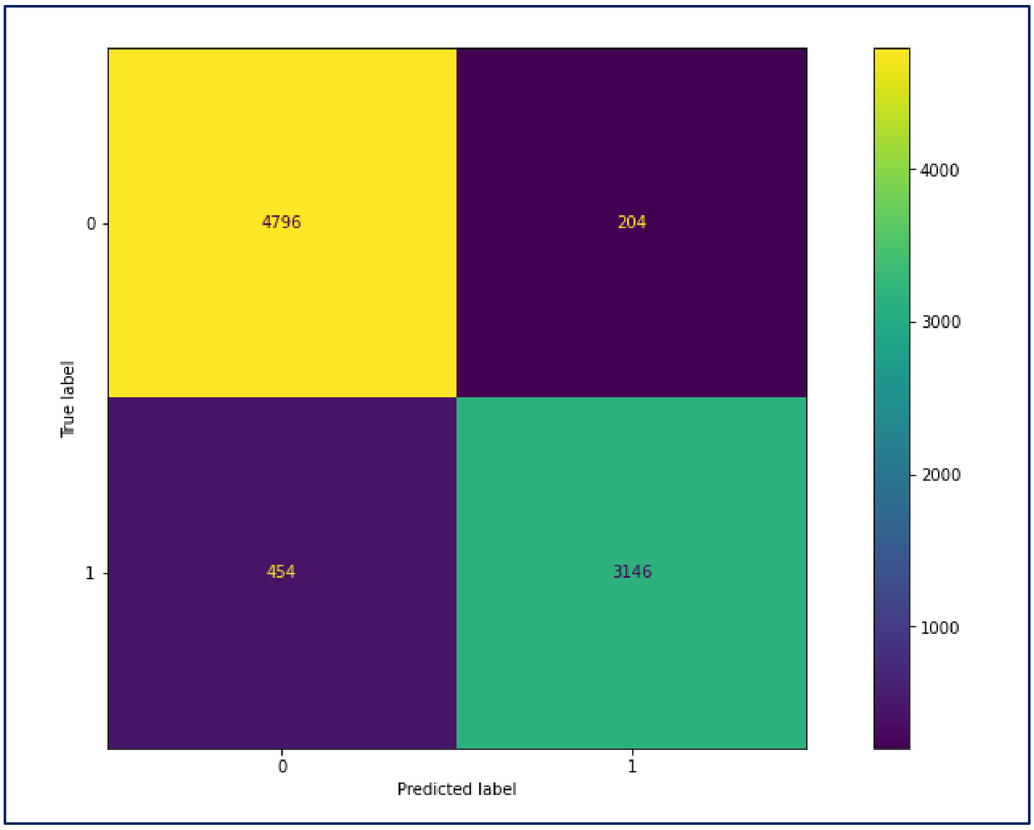
Fitting the XGBoost model with the parameters obtained

```
xgb_tuned_model = XGBClassifier(learning_rate = 0.2,max_depth = 7,
                                gamma = 0,n_estimators=400,random_state=42)
xgb_model = xgb_tuned_model.fit(train_final, y_train)
```

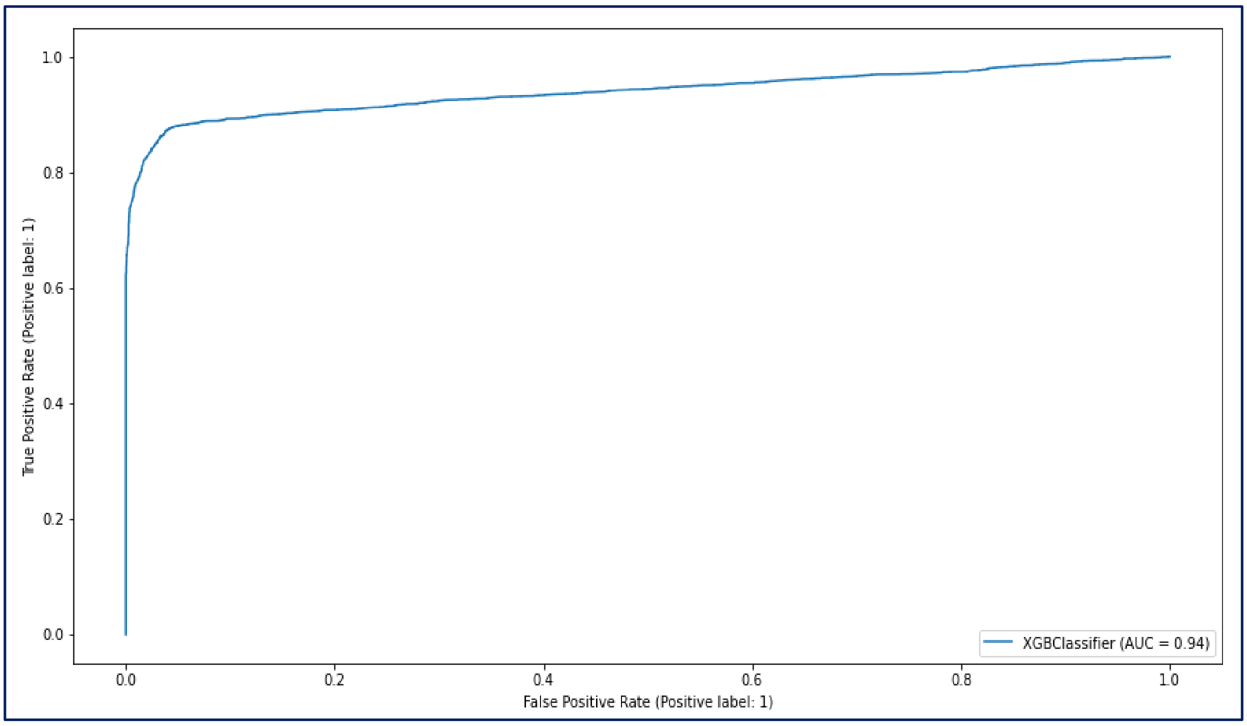
Scoring the XGBoost model with the selected parameters:

```
Train Report
              precision    recall  f1-score   support

           0       0.91      0.97      0.94     19812
           1       0.96      0.90      0.93     19812

    accuracy                           0.93     39624
   macro avg       0.94      0.93      0.93     39624
weighted avg       0.94      0.93      0.93     39624


Test Report
              precision    recall  f1-score   support

           0       0.91      0.96      0.94      5000
           1       0.94      0.87      0.91      3600

    accuracy                           0.92      8600
   macro avg       0.93      0.92      0.92      8600
weighted avg       0.92      0.92      0.92      8600
```

Confusion Matrix



Roc Curve



Insights

The confusion matrix of the initial XGB classifier model showed that it had a high number of false negatives, meaning that it misclassified a large number of positive instances as negative.

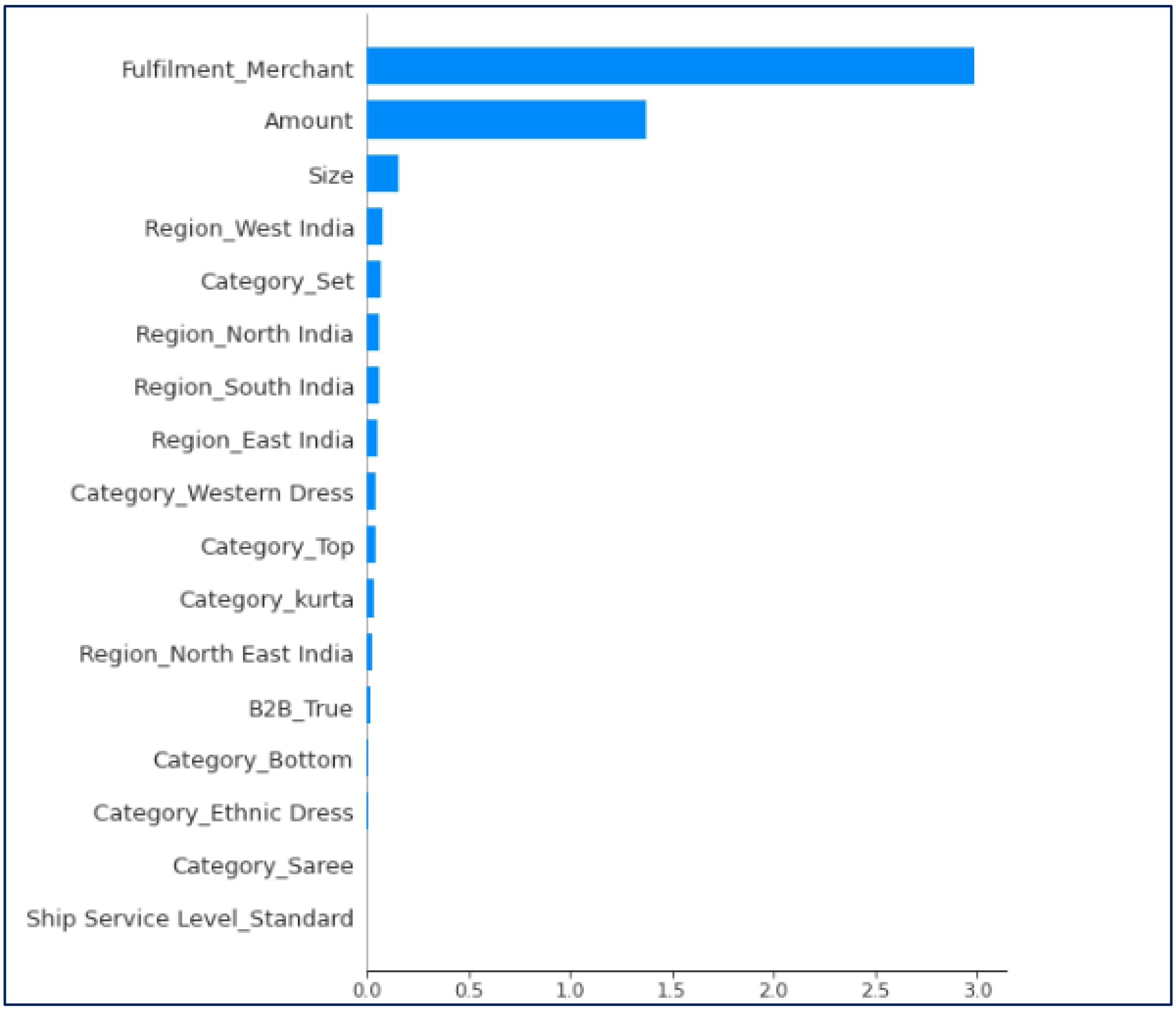
This led to a low recall score and a relatively high false negative rate. On the other hand, the hyper tuned model showed a significant improvement in terms of reducing the false negative rate and improving the recall score.

Additionally, the hyper tuned model also showed a lower false positive rate, which means that it was better at correctly identifying negative instances.

Overall, the hyper tuned model had a better balance between precision and recall, and it performed significantly better than the base XGB classifier model.

Model Interpretation and feature importance:

Now that we have identified the best model and parameters, it would be helpful to look at which features most impact the target variable, predictability and model performance.



Each bar in the plot represents a feature. The bars are sorted in descending order of the sum of the absolute values across all instances, which indicates the overall importance of each feature in the model.

The most important features are (Fulfilment_Merchant, Amount and Size) with the highest bars. These features have the greatest impact on the predicted target variable, and should be given more attention when interpreting the model and making predictions.

Conclusion

Identification and further usage of the selected model. The XGBoost model with given parameters that gave is the highest performance scores, including F1 and ROC-AUC scores, has great potential to be used in deployment and processing and more recent order rejection data. Its relatively efficient computation should assist with easier implementation of the model.

Disparity among Amazon delivered and third-party delivered orders. Because of the extreme order cancellation distribution, with an over-whelming majority of cancellations occurring on orders delivered via third-parties, it is important to recognise that Amazon may have limited ability to effect change in the same that that it may have had, in case the pain points had occurred with Amazon delivered orders. Outside of warning or removing third-party merchants that continue to under-perform, there are limited changes that Amazon can force into effect. Nevertheless, insights gained through this analysis might prove quite useful to the third party merchants themselves, especially those that might be keen to improve their own record.

Focus on single-product purchases. It must be noted that while this model should prove to be quite useful in predicting and analysing order rejection possibilities, in specific instance such as multi-product orders, the predictability performance of the model may differ.

Small B2B component. It is important to keep in mind that this dataset primarily refers to the B2B orders, and only has a very small B2C portion in the dataset. While our final model may still be acceptable to deploy on a B2C heavy dataset, confirmation of its performance on such a dataset will require testing.