NYC TAXI DATA ANALYSIS

Manim Tirkey

Virginia Tech

CS5805 Machine Learning 1

Computer Science and Applications Department

Instructor: Reza Jafari

Contact: mtirkey@vt.edu

Table of Contents

Table of Figures and Tables

Tables

Figures

10	Heatmap for Correlation Matrix (Numerical Target)
11	Heatmap for Covariance Matrix (Numerical Target)
12	Heatmap for Covariance Matrix of Selected Features (Numerical Target)
13	PCA Analysis - Tracing features with more than 95% cumulative explained variance (Categorical Target)
14	PCA Analysis on transformed feature matrix - Tracing features with more than 95% cumulative explained variance (Categorical Target)
15	Random Forest Analysis and Feature Importances (Categorical Target)
16	VIF Analysis (Categorical Target)
17	Heatmap for Correlation Matrix (Categorical Target)
18	Heatmap for Covariance Matrix (Categorical Target)
19	Heatmap for Covariance Matrix of Selected Features (Categorical Target)
20	Stepwise Regression Model summary and removal of insignificant features
21	Stepwise Regression Model - Scatterplot (test vs predicted)

22	Actual vs Predicted total_amount for each test sample
23	Confidence Interval analysis on Predicted Values
24	Prediction Interval analysis on Predicted Values
25	Snippet of output showing Actual vs Predicted Values
26	F-statistic, t-statistic and p-values of final model
27	R-squared, Adjusted R-Squares, MSE of regression model
28	Confusion Matrix - Pre Pruning Decision Tree
29	ROC and AUC for Pre Pruning
30	Pre Pruned Decision Tree
31	Accuracy vs Alphas for training and test set
32	Confusion Matrix - Post Pruned Decision Tree
33	ROC and AUC for Post Pruning
34	Post Pruned Decision Tree
35	Confusion Matrix - Logistic Regression
36	ROC and AUC for Logistic Regression
37	Error rate vs Number of Neighbors (k)
38	Confusion Matrix - KNN Classifier
39	ROC and AUC for KNN Classifier
40	Confusion Matrix - SVM (linear) Classifier
41	ROC and AUC for SVM (linear) Classifier
42	Confusion Matrix - SVM (polynomial) Classifier
43	ROC and AUC for SVM (polynomial) Classifier
44	Confusion Matrix - SVM (rbf) Classifier

45	ROC and AUC for SVM (rbf) Classifier
46	Confusion Matrix - Naïve Bayes Classifier
47	ROC and AUC for Naïve Bayes Classifier
48	Confusion Matrix - Random Forest (Bagging) Classifier
49	ROC and AUC for Random Forest (Bagging) Classifier
50	Confusion Matrix - Random Forest (Stacking) Classifier
51	ROC and AUC for Random Forest (Stacking) Classifier
52	Confusion Matrix - Random Forest (Boosting) Classifier
53	ROC and AUC for Random Forest (Boosting) Classifier
54	Confusion Matrix - MLP Classifier
55	ROC and AUC for MLP Classifier
56	K-selection in K-means using Elbow Method
57	K-selection in K-means using Silhouette Method
58	Applying Transaction Encoder on our Categorical Dataset
59	Itemsets and Associations

Abstract

This project explores an extensive dataset of New York City's yellow taxi trips, spanning approximately 3 million rides in January 2023. The dataset, sourced from the TPEP/LPEP initiatives by tech companies, offers a detailed snapshot of urban mobility, including temporal information, spatial dimensions, and various quantitative metrics. With a focus on understanding urban transportation trends, fare dynamics, and driver-passenger behavior, the analysis comprises multiple phases.

In the initial phase, feature engineering and preprocessing techniques such as SVD, PCA, and condition number analysis are employed, along with Random Forest regression and VIF analysis, to reduce dimensionality and address collinearity. Outliers are removed, and the dataset is balanced through downsampling for classification analysis, where the tip given serves as the target variable.The second phase involves Regression Analysis, employing statistical analysis and backward stepwise regression to derive an equation explaining the general model for the target variable.The third phase undertakes a comprehensive classification analysis, employing various methods including decision tree analysis, logistic regression, k-NN, SVM, Naive Bayes, random forest, and neural network classifications. The goal is to identify the best classifier for the target variable (tip given).In the final phase, Association Rule Mining and Clustering are applied. K-means clustering and Apriori are utilized to identify clusters and association rules within the dataset, providing valuable insights into patterns and relationships. Overall, this project aims to uncover valuable trends and patterns within the NYC taxi dataset, contributing to the fields of data science and urban planning.

Keywords: New York City, Yellow taxis, TPEP/LPEP initiatives, Urban mobility, Feature engineering, Preprocessing, SVD, PCA, Collinearity, Random Forest regression, VIF analysis, Outlier removal, Downsampling, Classification analysis, Regression analysis, Statistical analysis, Backward stepwise regression, Decision tree analysis, Logistic regression, k-NN, SVM, Naive Bayes, Random forest, Neural network, Association Rule Mining, Clustering, K-means, Apriori,, Urban planning

Introduction

This project undertakes a comprehensive exploration of New York City's yellow taxi dataset, offering a valuable resource for both data scientists and urban planners. With a detailed record encompassing approximately 3 million taxi trips during January 2023, the dataset provides a nuanced view of urban mobility. Temporal details, including pickup and drop-off hours, spatial dimensions denoted by location IDs, and various quantitative metrics such as trip distance, fare, payment methods, and passenger counts, are integral components of this extensive dataset.

Supplied under the TPEP/LPEP initiatives by technology companies, this dataset serves as a goldmine for dissecting urban transportation trends, deciphering fare dynamics, and probing into the intricacies of driver-passenger behavior. Within this analysis, the total amount paid by riders emerges as a potential numerical target variable. Concurrently, the remaining features, as outlined in the provided dictionary, are considered as independent features for the initial Principal Component Analysis (PCA). Furthermore, a focal point for classification analysis is the "tip given" variable, providing insights into the tipping behavior of riders.

The project unfolds in four distinctive phases:

Phase One: Feature Engineering and Preprocessing. This initial phase employs techniques such as Singular Value Decomposition (SVD), PCA, and condition number analysis, along with Random Forest regression and Variance Inflation Factor (VIF) analysis. The objective is to reduce the dimensionality of the dataset, ensuring no collinearity exists. Additionally, outliers are systematically removed, and the dataset is balanced by downsampling to address the minority class.

Phase Two: Regression Analysis. The second phase delves into Regression Analysis, employing statistical analysis and backward stepwise regression on the target variable. T-tests, F-tests, and confidence interval analysis are performed to compare predicted and actual values of the target variable, total amount.

Phase Three: Classification Analysis. An extensive classification analysis marks the third phase, involving decision tree analysis, logistic regression, KNN, SVM, Naive Bayes, random forest, and neural network classifications. Hyperparameter tuning is applied for each classifier, and a comprehensive evaluation is conducted using metrics such as confusion matrix, precision, recall, specificity, F-score, AUC, and mean accuracy through stratified K-fold cross-validation.

Phase Four: Association Rule Mining and Clustering. The final phase focuses on Association Rule Mining and Clustering. K-means clustering and Apriori are applied to identify clusters and association rules within the dataset. Silhouette analysis and within-cluster variation plots are employed for optimal K-means determination, while Apriori utilizes transaction encoding to reveal association rules.

In summary, this report aims to uncover valuable insights into New York City's taxi dataset, utilizing advanced analytics and mining techniques across various phases of analysis.

Description of the Dataset

The dataset of our interest is the NYC taxi dataset (specifically for the time period January 2023). Yellow and green taxi trip records include fields capturing pick-up and drop-off dates/times, pick-up and drop-off locations, trip distances, itemized fares, rate types, payment types, and driver-reported passenger counts. The data used in the attached datasets were collected and provided to the NYC Taxi and Limousine Commission (TLC) by technology providers authorized under the Taxicab & Livery Passenger Enhancement Programs (TPEP/LPEP).

Feature Description:

- VendorID - A code indicating the TPEP provider that provided the record. 1= Creative Mobile Technologies, LLC; 2= VeriFone Inc.
- tpep_pickup_datetime - The date and time when the meter was engaged.
- tpep_dropoff_datetime - The date and time when the meter was disengaged.
- Passenger_count - The number of passengers in the vehicle.
- Trip_distance - The elapsed trip distance in miles reported by the taximeter.
- PULocationID - TLC Taxi Zone in which the taximeter was engaged
- DOLocationID - TLC Taxi Zone in which the taximeter was disengaged
- RateCodeID - The final rate code in effect at the end of the trip. 1= Standard rate 2=JF 3=Newark 4=Nassau or Westchester 5=Negotiated fare 6=Group ride
- Store_and_fwd_flag - This flag indicates whether the trip record was held in vehicle memory before sending it to the vendor, aka “store and forward,” because the vehicle did not have a connection to the server.

- Payment_type - A numeric code signifying how the passenger paid for the trip. 1= Credit card 2= Cash 3= No charge 4= Dispute 5= Unknown 6= Voided trip
- Fare_amount - The time-and-distance fare calculated by the meter.
- Extra - Miscellaneous extras and surcharges. Currently, this only includes the $0.50 and $1 rush hour and overnight charges.
- MTA_tax - $0.50 MTA tax that is automatically triggered based on the metered rate in use.
- Improvement_surcharge - $0.30 improvement surcharge assessed trips at the flag drop. The improvement surcharge began being levied in 2015.
- Tip_amount - Tip amount – This field is automatically populated for credit card tips. Cash tips are not included.
- Tolls_amount - Total amount of all tolls paid on a trip.
- Total_amount - The total amount charged to passengers. Does not include cash tips.
- Congestion_Surcharge - Total amount collected in trip for NYS congestion surcharge.
- Airport_fee - $1.25 for pick up only at LaGuardia and John F. Kennedy Airports

This dataset has been selected as it contains both categorical and numerical data to explain each and every ride. Our preferred target is ‘Total Amount’, which is the amount of money paid by the rider to the driver. We will for Regression analysis treat ‘Total Amount’ as the dependent variable and other remaining features as the independent variables. Again, for Classification analysis we will treat ‘Tip Given’ as our dependent variable and other remaining features as the independent variables. Also, the categorical data in our dataset is **label encoded** by default.

This dataset is important for a few reasons. The analysis of this dataset can help urban planners about transportation planning and also give them a lead about traffic pattern analysis. We can also gain economic insights just by aggregating the data and can further analyze how much each driver and the association is earning and how they impact the local economy. We can also draw inferences from the traffic data for the environmental consequences. So, it is a very important dataset to analyze such patterns.

Phase I - Feature Engineering

Data Cleaning

We firstly proceed to check the amount of missing data. We discover that only five columns have missing values. Passenger Count, Rate Code ID, Store and Forward Flag, Congestion Surcharge and Airport Fee. The amount of missing data is huge (71743 rows to be precise).

VendorID	0
tpep_pickup_datetime	0
tpep_dropoff_datetime	0
passenger_count	71743
trip_distance	0
RatecodeID	71743
store_and_fwd_flag	71743
PULocationID	0
DOLocationID	0
payment_type	0
fare_amount	0
extra	0
mta_tax	0
tip_amount	0
tolls_amount	0
improvement_surcharge	0
total_amount	0
congestion_surcharge	71743
airport_fee	71743

Fig 1. Count of Empty Rows per feature

We do not want to lose data as it might disrupt our further analyses so we choose to 'fillna' our data according to certain constraints.

```
passenger_mode = df['passenger_count'].mode()[0]
df['passenger_count'].fillna(passenger_mode, inplace=True)
ratecodeID_mode = df['RatecodeID'].mode()[0]
df['RatecodeID'].fillna(ratecodeID_mode, inplace=True)
store_and_fwd_flag_mode = df['store_and_fwd_flag'].mode()[0]
df['store_and_fwd_flag'].fillna(store_and_fwd_flag_mode, inplace=True)
df['congestion_surcharge'].fillna(0.0, inplace=True)
df['airport_fee'].fillna(0.00, inplace=True)
```

For passenger count, Rate code ID and Store and Forward Flag we are filling the empty rows with the mode. For other continuous variables we are assuming that the driver has not entered the data as there was neither a pickup from the airport nor was there any congestion in traffic so we fill the empty rows with 0.

After replacing the NaN data we get the following output.

```
VendorID                0
tpep_pickup_datetime    0
tpep_dropoff_datetime   0
passenger_count         0
trip_distance           0
RatecodeID              0
store_and_fwd_flag      0
PULocationID            0
DOLocationID            0
payment_type            0
fare_amount             0
extra                   0
mta_tax                 0
tip_amount              0
tolls_amount            0
improvement_surcharge   0
total_amount            0
congestion_surcharge    0
airport_fee             0
```

Fig 2. Count of Empty Rows per feature after cleaning

The next step of our preprocessing was to label encode the only string type column of our dataset i.e. Store and Forward Flag. We simply encoded ‘Y’ with 1 and ‘N’ with 0.

```
category_mapping_fwd_flag = {'N': 0, 'Y': 1}
```

The next step was to extract meaningful data from the temporal data provided in the dataset. The data is labeled as ‘tpep_dropoff_datetime’ and ‘tpep_pickup_datetime’. We use this datetime format data to extract new features for our dataset.

```
df['trip_duration'] = (df['tpep_dropoff_datetime'] - df['tpep_pickup_datetime']).dt.total_seconds() / 60
df['pickup_day'] = df['tpep_pickup_datetime'].dt.day_name()
df['dropoff_day'] = df['tpep_dropoff_datetime'].dt.day_name()
df['pickup_hour'] = df['tpep_pickup_datetime'].dt.hour
df['dropoff_hour'] = df['tpep_dropoff_datetime'].dt.hour
```

The new features that we have extracted are trip duration (in minutes), pickup and dropoff day, pickup and dropoff hour.

trip_duration	pickup_day	dropoff_day	pickup_hour
8.43	Sunday	Sunday	0
6.32	Sunday	Sunday	0
12.75	Sunday	Sunday	0
9.62	Sunday	Sunday	0
10.83	Sunday	Sunday	0

Fig 3. Newly added features using Temporal Data

We also label encode the pickup and dropoff day later on for our analyses.

Outlier Detection and Removal

Outliers were a big problem for the analysis. Outliers directly skewed the data and gave absurd observations for the PCA analysis and other analyses. Outliers also skewed our plots and removed any scope for EDA. So, it was very important for us to remove outliers so as to get any meaningful observations from our plots and further analysis.

The method we apply for detecting and removing the outliers is the IQR method. We calculate the 25th and 75th quartile for our data and then relax our quartile ranges a little bit and remove the observations beyond that range.

It is very important to note that we did not remove all the outliers. We have only removed the outliers from the theoretical range.

```
def QQ_calc_2(temp):
    for column_name in temp.columns:
        if pd.api.types.is_numeric_dtype(temp[column_name]):
            Q1 = temp[column_name].quantile(0.25)
            Q3 = temp[column_name].quantile(0.75)
            IQR = Q3 - Q1

            lower_bound = Q1 - 1.5 * IQR
            upper_bound = Q3 + 1.5 * IQR

            temp = temp[(temp[column_name] >= lower_bound) & (temp[column_name] <= upper_bound)]

    return temp
```

We plotted a before and after boxplot of our target variable to check if the outliers have been removed.

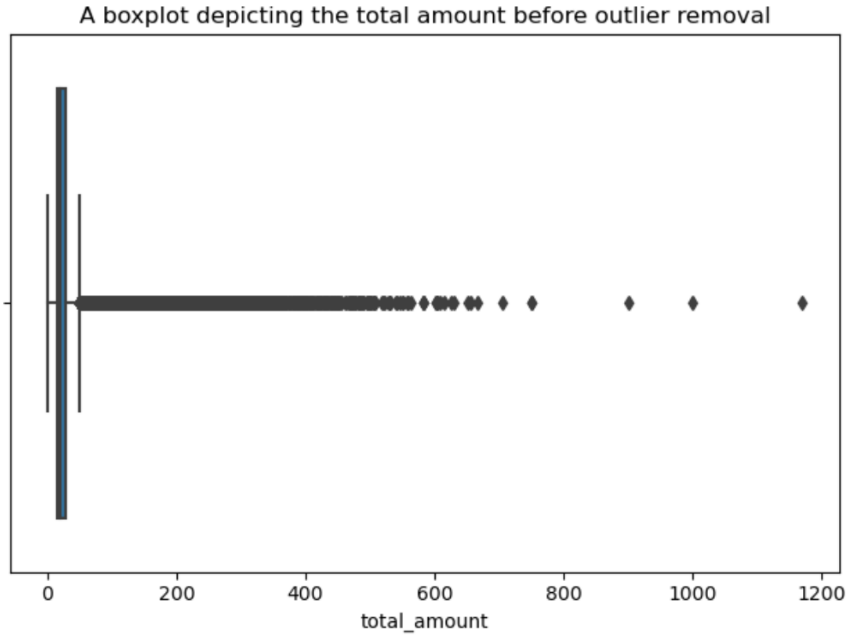

Fig 4. Boxplot - Total Amount with Outliers

We can easily notice a dense amount of outliers existing throughout our data. This data could've distorted our plots.

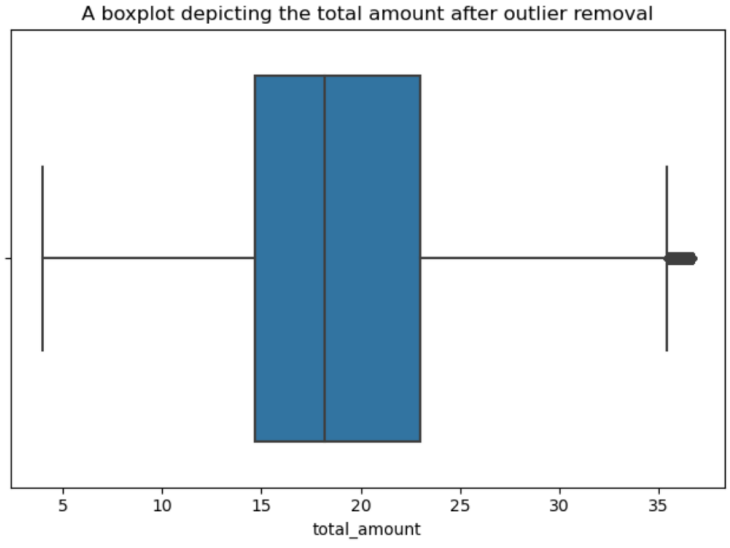

Fig 5. Boxplot - Total Amount without Outliers

After the removal of outliers, the dataset appears significantly cleaner. However, upon closer inspection, some may observe the persistence of certain outliers. This deliberate decision not to eliminate these outliers stems from the desire to maintain their presence for the purpose of data representation. By retaining these outliers, we aim to facilitate a more nuanced exploration of patterns through various plots and visualizations.

It's noteworthy that outliers have intentionally not been removed from columns such as Passenger Count, Payment Type, and Congestion Surcharge. This decision is rooted in the observation that a predominant type of observation dominates the majority of the data in these columns. The application of the IQR method assumes the remaining values in these columns to be outliers. Therefore, opting not to remove outliers in these specific columns is a deliberate choice, considering the unique characteristics of the data distribution in each case.

Now, we will perform two different feature reductions for both Regression Analysis and Classification Analysis.

Feature Reduction - Numerical Target (total_amount)

1. *PCA Analysis and Condition Number*

We performed PCA analysis on the standardized data and discovered the following observations.

a. We only need 9 components to explain more than 95% variance of our whole data.
b. The condition number for the original feature matrix is 997487526726133055475316666295910 4

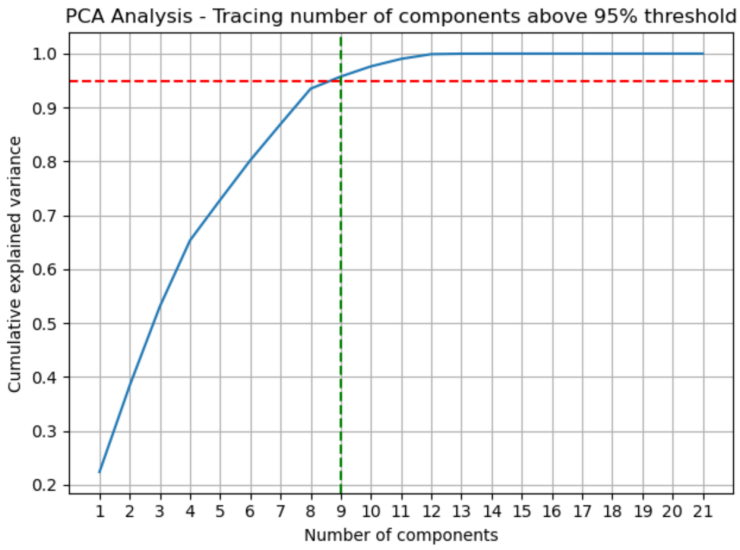

Fig 6. PCA Analysis - Tracing features with more than 95% cumulative explained variance (Numerical Target)

We then transformed our feature matrix for only 9 components and then calculated the condition number for that matrix as well.

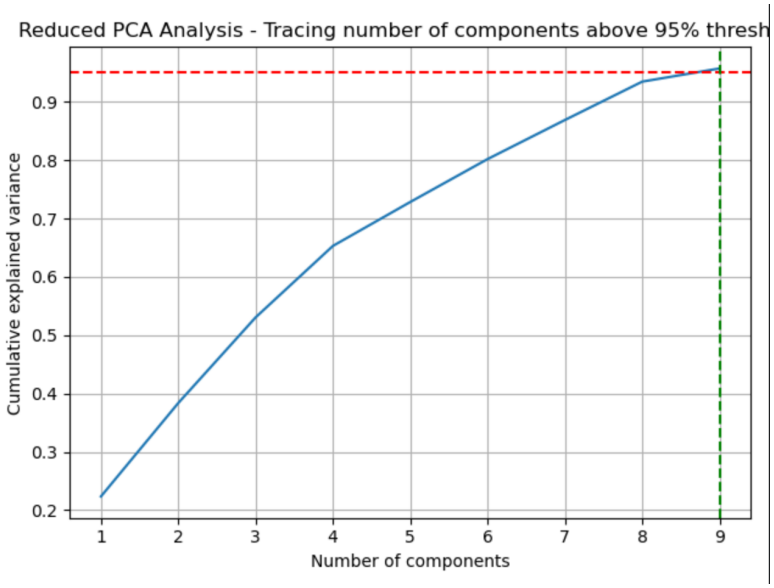

Fig 7. PCA Analysis on transformed feature matrix - Tracing features with more than 95% cumulative explained variance (Numerical Target)

We had the following observations for our transformed feature matrix.

a. We still need only 9 components to explain more than 95% variance in our data.

b. The condition number for the reduced feature matrix is 3.147

2. Singular Value Decomposition

We calculated the singular values for our original feature matrix and were seeking any drastic drop in the singular values.

We had similar observations as of PCA analysis and found the following condition number and singular values for the original feature matrix.

```
Original Condition Number =  997487526726133055475316666295910 4.000
SVD values of original matrix =  [395.629, 334.86,  320.596, 292.93,
229.576, 226.956, 216.581, 214.621, 125.724, 115.025,  98.511,
78.227,  24.387,  10.817, 0, 0, 0, 0, 0, 0, 0]
```

We can easily notice that after about 9 SVD values we can see a drastic drop in the values. So, we will select a threshold of 120 to select our features.

After selecting new features and transforming our feature matrix we see a new set of singular values and condition number.

```
Reduced Condition Number =  3.147
SVD values of reduced matrix =  [395.629 334.86  320.596 292.93
229.576 226.956 216.581 214.621 125.724]
```

The condition number confirms our selection of 9 variables as we see a dramatic drop on the values.

3. *Random Forest Analysis*

We ran a random forest analysis and plotted a graph for feature importances. The random forest analysis is a robust analysis and we found that it selects only a few features in order to reduce collinearity in our data.

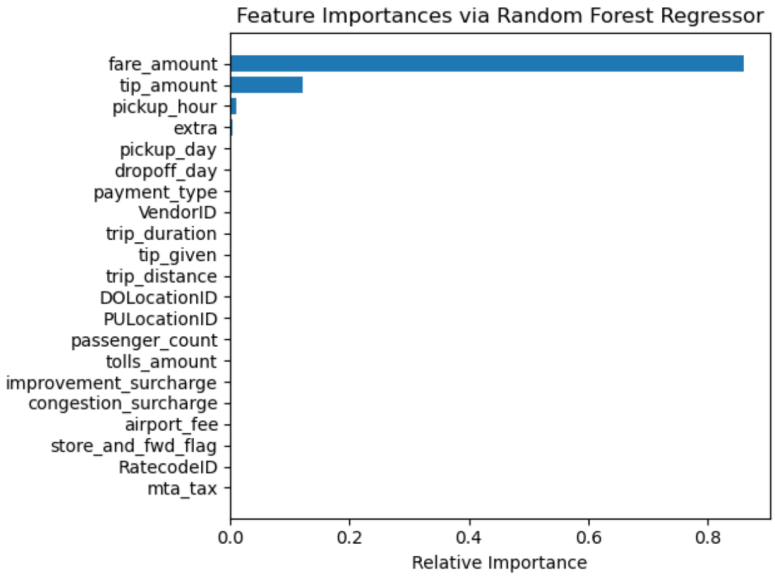

Fig 8. Random Forest Analysis and Feature Importances (Numerical Target)

We set the threshold for our relative importance to be 0.05 and found that only 2 features are important according to this analysis - Fare Amount and Tip Amount.

4. VIF Analysis

VIF Analysis is used for assessing the multicollinearity between the independent features. The VIF or Variance Inflation Factor is responsible for the selection of features which don't have multicollinearity based on a threshold we fix for our features.

We ran the VIF analysis iteratively, while removing the feature with highest VIF until all of the features had a VIF of around 5.

	feature	VIF
0	VendorID	2.735824
1	passenger_count	1.013884
2	trip_distance	14.839156
3	RatecodeID	0.000000
5	PULocationID	1.010844
6	DOLocationID	1.013888
7	payment_type	3.377602
8	fare_amount	53.636252
9	extra	2.806399
10	mta_tax	0.000000
11	tip_amount	3.119120
13	improvement_surcharge	0.000000
14	congestion_surcharge	0.000000
16	trip_duration	19.893982
17	pickup_day	213.958886
18	dropoff_day	213.909481
19	pickup_hour	1.164222
20	tip_given	5.108192

removed feature: pickup_day

	feature	VIF
0	VendorID	2.735753
1	passenger_count	1.013884
2	trip_distance	14.839156
3	RatecodeID	0.000000
4	PULocationID	1.010825
5	DOLocationID	1.013887
6	payment_type	3.377537
7	fare_amount	53.636248
8	extra	2.806390
9	mta_tax	0.000000
10	tip_amount	3.119108
11	improvement_surcharge	0.000000
12	congestion_surcharge	0.000000
13	trip_duration	19.893959
14	dropoff_day	1.043761
15	pickup_hour	1.163291
16	tip_given	5.108183

removed feature: fare_amount

	feature	VIF
0	VendorID	2.643210
1	passenger_count	1.013835
2	trip_distance	2.317003
3	RatecodeID	0.000000
4	PULocationID	1.010825
5	DOLocationID	1.013881
6	payment_type	3.377259
7	extra	2.799563
8	mta_tax	0.000000
9	tip_amount	3.106326
10	improvement_surcharge	0.000000
11	congestion_surcharge	0.000000
12	trip_duration	2.398464
13	dropoff_day	1.043681
14	pickup_hour	1.162427
15	tip_given	5.100739

Fig 9. VIF Analysis (Numerical Target)

```
Final Selected Columns by VIF:
['VendorID', 'passenger_count', 'trip_distance', 'RatecodeID',
'PULocationID', 'DOLocationID', 'payment_type', 'extra', 'mta_tax',
'tip_amount', 'improvement_surcharge', 'congestion_surcharge',
'trip_duration', 'dropoff_day', 'pickup_hour', 'tip_given']
```

Finally we decided to explore the covariance matrix and correlation matrix heatmaps to understand how the features correlate with each other.

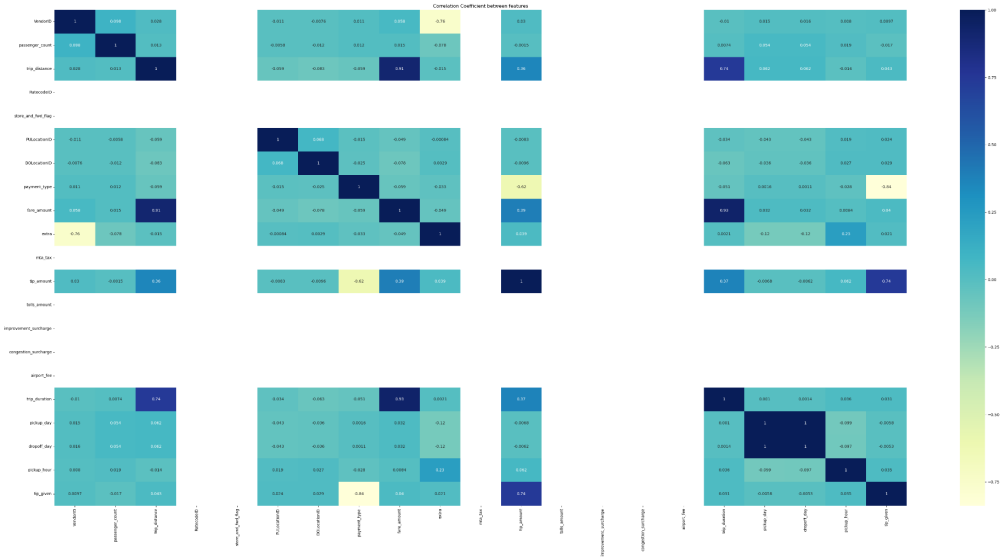

Fig 10. Heatmap for Correlation Matrix (Numerical Target)

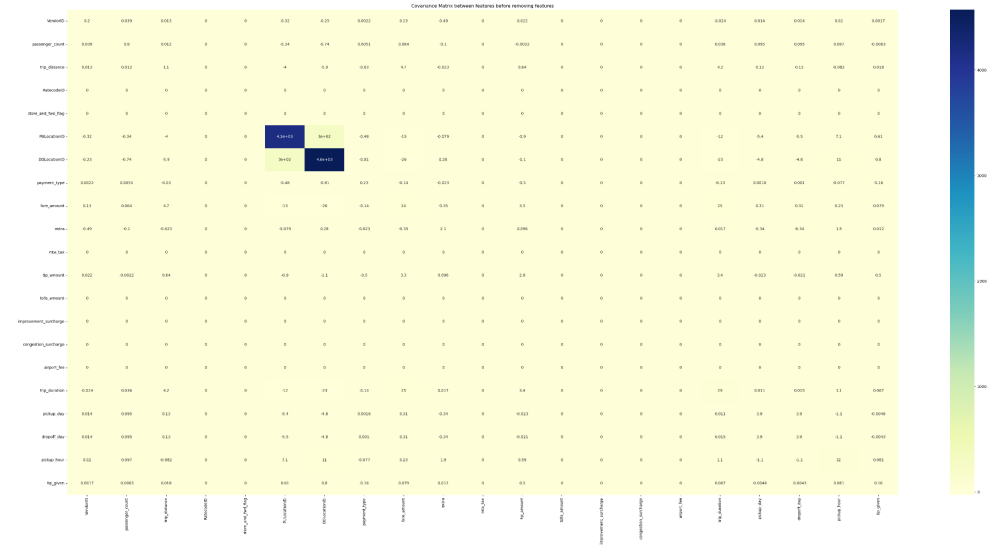

Fig 11. Heatmap for Covariance Matrix (Numerical Target)

We can quickly notice that there are few features which have huge gaps indicating that those features have constant values and are sparse in nature. This provides an insight that we need to remove these features as these features are for a lot of times responsible for Overfitting, irrelevant for our data, and lesser interpretability of our data.

Based on the analysis done in the PCA Analysis section, SVD section and by making observations from the Covariance and Correlation Matrices we have decided to drop the following 12 features from our dataset in order to reduce the Dimensionality of our Dataset.

```
['RatecodeID', 'store_and_fwd_flag', 'mta_tax', 'tolls_amount',
'improvement_surcharge','congestion_surcharge', 'airport_fee', 'tip_given',
'dropoff_day', 'trip_duration','tip_amount', 'fare_amount']
```

We plotted the covariance matrix of the final selected features as well and we can notice that we have successfully removed sparse and collinear features as well.

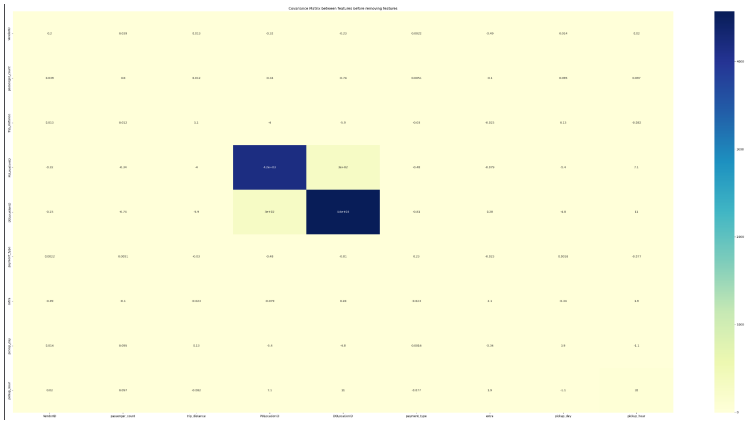

Fig 12. Heatmap for Covariance Matrix of Selected Features (Numerical Target)

Feature Reduction - Categorical Target (tip_given)

1. *PCA Analysis and Condition Number*

We performed PCA analysis on the standardized data and discovered the following observations.

a. We only need 8 components to explain more than 95% variance of our whole data.

b. The condition number for the original feature matrix is 201275057437004601535911263008194 56.000

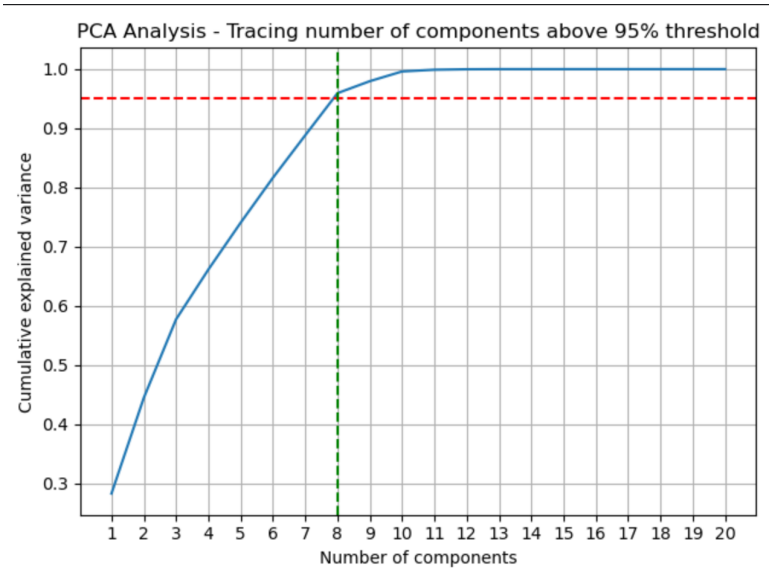

Fig 13. PCA Analysis - Tracing features with more than 95% cumulative explained variance (Categorical Target)

We then transformed our feature matrix for only 8 components and then calculated the condition number for that matrix as well.

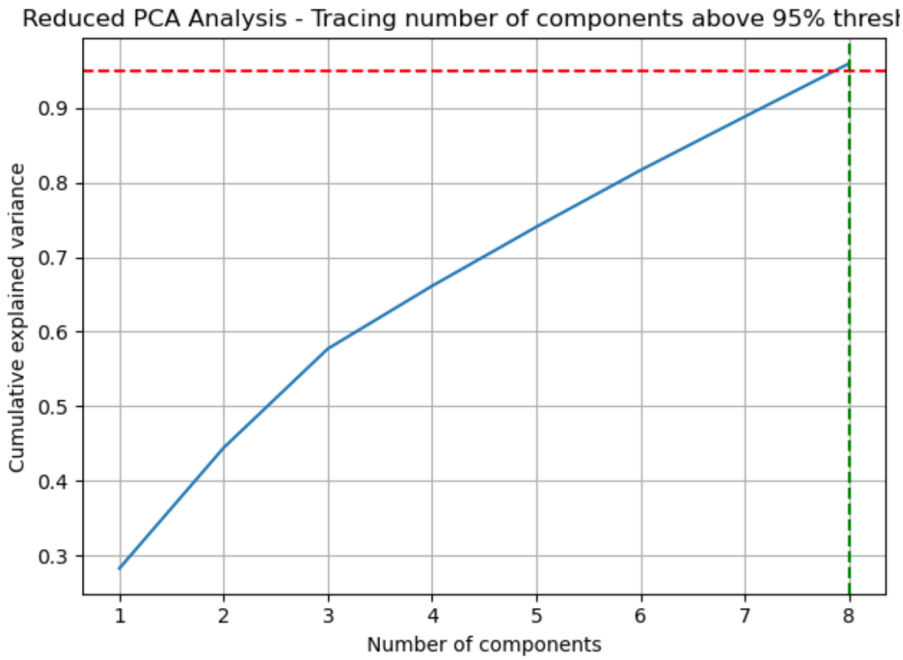

Fig 14. PCA Analysis on transformed feature matrix - Tracing features with more than 95% cumulative explained variance (Categorical Target)

We had the following observations for our transformed feature matrix.

a. We still need only 8 components to explain more than 95% variance in our data.

b. The condition number for the reduced feature matrix is 1.993

2. Singular Value Decomposition

We calculated the singular values for our original feature matrix and were seeking any drastic drop in the singular values.

We had similar observations as of PCA analysis and found the following condition number and singular values for the original feature matrix.

```
Original Condition Number =  201275057437004601535911263008194560.000
```

```
SVD values of original matrix =  [428.663 324.151 293.801 233.854
226.846 222.263 215.993 215.089 114.671
 103.351  43.347  23.179  11.999   0.      0.      0.      0.      0.
   0.      0.   ]
```

We can easily notice that after about 8 SVD values we can see a drastic drop in the values. So, we will select a threshold of 200 to select our features.

After selecting new features and transforming our feature matrix we see a new set of singular values and condition number.

```
Reduced Condition Number =  1.993
SVD values of reduced matrix =  [428.663 324.151 293.801 233.854
226.846 222.263 215.993 215.089]
```

The condition number confirms our selection of 8 variables as we see a dramatic drop on the values.

3. Random Forest Analysis

We ran a random forest analysis and plotted a graph for feature importances. The random forest analysis is a robust analysis and we found that it selects only a few features in order to reduce collinearity in our data.

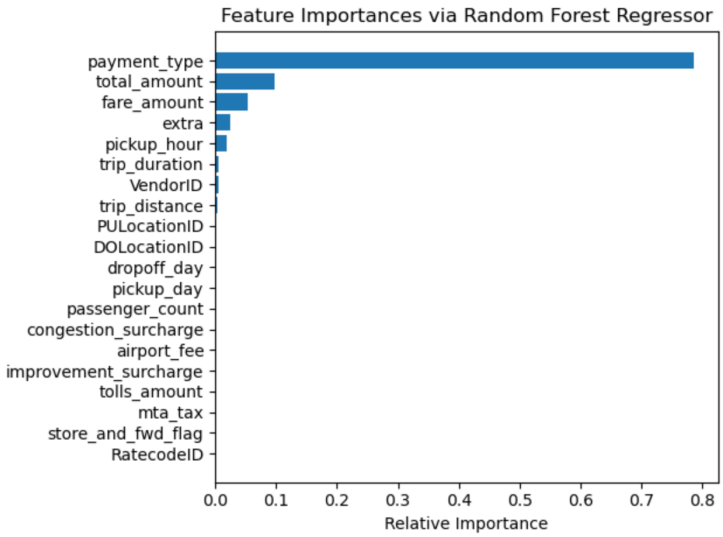

Fig 15. Random Forest Analysis and Feature Importances (Categorical Target)

We set the threshold for our relative importance to be 0.05 and found that only 3 features are important according to this analysis - Fare Amount, Total Amount and Payment Type.

4. VIF Analysis

VIF Analysis is used for assessing the multicollinearity between the independent features. The VIF or Variance Inflation Factor is responsible for the selection of features which don't have multicollinearity based on a threshold we fix for our features.

We ran the VIF analysis iteratively, while removing the feature with highest VIF until all of the features had a VIF of around 5.

	feature	VIF
0	VendorID	3.698823
1	passenger_count	1.013733
2	trip_distance	14.176860
3	RatecodeID	0.000000
5	PULocationID	1.009593
6	DOLocationID	1.015859
7	payment_type	1.847381
8	fare_amount	66.462258
9	extra	4.333966
10	mta_tax	0.000000
12	improvement_surcharge	0.000000
13	total_amount	21.282600
14	congestion_surcharge	0.000000
16	trip_duration	18.033172
17	pickup_day	173.904620
18	dropoff_day	173.874868
19	pickup_hour	1.163837

removed feature: pickup_day

	feature	VIF
0	VendorID	3.698823
1	passenger_count	1.013721
2	trip_distance	14.176735
3	RatecodeID	0.000000
4	PULocationID	1.009591
5	DOLocationID	1.015811
6	payment_type	1.847284
7	fare_amount	66.454050
8	extra	4.333954
9	mta_tax	0.000000
10	improvement_surcharge	0.000000
11	total_amount	21.280648
12	congestion_surcharge	0.000000
13	trip_duration	18.030997
14	dropoff_day	1.033014
15	pickup_hour	1.163701

removed feature: fare_amount

	feature	VIF
0	VendorID	3.633081
1	passenger_count	1.013617
2	trip_distance	5.496559
3	RatecodeID	0.000000
4	PULocationID	1.009569
5	DOLocationID	1.015705
6	payment_type	1.632513
7	extra	3.865805
8	mta_tax	0.000000
9	improvement_surcharge	0.000000
10	total_amount	15.734518
11	congestion_surcharge	0.000000
12	trip_duration	6.590022
13	dropoff_day	1.033013
14	pickup_hour	1.162985

removed feature: total_amount

	feature	VIF
0	VendorID	2.630893
1	passenger_count	1.012944
2	trip_distance	2.275253
3	RatecodeID	0.000000
4	PULocationID	1.009563
5	DOLocationID	1.015512
6	payment_type	1.010832
7	extra	2.785230
8	mta_tax	0.000000
9	improvement_surcharge	0.000000
10	congestion_surcharge	0.000000
11	trip_duration	2.257523
12	dropoff_day	1.032846
13	pickup_hour	1.162981

Fig 16. VIF Analysis (Categorical Target)

```
Final Selected Columns by VIF:
['VendorID', 'passenger_count', 'trip_distance', 'RatecodeID',
'PULocationID', 'DOLocationID', 'payment_type', 'extra', 'mta_tax',
'improvement_surcharge', 'congestion_surcharge', 'trip_duration',
'dropoff_day', 'pickup_hour']
```

Finally we decided to explore the covariance matrix and correlation matrix heatmaps to understand how the features correlate with each other.

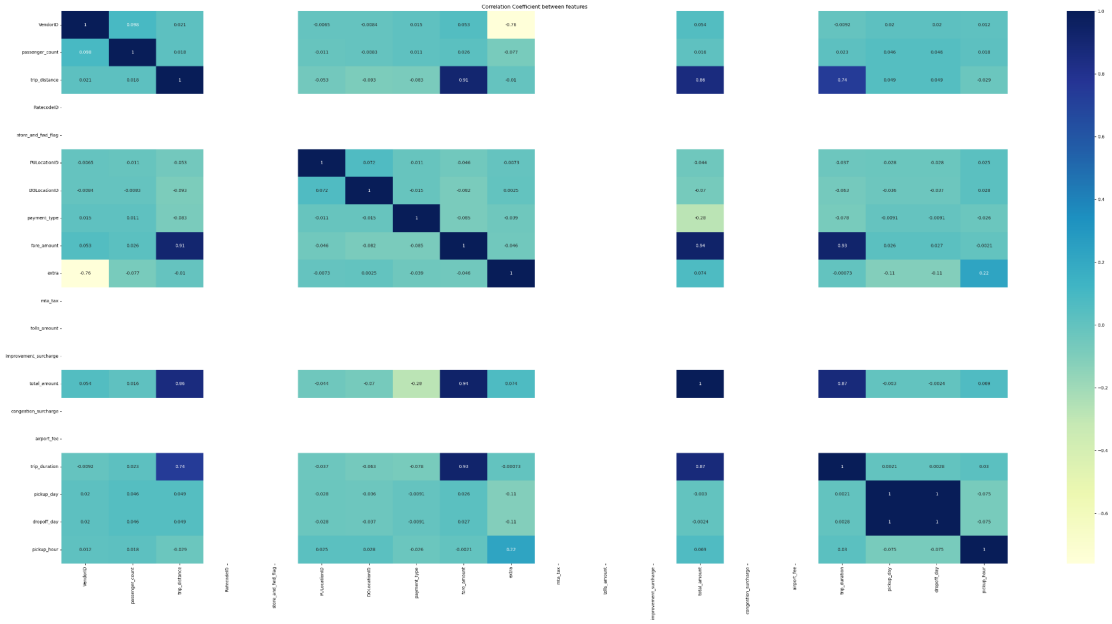

Fig 17. Heatmap for Correlation Matrix (Categorical Target)

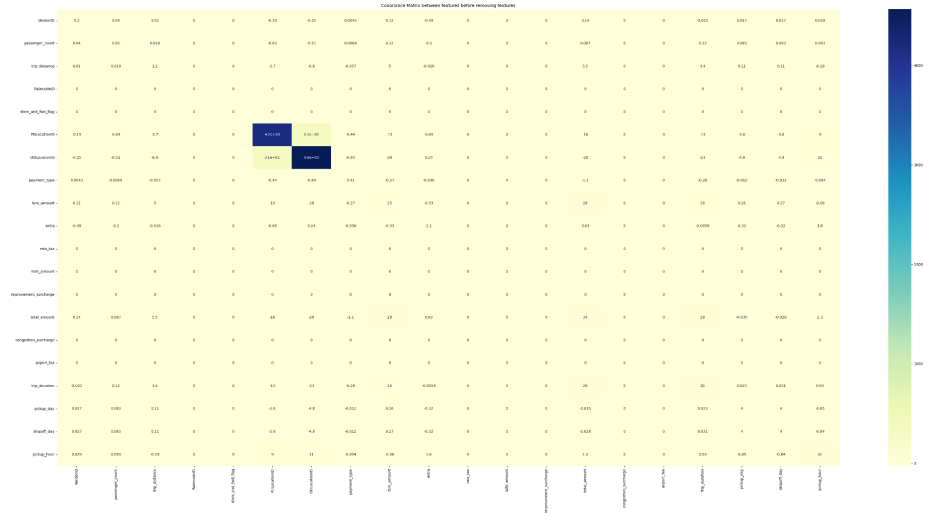

Fig 18. Heatmap for Covariance Matrix (Categorical Target)

We can quickly notice the same thing as numerical target analysis: there are few features which have huge gaps indicating that those features have constant values and are sparse in nature. This provides an insight that we need to remove these features as these features are for a lot of times responsible for Overfitting, irrelevant for our data, and lesser interpretability of our data.

Based on the analysis done in the PCA Analysis section, SVD section and by making observations from the Covariance and Correlation Matrices we have decided to drop the following 12 features from our dataset in order to reduce the Dimensionality of our Dataset.

```
['RatecodeID', 'store_and_fwd_flag', 'mta_tax', 'tolls_amount',
'improvement_surcharge','congestion_surcharge', 'airport_fee', 'dropoff_day',
'trip_distance', 'extra', 'fare_amount','total_amount']
```

We plotted the covariance matrix of the final selected features as well and we can notice that we have successfully removed sparse and collinear features as well.

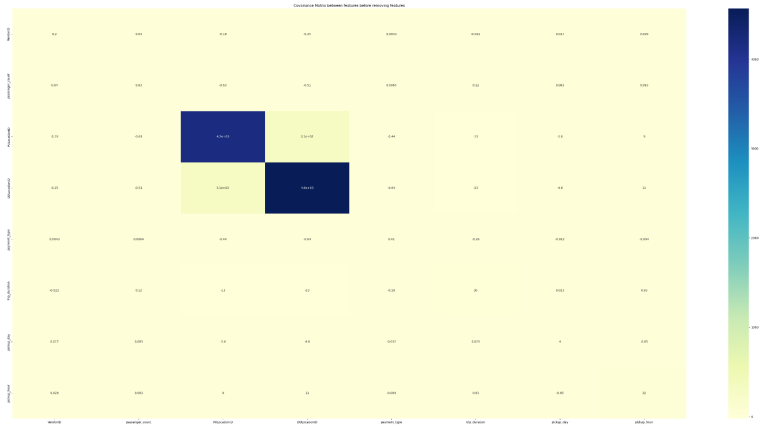

Fig 19. Heatmap for Covariance Matrix of Selected Features (Categorical Target)

Phase II - Regression Analysis

```
model = sm.OLS(y_train, X_train).fit()
```

We ran a regression analysis for our numerical Target - Total Amount. Our aim is to successfully predict the total amount of fare given by the passenger at the end of the trip given various features like pickup location, dropoff location, trip duration etc.

For this analysis we also perform some statistical analysis to explore how our model is performing based on p-values, t-statistics and F-statistics as well.

Stepwise Regression

We performed stepwise linear regression on the selected features to remove statistically insignificant features and build a better regression model. Following are the numerous iterations in which we removed features one by one while carefully analyzing all the features which will actually provide meaning to the final model.

OLS Regression Results

Dep. Variable:	total_amount	R-squared (uncentered):	0.812
Model:	OLS	Adj. R-squared (uncentered):	0.812
Method:	Least Squares	F-statistic:	1.914e+04
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	04:04:07	Log-Likelihood:	-23361.
No. Observations:	40000	AIC:	4.674e+04
Df Residuals:	39991	BIC:	4.682e+04
Df Model:	9		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
VendorID	0.2099	0.003	60.081	0.000	0.203	0.217
passenger_count	0.0004	0.002	0.186	0.852	-0.004	0.005
trip_distance	0.8547	0.002	389.965	0.000	0.850	0.859
PULocationID	0.0007	0.002	0.336	0.737	-0.004	0.005
DOLocationID	-0.0026	0.002	-1.193	0.233	-0.007	0.002
payment_type	-0.1752	0.002	-80.442	0.000	-0.179	-0.171
extra	0.2310	0.004	64.141	0.000	0.224	0.238
pickup_day	-0.0276	0.002	-12.503	0.000	-0.032	-0.023
pickup_hour	0.0320	0.002	13.735	0.000	0.027	0.037

Omnibus:	5960.909	Durbin-Watson:	2.011
Prob(Omnibus):	0.000	Jarque-Bera (JB):	17429.970
Skew:	0.795	Prob(JB):	0.00
Kurtosis:	5.816	Cond. No.	3.02

OLS Regression Results

Dep. Variable:	total_amount	R-squared (uncentered):	0.812
Model:	OLS	Adj. R-squared (uncentered):	0.812
Method:	Least Squares	F-statistic:	2.154e+04
Date:	Fri, 08 Dec 2023	Prob (F-statistic):	0.00
Time:	04:04:07	Log-Likelihood:	-23361.
No. Observations:	40000	AIC:	4.674e+04
Df Residuals:	39992	BIC:	4.681e+04
Df Model:	8		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
VendorID	0.2100	0.003	60.197	0.000	0.203	0.217
trip_distance	0.8547	0.002	389.983	0.000	0.850	0.859
PULocationID	0.0007	0.002	0.335	0.737	-0.004	0.005
DOLocationID	-0.0026	0.002	-1.194	0.233	-0.007	0.002
payment_type	-0.1752	0.002	-80.446	0.000	-0.179	-0.171
extra	0.2310	0.004	64.141	0.000	0.224	0.238
pickup_day	-0.0276	0.002	-12.510	0.000	-0.032	-0.023
pickup_hour	0.0321	0.002	13.746	0.000	0.027	0.037

Omnibus:	5960.666	Durbin-Watson:	2.011
Prob(Omnibus):	0.000	Jarque-Bera (JB):	17430.846
Skew:	0.795	Prob(JB):	0.00
Kurtosis:	5.816	Cond. No.	3.00

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           total_amount   R-squared (uncentered):                   0.812
Model:                            OLS   Adj. R-squared (uncentered):              0.812
Method:                 Least Squares   F-statistic:                          2.461e+04
Date:                Fri, 08 Dec 2023   Prob (F-statistic):                        0.00
Time:                        04:04:07   Log-Likelihood:                         -23361.
No. Observations:               40000   AIC:                                  4.674e+04
Df Residuals:                   39993   BIC:                                  4.680e+04
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
VendorID       0.2100      0.003     60.207      0.000       0.203       0.217
trip_distance  0.8547      0.002    390.421      0.000       0.850       0.859
DOLocationID  -0.0026      0.002     -1.176      0.240      -0.007       0.002
payment_type  -0.1752      0.002    -80.465      0.000      -0.179      -0.171
extra          0.2309      0.004     64.152      0.000       0.224       0.238
pickup_day    -0.0276      0.002    -12.533      0.000      -0.032      -0.023
pickup_hour    0.0321      0.002     13.755      0.000       0.027       0.037
==============================================================================
Omnibus:                     5960.326   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17428.440
Skew:                           0.795   Prob(JB):                         0.00
Kurtosis:                       5.816   Cond. No.                         3.00
==============================================================================
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:           total_amount   R-squared (uncentered):                   0.812
Model:                            OLS   Adj. R-squared (uncentered):              0.812
Method:                 Least Squares   F-statistic:                          2.872e+04
Date:                Fri, 08 Dec 2023   Prob (F-statistic):                        0.00
Time:                        04:04:07   Log-Likelihood:                         -23362.
No. Observations:               40000   AIC:                                  4.674e+04
Df Residuals:                   39994   BIC:                                  4.679e+04
Df Model:                           6
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
VendorID       0.2100      0.003     60.240      0.000       0.203       0.217
trip_distance  0.8549      0.002    391.851      0.000       0.851       0.859
payment_type  -0.1751      0.002    -80.468      0.000      -0.179      -0.171
extra          0.2310      0.004     64.193      0.000       0.224       0.238
pickup_day    -0.0276      0.002    -12.501      0.000      -0.032      -0.023
pickup_hour    0.0320      0.002     13.729      0.000       0.027       0.037
==============================================================================
Omnibus:                     5962.477   Durbin-Watson:                   2.011
Prob(Omnibus):                  0.000   Jarque-Bera (JB):            17434.460
Skew:                           0.795   Prob(JB):                         0.00
Kurtosis:                       5.816   Cond. No.                         3.00
==============================================================================
```

Summary of Model after dropping several Features					
AIC	BIC	Adjusted-R-Squared	Candidate Feature for Elimination	p_value of Candidate Feature	Dropped
46742.511	46828.477	0.812	const	0.928	None
46740.519	46817.889	0.812	passenger_count	0.852	'const'
46738.554	46887.327	0.812	PULocationID	0.737	'const' 'passenger_count'
46736.666	46796.843	0.812	DOLocationID	0.24	'const' 'passenger_count' 'PULocationID'
46736.05	46787.629	0.812	N/A	Adj R-Squared threshold exceeded for remaining features	'const' 'passenger_count' 'PULocationID' 'DOLocationID'

Fig 20. Stepwise Regression Model summary and removal of insignificant features

We can observe from the above figure that we had a fairly consistent value of Adjusted-R-Squared which is 0.812. We iteratively have removed the features with highest p-values as they pose to be the features with highest randomness. The AIC and BIC can both be observed to decrease by each iteration and this supports a more 'Parsimonious' data. These values indicate that our final data is less prone to Overfitting. An interesting observation that can be made is that 'const' is being dropped and the reason is that firstly it has a very high p-value contributing towards randomness. This also shows that our model doesn't have any intercept and we will see that in the equation of our model as well.

```
Stepwise Regression Equation:
total_amount = 0.000 + 0.210 * VendorID + 0.855 * trip_distance
-0.175 * payment_type + 0.231 * extra -0.028 * pickup_day + 0.032 *
pickup_hour
```

Test vs Predicted Plots

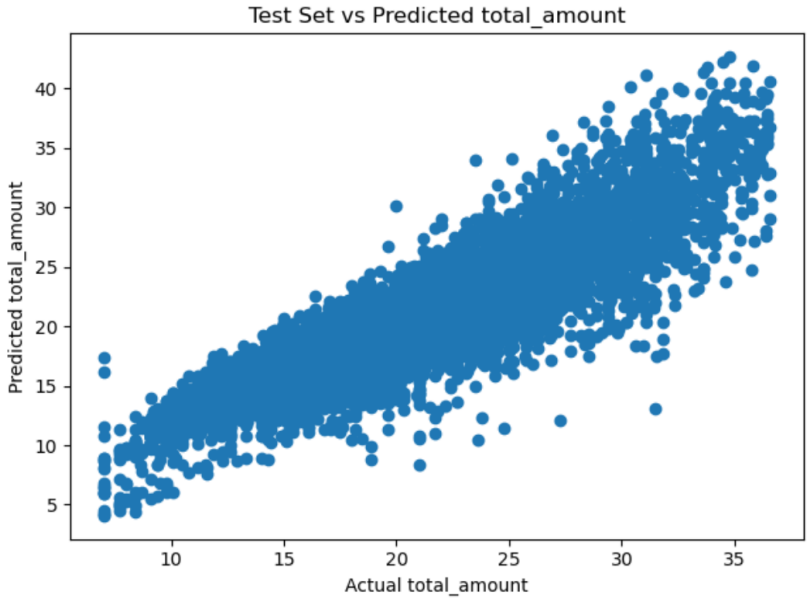

Fig 21. Stepwise Regression Model - Scatterplot (test vs predicted)

We can see that the scatter plot is following a linear trend showing that we can fit the regression model in a linear equation very easily.

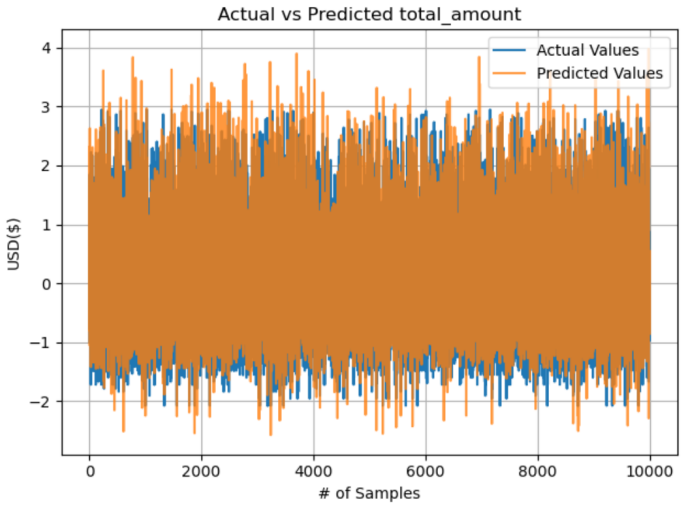

Fig 22. Actual vs Predicted total_amount for each test sample

The graph appears to be completely overlapping showing a good fit and we will also see this result in a quantitative value further as well.

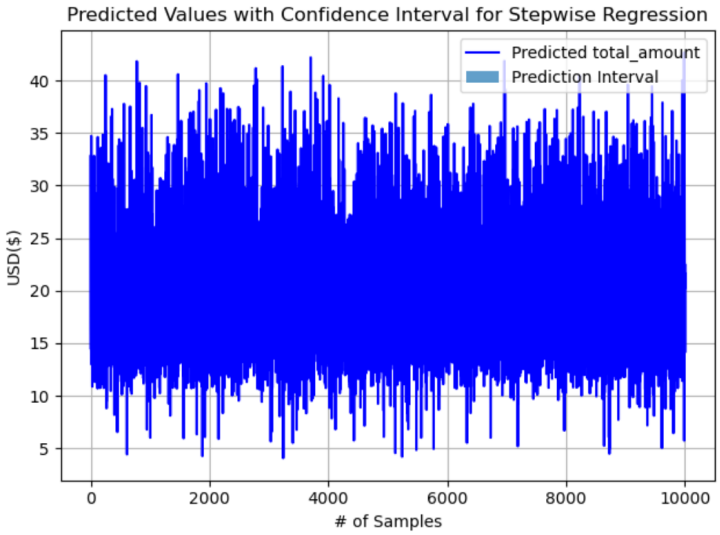

Fig 23. Confidence Interval analysis on Predicted Values

A good observation that can be made on this graph is that the confidence interval is very low. This has its own pros and cons. The pros are that it resembles higher precision for our predictions and we can put more confidence in our observations because of less uncertainty. The cons are that the range of possible observations is narrowed and it can also resemble a possibility of overfitting. But we will see with our statistical analysis later on that the possibility of overfitting is very low.

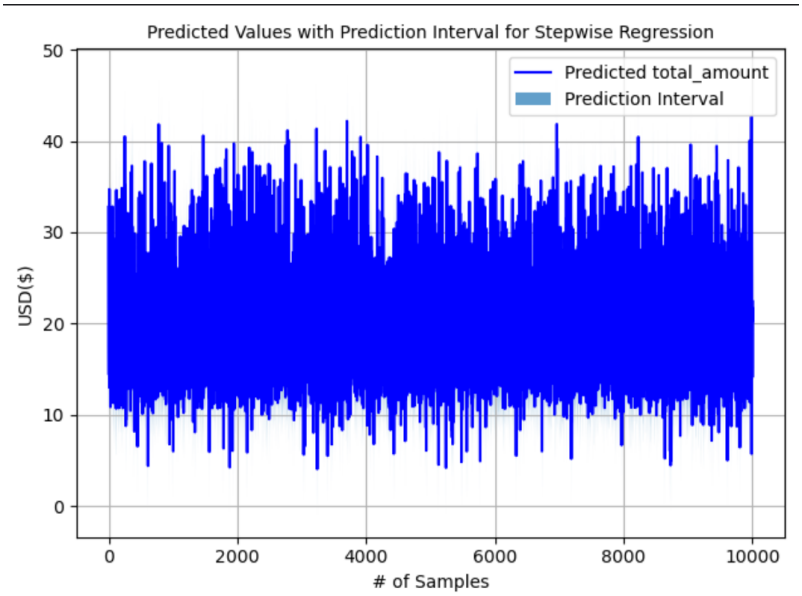

Fig 24. Prediction Interval analysis on Predicted Values

Similar observations as of the previous graph can be made about this graph. We can say that the model is prone to overfitting and has a narrower range of possible outcomes but again the pros are that we have a higher precision and have a better discrimination of values and also a better confidence on our predictions.

	Actual	Predicted
19616	-0.734	-0.798
27166	0.159	0.139
7373	1.829	2.321
37034	1.849	1.350
3372	-0.293	0.117

Fig 25. Snippet of output showing Actual vs Predicted Values

Statistical Analysis

For this statistical analysis we will refer to Fig 20. In figure 20, we are able to see that the final model has following statistic

F-statistic:	2.872e+04

t	P>|t|
60.240	0.000
391.851	0.000
-80.468	0.000
64.193	0.000
-12.501	0.000
13.729	0.000

Fig 26. F-statistic, t-statistic and p-values of final model

The final model has a high F-statistic meaning there are significant differences between the remaining features. So, our final model has no collinearity whatsoever.

The magnitude of t-statistic is high for all the features meaning greater difference between the sample mean and the population mean.

For all the features the p-value is 0, this means the observed values are highly unlikely to be random.

Statistics of Stepwise Regression Model					
Model	AIC	BIC	R-Squared	Adjusted-R-Squared	MSE
Backward Stepwise Regression	46738.041	46798.218	0.812	0.812	0.188

Fig 27. R-squared, Adjusted R-Squares, MSE of regression model

We see that MSE is very low (0.188) for our model indicating a good fit and we can expect our predictions to be more accurate.

Both R-squared and Adjusted R-Squared have a high value of 0.812. This means that 81.2% of variance in total_amount can be explained by the selected independent features.

Phase III - Classification Analysis

The classification analysis has been done on the target ‘tip given’. Our motive behind the prediction of ‘tip given’ surrounds the following ideas.

- Customer Satisfaction - tip given is an indicator of customer satisfaction which is a huge part of the service industry.
- Driver Performance Evaluation - if the driver is getting tips it means that the performance of the driver is good or we can correlate tipping with the performance of our driver.
- Incentive Structures - by analyzing the tipping patterns the taxi vendors can incentivize certain behaviors by the drivers so as to make profits on both ends (i.e. as a vendor and as a driver as well)
- Service Quality Improvement
- Pricing Strategy - tipping patterns can show us how to set fares for the rides. If the customer is avoiding tips we might associate it with the high fare prices as well.

For the classification analysis we have employed a grid search parameter for all the classifiers except Naïve Bayes as the whole model is based on probability and doesn’t have any hyper parameters to tune. We will look at the ROC, AUC and the confusion matrix for each classifier and evaluate the stratified k-fold cross validation as well. At the end of the section we will find a comparison for each classifier and we will declare the best classifier for our classification.

Decision Trees (Pre Pruning)

We run a simple classification model for our target using Decision Trees but we also employed Hyper Parameter Tuning in this model. This tuning helped us determine which parameters were optimal for finding the highest accuracy.

```
clf = DecisionTreeClassifier(random_state=5805)
clf.fit(X_train, y_train)

tuned_parameters = {
    'max_depth': [3, 5, 8],
    'min_samples_split': [10, 12, 13],
    'min_samples_leaf': [1, 2, 4],
    'max_features': [2, 3, 4],
    'splitter': ['best', 'random'],
    'criterion': ['gini', 'entropy', 'log_loss']
}

grid_search = GridSearchCV(clf, tuned_parameters, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)
best_clf = grid_search.best_estimator_
print("Best Parameters (pre pruning): ", grid_search.best_params_)
print("Best Accuracy (using grid search/ pre pruning): ", grid_search.best_score_.round(2))

best_clf.fit(X_train, y_train)
```

```
Best Parameters (pre pruning):  {'criterion': 'gini', 'max_depth': 3, 'max_features': 4, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
```

```
Best Accuracy (using grid search/ pre pruning):  0.94
Accuracy on the test set (with the best model/ pre pruning): 0.94
+----------------------+
| k = 5 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
|   4    |     0.94    |
|   5    |     0.94    |
+--------+-------------+
Pre-pruned Decision Tree classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94% as well.

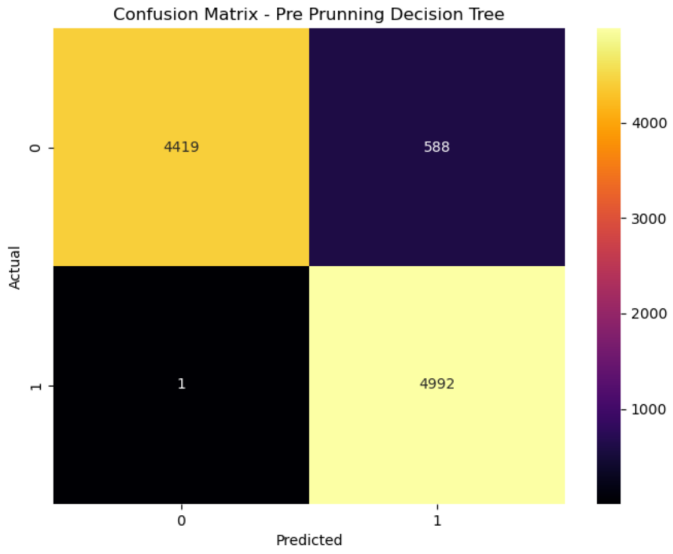

Fig 28. Confusion Matrix - Pre Pruning Decision Tree

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

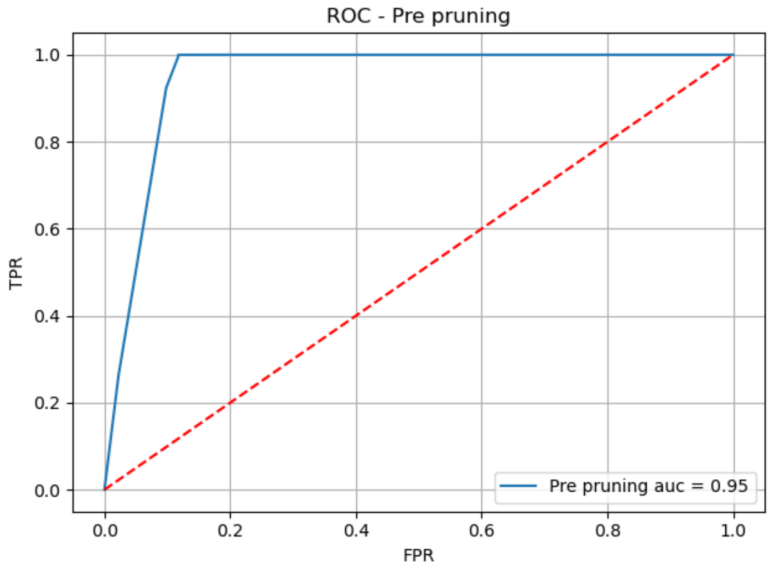

Fig 29. ROC and AUC for Pre Pruning

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Our Final Decision Tree looks like it has a fairly simple shape.

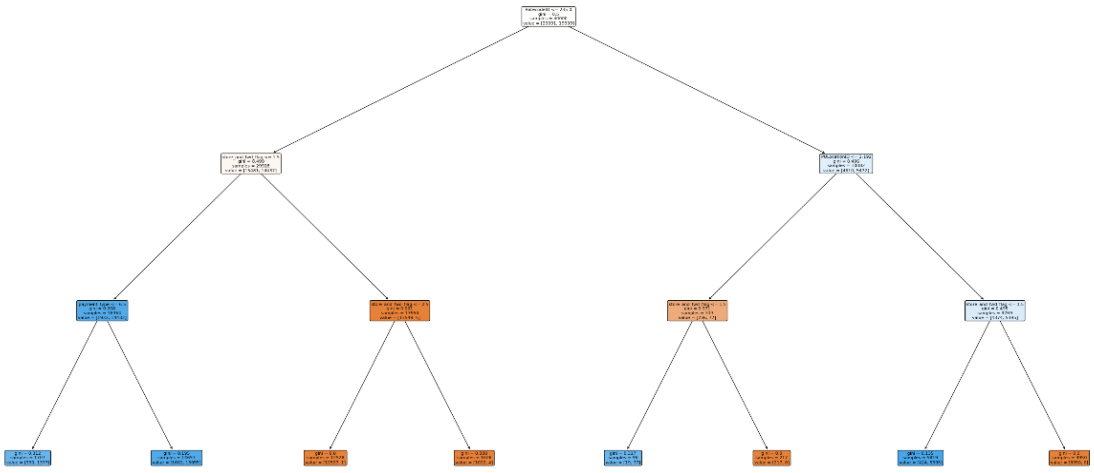

Fig 30. Pre Pruned Decision Tree

Decision Trees (Post Pruning)

We ran a simple classification model for our target using Decision Trees and found alphas which will help us determine which tree to follow after we make the whole decision tree.

```
path = clf.cost_complexity_pruning_path(X_train, y_train)
alphas = path['ccp_alphas']

accuracy_train, accuracy_test = [], []
for i in alphas:
    clf_x = DecisionTreeClassifier(random_state=5805, ccp_alpha=i)
    clf_x.fit(X_train, y_train)
    y_train_pred = clf_x.predict(X_train)
    y_test_pred = clf_x.predict(X_test)
    accuracy_train.append(accuracy_score(y_train, y_train_pred))
    accuracy_test.append(accuracy_score(y_test, y_test_pred))
```

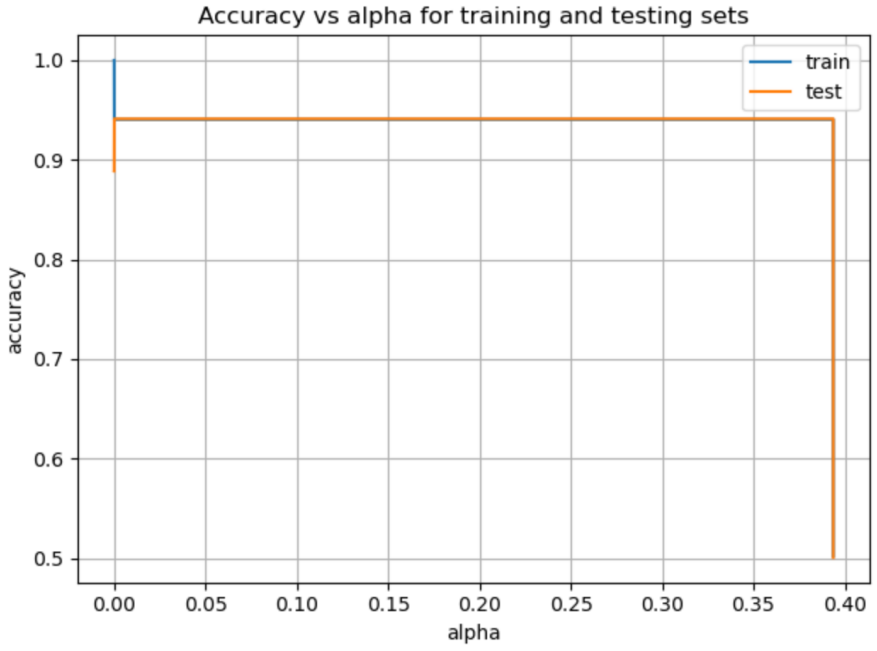

Fig 31. Accuracy vs Alphas for training and test set

We observe that the optimum alpha is obtained very early at around 0.00013

```
Max alpha =  0.00013
```

```
Train accuracy (post pruning)  0.94
Test accuracy (post pruning)  0.94
+---------------------+
| k = 5 fold accuracy  |
+--------+------------+
|  Fold  |   Accuracy  |
+--------+------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
|   4    |     0.94    |
|   5    |     0.94    |
+--------+------------+
Post-pruned Decision Tree classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94% as well.

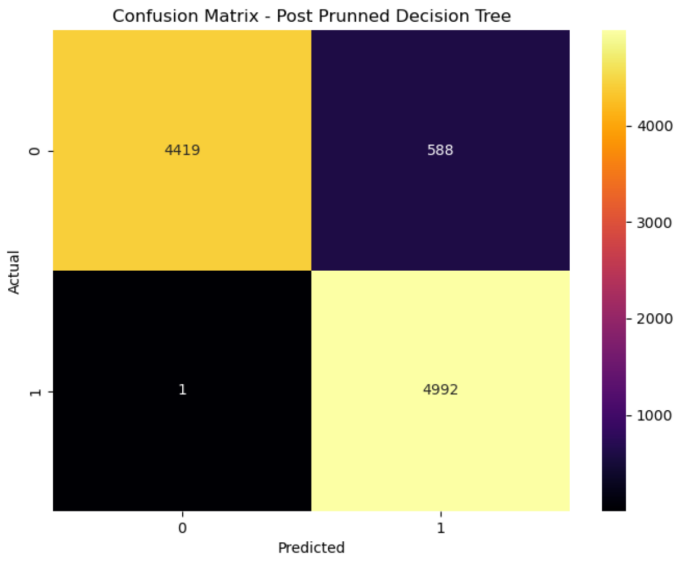

Fig 32. Confusion Matrix - Post Pruned Decision Tree

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

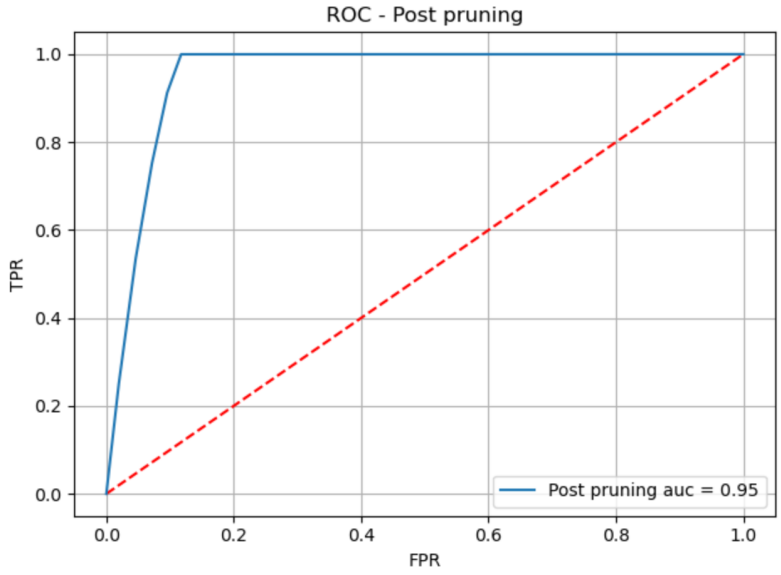

Fig 33. ROC and AUC for Post Pruning

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Our Final Decision Tree looks like it has a fairly simple shape.

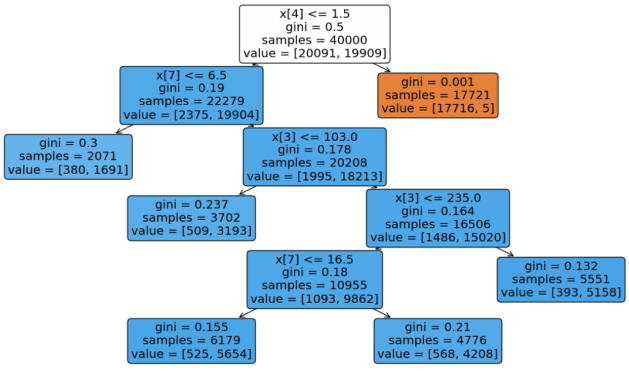

Fig 34. Post Pruned Decision Tree

Logistic Regression

We ran a simple classification model for our target using Logistic Regression and found the hyper parameters using grid search which will help us determine which value of C and which penalty will give us the best results.

```
model_logistic = LogisticRegression()

param_grid = {'C': [0.001, 0.01, 0.1, 1, 10, 100],
              'penalty': ['l1', 'l2']}

grid_search = GridSearchCV(model_logistic, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (Logistic Regression): ", grid_search.best_params_)

best_model_logistic = grid_search.best_estimator_
y_pred_logistic = best_model_logistic.predict(X_test)
cnf_matrix_logistic = confusion_matrix(y_test, y_pred_logistic)
y_proba_logistic = best_model_logistic.predict_proba(X_test)[:, 1]
logistic_fpr, logistic_tpr, _ = roc_curve(y_test, y_proba_logistic)
auc_logistic = roc_auc_score(y_test, y_proba_logistic)
accuracy_logistic = accuracy_score(y_test, y_pred_logistic)
```

```
Best Hyperparameters (Logistic Regression):  {'C': 0.001, 'penalty': 'l2'}
```

k = 5 fold accuracy	
Fold	Accuracy
1	0.93
2	0.93
3	0.93
4	0.93
5	0.93

Logistic Regression classifier, Accuracy Mean = 0.93

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 93% as well.

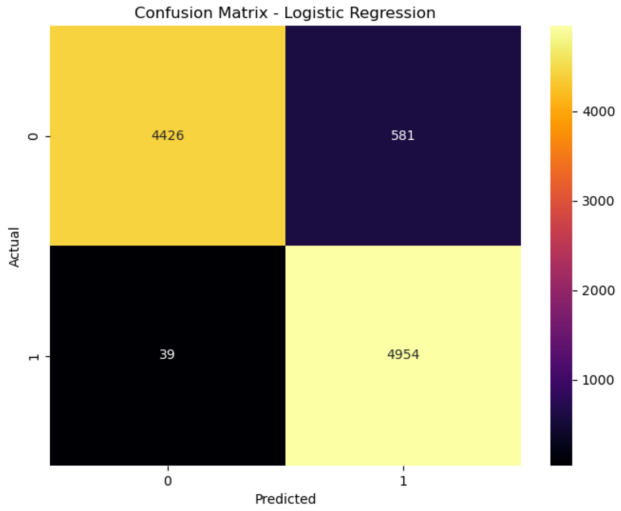

Fig 35. Confusion Matrix - Logistic Regression

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

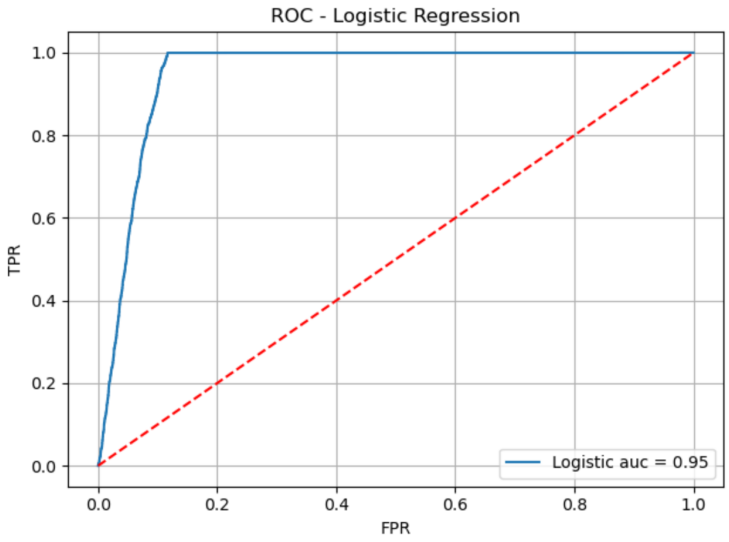

Fig 36. ROC and AUC for Logistic Regression

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

KNN

We ran a simple classification model for our target using KNN and found the hyper parameters (best k) using grid search. Also, we have standardized the dataset because we want to avoid the curse of dimensionality in our case.

```
k_range = list(range(1, 21))
param_grid = dict(n_neighbors=k_range)
grid = GridSearchCV(model, param_grid, cv=5, scoring='accuracy',
                    return_train_score=True, verbose=1)
grid.fit(X_train, y_train)

results = grid.cv_results_
k_values = results['param_n_neighbors'].data
error_rates = 1 - results['mean_test_score']
```

```
Best k: 17
```

```
+----------------------+
| k = 5 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.62    |
|   2    |     0.61    |
|   3    |     0.62    |
|   4    |     0.62    |
|   5    |     0.62    |
+--------+-------------+
KNN classifier, Accuracy Mean =  0.62
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows an inconsistent result of 62%. The reason for this drop might be the selection of k which must have caused overfitting in our classifier.

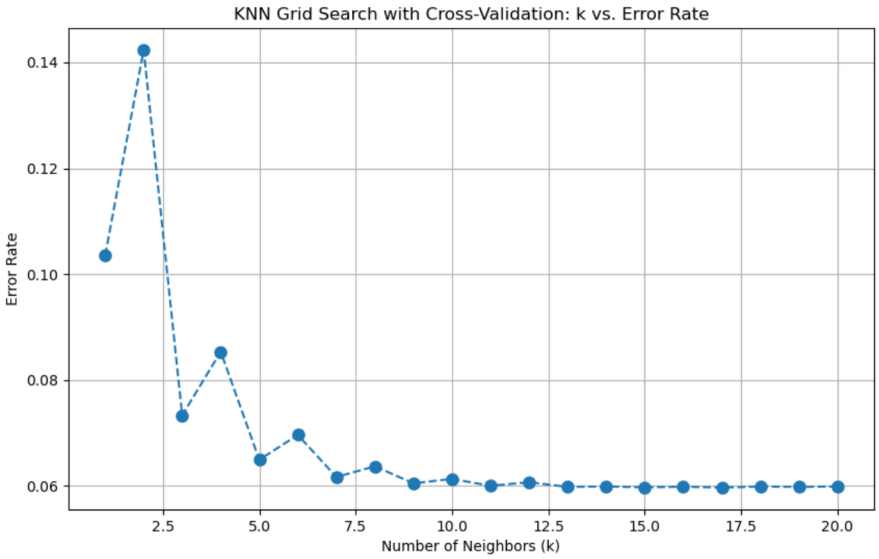

Fig 37. Error rate vs Number of Neighbors (k)

We can see that after 17 neighbors the error rate becomes stable, this indicates that the model is less prone to overfitting.

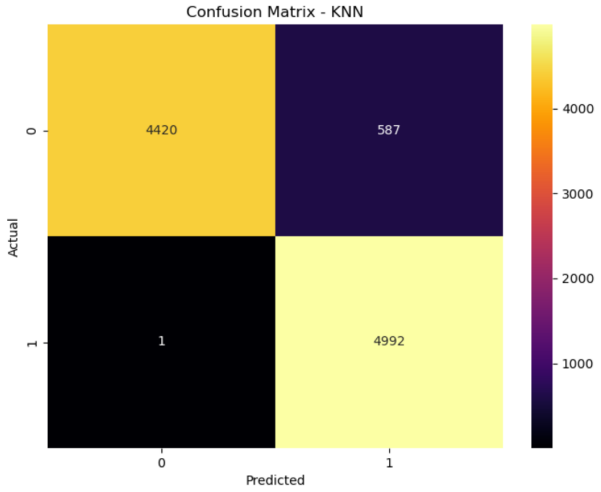

Fig 38. Confusion Matrix - KNN Classifier

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

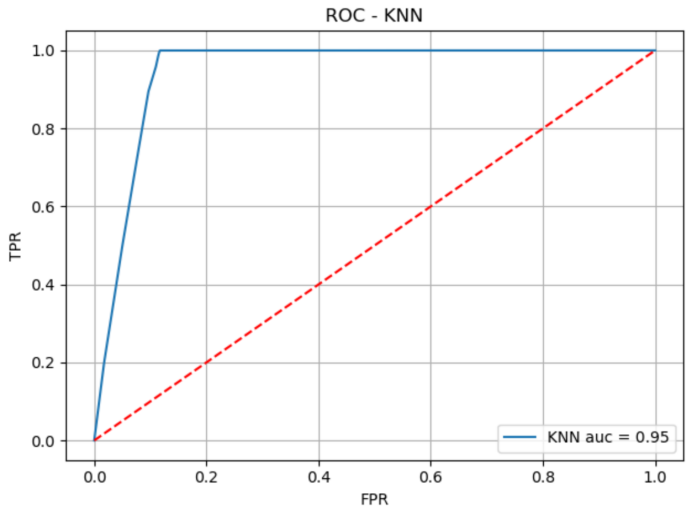

Fig 39. ROC and AUC for KNN Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Support Vector Machine - Linear Kernel

We ran a simple classification model for our target using SVM and chose our kernel as ‘linear’ and found the hyper parameters using grid search. Also, we have standardized the dataset because we want to avoid the curse of dimensionality in our case.

```
model_svm_linear = SVC(kernel='linear', probability=True)

param_grid = {'C': [0.01, 1, 10]}

grid_search = GridSearchCV(model_svm_linear, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (SVM Linear): ", grid_search.best_params_)

best_model_svm_linear = grid_search.best_estimator_
y_pred_svm_linear = best_model_svm_linear.predict(X_test)
```

```
Best Hyperparameters (SVM Linear):  {'C': 0.01}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
+--------+-------------+
SVM (linear) classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94%. We are only running a 3-fold cross validation as SVM in general is very compute intensive.

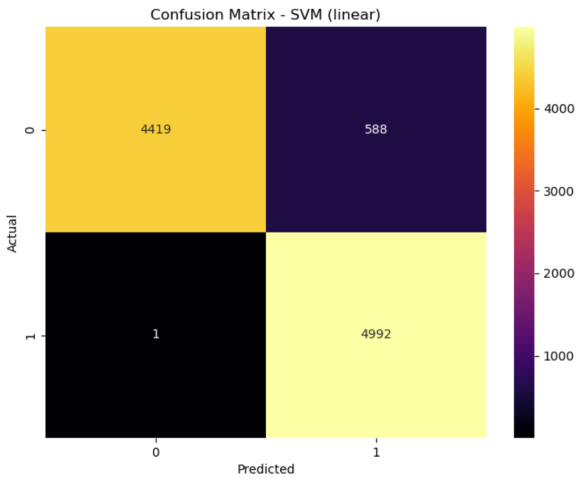

Fig 40. Confusion Matrix - SVM (linear) Classifier

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

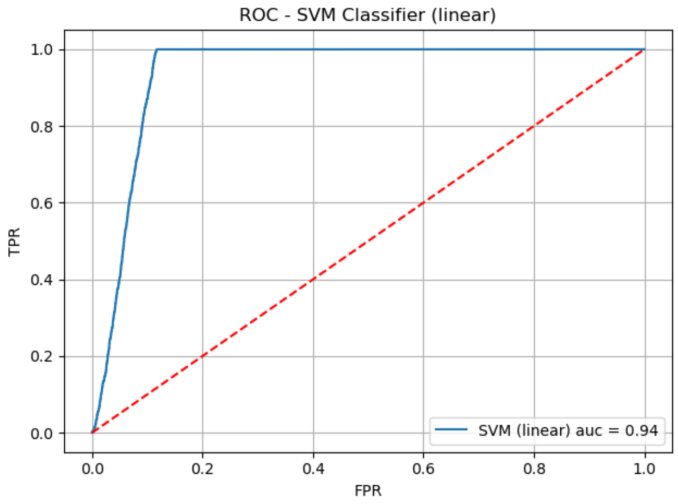

Fig 41. ROC and AUC for SVM (linear) Classifier

The ROC is straightforward and the AUC is 0.94 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Support Vector Machine - Polynomial Kernel

We ran a simple classification model for our target using SVM and chose our kernel as 'polynomial' and found the hyper parameters using grid search. Also, we have standardized the dataset because we want to avoid the curse of dimensionality in our case.

```
model_svm_poly = SVC(kernel='poly', probability=True)

param_grid = {'C': [1, 2, 4],
              'degree': [2, 3, 4]}

grid_search = GridSearchCV(model_svm_poly, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (SVM Polynomial): ", grid_search.best_params_)

best_model_svm_poly = grid_search.best_estimator_
y_pred_svm_poly = best_model_svm_poly.predict(X_test)
```

```
Best Hyperparameters (SVM Polynomial):  {'C': 1, 'degree': 3}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
+--------+-------------+
SVM (poly) classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94%. We are only running a 3-fold cross validation as SVM in general is very compute intensive.

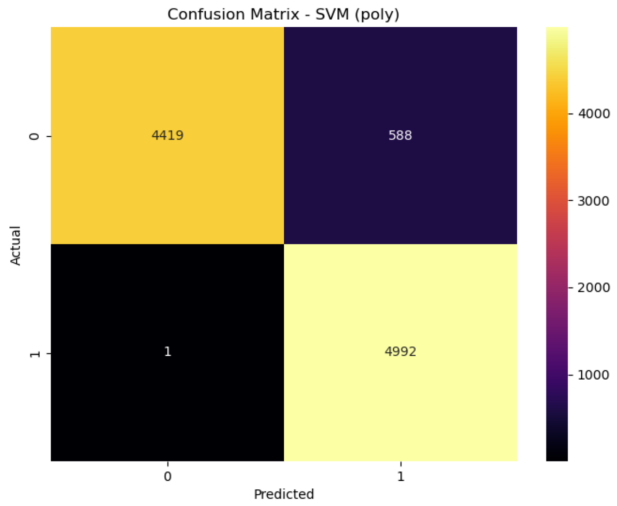

Fig 42. Confusion Matrix - SVM (polynomial) Classifier

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

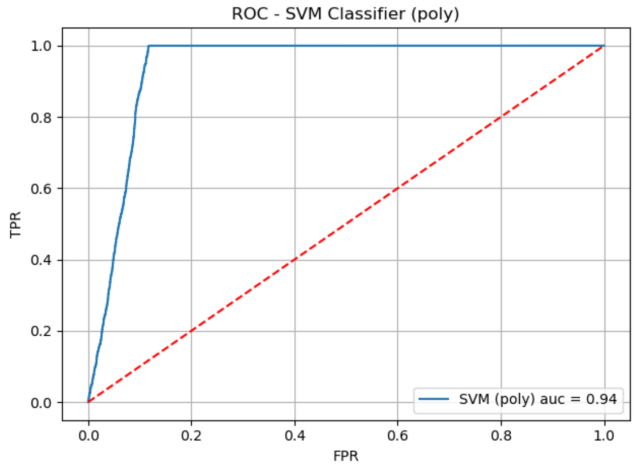

Fig 43. ROC and AUC for SVM (polynomial) Classifier

The ROC is straightforward and the AUC is 0.94 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Support Vector Machine - Radial Basis Kernel

We ran a simple classification model for our target using SVM and chose our kernel as ‘rbf’ and found the hyper parameters using grid search. Also, we have standardized the dataset because we want to avoid the curse of dimensionality in our case.

```
model_svm_rbf = SVC(kernel='rbf', probability=True)

param_grid = {'C': [0.1, 1, 10],
              'gamma': ['scale', 'auto']}

grid_search = GridSearchCV(model_svm_rbf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (SVM rbf): ", grid_search.best_params_)
```

```
Best Hyperparameters (SVM rbf):  {'C': 0.1, 'gamma': 'scale'}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
+--------+-------------+
SVM (rbf) classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94%. We are only running a 3-fold cross validation as SVM in general is very compute intensive.

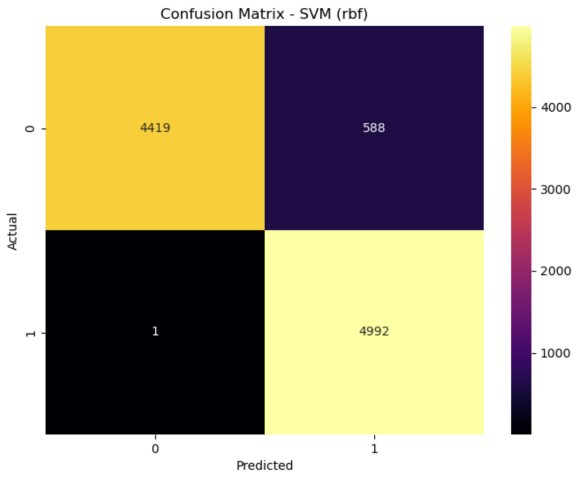

Fig 44. Confusion Matrix - SVM (rbf) Classifier

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

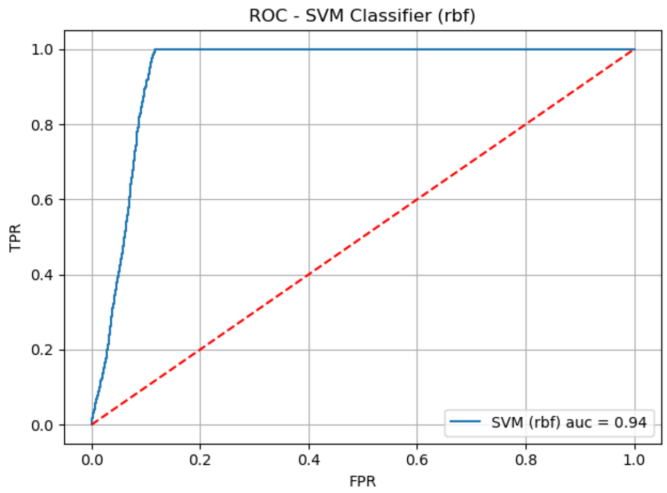

Fig 45. ROC and AUC for SVM (rbf) Classifier

The ROC is straightforward and the AUC is 0.94 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Naïve Bayes Classifier

We ran a simple classification model for our target using Naïve Bayes.

```
model_naive = GaussianNB()
model_naive.fit(X_train, y_train)

y_pred_naive = model_naive.predict(X_test)
```

```
+----------------------+
| k = 5 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
|   4    |     0.94    |
|   5    |     0.94    |
+--------+-------------+
Naive Bayes classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94%.

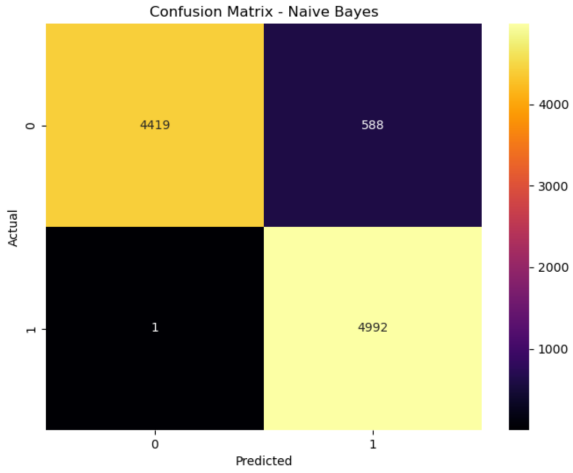

Fig 46. Confusion Matrix - Naïve Bayes Classifier

The confusion matrix contains only one false positive and the amount of false negatives is very less as well.

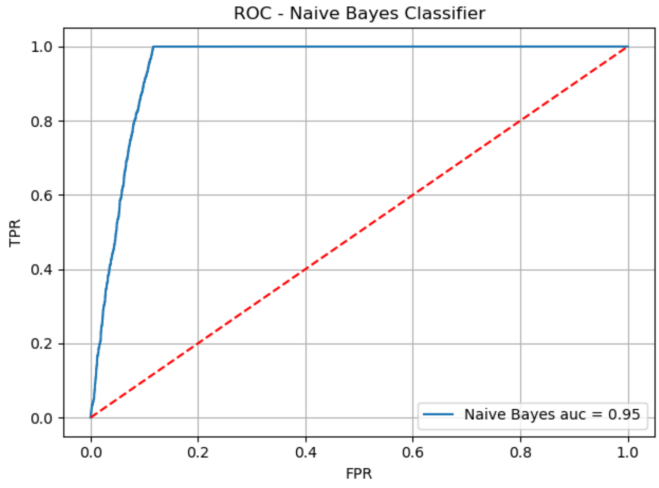

Fig 47. ROC and AUC for Naïve Bayes Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Random Forest Classifier - Bagging

We ran a combined classification model for our target using Random Forest and Bagging technique and found the hyper parameters using grid search.

```
base_classifier = RandomForestClassifier(random_state=5805)

model_bagging = BaggingClassifier(base_classifier, random_state=5805)

param_grid = {
    'base_estimator__n_estimators': [10, 15, 20],
    'n_estimators': [10, 15, 20],
    'max_samples': [0.5, 0.7, 1.0],
    'max_features': [0.5, 0.7, 1.0]
}

grid_search = GridSearchCV(model_bagging, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (Random Forest Bagging): ", grid_search.best_params_)

best_model_bagging = grid_search.best_estimator_
y_pred_bagging = best_model_bagging.predict(X_test)
```

```
Best Hyperparameters (Random Forest Bagging):  {'base_estimator__n_estimators': 20, 'max_features': 1.0, 'max_samples': 0.5, 'n_estimators': 15}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
+--------+-------------+
Random Forest (Bagging) classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent result of 94%. We are only running a 3-fold cross validation as Random Forest in general is very compute intensive.

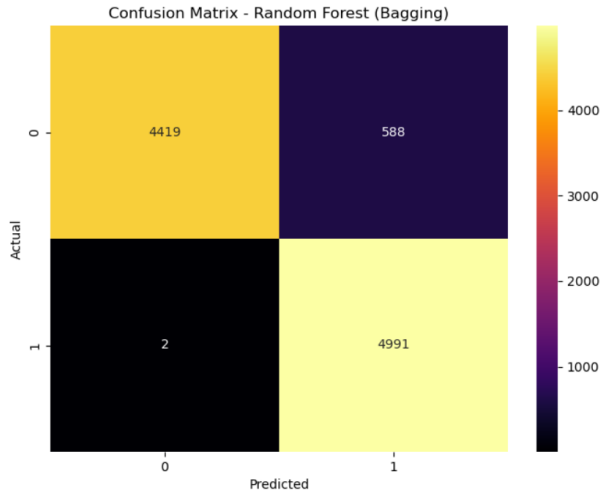

Fig 48. Confusion Matrix - Random Forest (Bagging) Classifier

The confusion matrix contains only two false positives and the amount of false negatives is very less as well.

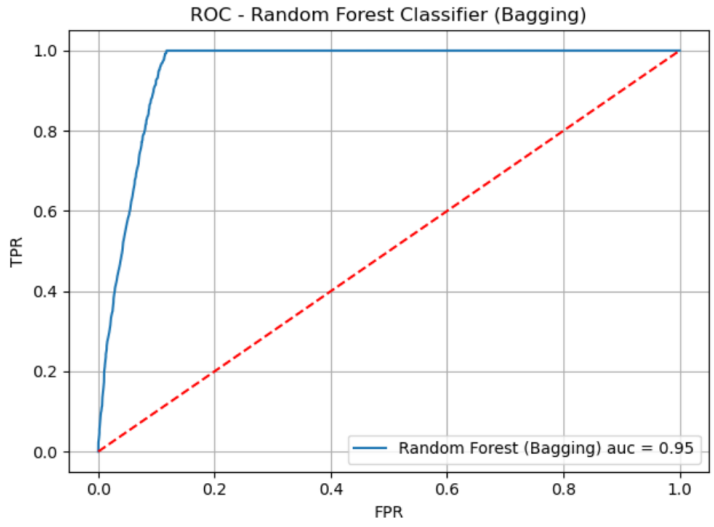

Fig 49. ROC and AUC for Random Forest (Bagging) Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Random Forest Classifier - Stacking

We ran a combined classification model for our target using Random Forest and Stacking technique. We stacked two simple classifiers namely Logistic Regression and KNN and found the hyper parameters using grid search.

```
base_classifiers = [
    ('log', LogisticRegression()),
    ('knn', KNeighborsClassifier(n_neighbors=3))
]

model_stacking = StackingClassifier(
    estimators=base_classifiers,
    final_estimator=RandomForestClassifier(random_state=5805)
)

param_grid = {
    'final_estimator__n_estimators': [10, 15, 20],
    'final_estimator__max_depth': [5, 10],
    'final_estimator__min_samples_split': [2, 5, 10]
}

grid_search = GridSearchCV(model_stacking, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (Random Forest Stacking): ", grid_search.best_params_)
```

```
Best Hyperparameters (Random Forest Stacking):  {'final_estimator__max_depth': 5, 'final_estimator__min_samples_split': 10, 'final_estimator__n_estimators': 10}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.89    |
|   2    |     0.89    |
|   3    |     0.89    |
+--------+-------------+
Random Forest (stacking) classifier, Accuracy Mean =  0.89
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a lesser average accuracy of 89%. We are only running a 3-fold cross validation as Random Forest in general is very compute intensive.

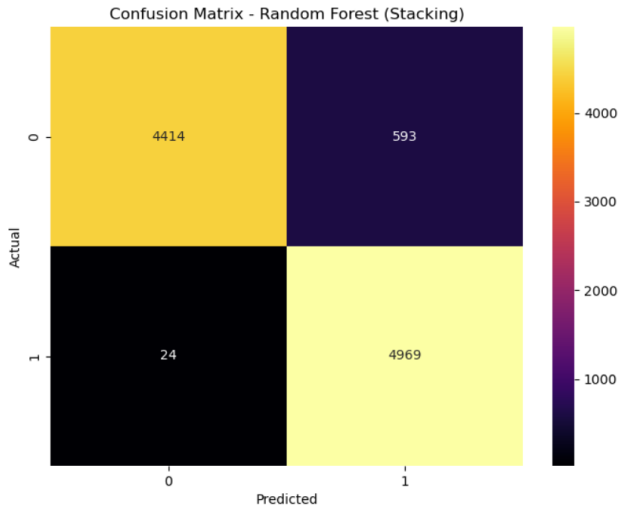

Fig 50. Confusion Matrix - Random Forest (Stacking) Classifier

The confusion matrix contains only 24 false positives and the amount of false negatives is very less as well.

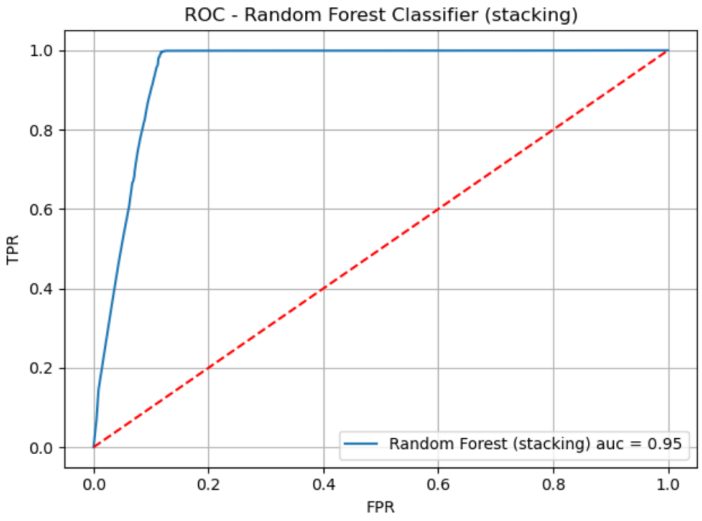

Fig 51. ROC and AUC for Random Forest (Stacking) Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Random Forest Classifier - Boosting

We employed boosting using the AdaBoost classification model for our target in our Random Forest technique. Subsequently, we found the hyper parameters using grid search.

```
base_classifier = RandomForestClassifier(random_state=5805)

model_boosting = AdaBoostClassifier(base_estimator=base_classifier, random_state=5805)

param_grid = {
    'base_estimator__n_estimators': [10, 15, 20],
    'n_estimators': [10, 15, 20],
    'learning_rate': [0.1, 0.5, 1.0]
}

grid_search = GridSearchCV(model_boosting, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (Random Forest Boosting): ", grid_search.best_params_)

best_model_boosting = grid_search.best_estimator_
y_pred_boosting = best_model_boosting.predict(X_test)
```

```
Best Hyperparameters (Random Forest Boosting):  {'base_estimator__n_estimators': 20, 'learning_rate': 0.1, 'n_estimators': 15}
```

```
+----------------------+
| k = 3 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
+--------+-------------+
Random Forest (boosting) classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent 94% average accuracy. We are only running a 3-fold cross validation as Random Forest in general is very compute intensive.

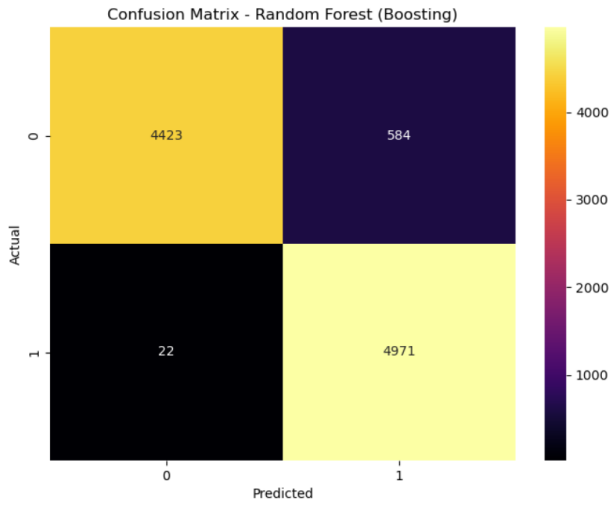

Fig 52. Confusion Matrix - Random Forest (Boosting) Classifier

The confusion matrix contains only 22 false positives and the amount of false negatives is very less as well.

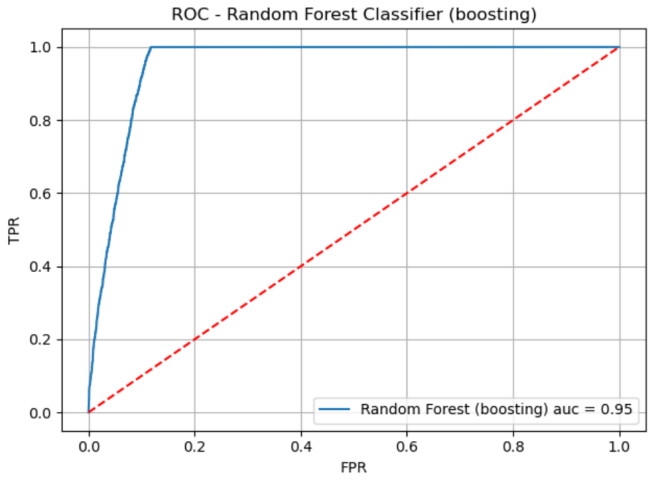

Fig 53. ROC and AUC for Random Forest (Boosting) Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Neural Networks - Multi Layered Perceptron

We employed the Neural Networks using a Multi Layered Perceptron to classify our target and we found the hyper parameters using grid search.

```
model_neural = MLPClassifier(random_state=5805)

param_grid = {
    'hidden_layer_sizes': [(64, 32), (32, 16), (16, 8)],
    'max_iter': [50, 100, 200],
    'alpha': [0.0001, 0.001, 0.01],
    'learning_rate_init': [0.001, 0.01, 0.1]
}

grid_search = GridSearchCV(model_neural, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X_train, y_train)

print("Best Hyperparameters (Neural Network): ", grid_search.best_params_)

best_model_neural = grid_search.best_estimator_
y_pred_neural = best_model_neural.predict(X_test)
```

```
Best Hyperparameters (Neural Network):  {'alpha': 0.0001, 'hidden_layer_sizes': (16, 8), 'learning_rate_init': 0.01, 'max_iter': 100}
```

```
+----------------------+
| k = 5 fold accuracy  |
+--------+-------------+
|  Fold  |   Accuracy  |
+--------+-------------+
|   1    |     0.94    |
|   2    |     0.94    |
|   3    |     0.94    |
|   4    |     0.94    |
|   5    |     0.94    |
+--------+-------------+
Neural Network classifier, Accuracy Mean =  0.94
```

The Accuracy of our model is 94% and when we employ stratified k-fold cross validation it shows a consistent 94% average accuracy.

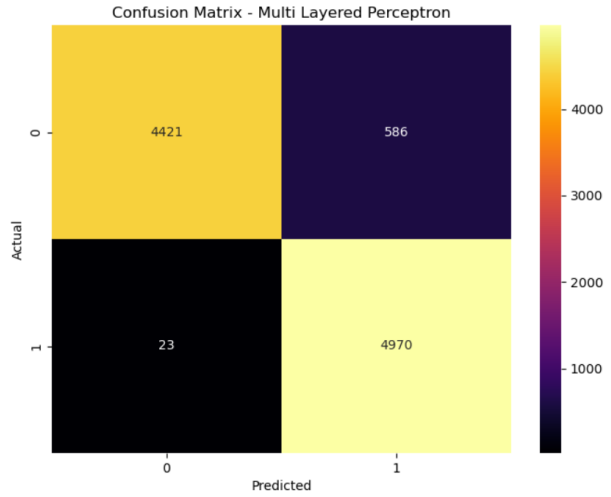

Fig 54. Confusion Matrix - MLP Classifier

The confusion matrix contains only 22 false positives and the amount of false negatives is very less as well.

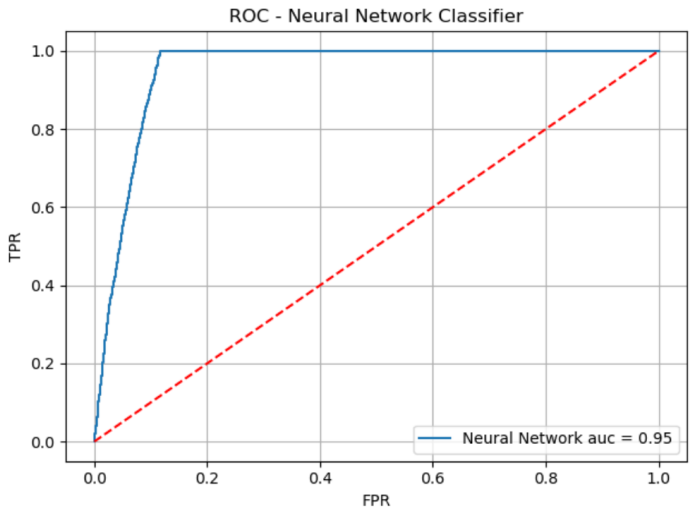

Fig 55. ROC and AUC for MLP Classifier

The ROC is straightforward and the AUC is 0.95 which shows a good fit and also shows that the model is a good distinguisher between our two classes.

Model Summary and Best Classifier

Model Comparison Table

Model	True Negative	False Positive	False Negative	True Positive	Accuracy	Recall	Specificity	F1 Score	AUC
Pre pruning Decision Tree	4419	588	1	4992	0.94	1.00	0.88	0.94	0.95
Post pruning Decision Tree	4419	588	1	4992	0.94	1.00	0.88	0.94	0.95
Logistic Regression	4426	581	39	4954	0.94	0.99	0.88	0.94	0.95
KNN	4420	587	1	4992	0.94	1.00	0.88	0.94	0.95
SVM Classifier (linear)	4419	588	1	4992	0.94	1.00	0.88	0.94	0.94
SVM Classifier (poly)	4419	588	1	4992	0.94	1.00	0.88	0.94	0.94
SVM Classifier (rbf)	4419	588	1	4992	0.94	1.00	0.88	0.94	0.94
Naive Bayes Classifier	4419	588	1	4992	0.94	1.00	0.88	0.94	0.95
Bagging Random Forest Classifier	4419	588	2	4991	0.94	1.00	0.88	0.94	0.95
Stacking Random Forest Classifier	4414	593	24	4969	0.94	1.00	0.88	0.94	0.95
Boosting Random Forest Classifier	4423	584	22	4971	0.94	1.00	0.88	0.94	0.95
Neural Network Classifier	4421	586	23	4970	0.94	1.00	0.88	0.94	0.95

Table 1 - Model Comparison Table

We can see that all the classifiers are performing well in classifying if the tip is given or not. We can see that AUC, accuracy, recall, specificity and even F1 score for each of our models are almost identical to each other making us believe that we can select any model to classify our target with greater efficiency. But, we must not forget the results that we gained from cross validations. Below is a summary of Average mean gained from Cross Validations.

Summary of Average Means from Cross Validations	
Model Name	**Average Cross Validated Accuracy**
Decision Tree (Pre Pruned)	0.94
Decision Tree (Post Pruned)	0.94
Logistic Regression	0.93
KNN	0.62

SVM (linear)	0.94
SVM (polynomial)	0.94
SVM (rbf)	0.94
Naïve Bayes	0.94
Random Forest (Bagging)	0.94
Random Forest (Stacking)	0.89
Random Forest (Boosting)	0.94
Neural Networks	0.94

Table 2 - Model Comparison Based on Average Cross Validated Accuracies

We can clearly notice that upon cross validation a few models show lesser average accuracy upon cross validation. Models like Random Forest (Stacking) and KNN fail to be a robust model for all the scenarios. All the remaining models are almost identical to each other in terms of cross validated accuracy.

Now, let us focus on our target to determine which factor influences our model selection. Our target is ‘tip given’ and it needs different perspectives to evaluate our model and predict if we need it or not. If we are the taxi vendors/drivers our goal will be to maximize True Positives as the vendors/drivers will want more income. As a marketing team as well, a team can focus on True Positives so as to market their products to passengers who give tips (electronically) and incentivize their gratitude. Something similar can be observed for True Negatives. If we focus on True Negatives we can avoid unnecessary interventions to the riders and thus support customer satisfaction as well. And if a marketing team wants to avoid resource wastage on people who do not tip (i.e. incentivizing people who do not tip) the team can focus on True Negatives as well. Based on these assumptions we can summarize the following for model selection:

- **Naïve Bayes** is a good candidate for the best model if we want to focus on True Positives as we might also notice that AUC is 0.95 for this classifier and we need a good discriminator for identifying tippers.
- **Random Forest (Boosting)** is a good candidate for the best model if we want to focus on True Negatives and we can also notice the AUC is again 0.95 for this classifier implying that it is a good discriminator for identifying NON-tippers.

Phase IV - Clustering and Association Analysis

In this phase we will focus on applying our data to unsupervised algorithms and try to notice patterns based on it. Our motive for this analysis is to gather data about certain behaviors which is not explicitly observable.

K-Mean Clustering

For this section we ran the K-means algorithms in two parts. The first part analyzes the data using the elbow method and prints a Within-Cluster Variation Plot for identifying the number of clusters inside the data. The second part runs Silhouette Analysis to identify for which number of clusters we will get the highest Silhouette score.

Elbow Method

```
from sklearn.cluster import KMeans

1 usage
def calculate_WSS(points, kmax):
    sse = []
    for k in range(1, kmax + 1):
        kmeans = KMeans(n_clusters=k).fit(points)
        centroids = kmeans.cluster_centers_
        pred_clusters = kmeans.predict(points)
        curr_sse = 0

        # Calculate squared Euclidean distance for each point from its cluster center
        for i in range(len(points)):
            curr_center = centroids[pred_clusters[i]]
            curr_sse += sum((points[i] - curr_center) ** 2)

        sse.append(curr_sse)
    return sse


k = 12
sse = calculate_WSS(X_cat.values, k)
plt.figure()
plt.plot( *args: np.arange(1, k + 1, 1), sse)
plt.xticks(np.arange(1, k + 1, 1))
plt.grid()
plt.xlabel('k')
plt.ylabel('WSS')
plt.title('k selection in k-mean Elbow Algorithm')
plt.show()
```

Here we are using the euclidean distance as a metric to calculate the distance between each point and the assigned centroids. We also plot the WSS vs k graph to determine which k is the best fit k for our data.

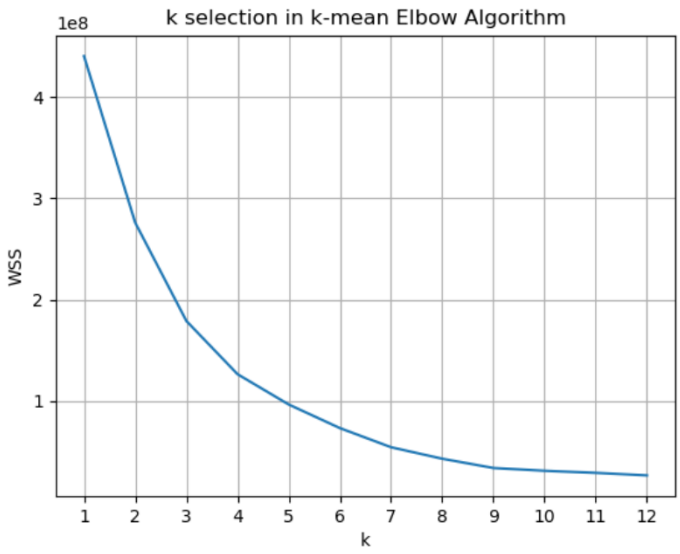

Fig 56. K-selection in K-means using Elbow Method

The graph is seen to show a steep plot at around k = 9 clusters. We will choose this k as our final number of clusters for our data.

Silhouette Method

```
from sklearn.metrics import silhouette_score

sil = []
kmax = 12

for k in range(2, kmax + 1):
    kmeans = KMeans(n_clusters=k).fit(X_cat.values)
    labels = kmeans.labels_
    sil.append(silhouette_score(X_cat.values, labels, metric='euclidean'))

plt.figure()
plt.plot( *args: np.arange(2, k + 1, 1), sil, 'bx-')
plt.xticks(np.arange(2, k + 1, 1))
plt.grid()
plt.xlabel('k')
plt.ylabel('Silhouette score')
plt.title('Silhouette Method')
plt.show()
```

In this method we assign a silhouette score to all of the clusters in the given range (12). The highest scorer will be our best selection for k. Our choice of metric here will be 'Euclidean' distance.

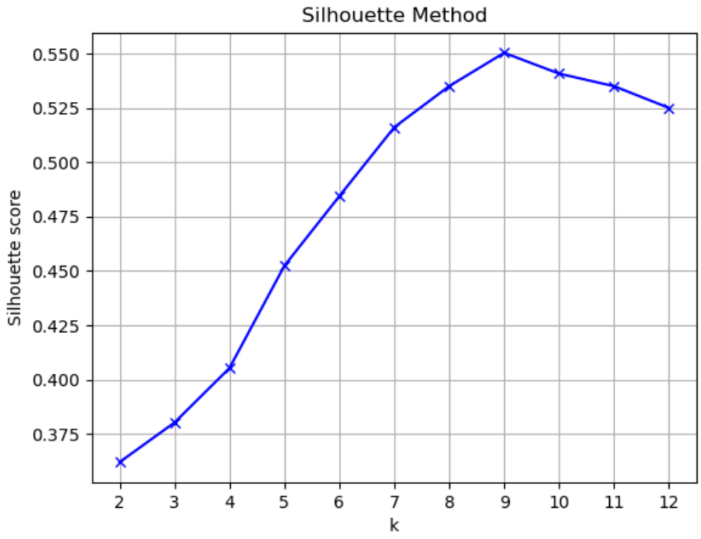

Fig 57. K-selection in K-means using Silhouette Method

We can clearly see that the silhouette score starts dropping after getting to the peak at k = 9 clusters. So, our final selection will be k = 9 clusters.

Since both method suggest that our data forms 9 different clusters we can say the following are the possible groups in our NYC taxi dataset:

- **Temporal Patterns -** These clusters might indicate that the groups are divided by rush hours, people who travel in different timings during the rush hour or maybe the data is divided into areas with specific demand of taxis during the rush hour.
- **Traffic Conditions -** The traffic conditions might be determined by the formation of these 9 clusters, like heavy traffic, moving traffic, no traffic etc.
- **Payment Behaviour -** These clusterings can also be a very good indicator of customers paying in different ranges. Some might be good tippers, some might not tip at all, some who tip very low, some who are tipping exceptionally, some people who are splitting the bill etc.

Apriori

In this section of our report we will apply the apriori algorithm to the categorical columns of our dataset and try to notice the behavior of our data.

```
from mlxtend.preprocessing import TransactionEncoder
from mlxtend.frequent_patterns import apriori, association_rules

df_categories = df_categories.fillna('')
transactions = df_categories.values.tolist()

a = TransactionEncoder()
a_data = a.fit_transform(transactions)
df_apriori = pd.DataFrame(a_data, columns=a.columns_)
print(df_apriori)

df_apriori = apriori(df_apriori, min_support=0.2, use_colnames=True, verbose=1)
print(df_apriori)
df_ar = association_rules(df_apriori, metric='confidence', min_threshold=0.6)
df_ar = df_ar.sort_values(['confidence', 'lift'], ascending=[False, False])
print(df_ar.to_string())
```

Firstly, we selected only the categorical data from our dataset and converted them into ‘string’ data type for better identification. The next step was to encode our data via a transaction encoder which will look as the following figure.

	Bronx	Brooklyn	Cash	Credit card	...	Tuesday	Unknown	Wednesday	Yes
0	False	False	True	False	...	True	True	False	False
1	False	False	False	True	...	False	False	False	True
2	False	False	False	True	...	True	False	False	True
3	False	False	False	True	...	False	False	False	True
4	False	False	False	True	...	False	False	False	True
...	...	...	...	...	...	...	...	...	...
49995	False	False	False	True	...	False	False	False	True
49996	False	False	False	True	...	False	False	False	True
49997	False	False	False	True	...	False	False	False	True
49998	False	False	False	True	...	False	False	False	True
49999	False	False	False	True	...	False	False	True	True

Fig 58. Applying Transaction Encoder on our Categorical Dataset

After encoding we apply the apriori algorithm to form a set of rules from this data.

```
Processing 9 combinations | Sampling itemset size 3
   support                      itemsets
0  0.82038                 (Credit card)
1  0.99574                   (Manhattan)
2  0.20370                          (No)
3  0.79630                         (Yes)
4  0.81680       (Credit card, Manhattan)
5  0.79612             (Credit card, Yes)
6  0.20290                (No, Manhattan)
7  0.79284               (Yes, Manhattan)
8  0.79266  (Credit card, Yes, Manhattan)
```

	antecedents	consequents	antecedent support	consequent support	support	confidence	lift	leverage	conviction	zhangs_metric
3	(Yes)	(Credit card)	0.79630	0.82038	0.79612	0.999774	1.218672	0.142851	794.618922	0.880876
9	(Yes, Manhattan)	(Credit card)	0.79284	0.82038	0.79266	0.999773	1.218671	0.142230	791.166227	0.866160
4	(No)	(Manhattan)	0.20370	0.99574	0.20290	0.996073	1.000334	0.000068	1.084703	0.000419
5	(Yes)	(Manhattan)	0.79630	0.99574	0.79284	0.995655	0.999915	-0.000068	0.980416	-0.000419
7	(Credit card, Yes)	(Manhattan)	0.79612	0.99574	0.79266	0.995654	0.999914	-0.000069	0.980194	-0.000424
0	(Credit card)	(Manhattan)	0.82038	0.99574	0.81680	0.995636	0.999896	-0.000085	0.976206	-0.000580
11	(Yes)	(Credit card, Manhattan)	0.79630	0.81680	0.79266	0.995429	1.218694	0.142242	40.077516	0.880948
8	(Credit card, Manhattan)	(Yes)	0.81680	0.79630	0.79266	0.970446	1.218694	0.142242	6.892384	0.979526
2	(Credit card)	(Yes)	0.82038	0.79630	0.79612	0.970428	1.218672	0.142851	6.888351	0.998967
10	(Credit card)	(Yes, Manhattan)	0.82038	0.79284	0.79266	0.966211	1.218671	0.142230	6.130950	0.998963
1	(Manhattan)	(Credit card)	0.99574	0.82038	0.81680	0.820294	0.999896	-0.000085	0.999524	-0.023895
6	(Manhattan)	(Yes)	0.99574	0.79630	0.79284	0.796232	0.999915	-0.000068	0.999666	-0.019668
12	(Manhattan)	(Credit card, Yes)	0.99574	0.79612	0.79266	0.796051	0.999914	-0.000069	0.999663	-0.019891

Fig 59. Itemsets and Associations

The itemset shows the percentage of appearance of the items in the data in that order. So, a combination of Credit Card and Manhattan has appeared for 81% of our data and a combination of Credit card, Yes (store_and_forward_flag value) and Manhattan has appeared for 79.2% of the whole data.

The latter associations show that if antecedents occur and it is much likely that consequents will occur at the same time. So, If a person is paying via Credit Card he is very likely to either be a visitor or a traveler from Manhattan and vice versa is true as well for this.

Recommendations

In this section we will summarize our learnings from the various analyses we performed on our dataset. There were a lot of learnings from this project.

The first phase was a very intricate analysis of how to remove redundant, collinear and sparse data from our dataset so as to make statistically focused calculations in both of our regression and classification analysis. We can see a lot of successful regression and classification models based on our needs.

We were successfully able to estimate the total amount to be paid by the passenger depending upon several riding factors. The MSE came out to be 0.188 for our regression model and when we analyzed our model statistically we found that the independent features are successfully describing about 81.2% of variance of the dependent variable.

We were successfully able to implement various classifiers which classified our target variable tip given. And based on the use case we were able to justify that Naïve Bayes and Random Forest (Boosting) were the best classifiers for our model. But, we need to remember that this is not a strict decision as most of our classifiers were performing similarly. It is based on our discretion which model we use for our predictions. Some classifiers showed evidence about possible overfitting and we need to be aware about such situations by constantly testing our models in different scenarios, like using stratified k-fold cross validations.

At the ending phase we used our dataset to explore unsupervised learning. We performed K-means clustering algorithms using elbow method and silhouette method and we saw that both of them were pointing towards k = 9 clusters. Lastly when we performed apriori on our encoded

dataset we found that certain combinations of categories occur very frequently in our dataset. Which indicate certain patterns existing within our dataset.

Appendix

Code Snippet for Cross Validator

```
def cross_validator(classifier, X_, y_, clf_name):
   n_splits = 5 if clf_name not in ('SVM (linear)', 'SVM (poly)', 'SVM (rbf)',
                                    'Random Forest (stacking)', 'Random Forest
(Bagging)',
                                    'Random Forest (boosting)') else 3

   stratified_kfold = StratifiedKFold(n_splits=n_splits, shuffle=True,
random_state=5805)

   accuracy_scores = []

   for train_index, test_index in stratified_kfold.split(X_, y_):
       X_train_cv, X_test_cv = X_.iloc[train_index], X_.iloc[test_index]
       y_train_cv, y_test_cv = y_.iloc[train_index], y_.iloc[test_index]

       classifier.fit(X_train_cv, y_train_cv)

       y_pred_cv = classifier.predict(X_test_cv)

       accuracy_cv = accuracy_score(y_test_cv, y_pred_cv)
       accuracy_scores.append(accuracy_cv)
```

```
    c = PrettyTable()

    c.field_names = ['Fold', 'Accuracy']

    for i, acc in enumerate(accuracy_scores, 1):

        c.add_row([i, acc.round(2)])


    mean_accuracy = np.mean(accuracy_scores)

    print(c.get_string(title=f'k = {n_splits} fold accuracy'))

    print(f'{clf_name} classifier, Accuracy Mean = {mean_accuracy: .2f}')
```

Code Snippet for identifying highest p value from OLS regression model

```
def highest_p_value(mod):

    summary_text = mod.summary().tables[1].as_text()

    summary_df = pd.read_csv(StringIO(summary_text), delimiter='\s+',
skiprows=[0], header=0)

    highest, variable_with_highest_p_value = 0, 0

    filtered_summary_df = summary_df[summary_df['P>|t|'] > 0.01]

    if not filtered_summary_df.empty:

        highest = filtered_summary_df['P>|t|'].max()

        variable_with_highest_p_value =
filtered_summary_df.loc[filtered_summary_df['P>|t|'].idxmax(), 'coef']

    AIC_val = mod.aic

    BIC_val = mod.bic

    adj_R_sq_val = mod.rsquared_adj

    return AIC_val, BIC_val, adj_R_sq_val, highest,
variable_with_highest_p_value
```

References

TLC Trip Record Data. TLC Trip Record Data - TLC. (n.d.). https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page

Yang, C., & Gonzales, E. J. (2014). Modeling Taxi Trip Demand by Time of Day in New York City. Transportation Research Record, 2429(1), 110-120. https://doi.org/10.3141/2429-12

(LEDU), E. E. (2018, September 12). *Understanding K-means clustering in machine learning*. Medium. https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

Chonyy, A. (2022, September 19). *Apriori: Association rule mining in-depth explanation and python implementation*. Medium. https://towardsdatascience.com/apriori-association-rule-mining-explanation-and-python-implementation-290b42afdfc6

Jackson, H. (2023, November 9). *K-Nearest Neighbor(KNN) algorithm*. GeeksforGeeks. https://www.geeksforgeeks.org/k-nearest-neighbours/

Williams, J. (2023a). *Passenger frequently asked questions*. Passenger Frequently Asked Questions - TLC. https://www.nyc.gov/site/tlc/passengers/passenger-frequently-asked-questions.page