Market Segmentation Analysis on Online Vehicle Booking Market

Contributors:

1. Spandan Dhadse
2. Akshay Pankar
3. Metturu Mouli
4. Advait Rahul Ghatge
5. Bhodigam Akshitha

Date 11/05/2023

1. Bhodigam Akshitha

1.Defining the question:

- How to evaluate the dataset in order to achieve the required insights?
- What attributes need to analysed and evaluated?

After analyzing the dataset:

- Cleaning the data containing null values and outliers and then analysing it based on the following factors:
- In which season does vehicle booking is at high rate?
- In which climatic conditions does vehicle booking is at high rate?
- In which hours does vehicle booking is at high rate?
- In which weekdays does vehicle booking is at high rate?
- In which months does vehicle booking is at high rate?
- What is the correlation between the features in the dataset?

2.Collecting the data:

Dataset:

https://www.kaggle.com/datasets/rickymaisnam/cab-booking/

Datasets that are used for analyzing the online vehicle booking criteria is:

- train.csv
- train_label.csv
- test.csv
- test_label.csv

This dataset contains four files which are taken from Kaggle.

Train Data:

```
train.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	5/2/2012 19:00	Summer	0	1	Clear + Few clouds	22.14	25.760	77	16.9979
1	9/5/2012 4:00	Fall	0	1	Clear + Few clouds	28.70	33.335	79	19.0012
2	1/13/2011 9:00	Spring	0	1	Clear + Few clouds	5.74	6.060	50	22.0028
3	11/18/2011 16:00	Winter	0	1	Clear + Few clouds	13.94	16.665	29	8.9981
4	9/13/2011 13:00	Fall	0	1	Clear + Few clouds	30.34	33.335	51	19.0012

Test Data:

```
test.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed
0	5/10/2012 11:00	Summer	0	1	Clear + Few clouds	21.32	25.000	48	35.0008
1	6/9/2012 7:00	Summer	0	0	Clear + Few clouds	23.78	27.275	64	7.0015
2	3/6/2011 20:00	Spring	0	0	Light Snow, Light Rain	11.48	12.120	100	27.9993
3	10/13/2011 11:00	Winter	0	1	Mist + Cloudy	25.42	28.790	83	0.0000
4	6/2/2012 12:00	Summer	0	0	Clear + Few clouds	25.42	31.060	43	23.9994

- Exploring the Train data to divide into further levels like into factors such as, week day, month, and splitting date and time for easier analysis.

```
train_append.head()
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	date	hour	weekday	month	Total_booking
0	5/2/2012 19:00	Summer	0	1	Clear + Few clouds	22.14	25.760	77	16.9979	5/2/2012	19	Wednesday	May	504
1	9/5/2012 4:00	Fall	0	1	Clear + Few clouds	28.70	33.335	79	19.0012	9/5/2012	4	Wednesday	September	5
2	1/13/2011 9:00	Spring	0	1	Clear + Few clouds	5.74	6.060	50	22.0028	1/13/2011	9	Thursday	January	139
3	11/18/2011 16:00	Winter	0	1	Clear + Few clouds	13.94	16.665	29	8.9981	11/18/2011	16	Friday	November	209
4	9/13/2011 13:00	Fall	0	1	Clear + Few clouds	30.34	33.335	51	19.0012	9/13/2011	13	Tuesday	September	184

3.Cleaning the data:

- **Removing major errors, duplicates, and outliers**: all of which are inevitable problems when aggregating data from numerous sources.
- **Removing unwanted data points**: extracting irrelevant observations that have no bearing on the intended analysis.
- **Checking if there are any Null values:** if there any null values present in the data.
- **Cleaning the data:** data needs to be cleaned from the outliers and from the null values.

Cleaning the Train data from the Outliers:

```
train_append.shape
```

```
(8708, 14)
```

```
for index in outliers:
    train_append.drop(index,inplace=True)
```

```
train_append.shape
```

```
(8466, 14)
```

Checking the null values in the Train data:

```
train_append.isnull().sum()
```

```
datetime        0
season          0
holiday         0
workingday      0
weather         0
temp            0
atemp           0
humidity        0
windspeed       0
date            0
hour            0
weekday         0
month           0
Total_booking   0
dtype: int64
```

- As, there are no null values we can continue our analysis.

4. Analysing the data:

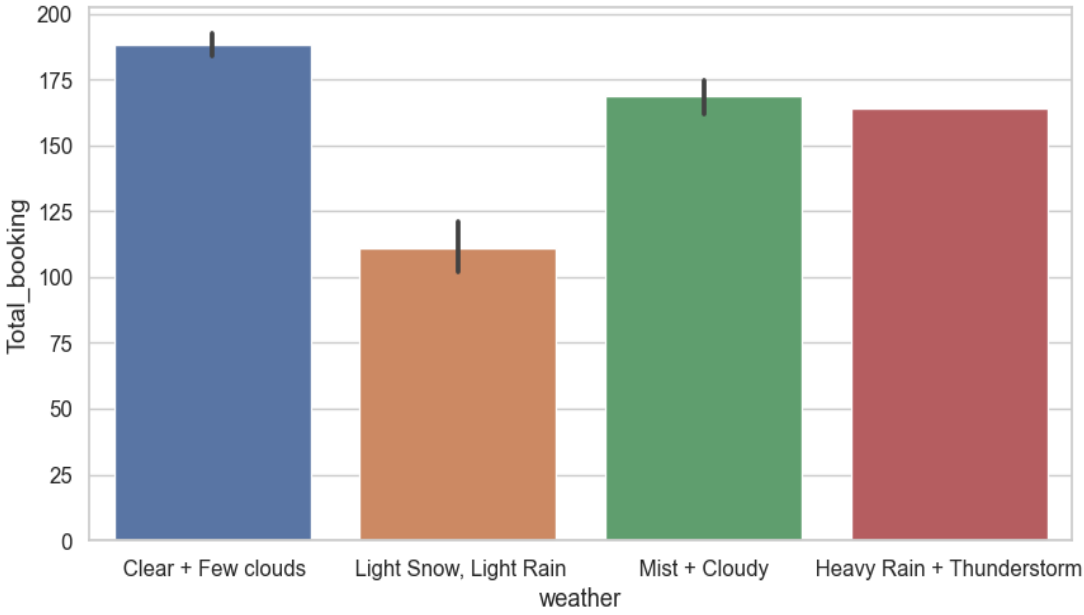
Analysis needs to be done in such a way, that all questions which are framed to be answered.

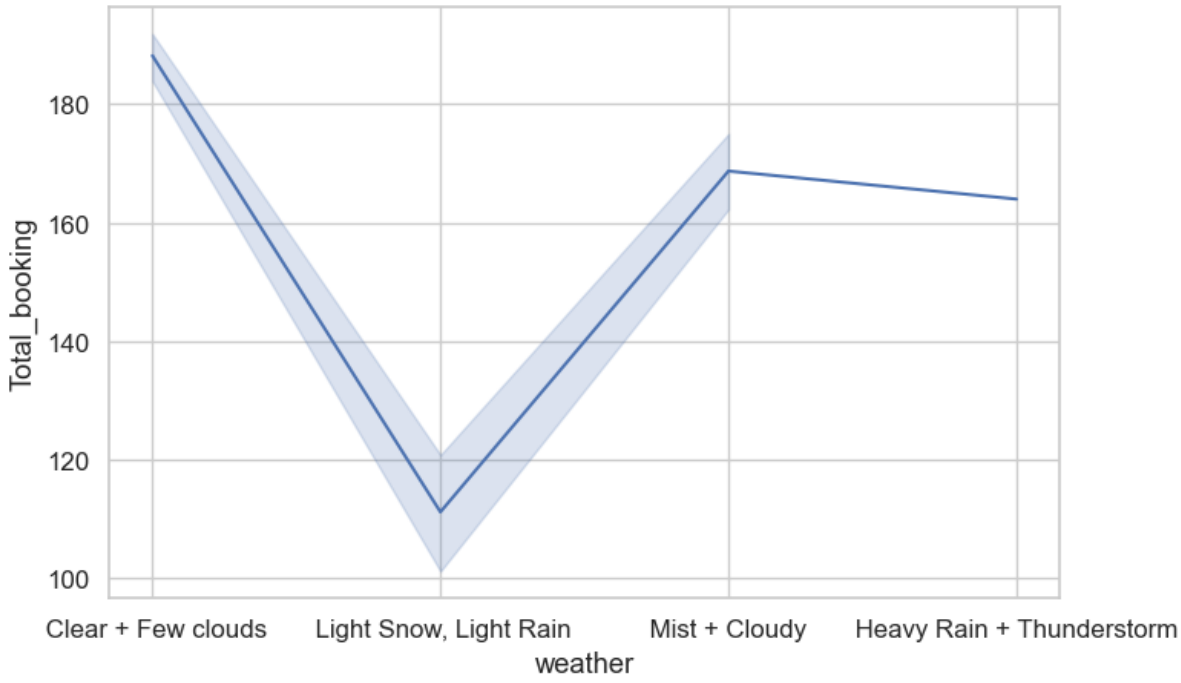
- As, we are aiming to find the statistics of vehicle booking mechanisms data needs to be analysed on different attributes present over the dataset.

- Analysis is done based on various factors such as, climatic conditions, Weather conditions.

- And also based on months, weekdays, hours.

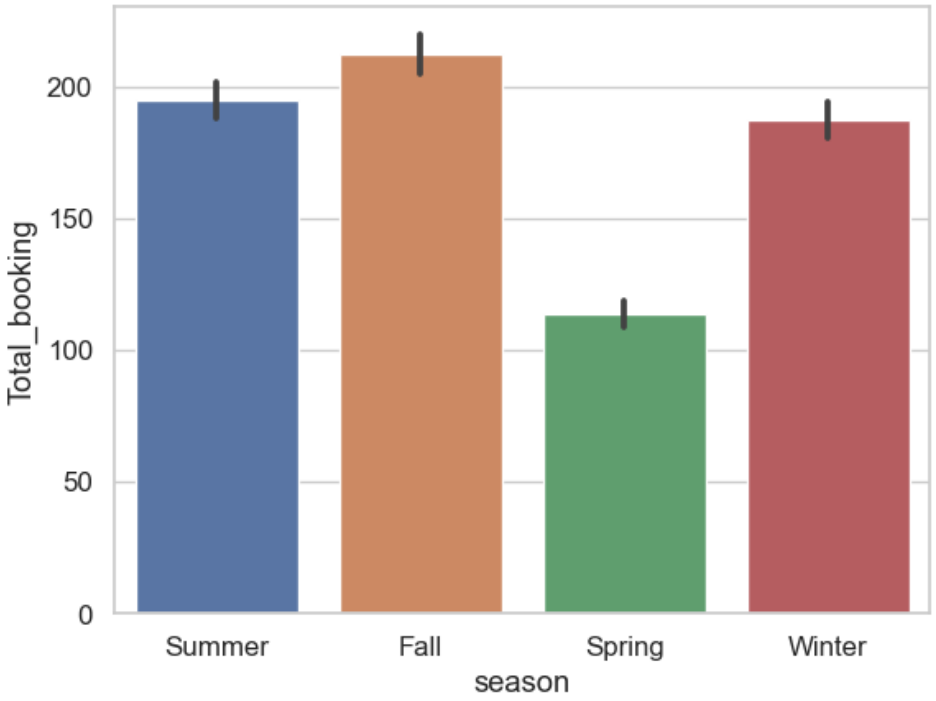
Based on Weather Conditions:

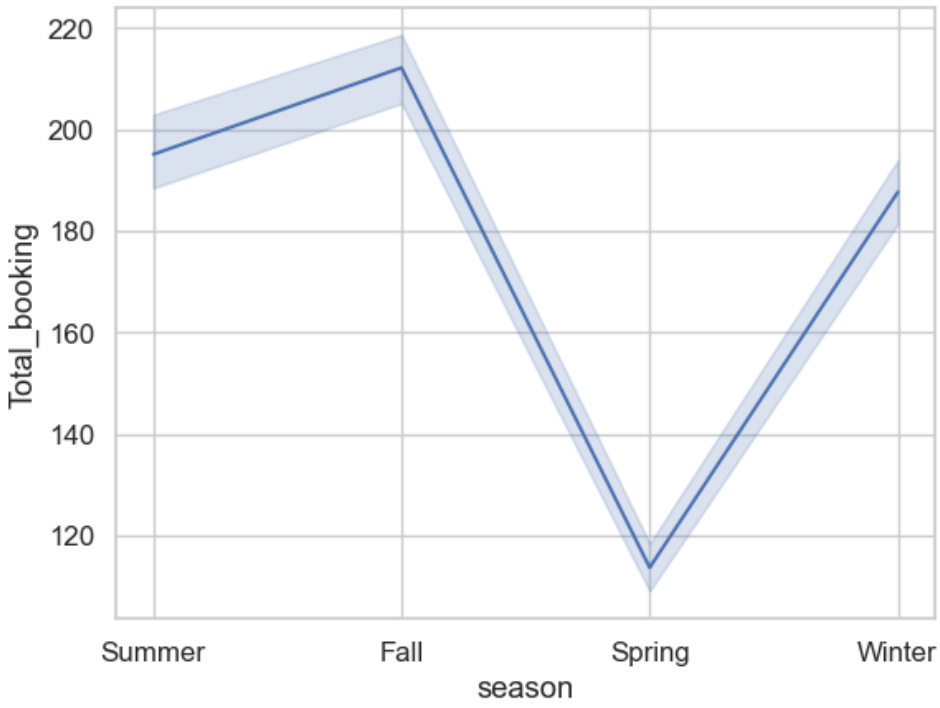




- Compared to the different weather conditions there are high number of cab bookings for Clear and few clouds criteria and low in case of light snow and rain.
- Both Mist and cloudy and for heavy rain and thunderstorm has equally number of cab bookings.

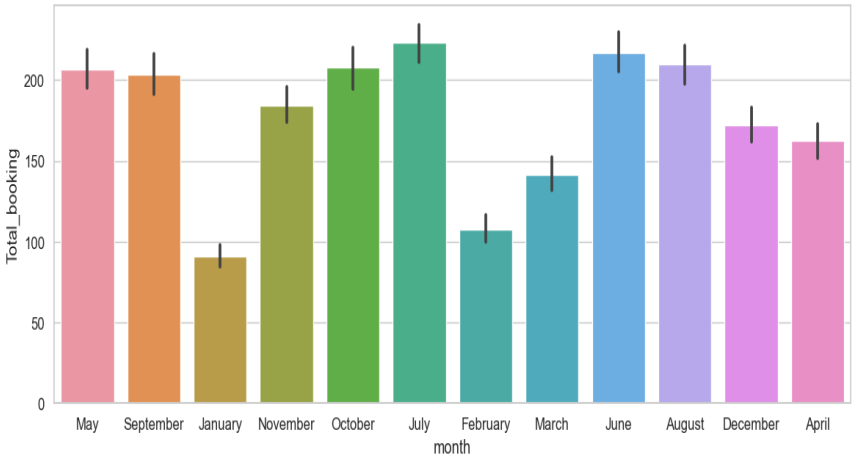
Based on seasons:

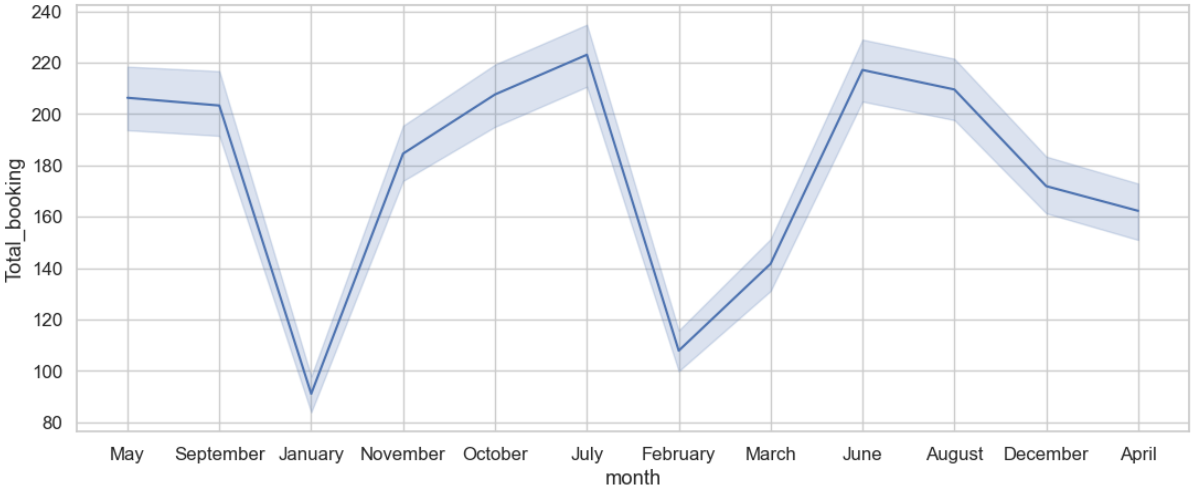




- Compared to the different seasons there are high number of cab bookings in case of Fall and low in case of spring.
- Both summer and winter has equally number of cab bookings, where summer has a little bit a greater number of cab bookings compared to winter.

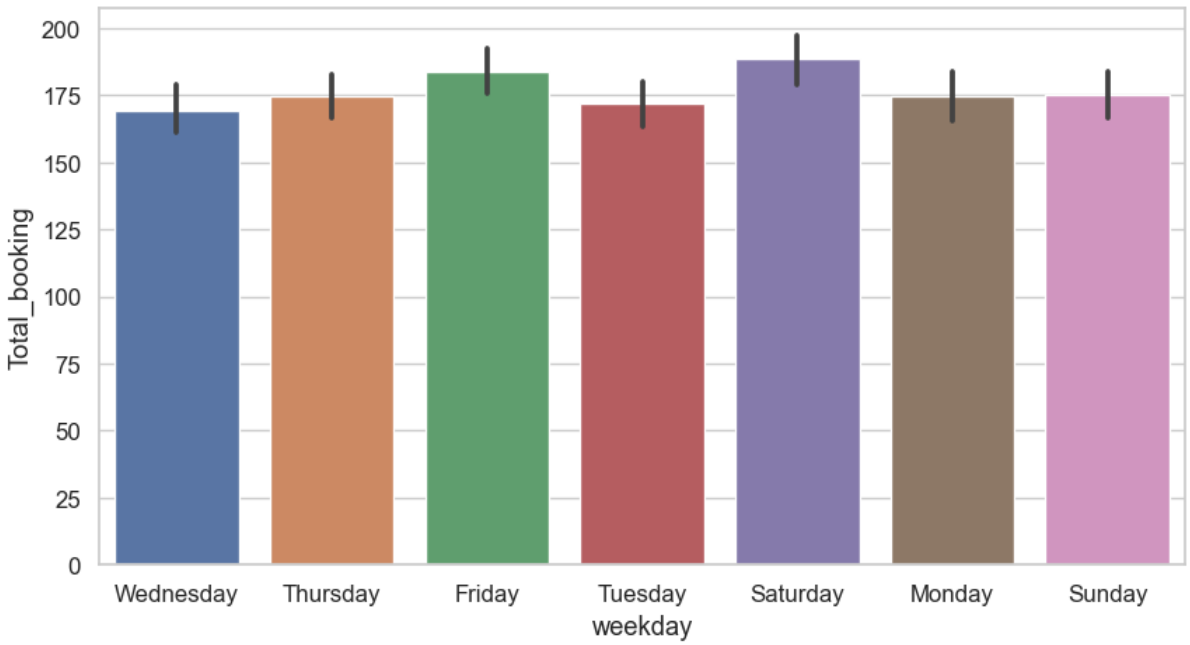
Based on months:

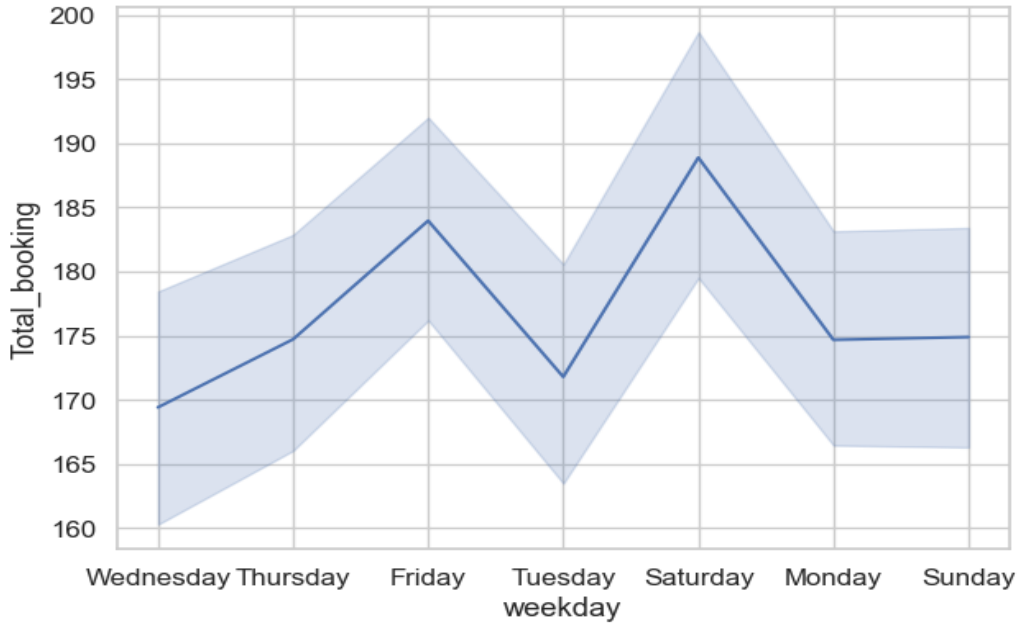




- When compared to the values in dataset, July has a greater number of cab bookings and the least is January.
- Rather than January, February and march almost every month has recorded a good number of cab booking rates.

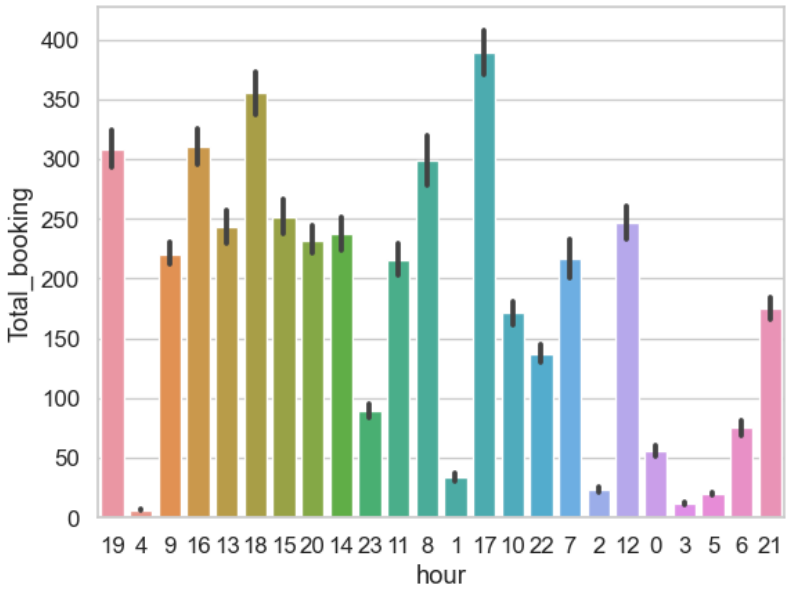
Based on weekdays:

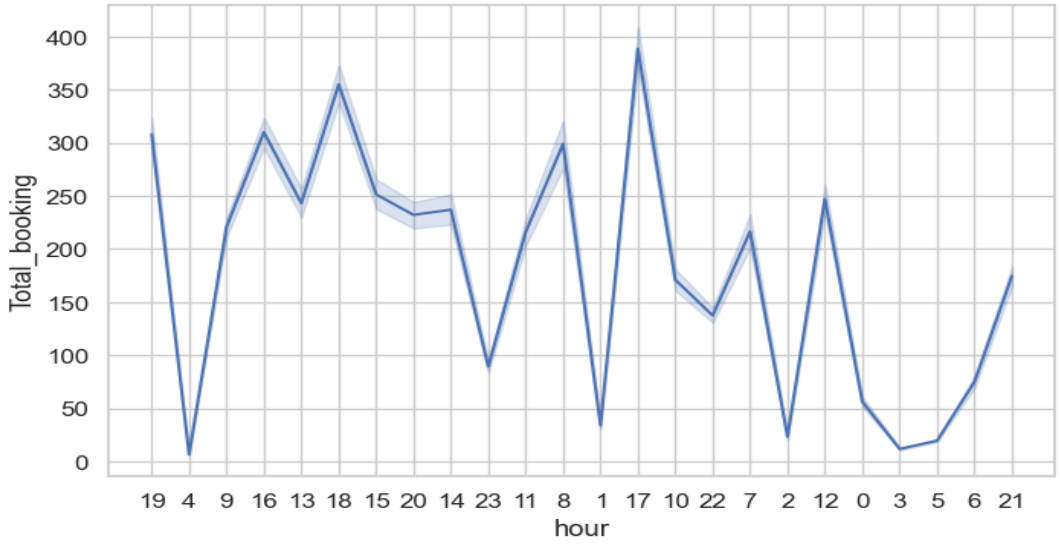




- Weekends having comparably high number of cab bookings when compared to the whole week.
- Remaining days the vehicles booking seems normal.
- So, focusing on weekends and providing good number of cab bookings can benefitable.

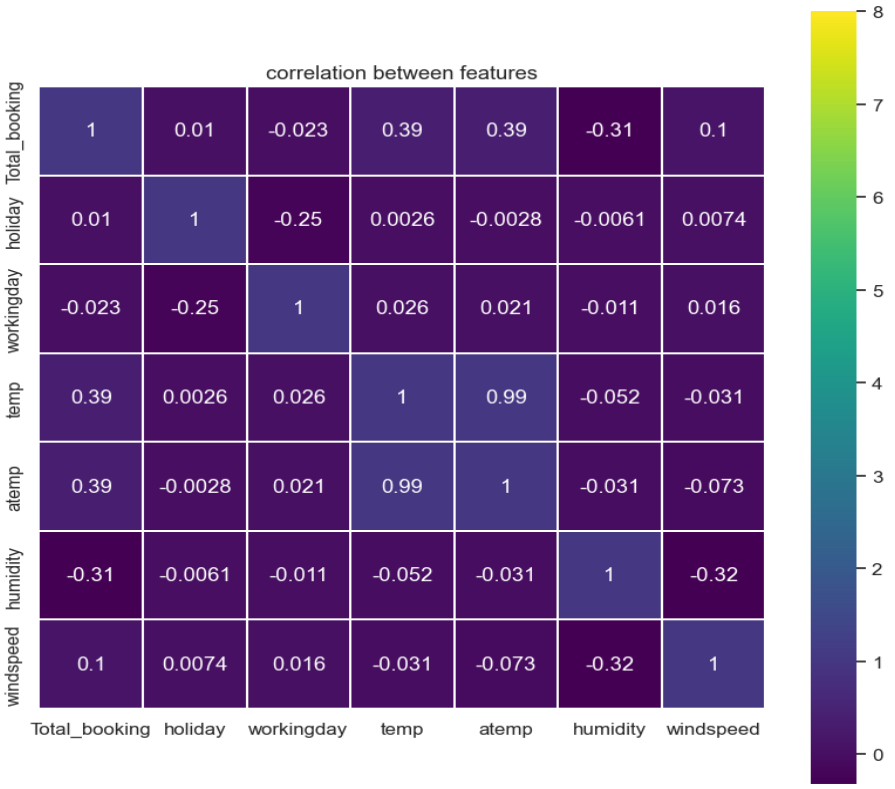
Based on hours:





- While analysing this data, working hours has good number of cab bookings when compared to the early mornings and nights.
- Mainly, there was a high demand for cab bookings in case of evenings at the peak hours of 13 to 19 i.e., is 1 pm to 7 pm in the evenings.
- Even mornings i.e., 8 am to 12 pm has a good demand for vehicle bookings.
- So, considering and focusing on this working hour and providing good number of vehicles can be profitable.

Correlation matrix:



- A correlation matrix is a statistical tool that shows the correlation coefficients between a set of variables.
- The matrix is usually presented as a table with rows and columns representing the variables, and the cells of the table containing the correlation coefficients.
- The correlation matrix is useful for identifying patterns and relationships among variables.
- For example, if two variables have a high positive correlation coefficient (i.e., close to 1), it means that they are strongly and positively related, and changes in one variable tend to be accompanied by changes in the other variable.
- On the other hand, if two variables have a high negative correlation coefficient (i.e., close to -1), it means that they are strongly and negatively related, and changes in one variable tend to be accompanied by changes in the opposite direction in the other variable.

5.Result:

The analysis of online cab bookings reveals that there are certain patterns in the data that can be used to optimize business strategy.

One such pattern is the correlation between weather conditions and the number of cab bookings. Clear and few clouds weather criteria are found to have the highest number of cab bookings, while light snow and rain have the lowest. This suggests that people are more likely to book vehicles when the weather is favorable for outdoor activities.

Another pattern observed is the variation in bookings based on the seasons. Fall season has the highest number of bookings, whereas spring has the lowest. This may be due to the fact that fall is a popular time for travel and tourism. Interestingly, summer and winter have the same number of bookings, with summer having slightly more. This indicates that people tend to travel more during the summer months but also book cabs for winter activities.

When analyzed by months, July has the highest number of cab bookings, while January has the lowest. However, almost every month, except for January, February, and March, has recorded a good number of vehicles booking rates.

This suggests that businesses could benefit from focusing on months with high demand, such as July, and optimizing their marketing and pricing strategies accordingly.

Weekends are found to have a higher number of bookings compared to weekdays. This may be due to people having more free time on weekends and being more likely to engage in leisure activities.

Focusing on weekends and providing a good number of cab bookings can be profitable.

Furthermore, working hours are found to have more bookings than early mornings and nights. Evenings from 1 pm to 7 pm and mornings from 8 am to 12 pm have the highest demand for cab bookings.

This could be due to people using vehicles for work-related activities during these times. Focusing on these working hours and providing a good number of cabs can be profitable.

In summary, analyzing online vehicle bookings data has revealed several patterns that businesses can use to optimize their strategies. By focusing on favorable weather conditions, high-demand seasons, weekends, and working hours, businesses can increase their profitability by providing a good number of vehicle bookings.

Conclusion:

Analysis Description:

Based on the analysis of online cab bookings data, it is clear that there is potential for starting an online cab booking service that can capitalize on the observed patterns in customer demand. Here are some strategies that can be used to start an online cab booking service:

1. Focus on providing cabs that are suitable for different weather conditions: As the analysis shows, customers are more likely to book vehicles when the weather is favorable for outdoor activities. Therefore, providing a range of vehicles that are suitable for different weather conditions, such as convertibles for clear days and SUVs for snowy days, can increase customer satisfaction and bookings.

2. Offer seasonal promotions and discounts: To take advantage of high-demand seasons, such as fall, businesses can offer promotions and discounts that encourage customers to book vehicles during those periods. This can help increase revenue during slow periods and build customer loyalty.

3. Optimize pricing strategies: By analyzing the data by month, businesses can identify periods of high demand and adjust pricing accordingly. For instance, higher prices can be set during peak demand periods, such as July, to maximize revenue, while lower prices can be set during slow periods to attract more customers.

4. Provide a variety of vehicles: Offering a wide range of vehicles, such as cars, trucks, vans, and SUVs, can appeal to a broader customer base and increase the chances of bookings.

5. Focus on providing excellent customer service: Providing prompt and efficient service, including easy online booking and timely vehicle delivery, can help build customer satisfaction and loyalty.

By incorporating these strategies, businesses can create a successful online vehicle booking service that caters to customer demand and maximizes profitability.

GitHub Link:

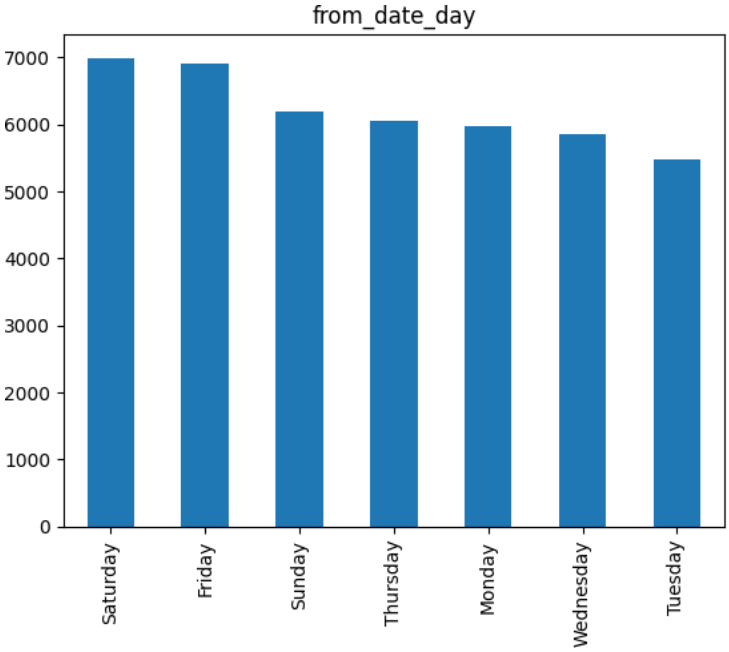
https://github.com/BhodigamAkshitha/cab-analysis/blob/685303c4f220911e5341b8e815e957b40b8dc65e/cab%20analysis.ipynb

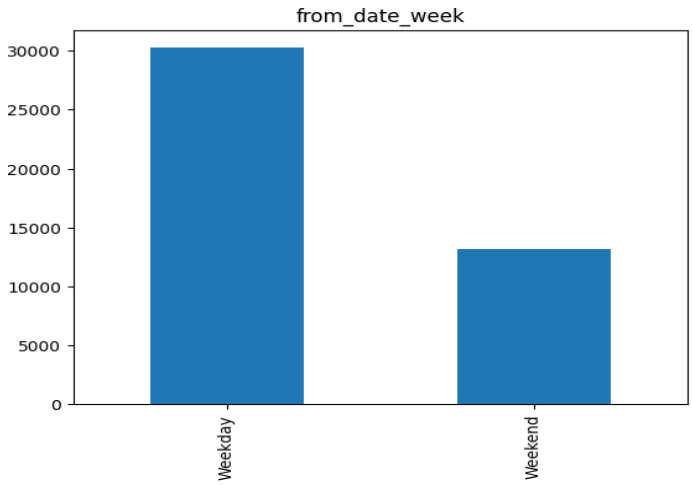
2. **Advait Rahul Ghatge**

Dataset used:

https://www.kaggle.com/datasets/akashkumar01/yourcabs

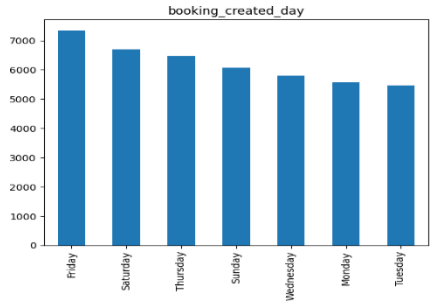
1. Visualization

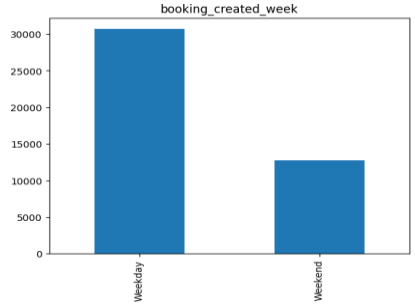




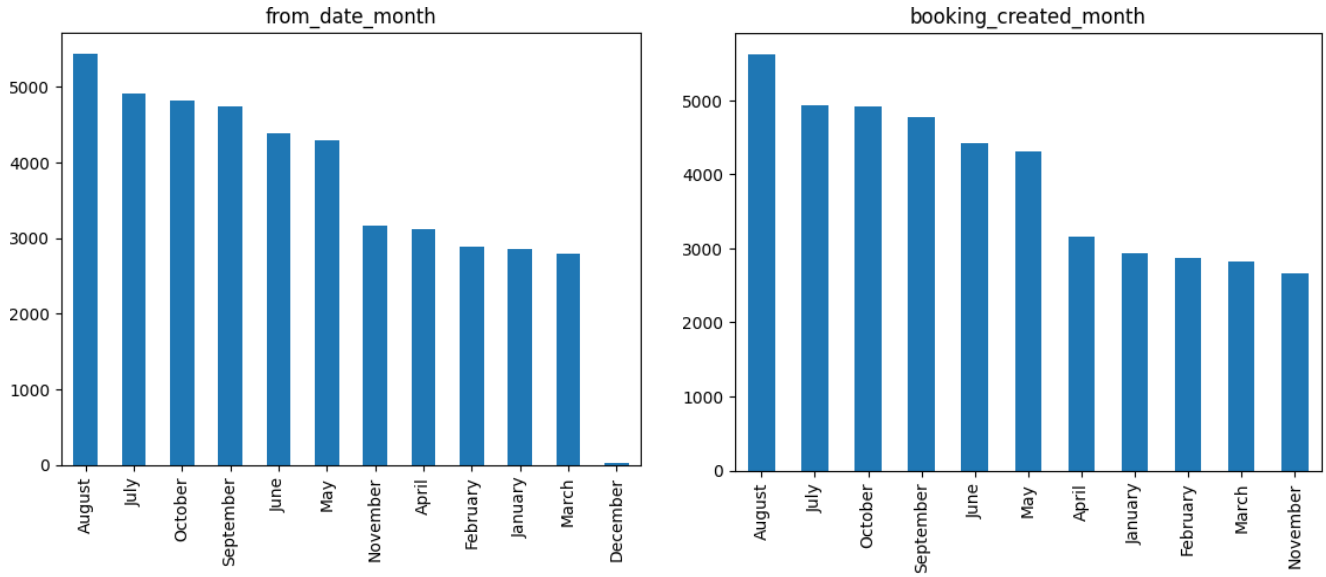
Data visualization is used to make complex data easier to understand, identify relationships and correlations, and communicate insights and findings to others. It also makes data more engaging, which can encourage people to explore it further. Finally, data visualization supports decision-making by providing a clear, visual representation of the data that can help identify trends and patterns that might be missed in other forms of analysis.

As we can see from the above graph, there are more cancellations on the weekend than there are on weekdays.

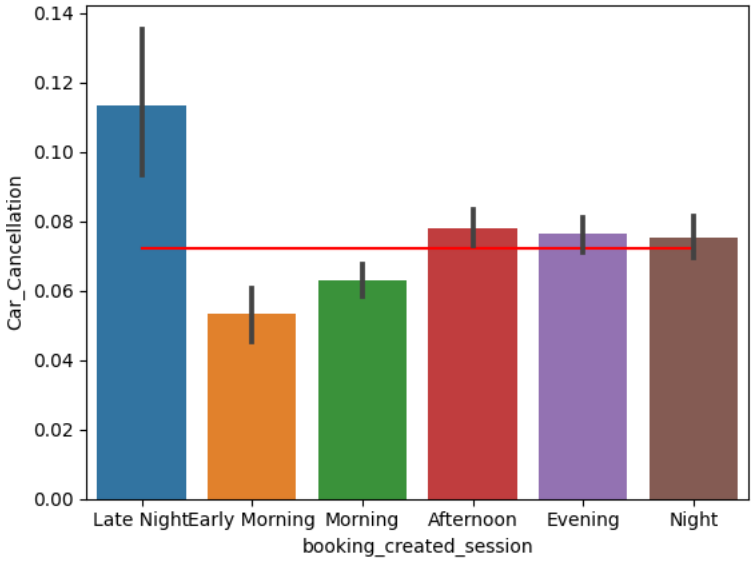




We can see that fewer people book cabs on Sundays and yet they cancel their rides moreoften.



Here, we observe that most people cancel and book their cabs in the monsoon season.



Here, we observe that most people cancel their rides late into the night.

1. Splitting of Data

The train-test split is used to estimate the performance of machine learning algorithms that are applicable for prediction-based Algorithms/Applications. This method is a fast and easy procedure to perform such that we can compare our own machine learning model results to machine results. By default, the Test set is split into 30 % of actual data and the training set is split into 70% of the actual data. We need to split a dataset into train and test sets to evaluate how well our machine learning model performs. The train set is used to fit the model, and the statistics of the train set are known. The second set is called the test data set, this set is solely used for predictions.

Train Test Split

```
[63] from sklearn.model_selection import train_test_split
     X_train,X_test,y_train,y_test = train_test_split(data_all,target,test_size=0.3,random_state=1)
```

```
X_train.head()
```

	online_booking	mobile_site_booking	traveltype_pointtopoint	traveltype_hourly	distance	time_diff	from_area_id_Low Cancellation	from_area_id_Medium Cancellation
21506	0	0	1	0	9.236426	2832.0	1	0
12367	0	1	1	0	1.809583	1504.0	1	0
5983	0	0	1	0	13.558000	214.0	1	0
18513	0	0	0	0	18.456702	37.0	1	0
16033	1	0	1	0	13.405912	128.0	1	0

5 rows × 53 columns

2. Model Building

The Machine Learning models can be understood as a program that has been trained to find patterns within new data and make predictions. These models are represented as a mathematical function that takes requests in the form of input data, makes predictions on input data, and then provides an output in response. First, these models are trained over a set of data, and then they are provided an algorithm to reason over data, extract the pattern from feed data and learn from those data. Once these models get trained, they can be used to predict the unseen dataset.

Logistic Regression

Logistic regression is a supervised machine learning algorithm mainly used for classification tasks where the goal is to predict the probability that an instance of belonging to a given class. It is used for classification algorithms its name is logistic regression. It’s referred to as regression because it takes the output of the linear regression function as input and uses a sigmoid function to estimate the probability for the given class. The difference between linear regression and logistic regression is that linear regression output is the continuous value that can be anything while logistic regression predicts the probability that an instance belongs to a given class or not.

Terminologies involved in Logistic Regression:

Here are some common terms involved in logistic regression:

Independent variables: The input characteristics or predictor factors applied to the dependent variable's predictions.

Dependent variable: The target variable in a logistic regression model, which we are trying to predict.

Logistic function: The formula used to represent how the independent and dependent variables relate to one another. The logistic function transforms the input variables into a probability value between 0 and 1, which represents the likelihood of the dependent variable being 1 or 0.

Odds: It is the ratio of something occurring to something not occurring. It is different from probability as probability is the ratio of something occurring to everything that could possibly occur.

Log-odds: The log-odds, also known as the logit function, is the natural logarithm of the odds. In logistic regression, the log odds of the dependent variable are modeled as a linear combination of the independent variables and the intercept.

Coefficient: The logistic regression model's estimated parameters, show how the independent and dependent variables relate to one another.

Intercept: A constant term in the logistic regression model, which represents the log odds when all independent variables are equal to zero.

Maximum likelihood estimation: The method used to estimate the coefficients of the logistic regression model, which maximizes the likelihood of observing the data given the model.

```
[67] from sklearn.linear_model import LogisticRegression
     lr = LogisticRegression(random_state=4)
     lr.fit(X_train,y_train)
```

```
LogisticRegression
LogisticRegression(random_state=4)
```

Evaluation:

```
[76] y_pred_lr=lr.predict(X_test)
     y_pred_dtc=dtc.predict(X_test)
     y_pred_rfc=rfc.predict(X_test)
```

```
[77] from sklearn import metrics
     from sklearn.metrics import confusion_matrix
```

```
[78] print('Logistic Regression Metrics')
     print('Accuracy:', metrics.accuracy_score(y_test, y_pred_lr))
     print('Precision:', metrics.precision_score(y_test, y_pred_lr))
     print('Recall:', metrics.recall_score(y_test, y_pred_lr))
     print('f1_score:', metrics.f1_score(y_test, y_pred_lr))
```

```
Logistic Regression Metrics
Accuracy: 0.929623944742901
Precision: 0.5581395348837209
Recall: 0.07725321888412018
f1_score: 0.1357210179076343
```

Decision Tree

Decision Tree is a supervised learning technique that can be used for both classification and Regression problems, but mostly it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the features of a dataset, branches represent the decision rules and each leaf node represents the outcome. In a Decision tree, there are two nodes, which are the Decision Node and Leaf Node. Decision nodes are used to make any decision and have multiple branches, whereas Leaf nodes are the output of those decisions and do not contain any further branches. The decisions or the test are performed on the basis of features of the given dataset. It is a graphical representation for getting all the possible solutions to a problem/decision based on given conditions. It is called a decision tree because, similar to a tree, it starts with the root node, which expands on further branches and constructs a tree-like structure. In order to build a tree, we use the CART algorithm, which stands for Classification and Regression Tree algorithm. A decision tree simply asks a question, and based on the answer (Yes/No), it further split the tree into sub trees.

```
[68] from sklearn.tree import DecisionTreeClassifier
     dtc = DecisionTreeClassifier(criterion='gini',random_state=4)
```

```
[69] from sklearn.model_selection import GridSearchCV
     param_dist = {'max_depth': [3, 5, 6, 7], 'min_samples_split': [50, 100, 150, 200, 250]}
     gscv_dtc = GridSearchCV(dtc, cv=10, param_grid=param_dist, n_jobs=-1)
     gscv_dtc.fit(X_train,y_train)
```

```
GridSearchCV
GridSearchCV(cv=10, estimator=DecisionTreeClassifier(random_state=4), n_jobs=-1,
             param_grid={'max_depth': [3, 5, 6, 7],
                         'min_samples_split': [50, 100, 150, 200, 250]})
  estimator: DecisionTreeClassifier
DecisionTreeClassifier(random_state=4)
    DecisionTreeClassifier
DecisionTreeClassifier(random_state=4)
```

```
[71] dtc=DecisionTreeClassifier(criterion='gini',random_state=4,max_depth=7,min_samples_split=50)
     dtc.fit(X_train,y_train)
```

```
DecisionTreeClassifier
DecisionTreeClassifier(max_depth=7, min_samples_split=50, random_state=4)
```

Evaluation:

```
[80] print('Decision Tree Metrics')
     print('Accuracy:', metrics.accuracy_score(y_test, y_pred_dtc))
     print('Precision:', metrics.precision_score(y_test, y_pred_dtc))
     print('Recall:', metrics.recall_score(y_test, y_pred_dtc))
     print('f1_score:', metrics.f1_score(y_test, y_pred_dtc))
```

```
Decision Tree Metrics
Accuracy: 0.9323100537221796
Precision: 0.6865671641791045
Recall: 0.09871244635193133
f1_score: 0.1726078799249531
```

Random Forest

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of ensemble learning, which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model. As the name suggests, "Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset." Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output. The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

```
[72] from sklearn.ensemble import RandomForestClassifier
     rfc = RandomForestClassifier(criterion='gini',random_state=4)
```

```
[73] param_dist = {'max_depth': [3, 5, 6, 7], 'min_samples_split': [50, 100, 150, 200, 250]}
     gscv_rfc = GridSearchCV(rfc, cv=10, param_grid=param_dist, n_jobs=-1)
     gscv_rfc.fit(X_train,y_train)
```

```
GridSearchCV
estimator: RandomForestClassifier
RandomForestClassifier
```

```
[74] gscv_rfc.best_params_
```

```
{'max_depth': 7, 'min_samples_split': 50}
```

```
[75] rfc=RandomForestClassifier(criterion='gini',random_state=4,max_depth=7,min_samples_split=50)
     rfc.fit(X_train,y_train)
```

```
RandomForestClassifier
RandomForestClassifier(max_depth=7, min_samples_split=50, random_state=4)
```

Evaluation:

```
print('Random Forest Metrics')
print('Accuracy:', metrics.accuracy_score(y_test, y_pred_rfc))
print('Precision:', metrics.precision_score(y_test, y_pred_rfc))
print('Recall:', metrics.recall_score(y_test, y_pred_rfc))
print('f1_score:', metrics.f1_score(y_test, y_pred_rfc))
```

```
Random Forest Metrics
Accuracy: 0.930775134305449
Precision: 1.0
Recall: 0.032188841201716736
f1_score: 0.06237006237006236
```

Conclusions:

The business problem tackled here is trying to improve customer service for YourCabs.com, a cab company in Bangalore.
The problem of interest is booking cancellations by the company due to unavailability of a car. The challenge is that cancellations can occur very close to the trip start time, thereby causing passengers inconvenience.

Content
The goal of is to create a predictive model for classifying new bookings as to whether they will eventually gets cancelled due to car unavailability by the business.
Note: cancellations mentioned are cancellations made by business. Columns details:
id - booking ID
user_id - the ID of the customer (based on mobile number) vehicle_model_id - vehicle model type.
package_id - type of package (1=4hrs & 40kms, 2=8hrs & 80kms, 3=6hrs & 60kms, 4= 10hrs & 100kms, 5=5hrs & 50kms, 6=3hrs & 30kms, 7=12hrs & 120kms)
travel_type_id - type of travel (1=long distance, 2= point to point, 3= hourly rental). from_area_id - unique identifier of area. Applicable only for point-to-point travel and packages
to_area_id - unique identifier of area. Applicable only for point-to-point travel from_city_id - unique identifier of city
to_city_id - unique identifier of city (only for intercity) from_date - time stamp of requested trip start to_date - time stamp of trip end
online_booking - if booking was done on desktop website mobile_site_booking - if booking was done on mobile website booking_created - time stamp of booking
from_lat - latitude of from area from_long - longitude of from area to_lat - latitude of to area
to_long - longitude of to area
Car_Cancellation (available only in training data) - whether the booking was cancelled (1) or not (0) due to unavailability of a car.
Cost_of_error (available only in training data) - the cost incurred if the booking is misclassified. For an un-cancelled booking, the cost of misclassificaiton is 1. For a cancelled booking, the cost is a function of the cancellation time relative to the trip start time.

3. Spandan Dhadse

Dataset Used: https://www.kaggle.com/datasets/arashnic/taxi-pricing-with-mobility-analytics

Data Dictionary

Trip_ID = ID for TRIP (Cannot be used for purposes of modelling)

Trip_Distance = The distance for the trip requested by the customer

Type_of_Cab = Category of the cab requested by the customer

Customer_Since_Months = Customer using cab services since n months; 0 month means current month

Life_Style_Index = Proprietary index created by Sigma Cabs showing lifestyle of the customer based on their behaviour

Confidence_Life_Style_Index = Category showing confidence on the index mentioned above

Destination_Type = Sigma Cabs divides any destination in one of the 14 categories.

Customer_Rating = Average of life time ratings of the customer till date

Cancellation_Last_1Month = Number of trips cancelled by the customer in last 1 month

Var1, Var2 and Var3 = Continuous variables masked by the company. Can be used for modelling purposes

Gender = Gender of the customer

Surge_Pricing_Type = Predictor variable can be of 3 types

Cleaning the data:

- Removing major errors, duplicates, and outliers: all of which are inevitable problems when aggregating data from numerous sources.
- Removing unwanted data points: extracting irrelevant observations that have no bearing on the intended analysis.
- Checking if there are any Null values: if there any null values present in the data.
- Cleaning the data: data needs to be cleaned from the outliers and from the null values.

Data Info:

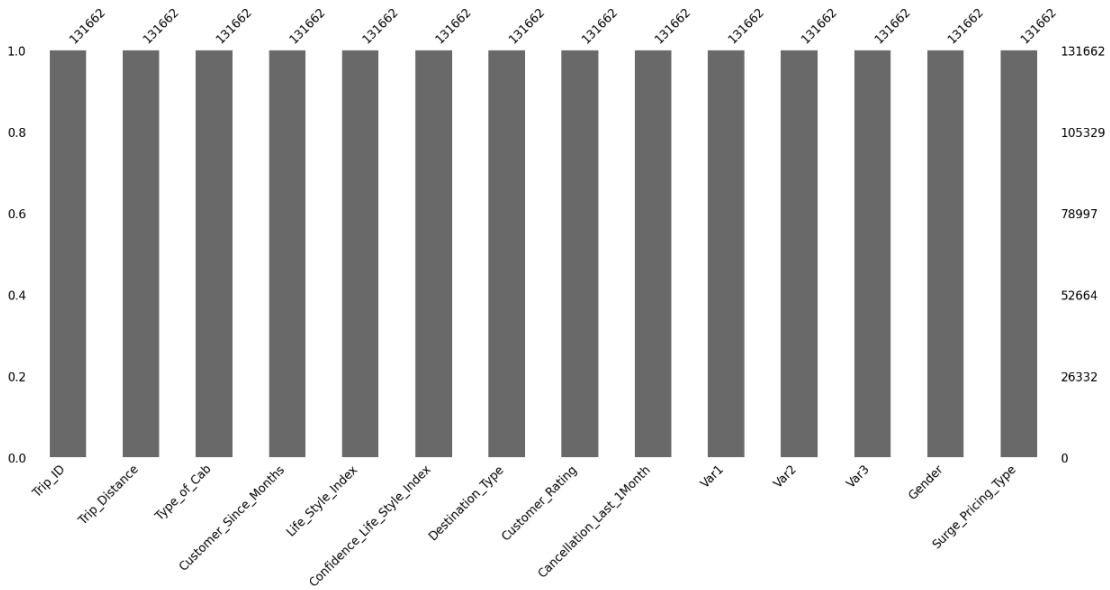
```
In [4]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 131662 entries, 0 to 131661
Data columns (total 14 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   Trip_ID                      131662 non-null  object
 1   Trip_Distance                131662 non-null  float64
 2   Type_of_Cab                  111452 non-null  object
 3   Customer_Since_Months        125742 non-null  float64
 4   Life_Style_Index             111469 non-null  float64
 5   Confidence_Life_Style_Index  111469 non-null  object
 6   Destination_Type             131662 non-null  object
 7   Customer_Rating              131662 non-null  float64
 8   Cancellation_Last_1Month     131662 non-null  int64
 9   Var1                         60632 non-null   float64
 10  Var2                         131662 non-null  int64
 11  Var3                         131662 non-null  int64
 12  Gender                       131662 non-null  object
 13  Surge_Pricing_Type           131662 non-null  int64
dtypes: float64(5), int64(4), object(5)
memory usage: 14.1+ MB
```

Removing Null values:

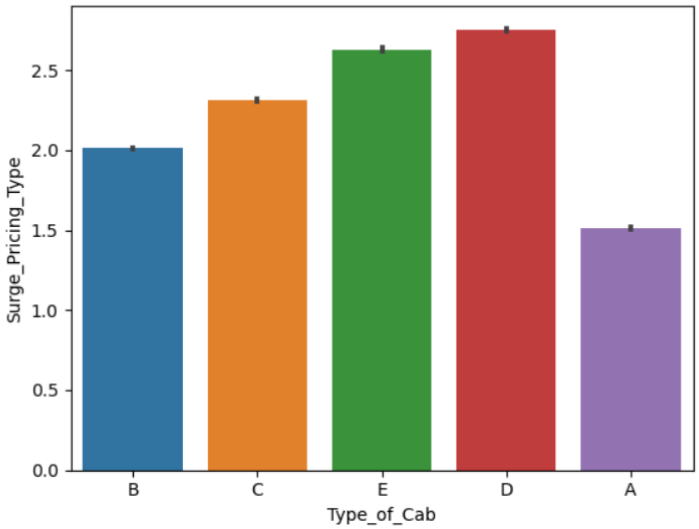
```
In [12]: msno.bar(df)
```

```
Out[12]: <Axes: >
```



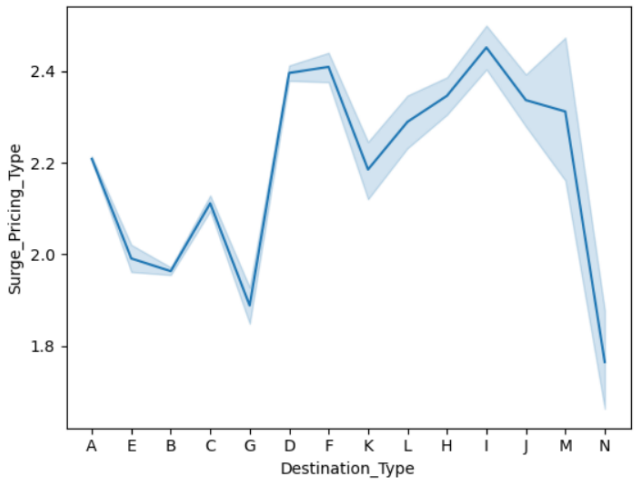
Does the type of cab have an impact on the surge pricing type?

```
In [21]: sns.barplot(x='Type_of_Cab', y='Surge_Pricing_Type', data=df)
Out[21]: <Axes: xlabel='Type_of_Cab', ylabel='Surge_Pricing_Type'>
```



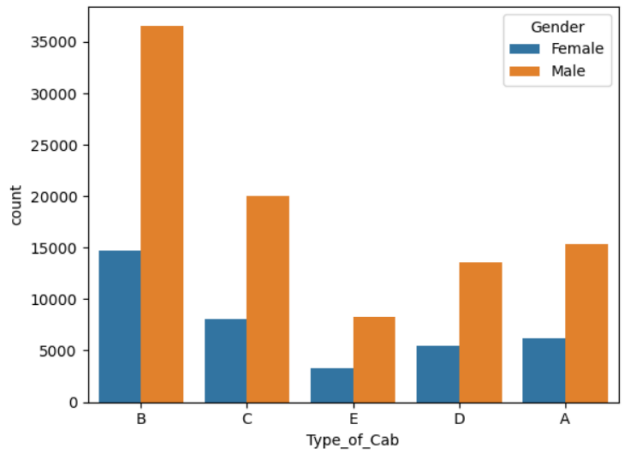
Visualize the relationship between destination type and surge pricing type

```
In [23]: # Create a lineplot to visualize the relationship between destination type and surge pricing type
         sns.lineplot(x='Destination_Type', y='Surge_Pricing_Type', data=df)
Out[23]: <Axes: xlabel='Destination_Type', ylabel='Surge_Pricing_Type'>
```



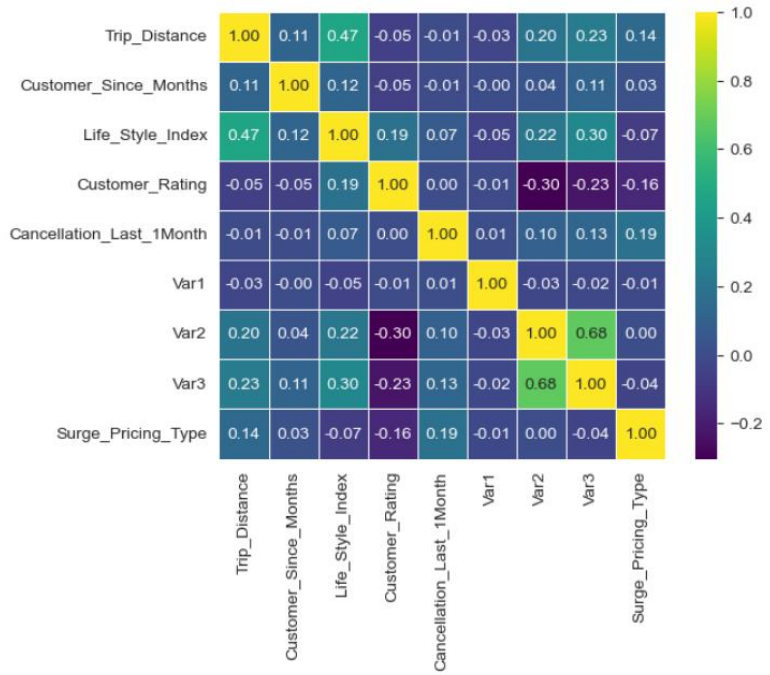
Visualize the relationship between gender and cab type

```
In [27]: # Create a countplot to visualize the relationship between gender and cab type
         sns.countplot(x='Type_of_Cab', hue='Gender', data=df)
Out[27]: <Axes: xlabel='Type_of_Cab', ylabel='count'>
```



The correlation heatmap shows that there is positive correlation of 0.47 between Trip Distance and Life Style Index and also Var 2 and Var3, also Life Style index shows Slightly positive Relationship between Var 2, Var 3 and Customer Rating.

```
In [31]: correlation_matrix=Numeric.corr()
         sns.heatmap(correlation_matrix, annot=True, cmap="viridis", fmt=".2f", linewidths=.5, square=True)
Out[31]: <Axes: >
```

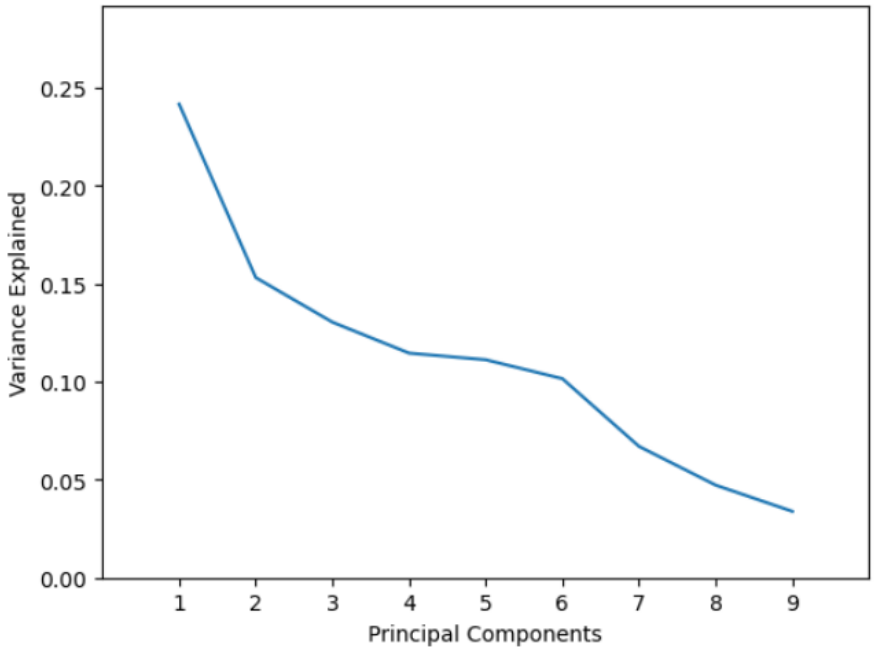



```
In [92]: comp=pd.DataFrame(pca_model.components_, columns=['Trip_Distance', 'Customer_Since_Months', 'Life_Style_Index',
                'Customer_Rating', 'Cancellation_Last_1Month', 'Var1', 'Var2', 'Var3',
                'Surge_Pricing_Type'])
         print(comp)

            Trip_Distance  Customer_Since_Months  Life_Style_Index  Customer_Rating  \
         0       0.389473               0.159124          0.386405        -0.217088
         1       0.340471               0.118593          0.560067         0.607117

            Cancellation_Last_1Month      Var1      Var2      Var3  Surge_Pricing_Type
         0                  0.132469 -0.045461  0.537883  0.560633           0.057553
         1                 -0.151914 -0.076049 -0.242637 -0.128951          -0.288768
```

Each row index with 0 refers to the first principal component. The highest correlation (in absolute terms) between the principal component and the original dataset features is with 'Var 3' feature (0.560633). This means that the with a large value for 'pc1' in the **comp** will have a high 'Var 3' value as well.
For the second principal component, the highest correlation (in absolute terms) is with "Customer_Rating" feature (0.560067) meaning that a with a large value for 'pc2' in the **comp** will have a low "Customer_Rating" value.

From the above, we see that the first principal component explains around 24.1% of the variance of the original dataset, while the second principal component explains around 15.3%. [0.2414846 , 0.15303703]

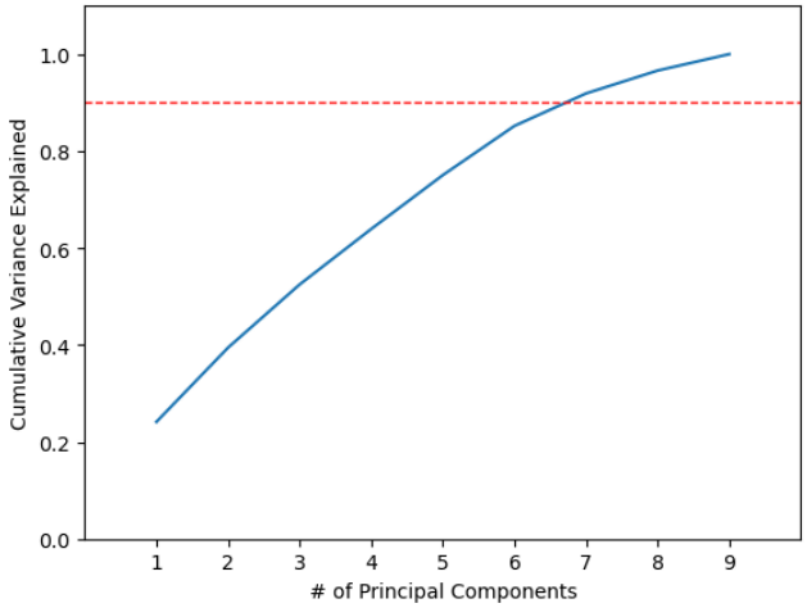
```
In [97]: plt.plot(list(range(1,10)), pca.explained_variance_ratio_)
         plt.axis([0, 10, 0, max(pca.explained_variance_ratio_)+0.05])
         plt.xticks(list(range(1,10)))
         plt.xlabel('Principal Components')
         plt.ylabel('Variance Explained')
         plt.show()
```



We calculate the

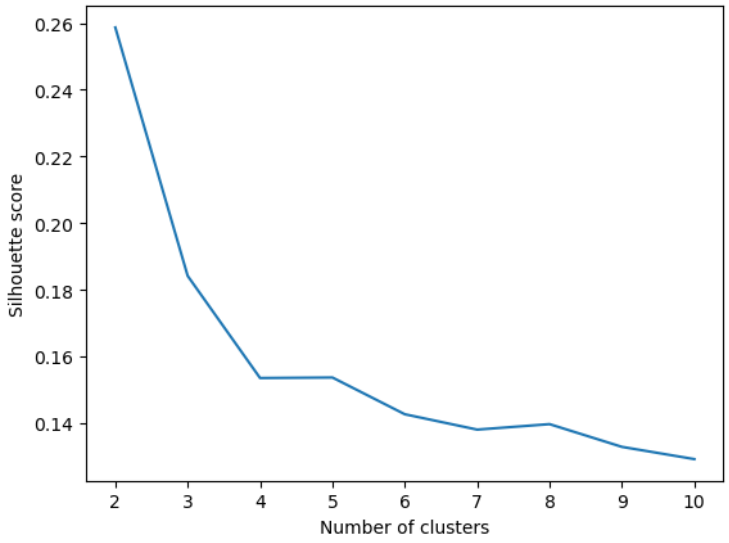
90% of cumulative variance is explained by including 6 principal components. Which reduced the 10 features to 6 which explains over 90% of Variance in the Data set.

```
In [100]: plt.plot(list(range(1,10)), np.cumsum(pca.explained_variance_ratio_))
          plt.axis([0, 10, 0, 1.1])
          plt.axhline(y=0.9, color='r', linestyle='--', linewidth=1)
          plt.xticks(list(range(1,10)))
          plt.xlabel('# of Principal Components')
          plt.ylabel('Cumulative Variance Explained')
          plt.show()
```



1. The first three lines import the necessary packages: pandas for data manipulation, sklearn.cluster.KMeans for K-means clustering, and matplotlib.pyplot for plotting the silhouette scores.
2. The next few lines load the data from a CSV file into a pandas Data Frame and select the relevant columns for analysis.
3. The data is pre-processed by one-hot encoding the 'Gender' column and dropping any rows with missing values.
4. The data is then standardized using Standard Scaler to make sure that each variable has equal importance in the clustering.
5. To determine the optimal number of clusters, the silhouette score is calculated for different values of K (number of clusters) ranging from 2 to 10. The silhouette score measures how well each data point fits into its assigned cluster, with values closer to 1 indicating a better fit.
6. The silhouette scores are plotted against the number of clusters to determine the optimal number of clusters to use for K-means clustering.
7. K-means clustering is performed on the standardized data with K=4 clusters.
8. The resulting cluster labels are added to the original Data Frame.
9. The means of each variable for each cluster are calculated and printed to analyse the characteristics of each cluster.

In [98]:
```
import pandas as pd
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt

# Load the data
df = pd.read_csv('sigma_cabs_kaggle.csv', usecols=['Customer_Since_Months', 'Cancellation_Last_1Month', 'Customer_Rating', 'Var1'

# Preprocess the data
df = pd.get_dummies(df, columns=['Gender'])
df = df.dropna()

# Scale the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
df_scaled = scaler.fit_transform(df)

# Determine the optimal number of clusters
from sklearn.metrics import silhouette_score
silhouette_scores = []
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, random_state=0).fit(df_scaled)
    silhouette_scores.append(silhouette_score(df_scaled, kmeans.labels_))

# Plot the silhouette scores to determine the optimal number of clusters
plt.plot(range(2, 11), silhouette_scores)
plt.xlabel('Number of clusters')
plt.ylabel('Silhouette score')
plt.show()

# Perform K-means clustering
kmeans = KMeans(n_clusters=4, random_state=0).fit(df_scaled)
clusters = kmeans.labels_

# Add the cluster labels to the original data
df['Cluster'] = clusters

# Analyze the clusters
cluster_means = df.groupby('Cluster').mean()
print(cluster_means)
```



The Silhouette score is a measure of how similar an observation is to its own cluster compared to other clusters. The score ranges from -1 to 1, with higher values indicating that the observation is well-matched to its own cluster and poorly-matched to neighbouring clusters.

The x-axis of the graph represents the number of clusters, and the y-axis represents the Silhouette score. The graph shows how the Silhouette score changes as the number of clusters increases.

The purpose of this graph is to help determine the optimal number of clusters for the K-means algorithm. In general, a higher Silhouette score indicates a better clustering solution, and thus the number of clusters with the highest Silhouette score can be considered as the optimal number of clusters. 2 clusters are feasible solution.

Problem Statement:

"Predict the surge pricing type for taxi trips based on the characteristics of the customer, the trip distance, the type of cab, the customer's loyalty, behavior, and satisfaction, as well as the destination category and other relevant variables."

```
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score

# preprocess the data
le = LabelEncoder()


# split the data into training and testing sets
X = df.drop(['Trip_ID', 'Surge_Pricing_Type'], axis=1)
y = df['Surge_Pricing_Type']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# train a random forest classifier model
clf = RandomForestClassifier(n_estimators=100, max_depth=10, random_state=42)
clf.fit(X_train, y_train)

# make predictions on the test set and evaluate the model
y_pred = clf.predict(X_test)
acc = accuracy_score(y_test, y_pred)
print("Accuracy:", acc)
```

✓ 15.9s

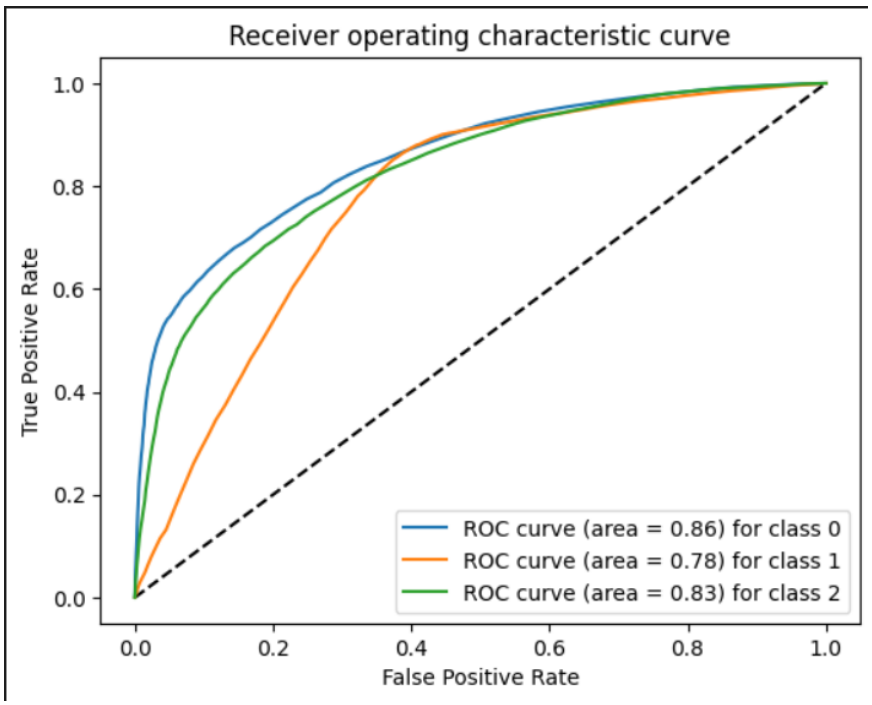
```
Accuracy: 0.6851858884289674
```

Draw meaningful conclusions:

```
Confusion Matrix:
 [[2909 1942  554]
 [ 479 9276 1630]
 [ 472 3231 5840]]
Classification Report:
              precision    recall  f1-score   support

           1       0.75      0.54      0.63      5405
           2       0.64      0.81      0.72     11385
           3       0.73      0.61      0.66      9543

    accuracy                           0.68     26333
   macro avg       0.71      0.65      0.67     26333
weighted avg       0.70      0.68      0.68     26333
```

The ROC curve provides a way to visualize and compare the performance of different classifiers. The ideal classifier would have a ROC curve that passes through the upper left corner of the graph, indicating a TPR of 1 and an FPR of 0. A random classifier would have a diagonal line from the bottom left corner to the top right corner of the graph, indicating that the TPR and FPR are equal regardless of the threshold setting.

The area under the ROC curve (AUC) is a commonly used metric for comparing different classifiers. A perfect classifier has an AUC of 1, while a random classifier has an AUC of 0.5. The higher the AUC value, the better the classifier's performance.



Feature Importance: [0.12480791 0.05755233 0.10545784 0.11719559 0.032733 0.06482676 0.07762807 0.09300875 0.01493569 0.00940334 0.00746156 0.0053901 0.00747626 0.008836 0.00433026 0.00489326 0.00263348 0.00233018 0.00173837 0.00167241 0.0014203 0.00108666 0.00123795 0.0012222 0.00022695 0.00018397 0.08664368 0.0524437 0.03107577 0.05703973 0.02310794]

GitHub Link:

https://github.com/spandyy/Feynn-Labs-Internship/tree/main/Project%202/part%202

4. **Metturu Mouli**

Introduction

Now a day's cab rental services are expanding with the multiplier rate. The ease of using the servicesand flexibility gives their customer a great experience with competitive prices.

Problem Statement

You are a cab rental start-up company. You have successfully run the pilot project and now want to launch your cab service across the country. You have collected the historical data from your pilot projectand now have a requirement to apply analytics for fare prediction. You need to design a system that predicts the fare amount for a cab ride in the city.

Data

Understanding of data is the very first and important step in the process of finding solution of any business problem. Here in our case our company has provided a data set with following features, weneed to go through each and every variable of it to understand and for better functioning.

Size of Dataset Provided: - 16067 rows, 7 Columns (including dependent variable)Missing Values: Yes

Outliers Presented: Yes

Below mentioned is a list of all the variable names with their meanings:

Variables	Description
fare_amount	Fare amount
pickup_datetime	Cab pickup date with time
pickup_longitude	Pickup location longitude
pickup_latitude	Pickup location latitude
dropoff_longitude	Drop location longitude
dropoff_latitude	Drop location latitude
passenger_count	Number of passengers sitting in the cab

Methodology

➢ Pre-Processing

When we required to build a predictive model, we require to look and manipulate the data before we start modelling which includes multiple preprocessing steps such as exploring the data, cleaning the dataas well as visualizing the data through graph and plots, all these steps is combined under one shed whichis **Exploratory Data Analysis**, which includes following steps:

- Data exploration and Cleaning
- Missing values treatment
- Outlier Analysis
- Feature Selection
- Features Scaling

  o Skewness and Log transformation

- Visualization

➢ Modelling

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modelling. Modelling plays an important role to find out the good inferences from the data. Choice of models depends upon the problem statement and data set. As per our problem statement and dataset, we will try some models on our preprocessed data and post comparing the output results wewill select the best suitable model for our problem. As per our data set following models need to be tested:

- Linear regression
- Decision Tree
- Random forest,
- Gradient Boosting

We have also used hyper parameter tunings to check the parameters on which our model runsbest.

Following are two techniques of hyper parameter tuning we have used:

- Random Search CV
- Grid Search CV

➢ Model Selection

The final step of our methodology will be the selection of the model based on the different output andresults shown by different models. We have multiple parameters which we will study further in our report to test whether the model is suitable for our problem statement or not.

Pre-Processing

Data exploration and Cleaning (Missing Values and Outliers)

The very first step which comes with any data science project is data exploration and cleaning whichincludes following points as per this project:

- Separate the combined variables.
- As we know we have some negative values in fare amount so we have to remove those values.
- Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to removethe rows
- having passengers counts more than 6 and less than 1.
- There are some outlier figures in the fare (like top 3 values) so we need to remove those.
- Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rowsif any latitude
- and longitude lies beyond the ranges.

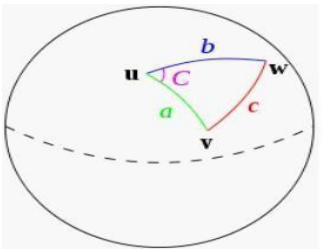
Creating some new variables from the given variables.

Here in our data set our variable name pickup_datetime contains date and time for pickup. So we tried toextract some important variables from pickup_datetime:

- Year
- Month
- Date
- Day of Week
- Hour
- Minute

Also, we tried to find out the distance using the haversine formula which says:

The **haversine formula** determines the great-circle distance between two points on a sphere given theirlongitudes and latitudes. Important in navigation, it is a special case of a more general formula in spherical trigonometry, the law of haversines, that relates the sides and angles of spherical triangles.



So our new extracted variables are:

- fare_amount
- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- passenger_count
- year
- Month
- Date
- Day of Week
- Hour
- Minute
- Distance

Selection of variables

Now as we know that all above variables are of now use so we will drop the redundant variables:

- pickup_datetime
- pickup_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

Now only following variables we will use for further steps:

```
1 train.head()
```

	fare_amount	passenger_count	year	Month	Date	Day	Hour	distance
0	4.5	1	2009	2009	15	0	17	1.030764
1	16.9	1	2010	2010	5	1	16	8.450134
2	5.7	2	2011	2011	18	3	0	1.389525
3	7.7	1	2012	2012	21	5	4	2.799270
4	5.3	1	2010	2010	9	1	7	1.999157

VariableNames	Variable Data Types
fare_amount	float64
passenger_count	object
year	object
Month	object
Date	object
Day of Week	object
Hour	object
distance	float64

Some more data exploration

In this report we are trying to predict the fare prices of a cab rental company. So here we have a data setof 16067 observations with 8 variables including one dependent variable.

Below are the names of Independent variables: passenger_count, year, Month, Date, Day of Week, Hour, distance

Our Dependent variable is: **fare_amount**

Uniqueness in Variable

We need to look at the unique number in the variables which help us to decide whether the variable iscategorical or numeric. So, by using python script 'unique' we tried to find out the unique values in each variable. We have also added the table below:

Variable Name	Unique Counts
fare_amount	450
passenger_count	7
year	7
Month	12
Date	31
Day of Week	7
Hour	24
distance	15424

Dividing the variables into two categories basis their data types:

Continuous variables - 'fare_amount', 'distance'.

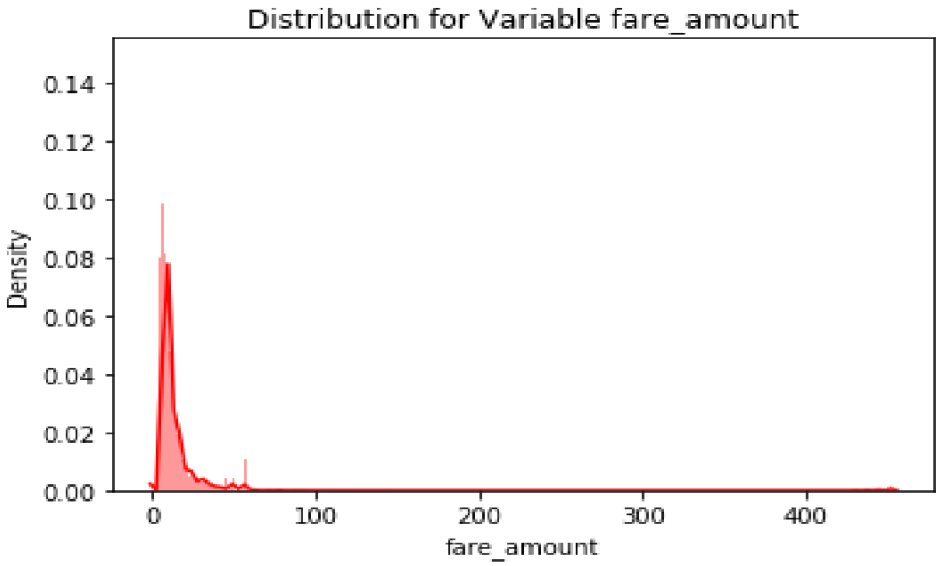
Categorical Variables - 'year', 'Month', 'Date', 'Day of Week', 'Hour', 'passenger_count'

Feature Scaling

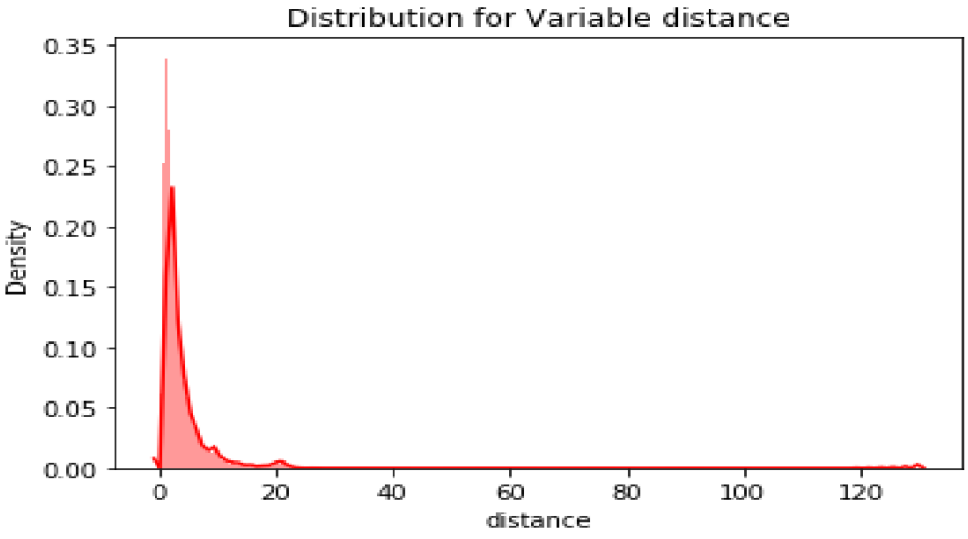
Skewness is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find thatour target variable absenteeism in hours having is one sided skewed so by using **log transform** technique we tried to reduce the skewness of the same.

Below mentioned graphs shows the probability distribution plot to check distribution before logtransformation:

fare_amount

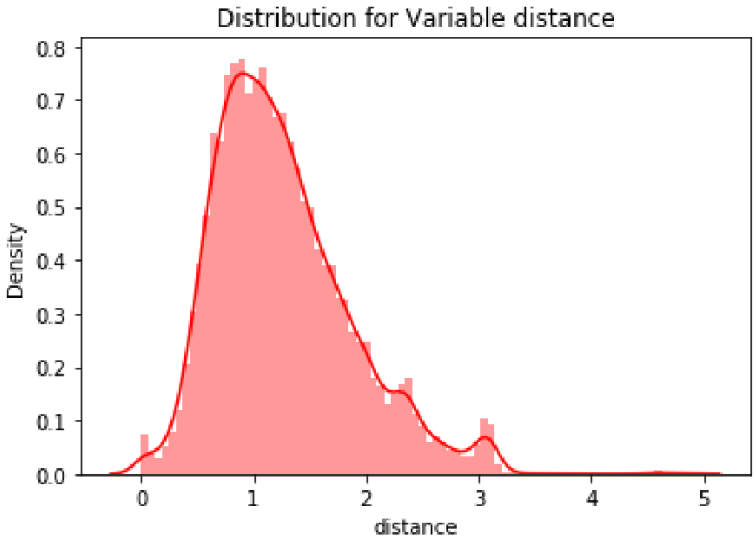


distance

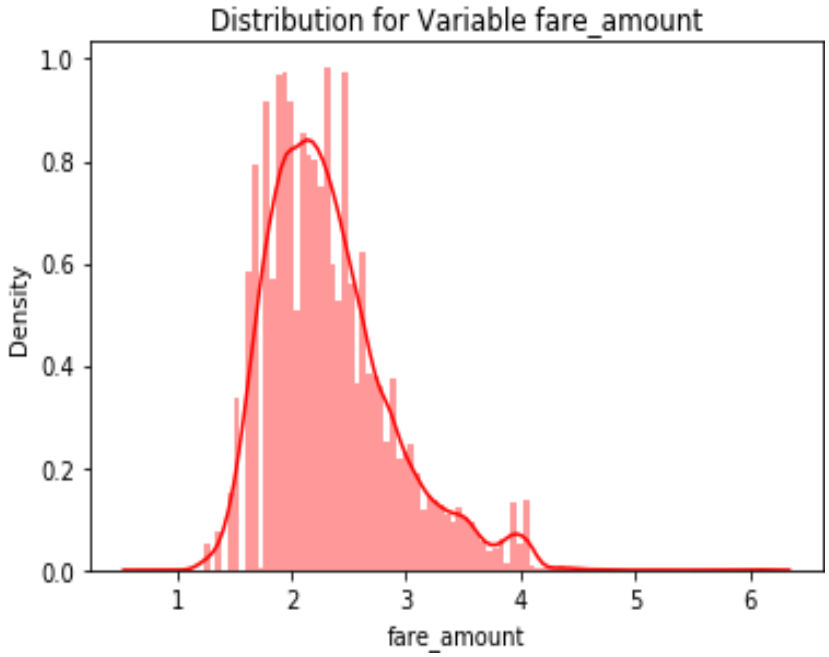


Below mentioned graphs shows the probability distribution plot to check distribution after log transformation:

distance



fare_amount



As our continuous variables appears to be normally distributed so we don't need to use featurescaling techniques like normalization and standardization for the same.

Modelling

After a thorough preprocessing, we will use some regression models on our processed data to predict thetarget variable. Following are the models which we have built –

- Linear Regression
- Decision Tree
- Random Forest
- Gradient Boosting

Before running any model, we will split our data into two parts which is train and test data. Herein our case we have taken 80% of the data as our train data. Below is the snipped image of the split of train test.

```
# Preparing the input features and target label matrices
X=np.array(train.iloc[:,1:])
y=np.array(train.iloc[:,0])
```

```
#Splitting the dataset into train and test dataset
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.2,random_state=1)
```

Linear Regression

Multiple linear regression is the most common form of linear regression analysis. Multiple regression isan extension of simple linear regression. It is used as a predictive analysis, when we want to predict the value of a variable based on the value of two or more other variables. The variable we want to predict iscalled the dependent variable (or sometimes, the outcome, target or criterion variable).

Below is a screenshot of the model we build and its output:

Linear Regression Model

```
#Training the data based on Linear Regression model
model_lr=LinearRegression()
model_lr.fit(X_train,y_train)
```

```
LinearRegression()
```

```
#Predicting the model on train data and test data
train_pred_lr=model_lr.predict(X_train)
test_pred_lr=model_lr.predict(X_test)
```

```
##calculating RMSE for test data
test_rmse_lr = np.sqrt(mean_squared_error(y_test, test_pred_lr))

##calculating RMSE for train data
train_rmse_lr= np.sqrt(mean_squared_error(y_train, train_pred_lr))
```

```
print("Root Mean Squared Error For Training data = ",train_rmse_lr)
print("Root Mean Squared Error For Test data = ",test_rmse_lr)


print("R2 score for training data is",r2_score(y_train,train_pred_lr))
print("R2 score for testing data is",r2_score(y_test,test_pred_lr))
```

```
Root Mean Squared Error For Training data =  0.2762019841074455
Root Mean Squared Error For Test data =  0.24628672006420377
R2 score for training data is 0.7479265933156383
R2 score for testing data is 0.7811405225793144
```

Decision Tree

A tree has many analogies in real life, and turns out that it has influenced a wide area of machine learning, covering both classification and regression. In decision analysis, a decision tree can be used tovisually and explicitly represent decisions and decision making. As the name goes, it uses a tree-like model of decisions.

Below is the screenshot of the query we executed and the result shown, we will compare the results ofeach model in a combined table later on.

Decision Tree Model

```
#Training the data using Decision Tree model
model_dt=DecisionTreeRegressor(max_depth=2)
model_dt.fit(X_train,y_train)
train_pred_dt=model_dt.predict(X_train)
test_pred_dt=model_dt.predict(X_test)
```

```
##calculating RMSE for test data
test_rmse_dt = np.sqrt(mean_squared_error(y_test, test_pred_dt))
##calculating RMSE for train data
train_rmse_dt= np.sqrt(mean_squared_error(y_train, train_pred_dt))
```

```
print("Root Mean Squared Error For Training data = ",train_rmse_dt)
print("Root Mean Squared Error For Test data = ",test_rmse_dt)


print("R2 score for training data is",r2_score(y_train,train_pred_dt))
print("R2 score for testing data is",r2_score(y_test,test_pred_dt))
```

```
Root Mean Squared Error For Training data =  0.2996210902077019
Root Mean Squared Error For Test data =  0.2867460617158617
R2 score for training data is 0.7033678616157003
R2 score for testing data is 0.7033268167661035
```

Random Forest

Random forests or random decision forests are an ensemble learning method for classification, regression and other task, that operate by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of theindividual trees. Random decision forests correct for decision trees' habit of overfitting to their training set.

To say it in simple words: Random forest builds multiple decision trees and merges them togetherto get a more accurate and stable prediction.

Below is a screenshot of the model we build and its output:

Random Forest Regressor Model

```
#Training the data using Random Forest Regressor
model_rf=RandomForestRegressor(n_estimators=101)  #n_esimators means No.of Trees
model_rf.fit(X_train,y_train)
train_pred_rf=model_rf.predict(X_train)
test_pred_rf=model_rf.predict(X_test)
```

```
#calculating RMSE for test data
test_rmse_rf = np.sqrt(mean_squared_error(y_test, test_pred_rf))
#calculating RMSE for train data
train_rmse_rf= np.sqrt(mean_squared_error(y_train, train_pred_rf))
```

```
print("Root Mean Squared Error For Training data = ",train_rmse_rf)
print("Root Mean Squared Error For Test data = ",test_rmse_rf)


print("R2 score for training data is",r2_score(y_train,train_pred_rf))
print("R2 score for testing data is",r2_score(y_test,test_pred_rf))
```

```
Root Mean Squared Error For Training data =  0.09633796876442949
Root Mean Squared Error For Test data =  0.2392909845201166
R2 score for training data is 0.9693332033708648
R2 score for testing data is 0.7933972771351916
```

Gradient Boosting

Gradient boosting is a machine learning technique for regression and classification problems, which produces a prediction model in the form of an ensemble of weak prediction models, typically decision trees. It builds the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function.

Below is a screenshot of the model we build and its output:

Gradient Boosting Regressor Model

```
#training the data using Gradient Boosting model
model_gb=GradientBoostingRegressor()
model_gb.fit(X_train,y_train)
train_pred_gb=model_gb.predict(X_train)
test_pred_gb=model_gb.predict(X_test)
```

```
##calculating RMSE for test data
test_rmse_gb = np.sqrt(mean_squared_error(y_test, test_pred_gb))

##calculating RMSE for train data
train_rmse_gb= np.sqrt(mean_squared_error(y_train, train_pred_gb))
```

```
print("Root Mean Squared Error For Training data = ",train_rmse_gb)
print("Root Mean Squared Error For Test data = ",test_rmse_gb)


print("R2 score for training data is",r2_score(y_train,train_pred_gb))
print("R2 score for testing data is",r2_score(y_test,test_pred_gb))
```

```
Root Mean Squared Error For Training data =  0.2278465768661526
Root Mean Squared Error For Test data =  0.22814258725278425
R2 score for training data is 0.8284627438023151
R2 score for testing data is 0.812199781789624
```

Hyper Parameters Tunings for optimizing the results

Model hyperparameters are set by the data scientist ahead of training and control implementation aspects of the model. The weights learned during training of a linear regression model are parameters while the number of trees in a random forest is a model hyperparameter because this is set by the data scientist. Hyperparameters can be thought of as model settings. These settings need to be tuned for each problem because the best model hyperparameters for one particular dataset will not be the best across all datasets. The process of hyperparameter tuning (also called hyperparameter optimization) means finding the combination of hyperparameter values for a machine learning model that performs the best - as measured on a validation dataset - for a problem.

Here we have used two hyper parameters tuning techniques

- Random Search CV
- Grid Search CV

1. **Random Search CV**: This algorithm set up a grid of hyperparameter values and select random combinations to train the model and score. The number of search iterations is set based on time/resources.
2. **Grid Search CV**: This algorithm set up a grid of hyperparameter values and for each combination, train a model and score on the validation data. In this approach, every single combination of hyperparameters values is tried which can be very inefficient.

```python
from sklearn.model_selection import train_test_split,RandomizedSearchCV
##Random Search CV on Random Forest Model

model_rrf = RandomForestRegressor(random_state = 0) #rrf=Rondom forest regressor
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
               'max_depth': depth}

randomcv_rf = RandomizedSearchCV(model_rrf, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)#cv=changed
randomcv_rf = randomcv_rf.fit(X_train,y_train)
predictions_RRF = randomcv_rf.predict(X_test)

view_best_params_RRF = randomcv_rf.best_params_

best_model = randomcv_rf.best_estimator_

predictions_RRF = best_model.predict(X_test)

#R^2
RRF_r2 = r2_score(y_test, predictions_RRF)
#Calculating RMSE
RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))

print('Random Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_RRF)
print('R-squared = {:0.2}.'.format(RRF_r2))
print('RMSE = ',RRF_rmse)
```

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.8.
RMSE =  0.23761684842855696
```

Check results after using Random Search CV on Random forest and gradient boosting model.

```
##Random Search CV on gradient boosting model

model_gbr = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
depth = list(range(1,100,2))

# Create the random grid
rand_grid = {'n_estimators': n_estimator,
               'max_depth': depth}

randomcv_gb = RandomizedSearchCV(gb, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
randomcv_gb = randomcv_gb.fit(X_train,y_train)
predictions_gb = randomcv_gb.predict(X_test)

view_best_params_gb = randomcv_gb.best_params_

best_model = randomcv_gb.best_estimator_

predictions_gb = best_model.predict(X_test)

#R^2
gb_r2 = r2_score(y_test, predictions_gb)
#Calculating RMSE
gb_rmse = np.sqrt(mean_squared_error(y_test,predictions_gb))

print('Random Search CV Gradient Boosting Model Performance:')
print('Best Parameters = ',view_best_params_gb)
print('R-squared = {:0.2}.'.format(gb_r2))
print('RMSE = ', gb_rmse)
```

```
Random Search CV Gradient Boosting Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.77.
RMSE =  0.25205774939967956
```

Check results after using Grid Search CV on Random forest and gradient boosting model:

```
## Grid Search CV for gradinet boosting
gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)
gridcv_gb = gridcv_gb.fit(X_train,y_train)
view_best_params_Ggb = gridcv_gb.best_params_

#Apply model on test data
predictions_Ggb = gridcv_gb.predict(X_test)

#R^2
Ggb_r2 = r2_score(y_test, predictions_Ggb)
#Calculating RMSE
Ggb_rmse = np.sqrt(mean_squared_error(y_test,predictions_Ggb))

print('Grid Search CV Gradient Boosting regression Model Performance:')
print('Best Parameters = ',view_best_params_Ggb)
print('R-squared = {:0.2}.'.format(Ggb_r2))
print('RMSE = ',(Ggb_rmse))
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.8.
RMSE =  0.23739342063769528
```

#Final parameter extraction for Gradient Boosting Model

```
## Grid Search CV for gradinet boosting
gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))

# Create the grid
grid_search = {'n_estimators': n_estimator,
               'max_depth': depth}

## Grid Search Cross-Validation with 5 fold CV
gridcv_gb = GridSearchCV(gb, param_grid = grid_search, cv = 5)
gridcv_gb = gridcv_gb.fit(X_train,y_train)
view_best_params_Ggb = gridcv_gb.best_params_

#Apply model on test data
predictions_Ggb = gridcv_gb.predict(X_test)

#R^2
Ggb_r2 = r2_score(y_test, predictions_Ggb)
#Calculating RMSE
Ggb_rmse = np.sqrt(mean_squared_error(y_test,predictions_Ggb))

print('Grid Search CV Gradient Boosting regression Model Performance:')
print('Best Parameters = ',view_best_params_Ggb)
print('R-squared = {:0.2}.'.format(Ggb_r2))
print('RMSE = ',(Ggb_rmse))
```

```
Grid Search CV Gradient Boosting regression Model Performance:
Best Parameters =  {'max_depth': 5, 'n_estimators': 19}
R-squared = 0.8.
RMSE =  0.23739342063769528
```

Conclusion:

Model Evaluation

The main concept of looking at what is called residuals or difference between our predictions f(x[I,]) and actual outcomes y[i].

In general, most data scientists use two methods to evaluate the performance of the model:

I. **RMSE** (Root Mean Square Error): is a frequently used measure of the difference between values predicted by a model and the values actually observed from the environment that is being modelled.

$$RMSE = \sqrt{\frac{\sum_{i=1}^{n} (X_{obs,i} - X_{mo\ del,i})^2}{n}}$$

II. **R Squared(R^2):** is a statistical measure of how close the data are to the fitted regression line. It is also known as the coefficient of determination, or the coefficientof multiple determination for multiple regression. In other words, we can say it explains as to how much of the variance of the target variable is explained.

III. We have shown both train and test data results, the main reason behind showing boththe results is to check whether our data is overfitted or not.

Below table shows the model results before applying hyper tuning:

Model Name	**RMSE**		**R Squared**	
	Train	**Test**	**Train**	**Test**
`Linear Regression	0.27	0.25	0.74	0.78
Decision Tree	0.30	0.28	0.70	0.70
Random Forest model	0.09	0.23	0.96	0.79
Gradient Boosting	0.22	0.22	0.82	0.81

Below table shows results post using hyper parameter tuning techniques:

Model Name	**Parameter**	**RMSE (Test)**	**R Squared (Test)**
Random Search CV	Random Forest	0.23	0.80
	Gradient Boosting	0.25	0.77
Grid Search CV	Random Forest	0.23	0.80
	Gradient Boosting	0.23	0.80

Above table shows the results after tuning the parameters of our two best suited models i.e. Random Forest and Gradient Boosting. For tuning the parameters, we have used Random Search CV and Grid Search CV under which we have given the range of n_estimators, depth and CV folds.

Model Selection

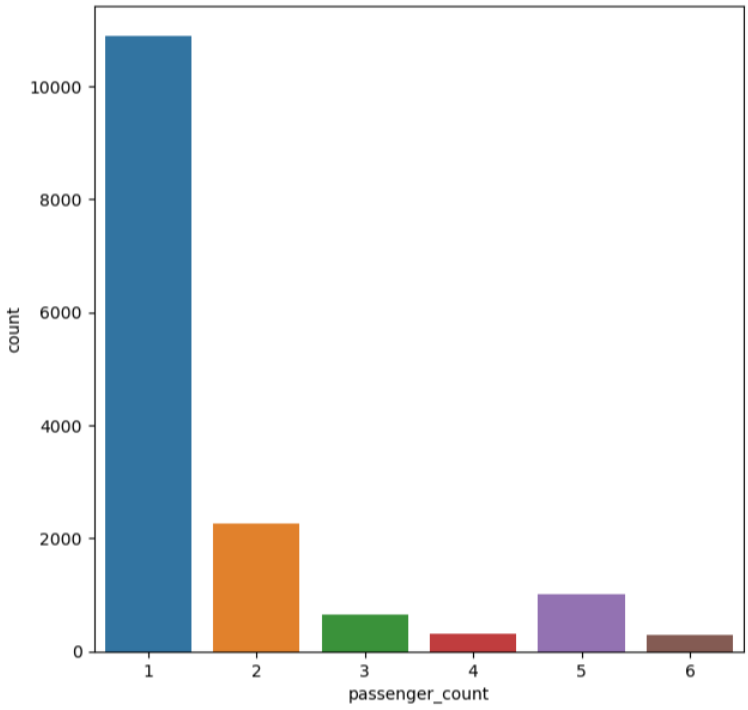
On the basis RMSE and R Squared results a good model should have least RMSE and max R Squared value. So, from above tables we can see:

- From the observation of all RMSE Value and R-Squared Value we have concluded that,
- Both the models- Gradient Boosting Default and Random Forest perform comparatively well while comparing their RMSE and R-Squared value.
- After this, I chose Random Forest CV and Grid Search CV to apply cross validation technique and see changes brought about by that.
- After applying tunings Random forest model shows best results compared to gradient boosting.
- So finally, we can say that Random forest model is the best method to make prediction for thisproject with highest explained variance of the target variables and lowest error chances with parameter tuning technique Grid Search CV.

Some more visualization facts:

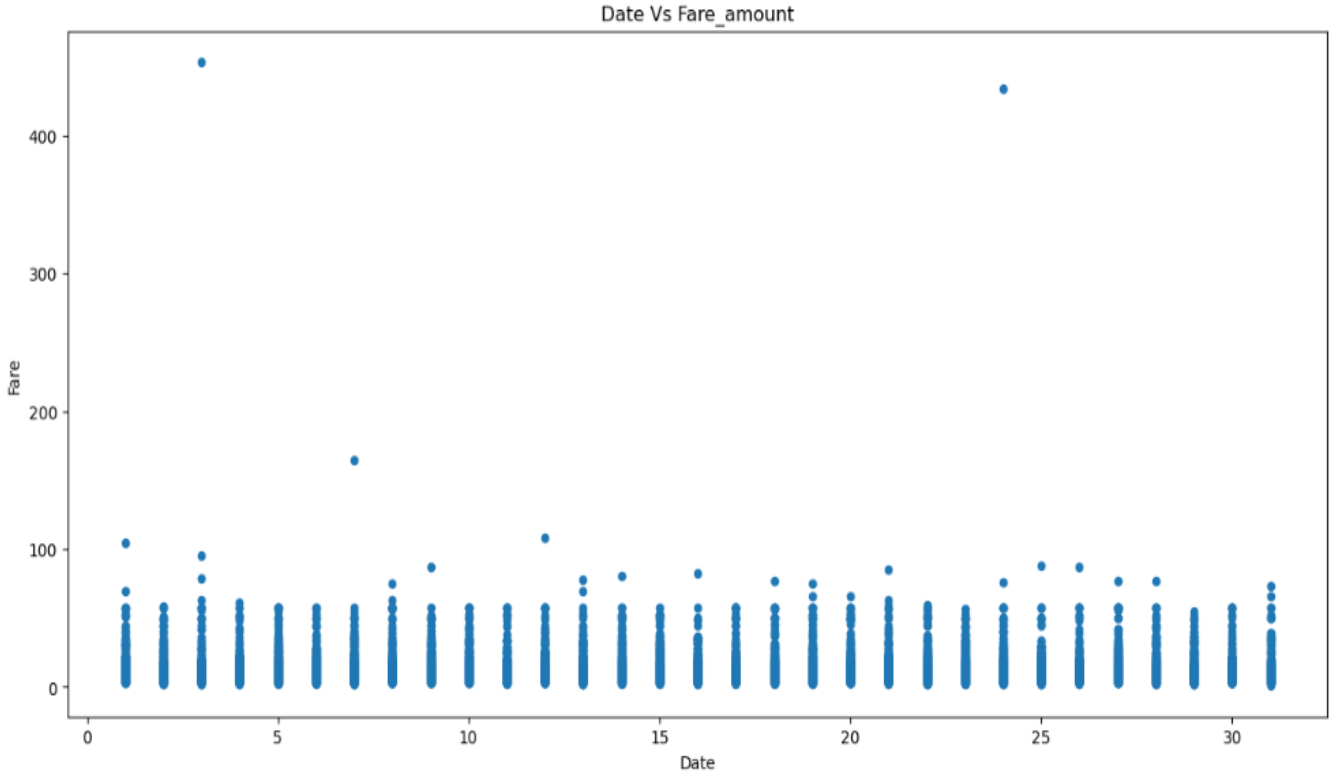
Number of passengers and fare

- We can see in below graph that single passengers are the most frequent travelers, and the highest fare also seems to come from cabs which carry just 1 passenger.



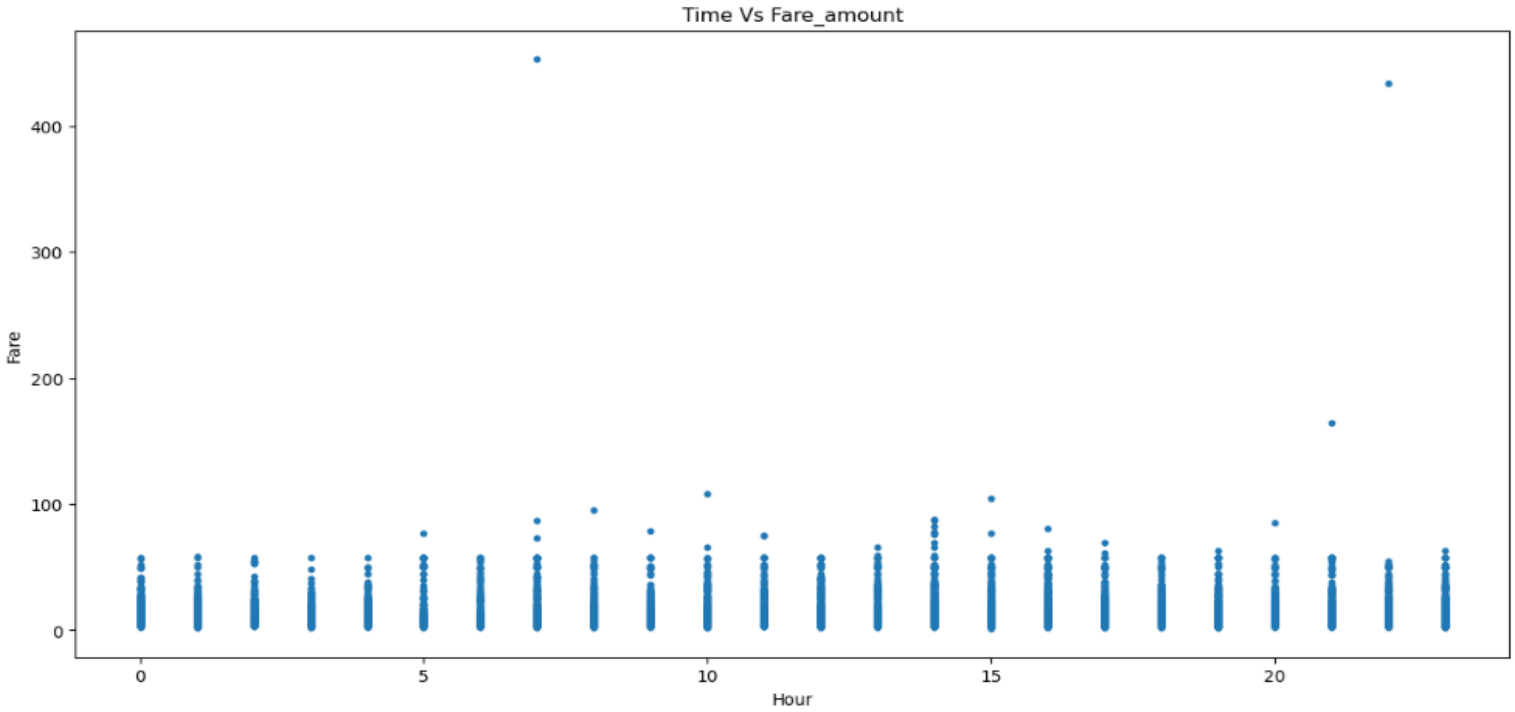
Date of month and fares

- The fares throughout the month mostly seem uniform.



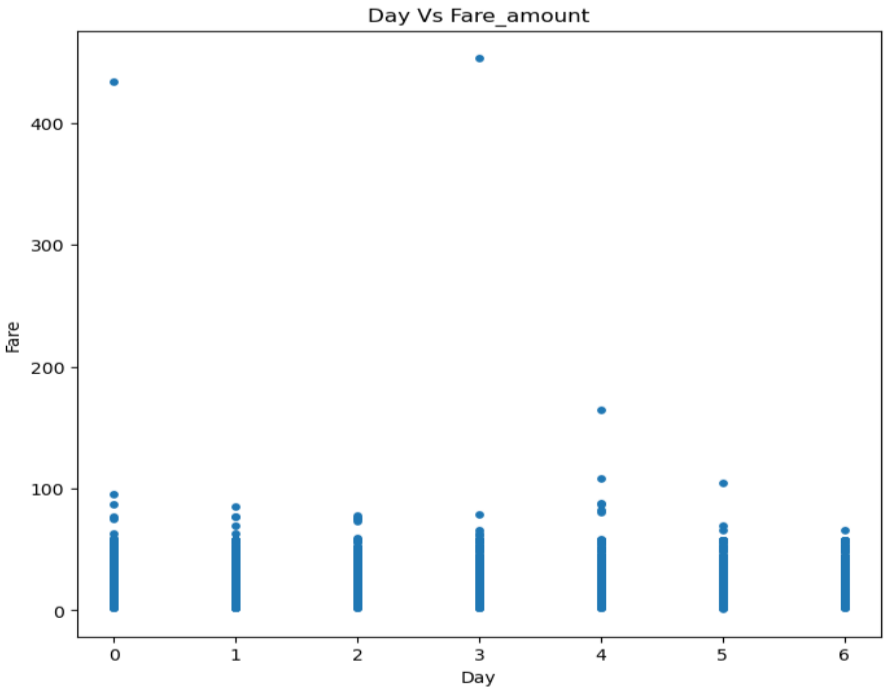
Hours and Fares

- During hours 6 PM to 11PM the frequency of cab boarding is very due to peak hours
- Fare prices during 2PM to 8PM is bit high compared to all other time might be due to highdemands.

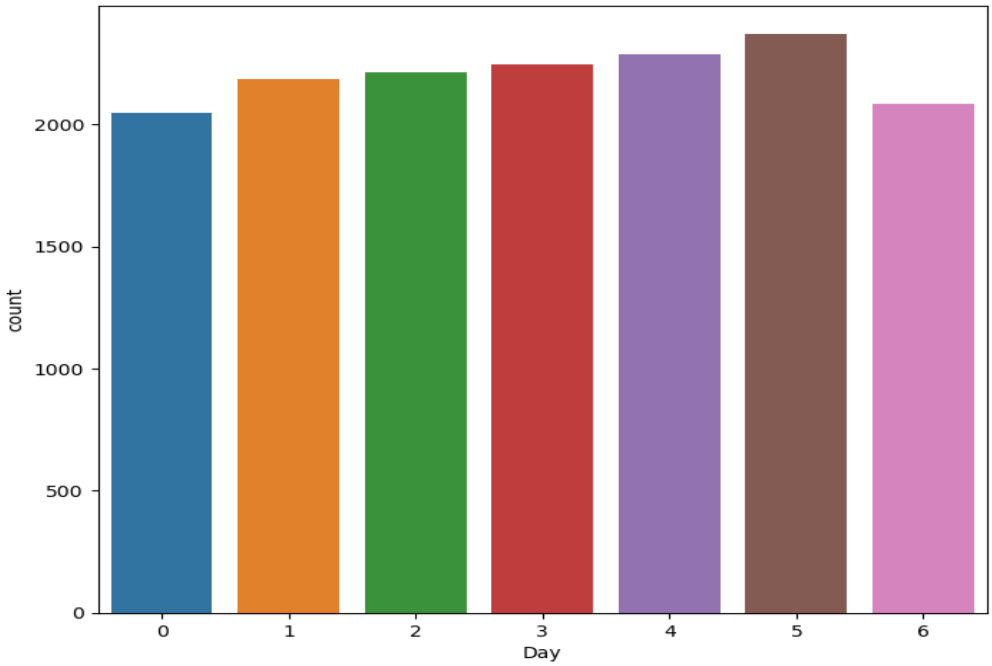


Week Day and fare

- Cab fare is high on Friday, Saturday and Monday, may be during weekend and first day of theworking day they charge high fares because of high demands of cabs



Impact of Day on the Number of Cab rides :



Observation : The day of the week does not seem to have much influence on the number of cabs rides.

GitHub Link:

https://github.com/moulimetturu/Car-Fare-Prediction/blob/main/Car%20Fare%20Prediction.ipynb

5. Akshay Pankar

PROBLEM STATEMENT: There is a huge demand of used cars in the Indian Market today. As sale of new car have slowed down in the recent past, the pre-owned car market has continued to grow over the past year and is larger than the new car market now. Consider this: In 2018-19, while new car sales were recorded at 3.6 million units, around 4 million second-hand cars were bought and sold. There is a slowdown in new car sales and that could mean that the demand is shifting towards the pre-owned market. In fact, some car sellers replace their old cars with pre-owned cars instead of buying new ones.

The goal of the case is as follows:

- The one major factor that holds anyone back while buying a used car is the price (target variable).You will be predicting the Price of used cars with the data collected from various sources and distributed across various locations in India.
- Compare different models and find out which one is the most suitable in this case in predicting the prices.

Read the data

```
Name                 object
Location             object
Year                float64
Kilometers_Driven   float64
Fuel_Type            object
Transmission         object
Owner_Type           object
Mileage              object
Engine               object
Power                object
Colour               object
Seats               float64
No. of Doors        float64
New_Price            object
Price               float64
dtype: object
```

Get the info

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5961 entries, 0 to 5960
Data columns (total 15 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Name               5961 non-null   object
 1   Location           5950 non-null   object
 2   Year               5959 non-null   float64
 3   Kilometers_Driven  5953 non-null   float64
 4   Fuel_Type          5961 non-null   object
 5   Transmission       5934 non-null   object
 6   Owner_Type         5946 non-null   object
 7   Mileage            5959 non-null   object
```

```
 8   Engine             5944 non-null   object
 9   Power              5929 non-null   object
 10  Colour             5950 non-null   object
 11  Seats              5956 non-null   float64
 12  No. of Doors       5960 non-null   float64
 13  New_Price          824 non-null    object
 14  Price              5961 non-null   float64
dtypes: float64(5), object(10)
memory usage: 698.7+ KB
```

Get the Summary Statistics

	count	unique	top	freq	mean	std	min	25%	50%	75%	max
Name	5961	212	Maruti Swift	343	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Location	5950	11	Mumbai	781	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Year	5959.0	NaN	NaN	NaN	2013.389159	3.243051	1998.0	2011.5	2014.0	2016.0	2019.0
Kilometers_Driven	5953.0	NaN	NaN	NaN	58711.100118	91712.207172	171.0	33931.0	53000.0	73000.0	6500000.0
Fuel_Type	5961	5	Diesel	3188	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Transmission	5934	2	Manual	4225	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Owner_Type	5946	4	First	4875	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Mileage	5959	439	18.9 kmpl	172	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Engine	5944	143	1197 CC	606	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Power	5929	369	74 bhp	233	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Colour	5950	3	White	2115	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Seats	5956.0	NaN	NaN	NaN	5.26914	0.789048	2.0	5.0	5.0	5.0	10.0
No. of Doors	5960.0	NaN	NaN	NaN	4.114933	0.344757	2.0	4.0	4.0	4.0	5.0
New_Price	824	540	4.78 Lakh	6	NaN	NaN	NaN	NaN	NaN	NaN	NaN
Price	5961.0	NaN	NaN	NaN	9.528103	11.214382	0.44	3.5	5.66	10.0	160.0

Check for Null Values

```
Name                    0
Location               11
Year                    2
Kilometers_Driven       8
Fuel_Type               0
Transmission           27
Owner_Type             15
Mileage                 2
Engine                 17
Power                  32
Colour                 11
Seats                   5
No. of Doors            1
New_Price            5137
Price                   0
dtype: int64
```

% Null values

```
Name                  0.000000
Location              0.208808
Year                  0.037965
Kilometers_Driven     0.151860
Fuel_Type             0.000000
Transmission          0.512528
Owner_Type            0.284738
Mileage               0.037965
Engine                0.322703
Power                 0.607441
Colour                0.208808
Seats                 0.094913
No. of Doors          0.018983
New_Price            97.513288
Price                 0.000000
dtype: float64
```

Since New_Price has over 97% Null values, we would drop this column for now

	Name	Location	Year	Kilometers_Driven	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Colour	Seats	No. of Doors	Price
0	Mahindra Scorpio	Pune	2012.0	99000.0	Diesel	Manual	Third	12.05 kmpl	2179 CC	120 bhp	Black/Silver	8.0	5.0	6.00
1	Maruti Baleno	Kochi	2018.0	18678.0	Petrol	Manual	First	21.1 kmpl	998 CC	100 bhp	Others	5.0	4.0	8.32
2	Mahindra Xylo	Bangalore	2013.0	197000.0	Diesel	Manual	First	11.68 kmpl	2498 CC	112 bhp	White	7.0	5.0	4.00
3	Hyundai Grand	Delhi	2014.0	45000.0	Diesel	Manual	First	24.0 kmpl	1120 CC	70 bhp	White	5.0	4.0	3.49
4	Toyota Innova	Delhi	2011.0	65000.0	Diesel	Manual	First	12.8 kmpl	2494 CC	102 bhp	Others	8.0	5.0	6.40

Check for Object Data Type – Using Head (5)

	Name	Location	Fuel_Type	Transmission	Owner_Type	Mileage	Engine	Power	Colour
0	Mahindra Scorpio	Pune	Diesel	Manual	Third	12.05 kmpl	2179 CC	120 bhp	Black/Silver
1	Maruti Baleno	Kochi	Petrol	Manual	First	21.1 kmpl	998 CC	100 bhp	Others
2	Mahindra Xylo	Bangalore	Diesel	Manual	First	11.68 kmpl	2498 CC	112 bhp	White
3	Hyundai Grand	Delhi	Diesel	Manual	First	24.0 kmpl	1120 CC	70 bhp	White
4	Toyota Innova	Delhi	Diesel	Manual	First	12.8 kmpl	2494 CC	102 bhp	Others

It seems that the columns Mileage, Engine , Power needs to be cleaned and converted to Float/int dtype

	Name	Location	Fuel_Type	Transmission	Owner_Type	Colour
0	Mahindra Scorpio	Pune	Diesel	Manual	Third	Black/Silver
1	Maruti Baleno	Kochi	Petrol	Manual	First	Others
2	Mahindra Xylo	Bangalore	Diesel	Manual	First	White
3	Hyundai Grand	Delhi	Diesel	Manual	First	White
4	Toyota Innova	Delhi	Diesel	Manual	First	Others

Create a new column 'Brand' which has the brand name of the Cars

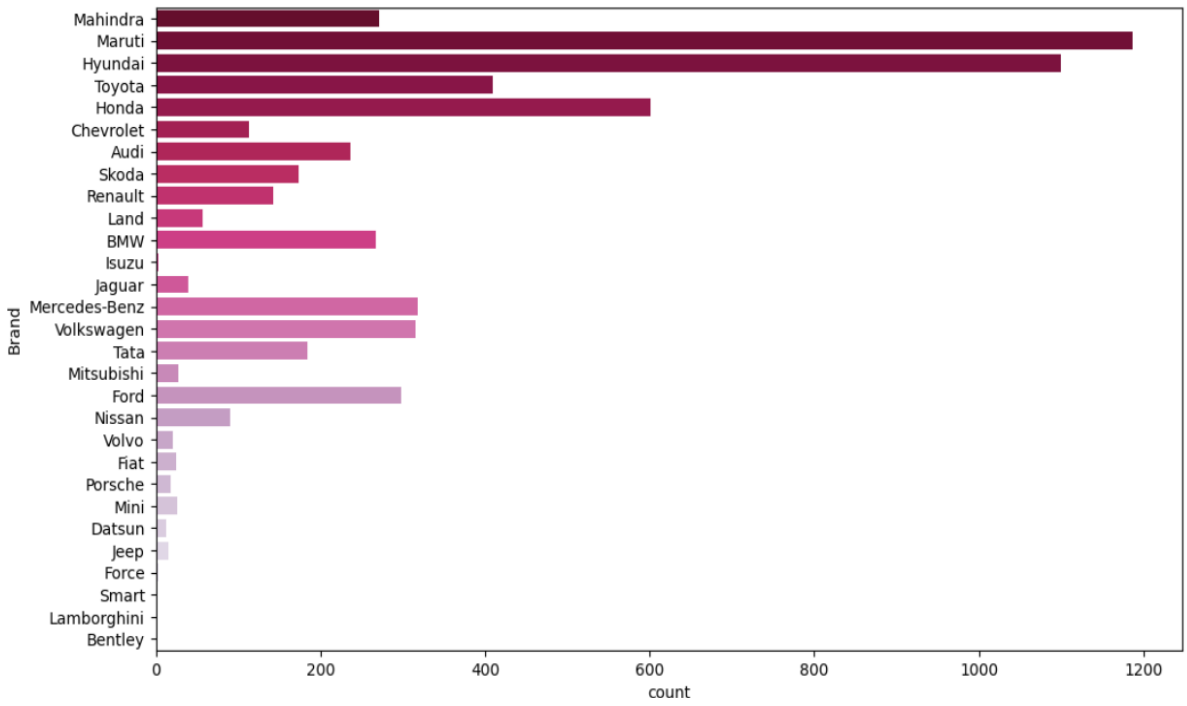
```
cars['Brand']=cars.Name.apply(lambda x:x.split(' ')[0])
```

```
cars.Brand.value_counts()
```

```
Maruti           1187
Hyundai          1100
Honda             601
Toyota            410
Mercedes-Benz     318
Volkswagen        315
Ford              298
Mahindra          272
BMW               267
Audi              236
Tata              184
Skoda             173
Renault           143
Chevrolet         113
Nissan             91
Land               57
Jaguar             40
Mitsubishi         27
Mini               26
Fiat               25
Volvo              21
Porsche            18
Jeep               15
Datsun             13
Force               3
ISUZU               2
Isuzu               1
Smart               1
Lamborghini         1
Bentley             1
Name: Brand, dtype: int64
```

- Brand Isuzu is available twice in different cases ('ISUZU', 'Isuzu'). To avoid this being considered as 2 different brands, correct to single format

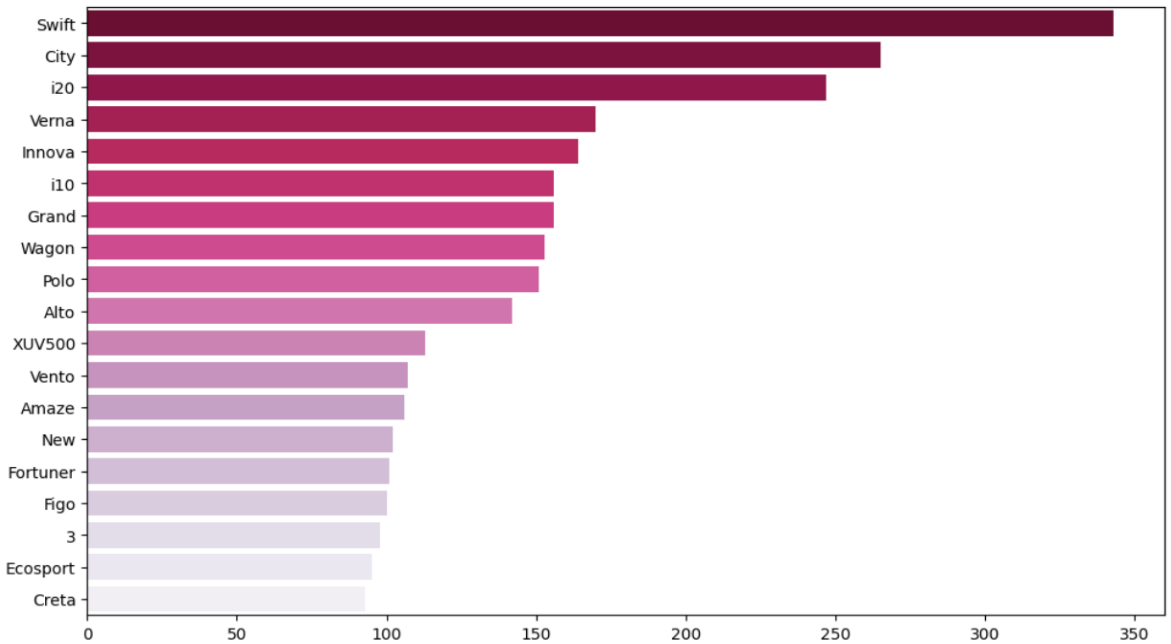
```
plt.figure(figsize=(12,7))

sns.countplot(y=(cars.Brand),palette='PuRd_r');
```



plt.figure(figsize=(12,7))

sns.barplot(y=cars.Model.value_counts()[cars.Model.value_counts()>90].index,

x=cars.Model.value_counts()[cars.Model.value_counts()>90].values,palette='PuRd_r');

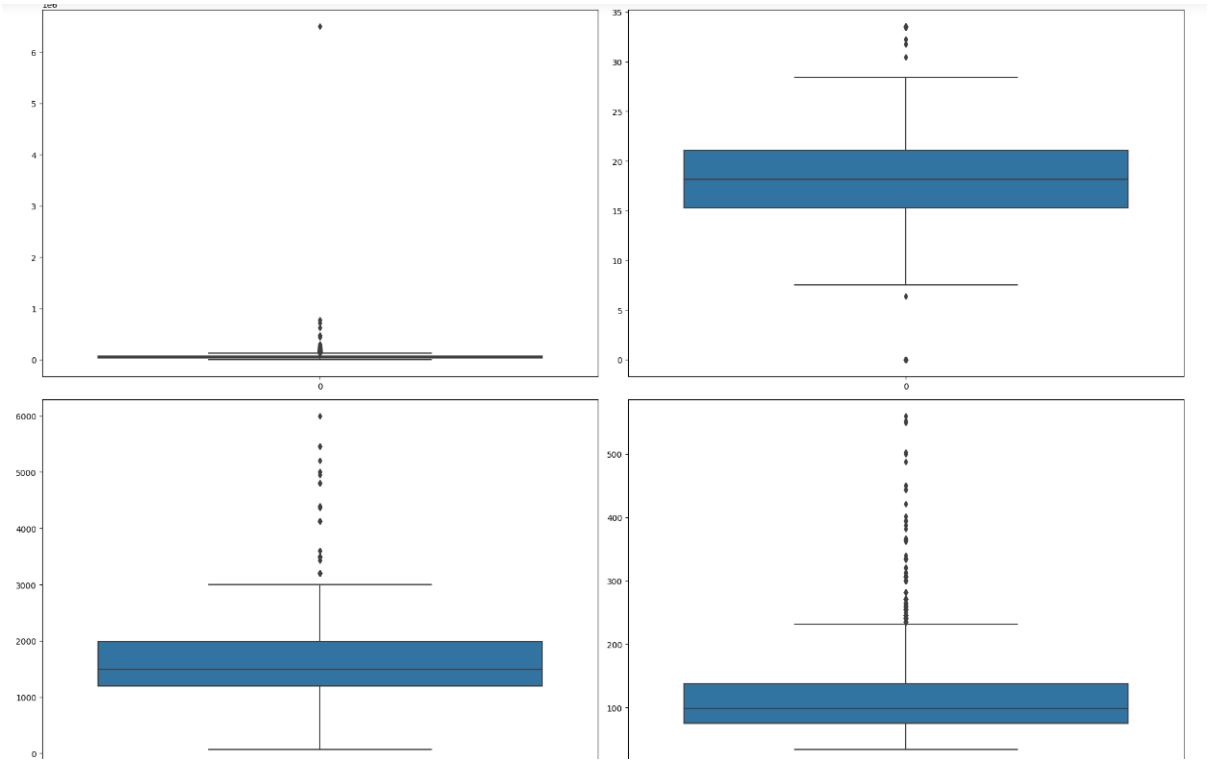


- There is a car by the Model name 'New', let's check which car is it
- There is a car by the Model name '3', let's check which car is it

Detail Procedure given in python File.

Check for Outliers

```
data_plot=cars[['Kilometers_Driven', 'Mileage', 'Engine', 'Power','CarAge']]

fig=plt.figure(figsize=(20,20))

for i in range(0,len(data_plot.columns)):

    ax=fig.add_subplot(3,2,i+1)

    sns.boxplot(data_plot[data_plot.columns[i]])

    plt.tight_layout()

print('Shape before Outliers Treatment',cars.shape)
```



Seems like Kilometers_Driven, Power, Engine have many outliers

Outliers Treatment

```
Q1 = cars.quantile(0.25) # Getting First quantile for all numerical variables

Q3 = cars.quantile(0.75) # Getting Second quantile for all numerical variables

IQR = Q3 - Q1          # Getting IQR values
```

lower_range= Q1-(1.5 * IQR) #Getting Lower Limit for all numerical variables

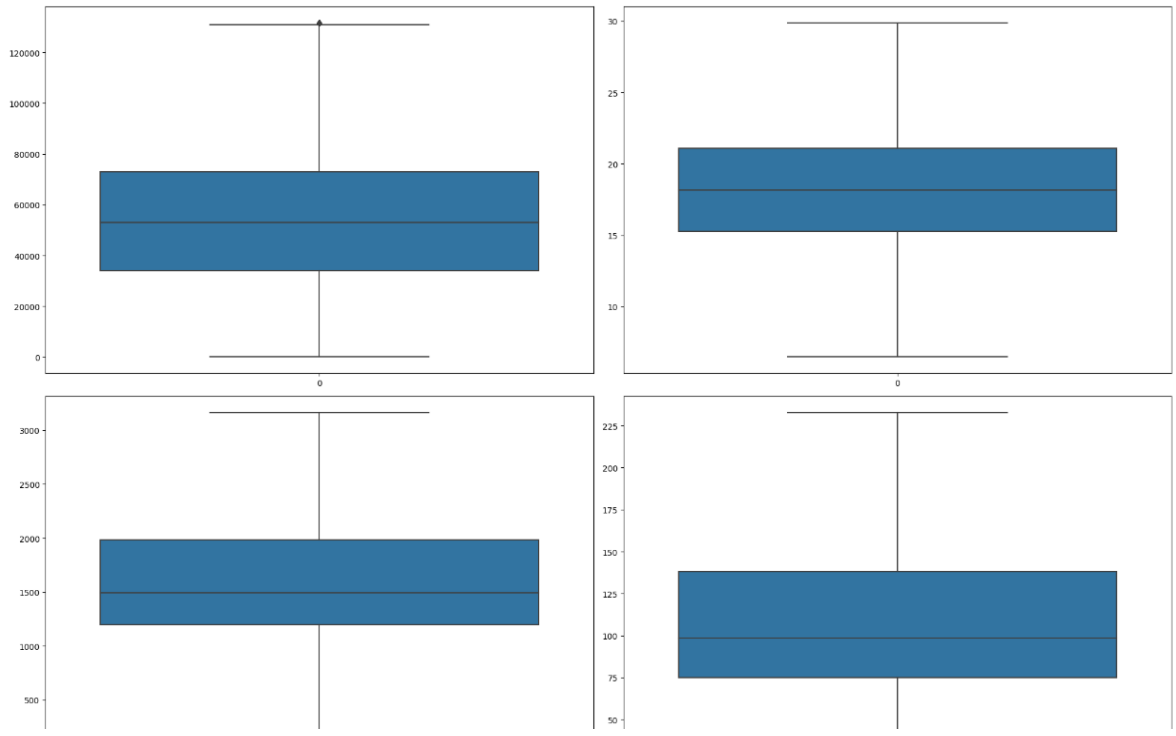
upper_range= Q3+(1.5 * IQR) #Getting Upper Limit for all numerical variables

More codes for UR and LR are in python file.

```
data_plot=cars[['Kilometers_Driven', 'Mileage', 'Engine', 'Power','CarAge']]

fig=plt.figure(figsize=(20,20))

for i in range(0,len(data_plot.columns)):

    ax=fig.add_subplot(3,2,i+1)

    sns.boxplot(data_plot[data_plot.columns[i]])

    plt.tight_layout()
```

```
print('Shape After Outliers Treatment',cars.shape)
```

Shape After Outliers Treatment (5959, 18)



Since this was Outliers Treatment by imputing UL and LL in place of Outliers thus, the number of rows didn't ch ange but had it been Outliers Removal, the number of rows would have changed.

Null Values Imputation

```
Index(['Location', 'Year', 'Kilometers_Driven', 'Transmission', 'Owner_Type',
       'Mileage', 'Engine', 'Power', 'Colour', 'Seats', 'No. of Doors',
       'Cars_Category', 'CarAge'],
      dtype='object')
```

Simple Imputer

The SimpleImputer class provides basic strategies for imputing missing values. Missing values can be imputed with a provided constant value, or using the statistics (mean, median or most frequent) of each column in which the missing values are located. This class also allows for different missing values encodings.

In [45]:
```
#you can use any other method as well
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'median',verbose=0)
imputer= imputer.fit(cars[non_objects].iloc[:,:])
```

```
C:\Users\Shikhar Shrivastava\anaconda3\lib\site-packages\sklearn\impute\_base.py:356: FutureWarning: The 'verbose' parameter was deprecated in version 1.1 and will be removed in 1.3. A warning will always be raised upon the removal of empty columns in the future version.
  warnings.warn(
```

In [46]:
```
cars[non_objects]=imputer.transform(cars[non_objects])
```

Impute the Object values using Mode

In [47]:
```
from sklearn.impute import SimpleImputer
imputer = SimpleImputer(missing_values = np.nan, strategy = 'most_frequent',verbose=0)
imputer= imputer.fit(cars[objects].iloc[:,:])
```

```
C:\Users\Shikhar Shrivastava\anaconda3\lib\site-packages\sklearn\impute\_base.py:356: FutureWarning: The 'verbose' parameter was deprecated in version 1.1 and will be removed in 1.3. A warning will always be raised upon the removal of empty columns in the future version.
  warnings.warn(
```

In [48]:
```
cars[objects]=imputer.transform(cars[objects])
```

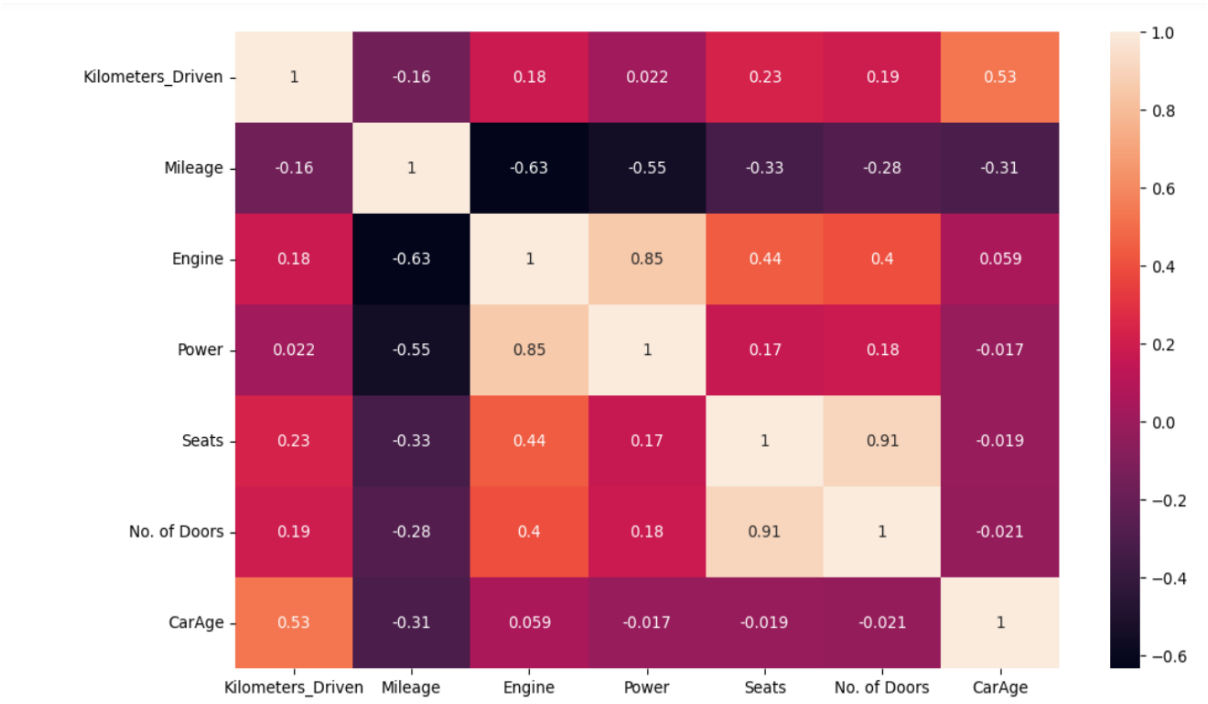
In [49]:
```
cars.isnull().sum()
```

Output :

```
Name                0
Location            0
Year                0
Kilometers_Driven   0
Fuel_Type           0
Transmission        0
Owner_Type          0
Mileage             0
Engine              0
Power               0
Colour              0
Seats               0
No. of Doors        0
Price               0
Brand               0
Model               0
Cars_Category       0
CarAge              0
dtype: int64
```

There are no Null values now

Correlation Plot



Conclusion :-

- Engine & Power, Seats & No.of Doors -> highly correlated
- For algorithms that are affected by correlation in independent variables, drop one of the variables in that set (Say: Engine and No. of Doors)

Encode the Data

There are two types of categorical data

- Ordinal: Order based like 'good','bad','worst' or Clothing sizes
- Nominal: Without any order or ranks like city names, Genders, etc

Here, for now let us convert these into Categorical using pd.Categories to avoid high dimensionality because of OHE(get_dummies)

You are free to use any encoding technique as long as it works.

Also, rememeber that on Official Site of Scikit-learn's Label Encoder it is mentioned that "This transformer should be used to encode target values, i.e. y, and not the input X

```
cars.select_dtypes(include='object').describe()
```

	Name	Location	Fuel_Type	Transmission	Owner_Type	Colour	Brand	Model	Cars_Category
count	5959	5959	5959	5959	5959	5959	5959	5959	5959
unique	212	11	5	2	4	3	29	208	2
top	Maruti Swift	Mumbai	Diesel	Manual	First	White	Maruti	Swift	Budget_Friendly
freq	343	792	3187	4250	4888	2126	1187	343	4520

```
cars["Cars_Category"]=cars["Cars_Category"].replace({"Budget_Friendly":1,"Not_Budget_Friendly":0})
```

```
for feature in cars.columns:
    if cars[feature].dtype == 'object':
        cars[feature] = pd.Categorical(cars[feature]).codes

        cars.Cars_Category=pd.Categorical(cars.Cars_Category).codes
```

```
cars.dtypes
```

```
Name                 int16
Location              int8
Year                float64
Kilometers_Driven   float64
Fuel_Type             int8
Transmission          int8
Owner_Type            int8
Mileage             float64
Engine              float64
Power               float64
Colour                int8
Seats               float64
No. of Doors        float64
Price                int32
Brand                 int8
Model                int16
Cars_Category         int8
CarAge              float64
dtype: object
```

Split the Data

Make 2 models using Decision Tree Regressor and Linear Regression and comparing the RMSE to find the best model

- Check Train and Test RMSE
- Check Train and Test Scores

x=cars.drop(['Price','Name','Year'],axis=1) # Year is transformed to CarAge

y=cars.Price

x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=123,test_size=0.30)

Inferences Drawn from Decision Tree

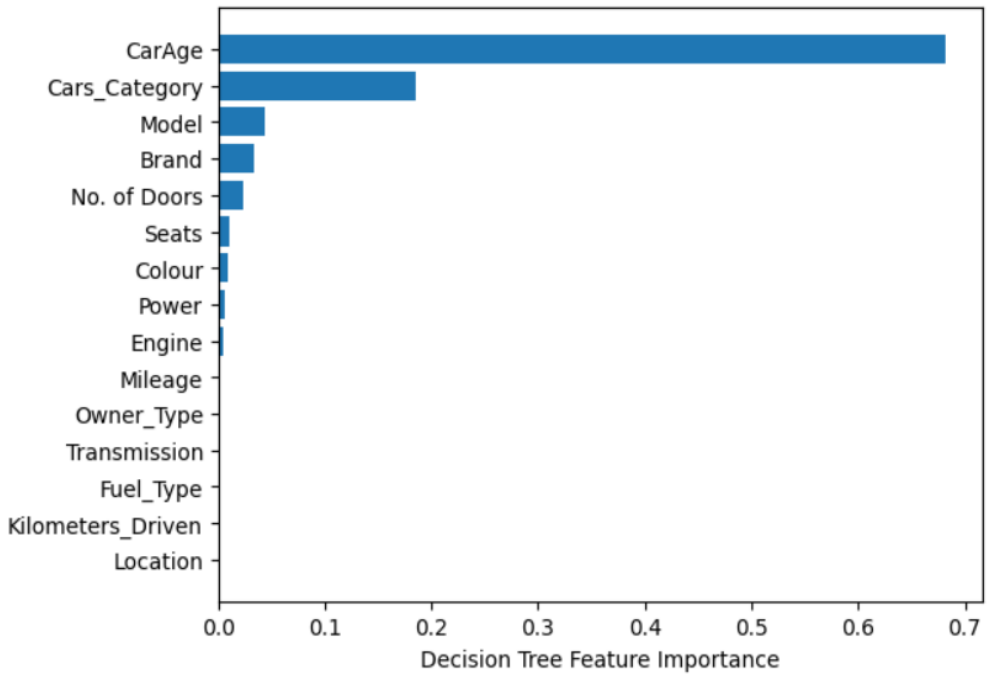
	Train RMSE	Test RMSE	Training Score	Test Score
Linear Regression	6.135898	6.101724	0.702456	0.706992
Decision Tree Regressor	0.018964	4.974656	0.999997	0.805240

Looks like Decision Tree, is over-fitting because of the difference in values between train and test RMSE., Let's Grid Search to get the best parameters

You can also visualise the tree and see where to prune it and decide the max_depth and other parameters

- Note : Hyperparameter tuning will be covered in coming sections

Grid Search on Decision Tree (Pruning the tree)



So from the above we are clearly able to see 'CarAge' is the most important feature and it has a most effect over the price of the car , whereas 'Location' is the least important feature and thus has the least effect on price of the car.

top 5 most important features are : **['CarAge','Cars_Category','Model','Brand','No.of Doors']**

Make 3 models using Decision Tree Classifier , Logistic Regression and LDA and comparing the Accuracy to find the best model

- Check Train and Test Accuracy to see that there is no huge Over/Under fitting

Codes In python File.

Output

	Train Accuracy	Test Accuracy
Decision Tree Classifier	0.999760	0.940716
LDA	0.925438	0.930089
Logistic Regression	0.910573	0.907159

Looks like Decision Tree Classifier, is under-fitting because train accuracy > test accuracy ., Let's Grid Search to get the best parameters or prune the tree

Grid Search CV:

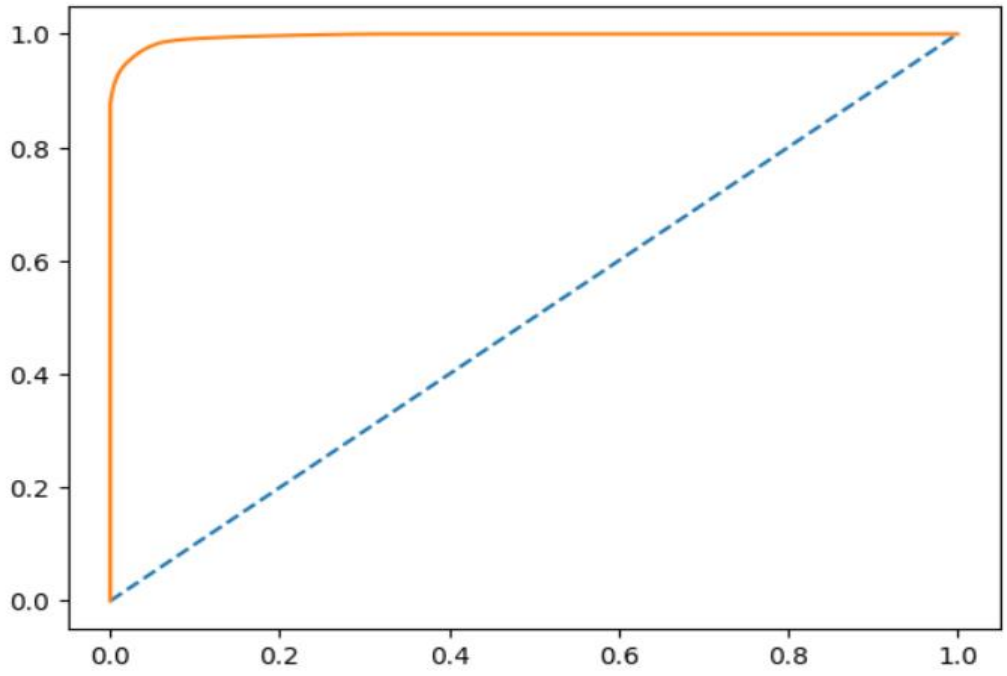
Output

	Train Accuracy	Test Accuracy
Decision Tree Classifier	0.973867	0.967002
LDA	0.925438	0.930089
Logistic Regression	0.910573	0.907159

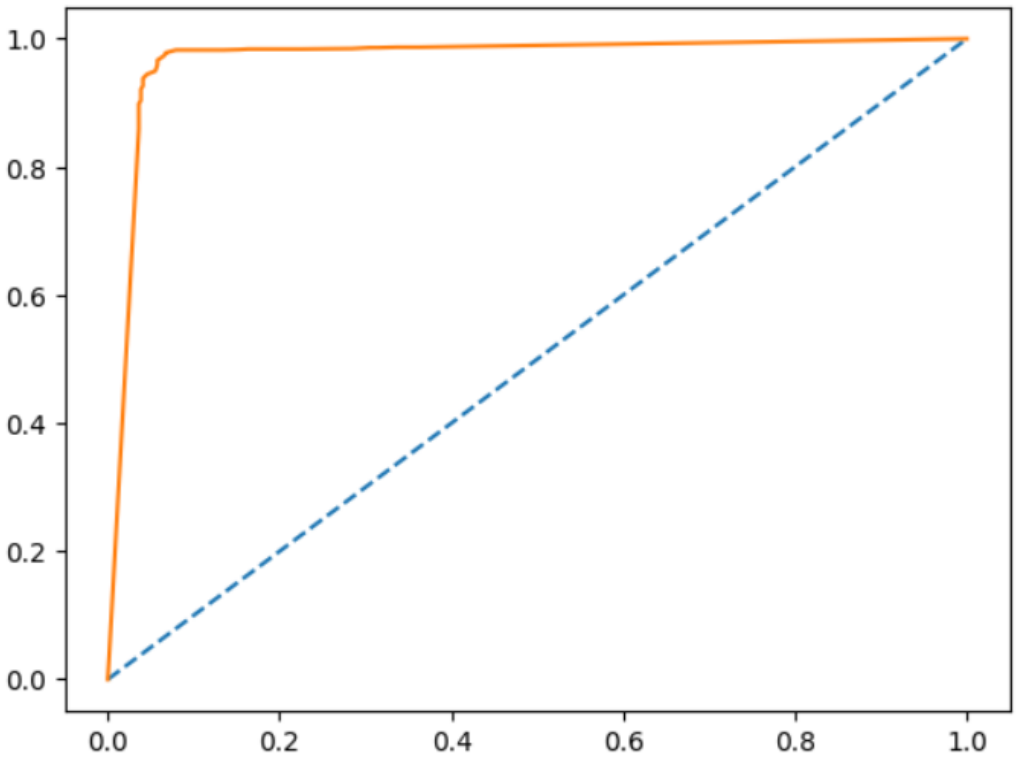
Clearly now the underfitting of the model in Decison tree is reduced and Decison tree classifier results in the best accuracy score thus this model will be selected for classification

ROC AUC curve

AUC: 0.996



AUC: 0.996



ROC AUC Curve values for best model indicates that there is high level of seperatibility among the classes of the target variable

Inference on test data:

For predicting price <10 lakh (Label 1: 'Budget Friendly'):

Precision (98%) – 98% of cars predicted are actually 'Budget Friendly' (price <10 lakh) out of all cars predicted to have (price <10 lakh).

Recall (98%) – Out of all the cars actually having (price <10 lakh), 98% of cars have been predicted correctly.

For predicting price>= 10lakh (Label 0: 'Not Budget Friendly'):

Precision (93%) – 93% of cars predicted are actually 'Not Budget Friendly' (price >=10 lakh) oout of all cars predicted to have (price >=10 lakh).

Recall (93%) – Out of all the cars actually having (price >=10 lakh), 93% of cars have been predicted correctly.

Overall accuracy of the model – 97 % of total predictions are correct

Accuracy, AUC, Precision and Recall for test data is almost inline with training data.

This proves no overfitting or underfitting has happened, and overall, the model is a good model for classification