Project Report

On

Cab fare Prediction

By

Swapnil Samudre

INDEX

1. Introduction
    1.1 Problem Statement.
    1.2 Data and Dataset.

2. Methodology
    2.1 Pre-Processing (EDA).
    2.2 Modeling.
    2.3 Model Selection.

3. Pre-processing
    3.1 Data exploration and Cleaning (Missing Values and Outliers)
    3.2 Creating some new variables from the given variables
    3.3 Selection of variables
    3.4 Dependent Variable
    3.5 Independent Variables
    3.6 Uniqueness of Variables
    3.7 Dividing the variables categories
    3.8 Feature Scaling
    3.9 Data Visualization

4. Modeling
    - Linear Regression
    - Decision Tree
    - Random Forest
    - Hyper-Parameter Tuning

5. Conclusion
    - Model Evaluation/Tuning
    - Model Selection

1 Introduction

Now a day’s cab rental services are emerging as more people are preferring cab over their own vehicle because of feasible and affordable fare rates.

1.1 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 project and 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.

1.2 Data and DataSet

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; we need to go through each and every variable of it to understand and for better functioning.
Size of Dataset Provided: - Train Data -16067 rows, 7 Columns (including dependent variable) and Test Data 9914 rows, 6 Columns (Excluding dependent variable)
Missing Values: Yes
Outliers Presented: Yes

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

2 Methodology

- **Pre-Processing (EDA):**

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

1. Data exploration and Cleaning
2. Missing values Analysis
3. Outlier Analysis
4. Feature Selection
5. Features Scaling
6. Visualization

- **Modeling:**
  - Linear regression
  - Decision Tree
  - Random forest

Once all the Pre-Processing steps has been done on our data set, we will now further move to our next step which is modeling. Modeling 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 we will select the best suitable model for our problem. As per our data set following models need to be tested:

- We have also used hyper parameter tunings to check the parameters on which our model runs best. 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 and results 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.

3 Pre-processing:

3.1 Data exploration and Cleaning (Missing Values and Outliers)

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

a. Separate the combined variables.

b. As we know we have some negative values in fare amount so we have to re move those values.

c. Passenger count would be max 6 if it is a SUV vehicle not more than that. We have to remove the rows having passengers counts more than 6 and less than 1.

d. There are some outlier figures in the fare (like top 3 values) so we need to remove those.

e. Latitudes range from -90 to 90. Longitudes range from -180 to 180. We need to remove the rows if any latitude and longitude lies beyond the ranges.

3. 2 Creating New variables:

Here in our data set our variable name pickup_datetime contains date and time for pickup. So we tried to extract 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 their longitudes 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

3.3 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_longitute:
- Pickup_latitute:
- dropoff_longitute:
- dropoff_latitude:
- Minute

3.4 Dependent Variable:

"passenger_count, Year, Month, Date, Day , Hour, distance".

3.5 Independent Variable:

"fare_amount"

3.6 Uniqueness in Variable:

We need to look at the unique number in the variables which help us to decide whether the variable is categorical or numeric. So, by using python script 'nunique' we tried to find out the unique values in each variable.

```
In [100]: train.nunique()
```

```
Out[100]: fare_amount        449
          passenger_count      6
          Year                 7
          Month               12
          Date                31
          Day                  7
          Hour                24
          distance         15424
          dtype: int64
```

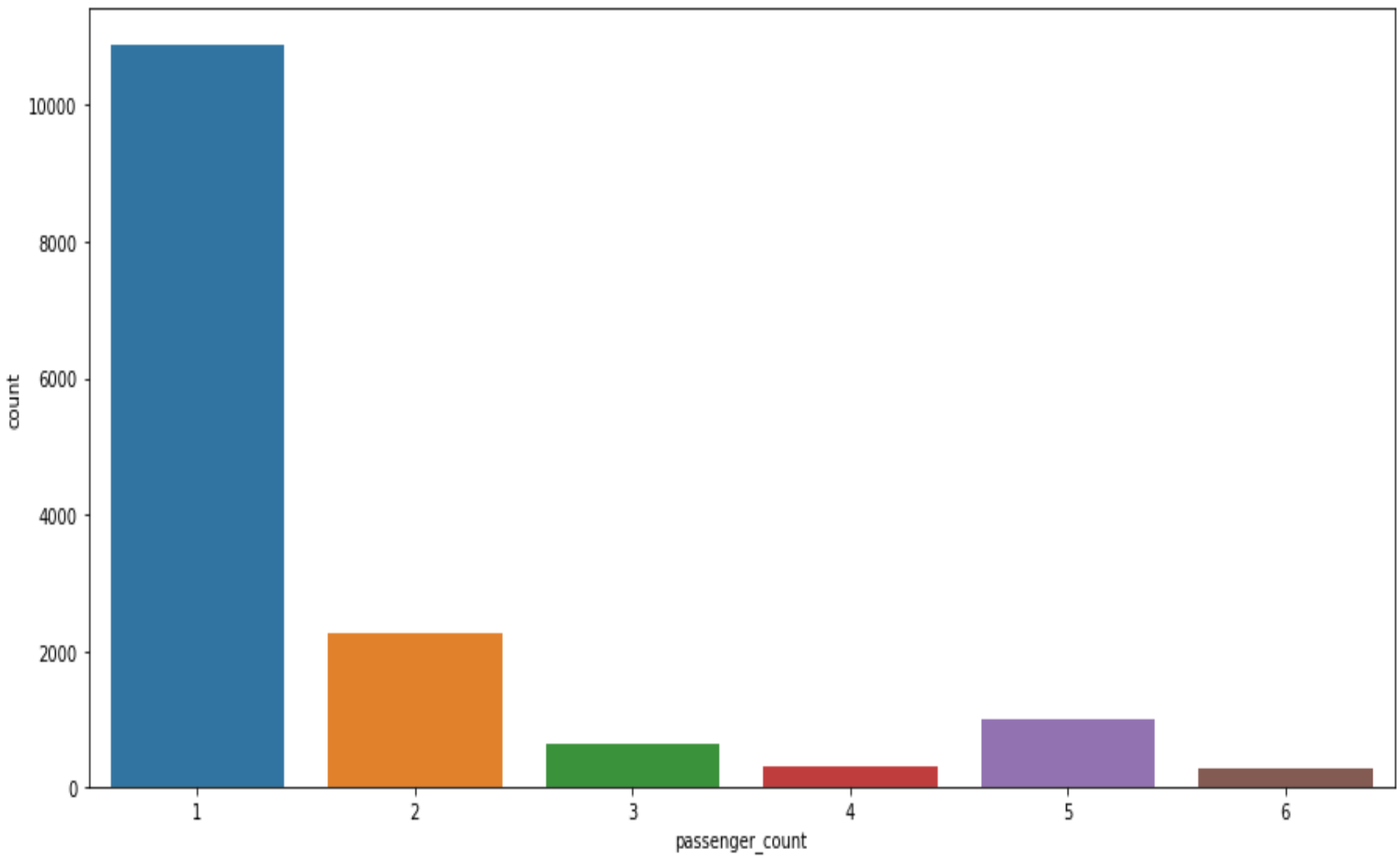
3.7 Feature Scaling:

Most of the times, your dataset will contain features highly varying in magnitudes, units and range. But since, most of the machine learning algorithms use Eucledian distance between two data points in their computations, this is a problem.

If left alone, these algorithms only take in the magnitude of features neglecting the units. The results would vary greatly between different units, 5kg and 5000gms. The features with high magnitudes will weigh in a lot more in the distance calcula tions than features with low magnitude. To supress this effect, we need to bring all features to the same level of magnitudes.

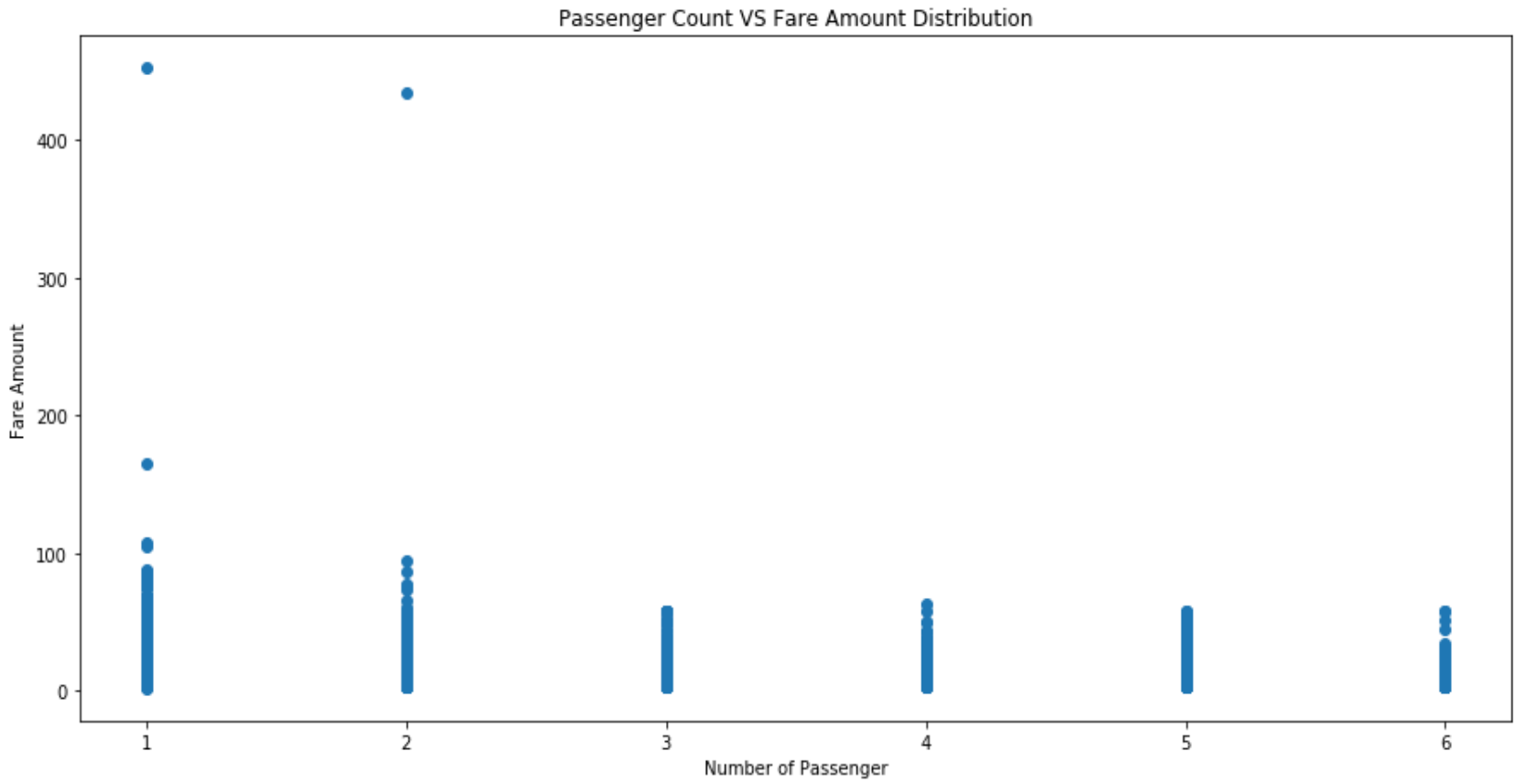
In our Dataset the data are Skewed, **Skewness** is asymmetry in a statistical distribution, in which the curve appears distorted or skewed either to the left or to the right. Skewness can be quantified to define the extent to which a distribution differs from a normal distribution. Here we tried to show the skewness of our variables and we find that our 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.

3.8 Data Visualization:



Here we can see that passenger_count of '1' and '2' are more preferred

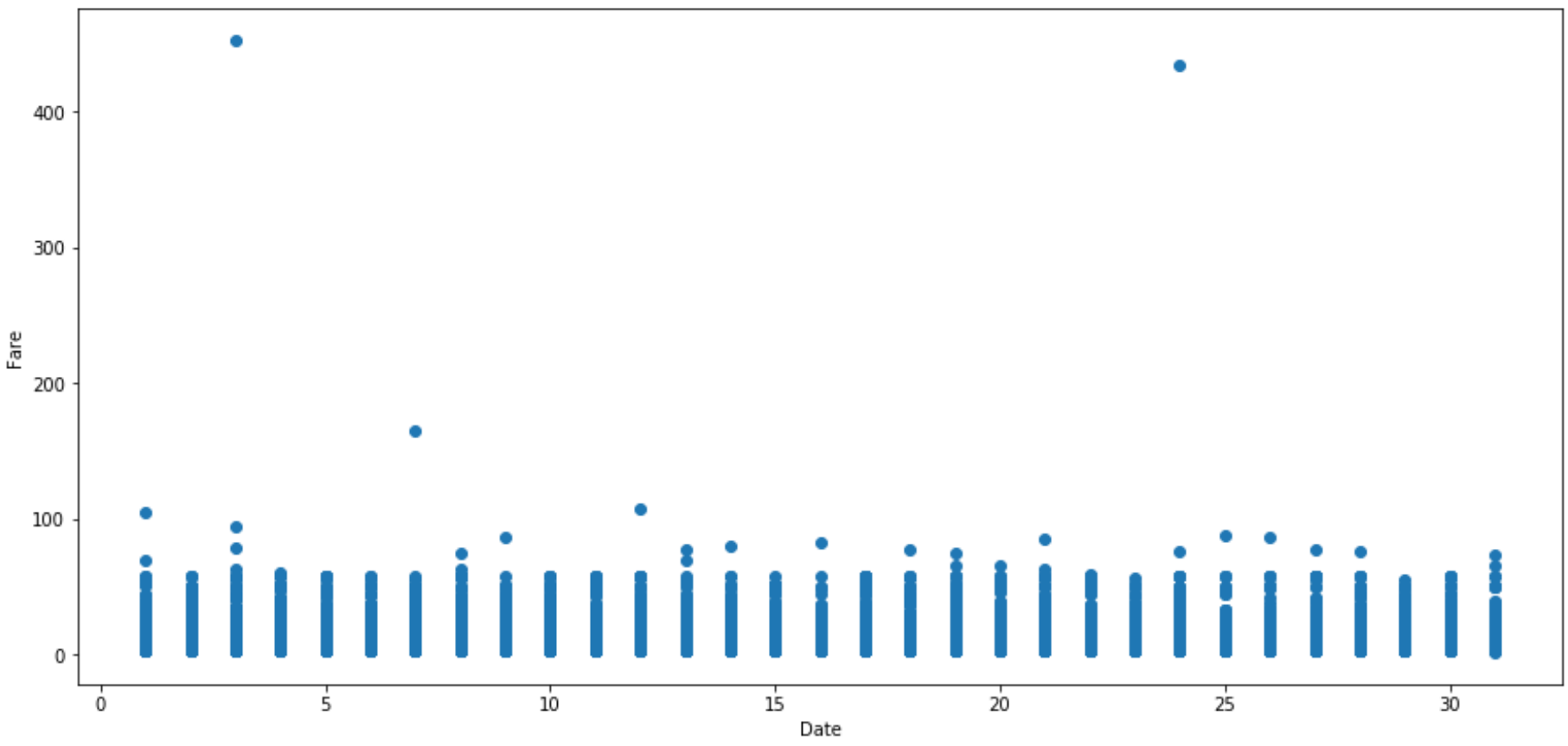
Fare amount and Number of passenger



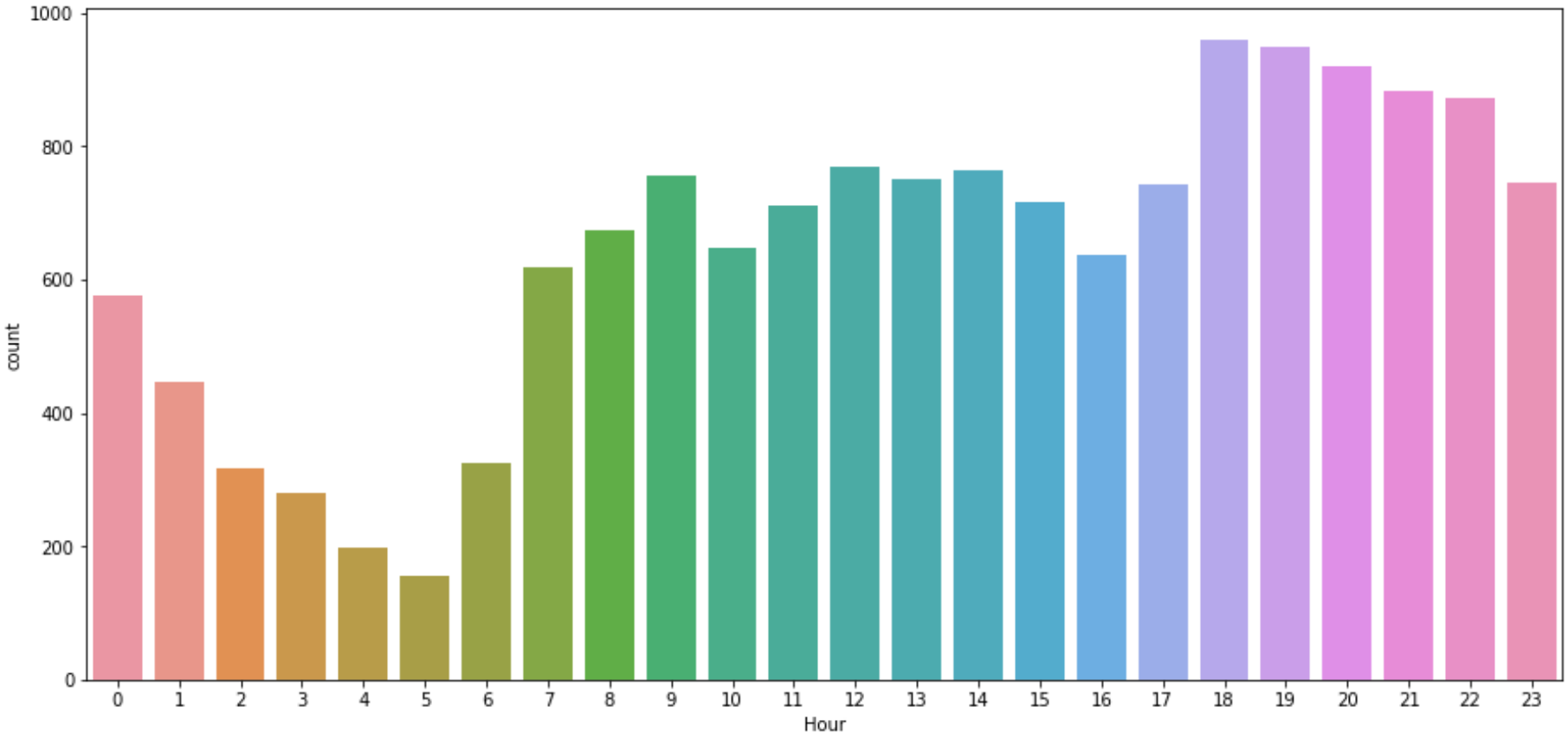
Here we can see that:

1. Single travelling passenger are more compared to others
2. Also we can say that highest fare amount is for single and double travelling passenger.

Date and Fare Amount



Hour Count



Here we can see that lowest cab ride are at 5 am and highhest cab ride at 6 pm due to office rush hours

4 Modeling

We will use Machine Learning Models on the processed data to predict the target variable. Following are the models which we have built:

- Linear Regression
- Decision Tree
- Random Forest

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

In [123]:
```
##train test split for further modelling
from sklearn.model_selection import train_test_split #splitting dataset
X_train, X_test, y_train, y_test = train_test_split( X,Y, test_size = 0.20, random_state = 1)
```

In [124]:
```
print(X_train.shape)
print(X_test.shape)
```

```
(12339, 7)
(3085, 7)
```

4.1 Linear Regression:

Multiple linear regression is the most common form of linear regression analysis. Multiple regression is an 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 is called 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:

In [129]:
```
from sklearn.linear_model import LinearRegression #ML algorithm
```

In [130]:
```
# Building model on top of training datasetfrom sklearn.linear_model import LinearRegression #ML algorithm
model = LinearRegression().fit(X_train , y_train)
```

In [131]:
```
#prediction on train data
pred_train_LR = model.predict(X_train)
```

In [132]:
```
#prediction on test data
pred_test_LR = model.predict(X_test)
```

In [133]:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
```

In [134]:
```
##calculating RMSE for test datafrom sklearn.metrics import mean_squared_error
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))

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

In [135]:
```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_LR))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_LR))
```

```
Root Mean Squared Error For Training data = 0.2758647452243595
Root Mean Squared Error For Test data = 0.24286207440988825
```

In [136]:
```
#calculate R^2 for train data
r2_score(y_train, pred_train_LR)
```

Out[136]: 0.7480566439279059

In [137]:
```
r2_score(y_test, pred_test_LR)
```

Out[137]: 0.7887921555089896

4.2 Decision Tree:

Decision Tree

In [138]:
```
from sklearn.tree import DecisionTreeRegressor
```

In [139]:
```
fit_DT  = DecisionTreeRegressor(criterion='mse', random_state=100,max_depth=4, min_samples_leaf=1)
```

In [140]:
```
fit_DT.fit(X_train, y_train)
```

Out[140]:
```
DecisionTreeRegressor(ccp_alpha=0.0, criterion='mse', max_depth=4,
                      max_features=None, max_leaf_nodes=None,
                      min_impurity_decrease=0.0, min_impurity_split=None,
                      min_samples_leaf=1, min_samples_split=2,
                      min_weight_fraction_leaf=0.0, presort='deprecated',
                      random_state=100, splitter='best')
```

In [141]:
```
pred_train_DT = fit_DT.predict(X_train)
```

In [142]:
```
pred_test_DT = fit_DT.predict(X_test)
```

In [143]:
```
RMSE_train_DT = np.sqrt(mean_squared_error(y_train, pred_train_DT))
```

In [144]:
```
RMSE_test_DT = np.sqrt(mean_squared_error(y_test, pred_test_DT))
```

In [145]:
```
print("Root Mean Squared Error For Training data = "+str(RMSE_train_DT))
print("Root Mean Squared Error For Test data = "+str(RMSE_test_DT))
```

```
Root Mean Squared Error For Training data = 0.25966827577326895
Root Mean Squared Error For Test data = 0.24049968648893114
```

In [146]:
```
r2_score(y_train, pred_train_DT)
```

Out[146]: 0.7767721924333111

In [147]:
```
r2_score(y_test, pred_test_DT)
```

Out[147]: 0.7928811276096276

4.3 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 the individual trees. Random decision forests correct for decision trees' habit of over fitting to their training set. To say it in simple words: Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

Random Forest

```
In [148]: from sklearn.ensemble import RandomForestRegressor

In [149]: model_RF = RandomForestRegressor(n_estimators= 500).fit(X_train, y_train)

E:\Ana3\lib\site-packages\ipykernel_launcher.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples,), for example using ravel().
  """Entry point for launching an IPython kernel.

In [150]: pred_train_RF = model_RF.predict(X_train)

In [151]: pred_test_RF = model_RF.predict(X_test)

In [152]: RMSE_train_RF  = np.sqrt(mean_squared_error(y_train,pred_train_RF))

In [153]: RMSE_test_RF  = np.sqrt(mean_squared_error(y_test,pred_test_RF))

In [154]: print("RMSE = ",RMSE_train_RF)
          print("RMSE = ", RMSE_test_RF)

RMSE =  0.09495650488547883
RMSE =  0.23004428373706645

In [155]: r2_score(y_train,pred_train_RF)
Out[155]: 0.9701489269425705

In [156]: r2_score(y_test,pred_test_RF)
Out[156]: 0.8104981141817381
```

4.4 Hyper Parameters Tunings:

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.

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

In [160]:
```
from sklearn.model_selection import GridSearchCV
## Grid Search CV for random Forest model
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,1))
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_RF = GridSearchCV(regr, param_grid = grid_search, cv = 5)
gridcv_RF = gridcv_RF.fit(X_train,y_train)
view_best_params_GRF = gridcv_RF.best_params_

#Apply model on test data
predictions_GRF = gridcv_RF.predict(X_test)

#R^2
GRF_r2 = r2_score(y_test, predictions_GRF)
#Calculating RMSE
GRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_GRF))

print('Grid Search CV Random Forest Regressor Model Performance:')
print('Best Parameters = ',view_best_params_GRF)
print('R-squared = {:0.2}.'.format(GRF_r2))
print('RMSE = ',(GRF_rmse))
```

```
Random Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'n_estimators': 15, 'max_depth': 9}
R-squared = 0.82.
RMSE =  0.22667922103841728
```

In [158]:
```
##Random Search CV on Random Forest Model

from sklearn.model_selection import RandomizedSearchCV
RRF = RandomForestRegressor(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_RF = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter = 5, cv = 5, random_state=0)
Randomcv_RF = Randomcv_RF.fit(X_train,y_train)
predictions_RRF = Randomcv_RF.predict(X_test)

view_best_params_RRF = Randomcv_RF.best_params_

#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)
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 19}
R-squared = 0.82.
RMSE =  0.2266642115231103
```

5 Conclusion

Model Evaluation/Tuning:

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 modeled.

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

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 coefficient of multiple determinations for multiple re gression.
In other words, we can say it explains as to how much of the variance of the target variable is explained.

Below table shows the model results before applying hyper tuning:

Model Name	RMSE		R-Squared	
	Train	Test	Train	Test
Linear Regression	0.27	0.24	0.74	0.78
Decision Tree	0.25	0.24	0.77	0.79
Random Forest	0.09	0.22	0.96	0.81

Below table shows results post using hyper parameter tuning techniques:

Parameter	Model Name	RMSE(test)	R-Squared(test)
Random Search CV	Random Forest	0.22	0.82
Grid Search CV		0.22	0.82

5.2 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 RMSE Value and R-Squared Value, both the Hyper Parameter perform equal on Random Forest.
- So I finally I have choosen Grid Search CV for predicting are test data.

End