Project Report

On

Cab fare Prediction

INDEX

Chapter 1

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

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: - 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

Chapter 2

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 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:

- Data exploration and Cleaning
- Missing values treament
- Outlier Analysis
- Feature Selection
- Features Scaling
    - 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 we will 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 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.

Chapter 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 remove 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 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 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.

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

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_longitude
- pickup_latitude
- dropoff_longitude
- dropoff_latitude
- Minute

3.4 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 set of 16067 observations with 8 variables including one dependent variable.

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

Our Dependent variable is: **fare_amount**

3.4.2 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. We have also added the table below:

3.4.3 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'

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

Chapter 4

Modelling

After a thorough preprocessing, we will use some regression models on our processed data to predict the target 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. 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.

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).

4.2 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 to visually 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 of each model in a combined table later on.

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 overfitting to their training set.

Random forest builds multiple decision trees and merges them together to get a more accurate and stable prediction.

4.4 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.

4.5 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.

Chapter 5

Conclusion

5.1 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.

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

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 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 both the 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	0.77
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	0.24	0.79
	Gradient Boosting	0.25	0.77
Grid Search CV	Random Forest	0.23	0.80
	Gradient Boosting	0.24	0.79

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.

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 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 this project with highest explained variance of the target variables and lowest error chances with parameter tuning technique Grid Search CV.

Finally, I used this method to predict the target variable for the test data file shared in the problem statement. Results that I found are attached with my submissions.

5.3 Some more visualization facts:

1. **<u>Number of passengers and fare</u>**

   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.

2. **<u>Date of month and fares</u>**

   The fares throughout the month mostly seem uniform.

3. **<u>Hours and Fares</u>**
   - 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 high demands.

4. **<u>Week Day and fare</u>**
   - Cab fare is high on Friday, Saturday and Monday, may be during weekend and first day of the working day they charge high fares because of high demands of cabs.

5. **<u>Impact of Day on the Number of Cab rides :</u>**

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

NOTE:

Since the missing values are less, I have skipped the entire column in python but in R (for the sake of project) I have imputed the missing values.

R CODE

R code:

```
# Cab Fare Prediction

rm(list = ls())
setwd("C:/Users/Welcome/Desktop/Cab prediction-Projects")
# #loading Libraries
x = c("ggplot2", "corrgram", "DMwR", "usdm", "caret", "randomForest", "e1071",
    "DataCombine", "doSNOW", "inTrees", "rpart.plot", "rpart",'MASS','xgboost','stats')
#load Packages
lapply(x, require, character.only = TRUE)
rm(x)

# The details of data attributes in the dataset are as follows:
# pickup_datetime - timestamp value indicating when the cab ride started.
# pickup_longitude - float for longitude coordinate of where the cab ride started.
# pickup_latitude - float for latitude coordinate of where the cab ride started.
# dropoff_longitude - float for longitude coordinate of where the cab ride ended.
# dropoff_latitude - float for latitude coordinate of where the cab ride ended.
# passenger_count - an integer indicating the number of passengers in the cab ride.


# loading datasets
train = read.csv("train_cab.csv", header = T, na.strings = c(" ", "", "NA"))
test = read.csv("test.csv")
test_pickup_datetime = test["pickup_datetime"]
# Structure of data
str(train)
str(test)
summary(train)
summary(test)
head(train,5)
head(test,5)

#############                Exploratory Data Analysis          #######################
# Changing the data types of variables
train$fare_amount = as.numeric(as.character(train$fare_amount))
train$passenger_count=round(train$passenger_count)
```

```
### Removing values which are not within desired range(outlier) depending upon basic understanding of dataset.

# 1.Fare amount has a negative value, which doesn't make sense. A price amount cannot be -ve and also cannot be 0. So we will remove these fields.
train[which(train$fare_amount < 1 ),]
nrow(train[which(train$fare_amount < 1 ),])
train = train[-which(train$fare_amount < 1 ),]

#2.Passenger_count variable
for (i in seq(4,11,by=1)){
  print(paste('passenger_count above ' ,i,nrow(train[which(train$passenger_count > i ),])))
  }
# so 20 observations of passenger_count is consistenly above from 6,7,8,9,10 passenger_counts, let's check them.
train[which(train$passenger_count > 6 ),]
# Also we need to see if there are any passenger_count==0
train[which(train$passenger_count <1 ),]
nrow(train[which(train$passenger_count <1 ),])
# We will remove these 58 observations and 20 observation which are above 6 value because a cab cannot hold these number of passengers.
train = train[-which(train$passenger_count < 1 ),]
train = train[-which(train$passenger_count > 6),]
# 3.Latitudes range from -90 to 90.Longitudes range from -180 to 180.Removing which does not satisfy these ranges
print(paste('pickup_longitude above 180=',nrow(train[which(train$pickup_longitude >180 ),])))
print(paste('pickup_longitude above -180=',nrow(train[which(train$pickup_longitude < -180 ),])))
print(paste('pickup_latitude above 90=',nrow(train[which(train$pickup_latitude > 90 ),])))
print(paste('pickup_latitude above -90=',nrow(train[which(train$pickup_latitude < -90 ),])))
print(paste('dropoff_longitude above 180=',nrow(train[which(train$dropoff_longitude > 180 ),])))
print(paste('dropoff_longitude above -180=',nrow(train[which(train$dropoff_longitude < -180 ),])))
print(paste('dropoff_latitude above -90=',nrow(train[which(train$dropoff_latitude < -90 ),])))
print(paste('dropoff_latitude above 90=',nrow(train[which(train$dropoff_latitude > 90 ),])))
# There's only one outlier which is in variable pickup_latitude.So we will remove it with nan.
# Also we will see if there are any values equal to 0.
nrow(train[which(train$pickup_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
```

```
nrow(train[which(train$dropoff_longitude == 0 ),])
nrow(train[which(train$pickup_latitude == 0 ),])
# there are values which are equal to 0. we will remove them.
train = train[-which(train$pickup_latitude > 90),]
train = train[-which(train$pickup_longitude == 0),]
train = train[-which(train$dropoff_longitude == 0),]

# Make a copy
df=train
# train=df

#############                    Missing Value Analysis                #############
missing_val = data.frame(apply(train,2,function(x){sum(is.na(x))}))
missing_val$Columns = row.names(missing_val)
names(missing_val)[1] =  "Missing_percentage"
missing_val$Missing_percentage = (missing_val$Missing_percentage/nrow(train)) * 100
missing_val = missing_val[order(-missing_val$Missing_percentage),]
row.names(missing_val) = NULL
missing_val = missing_val[,c(2,1)]
missing_val

unique(train$passenger_count)
unique(test$passenger_count)
train[,'passenger_count'] = factor(train[,'passenger_count'], labels=(1:6))
test[,'passenger_count'] = factor(test[,'passenger_count'], labels=(1:6))
# 1.For Passenger_count:
# Actual value = 1
# Mode = 1
# KNN = 1
train$passenger_count[1000]
train$passenger_count[1000] = NA
getmode <- function(v) {
  uniqv <- unique(v)
  uniqv[which.max(tabulate(match(v, uniqv)))]
}

# Mode Method
```

```
getmode(train$passenger_count)
# We can't use mode method because data will be more biased towards passenger_count=1

# 2.For fare_amount:
# Actual value = 18.1,
# Mean = 15.117,
# Median = 8.5,
# KNN = 18.28
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.968236       4635.700531          2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305          2.710835          2.632400
# passenger_count
# 1.266104
train$fare_amount[1000]
train$fare_amount[1000]= NA

# Mean Method
mean(train$fare_amount, na.rm = T)

#Median Method
median(train$fare_amount, na.rm = T)

# kNN Imputation
train = knnImputation(train, k = 181)
train$fare_amount[1000]
train$passenger_count[1000]
sapply(train, sd, na.rm = TRUE)
# fare_amount   pickup_datetime  pickup_longitude
# 435.661952       4635.700531          2.659050
# pickup_latitude dropoff_longitude  dropoff_latitude
# 2.613305          2.710835          2.632400
# passenger_count
# 1.263859
sum(is.na(train))
str(train)
```

```
summary(train)

df1=train
# train=df1
#####################                    Outlier Analysis               ##################

# We Will do Outlier Analysis only on Fare_amount just for now and we will do outlier analysis after feature engineering laitudes and longitudes.
# Boxplot for fare_amount
pl1 = ggplot(train,aes(x = factor(passenger_count),y = fare_amount))
pl1 + geom_boxplot(outlier.colour="red", fill = "grey" ,outlier.shape=18,outlier.size=1, notch=FALSE)+ylim(0,100)

# Replace all outliers with NA and impute
vals = train[,"fare_amount"] %in% boxplot.stats(train[,"fare_amount"])$out
train[which(vals),"fare_amount"] = NA

#lets check the NA's
sum(is.na(train$fare_amount))

#Imputing with KNN
train = knnImputation(train,k=3)

# lets check the missing values
sum(is.na(train$fare_amount))
str(train)

df2=train
# train=df2
##################                 Feature Engineering                    ##########################
# 1.Feature Engineering for timestamp variable
# we will derive new features from pickup_datetime variable
# new features will be year,month,day_of_week,hour
#Convert pickup_datetime from factor to date time
train$pickup_date = as.Date(as.character(train$pickup_datetime))
train$pickup_weekday = as.factor(format(train$pickup_date,"%u"))# Monday = 1
train$pickup_mnth = as.factor(format(train$pickup_date,"%m"))
train$pickup_yr = as.factor(format(train$pickup_date,"%Y"))
```

```
pickup_time = strptime(train$pickup_datetime,"%Y-%m-%d %H:%M:%S")
train$pickup_hour = as.factor(format(pickup_time,"%H"))

#Add same features to test set
test$pickup_date = as.Date(as.character(test$pickup_datetime))
test$pickup_weekday = as.factor(format(test$pickup_date,"%u"))# Monday = 1
test$pickup_mnth = as.factor(format(test$pickup_date,"%m"))
test$pickup_yr = as.factor(format(test$pickup_date,"%Y"))
pickup_time = strptime(test$pickup_datetime,"%Y-%m-%d %H:%M:%S")
test$pickup_hour = as.factor(format(pickup_time,"%H"))

sum(is.na(train))# there was 1 'na' in pickup_datetime which created na's in above feature engineered variables.
train = na.omit(train) # we will remove that 1 row of na's

train = subset(train,select = -c(pickup_datetime,pickup_date))
test = subset(test,select = -c(pickup_datetime,pickup_date))
# Now we will use month,weekday,hour to derive new features like sessions in a day,seasons in a year,week:weekend/weekday
# f = function(x){
#   if ((x >=5)& (x <= 11)){
#     return ('morning')
#   }
#   if ((x >=12) & (x <= 16)){
#     return ('afternoon')
#   }
#   if ((x >=17) & (x <= 20)){
#     return ('evening')
#   }
#   if ((x >=21) & (x <= 23)){
#     return ('night (PM)')
#   }
#   if ((x >=0) & (x <= 4)){
#     return ('night (AM)')
#   }
# }
# 2.Calculate the distance travelled using longitude and latitude
deg_to_rad = function(deg){
```

```
  (deg * pi) / 180
}
haversine = function(long1,lat1,long2,lat2){
  #long1rad = deg_to_rad(long1)
  phi1 = deg_to_rad(lat1)
  #long2rad = deg_to_rad(long2)
  phi2 = deg_to_rad(lat2)
  delphi = deg_to_rad(lat2 - lat1)
  dellamda = deg_to_rad(long2 - long1)

  a = sin(delphi/2) * sin(delphi/2) + cos(phi1) * cos(phi2) *
    sin(dellamda/2) * sin(dellamda/2)

  c = 2 * atan2(sqrt(a),sqrt(1-a))
  R = 6371e3
  R * c / 1000 #1000 is used to convert to meters
}
# Using haversine formula to calculate distance fr both train and test
train$dist =
haversine(train$pickup_longitude,train$pickup_latitude,train$dropoff_longitude,train$dropoff_latitude)
test$dist = haversine(test$pickup_longitude,test$pickup_latitude,test$dropoff_longitude,test$dropoff_latitude)

# We will remove the variables which were used to feature engineer new variables
train = subset(train,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))
test = subset(test,select = -c(pickup_longitude,pickup_latitude,dropoff_longitude,dropoff_latitude))

str(train)
summary(train)

################                    Feature selection           ###################
numeric_index = sapply(train,is.numeric) #selecting only numeric

numeric_data = train[,numeric_index]

cnames = colnames(numeric_data)
#Correlation analysis for numeric variables
corrgram(train[,numeric_index],upper.panel=panel.pie, main = "Correlation Plot")
```

```
#ANOVA for categorical variables with target numeric variable

#aov_results = aov(fare_amount ~ passenger_count * pickup_hour * pickup_weekday,data = train)
aov_results = aov(fare_amount ~ passenger_count + pickup_hour + pickup_weekday + pickup_mnth +
pickup_yr,data = train)

summary(aov_results)

# pickup_weekdat has p value greater than 0.05
train = subset(train,select=-pickup_weekday)

#remove from test set
test = subset(test,select=-pickup_weekday)

##################################             Feature Scaling
#############################################
#Normality check
# qqnorm(train$fare_amount)
# histogram(train$fare_amount)
library(car)
# dev.off()
par(mfrow=c(1,2))
qqPlot(train$fare_amount)                             # qqPlot, it has a x values derived from gaussian distribution, if data
is distributed normally then the sorted data points should lie very close to the solid reference line
truehist(train$fare_amount)                           # truehist() scales the counts to give an estimate of the probability
density.
lines(density(train$fare_amount))  # Right skewed      # lines() and density() functions to overlay a density plot
on histogram

#Normalisation

print('dist')
train[,'dist'] = (train[,'dist'] - min(train[,'dist']))/
   (max(train[,'dist'] - min(train[,'dist'])))

# #check multicollearity
# library(usdm)
```

```
# vif(train[,-1])
#
# vifcor(train[,-1], th = 0.9)

#################### Splitting train into train and validation subsets ###################
set.seed(1000)
tr.idx = createDataPartition(train$fare_amount,p=0.75,list = FALSE) # 75% in trainin and 25% in Validation Datasets
train_data = train[tr.idx,]
test_data = train[-tr.idx,]

rmExcept(c("test","train","df",'df1','df2','df3','test_data','train_data','test_pickup_datetime'))
###################Model Selection################
#Error metric used to select model is RMSE

#############            Linear regression               #################
lm_model = lm(fare_amount ~.,data=train_data)

summary(lm_model)
str(train_data)
plot(lm_model$fitted.values,rstandard(lm_model),main = "Residual plot",
     xlab = "Predicted values of fare_amount",
     ylab = "standardized residuals")



lm_predictions = predict(lm_model,test_data[,2:6])

qplot(x = test_data[,1], y = lm_predictions, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],lm_predictions)
# mae        mse       rmse       mape
# 3.5303114 19.3079726  4.3940838  0.4510407



#############                         Decision Tree           #####################

Dt_model = rpart(fare_amount ~ ., data = train_data, method = "anova")
```

```
summary(Dt_model)
#Predict for new test cases
predictions_DT = predict(Dt_model, test_data[,2:6])

qplot(x = test_data[,1], y = predictions_DT, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],predictions_DT)
# mae       mse       rmse      mape
# 1.8981592 6.7034713 2.5891063 0.2241461



#############                    Random forest          #####################
rf_model = randomForest(fare_amount ~.,data=train_data)

summary(rf_model)

rf_predictions = predict(rf_model,test_data[,2:6])

qplot(x = test_data[,1], y = rf_predictions, data = test_data, color = I("blue"), geom = "point")

regr.eval(test_data[,1],rf_predictions)
# mae       mse       rmse      mape
# 1.9053850 6.3682283 2.5235349 0.2335395

############         Improving Accuracy by using Ensemble technique ---- XGBOOST
###########################
train_data_matrix = as.matrix(sapply(train_data[-1],as.numeric))
test_data_data_matrix = as.matrix(sapply(test_data[-1],as.numeric))

xgboost_model = xgboost(data = train_data_matrix,label = train_data$fare_amount,nrounds = 15,verbose =
FALSE)

summary(xgboost_model)
xgb_predictions = predict(xgboost_model,test_data_data_matrix)

qplot(x = test_data[,1], y = xgb_predictions, data = test_data, color = I("blue"), geom = "point")
```

```
regr.eval(test_data[,1],xgb_predictions)
# mae       mse      rmse      mape
# 1.6183415 5.1096465 2.2604527 0.1861947

#############                    Finalizing and Saving Model for later use
####################
# In this step we will train our model on whole training Dataset and save that model for later use
train_data_matrix2 = as.matrix(sapply(train[-1],as.numeric))
test_data_matrix2 = as.matrix(sapply(test,as.numeric))

xgboost_model2 = xgboost(data = train_data_matrix2,label = train$fare_amount,nrounds = 15,verbose =
FALSE)

# Saving the trained model
saveRDS(xgboost_model2, "./final_Xgboost_model_using_R.rds")

# loading the saved model
super_model <- readRDS("./final_Xgboost_model_using_R.rds")
print(super_model)

# Lets now predict on test dataset
xgb = predict(super_model,test_data_matrix2)

xgb_pred = data.frame(test_pickup_datetime,"predictions" = xgb)

# Now lets write(save) the predicted fare_amount in disk as .csv format
write.csv(xgb_pred,"xgb_predictions_R.csv",row.names = FALSE)
```

PYTHON CODE

In [1]:
```
import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import GradientBoostingRegressor
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score
from pprint import pprint
from sklearn.model_selection import GridSearchCV
%matplotlib inline
```

In [2]:
```
os.chdir("C:/Users/Welcome/Desktop/Cab prediction-Project")
```

In [3]:
```
train  = pd.read_csv("train_cab.csv",na_values={"pickup_datetime":"43"})
test   = pd.read_csv("test.csv")
```

In [4]:
```
train.head()
```

Out[4]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	40.7
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	40.7
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	40.7
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	40.7
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	40.7

In [5]:
```
test.head()
```

Out[5]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
0	2015-01-27 13:08:24 UTC	-73.973320	40.763805	-73.981430	40.743835	
1	2015-01-27 13:08:24 UTC	-73.986862	40.719383	-73.998886	40.739201	
2	2011-10-08 11:53:44 UTC	-73.982524	40.751260	-73.979654	40.746139	
3	2012-12-01 21:12:12 UTC	-73.981160	40.767807	-73.990448	40.751635	
4	2012-12-01 21:12:12 UTC	-73.966046	40.789775	-73.988565	40.744427	

In [6]:
```
print("shape of training data is: ",train.shape)
```

```
shape of training data is:  (16067, 7)
```

In [7]:
```
print("shape of test data is: ",test.shape)
```

```
shape of test data is:  (9914, 6)
```

In [8]:
```
train.dtypes
```

Out[8]:
```
fare_amount           object
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count      float64
dtype: object
```

In [9]:
```
test.dtypes
```

Out[9]:
```
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

In [10]:
```
test.describe()
```

Out[10]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	9914.000000	9914.000000	9914.000000	9914.000000	9914.000000
mean	-73.974722	40.751041	-73.973657	40.751743	1.671273
std	0.042774	0.033541	0.039072	0.035435	1.278747
min	-74.252193	40.573143	-74.263242	40.568973	1.000000
25%	-73.992501	40.736125	-73.991247	40.735254	1.000000
50%	-73.982326	40.753051	-73.980015	40.754065	1.000000
75%	-73.968013	40.767113	-73.964059	40.768757	2.000000
max	-72.986532	41.709555	-72.990963	41.696683	6.000000

In [11]:
```
train.describe()
```

Out[11]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	16067.000000	16067.000000	16067.000000	16067.000000	16012.000000
mean	-72.462787	39.914725	-72.462328	39.897906	2.625070
std	10.578384	6.826587	10.575062	6.187087	60.844122
min	-74.438233	-74.006893	-74.429332	-74.006377	0.000000
25%	-73.992156	40.734927	-73.991182	40.734651	1.000000
50%	-73.981698	40.752603	-73.980172	40.753567	1.000000
75%	-73.966838	40.767381	-73.963643	40.768013	2.000000
max	40.766125	401.083332	40.802437	41.366138	5345.000000

In [12]:
```
train["fare_amount"] = pd.to_numeric(train["fare_amount"],errors = "coerce")
```

In [13]:
```
train.dtypes
```

Out[13]:
```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count      float64
dtype: object
```

In [14]:
```
train.shape
```

Out[14]:
```
(16067, 7)
```

In [15]:
```
train.dropna(subset= ["pickup_datetime"])
```

Out[15]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropc
0	4.5	2009-06-15 17:26:21 UTC	-73.844311	40.721319	-73.841610	
1	16.9	2010-01-05 16:52:16 UTC	-74.016048	40.711303	-73.979268	
2	5.7	2011-08-18 00:35:00 UTC	-73.982738	40.761270	-73.991242	
3	7.7	2012-04-21 04:30:42 UTC	-73.987130	40.733143	-73.991567	
4	5.3	2010-03-09 07:51:00 UTC	-73.968095	40.768008	-73.956655	
5	12.1	2011-01-06 09:50:45 UTC	-74.000964	40.731630	-73.972892	
6	7.5	2012-11-20 20:35:00 UTC	-73.980002	40.751662	-73.973802	
7	16.5	2012-01-04 17:22:00 UTC	-73.951300	40.774138	-73.990095	
8	NaN	2012-12-03 13:10:00 UTC	-74.006462	40.726713	-73.993078	
9	8.9	2009-09-02 01:11:00 UTC	-73.980658	40.733873	-73.991540	
10	5.3	2012-04-08 07:30:50 UTC	-73.996335	40.737142	-73.980721	
11	5.5	2012-12-24 11:24:00 UTC	0.000000	0.000000	0.000000	
12	4.1	2009-11-06 01:04:03 UTC	-73.991601	40.744712	-73.983081	
13	7.0	2013-07-02 19:54:00 UTC	-74.005360	40.728867	-74.008913	
14	7.7	2011-04-05 17:11:05 UTC	-74.001821	40.737547	-73.998060	
15	5.0	2013-11-23 12:57:00 UTC	0.000000	0.000000	0.000000	
16	12.5	2014-02-19 07:22:00 UTC	-73.986430	40.760465	-73.988990	
17	5.3	2009-07-22 16:08:00 UTC	-73.981060	40.737690	-73.994177	
18	5.3	2010-07-07 14:52:00 UTC	-73.969505	40.784843	-73.958732	
19	4.0	2014-12-06 20:36:22 UTC	-73.979815	40.751902	-73.979446	
20	10.5	2010-09-07 13:18:00 UTC	-73.985382	40.747858	-73.978377	
21	11.5	2013-02-12 12:15:46 UTC	-73.957954	40.779252	-73.961250	
22	4.5	2009-08-06 18:17:23 UTC	-73.991707	40.770505	-73.985459	

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
23	4.9	2010-12-06 12:29:00 UTC	-74.000632	40.747473	-73.986672	
24	6.1	2009-12-10 15:37:00 UTC	-73.969622	40.756973	-73.981152	
25	7.3	2011-06-21 16:15:00 UTC	-73.991875	40.754437	-73.977230	
26	NaN	2011-02-07 20:01:00 UTC	0.000000	0.000000	0.000000	
27	4.5	2011-06-28 19:47:00 UTC	-73.988893	40.760160	-73.986445	
28	9.3	2012-05-04 06:11:20 UTC	-73.989258	40.690835	-74.004133	
29	4.5	2013-08-11 00:52:00 UTC	-73.981020	40.737760	-73.980668	
...	...	...	...	...	...	
16037	6.5	2012-02-27 21:40:50 UTC	-73.992618	40.723878	-73.977073	
16038	5.7	2010-08-31 10:43:42 UTC	-73.990336	40.718973	-73.956060	
16039	12.9	2010-12-11 16:25:00 UTC	-73.936462	40.794292	-73.948747	
16040	6.5	2014-06-16 00:05:19 UTC	-73.980597	40.744267	-73.979330	
16041	11.0	2014-11-17 21:53:00 UTC	-73.983610	40.747090	-73.961310	
16042	8.5	2015-04-06 21:53:06 UTC	-73.991425	40.749832	-74.000107	
16043	8.5	2011-11-17 10:58:05 UTC	-73.973961	40.764055	-73.986807	
16044	16.5	2013-04-29 03:05:45 UTC	-73.982785	40.731421	-74.011358	
16045	6.5	2013-09-19 23:56:00 UTC	-73.995227	40.733475	-73.984030	
16046	6.0	2014-04-24 01:48:40 UTC	-73.976298	40.753948	-73.993062	
16047	6.1	2010-03-18 11:09:00 UTC	-73.970733	40.758193	-73.979457	
16048	9.7	2012-07-10 17:32:00 UTC	-73.988040	40.774902	-74.005265	
16049	15.7	2012-07-31 12:27:00 UTC	-74.008657	40.715975	-73.975653	
16050	8.5	2013-01-23 07:36:49 UTC	-73.996715	40.742504	-73.977987	
16051	11.5	2014-10-01 20:05:00 UTC	-73.975540	40.755590	-73.944780	
16052	10.0	2014-10-03 22:24:00 UTC	-73.987298	40.722007	-74.000267	

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropc
16053	4.0	2014-09-23 09:49:00 UTC	-73.954977	40.788582	-73.964227	
16054	5.3	2009-11-28 15:58:02 UTC	-73.993929	40.756944	-73.993044	
16055	48.3	2012-09-05 17:34:00 UTC	-73.994077	40.741242	-73.830257	
16056	38.3	2012-12-17 14:59:16 UTC	0.000000	0.000000	0.000000	
16057	5.0	2013-01-31 15:46:00 UTC	-73.963582	40.774242	-73.956525	
16058	5.5	2014-04-19 14:58:57 UTC	-73.974265	40.756048	-73.980885	
16059	5.3	2010-01-03 18:26:00 UTC	-73.973297	40.743768	-73.986060	
16060	22.0	2014-10-01 09:15:00 UTC	-73.954582	40.778047	-74.005982	
16061	10.9	2009-05-20 18:56:42 UTC	-73.994191	40.751138	-73.962769	
16062	6.5	2014-12-12 07:41:00 UTC	-74.008820	40.718757	-73.998865	
16063	16.1	2009-07-13 07:58:00 UTC	-73.981310	40.781695	-74.014392	
16064	8.5	2009-11-11 11:19:07 UTC	-73.972507	40.753417	-73.979577	
16065	8.1	2010-05-11 23:53:00 UTC	-73.957027	40.765945	-73.981983	
16066	8.5	2011-12-14 06:24:33 UTC	-74.002111	40.729755	-73.983877	

16066 rows × 7 columns

In [16]:
```
train['pickup_datetime'] =  pd.to_datetime(train['pickup_datetime'], format='%
Y-%m-%d %H:%M:%S UTC')
```

In [17]:
```
train['year'] = train['pickup_datetime'].dt.year
train['Month'] = train['pickup_datetime'].dt.month
train['Date'] = train['pickup_datetime'].dt.day
train['Day'] = train['pickup_datetime'].dt.dayofweek
train['Hour'] = train['pickup_datetime'].dt.hour
train['Minute'] = train['pickup_datetime'].dt.minute
```

In [18]:
```
train.dtypes
```

Out[18]:
```
fare_amount                 float64
pickup_datetime      datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
dropoff_latitude            float64
passenger_count             float64
year                        float64
Month                       float64
Date                        float64
Day                         float64
Hour                        float64
Minute                      float64
dtype: object
```

In [19]:
```
test["pickup_datetime"] = pd.to_datetime(test["pickup_datetime"],format= "%Y-%m-%d %H:%M:%S UTC")
```

In [20]:
```
test['year'] = test['pickup_datetime'].dt.year
test['Month'] = test['pickup_datetime'].dt.month
test['Date'] = test['pickup_datetime'].dt.day
test['Day'] = test['pickup_datetime'].dt.dayofweek
test['Hour'] = test['pickup_datetime'].dt.hour
test['Minute'] = test['pickup_datetime'].dt.minute
```

In [21]:
```
test.dtypes
```

Out[21]:
```
pickup_datetime      datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
dropoff_latitude            float64
passenger_count               int64
year                          int64
Month                         int64
Date                          int64
Day                           int64
Hour                          int64
Minute                        int64
dtype: object
```

In [22]:
```
train = train.drop(train[train['pickup_datetime'].isnull()].index, axis=0)
print(train.shape)
print(train['pickup_datetime'].isnull().sum())
```

```
(16066, 13)
0
```

In [23]:
```
train["passenger_count"].describe()
```

Out[23]:
```
count    16011.000000
mean         2.625171
std         60.846021
min          0.000000
25%          1.000000
50%          1.000000
75%          2.000000
max       5345.000000
Name: passenger_count, dtype: float64
```

In [24]:
```
train = train.drop(train[train["passenger_count"]> 6 ].index, axis=0)
```

In [25]:
```
train = train.drop(train[train["passenger_count"] == 0 ].index, axis=0)
```

In [26]:
```
train["passenger_count"].describe()
```

Out[26]:
```
count    15934.000000
mean         1.649581
std          1.265943
min          0.120000
25%          1.000000
50%          1.000000
75%          2.000000
max          6.000000
Name: passenger_count, dtype: float64
```

In [27]:
```
train["passenger_count"].sort_values(ascending= True)
```

Out[27]:
```
8862      0.12
0         1.00
9790      1.00
9791      1.00
9792      1.00
9793      1.00
9794      1.00
9795      1.00
9796      1.00
9797      1.00
9798      1.00
9801      1.00
9804      1.00
9806      1.00
9807      1.00
9808      1.00
9809      1.00
9811      1.00
9812      1.00
9814      1.00
9818      1.00
9819      1.00
9789      1.00
9788      1.00
9785      1.00
9784      1.00
9754      1.00
9756      1.00
9757      1.00
9758      1.00
          ...
734        NaN
773        NaN
788        NaN
842        NaN
899        NaN
941        NaN
1361       NaN
1399       NaN
1400       NaN
1459       NaN
1748       NaN
1790       NaN
1851       NaN
1921       NaN
1984       NaN
1987       NaN
2104       NaN
2230       NaN
2378       NaN
7787       NaN
7805       NaN
7847       NaN
7892       NaN
7937       NaN
8007       NaN
8076       NaN
```

```
8139      NaN
8259      NaN
8306      NaN
16066     NaN
Name: passenger_count, Length: 15989, dtype: float64
```

In [28]:
```
train = train.drop(train[train['passenger_count'].isnull()].index, axis=0)
print(train.shape)
print(train['passenger_count'].isnull().sum())
```

```
(15934, 13)
0
```

In [29]:
```
train = train.drop(train[train["passenger_count"] == 0.12 ].index, axis=0)
train.shape
```

Out[29]:
```
(15933, 13)
```

In [30]:
```
train["fare_amount"].sort_values(ascending=False)
```

Out[30]:
```
1015     54343.00
1072      4343.00
607        453.00
980        434.00
1335       180.00
1483       165.00
6630       128.83
14142      108.00
12349      104.67
12915       96.00
7810        95.00
9431        88.00
10077       87.30
12614       87.00
4620        85.50
14519       82.50
12437       80.75
2639        79.00
4013        77.70
13962       77.15
2013        77.00
6668        76.80
8363        76.00
10524       75.80
11019       75.33
13615       75.00
15023       73.30
1494        70.00
4118        69.70
9651        66.30
           ...
1427         1.14
2780         0.01
10002        0.00
2486        -2.50
2039        -2.90
13032       -3.00
8             NaN
26            NaN
69            NaN
126           NaN
168           NaN
240           NaN
305           NaN
350           NaN
455           NaN
498           NaN
667           NaN
703           NaN
746           NaN
836           NaN
840           NaN
913           NaN
1123          NaN
1574          NaN
1628          NaN
1712          NaN
```

```
2412          NaN
2458          NaN
8178          NaN
8226          NaN
Name: fare_amount, Length: 15933, dtype: float64
```

In [31]:
```
Counter({False: 15930, True: 3})
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-31-66465da172e0> in <module>
----> 1 Counter({False: 15930, True: 3})

NameError: name 'Counter' is not defined
```

In [32]:
```
train = train.drop(train[train["fare_amount"]<0].index, axis=0)
train.shape
```

Out[32]:
```
(15930, 13)
```

In [33]:
```
train["fare_amount"].min()
```

Out[33]:
```
0.0
```

In [34]:
```
train = train.drop(train[train["fare_amount"]<1].index, axis=0)
train.shape
```

Out[34]:
```
(15928, 13)
```

In [35]:
```
train = train.drop(train[train["fare_amount"]> 454 ].index, axis=0)
train.shape
```

Out[35]:
```
(15926, 13)
```

In [36]:
```
train = train.drop(train[train['fare_amount'].isnull()].index, axis=0)
print(train.shape)
print(train['fare_amount'].isnull().sum())
```

```
(15902, 13)
0
```

In [37]:
```
train["fare_amount"].describe()
```

Out[37]:
```
count    15902.000000
mean        11.376356
std         10.814908
min          1.140000
25%          6.000000
50%          8.500000
75%         12.500000
max        453.000000
Name: fare_amount, dtype: float64
```

In [38]:
```
train[train['pickup_latitude']<-90]
train[train['pickup_latitude']>90]
```

Out[38]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropof
5686	3.3	2011-07-30 11:15:00	-73.947235	401.083332	-73.951392	4

In [39]:
```
train = train.drop((train[train['pickup_latitude']<-90]).index, axis=0)
train = train.drop((train[train['pickup_latitude']>90]).index, axis=0)
```

In [40]:
```
train[train['pickup_longitude']<-180]
train[train['pickup_longitude']>180]
```

Out[40]:

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati

In [41]:
```
train[train['dropoff_latitude']<-90]
train[train['dropoff_latitude']>90]
```

Out[41]:

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati

In [42]:
```
train[train['dropoff_longitude']<-180]
train[train['dropoff_longitude']>180]
```

Out[42]:

fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_lati

In [43]:
```
train.shape
```

Out[43]:
```
(15901, 13)
```

In [44]:
```
train.isnull().sum()
```

Out[44]:
```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
year                 0
Month                0
Date                 0
Day                  0
Hour                 0
Minute               0
dtype: int64
```

In [45]:
```
test.isnull().sum()
```

Out[45]:
```
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
year                 0
Month                0
Date                 0
Day                  0
Hour                 0
Minute               0
dtype: int64
```

In [46]:
```
from math import radians, cos, sin, asin, sqrt
```

In [49]:
```
def haversine(lat1, lon1, lat2, lon2):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
```

In [50]:
```
lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-50-44b3407d432c> in <module>
----> 1 lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

NameError: name 'lon1' is not defined
```

In [51]:
```
def haversine(lon1, lat1, lon2, lat2):
```

```
  File "<ipython-input-51-1f5e528b4d95>", line 1
    def haversine(lon1, lat1, lon2, lat2):
                                          ^
SyntaxError: unexpected EOF while parsing
```

In [52]:
```
from math import radians, cos, sin, asin, sqrt
def haversine(lon1, lat1, lon2, lat2):
```

```
  File "<ipython-input-52-50c3efefd6ad>", line 2
    def haversine(lon1, lat1, lon2, lat2):
                                          ^
SyntaxError: unexpected EOF while parsing
```

In [53]:
```
import numpy as np
```

In [54]:
```
import pandas as pd
```

In [55]:
```
def haversine_np(lon1, lat1, lon2, lat2):
```

```
  File "<ipython-input-55-b65bd143c952>", line 1
    def haversine_np(lon1, lat1, lon2, lat2):
                                             ^
SyntaxError: unexpected EOF while parsing
```

In [56]:
```
from math import radians, cos, sin, asin, sqrt
import numpy as np
```

In [57]:
```
def haversine_np(lon1, lat1, lon2, lat2):
```

```
  File "<ipython-input-57-b65bd143c952>", line 1
    def haversine_np(lon1, lat1, lon2, lat2):
                                             ^
SyntaxError: unexpected EOF while parsing
```

In [59]:
```
def haversine_np(lon1, lat1, lon2, lat2):
     lon1, lat1, lon2, lat2 = map(np.radians, [lon1, lat1, lon2, lat2])
    dlon = lon2 - lon1
          dlat = lat2 - lat1
        a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
        c =  2 * asin(sqrt(a))
        km = 6371* c
        return km
```

```
  File "<tokenize>", line 3
    dlon = lon2 - lon1
    ^
IndentationError: unindent does not match any outer indentation level
```

In [60]:
```
from math import radians, cos, sin, asin, sqrt

def haversine(a):
    lon1=a[0]
    lat1=a[1]
    lon2=a[2]
    lat2=a[3]
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])

    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c =  2 * asin(sqrt(a))
    # Radius of earth in kilometers is 6371
    km = 6371* c
    return km
```

In [61]:
```
train
```

Out[61]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	
5	12.1	2011-01-06 09:50:45	-74.000964	40.731630	-73.972892	
6	7.5	2012-11-20 20:35:00	-73.980002	40.751662	-73.973802	
7	16.5	2012-01-04 17:22:00	-73.951300	40.774138	-73.990095	
9	8.9	2009-09-02 01:11:00	-73.980658	40.733873	-73.991540	
10	5.3	2012-04-08 07:30:50	-73.996335	40.737142	-73.980721	
11	5.5	2012-12-24 11:24:00	0.000000	0.000000	0.000000	
12	4.1	2009-11-06 01:04:03	-73.991601	40.744712	-73.983081	
13	7.0	2013-07-02 19:54:00	-74.005360	40.728867	-74.008913	
14	7.7	2011-04-05 17:11:05	-74.001821	40.737547	-73.998060	
15	5.0	2013-11-23 12:57:00	0.000000	0.000000	0.000000	
16	12.5	2014-02-19 07:22:00	-73.986430	40.760465	-73.988990	
17	5.3	2009-07-22 16:08:00	-73.981060	40.737690	-73.994177	
18	5.3	2010-07-07 14:52:00	-73.969505	40.784843	-73.958732	
19	4.0	2014-12-06 20:36:22	-73.979815	40.751902	-73.979446	
20	10.5	2010-09-07 13:18:00	-73.985382	40.747858	-73.978377	
21	11.5	2013-02-12 12:15:46	-73.957954	40.779252	-73.961250	
22	4.5	2009-08-06 18:17:23	-73.991707	40.770505	-73.985459	
23	4.9	2010-12-06 12:29:00	-74.000632	40.747473	-73.986672	

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
24	6.1	2009-12-10 15:37:00	-73.969622	40.756973	-73.981152	
25	7.3	2011-06-21 16:15:00	-73.991875	40.754437	-73.977230	
27	4.5	2011-06-28 19:47:00	-73.988893	40.760160	-73.986445	
28	9.3	2012-05-04 06:11:20	-73.989258	40.690835	-74.004133	
29	4.5	2013-08-11 00:52:00	-73.981020	40.737760	-73.980668	
30	5.5	2014-02-19 16:03:00	-73.976075	40.752422	-73.981082	
32	31.9	2009-01-09 16:10:00	-73.873027	40.773883	-73.984545	
...	...	...	...	...	...	
16036	10.5	2010-08-17 11:34:00	-73.990103	40.729750	-73.978462	
16037	6.5	2012-02-27 21:40:50	-73.992618	40.723878	-73.977073	
16038	5.7	2010-08-31 10:43:42	-73.990336	40.718973	-73.956060	
16039	12.9	2010-12-11 16:25:00	-73.936462	40.794292	-73.948747	
16040	6.5	2014-06-16 00:05:19	-73.980597	40.744267	-73.979330	
16041	11.0	2014-11-17 21:53:00	-73.983610	40.747090	-73.961310	
16042	8.5	2015-04-06 21:53:06	-73.991425	40.749832	-74.000107	
16043	8.5	2011-11-17 10:58:05	-73.973961	40.764055	-73.986807	
16044	16.5	2013-04-29 03:05:45	-73.982785	40.731421	-74.011358	
16045	6.5	2013-09-19 23:56:00	-73.995227	40.733475	-73.984030	
16046	6.0	2014-04-24 01:48:40	-73.976298	40.753948	-73.993062	
16047	6.1	2010-03-18 11:09:00	-73.970733	40.758193	-73.979457	
16048	9.7	2012-07-10 17:32:00	-73.988040	40.774902	-74.005265	
16049	15.7	2012-07-31 12:27:00	-74.008657	40.715975	-73.975653	
16050	8.5	2013-01-23 07:36:49	-73.996715	40.742504	-73.977987	
16051	11.5	2014-10-01 20:05:00	-73.975540	40.755590	-73.944780	

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropc
16052	10.0	2014-10-03 22:24:00	-73.987298	40.722007	-74.000267	
16053	4.0	2014-09-23 09:49:00	-73.954977	40.788582	-73.964227	
16054	5.3	2009-11-28 15:58:02	-73.993929	40.756944	-73.993044	
16055	48.3	2012-09-05 17:34:00	-73.994077	40.741242	-73.830257	
16056	38.3	2012-12-17 14:59:16	0.000000	0.000000	0.000000	
16057	5.0	2013-01-31 15:46:00	-73.963582	40.774242	-73.956525	
16058	5.5	2014-04-19 14:58:57	-73.974265	40.756048	-73.980885	
16059	5.3	2010-01-03 18:26:00	-73.973297	40.743768	-73.986060	
16060	22.0	2014-10-01 09:15:00	-73.954582	40.778047	-74.005982	
16061	10.9	2009-05-20 18:56:42	-73.994191	40.751138	-73.962769	
16062	6.5	2014-12-12 07:41:00	-74.008820	40.718757	-73.998865	
16063	16.1	2009-07-13 07:58:00	-73.981310	40.781695	-74.014392	
16064	8.5	2009-11-11 11:19:07	-73.972507	40.753417	-73.979577	
16065	8.1	2010-05-11 23:53:00	-73.957027	40.765945	-73.981983	

15901 rows × 13 columns

In [62]:
```
train['distance'] = train[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']].apply(haversine,axis=1)
```

In [63]:
```
test['distance'] = test[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude']].apply(haversine,axis=1)
```

In [64]:
```
train.head()
```

Out[64]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.7
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.7
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.7
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.7
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.7

In [65]:
```
test.head()
```

Out[65]:

	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
0	2015-01-27 13:08:24	-73.973320	40.763805	-73.981430	40.743835	
1	2015-01-27 13:08:24	-73.986862	40.719383	-73.998886	40.739201	
2	2011-10-08 11:53:44	-73.982524	40.751260	-73.979654	40.746139	
3	2012-12-01 21:12:12	-73.981160	40.767807	-73.990448	40.751635	
4	2012-12-01 21:12:12	-73.966046	40.789775	-73.988565	40.744427	

In [66]:
```
train.nunique()
```

Out[66]:
```
fare_amount            459
pickup_datetime      15856
pickup_longitude     13672
pickup_latitude      14110
dropoff_longitude    13763
dropoff_latitude     14136
passenger_count          7
year                     7
Month                   12
Date                    31
Day                      7
Hour                    24
Minute                  60
distance             15448
dtype: int64
```

In [67]:
```
test.nunique()
```

Out[67]:
```
pickup_datetime      1753
pickup_longitude     9124
pickup_latitude      9246
dropoff_longitude    9141
dropoff_latitude     9360
passenger_count         6
year                    7
Month                  12
Date                   31
Day                     7
Hour                   24
Minute                 60
distance             9830
dtype: int64
```

In [68]:
```
train['distance'].sort_values(ascending=False)
```

Out[68]:

```
9147     8667.542104
8647     8667.497512
2397     8667.454421
472      8667.304968
11653    8666.701504
13340    8666.613646
10215    8666.584706
4597     8666.566030
10458    8665.976222
10672    8665.702390
10488    8665.555634
1260     8665.268588
4278     8665.223767
6188     8664.191488
12983    8664.131808
6302     8663.039123
12705    8661.362152
14197    8657.136619
15783    8656.714168
15749    6028.926779
2280     6026.494216
5864     5420.988959
7014     4447.086698
10710     129.950482
14536     129.560455
11619     127.509261
12228     123.561157
5663      101.094619
1684       99.771579
3075       97.985088
              ...
7684        0.000000
4298        0.000000
13143       0.000000
3128        0.000000
8645        0.000000
8377        0.000000
4240        0.000000
2447        0.000000
4367        0.000000
11565       0.000000
13081       0.000000
13062       0.000000
4454        0.000000
13013       0.000000
13015       0.000000
808         0.000000
6462        0.000000
799         0.000000
4430        0.000000
10783       0.000000
13037       0.000000
14485       0.000000
15524       0.000000
9342        0.000000
13045       0.000000
13050       0.000000
```

```
11593       0.000000
2346        0.000000
8331        0.000000
1637        0.000000
Name: distance, Length: 15901, dtype: float64
```

In [69]:
```
train=train.drop(train[train['distance']==0].index,axis=0)
```

In [70]:
```
train=train.drop(train[train['distance']>130].index,axis=0)
```

In [71]:
```
train.shape
```

Out[71]:
```
(15424, 14)
```

In [72]:
```
train.head()
```

Out[72]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_la
0	4.5	2009-06-15 17:26:21	-73.844311	40.721319	-73.841610	40.7
1	16.9	2010-01-05 16:52:16	-74.016048	40.711303	-73.979268	40.7
2	5.7	2011-08-18 00:35:00	-73.982738	40.761270	-73.991242	40.7
3	7.7	2012-04-21 04:30:42	-73.987130	40.733143	-73.991567	40.7
4	5.3	2010-03-09 07:51:00	-73.968095	40.768008	-73.956655	40.7

In [73]:
```
drop = ['pickup_datetime', 'pickup_longitude', 'pickup_latitude','dropoff_long
itude', 'dropoff_latitude', 'Minute']
train = train.drop(drop, axis = 1)
```

In [74]:
```
train.head()
```

Out[74]:

	fare_amount	passenger_count	year	Month	Date	Day	Hour	distance
0	4.5	1.0	2009.0	6.0	15.0	0.0	17.0	1.030764
1	16.9	1.0	2010.0	1.0	5.0	1.0	16.0	8.450134
2	5.7	2.0	2011.0	8.0	18.0	3.0	0.0	1.389525
3	7.7	1.0	2012.0	4.0	21.0	5.0	4.0	2.799270
4	5.3	1.0	2010.0	3.0	9.0	1.0	7.0	1.999157

In [75]:
```
train.dtypes()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-75-2c798b256679> in <module>
----> 1 train.dtypes()

TypeError: 'Series' object is not callable
```

In [76]:
```
train.dtypes
```

Out[76]:
```
fare_amount        float64
passenger_count    float64
year               float64
Month              float64
Date               float64
Day                float64
Hour               float64
distance           float64
dtype: object
```

In [77]:
```
train['passenger_count']=train["passenger_count"].astype('int64')
```

In [78]:
```
train['year']=train["year"].astype('int64')
```

In [81]:
```
train['Month'] = train['Month'].astype('int64')
train['Date'] = train['Date'].astype('int64')
train['Day'] = train['Day'].astype('int64')
train['Hour'] = train['Hour'].astype('int64')
```

In [82]:
```
train.dtypes
```

Out[82]:
```
fare_amount        float64
passenger_count      int64
year                 int64
Month                int64
Date                 int64
Day                  int64
Hour                 int64
distance           float64
dtype: object
```

In [83]:
```
drop_test = ['pickup_datetime', 'pickup_longitude', 'pickup_latitude','dropoff
_longitude', 'dropoff_latitude', 'Minute']
test = test.drop(drop_test, axis = 1)
```

In [84]:
```
test.head()
```

Out[84]:

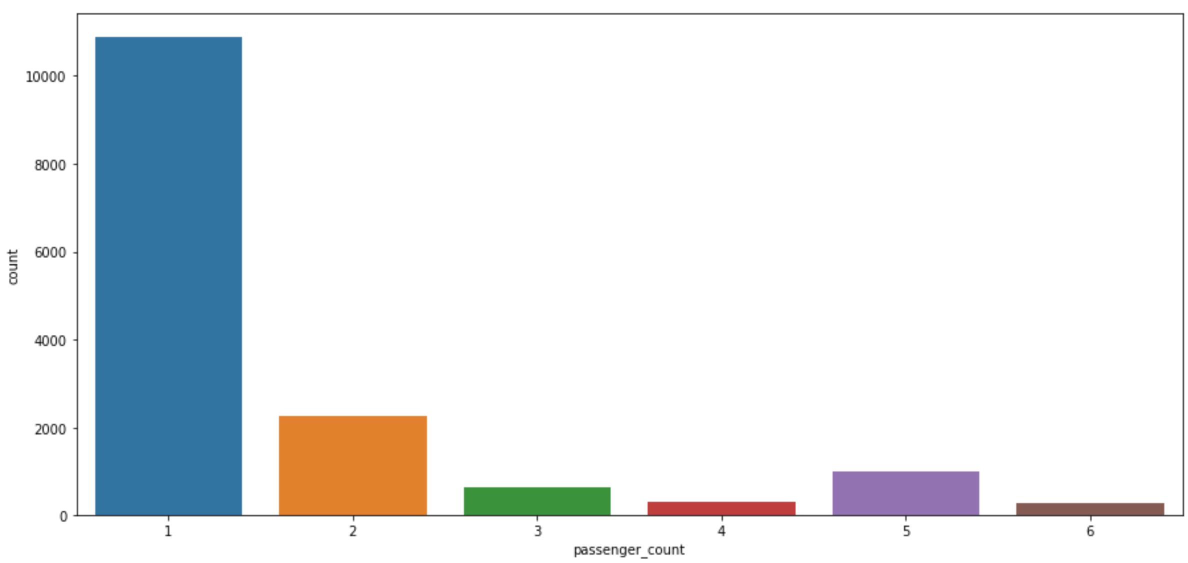
	passenger_count	year	Month	Date	Day	Hour	distance
0	1	2015	1	27	1	13	2.323259
1	1	2015	1	27	1	13	2.425353
2	1	2011	10	8	5	11	0.618628
3	1	2012	12	1	5	21	1.961033
4	1	2012	12	1	5	21	5.387301

In [86]:
```
test.dtypes
```

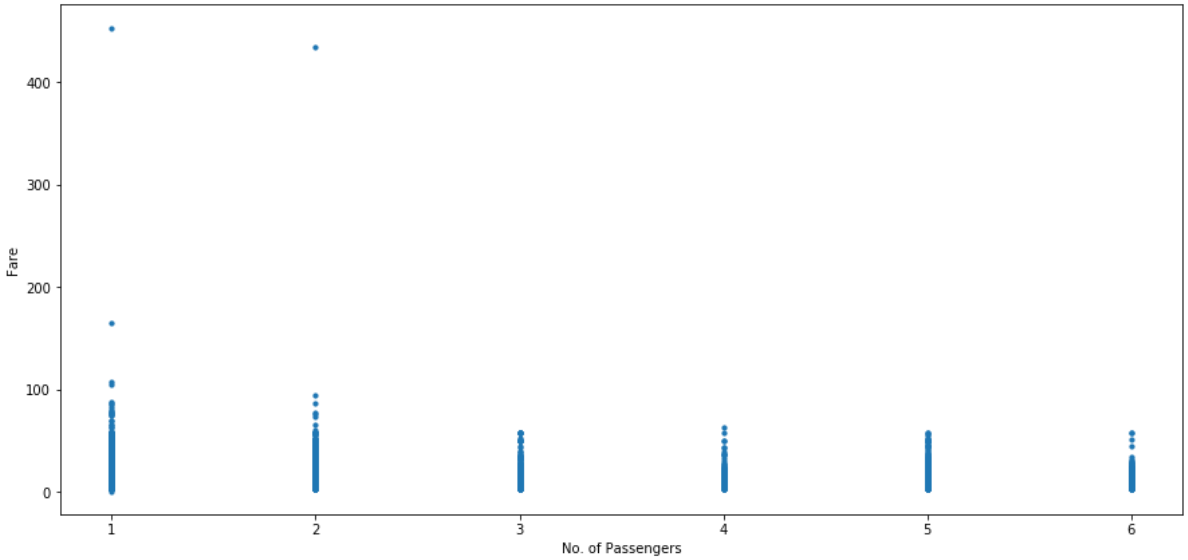
Out[86]:
```
passenger_count      int64
year                 int64
Month                int64
Date                 int64
Day                  int64
Hour                 int64
distance           float64
dtype: object
```

In [87]:
```
plt.figure(figsize=(15,7))
sns.countplot(x="passenger_count", data=train)
```

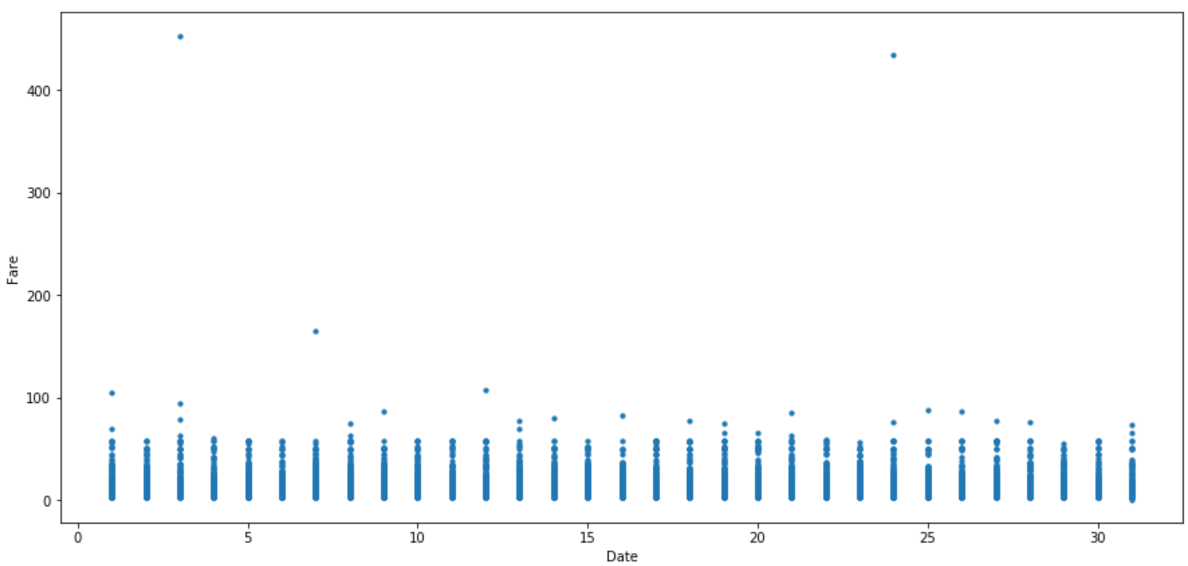
Out[87]: <matplotlib.axes._subplots.AxesSubplot at 0x8fdf470>



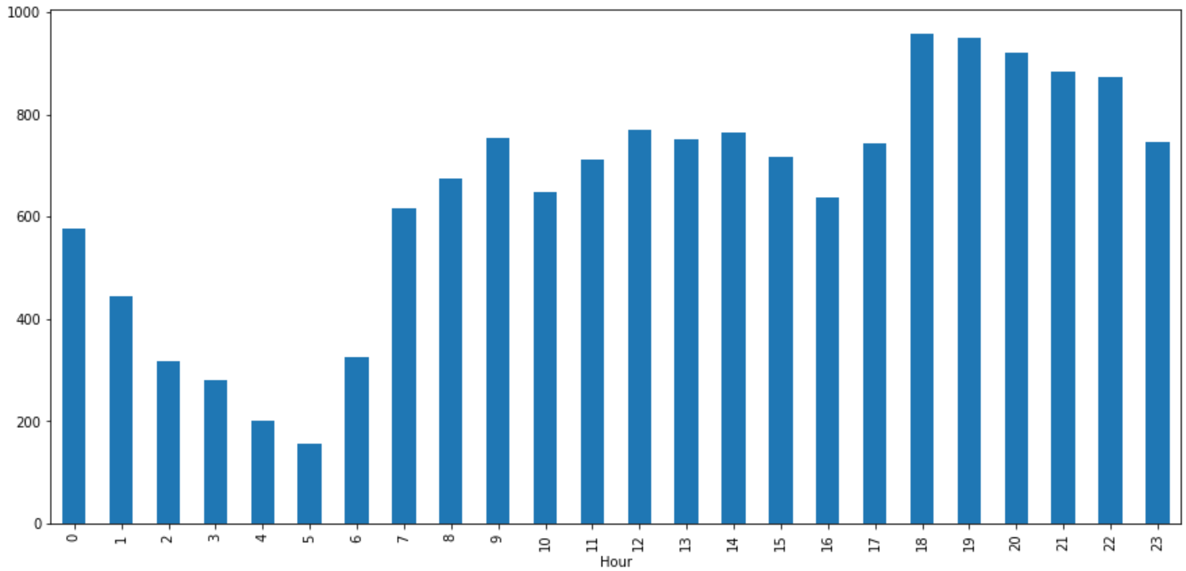
In [88]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=train['passenger_count'], y=train['fare_amount'], s=10)
plt.xlabel('No. of Passengers')
plt.ylabel('Fare')
plt.show()
```



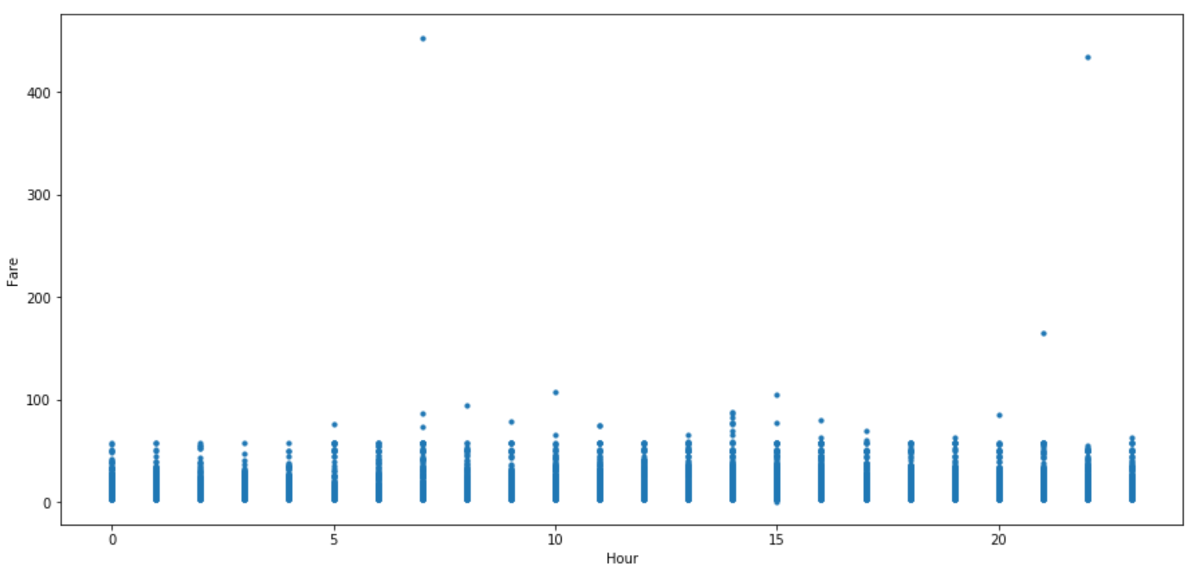
In [89]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=train['Date'], y=train['fare_amount'], s=10)
plt.xlabel('Date')
plt.ylabel('Fare')
plt.show()
```



In [90]:
```
plt.figure(figsize=(15,7))
train.groupby(train["Hour"])['Hour'].count().plot(kind="bar")
plt.show()
```

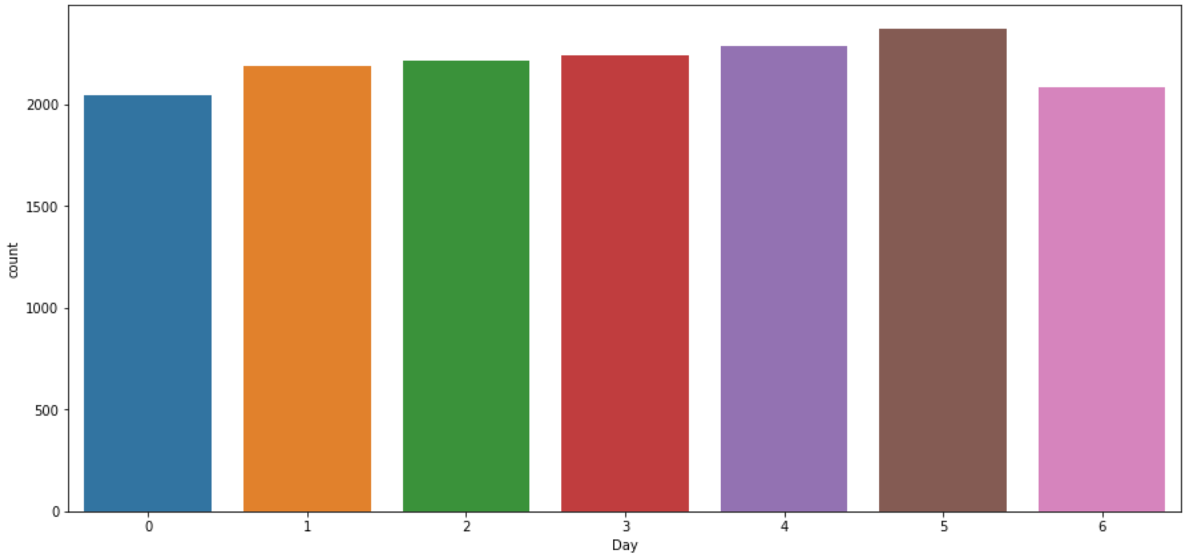


In [91]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=train['Hour'], y=train['fare_amount'], s=10)
plt.xlabel('Hour')
plt.ylabel('Fare')
plt.show()
```

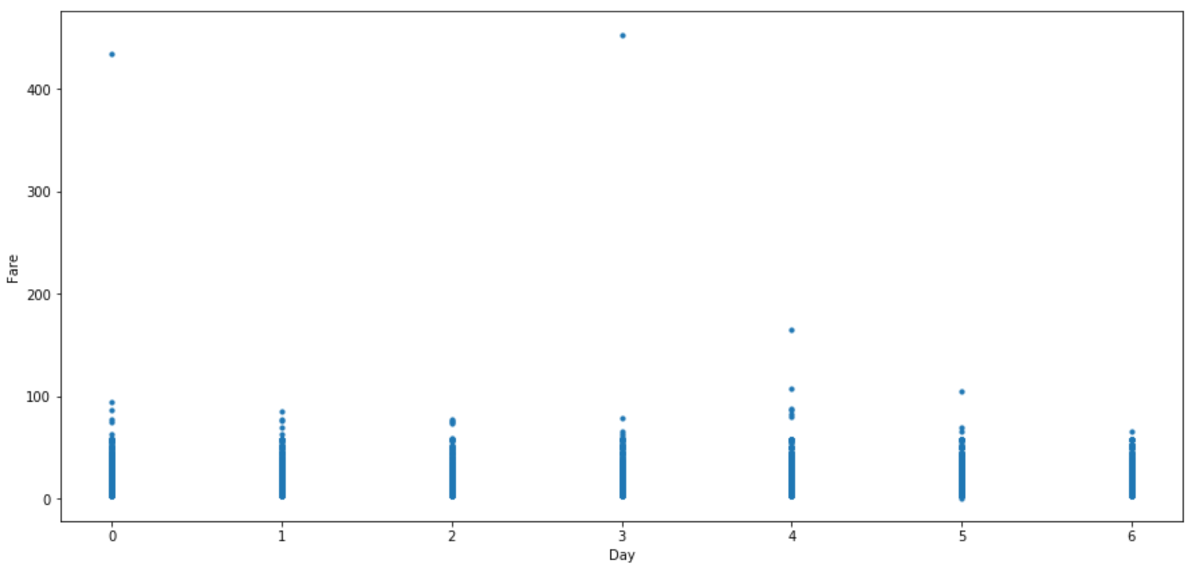


In [92]:
```
plt.figure(figsize=(15,7))
sns.countplot(x="Day", data=train)
```

Out[92]: <matplotlib.axes._subplots.AxesSubplot at 0xca63ac8>

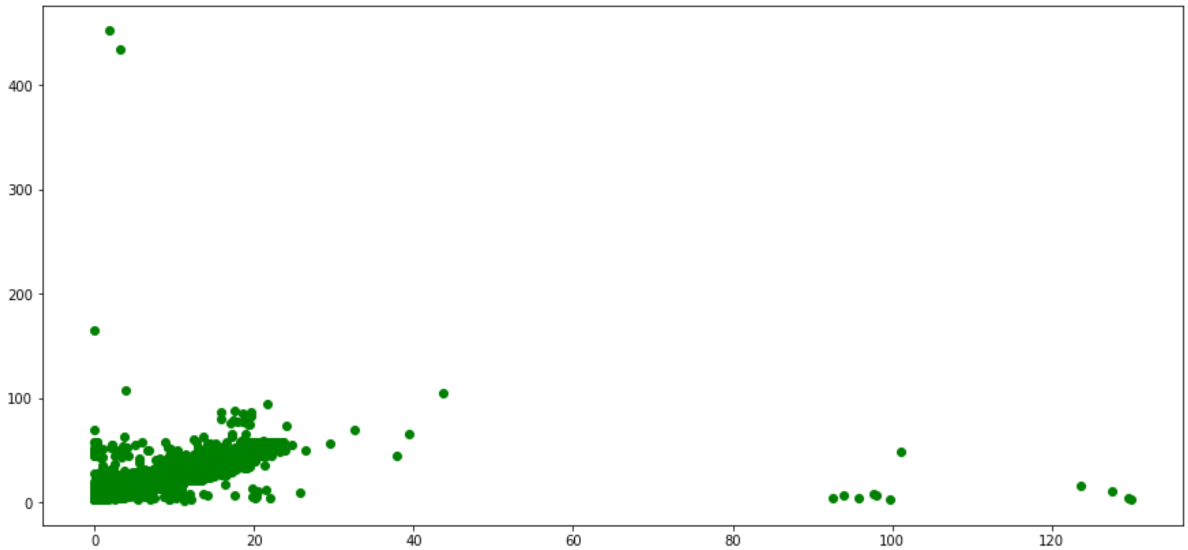


In [93]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=train['Day'], y=train['fare_amount'], s=10)
plt.xlabel('Day')
plt.ylabel('Fare')
plt.show()
```

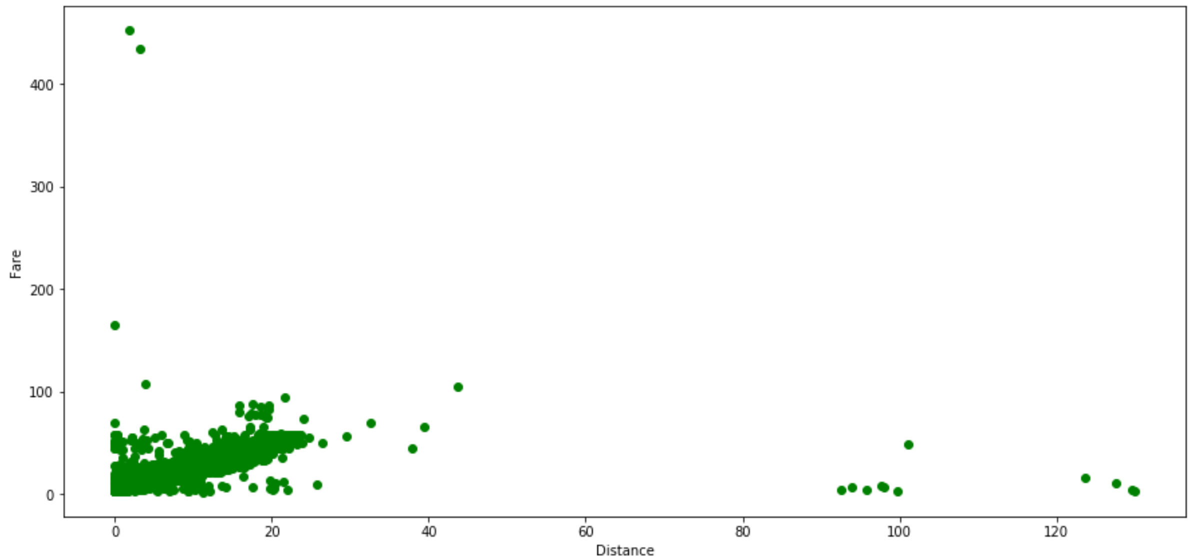


In [94]:
```
plt.figure(figsize=(15,7))
plt.scatter(x=train['distance'],y=train['fare_amount'],c="g")
plt.xlable('Distance')
plt.ylable('Fare')
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-94-0a1792cbfce2> in <module>
      1 plt.figure(figsize=(15,7))
      2 plt.scatter(x=train['distance'],y=train['fare_amount'],c="g")
----> 3 plt.xlable('Distance')
      4 plt.ylable('Fare')
      5 plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'xlable'
```

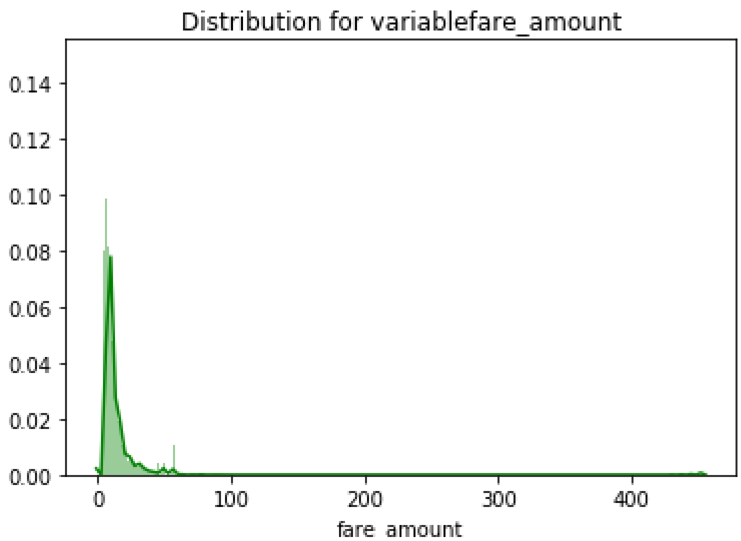


In [95]:
```
plt.figure(figsize=(15,7))
plt.scatter(x = train['distance'],y = train['fare_amount'],c = "g")
plt.xlabel('Distance')
plt.ylabel('Fare')
plt.show()
```



In [96]:
```
for i in['fare_amount','distance']:
    print(i)
    sns.distplot(train[i],bins='auto',color='green')
    plt.title("Distribution for variable" +i)
    plt.ylable("Density")
    plt.show()
```

```
fare_amount

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-96-378b2713d885> in <module>
      3     sns.distplot(train[i],bins='auto',color='green')
      4     plt.title("Distribution for variable" +i)
----> 5     plt.ylable("Density")
      6     plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'ylable'
```

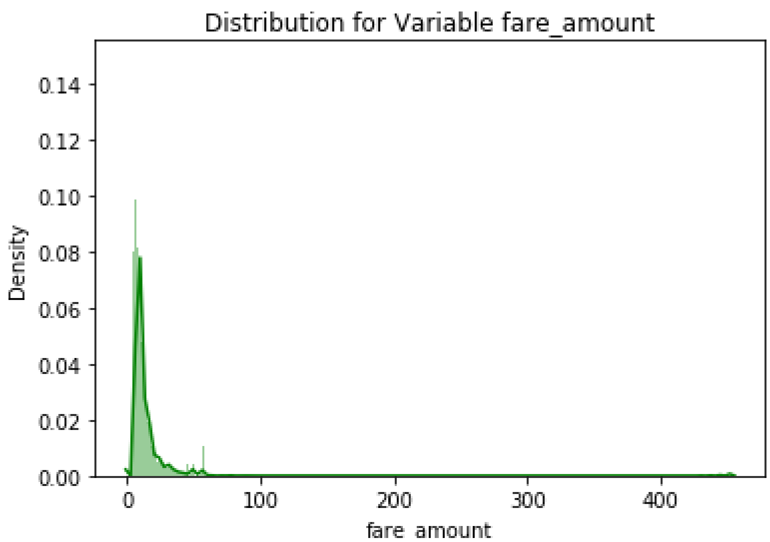


In [97]:
```
for i in['fare_amount', 'distance']:
    print(i)
    sns.displot(train[i],bins='auto',color='green')
    plt.title("Distribution for Variable"+i)
    plt.ylable("Density")
    plt.show()
```

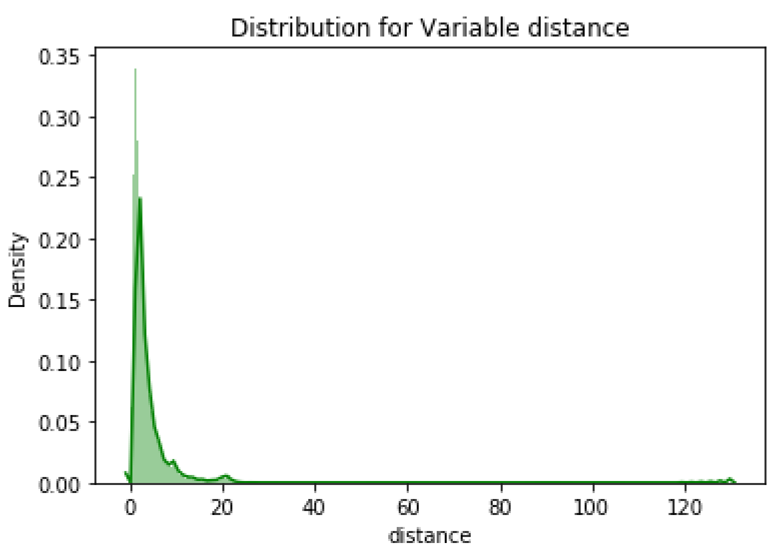
```
fare_amount

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-97-87c433da0521> in <module>
      1 for i in['fare_amount', 'distance']:
      2     print(i)
----> 3     sns.displot(train[i],bins='auto',color='green')
      4     plt.title("Distribution for Variable"+i)
      5     plt.ylable("Density")

AttributeError: module 'seaborn' has no attribute 'displot'
```

In [98]:
```
for i in ['fare_amount', 'distance']:
    print(i)
    sns.distplot(train[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()
```

```
fare_amount
```




```
distance
```

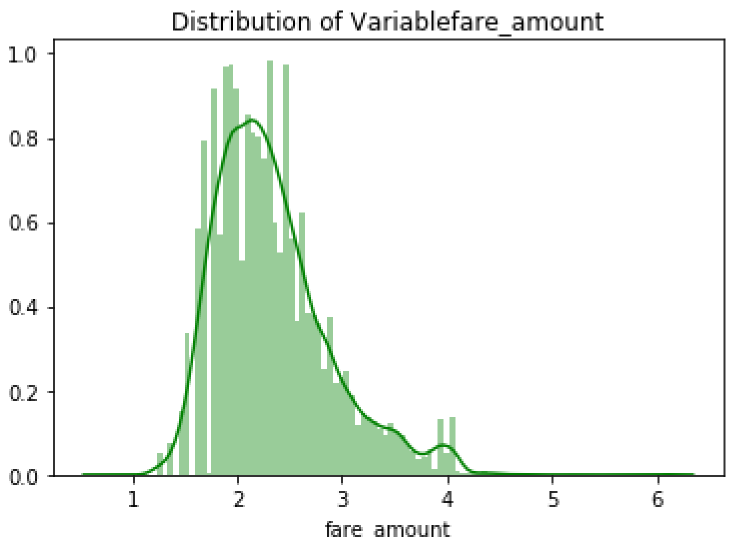


In [99]:
```
train['fare_amount']=np.log1p(train['fare_amount'])
```

In [100]:
```
train['distance']=np.log1p(train['distance'])
```

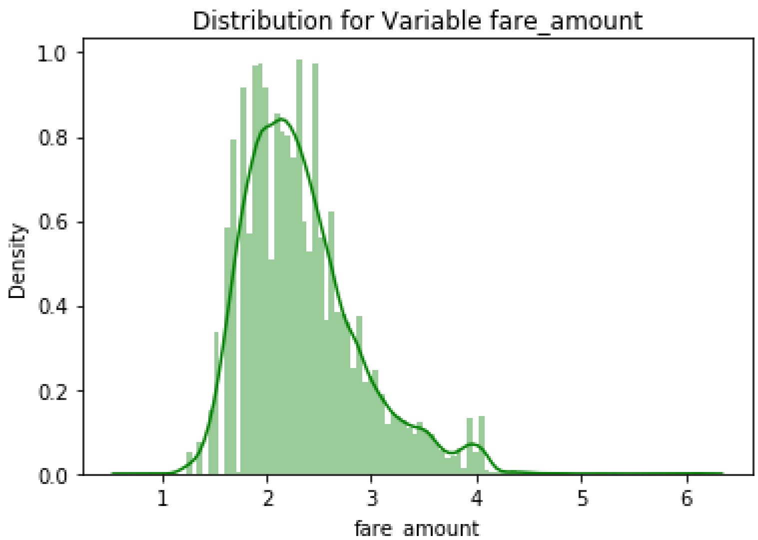
In [103]:
```
for i in ["fare_amount","distance"]:
          print(i)
          sns.distplot(train[i],bins='auto',color='green')
          plt.title("Distribution of Variable" +i)
          plt.ylable("Density")
          plt.show()
```

```
fare_amount

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-103-12759390a634> in <module>
      3           sns.distplot(train[i],bins='auto',color='green')
      4           plt.title("Distribution of Variable" +i)
----> 5           plt.ylable("Density")
      6           plt.show()

AttributeError: module 'matplotlib.pyplot' has no attribute 'ylable'
```

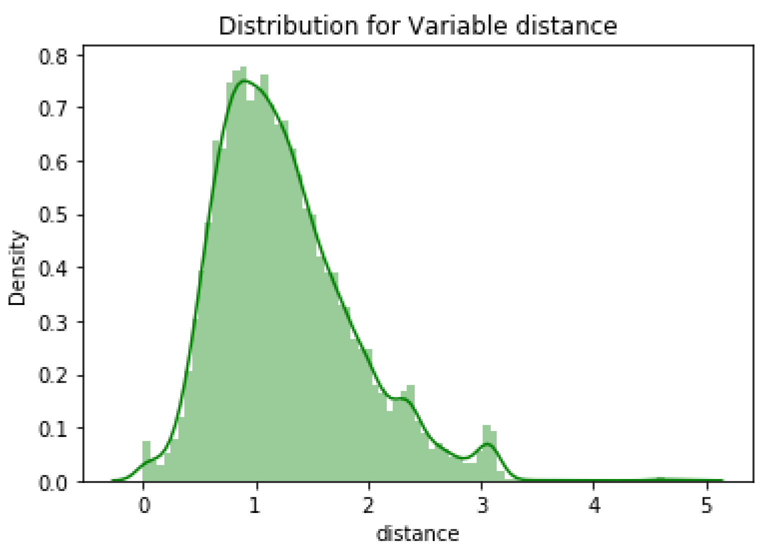


In [104]:
```
for i in ['fare_amount', 'distance']:
    print(i)
    sns.distplot(train[i],bins='auto',color='green')
    plt.title("Distribution for Variable "+i)
    plt.ylabel("Density")
    plt.show()
```

```
fare_amount
```



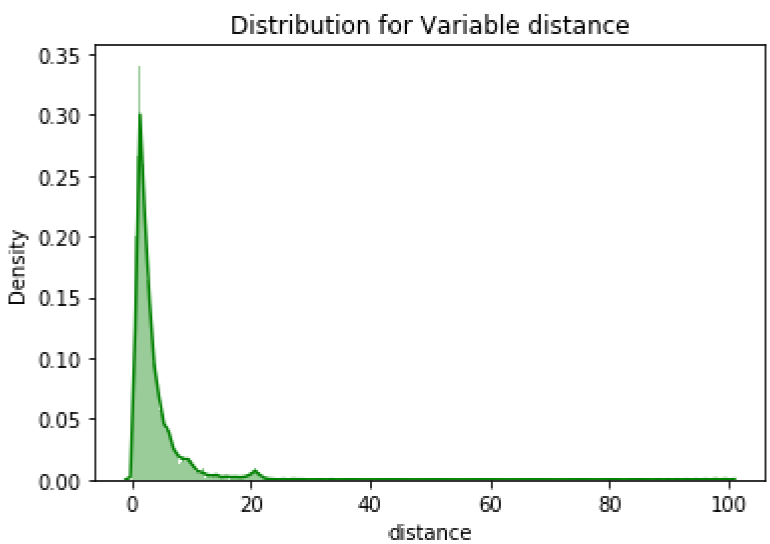

```
distance
```



In [105]:
```
sns.displot(test['distance'],bins='auto',color='green')
plt.title("Distribution for Variable" +i)
plt.ylabel("Density")
plt.show()
```

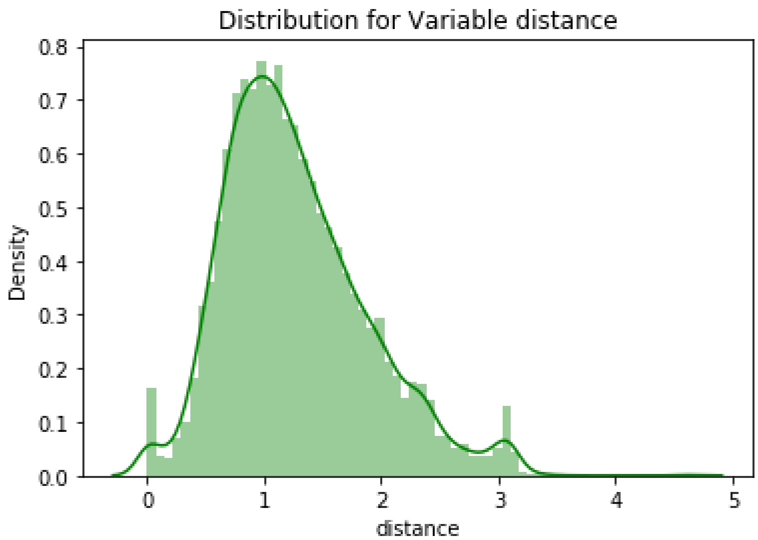
```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-105-3c7492627f0e> in <module>
----> 1 sns.displot(test['distance'],bins='auto',color='green')
      2 plt.title("Distribution for Variable" +i)
      3 plt.ylabel("Density")
      4 plt.show()

AttributeError: module 'seaborn' has no attribute 'displot'
```

In [106]:
```
sns.distplot(test['distance'],bins='auto',color='green')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()
```



In [107]:
```
test['distance'] = np.log1p(test['distance'])
```

In [108]:
```
sns.distplot(test['distance'],bins='auto',color='green')
plt.title("Distribution for Variable "+i)
plt.ylabel("Density")
plt.show()
```



In [109]:
```
X_train, X_test, y_train, y_test = train_test_split( train.iloc[:, train.colum
ns != 'fare_amount'],
                                    train.iloc[:, 0], test_size = 0.20, random_state = 1)
```

In [110]:
```
print(X_train.shape)
```

```
(12339, 7)
```

In [111]:
```
print(X_test.shape)
```

```
(3085, 7)
```

In [113]:
```
LR = LinearRegression().fit(X_train,y_train)
```

In [114]:
```
pred_train_LR = LR.predict(X_train)
```

In [115]:
```
pred_test_LR = LR.predict(X_test)
```

In [116]:
```
RMSE_test_LR = np.sqrt(mean_squared_error(y_test, pred_test_LR))
```

In [117]:
```
RMSE_train_LR = np.sqrt(mean_squared_error(y_train, pred_train_LR))
```

In [118]:
```
print("Root Mean Squared Error-Training data="+str(RMSE_train_LR))
print("Root Mean Squared Error-Test Data"+str(RMSE_test_LR))
```

```
Root Mean Squared Error-Training data=0.27531100179673135
Root Mean Squared Error-Test Data0.24540661786977466
```

In [119]:
```
from sklearn.metrics import r2_score
```

In [120]:
```
r2_score(y_train,pred_train_LR)
```
Out[120]:
```
0.7495502651880406
```

In [121]:
```
r2_score(y_test,pred_test_LR)
```
Out[121]:
```
0.7827019104296646
```

In [122]:
```
DT = DecisionTreeRegressor(max_depth=2).fit(X_train,y_train)
```

In [123]:
```
pred_train_DT = DT.predict(X_train)
pred_test_DT = DT.predict(X_test)
```

In [125]:
```
RMSE_train_DT = np.sqrt(mean_squared_error(y_train,pred_train_DT))
RMSE_test_DT = np.sqrt(mean_squared_error(y_test,pred_test_DT))
```

In [127]:
```
print("RMSE-Training="+str(RMSE_train_DT))
```
```
RMSE-Training=0.29962109020770195
```

In [128]:
```
print("RMSE-Training="+str(RMSE_test_DT))
```
```
RMSE-Training=0.2867460617158615
```

In [130]:
```
r2_score(y_train,pred_train_DT)
r2_score(y_test,pred_test_DT)
```
Out[130]:
```
0.7033268167661039
```

In [131]:
```
RF = RandomForestRegressor(n_estimators=200).fit(X_train,y_train)
```

In [132]:
```
pred_train_RF = RF.predict(X_train)
```

In [133]:
```
pred_test_RF = RF.predict(X_test)
```

In [134]:
```
RMSE_train_RF = np.sqrt(mean_squared_error(y_train, pred_train_RF))
```

In [135]:
```
RMSE_test_RF = np.sqrt(mean_squared_error(y_test, pred_test_RF))
```

In [136]:
```
print("RMSE - Training data = "+str(RMSE_train_RF))
print("RMSE - Test data = "+str(RMSE_test_RF))
```
```
RMSE - Training data = 0.09536265926338224
RMSE - Test data = 0.23536810226935345
```

In [137]:
```
r2_score(y_train,pred_train_RF)
```
Out[137]:
```
0.969950991321187
```

In [138]:
```
r2_score(y_test, pred_test_RF)
```

Out[138]: 0.8001157480242431

In [139]:
```
GB = GradientBoostingRegressor().fit(X_train, y_train)
```

In [141]:
```
pred_train_GB = GB.predict(X_train)
```

In [142]:
```
pred_test_GB = GB.predict(X_test)
```

In [143]:
```
RMSE_train_GB = np.sqrt(mean_squared_error(y_train, pred_train_GB))
```

In [144]:
```
RMSE_test_GB = np.sqrt(mean_squared_error(y_test, pred_test_GB))
```

In [145]:
```
print("RMSE - Training data ="+str(RMSE_train_GB))
print("RMSE - Test data ="+str(RMSE_test_GB))
```

```
RMSE - Training data =0.22754316149645537
RMSE - Test data =0.22742812508816154
```

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

Out[147]: 0.8133741881626637

In [148]:
```
r2_score(y_train, pred_train_GB)
```

Out[148]: 0.8289193000175024

In [149]:
```
from sklearn.ensemble import RandomForestRegressor

rf = RandomForestRegressor(random_state = 42)
from pprint import pprint
print("Parameters:\n")
pprint(rf.get_params())
```

```
Parameters:

{'bootstrap': True,
 'criterion': 'mse',
 'max_depth': None,
 'max_features': 'auto',
 '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,
 'n_estimators': 'warn',
 'n_jobs': None,
 'oob_score': False,
 'random_state': 42,
 'verbose': 0,
 'warm_start': False}
```

In [150]:
```
from sklearn.model_selection import train_test_split, RandomizedSearchCV
```

In [151]:
```
RRF = RandomForestRegressor(random_state = 0)
n_estimator = list(range(1,20,2))
```

In [152]:
```
depth = list(range(1,100,2))
```

In [153]:
```
rand_grid ={"n_estimators": n_estimator,"max_depth":depth}
```

In [154]:
```
randomcv_rf = RandomizedSearchCV(RRF, param_distributions = rand_grid, n_iter=
5, cv = 5, random_state = 0)
```

In [156]:
```
randomcv_rf = randomcv_rf.fit(X_train, y_train)
```

In [157]:
```
predictions_RRF = randomcv_rf.predict(X_test)
```

In [158]:
```
view_best_params_RRf = randomcv_rf.best_params_
```

In [159]:
```
best_model = randomcv_rf.best_estimator_
best_model=randomcv_rf.best_estimator_
```

In [160]:
```
predictions_RRF = best_model.predict(X_test)
```

In [161]:
```
RRF_rmse = np.sqrt(mean_squared_error(y_test,predictions_RRF))
```

In [166]:
```
print('Random Search CV Random Forest Regressor Model Performance:')
print('RMSE = ',RRF_rmse)
```

```
Random Search CV Random Forest Regressor Model Performance:
RMSE =  0.23730781853507238
```

In [168]:
```
gb = GradientBoostingRegressor(random_state = 42)
from pprint import pprint
print("parameters currently in use\n")
pprint(gb.get_params())
```

```
parameters currently in use

{'alpha': 0.9,
 'criterion': 'friedman_mse',
 'init': None,
 'learning_rate': 0.1,
 'loss': 'ls',
 'max_depth': 3,
 '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,
 'n_estimators': 100,
 'n_iter_no_change': None,
 'presort': 'auto',
 'random_state': 42,
 'subsample': 1.0,
 'tol': 0.0001,
 'validation_fraction': 0.1,
 'verbose': 0,
 'warm_start': False}
```

In [169]:
```
gb = GradientBoostingRegressor(random_state = 0)
n_estimator= list(range(1,20,2))
depth = list(range(1,100,2))
rand_grid = {"n_estimators": n_estimator,"max_depth": depth}
```

In [175]:
```
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)

gb_r2 = r2_score(y_test, predictions_gb)

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 =  0.25199340493550487
```

In [173]:
```
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
```

```
  File "<ipython-input-173-6e04cc40f5e4>", line 6
    predictions_gb = best_model.predict(X_test
                                              ^
SyntaxError: unexpected EOF while parsing
```

In [176]:
```
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)

gb_r2 = r2_score(y_test, predictions_gb)

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 =  0.25199340493550487
```

In [177]:
```
from sklearn.model_selection import GridSearchCV
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))
grid_search = {'n_estimators': n_estimator,'max_depth': depth}
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_
predictions_GRF = gridcv_rf.predict(X_test)
GRF_r2 = r2_score(y_test, predictions_GRF)
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))
```

```
Grid Search CV Random Forest Regressor Model Performance:
Best Parameters =  {'max_depth': 7, 'n_estimators': 18}
R-squared = 0.8.
RMSE =  0.23637990451376567
```

In [179]:
```
gb = GradientBoostingRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))
grid_search = {'n_estimators': n_estimator,'max_depth': depth}
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_
predictions_Ggb = gridcv_gb.predict(X_test)
Ggb_r2 = r2_score(y_test, predictions_Ggb)
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 =  0.23724212611002213
```

In [180]:
```
regr = RandomForestRegressor(random_state = 0)
n_estimator = list(range(11,20,1))
depth = list(range(5,15,2))
grid_search ={"n_estimators":n_estimator, "max_depth": depth}
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_
predictions_GRF_test_Df = gridcv_rf.predict(test)
```

In [182]:
```
predictions_GRF_test_Df
```

Out[182]:
```
array([2.36760025, 2.39383317, 1.6809062 , ..., 4.01224357, 3.29348722,
       2.0360277 ])
```

In [183]:
```
test['Predicted_fare']=predictions_GRF_test_Df
```

In [184]:
```
test.head()
```

Out[184]:

	passenger_count	year	Month	Date	Day	Hour	distance	Predicted_fare
0	1	2015	1	27	1	13	1.200946	2.367600
1	1	2015	1	27	1	13	1.231205	2.393833
2	1	2011	10	8	5	11	0.481579	1.680906
3	1	2012	12	1	5	21	1.085538	2.209257
4	1	2012	12	1	5	21	1.854312	2.815112

In [185]:
```
test.to_csv('test.csv')
```

In [ ]: