Project Report

On

Cab Fare Prediction

Author : Mitta Jitendar Reddy

Table of Contents

Chapter1 Introduction

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 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 Business Understanding:

Our main goal from problem statement is prediction of fare amount. In general, the fare is mainly depending on how far the customer travel. Sometimes fares will be increase due to high demand this will occur in some cases which we need to identify using the previous data may be there will be high demand in July or any and also depends on the day of week like Sunday (fare is high on). Mainly the fare amount changes extensively based upon the place. We need to consider all such cases.

1.3 Data Understanding:

After a business understanding we need to understand the data provided it is more important because on which we extract features and finally train our model. From the pilot project we got a data. In our dataset we have 16067 rows, 7 Columns.

In 7 columns there are 6 independent variables and one dependent variable

In data there are missing values and outliers.

Independent variables:

pickup_datetime

pickup_longitude

pickup_latitude

dropoff_longitude

dropoff_latitude

passenger_count

Dependent variables:

fare_amount

Chapter 2

Data Preprocessing

Data preprocessing is the major step before training the model because the RealWorld data we obtained is incomplete and inconsistent there are lack of behaviors. In some cases, we need to extract the new features from the data we have this can be done on a clear understanding of business problem and data.

2.1 Missing Value Analysis

In our data there are missing values which we need to identify.

Variables	**Missing value count**
fare_amount	24
Pickup_datetime	0
Pickup_longitude	0
Pickup_latitude	0
dropoff_longitude	0
dropoff_latitude	0
Passenger_count	55

Total ratio of missing values is (55+24)/16067 = 0.0049

Missing count is <<1% we can remove them.

After remove rows left is 15988

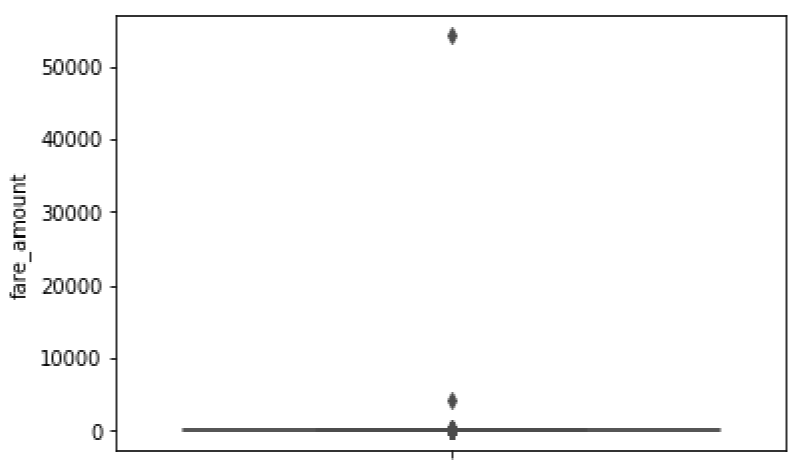
2.2 Identify Outliers

The outliers can be identified by picking up each variable. First, we will check using variable “fare_amount”.

Below table is the description of “fare_amount” variable.

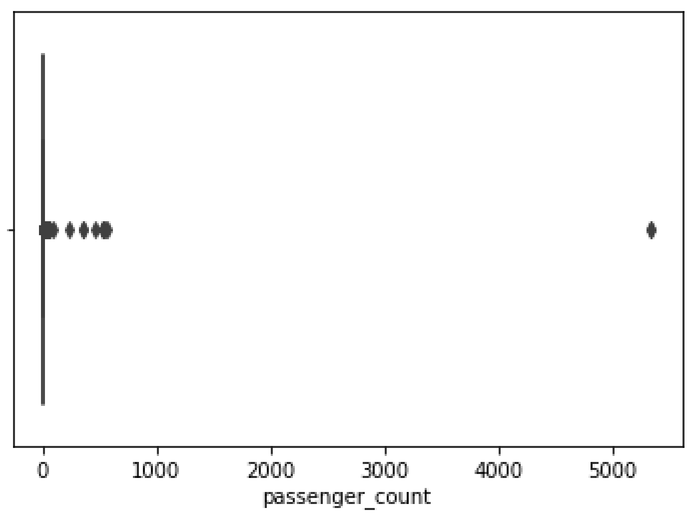
count	15987.000000
mean	15.056410
std	431.212947
min	-3.0000000
25%	6.000000
50%	8.500000
75%	12.500000
max	54343.000000

On Analyzing the above table max fare_amount is too large compared to mean. From that we can easily say there are outliers present in the data.



From the above box plot it looks like there are two outliers that can find using fare_amount.so remove the records which have fare amount >1000.

Now, we find outliers using passenger_count,



From the boxplot we can say there are outliers in the data that can find using “passenger_count”. In RealWorld in any cab there are at most 6 passengers can travel.

So remove the records who have passenger_count>6.

Now, perform outlier analysis on latitudes and longitudes. We all know that latitudes are range from –90 to 90. Longitudes are range from –180 to 180.

If there any records excluding those values then remove them as outliers.

Chapter 3

Feature Engineering

3.1 Extracting more features

We cannot perform directly using the given data. The model needs to understand the data for which we need to convert the data according to that. So first split the “pickup_datetime” feature into multiple features.

From this we can know the customer travel month, day of week, year, time of travel. Which the model can understand more better.

Using our techniques create more features and into the main data.

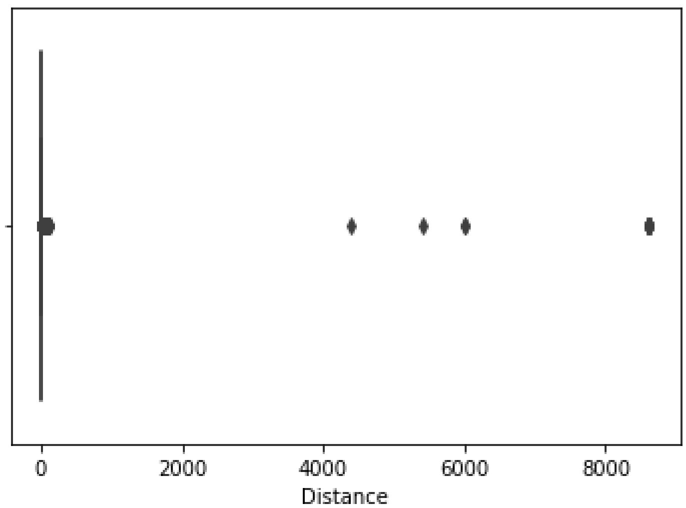
And also convert the latitude and longitude data more meaningful. For prediction of cab fares most important feature is the distance this can achieve through the latitudes and longitudes.

So find the distance using pickup_latitude, pickup_longitude,dropoff_latitude,dropoff_longitude this can be done using geodesic distance.

Through which our new distance feature is added.

3.2 Find Outliers using additional features

We added Distance as new feature from which there is a chance of identifying outliers. Below is the boxplot of “distance” feature



On analysis of above boxplot there are few outliers values greater than 4000. So remove them from master data.

3.3 Feature Selection

We have now total 12 independent features and 1 dependent feature.

12 independent features are:

- Pickup_latitude
- Pickup_longitude
- Dropoff_latitude
- Dropoff_longitude
- Month
- Day_of_week
- Year
- Passenger count
- Hour
- Date
- Pickup_datetime
- Distance

1 dependent feature is:

- Fare_amount

The importance of having pickup and drop-off latitudes and longitudes are to find the distance we calculate and added it as a new feature. So, there

Is no use of still having those variables we can remove those columns.

The importance of having pickup_datetime column is to know at which month and time the customer is travel. We already make a new columns to identify those features so we can remove them.

Final features are used for training are:

- Month
- Day_of_week
- Year
- Passenger count
- Hour
- Date
- Distance

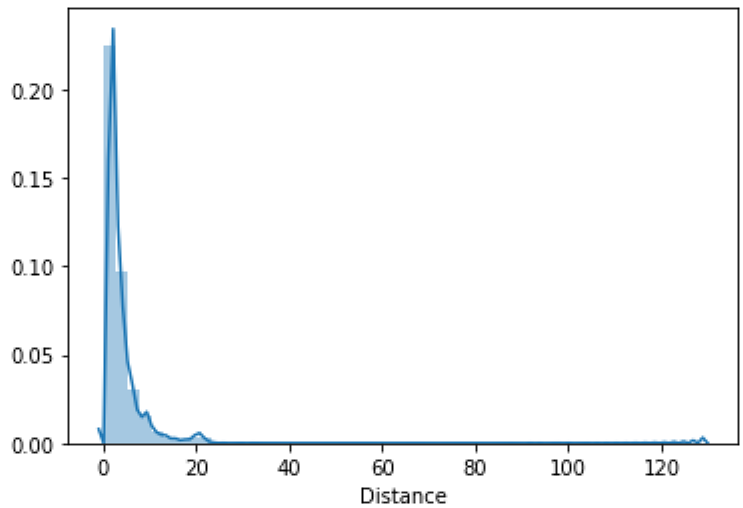
	fare_amount	passenger _count	year	Month	Date	Day	Hour	Distance
0	4.5	1	2009	6	15	0	17	1.023623
1	16.9	1	2010	1	5	1	16	8.394418
2	5.7	2	2011	8	18	3	0	1.381067
3	7.7	1	2012	4	21	5	4	2.779557
4	5.3	1	2010	3	9	1	7	1.986735

3.4 Feature Scaling

Before a move on to train the model, scaling is important. If our data is skewed then it will get effect on the final results. **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.

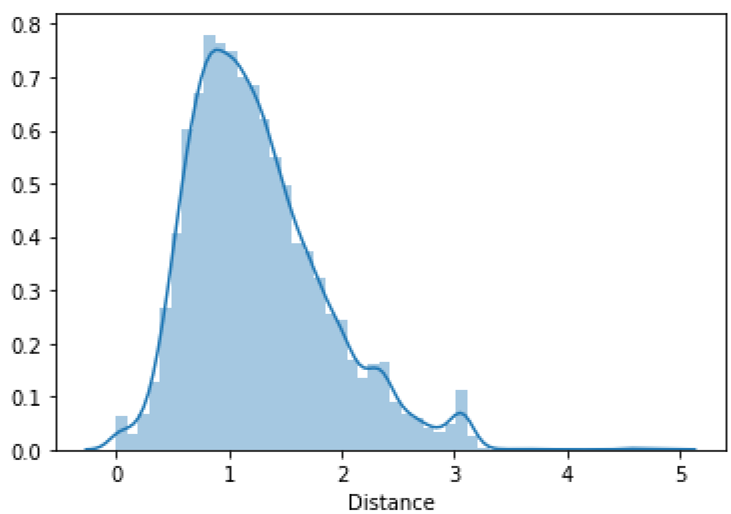
This can be overcome using log transform of the data.

Checking skewness for “distance ” feature



Distance feature is affected with skewness convert into normal distribution using log-transform.

Below graph represents “Distance” feature plot after log-transform.



Now, the graph looks like nearly the bell shaped

Chapter 4: Modeling

- Linear Regression
- Decision Tree Regression
- K-Nearest Neighbors Regression
- Random Forest Regression

Our target variable is continuous so it comes under regression. Now, we need to train the regression models. From which we need to select the best one.

we will use some regression models on our processed data to predict the target variable. Following are the models which we have built

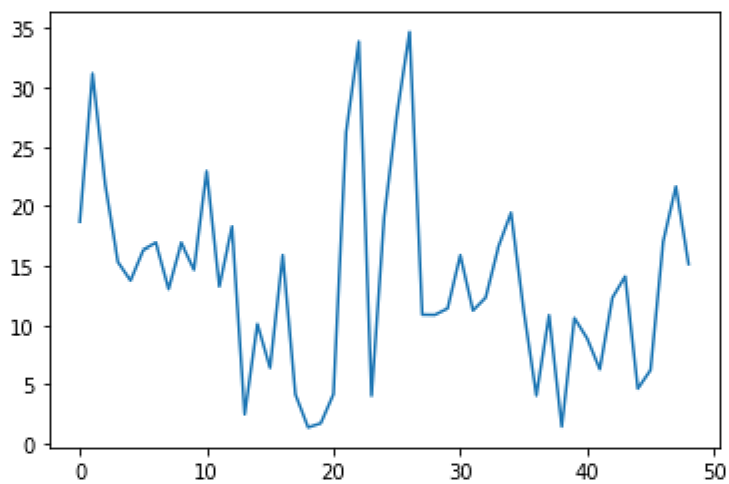
- Linear Regression
- Decision Trees
- K-Nearest Neighbors
- Random Forest

We have now only training data from which we split the data into train data and test data. Due to which train data is used to train the model and using test data we can know how much the model is fit and which model will be the best among them.

```
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=234)
```

Linear Regression

Multiple regression, is a statistical technique that uses **several** explanatory variables to predict the outcome of a response variable. While training the linear regression it initially assigns weights randomly to all independent variables to calculate the dependent variable value after that using training data in each iteration weights are adjusted to the minimized value. After fit the data test using the test



The above plot is the predictions of first 50 records of test data

Due to sklearn library our job is made simple in programming

We can train

Coefficients:

Estimate	Std.	Error	t value	Pr(>|t|)
(Intercept)	-353.98592	708.98472	-0.499	0.6177
passenger_count	-0.21505	0.54045	-0.398	0.6908
Year	0.17452	0.35238	0.495	0.6205
Month	-0.07683	0.18932	-0.406	0.6849
Weekday2	2.41709	2.42723	0.996	0.3195
Weekday3	-0.30658	2.35511	-0.130	0.8964
Weekday4	-0.47749	2.45739	-0.194	0.8460
Weekday5	5.40513	2.43788	2.217	0.0268 *
Weekday6	-0.82096	2.41676	-0.340	0.7341
Weekday7	-0.84045	2.38094	-0.353	0.7242
Hour	0.07093	0.10166	0.698	0.4855
distance	11.67767	1.06660	10.948	<2e-16 ***

After training the data using linear regression then above table is the summary of the model. From the all independent variables p-value for distance is very low which represents this “distance” variable is more important for output variable.

After training the model the errors are:

Root mean square error is 6.472476

Mean absolute percentage error is 0.671424

Mean absolute error is 0.671424

Decision Tree Regressor

Decision tree regressor models in the form of tree structure. It brokes the data into smaller and smaller subsets while at the same time an associated decision tree grows.

In a decision tree the we can set hyperparameters as length of the tree, minimum samples in each subset.

This best hyperparameters can find using grid search and random search.

In python we can train decision tree using

```
DTR=DecisionTreeRegressor()

DTR.fit(X_train,Y_train)
```

After training the decision tree the errors are:

Root mean square error is 9.975412

Mean absolute percentage error is 0.2944733

Mean absolute error is 4.156102

K-Nearest Neighbors:

K-nearest neighbors train the model by grouping all nearest neighbors into one and find the output value by averaging all the values present in the group. Using K-Nearest Neighbors we can done classification by majority vote and regression by averaging the values.

After training the decision tree the errors are:

Root mean square error is 8.763256589457352

Mean absolute percentage error is 0.2844733

Mean absolute error is 5.156102

Random Forest

Random forest is an ensemble method means it trains the model with multiple decision trees. Here the main hyperparameter is how many decision trees are used to train the model. After training the random forest model the test data get target value from multiple trees. Later aggregate all the models result from that we get final value.

After training the decision tree the errors are:

Root mean square error is 6.831194

Mean absolute percentage error is 0.2421828

Mean absolute error is 3.66458

While applying grid search we can find best hyperparameters for the model due to which our model accuracy will increase

```
rf=RandomForest()
clf=GridSearchCV(rf, {'n_estimators':[70,80,90,100,110,120,130],'max_depth':[2,4,6,8,10]})
clf.fit(X_train,Y_train)
```

Conclusion

Model Evaluation

For evaluating models as the target is continuous we use techniques like RMSE, MAPE etc. On evaluating all trained models, the model is well trained using decision tree and random forest we can say this because the mean absolute percentage error is very low for the random forest (0.24).

MAPE for Linear Regression is 0.671424

MAPE for Decision Tree is 0.2944733

MAPE for KNN is 0.2844733

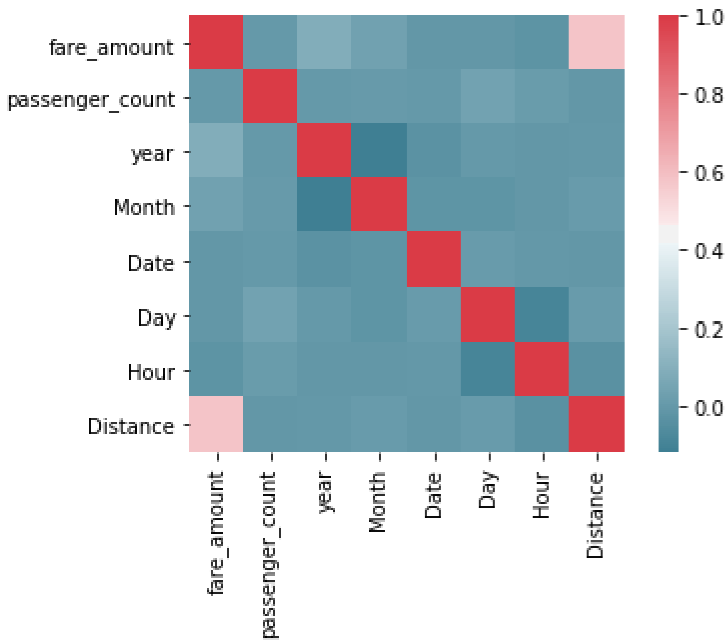
MAPE for Random Forest is 0.2521828

MODEL SELECTION

After using grid search we know the best parameters for decision trees and random forest. From the above MAPE errors we conclude that Random forest model has low error. Due to apply of grid search on hyperparameters the model accuracy is increased now the random forest accuracy is 0.24. which has great accuracy among all the models.

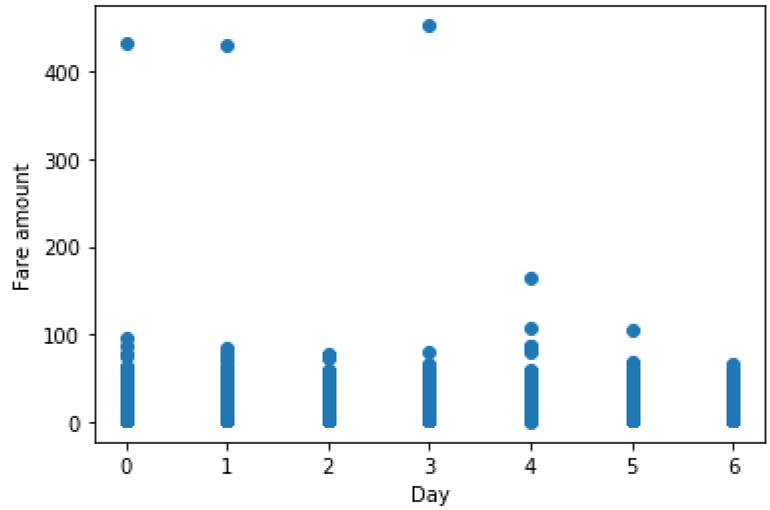
Other Visualizations

Below diagram is the correlation plot for the data



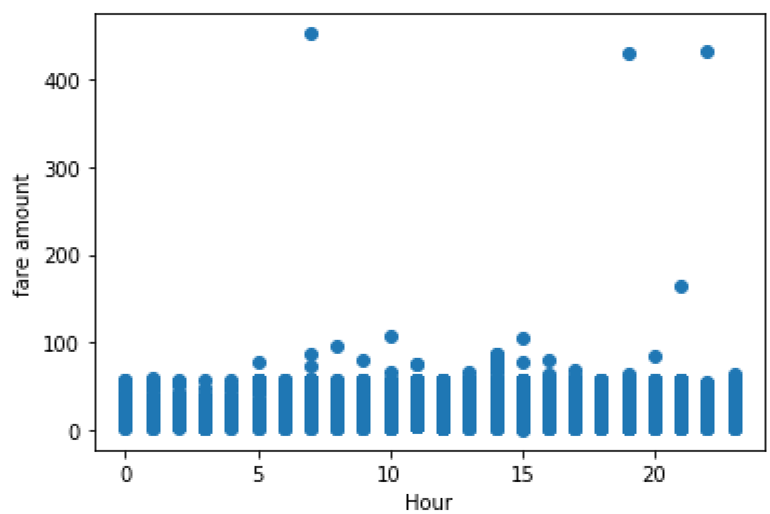
From which we can say there is no high correlative variables are there.

->On analyzing the Day vs fare_amount

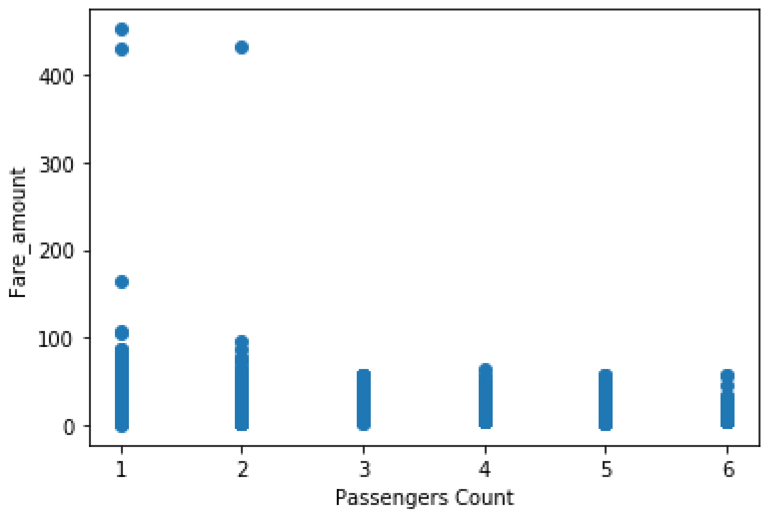


It looks like on Thursday there are high fares compare to all other days in a week

->On analyzing the hour vs fare amount it looks at 7am and 14pm price is high



->On analyzing the fare amount vs passengers count having count 1 and 2 are paying more



Python code

```
#loading the required libraries
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.tree import DecisionTreeRegressor
from sklearn.model_selection import cross_val_score
import requests
from shapely.geometry import mapping, shape
from shapely.prepared import prep
from shapely.geometry import Point
from geopy.distance import geodesic
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.neighbors import KNeighborsRegressor
from sklearn.preprocessing import OneHotEncoder,LabelEncoder
```

```
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score


#Load data files
data=pd.read_csv("C:/Users/mitta/projects/train_cab.csv")
train=pd.read_csv("C:/Users/mitta/projects/train_cab.csv")
test=pd.read_csv("C:/Users/mitta/projects/ test.csv")

#check the number of rows and columns
print(data.shape)
print(train.shape)
print(test.shape)

#Displays the first 5 rows of the train dataset
train.head()

#displays the first 5 rows of the test dataset
test.head()
```

```
train.dtypes


#Total number of missing values in each column
train.isna().sum()


"""# **Data Cleaning and Missing value Analysis**"""


#fare_amount is in object form convert into integer format
train['fare_amount']=pd.to_numeric(train['fare_amount'],errors="coerce")


#After converting fare amount into integer type one extra null value is added here.Before converting into integer
#fare_amount column has 24 null values now there are 25 null values
train.isna().sum()


#extra null value is due to the index location 1123 here fare amount value is 430- . so make it 430 and assign
data.iloc[1123]


#assign 430 to there location
train.iloc[1123,0]=430


train.isna().sum()
```

```
#pickup_datetime column is in object convert into datetime format
train['pickup_datetime']=pd.to_datetime(train["pickup_datetime"],errors="coerce")

#now one null value is added in pickup_datetime column by observation
train.isna().sum()

#finding the row which has null value in pickup_datetime
train[train['pickup_datetime'].isna()]

#Here null value is due to here pickup_datetime value is 43 whcich doesnot represent datetime
data.iloc[1327]

#so make a null value
data.iloc[1327,1]=np.nan

train.isna().sum()

# Delete all rows which contain null values
train=train.dropna(subset=["pickup_datetime","fare_amount","passenger_count"])
```

#data types of each column

train.dtypes

#Now there is no null value

train.isna().sum()

test.isna().sum()

test.dtypes

#Now we need to convert the datatypes of test dataset similar to train dataset

test["pickup_datetime"]=pd.to_datetime(test["pickup_datetime"],errors="coerce")

sns.boxplot(x=train['passenger_count'])

#In general every cab allows the passenger count is max 6

len(train[train["passenger_count"]>6])

#checking the records having passenger count>6

train[train["passenger_count"]>6]

#Remove the records which have passenger count >6

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

#passenger count should be greater that 1. Less than 1 is not possible
len(train[train["passenger_count"]<1])

#Remove the records passengers count <1
train=train.drop(train[train["passenger_count"]<1].index,axis=0)

train.shape

#on seeing this observation maximum fare amount is too large there is sudden drop off in fare amount
train.fare_amount.sort_values(ascending=False)

sns.boxplot(y=train['fare_amount'])

#Drop the records having fare_amount > 454
train=train.drop(train[train["fare_amount"]>454].index,axis=0)

len(train[train["fare_amount"]<=0])

#Fare_amount should not be <0 . So remove the records if it has values <0
train=train.drop(train[train["fare_amount"]<=0].index,axis=0)
```

```
train.shape

train.columns

"""We know that Latitudes are range in between -90 to 90
and Longitudes are range between -180 to 180. In our dataset if there are any records not between those remove that records
"""

#Remove the records which are not in actual ranges
train=train.drop(train[train["pickup_longitude"]<-180].index,axis=0)
train=train.drop(train[train["pickup_latitude"]<-90].index,axis=0)
train=train.drop(train[train["dropoff_longitude"]<-180].index,axis=0)
train=train.drop(train[train["dropoff_latitude"]<-90].index,axis=0)
train=train.drop(train[train["pickup_longitude"]>180].index,axis=0)
train=train.drop(train[train["pickup_latitude"]>90].index,axis=0)
train=train.drop(train[train["dropoff_longitude"]>180].index,axis=0)
train=train.drop(train[train["dropoff_latitude"]>90].index,axis=0)

train.columns

"""# FEATURE ENGINEERING"""
```

```
#Add more columns using pickup_datetime
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.describe()

#similarly do in test set
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.describe()

# From latitudes and longitudes we can know the distance i.e how long the passenger travel which
#is useful for finding fare amount. This distance method calculats the distance between two latitudes and longitudes
def distance(values):
```

```
  longitude_s=values[0]
  latitude_s=values[1]
  longitude_e=values[2]
  latitude_e=values[3]
  start=(latitude_s,longitude_s)
  end=(latitude_e,longitude_e)
  return geodesic(start, end).miles*1.6
```

#using the pickup and drop latitudes and longitudes calculates the distances and store into new column Distance

```
train["Distance"]=train[['pickup_longitude',
'pickup_latitude','dropoff_longitude',
'dropoff_latitude']].apply(distance,axis=1)
```

```
sns.boxplot(x=train['Distance'])
```

```
train.Distance.sort_values(ascending=False)[0:50]
```

#In distance there is sudden drop after 130 so remove those columns which are too large

```
train=train.drop(train[train["Distance"]>130].index,axis=0)
```

#Distance ==0 means they didnt travel. so remove those records

```
train=train.drop(train[train["Distance"]==0].index,axis=0)
```

```
train.shape

#Convert data types into required format
train["passenger_count"]=train["passenger_count"].astype('int64')
train["year"]=train["year"].astype('int64')
train["Month"]=train["Month"].astype('int64')
train["Date"]=train["Date"].astype('int64')
train["Day"]=train["Day"].astype('int64')
train["Hour"]=train["Hour"].astype('int64')

train.dtypes

# Also calculates the distance in test dataset using pickup oand drop off location
test["Distance"]=test[['pickup_longitude',
'pickup_latitude','dropoff_longitude',
'dropoff_latitude']].apply(distance,axis=1)

test["Distance"].describe()

#Using this we can know the all countries borders latitudes and longitudes
data = requests.get("https://raw.githubusercontent.com/datasets/geo-countries/master/data/countries.geojson").json()
```

```
countries = {}
for feature in data["features"]:
    geom = feature["geometry"]
    country = feature["properties"]["ADMIN"]
    countries[country] = prep(shape(geom))

#This method helps to find country based on the latitude and longitude
def get_country(a):
  lon=a[0]
  lat=a[1]
  point = Point(lon, lat)
  for country, geom in countries.items():
    if geom.contains(point):
      return country
  return "unknown"

# Find the Country using latitude and longitude
train["Country"]=train[['pickup_longitude',
'pickup_latitude']].apply(get_country,axis=1)

train["Country"].value_counts()
```

```
"""Few latitudes and Longitudes are belongs to antarctca"""

train[train["Country"]=='unknown']

#Also find country in test dataset
test["Country"]=test[['pickup_longitude',
'pickup_latitude']].apply(get_country,axis=1)

test["Country"].value_counts()

train["Country"].value_counts()

"""In test data there is no antarctica and we know there are no paved roads
in antarctica. So we can remove those records"""

train=train.drop(train[train["Country"]=="Antarctica"].index,axis=0)

#Remove the columns which was not used now
train=train.drop(['pickup_datetime','pickup_longitude',
'pickup_latitude','dropoff_longitude', 'dropoff_latitude'],axis=1)

train.head()

# Remove thr columns in test data also
```

```
test=test.drop(['pickup_datetime','pickup_longitude',
'pickup_latitude','dropoff_longitude', 'dropoff_latitude'],axis=1)

test.head()

"""# Data Visualization
From this we know the features which affects the fare amount
"""

sns.distplot(train["passenger_count"])

plt.scatter(train["passenger_count"],train["fare_amount"])
plt.xlabel("Passengers Count")
plt.ylabel("Fare_amount")

"""Observation:
From the above plot single and double passengers are travelled more
frequently and the fare amount also high for single and double passengers
"""

plt.scatter(train["Month"],train["fare_amount"])
plt.xlabel("Month")
plt.ylabel("fare amount")
```

```
"""The impact of fare on november is low"""

plt.scatter(train["Hour"],train["fare_amount"])
plt.xlabel("Hour")
plt.ylabel("fare amount")

"""Impact of fare on 7am and 14pm is high"""

plt.scatter(train["Day"],train["fare_amount"])
plt.xlabel("Day")
plt.ylabel("Fare amount")

corr=train.corr()
f,ax=plt.subplots()
sns.heatmap(corr,mask=np.zeros_like(corr,dtype=np.bool),cmap=sns.divergi
ng_palette(220,10,as_cmap=True),square=True,ax=ax)

"""# Feature Scaling"""

sns.distplot(train["Distance"])

"""Due to skewness is high apply log transform"""
```

```
train['Distance'] = np.log1p(train['Distance'])

sns.distplot(train["Distance"])

test['Distance'] = np.log1p(test['Distance'])

"""# Apply ML Algorithms"""

X=train.iloc[:,1:-1]
Y=train.iloc[:,0]

X.head()

Y.head()

#split the date in training set and test set
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.2,random_state=234)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
"""# Linear Regression"""

#Apply linear regression model on the training data
lr=LinearRegression()
lr.fit(X_train,Y_train)

#prdict the test data on the developed model
Y_pred=lr.predict(X_test)

plt.plot(Y_pred[1:50])

#This model is designed to find accuracy
def accuracy(Y_pred,Y):
  x=np.sqrt(mean_squared_error(Y_pred, Y))
  return x

# RMS error of the model is
print("Root mean squared error for training data "+str(accuracy(lr.predict(X_train),Y_train)))
print("Root mean squared error for test data "+str(accuracy(Y_pred,Y_test)))

"""# Decision Tree"""
```

```
#Apply decision tree for the training data
DTR=DecisionTreeRegressor()
DTR.fit(X_train,Y_train)

#Calculate RMSE of the model

print("Root mean squared error for training data
"+str(accuracy(DTR.predict(X_train),Y_train)))
print("Root mean squared error for test data
"+str(accuracy(DTR.predict(X_test),Y_test)))

"""# KNN"""

#Develop KNN for the training data
neigh = KNeighborsRegressor()

#For Knn model find the best value for n_neighbors using grid search
clf=GridSearchCV(neigh,{'n_neighbors':np.arange(2,30)})
clf.fit(X_train,Y_train)

#print best value for n_neighbors
print(clf.best_params_)
```

```
neigh = KNeighborsRegressor(n_neighbors=23)
neigh.fit(X_train, Y_train)

print("Root mean squared error for training data "+str(accuracy(neigh.predict(X_train),Y_train)))
print("Root mean squared error for test data "+str(accuracy(neigh.predict(X_test),Y_test)))

"""# Random Forest Regressor"""

#Apply Random Forest model on the training data
rf = RandomForestRegressor(random_state=0)

#Find the best values for n_estimators and mox_depth using grid search cros-validation
clf=GridSearchCV(rf,{'n_estimators':[70,80,90,100,110,120,130],'max_depth':[2,4,6,8,10]})
clf.fit(X_train,Y_train)

#print best parameters
print(clf.best_params_)

# Develop the model with best parameters obtained
rf = RandomForestRegressor(n_estimators=80,max_depth=4,random_state=0)
```

```
rf.fit(X_train,Y_train)


#RMSE of the random forest mode
print("Root mean squared error for training data
"+str(accuracy(rf.predict(X_train),Y_train)))
print("Root mean squared error for test data
"+str(accuracy(rf.predict(X_test),Y_test)))


print(rf.predict(X_train)[:5])
print(Y_train.iloc[:5])
print(train.iloc[8882])


"""# Predict of fare from given test dataset"""


test_main=test.iloc[:,0:-1]


test_main.head()


"""We apply random forest for the final model because it has low error
compare to linear regression, Decision Trees,KNN regressor"""


#predict values for final dataset
rf = RandomForestRegressor(random_state=0)
```

```
clf=GridSearchCV(rf,{'n_estimators':[70,80,90,100,110,120],'max_depth':[2,4,6,8,10]})

clf.fit(train.iloc[:,1:-1],train.iloc[:,0])

clf.best_params_

rf = RandomForestRegressor(n_estimators=clf.best_params_['n_estimators'],max_depth=clf.best_params_['max_depth'],random_state=0)

rf.fit(X_train,Y_train)

final_predict=rf.predict(test_main)


test=test.iloc[:,:-1]


test['fare_amount']=final_predict


test.head()


test.to_csv("test.csv",index=False)


fg=pd.read_csv("/content/test.csv")


fg.head()
```

-------------------------R Code----------------------

```
rm(list=ls())
#-----------  set working directory  ------------------

setwd("C:/Users/mitta/Desktop/project")
getwd()

# ---  install and load required libraries  ---------------

x=c('lubridate','tidyverse','caret',"Metrics",'rpart','randomForest')
install.packages(x)
lapply(x, require, character.only = TRUE)
```

```
#------------ Load train and test data  ----------------

train=read.csv("./train_cab/train_cab.csv",na.strings = c('NA',""))
#Here we use na strings because in the data there are balnk cells which we should treat as NA
test=read.csv("./test/test.csv")

train=data.frame(train)

head(train)
head(test)

#--------  find any missing values in the data -----------
missing_val=data.frame(apply(train,2,function(x){sum(is.na(x))}))
print(missing_val)

#percentage of missing values
percentage_missing=(sum(missing_val[1])/nrow(train))*100
print(percentage_missing)

#percentage missing is less than 1%
#so remove the records contain null
train=train[complete.cases(train), ]
```

```
# convert fare amount into integer as it is in factor first convert into character then numeric

train$fare_amount=as.numeric(as.character(train$fare_amount))

# Find any missing values after conversion

missing_val=data.frame(apply(train,2,function(x){sum(is.na(x))}))

print(missing_val)

#one value set as NA after conversion due to coersion (430-) so set modified value

train[1124,1]=430

train=train[complete.cases(train), ]

missing_val=data.frame(apply(train,2,function(x){sum(is.na(x))}))

print(missing_val)

# in test data there are no null values

missing_val=data.frame(apply(test,2,function(x){sum(is.na(x))}))

print(missing_val)

# In general passenger count >6 and <1 are considered as outliers remove them

train=train[which(!train$passenger_count<1),]

train=train[which(!train$passenger_count>6),]
```

```
ggplot(train,aes_string(x=train$Month,y=train$fare_amount))+geom_point(
size=4)+theme_bw()+scale_y_continuous(breaks = pretty(500))

#-- fare_amount is too large in few records and less than 0 , remove those
records
train=train[which(!train$fare_amount>454),]
train=train[which(!train$fare_amount<=0),]


# -----   feature extraction------
# using pickup_datetime feature extract the additional features
# first convert the feature into datetime format later extract new features
train$pickup_datetime =gsub(" UTC","",train$pickup_datetime)

train$Date <- as.Date(train$pickup_datetime)
train$Year <- substr(as.character(train$Date),1,4)
train$Month <- substr(as.character(train$Date),6,7)
train$Weekday <- weekdays(as.POSIXct(train$Date), abbreviate = F)
train$Hour=substr(as.character(train$pickup_datetime),12,13)

train$Month=as.integer(train$Month)
train$Year=as.integer(train$Year)
train$Hour=as.integer(train$Hour)
train$Date=NULL
```

similarly do it on the test data

```
test$pickup_datetime =gsub(" UTC","",test$pickup_datetime)

test$Date <- as.Date(test$pickup_datetime)
test$Year <- substr(as.character(test$Date),1,4)
test$Month <- substr(as.character(test$Date),6,7)
test$Weekday <- weekdays(as.POSIXct(test$Date), abbreviate = F)
test$Hour=substr(as.character(test$pickup_datetime),12,13)

test$Month=as.integer(test$Month)
test$Year=as.integer(test$Year)
test$Hour=as.integer(test$Hour)
test$Date=NULL
```

now using pickup and dropoff latitudes and longitudes details calculate distance

```
lat1 = train['pickup_latitude']
lat2 = train['dropoff_latitude']
long1 = train['pickup_longitude']
long2 = train['dropoff_longitude']

lat1=lat1/57.29577951
```

```
lat2=lat2/57.29577951

long1=long1/57.29577951
long2=long2/57.29577951

dlon = long2 - long1
dlat = lat2 - lat1
a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2
c = 2 * asin(sqrt(a))
r=c*6371

train$distance=r$dropoff_latitude
summary(train$distance)

# now on seeing the data there is sudden change in distance values
#upto 130 that is reasonable after that it looks like outliers, remove them
train=train[which(!train$distance>130),]
train=train[which(!train$distance==0),]
train=train[which(!train$distance<0),]

# similarly do it on the test data

lat1 = test['pickup_latitude']
lat2 = test['dropoff_latitude']
```

```
long1 = test['pickup_longitude']

long2 = test['dropoff_longitude']

lat1=lat1/57.29577951

lat2=lat2/57.29577951

long1=long1/57.29577951

long2=long2/57.29577951

dlon = long2 - long1

dlat = lat2 - lat1

a = sin(dlat / 2)**2 + cos(lat1) * cos(lat2) * sin(dlon / 2)**2

c = 2 * asin(sqrt(a))

r=c*6371

test$distance=r$dropoff_latitude

summary(test$distance)

# now remove all unwanted columns for training the data

train=subset(train,select=-
c(pickup_datetime,pickup_longitude,pickup_latitude,dropoff_longitude,dro
poff_latitude))

test=subset(test,select=-
c(pickup_datetime,pickup_longitude,pickup_latitude,dropoff_longitude,dro
poff_latitude))
```

```
train=as.data.frame(train)

train=train[complete.cases(train), ]


# find out if there is any skewness in the data

ggplot(train,aes_string(x=train$Month))+geom_histogram(fill="cornsilk",colour="black")+
  geom_density()+theme_bw()+xlab(" Month")

#in month data it is uniform

ggplot(train,aes_string(x=train$Year))+geom_histogram(fill="cornsilk",bins=15,colour="black")+
  geom_density()+theme_bw()+xlab(" Year")

# in year also it is not skewed much

ggplot(train,aes_string(x=train$distance))+geom_histogram(fill="cornsilk",colour="black")+
geom_density()+theme_bw()+xlab(" distance")

# but in distance values it is skewed much so apply log transform to normalize


train$distance=log1p(train$distance)


ggplot(train,aes_string(x=train$distance))+geom_histogram(fill="cornsilk",colour="black")+
  geom_density()+theme_bw()+xlab(" distance")

#Now skeawness decreases much
```

Now convert weekday feature factor into numerical using technique label encoder

train$Weekday=factor(train$Weekday,labels=(1:length(levels(factor(train$Weekday)))))

test$Weekday=factor(test$Weekday,labels=(1:length(levels(factor(test$Weekday)))))

prepare data for training and testing

X=train[1:1200,]

X_test=train[1201:nrow(train),]

#linear regression

X_test=train[1201:nrow(train),]

lm_model=lm(fare_amount~.,data=X)

summary(lm_model)

from the summary of the model year is the most important feature

predictions_LR=predict(lm_model,X_test[,2:7])

rmse(predictions_LR,X_test$fare_amount)

```r
mape(predictions_LR,X_test$fare_amount)
mae(predictions_LR,X_test$fare_amount)



# Decision Tree
DT=rpart(fare_amount~.,data=X)
predictions_DT=predict(DT,X_test[,2:7])
rmse(predictions_DT,X_test$fare_amount)
mape(predictions_DT,X_test$fare_amount)
mae(predictions_DT,X_test$fare_amount)



#Random forest

rf=randomForest(fare_amount~.,data=X)
predictions_rf=predict(rf,X_test[,2:7])
rmse(predictions_rf,X_test$fare_amount)
mape(predictions_rf,X_test$fare_amount)
mae(predictions_rf,X_test$fare_amount)

# cross validation of ntree for random forest
for (i in c(70,80,90,100,120,130,140,160,180,200,220,300,400,500)){
  rf=randomForest(fare_amount~.,data=X,ntree=i)
```

```
  print("For ntree=")
  print(i)
  print(rmse(predict(rf,X_test[,2:7]),X_test$fare_amount))
  print(mape(predict(rf,X_test[,2:7]),X_test$fare_amount))
  print(mae(predict(rf,X_test[,2:7]),X_test$fare_amount))
}

# from the observations ntree =90 is the best case


# ----------Apply Randomforest on final data -------------------
rf=randomForest(fare_amount~.,data=train,ntree=90)
final_predict_test=as.data.frame(predict(rf,test))

test$fare_amount=final_predict_test[1]
write.csv(test,"new_test.csv",row.names=T)
```

---END---