#Predict the price of the Uber ride from a given pickup point to the agreed drop-off location. Perform following tasks:

1. Pre-process the dataset.
2. Identify outliers.
3. Check the correlation.
4. Implement linear regression and random forest regression models.
5. Evaluate the models and compare their respective scores like R2, RMSE, etc. Dataset link: https://www.kaggle.com/datasets/yasserh/uber-fares-dataset (https://www.kaggle.com/datasets/yasserh/uber-fares-dataset)

In [71]:
```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

In [72]:
```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

In [73]:
```
df.head()
```

...

In [74]:
```
df.info() #To get the required information of the dataset
```

...

In [75]:
```
df.columns #TO get number of columns in the dataset
```

...

In [76]:
```
df = df.drop(['Unnamed: 0', 'key'], axis= 1) #To drop unnamed column as it isn't
```

In [77]:
```
df.head()
```

...

In [78]:
```
df.shape #To get the total (Rows,Columns)
```

...

In [79]:
```
df.dtypes #To get the type of each column
```

...

In [80]:
```
df.info()
```

...

In [81]:
```
df.describe() #To get statistics of each columns
```

...

Filling Missing values

In [82]:
```
df.isnull().sum()
```

...

In [83]:
```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = T
```

In [84]:
```
df.isnull().sum()
```

...

In [85]:
```
df.dtypes
```

...

Column pickup_datetime is in wrong format (Object). Convert it to DateTime Format

In [86]:
```
df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

In [87]:
```
df.dtypes
```

...

To segregate each time of date and time

In [88]:
```
df= df.assign(hour = df.pickup_datetime.dt.hour,
              day= df.pickup_datetime.dt.day,
              month = df.pickup_datetime.dt.month,
              year = df.pickup_datetime.dt.year,
              dayofweek = df.pickup_datetime.dt.dayofweek)
```

In [89]:
```
df.head()
```

...

In [90]:
```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

df = df.drop('pickup_datetime',axis=1)
```

In [91]:
```
df.head()
```

...

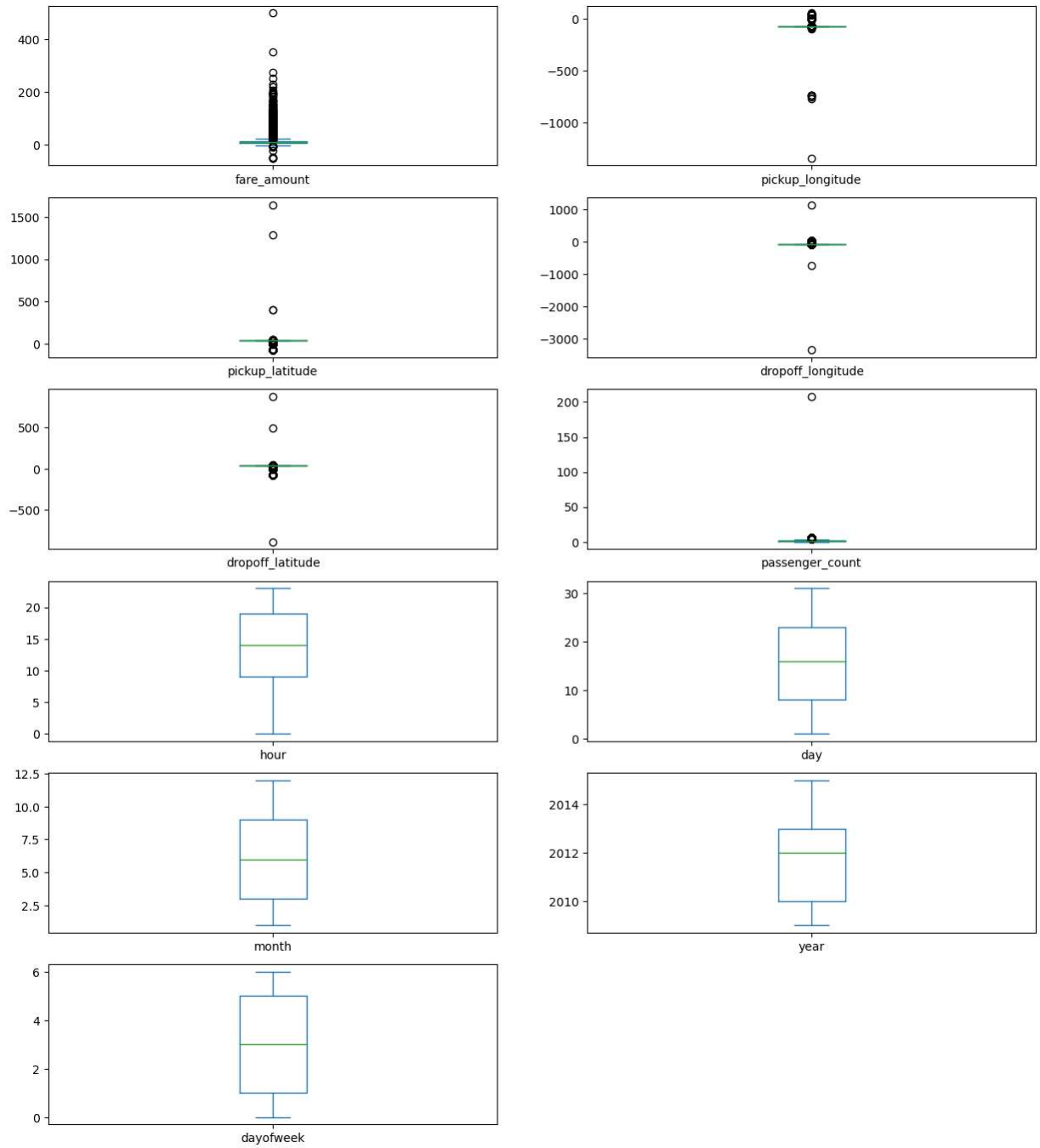
In [92]:
```
df.dtypes
```

...

Checking outliers and filling them

In [93]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to
```

Out[93]:
```
fare_amount            AxesSubplot(0.125,0.786098;0.352273x0.0939024)
pickup_longitude     AxesSubplot(0.547727,0.786098;0.352273x0.0939024)
pickup_latitude        AxesSubplot(0.125,0.673415;0.352273x0.0939024)
dropoff_longitude    AxesSubplot(0.547727,0.673415;0.352273x0.0939024)
dropoff_latitude       AxesSubplot(0.125,0.560732;0.352273x0.0939024)
passenger_count      AxesSubplot(0.547727,0.560732;0.352273x0.0939024)
hour                   AxesSubplot(0.125,0.448049;0.352273x0.0939024)
day                  AxesSubplot(0.547727,0.448049;0.352273x0.0939024)
month                  AxesSubplot(0.125,0.335366;0.352273x0.0939024)
year                 AxesSubplot(0.547727,0.335366;0.352273x0.0939024)
dayofweek              AxesSubplot(0.125,0.222683;0.352273x0.0939024)
dtype: object
```



In [94]:
```
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

def treat_outliers_all(df1 , col_list):
    for c in col_list:
        df1 = remove_outlier(df , c)
    return df1
```

In [95]:
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```

In [96]:
```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot sho
```

...

In [97]:
```
pip install haversine
```

```
Requirement already satisfied: haversine in d:\anaconda\lib\site-packages (2.7.
0)
Note: you may need to restart the kernel to use updated packages.
```

In [98]:
```
import haversine as hs  #Calculate the distance using Haversine to calculate the
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_latitud
        loc1=(lati1,long1)
        loc2=(lati2,long2)
        c = hs.haversine(loc1,loc2)
        travel_dist.append(c)

print(travel_dist)
df['dist_travel_km'] = travel_dist
df.head()
```

...

In [99]:
```
#Uber doesn't travel over 130 kms so minimize the distance
df= df.loc[(df.dist_travel_km >= 1) | (df.dist_travel_km <= 130)]
print("Remaining observastions in the dataset:", df.shape)
```

```
Remaining observastions in the dataset: (200000, 12)
```

In [100]:
```
#Finding inccorect latitude (Less than or greater than 90) and longitude (greater
incorrect_coordinates = df.loc[(df.pickup_latitude > 90) |(df.pickup_latitude < -
                               (df.dropoff_latitude > 90) |(df.dropoff_latitu
                               (df.pickup_longitude > 180) |(df.pickup_longit
                               (df.dropoff_longitude > 90) |(df.dropoff_longi
                                ]
```

In [101]:
```
df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

In [102]:
```
df.head()
```

...

In [103]:
```
df.isnull().sum()
```

...

In [104]:
```
sns.heatmap(df.isnull()) #Free for null values
```

...

In [105]:
```
corr = df.corr() #Function to find the correlation
```

In [106]:
```
corr
```

...

In [107]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means high
```

...

Dividing the dataset into feature and target values

In [108]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitud
```

In [109]:
```
y = df['fare_amount']
```

Dividing the dataset into training and testing dataset

In [110]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)
```

Linear Regression

In [111]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [112]:
```
regression.fit(X_train,y_train)
```

...

In [113]:
```
regression.intercept_ #To find the linear intercept
```

...

In [114]:
```
regression.coef_ #To find the linear coeeficient
```

...

In [115]:
```
prediction = regression.predict(X_test) #To predict the target values
```

In [116]:
```
print(prediction)
```

```
[ 5.01523896  4.90594076 10.69941484 ... 13.36805853  7.0729064
  6.40670012]
```

In [117]:
```
y_test
```

...

Metrics Evaluation using R2, Mean Squared Error, Root Mean Sqared Error

In [118]:
```
from sklearn.metrics import r2_score
```

In [119]:
```
r2_score(y_test,prediction)
```

Out[119]:
```
0.662655860609985
```

In [120]:
```
from sklearn.metrics import mean_squared_error
```

In [121]:
```
MSE = mean_squared_error(y_test,prediction)
```

In [122]:
```
MSE
```

Out[122]:
```
10.02348655600728
```

In [123]:
```
RMSE = np.sqrt(MSE)
```

In [124]:
```
RMSE
```

Out[124]:
```
3.1659890328311753
```

Random Forest Regression

In [125]:
```
from sklearn.ensemble import RandomForestRegressor
```

In [126]:
```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of t
```

In [127]:
```
rf.fit(X_train,y_train)
```

Out[127]:
```
RandomForestRegressor()
```

In [128]:
```
y_pred = rf.predict(X_test)
```

In [129]:
```
y_pred
```

Out[129]:
```
array([ 5.433 ,  4.4   , 10.8135, ..., 22.2365,  4.651 ,  6.384 ])
```

Metrics evaluatin for Random Forest

In [130]:
```
R2_Random = r2_score(y_test,y_pred)
```

In [131]:
```
R2_Random
```

Out[131]:
```
0.7944206535817593
```

In [132]:
```
MSE_Random = mean_squared_error(y_test,y_pred)
```

In [133]:
```
MSE_Random
```

Out[133]:
```
6.108367018742377
```

In [134]:
```
RMSE_Random = np.sqrt(MSE_Random)
```

In [135]:
```
RMSE_Random
```

Out[135]:
```
2.4715110800363362
```