#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

```
#Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
from google.colab import files
uploaded=files.upload()
```

```
Choose Files No file chosen    Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving uber.csv to uber.csv
```

```
#importing the dataset
df  = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

```
df.head()
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latit
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740

```
df.info() #To get the required information of the dataset
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 9 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   Unnamed: 0         200000 non-null  int64
 1   key                200000 non-null  object
```

```
 2   fare_amount        200000 non-null  float64
 3   pickup_datetime    200000 non-null  object
 4   pickup_longitude   200000 non-null  float64
 5   pickup_latitude    200000 non-null  float64
 6   dropoff_longitude  199999 non-null  float64
 7   dropoff_latitude   199999 non-null  float64
 8   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(2), object(2)
memory usage: 13.7+ MB
```

```
df.columns #TO get number of columns in the dataset
```

```
Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

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

```
df.head()
```

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropo
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	
		2009-06-26				

```
df.shape #To get the total (Rows,Columns)
```

```
(200000, 7)
```

```
df.dtypes #To get the type of each column
```

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 7 columns):
 #   Column             Non-Null Count   Dtype
---  ------             --------------   -----
 0   fare_amount        200000 non-null  float64
 1   pickup_datetime    200000 non-null  object
 2   pickup_longitude   200000 non-null  float64
 3   pickup_latitude    200000 non-null  float64
```

```
 4   dropoff_longitude  199999 non-null  float64
 5   dropoff_latitude   199999 non-null  float64
 6   passenger_count    200000 non-null  int64
dtypes: float64(5), int64(1), object(1)
memory usage: 10.7+ MB
```

```
df.describe() #To get statistics of each columns
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890
std	9.901776	11.437787	7.720539	13.117408	6.794829
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042
75%	12.500000	-73.967154	40.767158	-73.963658	40.768001
max	499.000000	57.418457	1644.421482	1153.572603	872.697628

Filling Missing values

```
df.isnull().sum()
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

```
df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)
```

```
df.isnull().sum()
```

```
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
dtype: int64
```

```
df.dtypes
```

```
fare_amount          float64
pickup_datetime       object
pickup_longitude     float64
pickup_latitude      float64
dropoff_longitude    float64
```

```
dropoff_latitude     float64
passenger_count        int64
dtype: object
```

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

[ ] ↳ *2 cells hidden*

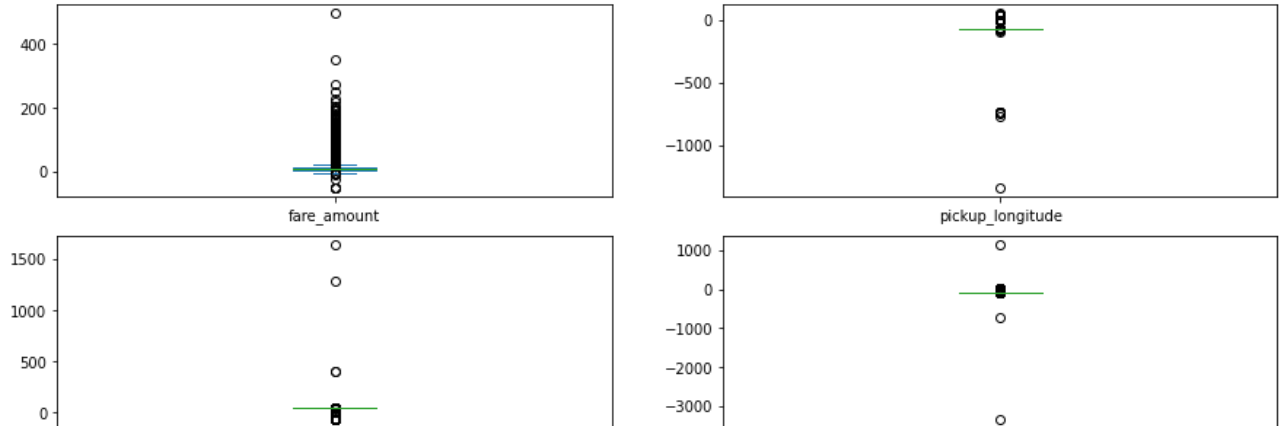
▸ To segregate each time of date and time

[ ] ↳ *5 cells hidden*

▾ Checking outliers and filling them

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to check the outliers
```

```
fare_amount            AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude        AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude       AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                   AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                  AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek              AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```



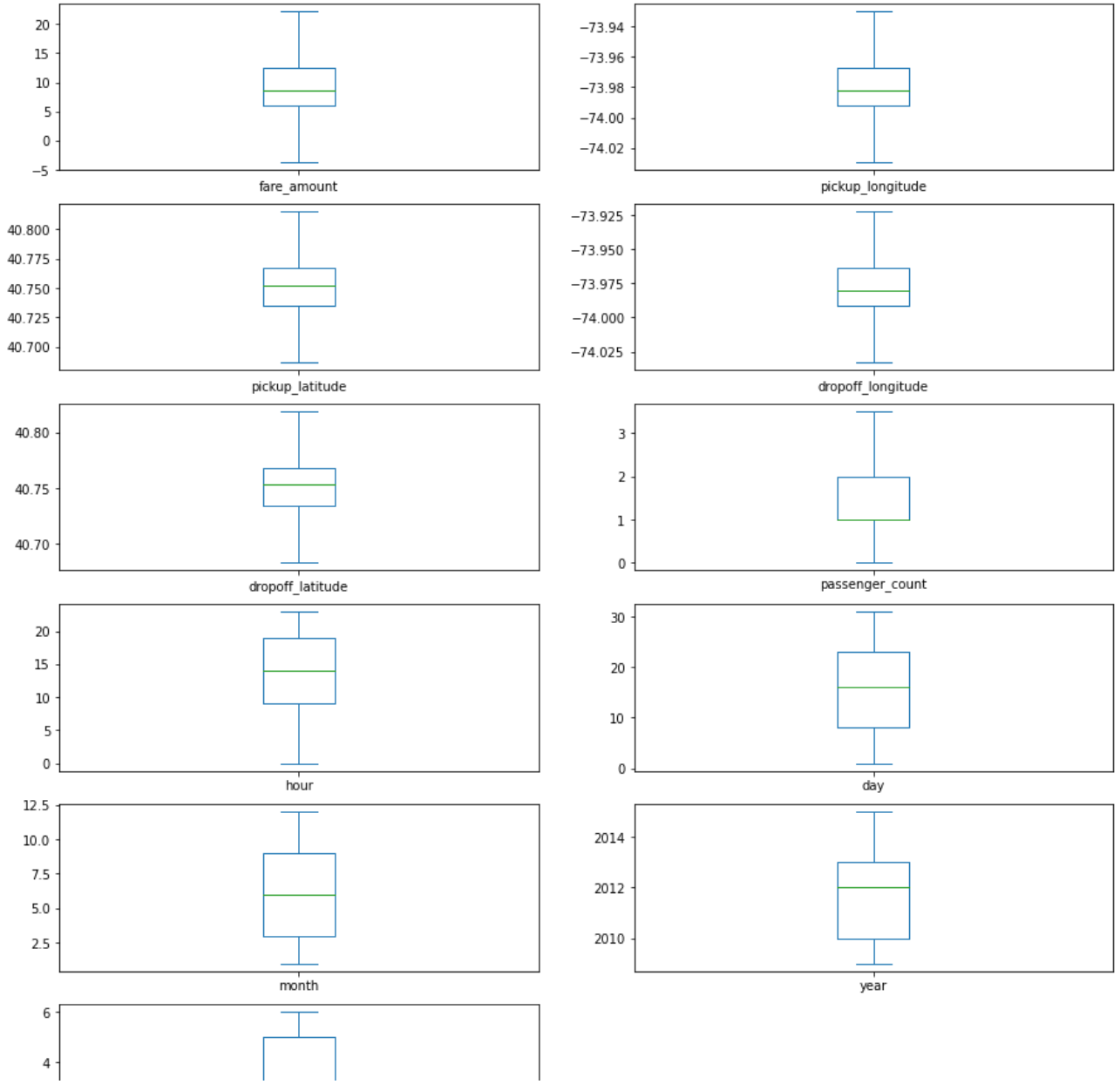

```
#Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return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

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

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

```
df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot shows that dataset is
```

```
fare_amount            AxesSubplot(0.125,0.787927;0.352273x0.0920732)
pickup_longitude     AxesSubplot(0.547727,0.787927;0.352273x0.0920732)
pickup_latitude        AxesSubplot(0.125,0.677439;0.352273x0.0920732)
dropoff_longitude    AxesSubplot(0.547727,0.677439;0.352273x0.0920732)
dropoff_latitude       AxesSubplot(0.125,0.566951;0.352273x0.0920732)
passenger_count      AxesSubplot(0.547727,0.566951;0.352273x0.0920732)
hour                   AxesSubplot(0.125,0.456463;0.352273x0.0920732)
day                  AxesSubplot(0.547727,0.456463;0.352273x0.0920732)
month                  AxesSubplot(0.125,0.345976;0.352273x0.0920732)
year                 AxesSubplot(0.547727,0.345976;0.352273x0.0920732)
dayofweek              AxesSubplot(0.125,0.235488;0.352273x0.0920732)
dtype: object
```




```
pip install haversine
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/sim
Collecting haversine
  Downloading haversine-2.6.0-py2.py3-none-any.whl (6.8 kB)
Installing collected packages: haversine
Successfully installed haversine-2.6.0
```

```
import haversine as hs
```

```
#pip install haversine
  #Calculate the distance using Haversine to calculate the distance between to points. Can't use Euc
travel_dist = []
for pos in range(len(df['pickup_longitude'])):
        long1,lati1,long2,lati2 = [df['pickup_longitude'][pos],df['pickup_latitude'][pos],df['dropo
        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()
```

```
IOPub data rate exceeded.
The notebook server will temporarily stop sending output
to the client in order to avoid crashing it.
To change this limit, set the config variable
`--NotebookApp.iopub_data_rate_limit`.

Current values:
NotebookApp.iopub_data_rate_limit=1000000.0 (bytes/sec)
NotebookApp.rate_limit_window=3.0 (secs)
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pass
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.929786	40.744085	-73.973082	40.761247	

```
#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)
```

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


df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')


df.head()
```

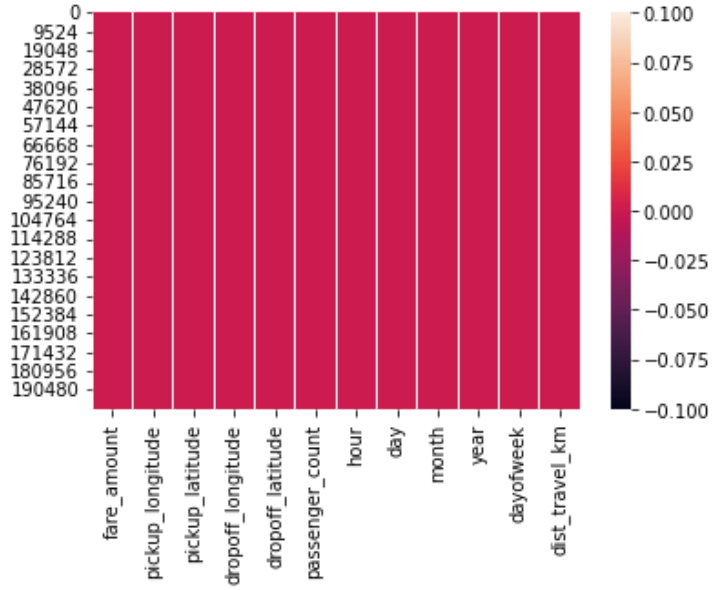
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pass
0	7.5	-73.999817	40.738354	-73.999512	40.723217	

```
df.isnull().sum()
```

```
fare_amount          0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    0
dropoff_latitude     0
passenger_count      0
hour                 0
day                  0
month                0
year                 0
dayofweek            0
dist_travel_km       0
dtype: int64
```

```
sns.heatmap(df.isnull()) #Free for null values
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0dea1ca710>
```



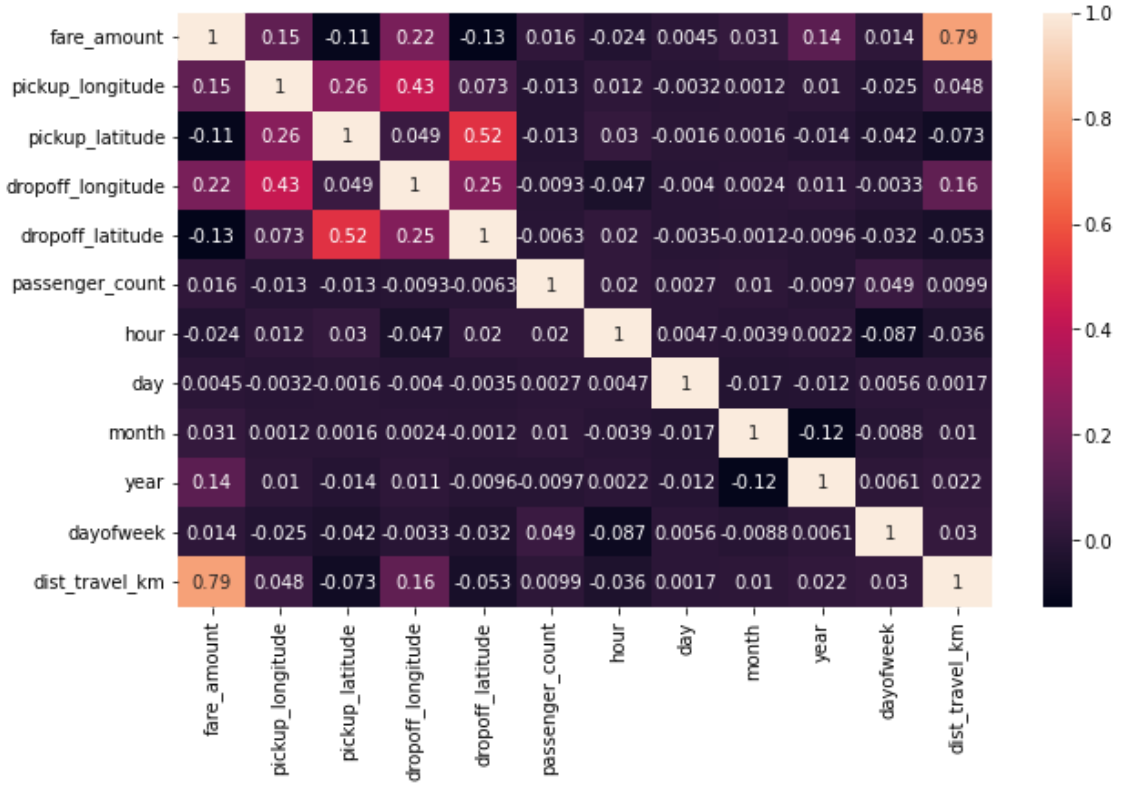

```
corr = df.corr() #Function to find the correlation
```

```
corr
```

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_l
fare_amount	1.000000	0.154069	-0.110842	0.218675	-(
pickup_longitude	0.154069	1.000000	0.259497	0.425619	(
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	(
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	(

```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f0dea1ca590>
```



Dividing the dataset into feature and target values

[ ] ↳ *2 cells hidden*

Dividing the dataset into training and testing dataset

[ ] ↳ *1 cell hidden*

Linear Regression

[ ] ↳ *7 cells hidden*

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

[ ] ↳ *7 cells hidden*

Random Forest Regression

[ ] ↳ *5 cells hidden*

Metrics evaluatin for Random Forest

[ ] ↳ *6 cells hidden*