

```
In [1]: #Importing the required libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
```

```
In [2]: #importing the dataset
df = pd.read_csv("uber.csv")
```

1. Pre-process the dataset.

```
In [3]: df.head()
```

```
Out[3]:
```

	Unnamed: 0	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude
0	24238194	2015-05-07 19:52:06.0000003	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354
1	27835199	2009-07-17 20:04:56.0000002	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225
2	44984355	2009-08-24 21:45:00.00000061	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085

```
In [4]: 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
```

In [5]: `df.columns` *#To get number of columns in the dataset*

Out[5]: Index(['Unnamed: 0', 'key', 'fare_amount', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude', 'passenger_count'], dtype='object')

In [6]: `df = df.drop(['Unnamed: 0', 'key'], axis=1)` *#To drop unnamed column as it isn't*

In [7]: `df.head()`

Out[7]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.7232
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.7503
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.7726
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.8033
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.7612

In [8]: `df.shape` *#To get the total (Rows,Columns)*

Out[8]: (200000, 7)

In [9]: `df.dtypes` *#To get the type of each column*

Out[9]:

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

In [10]: `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
```

In [11]: `df.describe()` *#To get statistics of each columns*

Out[11]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passen
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	2000
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	2

Filling Missing values

In [12]: `df.isnull().sum()`

Out[12]:

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

In [13]: `df['dropoff_latitude'].fillna(value=df['dropoff_latitude'].mean(),inplace = True)`
`df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = True)`

```
In [14]: df.isnull().sum()
```

```
Out[14]: fare_amount      0
pickup_datetime      0
pickup_longitude      0
pickup_latitude      0
dropoff_longitude      0
dropoff_latitude      0
passenger_count      0
dtype: int64
```

```
In [15]: df.dtypes
```

```
Out[15]: 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

```
In [16]: df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce')
```

```
In [17]: df.dtypes
```

```
Out[17]: fare_amount      float64
pickup_datetime      datetime64[ns, UTC]
pickup_longitude      float64
pickup_latitude      float64
dropoff_longitude      float64
dropoff_latitude      float64
passenger_count      int64
dtype: object
```

To segregate each time of date and time

```
In [18]: 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 [19]: `df.head()`

Out[19]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
0	7.5	2015-05-07 19:52:06+00:00	-73.999817	40.738354	-73.999512	40.7232
1	7.7	2009-07-17 20:04:56+00:00	-73.994355	40.728225	-73.994710	40.7503
2	12.9	2009-08-24 21:45:00+00:00	-74.005043	40.740770	-73.962565	40.7726
3	5.3	2009-06-26 08:22:21+00:00	-73.976124	40.790844	-73.965316	40.8033
4	16.0	2014-08-28 17:47:00+00:00	-73.925023	40.744085	-73.973082	40.7612

In [20]: `# drop the column 'pickup_datetime' using drop()
'axis = 1' drops the specified column

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

In [21]: `df.head()`

Out[21]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
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	-73.925023	40.744085	-73.973082	40.761247	

In [22]: `df.dtypes`

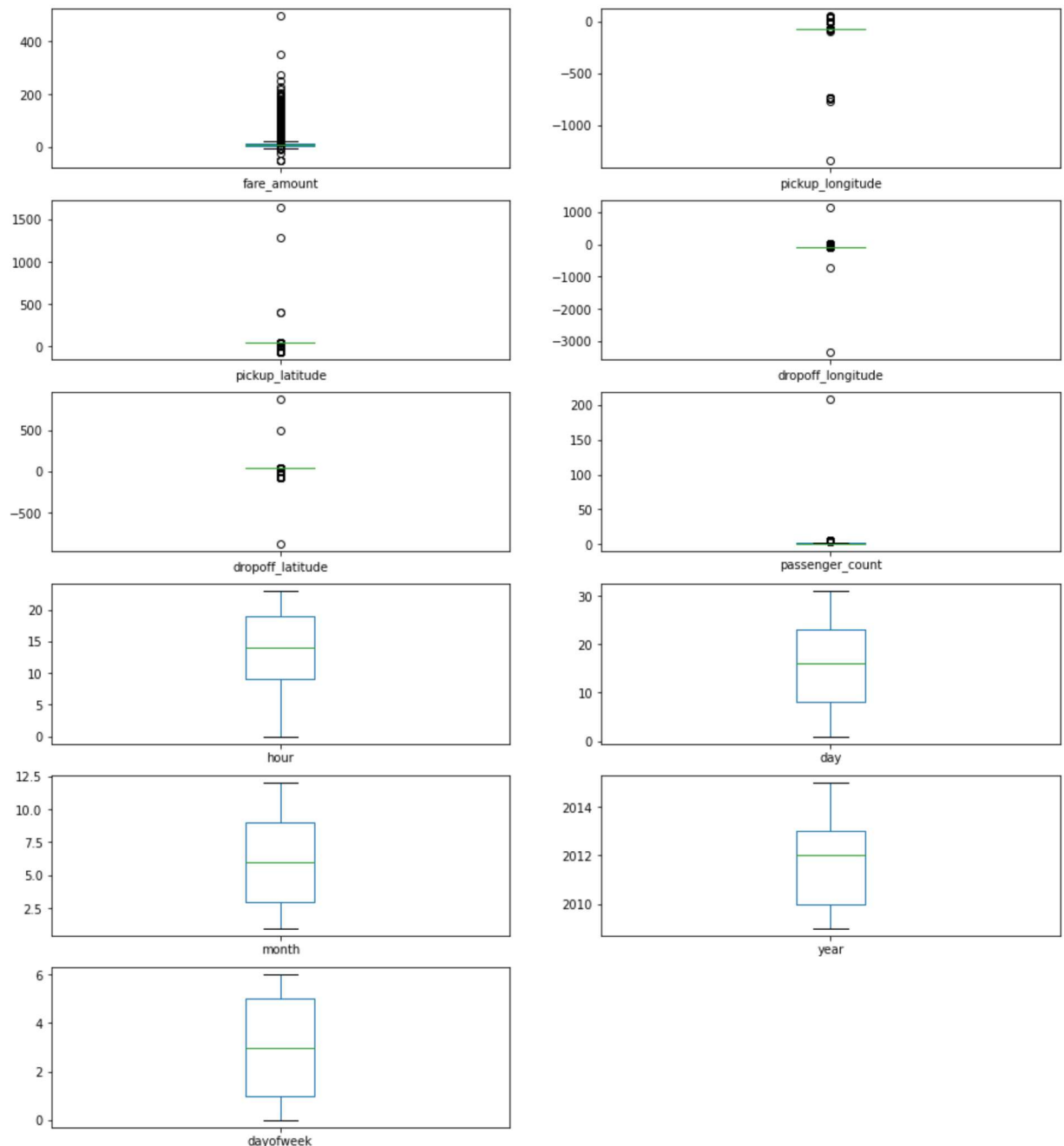
Out[22]:

fare_amount	float64
pickup_longitude	float64
pickup_latitude	float64
dropoff_longitude	float64
dropoff_latitude	float64
passenger_count	int64
hour	int64
day	int64
month	int64
year	int64
dayofweek	int64
dtype:	object

Checking outliers and filling them

```
In [23]: df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot to
```

```
Out[23]: 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
```



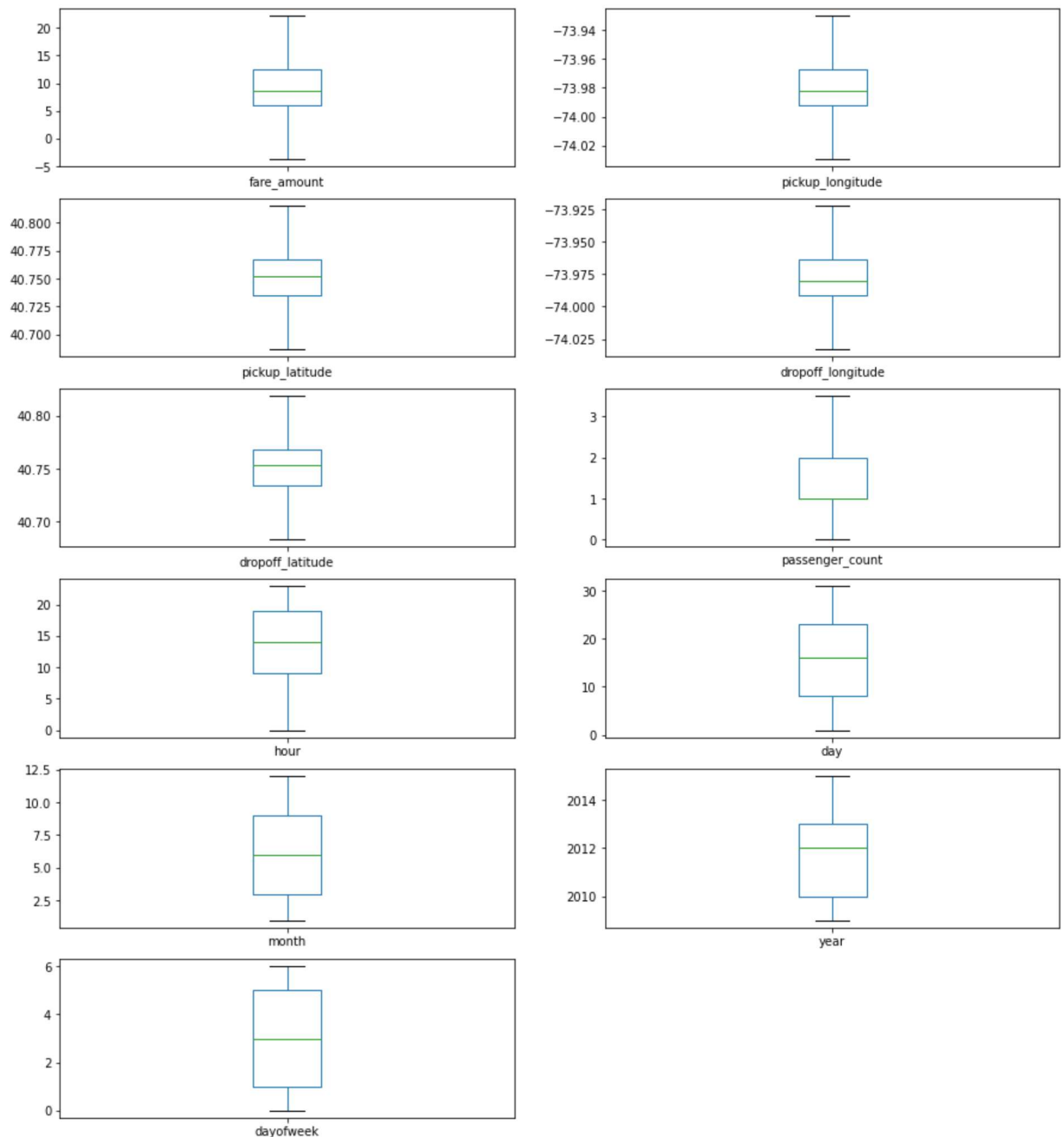
```
In [24]: #Using the InterQuartile Range to fill the values
def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    df[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

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

```
In [25]: df = treat_outliers_all(df , df.iloc[:, 0::])
```

```
In [26]: df.plot(kind = "box",subplots = True,layout = (7,2),figsize=(15,20)) #Boxplot sho
```

```
Out[26]: 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
```




```

In [27]: #pip install haversine
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_latitude'][pos],df['dropoff_longitude'][pos],df['dropoff_latitude'][pos]]
    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)

```

Out[27]:

```

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

```
In [28]: #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 [29]: #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_latitude < -
                                (df.pickup_longitude > 180) |(df.pickup_longitude < -
                                (df.dropoff_longitude > 90) |(df.dropoff_longitude < -
                                ]
```

```
In [30]: df.drop(incorrect_coordinates, inplace = True, errors = 'ignore')
```

```
In [31]: df.head()
```

```
Out[31]:
```

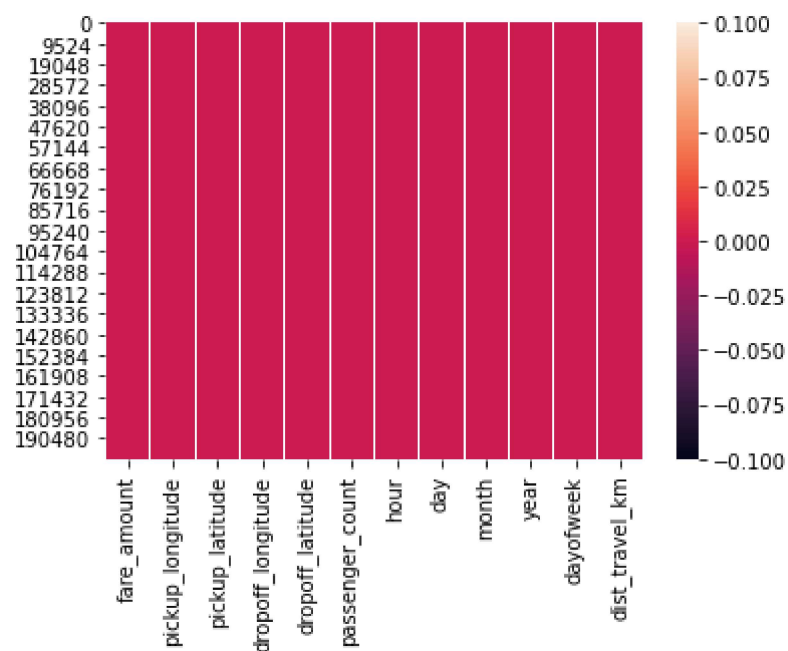
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.929786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

```
In [32]: df.isnull().sum()
```

```
Out[32]: 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
```

```
In [33]: sns.heatmap(df.isnull()) #Free for null values
```

```
Out[33]: <AxesSubplot:>
```



```
In [34]: corr = df.corr() #Function to find the correlation
```

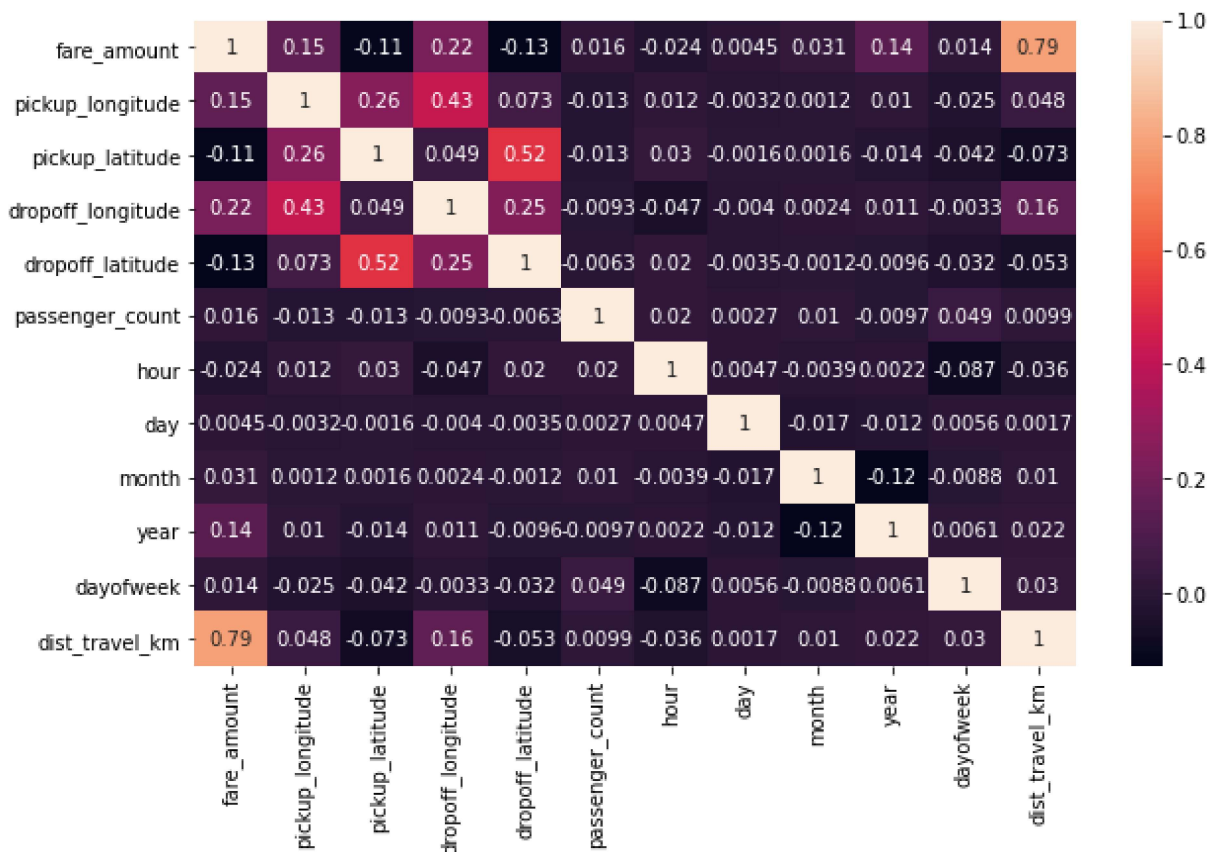
In [35]: corr

Out[35]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude
fare_amount	1.000000	0.154069	-0.110842	0.218675	-0.12589
pickup_longitude	0.154069	1.000000	0.259497	0.425619	0.07329
pickup_latitude	-0.110842	0.259497	1.000000	0.048889	0.51571
dropoff_longitude	0.218675	0.425619	0.048889	1.000000	0.24566
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	1.00000
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	-0.00630
hour	-0.023623	0.011579	0.029681	-0.046558	0.01978
day	0.004534	-0.003204	-0.001553	-0.004007	-0.00347
month	0.030817	0.001169	0.001562	0.002391	-0.00119
year	0.141277	0.010198	-0.014243	0.011346	-0.00960
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	-0.03191
dist_travel_km	0.786385	0.048446	-0.073362	0.155191	-0.05270

In [36]: fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True) *#Correlation Heatmap (Light values means high*

Out[36]: <AxesSubplot:>




```
In [190]: y_test
```

```
Out[190]: 155740      4.90
          47070      10.00
          116192     14.50
          164589      6.50
          154309     11.30
          ...
          76552      7.70
          27926      10.90
          38972      6.50
          120341     22.25
          178449      8.10
          Name: fare_amount, Length: 66000, dtype: float64
```

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

```
In [191]: from sklearn.metrics import r2_score
```

```
In [192]: r2_score(y_test, prediction)
```

```
Out[192]: 0.6651880468683617
```

```
In [193]: from sklearn.metrics import mean_squared_error
```

```
In [194]: MSE = mean_squared_error(y_test, prediction)
```

```
In [195]: MSE
```

```
Out[195]: 9.961516917717704
```

```
In [196]: RMSE = np.sqrt(MSE)
```

```
In [197]: RMSE
```

```
Out[197]: 3.156187085348032
```

Random Forest Regression

```
In [198]: from sklearn.ensemble import RandomForestRegressor
```

```
In [199]: rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of t
```

```
In [200]: rf.fit(X_train, y_train)
```

```
Out[200]: RandomForestRegressor()
```

```
In [201]: y_pred = rf.predict(X_test)
```

```
In [202]: y_pred
```

```
Out[202]: array([ 5.714 , 10.285 , 12.68  , ...,  6.338 , 19.4685,  7.712  ])
```

Metrics evaluatin for Random Forest

```
In [210]: R2_Random = r2_score(y_test,y_pred)
```

```
In [211]: R2_Random
```

```
Out[211]: 0.7948374920410631
```

```
In [205]: MSE_Random = mean_squared_error(y_test,y_pred)
```

```
In [206]: MSE_Random
```

```
Out[206]: 6.104112397417331
```

```
In [207]: RMSE_Random = np.sqrt(MSE_Random)
```

```
In [208]: RMSE_Random
```

```
Out[208]: 2.4706501972997574
```