Vinit Wadgaonkar ¶

#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 [23]:

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

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

##  1.  Pre-process the dataset.
```

In [24]:

```
df.head()
```

Out[24]:

	Unnamed: 0	Unnamed: 1	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	24238194.0	24238194	52:06.0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	27835199.0	27835199	04:56.0	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	44984355.0	44984355	45:00.0	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	25894730.0	25894730	22:21.0	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	17610152.0	17610152	47:00.0	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

In [25]:

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

Out[25]:

```
(200000, 10)
```

In [26]:

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

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200000 entries, 0 to 199999
Data columns (total 10 columns):
 #   Column             Non-Null Count   Dtype  
---  ------             --------------   -----  
 0   Unnamed: 0         11 non-null      float64
 1   Unnamed: 1         200000 non-null  int64  
 2   key                200000 non-null  object 
 3   fare_amount        200000 non-null  float64
 4   pickup_datetime    200000 non-null  object 
 5   pickup_longitude   200000 non-null  float64
 6   pickup_latitude    200000 non-null  float64
 7   dropoff_longitude  199999 non-null  float64
 8   dropoff_latitude   199999 non-null  float64
 9   passenger_count    200000 non-null  int64  
dtypes: float64(6), int64(2), object(2)
memory usage: 15.3+ MB
```

In [27]:

```
df.describe()
```

Out[27]:

	Unnamed: 0	Unnamed: 1	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	1.100000e+01	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	3.150848e+07	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	1.602404e+07	1.601382e+07	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	2.205147e+06	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	2.092417e+07	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	2.783520e+07	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	4.472760e+07	4.155530e+07	12.500000	-73.967153	40.767158	-73.963659	40.768001	2.000000
max	5.061106e+07	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

In [28]:

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

Out[28]:

```
Index(['Unnamed: 0', 'Unnamed: 1', 'key', 'fare_amount', 'pickup_datetime',
       'pickup_longitude', 'pickup_latitude', 'dropoff_longitude',
       'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [29]:

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

In [30]:

```
df.head()
```

Out[30]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

In [31]:

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

Out[31]:

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

In [32]:

```
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 [33]:

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

Out[33]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	199999.000000	199999.000000	200000.000000
mean	11.359955	-72.527638	39.935885	-72.525292	39.923890	1.684535
std	9.901776	11.437787	7.720539	13.117408	6.794829	1.385997
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	12.500000	-73.967153	40.767158	-73.963659	40.768001	2.000000
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

Filling Missing values

In [34]:

```
df.isnull()
```

Out[34]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
199995	False	False	False	False	False	False	False
199996	False	False	False	False	False	False	False
199997	False	False	False	False	False	False	False
199998	False	False	False	False	False	False	False
199999	False	False	False	False	False	False	False

200000 rows × 7 columns

In [35]:

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

Out[35]:

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

In [36]:

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

Out[36]:

```
2
```

In [37]:

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

In [38]:

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

Out[38]:

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

In [39]:

```
df.dtypes
```

Out[39]:

```
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

df.pickup_datetime = pd.to_datetime(df.pickup_datetime, errors='coerce',utc=True)

errors : *{'ignore', 'raise', 'coerce'}, default 'raise'*

- If 'raise', then invalid parsing will raise an exception.
- If 'coerce', then invalid parsing will be set as NaN.
- If 'ignore', then invalid parsing will return the input.

In [40]:

```
df.dtypes
```

Out[40]:

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

In [41]:

```
df.head()
```

Out[41]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.5	2015-05-07 19:52:06 UTC	-73.999817	40.738354	-73.999512	40.723217	1
1	7.7	2009-07-17 20:04:56 UTC	-73.994355	40.728225	-73.994710	40.750325	1
2	12.9	2009-08-24 21:45:00 UTC	-74.005043	40.740770	-73.962565	40.772647	1
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	40.803349	3
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	40.761247	5

To segregate each time of date and time

In [42]:

```
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,
             dayName = df.pickup_datetime.dt.day_name()
               )
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [42], in <cell line: 1>()
----> 1 df= df.assign(hour = df.pickup_datetime.dt.hour,
      2              day= df.pickup_datetime.dt.day,
      3              month = df.pickup_datetime.dt.month,
      4              year = df.pickup_datetime.dt.year,
      5              dayofweek = df.pickup_datetime.dt.dayofweek,
      6             dayName = df.pickup_datetime.dt.day_name()
      7               )

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5575, in NDFrame.__getattr__(self, name)
   5568 if (
   5569     name not in self._internal_names_set
   5570     and name not in self._metadata
   5571     and name not in self._accessors
   5572     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5573 ):
   5574     return self[name]
-> 5575 return object.__getattribute__(self, name)

File ~\anaconda3\lib\site-packages\pandas\core\accessor.py:182, in CachedAccessor.__get__(self, obj, cls)
    179 if obj is None:
    180     # we're accessing the attribute of the class, i.e., Dataset.geo
    181     return self._accessor
--> 182 accessor_obj = self._accessor(obj)
    183 # Replace the property with the accessor object. Inspired by:
    184 # https://www.pydanny.com/cached-property.html (https://www.pydanny.com/cached-property.html)
    185 # We need to use object.__setattr__ because we overwrite __setattr__ on
    186 # NDFrame
    187 object.__setattr__(obj, self._name, accessor_obj)

File ~\anaconda3\lib\site-packages\pandas\core\indexes\accessors.py:509, in CombinedDatetimelikeProperties.__new__(cls, dat
a)
    506 elif is_period_dtype(data.dtype):
    507     return PeriodProperties(data, orig)
--> 509 raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

#OR

In [43]:

```
# Only series datatype has the dt attribute
df['hour'] = df.pickup_datetime.dt.hour
df['day']= df["pickup_datetime"].dt.day  # is a series
df['month'] = df.pickup_datetime.dt.month # is a series
df['year'] = df.pickup_datetime.dt.year
df['dayofweek'] = df.pickup_datetime.dt.dayofweek
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [43], in <cell line: 2>()
      1 # Only series datatype has the dt attribute
----> 2 df['hour'] = df.pickup_datetime.dt.hour
      3 df['day']= df["pickup_datetime"].dt.day  # is a series
      4 df['month'] = df.pickup_datetime.dt.month # is a series

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:5575, in NDFrame.__getattr__(self, name)
   5568 if (
   5569     name not in self._internal_names_set
   5570     and name not in self._metadata
   5571     and name not in self._accessors
   5572     and self._info_axis._can_hold_identifiers_and_holds_name(name)
   5573 ):
   5574     return self[name]
-> 5575 return object.__getattribute__(self, name)

File ~\anaconda3\lib\site-packages\pandas\core\accessor.py:182, in CachedAccessor.__get__(self, obj, cls)
    179 if obj is None:
    180     # we're accessing the attribute of the class, i.e., Dataset.geo
    181     return self._accessor
--> 182 accessor_obj = self._accessor(obj)
    183 # Replace the property with the accessor object. Inspired by:
    184 # https://www.pydanny.com/cached-property.html (https://www.pydanny.com/cached-property.html)
    185 # We need to use object.__setattr__ because we overwrite __setattr__ on
    186 # NDFrame
    187 object.__setattr__(obj, self._name, accessor_obj)

File ~\anaconda3\lib\site-packages\pandas\core\indexes\accessors.py:509, in CombinedDatetimelikeProperties.__new__(cls, dat
a)
    506 elif is_period_dtype(data.dtype):
    507     return PeriodProperties(data, orig)
--> 509 raise AttributeError("Can only use .dt accessor with datetimelike values")

AttributeError: Can only use .dt accessor with datetimelike values
```

In [ ]:

```
df.head()
```

In [44]:

```
df.describe() # only for quantitative columns
```

Out[44]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000	200000.000000
mean	11.359955	-72.527638	39.935885	-72.525299	39.923890	1.684535
std	9.901776	11.437787	7.720539	13.117375	6.794812	1.385997
min	-52.000000	-1340.648410	-74.015515	-3356.666300	-881.985513	0.000000
25%	6.000000	-73.992065	40.734796	-73.991407	40.733823	1.000000
50%	8.500000	-73.981823	40.752592	-73.980093	40.753042	1.000000
75%	12.500000	-73.967153	40.767158	-73.963659	40.768001	2.000000
max	499.000000	57.418457	1644.421482	1153.572603	872.697628	208.000000

In [45]:

```
x=df['fare_amount'].value_counts()  # list(Series) # counts are sorted in descending order ie. 1st count is max count
```

In [46]:

```
print(type(x))
print("Max count = ",x[6.50])
x
```

```
<class 'pandas.core.series.Series'>
Max count =  9684
```

Out[46]:

```
6.50     9684
4.50     8247
8.50     7521
5.70     5858
5.30     5838
         ...
60.04       1
73.25       1
69.90       1
25.94       1
89.10       1
Name: fare_amount, Length: 1240, dtype: int64
```

In [47]:

```
# drop the column 'pickup_daetime' using drop()
# 'axis = 1' drops the specified column

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

In [48]:

```
df.head()
```

Out[48]:

	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	3
4	16.0	-73.925023	40.744085	-73.973082	40.761247	5

In [49]:

```
df.dtypes
```

Out[49]:

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

In [50]:

```
df.drop(["dayName"],axis=1,inplace=True)
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [50], in <cell line: 1>()
----> 1 df.drop(["dayName"],axis=1,inplace=True)

File ~\anaconda3\lib\site-packages\pandas\util\_decorators.py:311, in deprecate_nonkeyword_arguments.<locals>.decorate.<loc
als>.wrapper(*args, **kwargs)
    305 if len(args) > num_allow_args:
    306     warnings.warn(
    307         msg.format(arguments=arguments),
    308         FutureWarning,
    309         stacklevel=stacklevel,
    310     )
--> 311 return func(*args, **kwargs)

File ~\anaconda3\lib\site-packages\pandas\core\frame.py:4954, in DataFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   4806 @deprecate_nonkeyword_arguments(version=None, allowed_args=["self", "labels"])
   4807 def drop(
   4808     self,
   (...)
   4815     errors: str = "raise",
   4816 ):
   4817     """
   4818     Drop specified labels from rows or columns.
   4819
   (...)
   4952             weight  1.0     0.8
   4953     """
-> 4954     return super().drop(
   4955         labels=labels,
   4956         axis=axis,
   4957         index=index,
   4958         columns=columns,
   4959         level=level,
   4960         inplace=inplace,
   4961         errors=errors,
   4962     )

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:4267, in NDFrame.drop(self, labels, axis, index, columns, level,
inplace, errors)
   4265 for axis, labels in axes.items():
   4266     if labels is not None:
-> 4267         obj = obj._drop_axis(labels, axis, level=level, errors=errors)
   4269 if inplace:
   4270     self._update_inplace(obj)

File ~\anaconda3\lib\site-packages\pandas\core\generic.py:4311, in NDFrame._drop_axis(self, labels, axis, level, errors, co
nsolidate, only_slice)
   4309         new_axis = axis.drop(labels, level=level, errors=errors)
   4310     else:
-> 4311         new_axis = axis.drop(labels, errors=errors)
   4312     indexer = axis.get_indexer(new_axis)
   4314 # Case for non-unique axis
   4315 else:

File ~\anaconda3\lib\site-packages\pandas\core\indexes\base.py:6644, in Index.drop(self, labels, errors)
   6642 if mask.any():
   6643     if errors != "ignore":
-> 6644         raise KeyError(f"{list(labels[mask])} not found in axis")
   6645     indexer = indexer[~mask]
   6646 return self.delete(indexer)

KeyError: "['dayName'] not found in axis"
```

Checking outliers and filling them

In [ ]:

```
number_of_columns = len(df.columns)
```

In [ ]:

```
number_of_columns
```

##Number of columns is 12 (11 quntitative and 1 categoraical) , but in the following box plots only 11 subplots are visible. This is because boxplots are only for quantitative valued columns and not for categoraical valued columns

In [ ]:

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

In [ ]:

```
# df.plot(kind = "box", subplots = True, layout = (5,2), figsize=(15,20)) # 5 rows x 2 columns = 10 spaces for 10 subplots ; where 10 subp
df.plot(kind = "box", subplots = True, layout = (7,2), figsize=(15,20)) # 7 rows x 2 columns = 14 spaces for 14 subplots ; where 14 subplo
```

In [ ]:

```
df.plot(kind = "box", subplots = True, layout = (4,3), figsize=(15,20)) # 4 rows x 3 columns = 12 spaces for 12 subplots ; where 11 subplo
```

###CONCLUSION : Number of spaces for subplots generated can be >= actual number of subplots (of the quantitative columns) but not < the actual number of subplots

###Explanation of clip function

In [51]:

```
a  = np.arange(10,100,10) # considers from 10 to 99 with step size of 10
a1 = np.clip(a,30,60)    # np.clip(list, lowerlimit, upper limit)

print("Before clipping :",a )
print(" After clipping :",a1)
```

```
Before clipping : [10 20 30 40 50 60 70 80 90]
 After clipping : [30 30 30 40 50 60 60 60 60]
```

In [52]:

```
# Elimination of the outliers

# Using the InterQuartile Range to fill the values

def remove_outlier(df1 , col):
    Q1 = df1[col].quantile(0.25)
    Q2 = df1[col].quantile(0.50)
    Q3 = df1[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_whisker = Q1-1.5*IQR
    upper_whisker = Q3+1.5*IQR
    print("-----**********************************************-----")
    print("col=",col,"Q1=",Q1,"Q2=",Q2,"Q3=",Q3)
    print("-----**********************************************-----")
    df1[col] = np.clip(df1[col] , lower_whisker , upper_whisker)
    return df1

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

In [53]:

```
df = treat_outliers_all(df , df.columns)
```

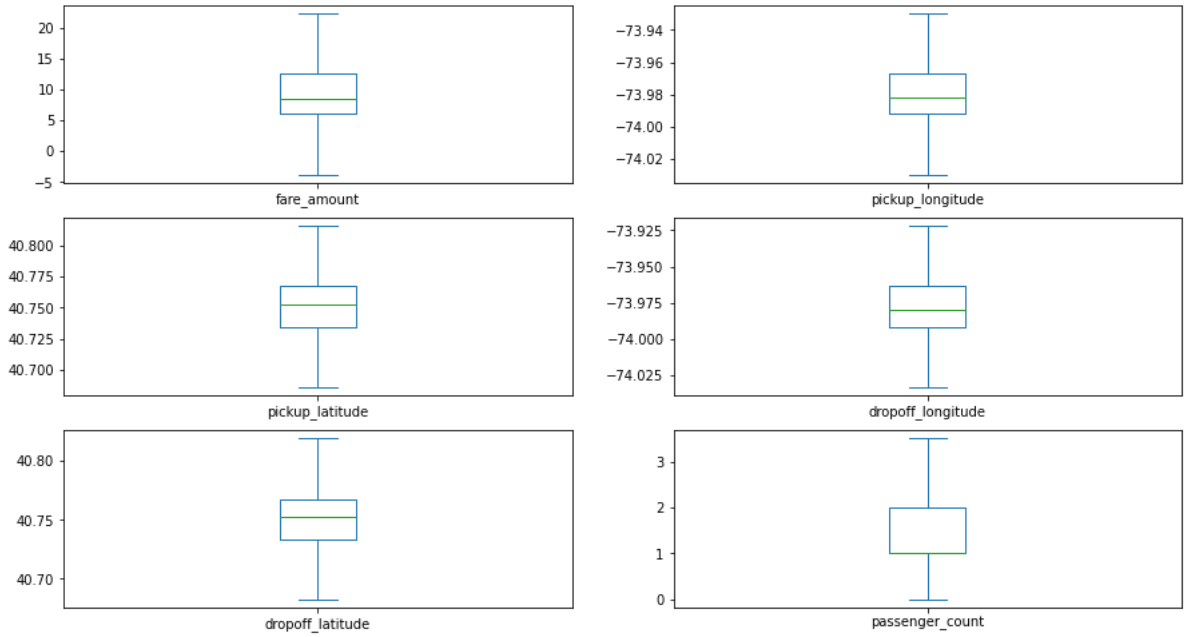
```
col_list Index(['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count'],
      dtype='object')
-----**********************************************-----
col= fare_amount Q1= 6.0 Q2= 8.5 Q3= 12.5
-----**********************************************-----
-----**********************************************-----
col= pickup_longitude Q1= -73.992065 Q2= -73.981823 Q3= -73.9671535
-----**********************************************-----
-----**********************************************-----
col= pickup_latitude Q1= 40.73479575 Q2= 40.752592 Q3= 40.767158
-----**********************************************-----
-----**********************************************-----
col= dropoff_longitude Q1= -73.991407 Q2= -73.980093 Q3= -73.96365875000001
-----**********************************************-----
-----**********************************************-----
col= dropoff_latitude Q1= 40.733823 Q2= 40.753042 Q3= 40.768001139999996
-----**********************************************-----
-----**********************************************-----
col= passenger_count Q1= 1.0 Q2= 1.0 Q3= 2.0
-----**********************************************-----
```

In [54]:

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

Out[54]:

```
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)
dtype: object
```



In [55]:

```
#Finding inccorect latitude (Less than -90 or greater than 90) and Longitude (greater than 180 or less than -180)
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)

                                ]
```

In [56]:

```
incorrect_coordinates
```

Out[56]:

fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count

In [57]:

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

In [58]:

```
df
```

Out[58]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	7.50	-73.999817	40.738354	-73.999512	40.723217	1.0
1	7.70	-73.994355	40.728225	-73.994710	40.750325	1.0
2	12.90	-74.005043	40.740770	-73.962565	40.772647	1.0
3	5.30	-73.976124	40.790844	-73.965316	40.803349	3.0
4	16.00	-73.929786	40.744085	-73.973082	40.761247	3.5
...	...	...	...	...	...	...
199995	3.00	-73.987042	40.739367	-73.986525	40.740297	1.0
199996	7.50	-73.984722	40.736837	-74.006672	40.739620	1.0
199997	22.25	-73.986017	40.756487	-73.922036	40.692588	2.0
199998	14.50	-73.997124	40.725452	-73.983215	40.695416	1.0
199999	14.10	-73.984395	40.720077	-73.985508	40.768793	1.0

200000 rows × 6 columns

In [59]:

```
!pip install haversine
```

```
Requirement already satisfied: haversine in c:\users\vinit\anaconda3\lib\site-packages (2.7.0)

WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
WARNING: Ignoring invalid distribution -atplotlib (c:\users\vinit\anaconda3\lib\site-packages)
```

In [60]:

```
import haversine as hs  #Calculate the distance using Haversine to calculate the distance between to points. Can't use Eucladian as it is
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_latitu
        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[60]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	dist_travel_km
0	7.5	-73.999817	40.738354	-73.999512	40.723217	1.0	1.683325
1	7.7	-73.994355	40.728225	-73.994710	40.750325	1.0	2.457593
2	12.9	-74.005043	40.740770	-73.962565	40.772647	1.0	5.036384
3	5.3	-73.976124	40.790844	-73.965316	40.803349	3.0	1.661686
4	16.0	-73.929786	40.744085	-73.973082	40.761247	3.5	4.116088

In [61]:

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

```
Remaining observastions in the dataset: (163040, 7)
```

In [62]:

```
bool_df = df.isnull()
bool_df
```

Out[62]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	dist_travel_km
0	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...
199994	False	False	False	False	False	False	False
199996	False	False	False	False	False	False	False
199997	False	False	False	False	False	False	False
199998	False	False	False	False	False	False	False
199999	False	False	False	False	False	False	False

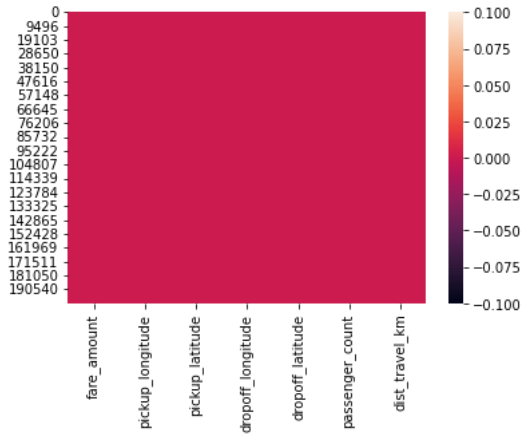
163040 rows × 7 columns

In [63]:

```
sns.heatmap(bool_df) #Free of null values , no correlation exists since r = 0 (False is a constant value which is not related to the index
```

Out[63]:

<AxesSubplot:>



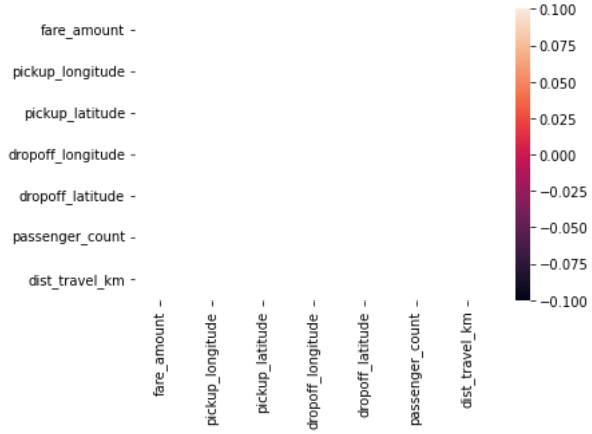
In [64]:

```
sns.heatmap(bool_df.corr())
```

```
C:\Users\vinit\anaconda3\lib\site-packages\seaborn\matrix.py:198: RuntimeWarning: All-NaN slice encountered
  vmin = np.nanmin(calc_data)
C:\Users\vinit\anaconda3\lib\site-packages\seaborn\matrix.py:203: RuntimeWarning: All-NaN slice encountered
  vmax = np.nanmax(calc_data)
```

Out[64]:

<AxesSubplot:>



In [65]:

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

In [66]:

```
corr
```

Out[66]:

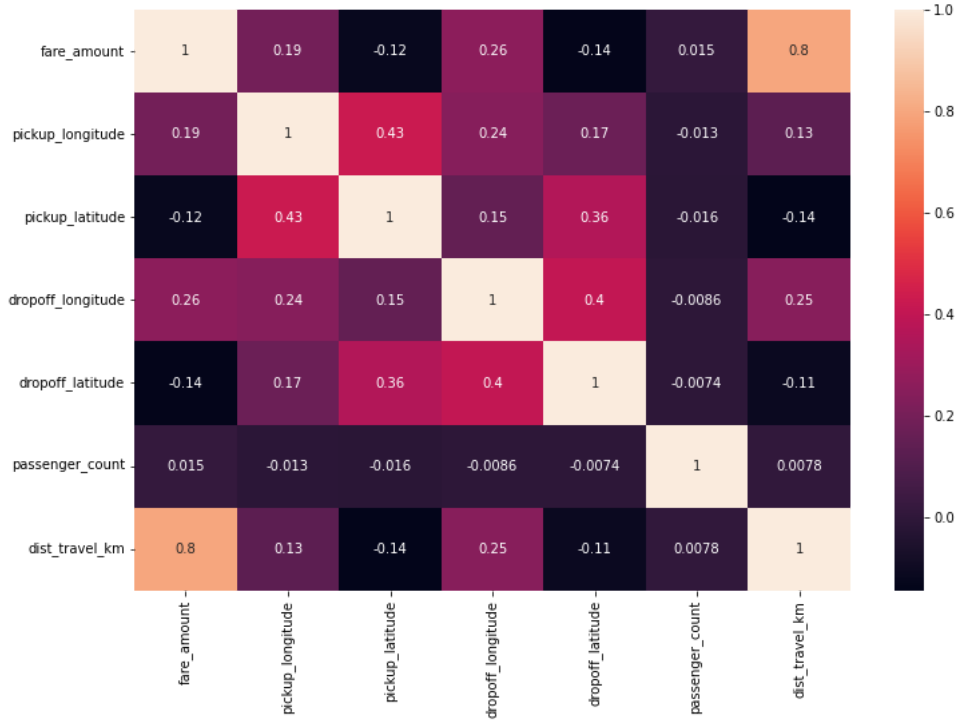
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	dist_travel_km
fare_amount	1.000000	0.193177	-0.119469	0.258583	-0.137745	0.015056	0.798926
pickup_longitude	0.193177	1.000000	0.425309	0.241569	0.169203	-0.012892	0.130003
pickup_latitude	-0.119469	0.425309	1.000000	0.148761	0.358836	-0.016186	-0.143530
dropoff_longitude	0.258583	0.241569	0.148761	1.000000	0.401751	-0.008642	0.247210
dropoff_latitude	-0.137745	0.169203	0.358836	0.401751	1.000000	-0.007419	-0.111577
passenger_count	0.015056	-0.012892	-0.016186	-0.008642	-0.007419	1.000000	0.007754
dist_travel_km	0.798926	0.130003	-0.143530	0.247210	-0.111577	0.007754	1.000000

In [67]:

```
plt.figure(figsize = (12,8))
sns.heatmap(df.corr(),annot = True) #Correlation Heatmap (Light values means highly correlated)
```

Out[67]:

```
<AxesSubplot:>
```



Dividing the dataset into feature and target values

In [70]:

```
df.columns
```

Out[70]:

```
Index(['fare_amount', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count',
       'dist_travel_km'],
      dtype='object')
```

###OR, since all columns except "fare_amount" are the features, the following can be done instead

In [72]:

```
x = df[df.columns[5:]] # since 0th column is fare_amount
x
```

Out[72]:

	passenger_count	dist_travel_km
0	1.0	1.683325
1	1.0	2.457593
2	1.0	5.036384
3	3.0	1.661686
4	3.5	4.116088
...	...	...
199994	1.0	1.122879
199996	1.0	1.875053
199997	2.0	8.919323
199998	1.0	3.539720
199999	1.0	5.417791

163040 rows × 2 columns

In [73]:

```
y = df['fare_amount']
y
```

Out[73]:

```
0          7.50
1          7.70
2         12.90
3          5.30
4         16.00
          ...
199994    12.00
199996     7.50
199997    22.25
199998    14.50
199999    14.10
Name: fare_amount, Length: 163040, dtype: float64
```

Dividing the dataset into training and testing dataset

In [74]:

```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.70,random_state=7) # 30% for training, 70% for testing / evaluation
```

In [75]:

```
X_train
```

Out[75]:

	passenger_count	dist_travel_km
85111	3.0	4.748305
134870	1.0	1.449640
82644	3.0	1.631489
110503	1.0	1.364317
95194	1.0	1.324667
...	...	...
81555	3.5	2.747475
65586	3.5	9.029333
13187	3.5	1.466365
60932	3.5	5.812440
75614	1.0	2.177099

48912 rows × 2 columns

In [76]:

```
X_test
```

Out[76]:

	passenger_count	dist_travel_km
4601	3.5	1.825137
16992	2.0	5.257621
22135	2.0	1.321082
143093	2.0	3.301984
42087	1.0	1.291802
...	...	...
43618	1.0	8.272751
14485	1.0	2.107203
154067	1.0	3.909568
26162	1.0	1.703482
175834	3.5	4.127410

114128 rows × 2 columns

In [77]:

```
y_train
```

Out[77]:

```
85111     10.1
134870     6.5
82644      4.9
110503     5.7
95194      7.0
          ...
81555      8.5
65586     19.7
13187      5.5
60932     17.3
75614      9.0
Name: fare_amount, Length: 48912, dtype: float64
```

In [78]:

```
y_test
```

Out[78]:

```
4601       6.50
16992     22.25
22135      7.30
143093    12.10
42087      5.30
          ...
43618     22.25
14485      8.00
154067    10.50
26162      9.00
175834    10.90
Name: fare_amount, Length: 114128, dtype: float64
```

In [79]:

```
len(y) # 100% of entries or examples
```

Out[79]:

```
163040
```

In [80]:

```
len(y_train) # 30% for training
```

Out[80]:

```
48912
```

In [81]:

```
len(y_test) # 70% for testing
```

Out[81]:

```
114128
```

In [82]:

```
len(y_train) + len(y_test) # total becomes 100%
```

Out[82]:

```
163040
```

Linear Regression

In [83]:

```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [84]:

```
regression.fit(X_train,y_train)
```

Out[84]:

```
LinearRegression()
```

In [85]:

```
regression.intercept_ #To find the linear intercept
```

Out[85]:

```
4.586295329672887
```

In [86]:

```
regression.coef_ #To find the linear coefficients ie parameters (11 features so 11 parameters)
```

Out[86]:

```
array([0.04191373, 1.9275112 ])
```

In [87]:

```
for i in range(0,len(regression.coef_)):
    print("theta",i,"=",regression.coef_[i])
```

```
theta 0 = 0.041913729047169605
theta 1 = 1.927511196162005
```

In [88]:

```
y_pred = regression.predict(X_test) #To predict the target values
```

In [89]:

```
comparison = pd.DataFrame({"Actual Label":y_test,"Predicted Label":y_pred})
```

In [ ]:

```

```

In [90]:

```
comparison
```

Out[90]:

	Actual Label	Predicted Label
4601	6.50	8.250965
16992	22.25	14.804246
22135	7.30	7.216523
143093	12.10	11.034734
42087	5.30	7.118171
...	...	...
43618	22.25	20.574030
14485	8.00	8.689866
154067	10.50	12.163946
26162	9.00	7.911690
175834	10.90	12.688623

114128 rows × 2 columns

In [91]:

```
comparison.reset_index()
```

Out[91]:

	index	Actual Label	Predicted Label
0	4601	6.50	8.250965
1	16992	22.25	14.804246
2	22135	7.30	7.216523
3	143093	12.10	11.034734
4	42087	5.30	7.118171
...	...	...	...
114123	43618	22.25	20.574030
114124	14485	8.00	8.689866
114125	154067	10.50	12.163946
114126	26162	9.00	7.911690
114127	175834	10.90	12.688623

114128 rows × 3 columns

In [92]:

```
comparison.reset_index().drop(["index"],axis=1)
```

Out[92]:

	Actual Label	Predicted Label
0	6.50	8.250965
1	22.25	14.804246
2	7.30	7.216523
3	12.10	11.034734
4	5.30	7.118171
...	...	...
114123	22.25	20.574030
114124	8.00	8.689866
114125	10.50	12.163946
114126	9.00	7.911690
114127	10.90	12.688623

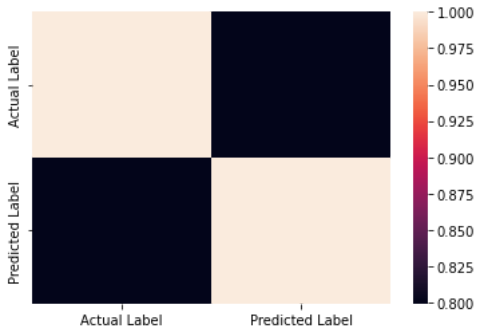
114128 rows × 2 columns

In [93]:

```
sns.heatmap(comparison.corr())
```

Out[93]:

<AxesSubplot:>



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

In [94]:

```
from sklearn.metrics import r2_score
```

In [95]:

```
r2_score(y_test,y_pred)
```

Out[95]:

```
0.6389302826000085
```

In [96]:

```
from sklearn.metrics import mean_squared_error
```

In [97]:

```
MSE = mean_squared_error(y_test,y_pred)
```

In [98]:

```
MSE
```

Out[98]:

```
10.33461066195151
```

In [99]:

```
RMSE = np.sqrt(MSE)
```

In [100]:

```
RMSE
```

Out[100]:

```
3.2147489267361937
```

Random Forest Regression

In [101]:

```
from sklearn.ensemble import RandomForestRegressor
```

In [102]:

```
rf = RandomForestRegressor(n_estimators=100) #Here n_estimators means number of trees you want to build before making the prediction
```

In [103]:

```
rf.fit(X_train,y_train)
```

Out[103]:

```
RandomForestRegressor()
```

In [104]:

```
y_pred = rf.predict(X_test)
```

In [105]:

```
y_pred
```

Out[105]:

```
array([11.07  , 12.9815,  7.509 , ..., 12.031 ,  9.148 , 13.349 ])
```

Metrics evaluation for Random Forest

In [106]:

```
R2_Random = r2_score(y_test,y_pred)
```

In [107]:

```
R2_Random
```

Out[107]:

```
0.5559602768319912
```

In [108]:

```
MSE_Random = mean_squared_error(y_test,y_pred)
```

In [109]:

```
MSE_Random
```

Out[109]:

```
12.709394990049667
```

In [110]:

```
RMSE_Random = np.sqrt(MSE_Random)
```

In [111]:

```
RMSE_Random
```

Out[111]:

```
3.565023841441971
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: