Design an algorithm that will tell the fare to be charged for a passenger.

Problem Statement Scenario: A fare calculator helps a customer in identifying the fare valid for the trip. They are often used by passengers who are new to a city or tourists to get an estimate of the travel costs. You are provided with a dataset with features like fare amount, pickup and drop location, passenger count, and so on.

Following actions should be performed:

Understand the type of data.

Identify the output variable.

Identify the factors which affect the output variable.

Check if there are any biases in your dataset.

Count the null values existing in columns.

Remove the null value rows in the target variable.

Perform train test split.

Predict the accuracy using regression models.

Check and compare the accuracy of the different models.

In [3]:
```
import pandas as pd
import numpy as np
import seaborn as sns
%matplotlib inline
import matplotlib.pyplot as plt
```

In [6]:
```
train_df = pd.read_csv(r"train.csv", nrows=200)
test_df = pd.read_csv(r"test.csv")
print(train_df.shape)
print(train_df.columns)
print(test_df.shape)
print(test_df.columns)
```

```
(200, 8)
```

```
Index(['key', 'fare_amount', 'pickup_datetime', 'pickup_longitude',
       'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
       'passenger_count'],
      dtype='object')
(9914, 7)
Index(['key', 'pickup_datetime', 'pickup_longitude', 'pickup_latitude',
       'dropoff_longitude', 'dropoff_latitude', 'passenger_count'],
      dtype='object')
```

In [7]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 200 entries, 0 to 199
Data columns (total 8 columns):
 #   Column             Non-Null Count  Dtype  
---  ------             --------------  -----  
 0   key                200 non-null    object 
 1   fare_amount        200 non-null    float64
 2   pickup_datetime    200 non-null    object 
 3   pickup_longitude   200 non-null    float64
 4   pickup_latitude    200 non-null    float64
 5   dropoff_longitude  200 non-null    float64
 6   dropoff_latitude   200 non-null    float64
 7   passenger_count    200 non-null    int64  
dtypes: float64(5), int64(1), object(2)
memory usage: 12.6+ KB
```

In [8]:
```
# changing pick-up date time object type to DATETIME

train_df["pickup_datetime"]=pd.to_datetime(train_df['pickup_datetime'])
```

In [9]:
```
train_df.head()
```

Out[9]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1

In [10]:
```
# Taxi fair price is affected by several factors: distance, location, special requirements etc

train_df.describe()
```

Out[10]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count
count	200.000000	200.000000	200.000000	200.000000	200.000000	200.000000
mean	11.088250	-72.129763	39.730607	-72.128149	39.731270	1.770000
std	8.691217	11.579020	6.377990	11.578776	6.378135	1.391684
min	3.300000	-74.035839	0.000000	-74.035839	0.000000	1.000000
25%	5.700000	-73.994132	40.733160	-73.992982	40.731490	1.000000
50%	8.500000	-73.982926	40.748692	-73.981733	40.751558	1.000000
75%	12.600000	-73.970148	40.763612	-73.969754	40.764677	2.000000
max	58.000000	0.000000	40.828531	0.000000	40.868610	6.000000

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

Out[12]:
```
key                0
fare_amount        0
pickup_datetime    0
pickup_longitude   0
pickup_latitude    0
dropoff_longitude  0
dropoff_latitude   0
passenger_count    0
dtype: int64
```

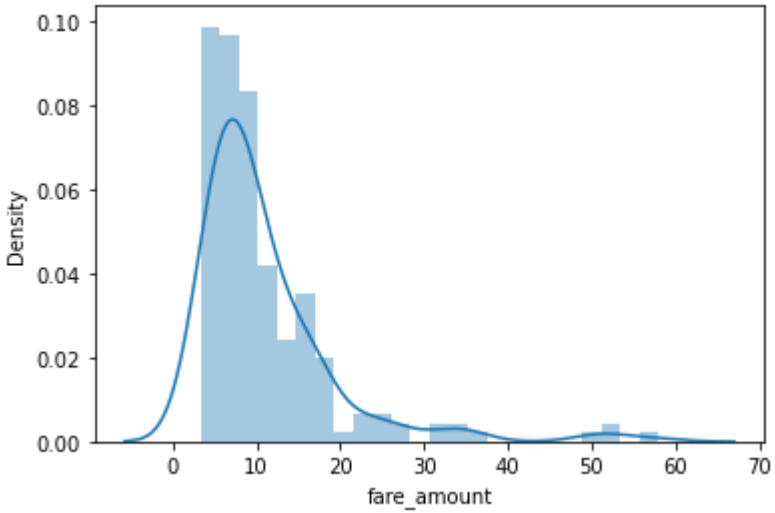
Above there aren't null values so we do not need to drop any data

In [13]:
```
sns.distplot(train_df['fare_amount'])
```

```
C:\Users\ctoqu\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

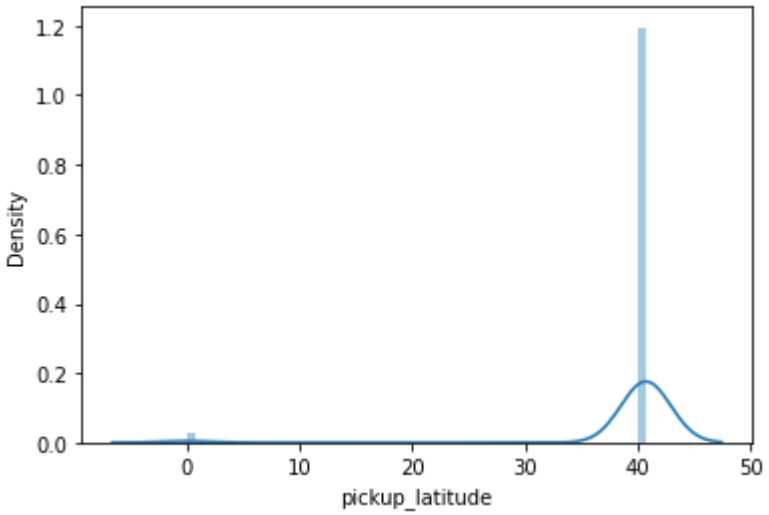
Out[13]: <AxesSubplot:xlabel='fare_amount', ylabel='Density'>



In [14]:
```
sns.distplot(train_df['pickup_latitude'])
```

```
C:\Users\ctoqu\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated funct
ion and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with
similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

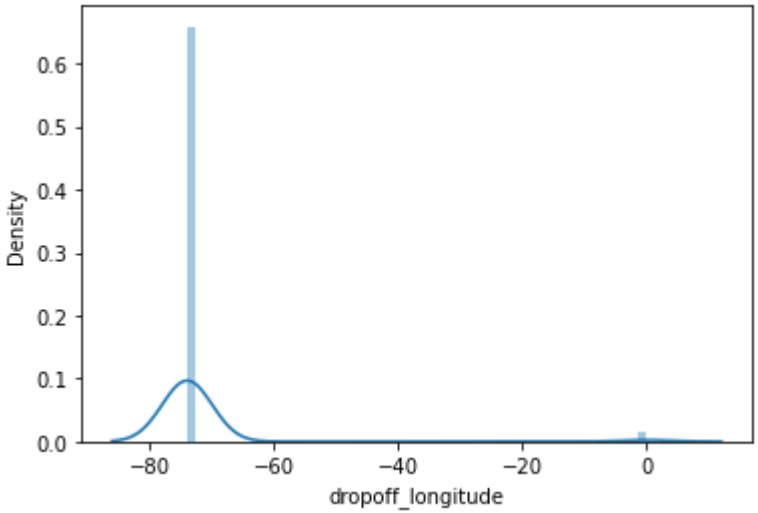
Out[14]: <AxesSubplot:xlabel='pickup_latitude', ylabel='Density'>



In [15]:
```
sns.distplot(train_df['dropoff_longitude'])
```

```
C:\Users\ctoqu\anaconda3\lib\site-packages\seaborn\distributions.py:2551: FutureWarning: `distplot` is a deprecated function and will be removed in a future version. Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).
  warnings.warn(msg, FutureWarning)
```

Out[15]: <AxesSubplot:xlabel='dropoff_longitude', ylabel='Density'>



In [16]:
```
# Looking for min and max values in longitud and latitud

print("drop_off latitude min value",test_df["dropoff_latitude"].min())
print("drop_off latitude max value",test_df["dropoff_latitude"].max())
print("drop_off longitude min value", test_df["dropoff_longitude"].min())
print("drop_off longitude max value",test_df["dropoff_longitude"].max())
print("pickup latitude min value",test_df["pickup_latitude"].min())
print("pickup latitude max value",test_df["pickup_latitude"].max())
print("pickup longitude min value",test_df["pickup_longitude"].min())
print("pickup longitude max value",test_df["pickup_longitude"].max())
```

```
drop_off latitude min value 40.568973
drop_off latitude max value 41.696683
drop_off longitude min value -74.263242
drop_off longitude max value -72.990963
pickup latitude min value 40.573143
pickup latitude max value 41.709555
pickup longitude min value -74.252193
pickup longitude max value -72.986532
```

In [20]:
```
# now that we know that range on the data set we want to keep same  in the train set.
```

```
#This allows to remove noisy data and keeping the values for NY only

min_longitude = -74.263242
min_latitude = 40.568973
max_longitude = -72.986532
max_latitude = 41.696683
```

In [22]:
```
#Dropping values out of range

tempdf=train_df[(train_df["dropoff_latitude"]<min_latitude) | (train_df["pickup_latitude"]<min_latitude) | (train_df["drc
print('before droping',train_df.shape)
train_df.drop(tempdf.index,inplace=True)
print("After droping",train_df.shape)
```

```
before droping (200, 8)
After droping (195, 8)
```

In [23]:
```
#Removing the rows with fare amount is negative

print("before droping", train_df.shape)
train_df=train_df[train_df['fare_amount']>0]
print("after droping", train_df.shape)
```

```
before droping (195, 8)
after droping (195, 8)
```

In [24]:
```
# Creating extra features that affect the fare e.g. day of week, holiday, evening etc

import calendar
train_df['day']=train_df['pickup_datetime'].apply(lambda x:x.day)
train_df['hour']=train_df['pickup_datetime'].apply(lambda x:x.hour)
train_df['weekday']=train_df['pickup_datetime'].apply(lambda x:calendar.day_name[x.weekday()])
train_df['month']=train_df['pickup_datetime'].apply(lambda x:x.month)
train_df['year']=train_df['pickup_datetime'].apply(lambda x:x.year)
```

In [25]:
```
train_df.head()
```

Out[25]:

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	d
0	2009-06-15 17:26:21.0000001	4.5	2009-06-15 17:26:21+00:00	-73.844311	40.721319	-73.841610	40.712278	1	
1	2010-01-05 16:52:16.0000002	16.9	2010-01-05 16:52:16+00:00	-74.016048	40.711303	-73.979268	40.782004	1	

	key	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	d
2	2011-08-18 00:35:00.00000049	5.7	2011-08-18 00:35:00+00:00	-73.982738	40.761270	-73.991242	40.750562	2	
3	2012-04-21 04:30:42.0000001	7.7	2012-04-21 04:30:42+00:00	-73.987130	40.733143	-73.991567	40.758092	1	
4	2010-03-09 07:51:00.000000135	5.3	2010-03-09 07:51:00+00:00	-73.968095	40.768008	-73.956655	40.783762	1	

In [26]:
```
#Converting weekday from names to number

train_df.weekday = train_df.weekday.map({'Sunday':0,'Monday':1,'Tuesday':2,'Wednesday':3,'Thursday':4,'Friday':5,'Saturda
```

In [27]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195 entries, 0 to 199
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   key                195 non-null    object
 1   fare_amount        195 non-null    float64
 2   pickup_datetime    195 non-null    datetime64[ns, UTC]
 3   pickup_longitude   195 non-null    float64
 4   pickup_latitude    195 non-null    float64
 5   dropoff_longitude  195 non-null    float64
 6   dropoff_latitude   195 non-null    float64
 7   passenger_count    195 non-null    int64
 8   day                195 non-null    int64
 9   hour               195 non-null    int64
 10  weekday            195 non-null    int64
 11  month              195 non-null    int64
 12  year               195 non-null    int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(6), object(1)
memory usage: 21.3+ KB
```

In [29]:
```
#We will keep only the rows where the number of passangers are less or equal to 8

train_df=train_df[train_df['passenger_count']<=8]
```

In [30]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
```

```
Int64Index: 195 entries, 0 to 199
Data columns (total 13 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   key                195 non-null    object
 1   fare_amount        195 non-null    float64
 2   pickup_datetime    195 non-null    datetime64[ns, UTC]
 3   pickup_longitude   195 non-null    float64
 4   pickup_latitude    195 non-null    float64
 5   dropoff_longitude  195 non-null    float64
 6   dropoff_latitude   195 non-null    float64
 7   passenger_count    195 non-null    int64
 8   day                195 non-null    int64
 9   hour               195 non-null    int64
 10  weekday            195 non-null    int64
 11  month              195 non-null    int64
 12  year               195 non-null    int64
dtypes: datetime64[ns, UTC](1), float64(5), int64(6), object(1)
memory usage: 21.3+ KB
```

In [31]:
```
# Key column and pick-up_datetime are not needed as we already created variables from it

train_df.drop(["key","pickup_datetime"], axis=1, inplace=True)
```

In [32]:
```
train_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 195 entries, 0 to 199
Data columns (total 11 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   fare_amount        195 non-null    float64
 1   pickup_longitude   195 non-null    float64
 2   pickup_latitude    195 non-null    float64
 3   dropoff_longitude  195 non-null    float64
 4   dropoff_latitude   195 non-null    float64
 5   passenger_count    195 non-null    int64
 6   day                195 non-null    int64
 7   hour               195 non-null    int64
 8   weekday            195 non-null    int64
 9   month              195 non-null    int64
 10  year               195 non-null    int64
dtypes: float64(5), int64(6)
memory usage: 18.3 KB
```

train test split

In [33]:
```
from sklearn.model_selection import train_test_split
```

In [35]:
```
x=train_df.drop("fare_amount", axis=1)
```

In [36]:
```
y=train_df["fare_amount"]
```

In [37]:
```
x_train,x_test,y_train,y_test =train_test_split(x,y,test_size=0.2,random_state=101)
```

In [38]:
```
x_train.head()
```

Out[38]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	weekday	month	year
83	-74.009728	40.705167	-73.970897	40.749307	1	8	21	1	7	2013
120	-73.972018	40.750142	-74.006008	40.736220	5	8	21	4	11	2012
174	-73.944023	40.775959	-73.955048	40.785080	1	27	7	5	2	2015
125	-73.971696	40.763378	-73.962035	40.776598	1	10	16	5	5	2013
4	-73.968095	40.768008	-73.956655	40.783762	1	9	7	2	3	2010

In [39]:
```
x_test.head()
```

Out[39]:

	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_count	day	hour	weekday	month	year
42	-73.978450	40.762920	-74.008482	40.716502	1	22	8	5	11	2013
167	-73.954598	40.786760	-73.966013	40.768112	1	13	8	3	8	2014
64	-74.003919	40.753019	-73.992368	40.735362	1	7	10	1	11	2011
35	-73.983330	40.738720	-73.933197	40.847225	1	11	3	0	1	2015
127	-73.988492	40.717977	-73.978180	40.737407	1	6	21	1	1	2014

In [42]:
```
x_train.shape
```

Out[42]:
```
(156, 10)
```

In [43]:
```
x_test.shape
```

Out[43]:
```
(39, 10)
```

Regression Model

Starting with linear regression

In [44]:
```
from sklearn.linear_model import LinearRegression
```

In [45]:
```
lrmodel=LinearRegression()
lrmodel.fit(x_train,y_train)
```

Out[45]:
```
LinearRegression()
```

In [46]:
```
predictedvalues = lrmodel.predict(x_test)
```

In [49]:
```
#Calculating rmse for the LR model

from sklearn.metrics import mean_squared_error
lrmodelrmse =np.sqrt(mean_squared_error(predictedvalues, y_test))
print('RMSE value for linear regression is', lrmodelrmse)
```

```
RMSE value for linear regression is 9.802436637397076
```

In [56]:
```
#Random Forest and its rmse

from sklearn.ensemble import RandomForestRegressor
rfrmodel = RandomForestRegressor(n_estimators=100, random_state=101)
```

In [57]:
```
rfrmodel.fit(x_train,y_train)
rfrmodel_pred= rfmodel.predict(x_test)
```

In [58]:
```
rfrmodel_rmse=np.sqrt(mean_squared_error(rfrmodel_pred,y_test))
print("RMSE value for Random forest regression is",rfrmodel_rmse)
```

```
RMSE value for Random forest regression is 7.668953827207864
```

RandomForest Regressor gives a better value, we can use it as final model

In [ ]:
```

```