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

In [1]:
```
import pandas as pd
df=pd.read_csv("uber.csv")
```

In [2]:
```
df.head()
```

Out[2]:

	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
3	25894730	2009-06-26 08:22:21.0000001	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790
4	17610152	2014-08-28 17:47:00.000000188	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744

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

Out[3]:

	Unnamed: 0	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dro
count	2.000000e+05	200000.000000	200000.000000	200000.000000	199999.000000	1
mean	2.771250e+07	11.359955	-72.527638	39.935885	-72.525292	
std	1.601382e+07	9.901776	11.437787	7.720539	13.117408	
min	1.000000e+00	-52.000000	-1340.648410	-74.015515	-3356.666300	
25%	1.382535e+07	6.000000	-73.992065	40.734796	-73.991407	
50%	2.774550e+07	8.500000	-73.981823	40.752592	-73.980093	
75%	4.155530e+07	12.500000	-73.967154	40.767158	-73.963658	
max	5.542357e+07	499.000000	57.418457	1644.421482	1153.572603	

In [4]:
```
df.info()
```

```
<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
```

Preprocessing data

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
Unnamed: 0           0
key                  0
fare_amount          0
pickup_datetime      0
pickup_longitude     0
pickup_latitude      0
dropoff_longitude    1
dropoff_latitude     1
passenger_count      0
dtype: int64
```

In [6]:
```
df.dropna(axis=0,inplace=True)
```

In [7]:
```
df.isnull().sum()
```

Out[7]:
```
Unnamed: 0           0
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
```

In [8]:
```
#dropping unwanted columns
df.drop(['Unnamed: 0','key'],inplace=True,axis=1)
```

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

Out[9]:

	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	
3	5.3	2009-06-26 08:22:21 UTC	-73.976124	40.790844	-73.965316	
4	16.0	2014-08-28 17:47:00 UTC	-73.925023	40.744085	-73.973082	

In [10]:
```
df["pickup_datetime"]=pd.to_datetime(df["pickup_datetime"])
```

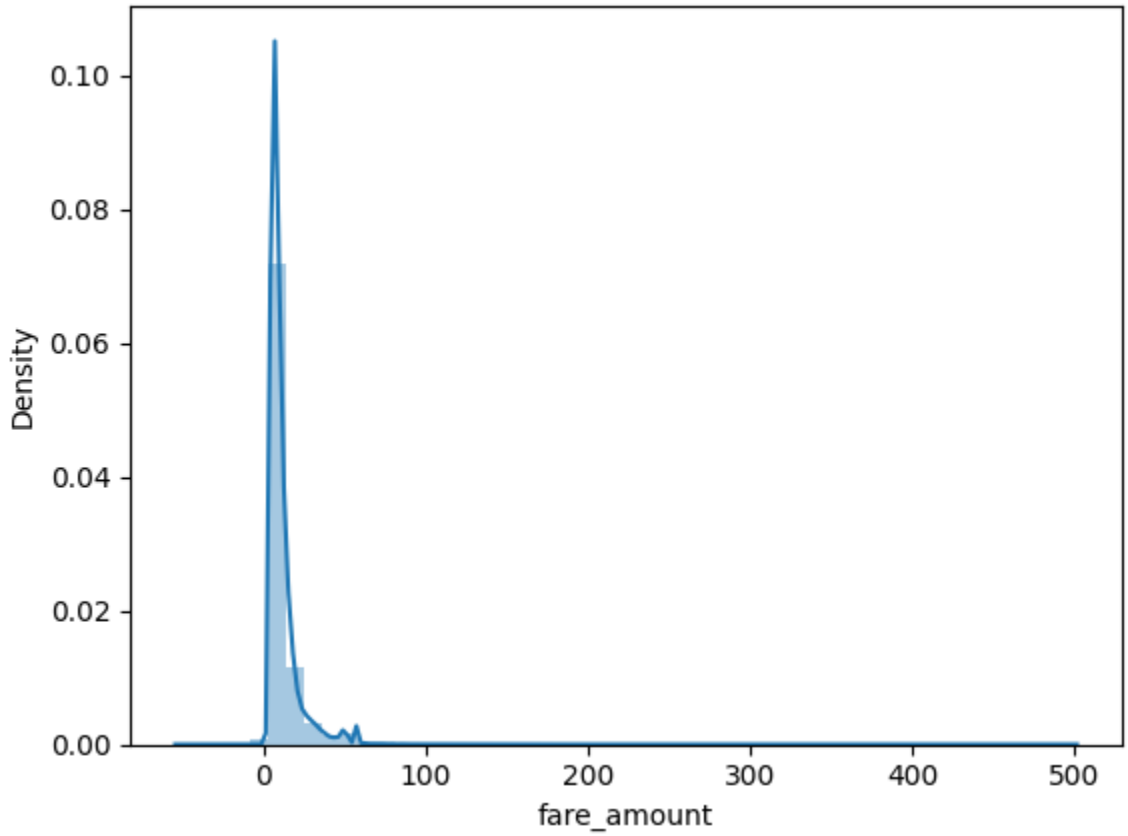
2. Identify outliers.

In [11]:
```
import seaborn as sb
import warnings

warnings.filterwarnings("ignore")
```

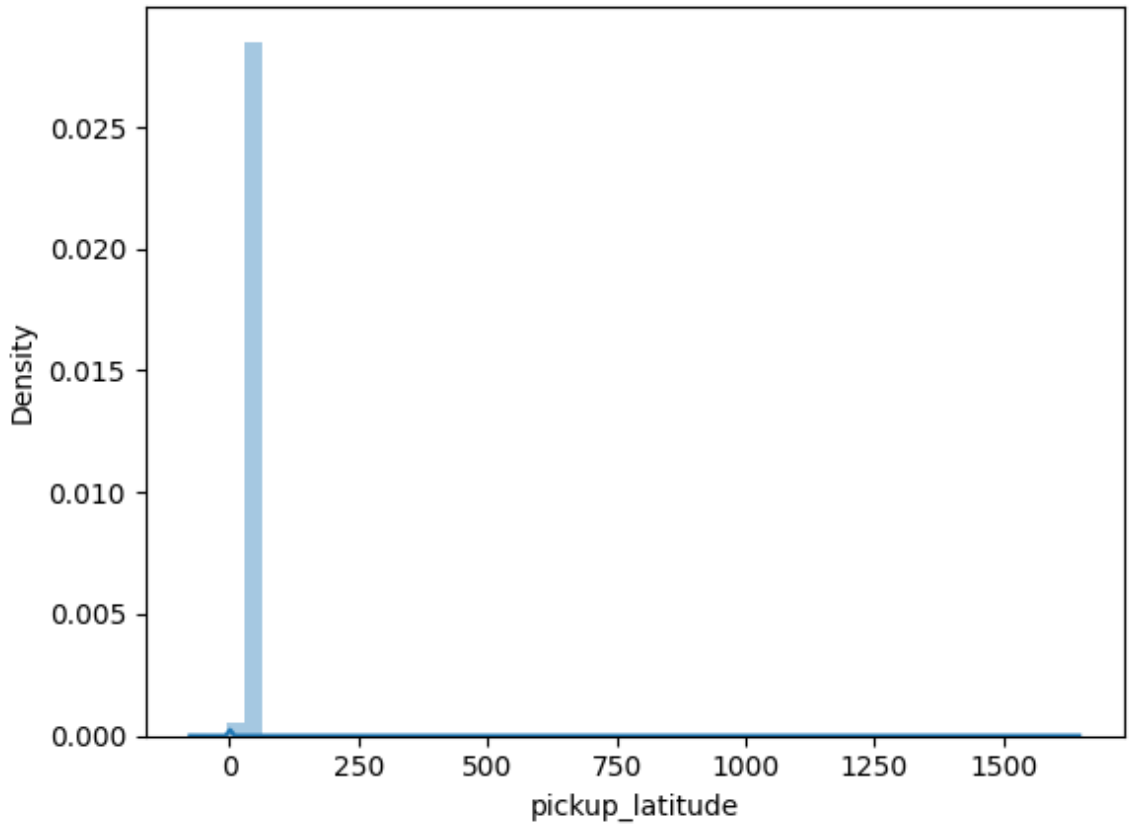
In [12]:
```
sb.distplot(df['fare_amount'])
```

Out[12]:
```
<AxesSubplot: xlabel='fare_amount', ylabel='Density'>
```



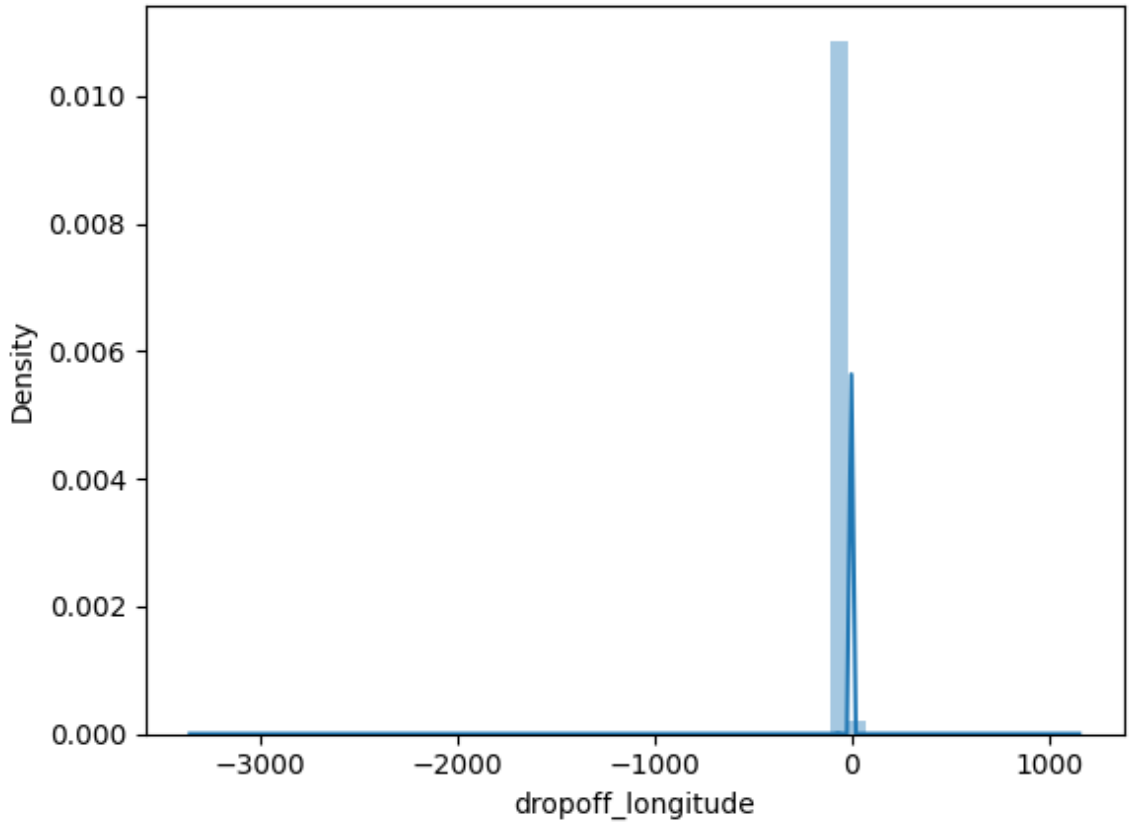
In [13]:
```
sb.distplot(df['pickup_latitude'])
```

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



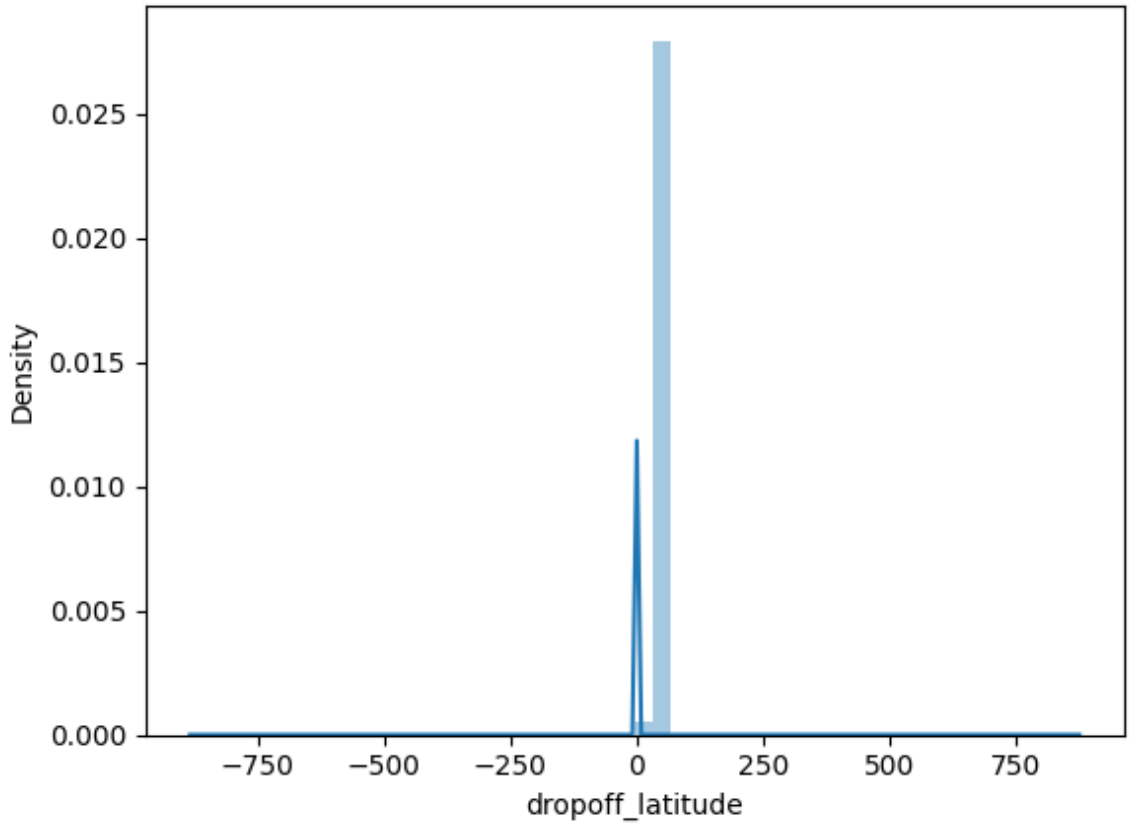
In [14]:
```
sb.distplot(df['dropoff_longitude'])
```

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



In [15]:
```
sb.distplot(df['dropoff_latitude'])
```

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



In [16]:
```
def find_outliers_IQR(df):
    q1=df.quantile(0.25)
    q3=df.quantile(0.75)
    IQR = q3-q1
```

```
    outliers=df[((df<(q1-1.5*IQR)) | (df>(q3+1.5*IQR)))]
    return outliers
```

In [17]:
```
outliers = find_outliers_IQR(df["fare_amount"])
print("number of outliers: "+ str(len(outliers)))
print("max outlier value: "+ str(outliers.max()))
print("min outlier value: "+ str(outliers.min()))
outliers
```

```
number of outliers: 17166
max outlier value: 499.0
min outlier value: -52.0
```

Out[17]:
```
6         24.50
30        25.70
34        39.50
39        29.00
48        56.80
          ...
199976    49.70
199977    43.50
199982    57.33
199985    24.00
199997    30.90
Name: fare_amount, Length: 17166, dtype: float64
```

In [18]:
```
#upper and lower limit which can be used for capping of outliers

upper_limit = df['fare_amount'].mean() + 3*df['fare_amount'].std()
print(upper_limit)
lower_limit = df['fare_amount'].mean() - 3*df['fare_amount'].std()
print(lower_limit)
```

```
41.06517154773827
-18.345388448822774
```

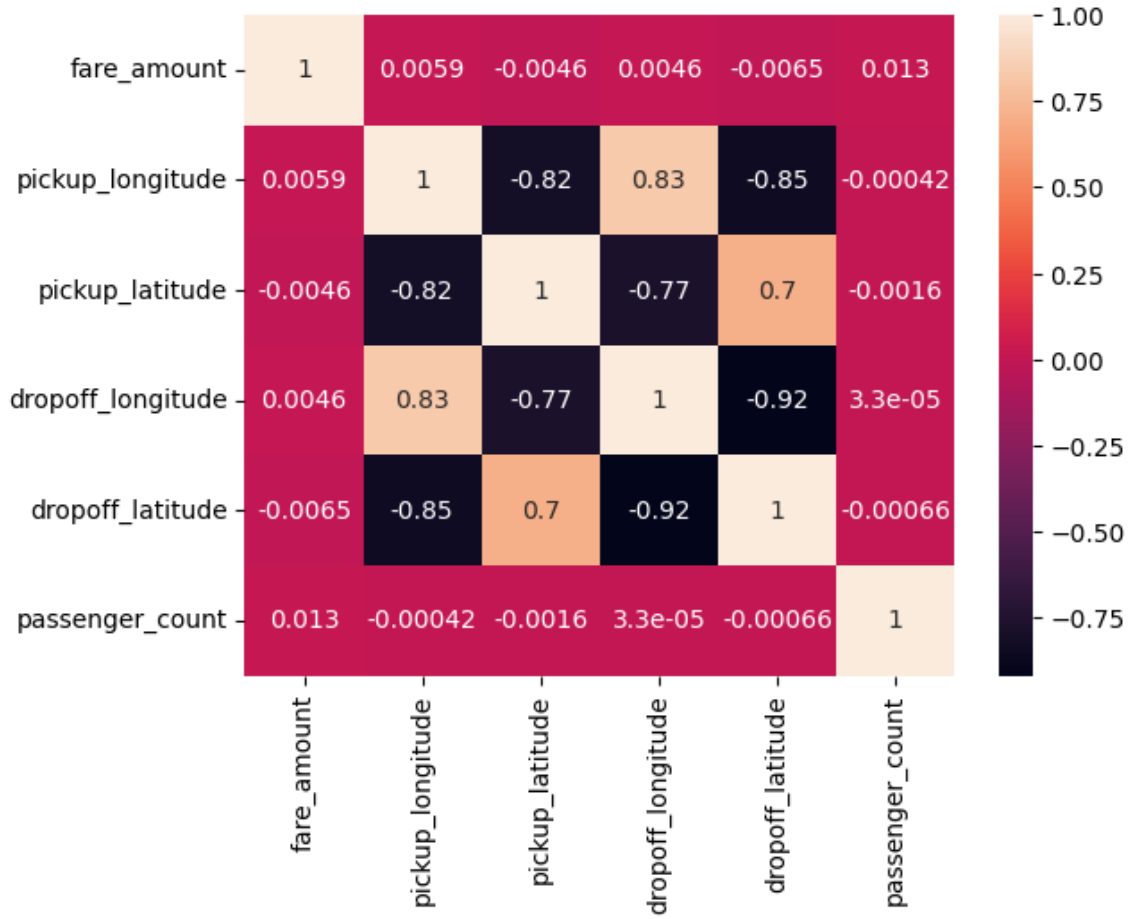
In [19]:
```
import numpy as np
df['fare_amount'] = np.where(
    df['fare_amount']>upper_limit,
    upper_limit,
    np.where(
        df['fare_amount']<lower_limit,
        lower_limit,
        df['fare_amount']
    )
)
df['fare_amount']
```

Out[19]:
```
0          7.5
1          7.7
2         12.9
3          5.3
4         16.0
          ...
199995     3.0
199996     7.5
199997    30.9
199998    14.5
199999    14.1
Name: fare_amount, Length: 199999, dtype: float64
```

3. Check the correlation.

In [20]:
```
corrMatrix=df.corr()
sb.heatmap(corrMatrix,annot=True)
```

Out[20]: <AxesSubplot: >



In [21]:
```
df= df.assign(hour = df.pickup_datetime.dt.hour, day= df.pickup_datetime.dt.day,
df.drop(['pickup_datetime'],inplace=True,axis=1)
```

4. Implement linear regression and random forest regression models.

In [22]:
```
X=df.drop(['fare_amount'],axis=1)
Y=df['fare_amount']
from sklearn.model_selection import train_test_split
X_train,X_test,Y_train,Y_test=train_test_split(X,Y,test_size=0.3,random_state=2)
```

In [23]:
```
from sklearn.linear_model import LinearRegression
regression = LinearRegression()
```

In [24]:
```
regression.fit(X_train,Y_train)
```

Out[24]:
```
▾ LinearRegression
LinearRegression()
```

In [25]:
```
from sklearn import metrics
y_pred = regression.predict(X_test)
rmse=np.sqrt(metrics.mean_squared_error(Y_test,y_pred))
print("RMSE of Linear Regression: ",rmse)
```

```
RMSE of Linear Regression:  8.002288285378782
```

In [26]:
```
from sklearn.ensemble import RandomForestRegressor
rfModel=RandomForestRegressor(n_estimators=100, random_state=101)
```

In [27]:
```
rfModel.fit(X_train,Y_train)
y_pred=rfModel.predict(X_test)
y_pred
```

Out[27]:
```
array([ 6.708,  6.255, 13.09 , ..., 10.283, 11.566, 11.642])
```

In [28]:
```
rmse=np.sqrt(metrics.mean_squared_error(Y_test,y_pred))
print("RMSE of Random Forest Regression: ",rmse)
```

```
RMSE of Random Forest Regression:  3.387158421541573
```