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

In [ ]:
```
df =pd.read_csv("uber.csv")
```

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

Out[ ]:

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

In [ ]:
```
df = df.drop(['Unnamed: 0', 'key'],axis=1)
```

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

```
<ipython-input-9-afdf1a420355>:1: FutureWarning: A value is trying to be set on a co
py of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


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

In [ ]:
```
df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = Tru
```

```
<ipython-input-10-16eb94382054>:1: FutureWarning: A value is trying to be set on a c
opy of a DataFrame or Series through chained assignment using an inplace method.
The behavior will change in pandas 3.0. This inplace method will never work because
the intermediate object on which we are setting values always behaves as a copy.

For example, when doing 'df[col].method(value, inplace=True)', try using 'df.method
({col: value}, inplace=True)' or df[col] = df[col].method(value) instead, to perform
the operation inplace on the original object.


  df['dropoff_longitude'].fillna(value=df['dropoff_longitude'].median(),inplace = Tr
ue)
```

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

Out[ ]:

	fare_amount	pickup_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dr
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 [ ]:
```
df.isna().sum()
```

Out[ ]:

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

In [ ]:
```
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 [ ]:
```
df = df.drop('pickup_datetime',axis=1)
```

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

Out[ ]:

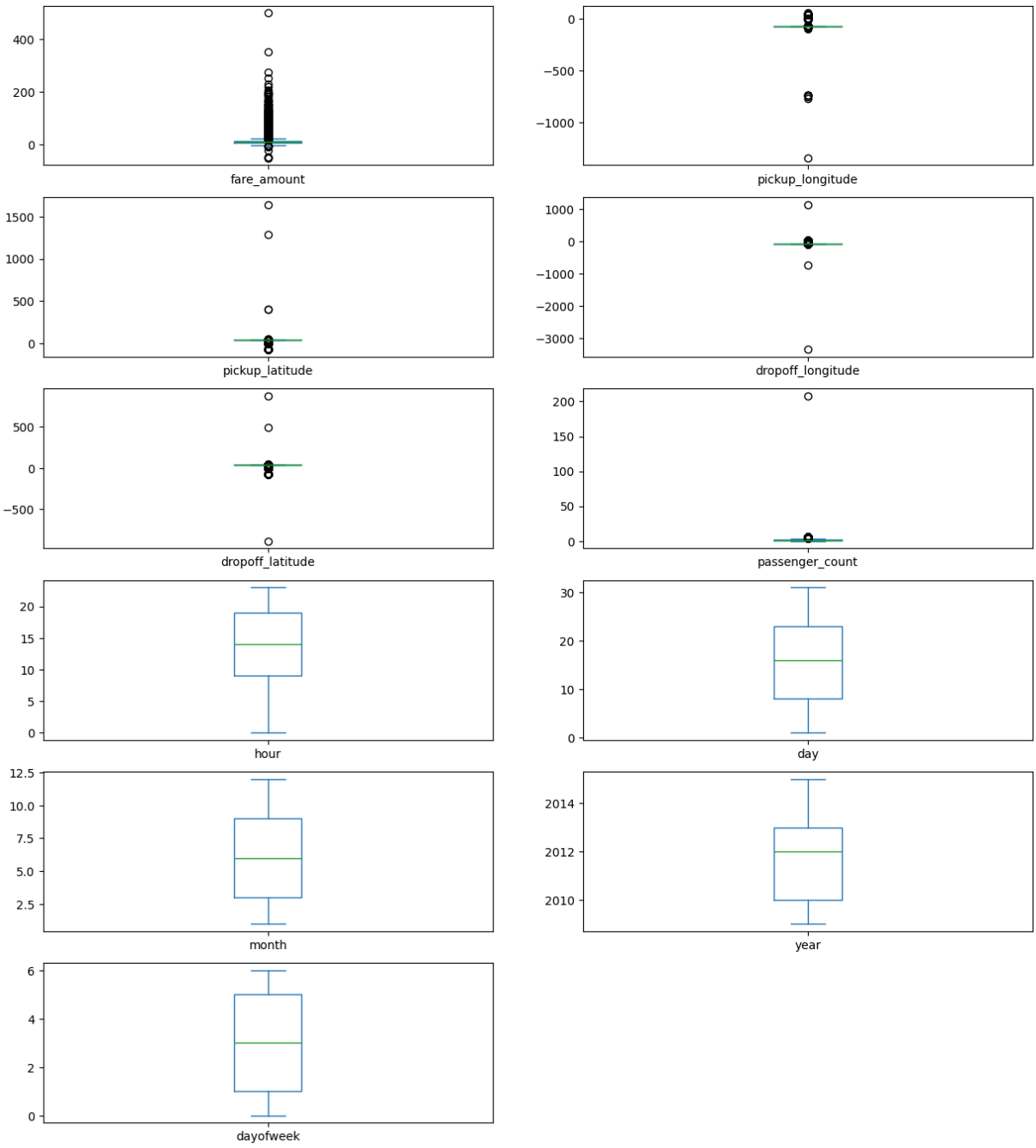
	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
0	7.5	-73.999817	40.738354	-73.999512	40.723217	
1	7.7	-73.994355	40.728225	-73.994710	40.750325	
2	12.9	-74.005043	40.740770	-73.962565	40.772647	
3	5.3	-73.976124	40.790844	-73.965316	40.803349	
4	16.0	-73.925023	40.744085	-73.973082	40.761247	

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

Out[ ]:

	0
fare_amount	Axes(0.125,0.786098;0.352273x0.0939024)
pickup_longitude	Axes(0.547727,0.786098;0.352273x0.0939024)
pickup_latitude	Axes(0.125,0.673415;0.352273x0.0939024)
dropoff_longitude	Axes(0.547727,0.673415;0.352273x0.0939024)
dropoff_latitude	Axes(0.125,0.560732;0.352273x0.0939024)
passenger_count	Axes(0.547727,0.560732;0.352273x0.0939024)
hour	Axes(0.125,0.448049;0.352273x0.0939024)
day	Axes(0.547727,0.448049;0.352273x0.0939024)
month	Axes(0.125,0.335366;0.352273x0.0939024)
year	Axes(0.547727,0.335366;0.352273x0.0939024)
dayofweek	Axes(0.125,0.222683;0.352273x0.0939024)

dtype: object



In [ ]:
```
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
```

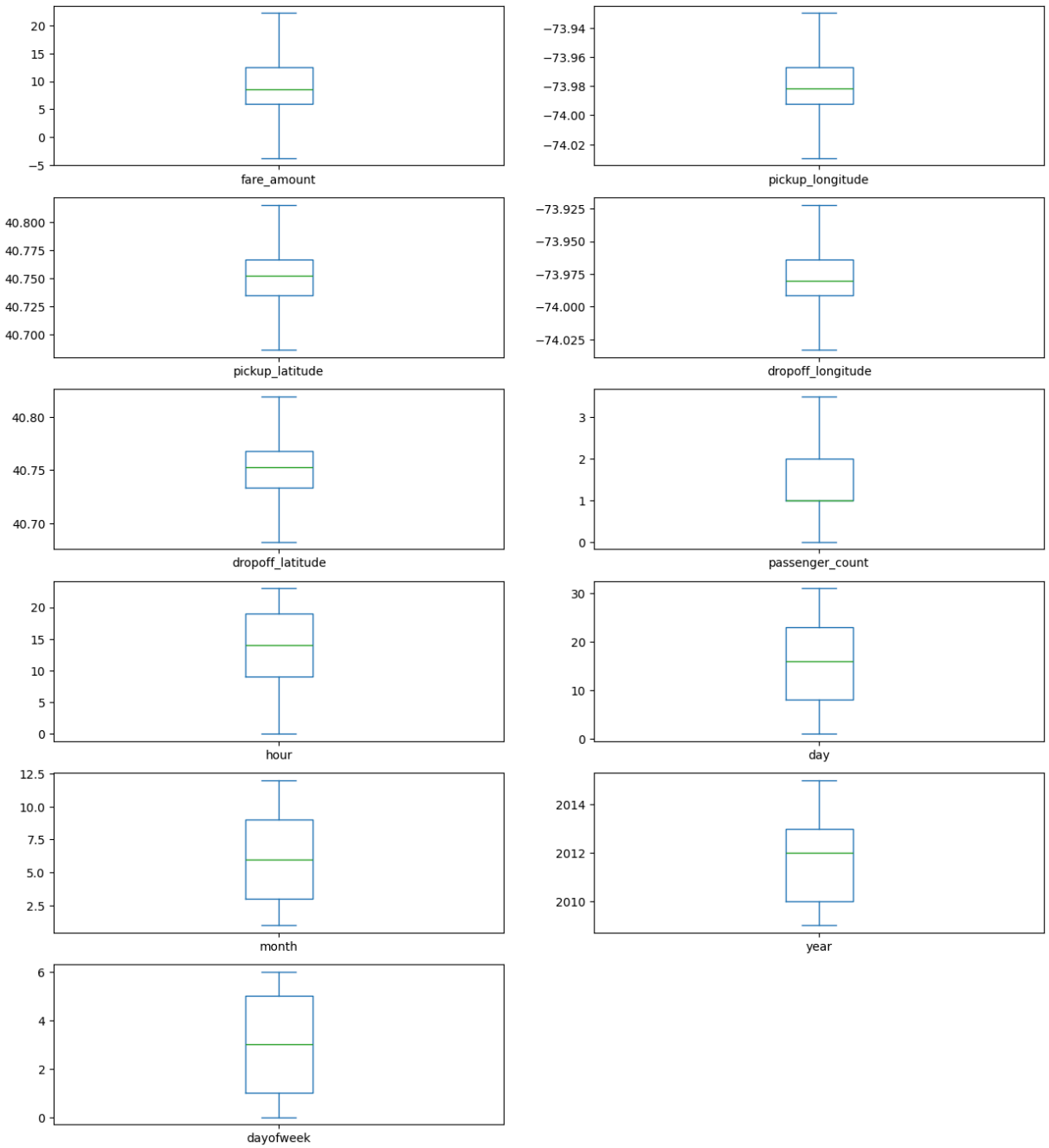
In [ ]:
```
df = treat_outliers_all(df , df.iloc[: , 0::])
```

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

Out[ ]:

	0
fare_amount	Axes(0.125,0.786098;0.352273x0.0939024)
pickup_longitude	Axes(0.547727,0.786098;0.352273x0.0939024)
pickup_latitude	Axes(0.125,0.673415;0.352273x0.0939024)
dropoff_longitude	Axes(0.547727,0.673415;0.352273x0.0939024)
dropoff_latitude	Axes(0.125,0.560732;0.352273x0.0939024)
passenger_count	Axes(0.547727,0.560732;0.352273x0.0939024)
hour	Axes(0.125,0.448049;0.352273x0.0939024)
day	Axes(0.547727,0.448049;0.352273x0.0939024)
month	Axes(0.125,0.335366;0.352273x0.0939024)
year	Axes(0.547727,0.335366;0.352273x0.0939024)
dayofweek	Axes(0.125,0.222683;0.352273x0.0939024)

dtype: object



In [ ]:
```
pip install haversine
```

```
Collecting haversine
  Downloading haversine-2.8.1-py2.py3-none-any.whl.metadata (5.9 kB)
Downloading haversine-2.8.1-py2.py3-none-any.whl (7.7 kB)
Installing collected packages: haversine
Successfully installed haversine-2.8.1
```

In [ ]:
```
import haversine as hs  # Import the Haversine library for distance calculation

# Initialize an empty list to store distances
travel_dist = []

# Loop through each row in the DataFrame (assuming pickup and dropoff points are at
for pos in range(len(df['pickup_longitude'])):
    # Extract longitude and latitude for both pickup and dropoff locations
```

```
    long1, lati1, long2, lati2 = [
        df['pickup_longitude'][pos],
        df['pickup_latitude'][pos],
        df['dropoff_longitude'][pos],
        df['dropoff_latitude'][pos]
    ]

    # Create tuples representing the coordinates of the pickup and dropoff location
    loc1 = (lati1, long1)
    loc2 = (lati2, long2)

    # Calculate the distance between pickup and dropoff points using Haversine
    c = hs.haversine(loc1, loc2)  # Distance in kilometers by default

    # Append the calculated distance to the travel_dist list
    travel_dist.append(c)

# Add the calculated distances as a new column in the DataFrame
df['dist_travel_km'] = travel_dist

# Display the first few rows to verify
df.head()
```

Out[ ]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	pas
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 [ ]:

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

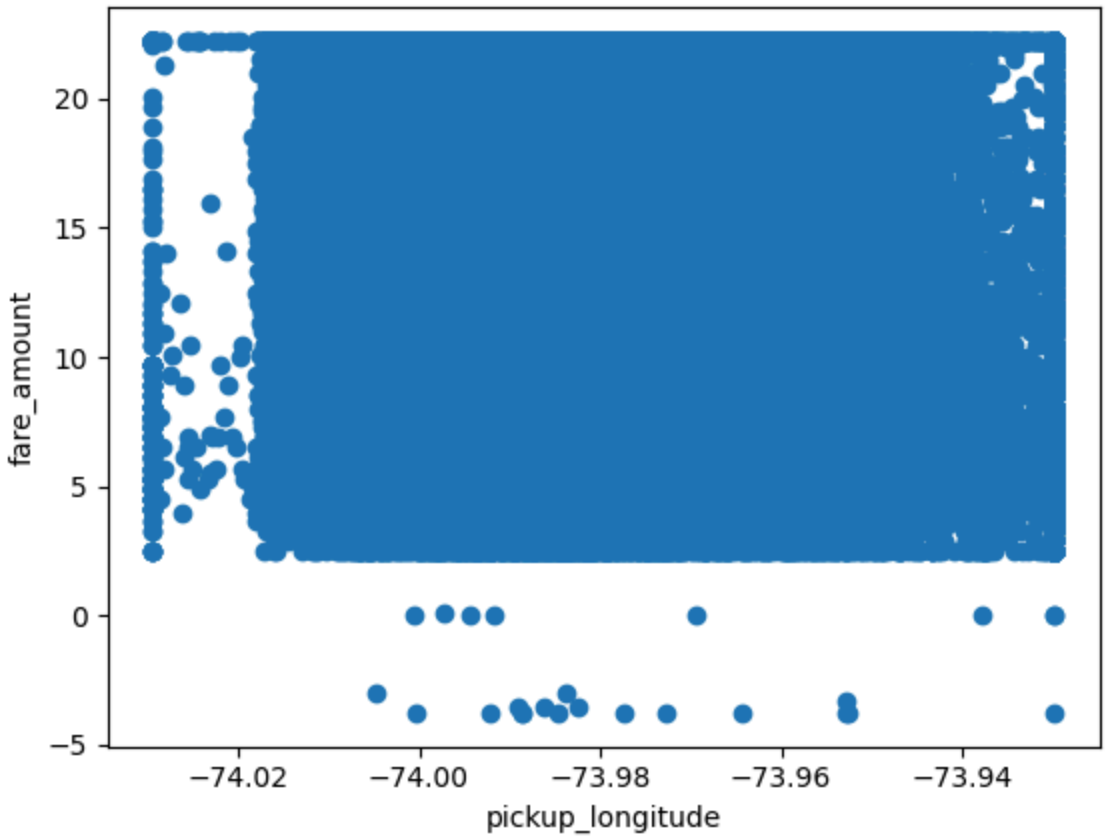
Out[ ]:

	0
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

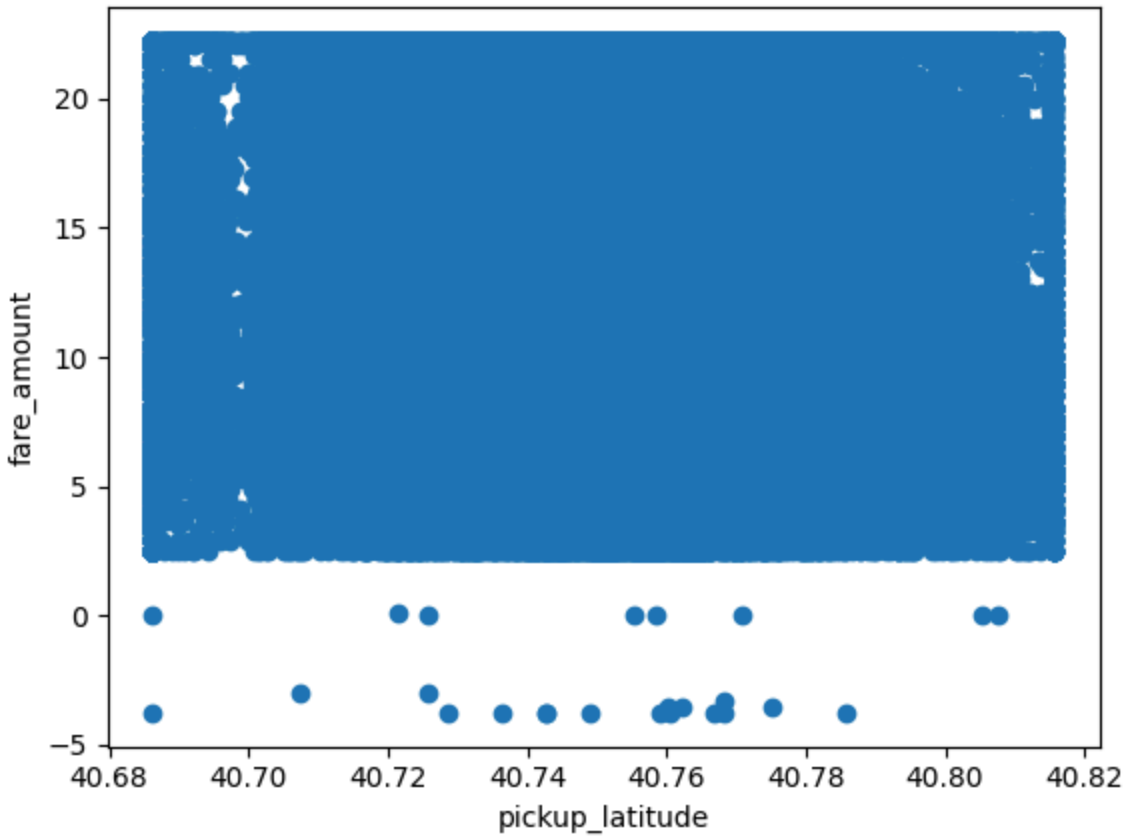
In [ ]:
```
plt.scatter(df['pickup_longitude'], df['fare_amount'])
plt.xlabel("pickup_longitude")
plt.ylabel("fare_amount")
```

Out[ ]:
```
Text(0, 0.5, 'fare_amount')
```



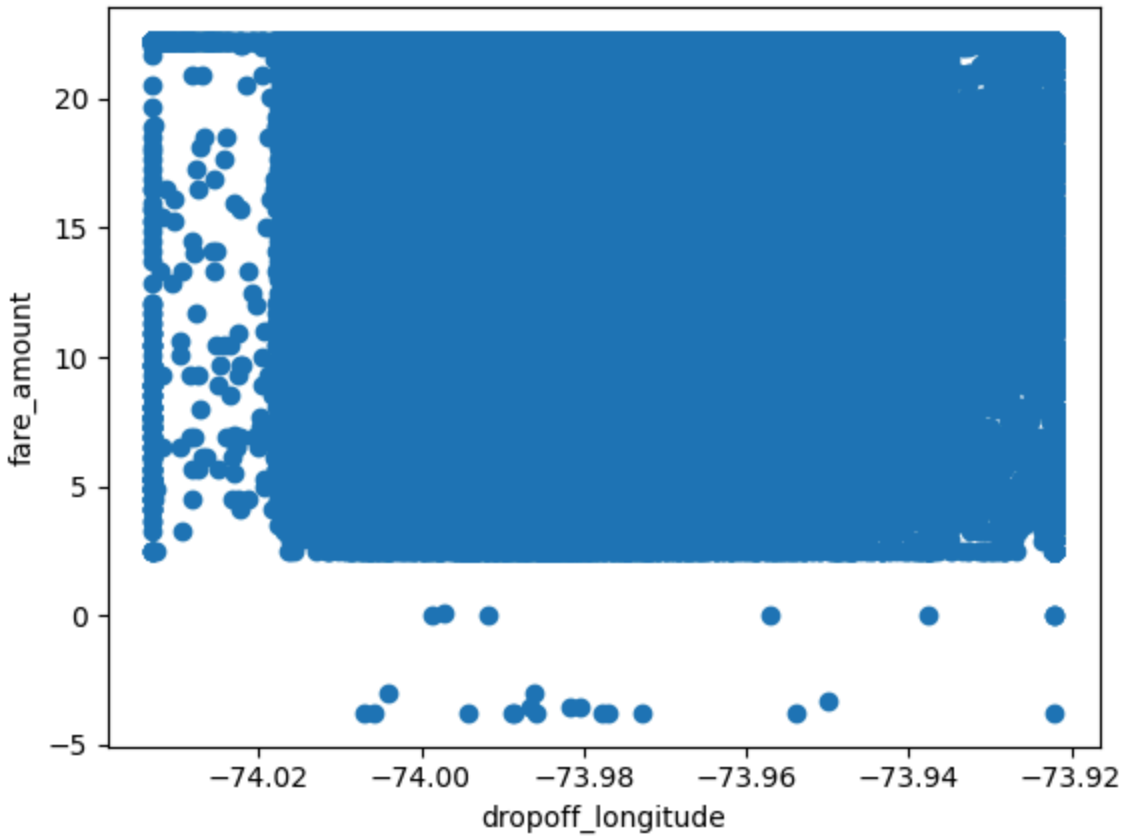
In [ ]:
```
plt.scatter(df['pickup_latitude'], df['fare_amount'])
plt.xlabel("pickup_latitude")
plt.ylabel("fare_amount")
```

Out[ ]:
```
Text(0, 0.5, 'fare_amount')
```



In [ ]:
```
plt.scatter(df['dropoff_longitude'], df['fare_amount'])
plt.xlabel("dropoff_longitude")
plt.ylabel("fare_amount")
```

Out[ ]:
```
Text(0, 0.5, 'fare_amount')
```



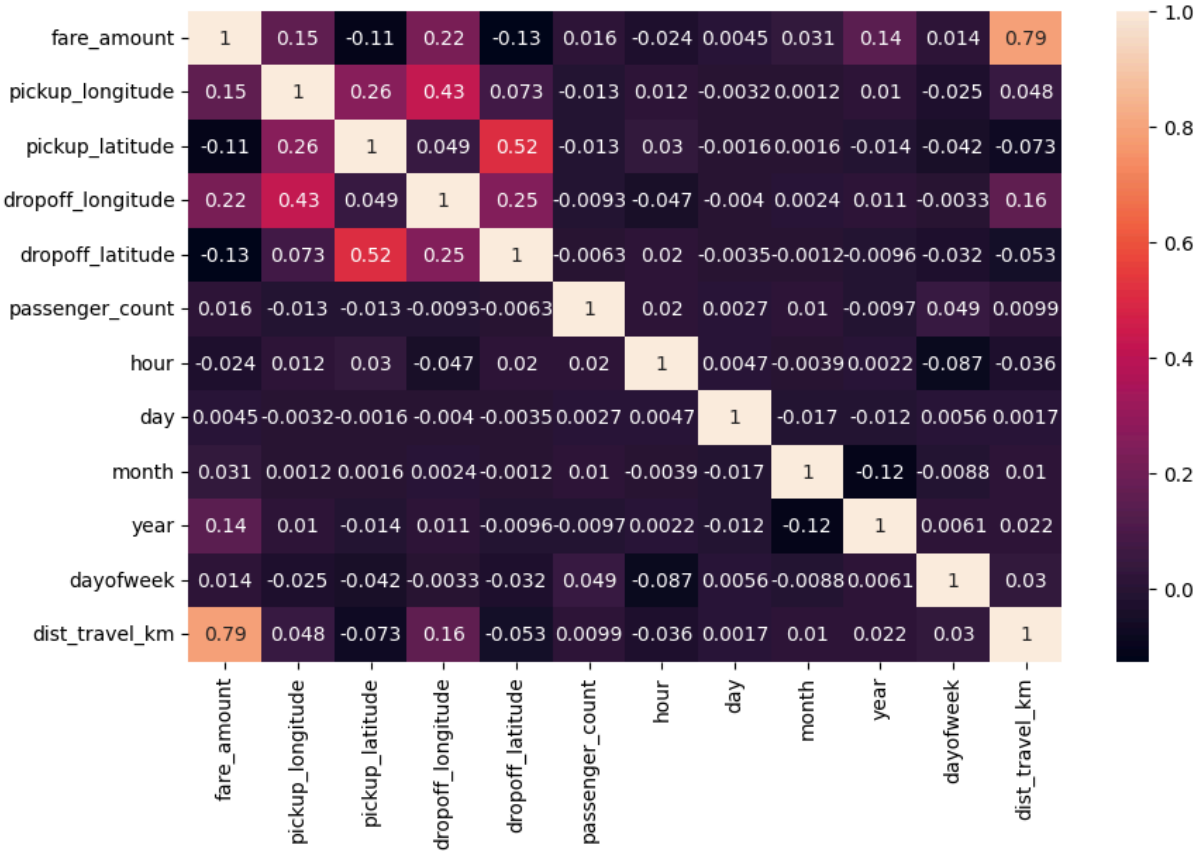
In [ ]:
```
corr = df.corr() #Function to find the correlation
corr_matrix
```

Out[ ]:

	fare_amount	pickup_longitude	pickup_latitude	dropoff_longitude	drop
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	
dropoff_latitude	-0.125898	0.073290	0.515714	0.245667	
passenger_count	0.015778	-0.013213	-0.012889	-0.009303	
hour	-0.023623	0.011579	0.029681	-0.046558	
day	0.004534	-0.003204	-0.001553	-0.004007	
month	0.030817	0.001169	0.001562	0.002391	
year	0.141277	0.010198	-0.014243	0.011346	
dayofweek	0.013652	-0.024652	-0.042310	-0.003336	
dist_travel_km	0.786385	0.048446	-0.073362	0.155191	

In [ ]:
```
fig,axis = plt.subplots(figsize = (10,6))
sns.heatmap(df.corr(),annot = True)
```

Out[ ]: <Axes: >



In [ ]:
```
x = df[['pickup_longitude','pickup_latitude','dropoff_longitude','dropoff_latitude'
```

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

In [ ]:
```
from sklearn.model_selection import train_test_split
X_train,X_test,y_train,y_test = train_test_split(x,y,test_size = 0.33)
```

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

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

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

In [ ]:
```
regression.intercept_
```

Out[ ]: 3610.147988157156

In [ ]:
```
regression.coef_
```

Out[ ]:
```
array([ 2.53813281e+01, -6.87626880e+00,  1.96984800e+01, -1.79374170e+01,
        7.39830690e-02,  5.21400446e-03,  2.63533552e-03,  5.93819545e-02,
        3.67996369e-01, -3.05631538e-02,  1.84623656e+00])
```

In [ ]:
```
prediction = regression.predict(X_test)
```

In [ ]:
```
print(prediction)
```

```
[ 6.47689425  6.84485687  7.31453612 ...  6.43848705 16.45651411
  6.97078372]
```

In [ ]:
```
y_test
```

Out[ ]:

	fare_amount
48247	7.3
181447	6.1
60251	5.3
89453	5.7
12827	17.0
...	...
119159	4.5
183359	8.1
186896	16.0
119361	21.3
11986	6.5

66000 rows × 1 columns

dtype: float64

In [ ]:
```
from sklearn.metrics import r2_score
r2_score(y_test,prediction)
```

Out[ ]:
```
0.6626598908424755
```

In [ ]:
```
from sklearn.metrics import mean_squared_error
MSE = mean_squared_error(y_test,prediction)
MSE
```

Out[ ]:
```
9.938457733483709
```

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

Out[ ]: 3.152531955981368

In [ ]:
```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=100)
```

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

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

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

Out[ ]: array([6.761 , 5.72 , 6.842 , ..., 8.009 , 19.3045, 6.51])

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

Out[ ]: 2.4671486279718593

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

Out[ ]: 6.086822352503428

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

Out[ ]: 2.4671486279718593

In [ ]: