This dataset was imported from NYC taxi (url:https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page (https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page))

Data-directory: https://www.nyc.gov/assets/tlc/downloads/pdf/ (https://www.nyc.gov/assets/tlc/downloads/pdf

The dataset relates to taxi trips in NY, February 2016

But this dates are

2016-01-03 [00:00:00 - 23:59:00]--> Tuesday

2016-02-03 [00:00:00 - 06:14:00] --> Wednesday

2016-10-03 [07:08:00 - 18:32:00] --> Thursday

2016-11-03 [00:00:00 - 14:19:00] --> Friday

One line of Data represents Taxi Data

What is PARQUET

What is range of Lat Log of the Two Airports??

Parallel vs Non-parallel computation time comparision

How to do parallel computation, What libraries to import , how to implement

Report: https://www.nyc.gov/site/tlc/about/industry-

[reports.page (https://www.nyc.gov/site/tlc/about/industry-reports.page)](https://www.nyc.gov/site/tlc/about/industry-reports.page)

What is %%timeit Magic function??

Data upload permissions on GPU -- linux

Other Source : [https://medium.com/analytics-vidhya/taxi-demand-prediction-on-time-series-data-with-holt-winter-forecasting-loss-0-02-2bcdeec48499 (https://medium.com/analytics-vidhya/taxi-demand-prediction-on-time-series-data-with-holt-winter-forecasting-loss-0-02-2bcdeec48499)](https://medium.com/analytics-vidhya/taxi-demand-prediction-on-time-series-data-with-holt-winter-forecasting-loss-0-02-2bcdeec48499)

Objective: How to reduce waiting time for potential customer

Identify all locations that have more than 10 pickups in 15 minutes slots

In [7]:
```
import numpy as np
import pandas as pd
from scipy import stats
import statsmodels.api as sm

```

In [8]:
```
import seaborn as sns
import matplotlib.pyplot as plt
import statistics
```

In [9]:
```
taxi_data=pd.read_csv(r"C:\Users\Sanjoy\Desktop\Bittu's File\SEM V\AI\
```

In [6]:
```
taxi_data.shape
```

Out[6]: (100000, 19)

In [5]:
```
1 taxi_data.columns
```

Out[5]:
```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amo
unt',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount'],
      dtype='object')
```

Check if total amount is equal to fare_amount + extra + mta_tax + tip_amount + toll_amount + Imp_sur

In [6]:
```
1 taxi_data.head(20)
```

Out[6]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	1	01-03-2016 00:00	01-03-2016 00:07	1	2.50
1	1	01-03-2016 00:00	01-03-2016 00:11	1	2.90
2	2	01-03-2016 00:00	01-03-2016 00:31	2	19.98
3	2	01-03-2016 00:00	01-03-2016 00:00	3	10.78
4	2	01-03-2016 00:00	01-03-2016 00:00	5	30.43
5	2	01-03-2016 00:00	01-03-2016 00:00	5	5.92
6	2	01-03-2016 00:00	01-03-2016 00:00	6	5.72
7	1	01-03-2016 00:00	01-03-2016 00:16	1	6.20
8	1	01-03-2016 00:00	01-03-2016 00:05	1	0.70
9	2	01-03-2016 00:00	01-03-2016 00:24	3	7.18
10	2	01-03-2016 00:00	01-03-2016 00:02	2	0.54
11	1	01-03-2016 00:00	01-03-2016 00:07	1	1.70
12	1	01-03-2016 00:00	01-03-2016 00:03	1	1.10
13	2	01-03-2016 00:00	01-03-2016 00:09	1	2.10
14	2	01-03-2016 00:00	01-03-2016 00:24	1	8.54
15	2	01-03-2016 00:00	01-03-2016 00:08	1	2.00
16	1	01-03-2016 00:00	01-03-2016 00:09	1	3.20
17	2	01-03-2016 00:00	01-03-2016 00:08	1	1.59
18	2	01-03-2016 00:00	01-03-2016 00:32	3	16.81
19	1	01-03-2016 00:00	01-03-2016 00:03	2	0.50

In [7]:
```
1 taxi_data.describe()
```

Out[7]:

	VendorID	passenger_count	trip_distance	pickup_longitude	pickup_latitude	10(
count	100000.00000	100000.000000	100000.000000	100000.000000	100000.000000	
mean	1.88327	1.929170	3.034270	-73.288983	40.375220	
std	0.32110	1.589408	3.846951	7.089652	3.901413	
min	1.00000	0.000000	0.000000	-121.933327	0.000000	
25%	2.00000	1.000000	0.990000	-73.990959	40.738891	
50%	2.00000	1.000000	1.670000	-73.980202	40.755299	
75%	2.00000	2.000000	3.200000	-73.964203	40.769021	
max	2.00000	6.000000	184.400000	0.000000	41.204548	

In [8]:
```
1 taxi_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column                 Non-Null Count   Dtype
---  ------                 --------------   -----
 0   VendorID               100000 non-null  int64
 1   tpep_pickup_datetime   100000 non-null  object
 2   tpep_dropoff_datetime  100000 non-null  object
 3   passenger_count        100000 non-null  int64
 4   trip_distance          100000 non-null  float64
 5   pickup_longitude       100000 non-null  float64
 6   pickup_latitude        100000 non-null  float64
 7   RatecodeID             100000 non-null  int64
 8   store_and_fwd_flag     100000 non-null  object
 9   dropoff_longitude      100000 non-null  float64
 10  dropoff_latitude       100000 non-null  float64
 11  payment_type           100000 non-null  int64
 12  fare_amount            100000 non-null  float64
 13  extra                  100000 non-null  float64
 14  mta_tax                100000 non-null  float64
 15  tip_amount             100000 non-null  float64
 16  tolls_amount           100000 non-null  float64
 17  improvement_surcharge  100000 non-null  float64
 18  total_amount           100000 non-null  float64
dtypes: float64(12), int64(4), object(3)
memory usage: 14.5+ MB
```

In [9]:
```
desired_columns = [
    taxi_data.columns.get_loc('total_amount'),
    taxi_data.columns.get_loc('trip_distance'),
    taxi_data.columns.get_loc('tip_amount'),
    taxi_data.columns.get_loc('fare_amount'),
    taxi_data.columns.get_loc('tolls_amount'),
    taxi_data.columns.get_loc('extra'),
    taxi_data.columns.get_loc('mta_tax'),
    taxi_data.columns.get_loc('improvement_surcharge'),
    taxi_data.columns.get_loc('payment_type'),
    taxi_data.columns.get_loc('VendorID'),
    taxi_data.columns.get_loc('tpep_pickup_datetime'),
    taxi_data.columns.get_loc('tpep_dropoff_datetime'),
    taxi_data.columns.get_loc('pickup_longitude'),
    taxi_data.columns.get_loc('pickup_latitude'),
    taxi_data.columns.get_loc('RatecodeID'),
    taxi_data.columns.get_loc('store_and_fwd_flag'),
    taxi_data.columns.get_loc('dropoff_longitude'),
    taxi_data.columns.get_loc('dropoff_latitude'),
    taxi_data.columns.get_loc('passenger_count')

]

taxi_data_reordered = taxi_data.iloc[:, desired_columns]
```

In [10]:
```
taxi_data_reordered.head(10)
```

Out[10]:

	total_amount	trip_distance	tip_amount	fare_amount	tolls_amount	extra	mta_tax	impr
0	12.35	2.50	2.05	9.0	0.00	0.5	0.5	
1	15.35	2.90	3.05	11.0	0.00	0.5	0.5	
2	63.80	19.98	8.00	54.5	0.00	0.5	0.5	
3	41.62	10.78	3.78	31.5	5.54	0.0	0.5	
4	113.80	30.43	0.00	98.0	15.50	0.0	0.0	
5	30.36	5.92	5.06	23.5	0.00	1.0	0.5	
6	24.30	5.72	0.00	23.0	0.00	0.5	0.5	
7	21.80	6.20	0.00	20.5	0.00	0.5	0.5	
8	8.80	0.70	2.00	5.5	0.00	0.5	0.5	
9	28.00	7.18	3.20	23.5	0.00	0.5	0.5	

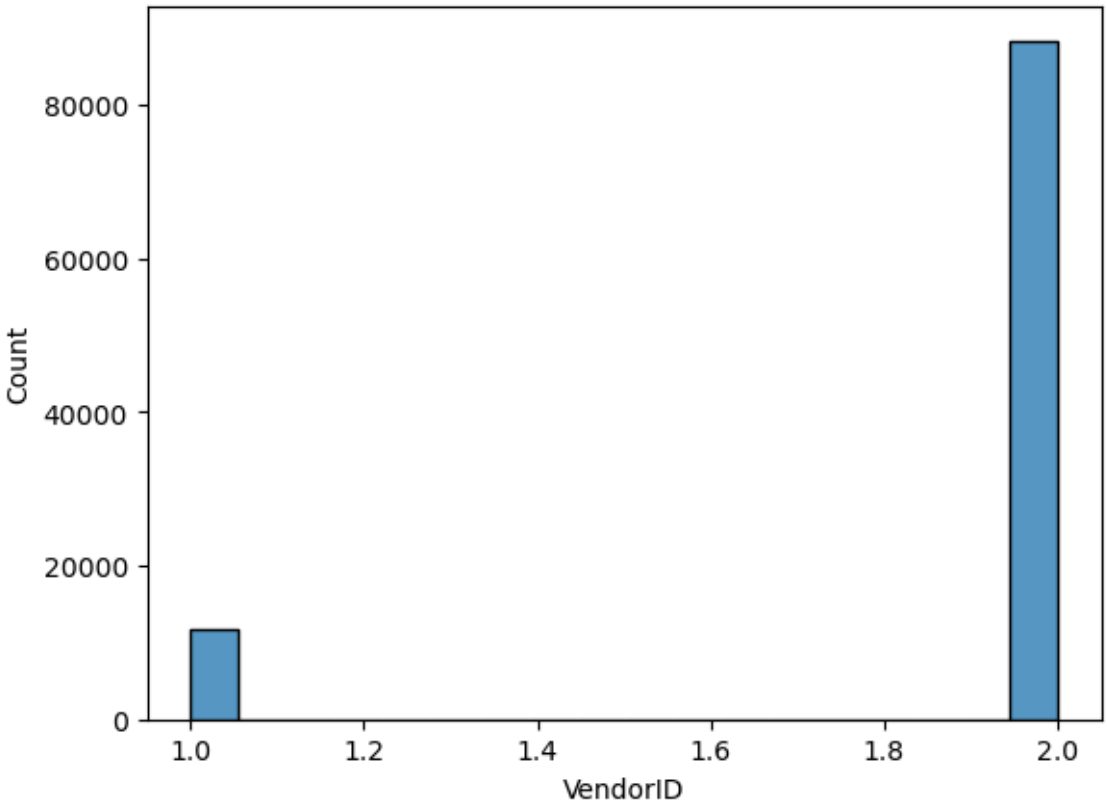
In [5]:
```
sns.histplot(taxi_data)
```

. . .

Pass the Feature's number in iloc

In [11]:
```
sns.histplot(taxi_data["VendorID"])
# Vendor-2 dominate
# what is vendor ?
```

Out[11]: <AxesSubplot:xlabel='VendorID', ylabel='Count'>



Vendor-2 Dominant

plt.axis([-74.9,-73.5, 0,50000]) -74.1, -73.7

In [ ]:
```
# sns.histplot(data=taxi_data, x="pickup_longitude", y="pickup_latitud
# plt.xlabel("Pickup Longitude")
# plt.ylabel("Pickup Latitude")
# sns.histplot(taxi_data["pickup_longitude"])
plt.hist(taxi_data["pickup_longitude"],bins = 100000);
plt.axis([-74.1,-73.7, 0,10000])
# plt.ylim(0,1000)
plt.show();
```

Logitude latitude , distance, What is 1 degree change in Latitude(How many meters/km ?) -- same longitude

One degree of latitude is more or less equal to 111.12 kilometers (or 111,120 meters or 69 miles)

one degree of longitude is more or less 111.32 kilometers (or 111,320 meters).

In [12]:
```
0.005 * 111
```

Out[12]:
```
0.555
```

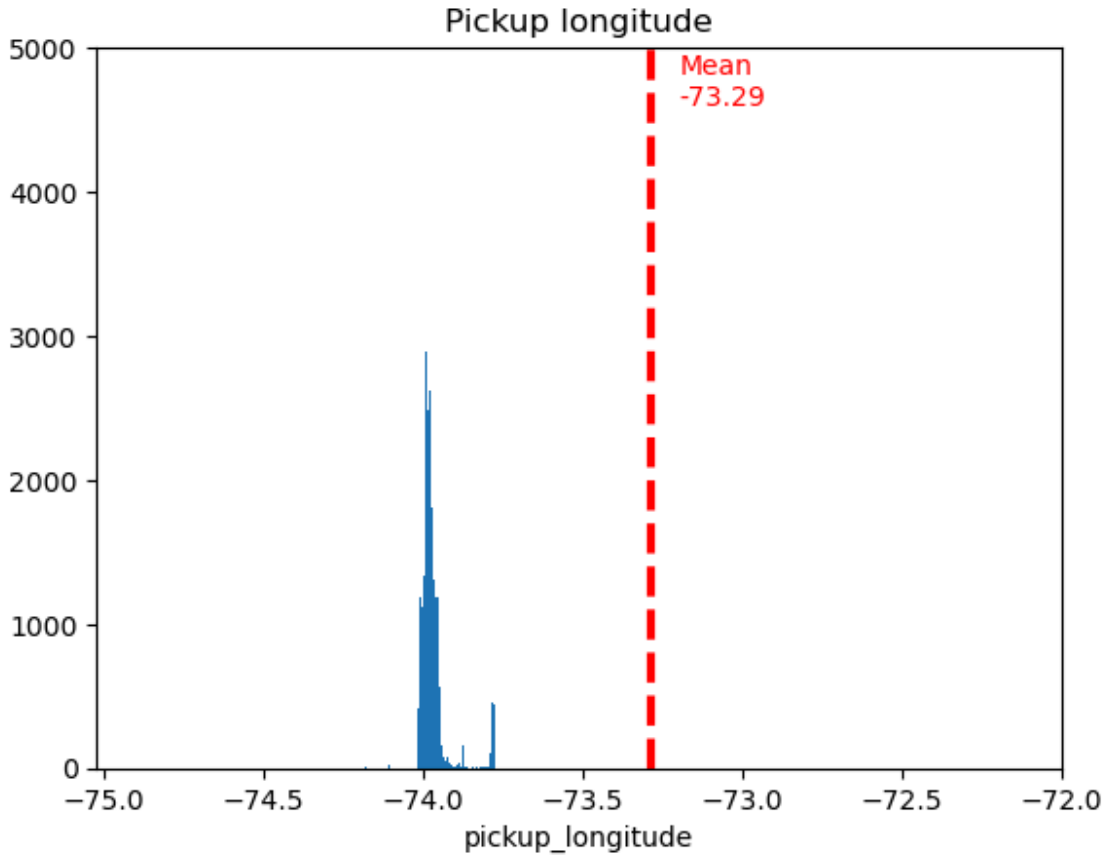
In [ ]:
```
plt.hist(taxi_data["pickup_longitude"],bins = 100000);
plt.axis([-74.07,-73.93, 0,10000])
plt.show();
```

In [4]:
```
mean_pickup_longitude = taxi_data["pickup_longitude"].mean()
mean_pickup_longitude
```

Out[4]:
```
-73.28898292891198
```

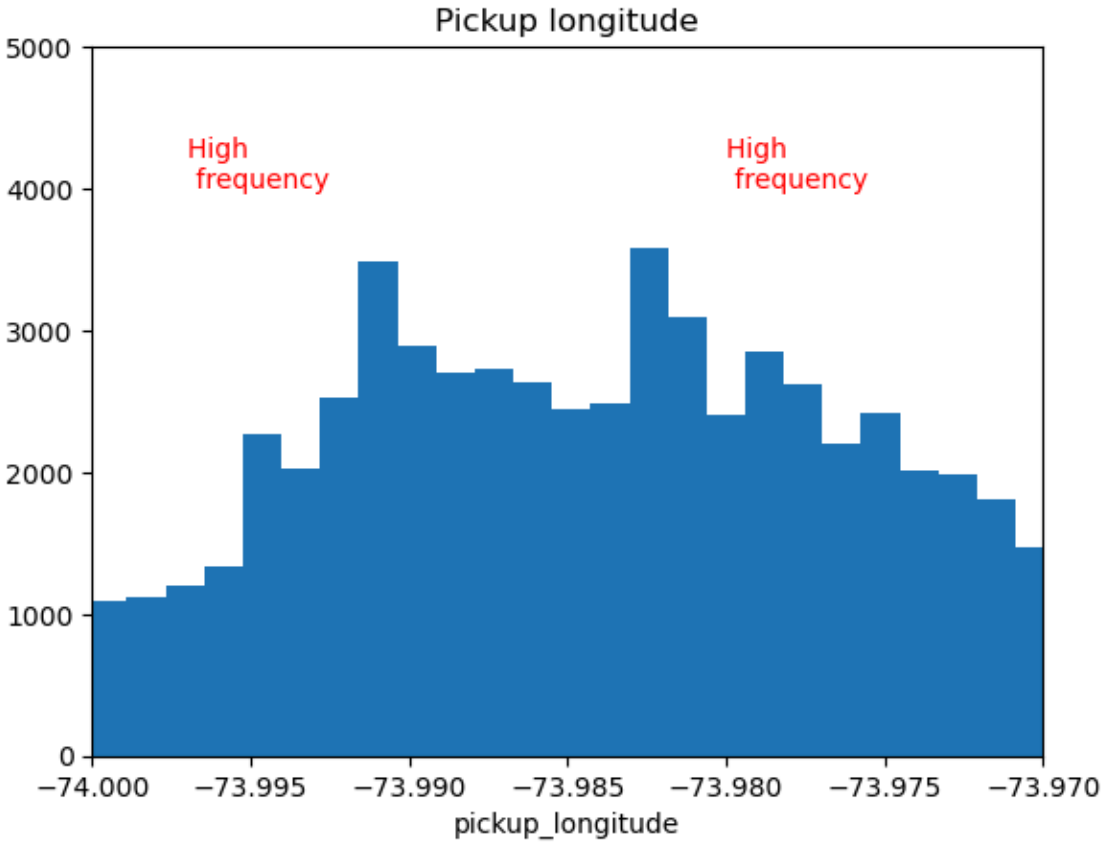
In [4]:

```
plt.hist(taxi_data["pickup_longitude"], bins=100000)

plt.axvline(x=taxi_data["pickup_longitude"].mean(), color="red", lines
mean_value = round(taxi_data["pickup_longitude"].mean(), 2)

plt.text(s=f'Mean\n{mean_value}', x= -73.2, y=4600, size=10, color='re
plt.axis([-75.02, -72.0, 0, 5000])

plt.xlabel("pickup_longitude")
plt.title("Pickup longitude")
plt.show()

```

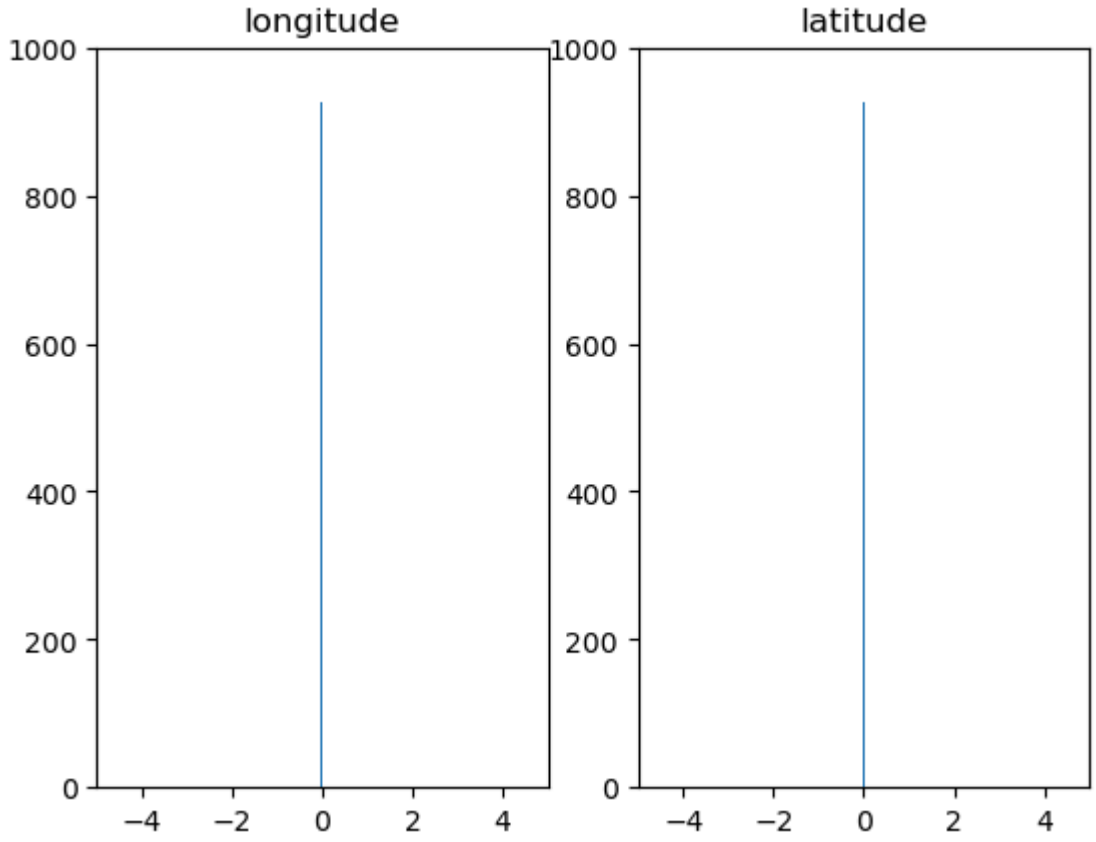


In [19]:

```
plt.hist(taxi_data["pickup_longitude"],bins = 100000);
plt.axis([-74.00,-73.97, 0,5000])
plt.text(s=f'High \n frequency', x= -73.990, y=3600, size=10, color='r
plt.text(s=f'High \n frequency', x= -73.983, y=4000, size=10, color='r
plt.title("Pickup longitude")
plt.xlabel("pickup_longitude")
plt.show();
```



In [14]:
```
plt.subplot(1,2,1)
plt.hist(taxi_data["pickup_longitude"],bins = 5000);
plt.axis([-5,5, 0,1000])
plt.title("longitude")
# plt.show()

plt.subplot(1,2,2)
plt.hist(taxi_data["pickup_latitude"],bins = 5000);
plt.axis([-5,5, 0,1000])
plt.title("latitude")
plt.show();
```



In [15]:
```
sum(taxi_data.pickup_latitude==taxi_data.pickup_longitude)
```

Out[15]: 925

In [ ]:
```

```

In [5]:
```
target_longitude = -73.9903717
matching_rows = taxi_data[taxi_data['pickup_longitude'] == target_long
```

In [6]:
```
matching_pickup_latitude = matching_rows['pickup_latitude']

print("Matching pickup latitude:")
print(matching_pickup_latitude)
```

```
Matching pickup latitude:
2982     40.756046
6424     40.751629
6713     40.756248
14244    40.750999
15019    40.756748
18931    40.746216
25322    40.746922
28113    40.750927
32873    40.745663
36121    40.714481
37396    40.731499
42118    40.756222
42588    40.728970
51044    40.756092
54833    40.756691
56213    40.730938
60558    40.740505
71289    40.714436
73024    40.725052
74314    40.731354
79804    40.686600
90512    40.756268
90739    40.756489
90893    40.756187
91180    40.756248
91278    40.756126
92836    40.755966
Name: pickup_latitude, dtype: float64
```

In [7]:
```
matching_row_nos = matching_pickup_latitude.index
```

In [ ]:
```
 np.hstack(dir(matching_pickup_latitude))
# type(pickup_latitudes)
```

In [ ]:
```

```

Initial Observation

Latitude:40.756046,Longitude:-73.9903717--> 265 W 40th St, New York, NY 10036, United States

In [ ]:
```

```

In [8]:

```
row_indices = [2982, 6424, 6713, 14244, 15019, 18931, 25322, 28113, 32
               37396, 42118, 42588, 51044, 54833, 56213, 60558, 71289,
               74314, 79804, 90512, 90739, 90893, 91180, 91278, 92836]

selected_rows = taxi_data.loc[row_indices]

print("Selected rows:")
print(selected_rows)
```

```
Selected rows:
       VendorID tpep_pickup_datetime tpep_dropoff_datetime  passenger_cou
nt  \
2982          2     10-03-2016 07:24      10-03-2016 07:36
1
6424          2     10-03-2016 07:43      10-03-2016 07:49
1
6713          2     10-03-2016 07:45      10-03-2016 07:52
2
14244         2     10-03-2016 08:22      10-03-2016 08:34
1
15019         2     10-03-2016 08:26      10-03-2016 08:37
1
18931         2     10-03-2016 08:47      10-03-2016 09:06
2
25322         2     10-03-2016 09:20      10-03-2016 09:29
1
28113         2     10-03-2016 09:36      10-03-2016 09:40
1
32873         2     10-03-2016 10:04      10-03-2016 10:09
5
36121         2     10-03-2016 10:24      10-03-2016 10:36
1
37396         2     10-03-2016 10:32      10-03-2016 10:49
1
42118         2     10-03-2016 11:02      10-03-2016 11:10
1
42588         2     10-03-2016 11:05      10-03-2016 11:22
1
51044         2     10-03-2016 11:58      10-03-2016 12:16
1
54833         2     10-03-2016 12:20      10-03-2016 12:31
1
56213         2     10-03-2016 12:29      10-03-2016 12:52
1
60558         2     10-03-2016 12:52      10-03-2016 12:57
1
71289         1     01-03-2016 00:20      01-03-2016 00:32
3
73024         2     01-03-2016 00:36      01-03-2016 00:49
6
74314         1     01-03-2016 00:48      01-03-2016 00:51
1
79804         2     01-03-2016 01:19      01-03-2016 01:31
1
90512         1     01-03-2016 05:38      01-03-2016 05:47
1
90739         2     01-03-2016 05:41      01-03-2016 05:43
1
90893         2     01-03-2016 05:43      01-03-2016 05:48
1
91180         2     01-03-2016 05:46      01-03-2016 05:51
3
91278         1     01-03-2016 05:47      01-03-2016 05:52
1
92836         1     01-03-2016 06:04      01-03-2016 06:10
1

       trip_distance  pickup_longitude  pickup_latitude  RatecodeID  \
2982            1.27        -73.990372        40.756046           1
6424            0.89        -73.990372        40.751629           1
```

6713	1.01	-73.990372	40.756248	1
14244	1.36	-73.990372	40.750999	1
15019	1.54	-73.990372	40.756748	1
18931	1.95	-73.990372	40.746216	1
25322	0.79	-73.990372	40.746922	1
28113	0.93	-73.990372	40.750927	1
32873	0.79	-73.990372	40.745663	1
36121	2.93	-73.990372	40.714481	1
37396	1.43	-73.990372	40.731499	1
42118	0.97	-73.990372	40.756222	1
42588	2.13	-73.990372	40.728970	1
51044	4.60	-73.990372	40.756092	1
54833	1.00	-73.990372	40.756691	1
56213	2.28	-73.990372	40.730938	1
60558	0.64	-73.990372	40.740505	1
71289	6.80	-73.990372	40.714436	1
73024	3.44	-73.990372	40.725052	1
74314	0.60	-73.990372	40.731354	1
79804	4.37	-73.990372	40.686600	1
90512	2.30	-73.990372	40.756268	1
90739	0.58	-73.990372	40.756489	1
90893	1.18	-73.990372	40.756187	1
91180	1.07	-73.990372	40.756248	1
91278	0.80	-73.990372	40.756126	1
92836	1.10	-73.990372	40.755966	1

	store_and_fwd_flag	dropoff_longitude	dropoff_latitude	payment_type
2982	N	-73.971786	40.754574	1
6424	N	-74.003075	40.756336	1
6713	N	-73.976158	40.756859	1
14244	N	-73.972656	40.746361	1
15019	N	-73.970253	40.762951	2
18931	N	-73.969604	40.759541	1
25322	N	-73.998611	40.753540	1
28113	N	-73.993225	40.740726	2
32873	N	-73.993858	40.738590	2
36121	N	-74.013840	40.715569	1
37396	N	-73.978996	40.750275	1
42118	N	-73.978966	40.762115	2
42588	N	-73.994003	40.751240	2
51044	N	-74.017006	40.704830	1
54833	N	-73.984261	40.748760	1
56213	N	-73.974380	40.759876	2
60558	N	-73.993912	40.732872	

```
1
71289                  N        -73.891869        40.746826
2
73024                  N        -73.991562        40.758957
1
74314                  N        -73.982964        40.731594
2
79804                  N        -73.932457        40.717739
1
90512                  N        -73.990143        40.734352
1
90739                  N        -73.985001        40.762123
1
90893                  N        -73.972397        40.753651
2
91180                  N        -73.975319        40.756100
1
91278                  N        -73.978523        40.754139
2
92836                  N        -73.975357        40.758064
1
```

	fare_amount	extra	mta_tax	tip_amount	tolls_amount	improvement_surcharge	total_amount
2982	8.5	0.0	0.5	12.00	0.0	0.3	21.30
6424	6.0	0.0	0.5	1.36	0.0	0.3	8.16
6713	6.5	0.0	0.5	1.00	0.0	0.3	8.30
14244	9.0	0.0	0.5	2.45	0.0	0.3	12.25
15019	8.5	0.0	0.5	0.00	0.0	0.3	9.30
18931	13.0	0.0	0.5	2.76	0.0	0.3	16.56
25322	7.0	0.0	0.5	1.56	0.0	0.3	9.36
28113	5.0	0.0	0.5	0.00	0.0	0.3	5.80
32873	5.5	0.0	0.5	0.00	0.0	0.3	6.30
36121	12.0	0.0	0.5	1.50	0.0	0.3	14.30
37396	11.5	0.0	0.5	2.00	0.0	0.3	14.30
42118	7.0	0.0	0.5	0.00	0.0	0.3	7.80
42588	12.0	0.0	0.5	0.00	0.0	0.3	12.80
51044	17.5	0.0	0.5	4.58	0.0	0.3	22.88
54833	8.0	0.0	0.5	2.64	0.0	0.3	11.44
56213	15.0	0.0	0.5	0.00	0.0	0.3	15.80
60558	5.0	0.0	0.5	1.16	0.0	0.3	6.96
71289	20.5	0.5	0.5	0.00	0.0	0.3	21.80
73024	12.5	0.5	0.5	2.76	0.0	0.3	16.56
74314	4.5	0.5	0.5	0.00	0.0	0.3	5.80
79804	14.5	0.5	0.5	2.00	0.0	0.3	17.80
90512	9.5	0.5	0.5	2.00	0.0	0.3	12.80
90739	4.0	0.5	0.5	1.32	0.0	0.3	6.62
90893	6.0	0.5	0.5	0.00	0.0	0.3	7.30
91180	5.5	0.5	0.5	1.36	0.0	0.3	8.16
91278	5.5	0.5	0.5	0.00	0.0	0.3	6.80
92836	6.5	0.0	0.5	1.46	0.0	0.3	8.76

36121	0.3	14.30
37396	0.3	14.30
42118	0.3	7.80
42588	0.3	12.80
51044	0.3	22.88
54833	0.3	11.44
56213	0.3	15.80
60558	0.3	6.96
71289	0.3	21.80
73024	0.3	16.56
74314	0.3	5.80
79804	0.3	17.80
90512	0.3	12.80
90739	0.3	6.62
90893	0.3	7.30
91180	0.3	8.16
91278	0.3	6.80
92836	0.3	8.76

Using iloc Rearrange variablesin this order: Total_amount, Trip_distance,

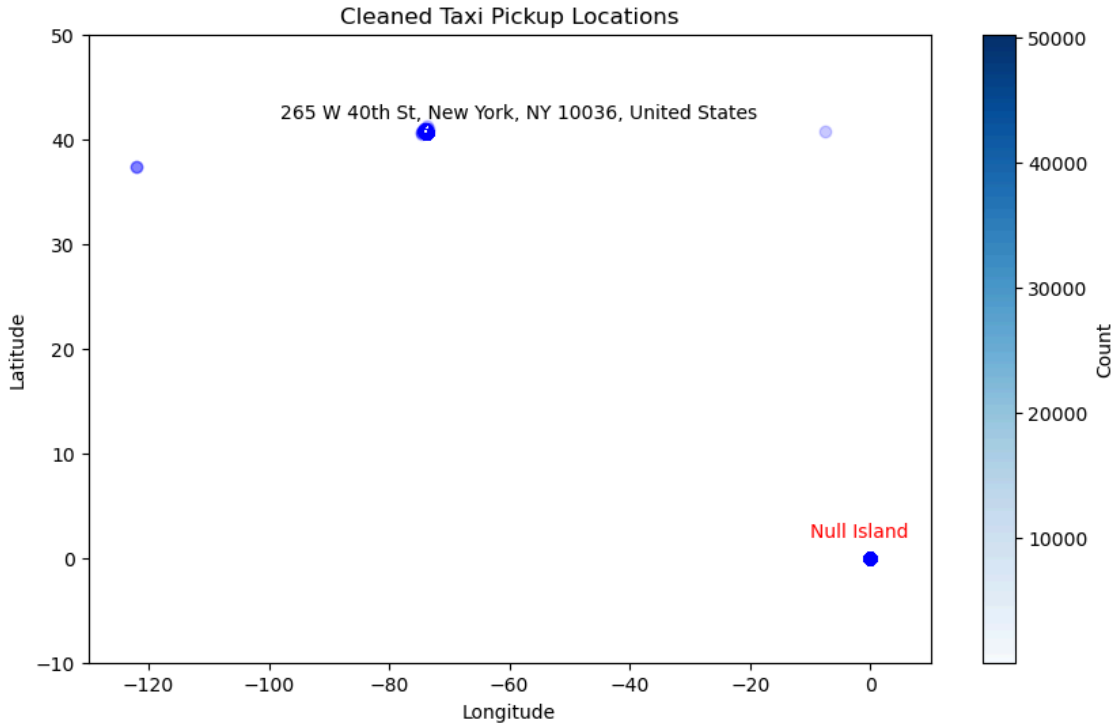
Histrograms of total_amount, fare_amount, trip_distance, tip_amount, toll_amount(bar-plot-toll: 0/non-zero). Histrogram of Non-zero toll_amounts(Mean).

Scatter Plot With Histogram For

pickup_lat, long ; drop_off lat,long

insert plt text (10,-20) -> Null

In [22]:
```
plt.figure(figsize=(10, 6))
plt.scatter(taxi_data['pickup_longitude'], taxi_data['pickup_latitude'


plt.hist2d(taxi_data['pickup_longitude'], taxi_data['pickup_latitude']
plt.colorbar(label='Count')
plt.text(s=f'Null Island', x=-10, y=2, size=10, color='red')
plt.text(s=f'265 W 40th St, New York, NY 10036, United States', x=-98,


plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Cleaned Taxi Pickup Locations')
plt.axis([-130,10,-10,50])
plt.show()
```



remove

Null Island is the location at zero degrees latitude and zero degrees longitude (0°N 0°E), i.e., where the prime meridian and the equator intersect. Since there is no landmass located at these coordinates, it is not an actual island

40 latitude and zero degrees longitude---> Spain

In [21]:
```
# Assuming you have a DataFrame named 'taxi_data'
cleaned_pickup_data = taxi_data[(taxi_data['pickup_longitude'] != 0) &
```

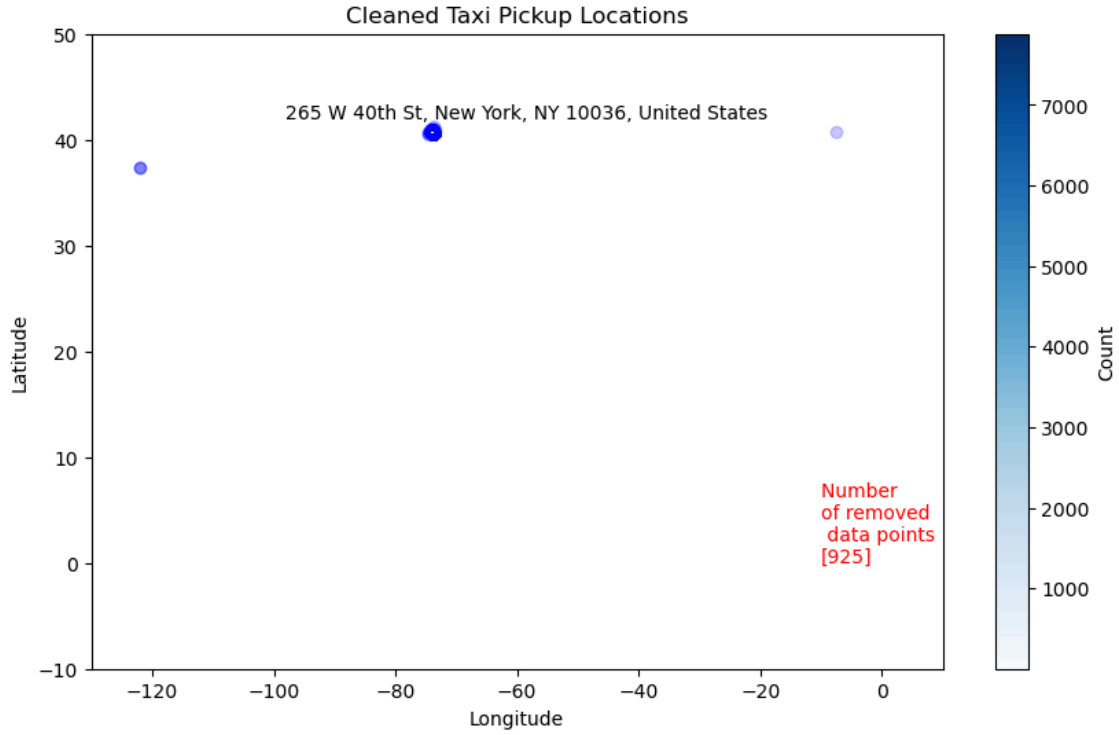
In [22]:
```
cleaned_pickup_data.shape
```

Out[22]: (99075, 19)

In [23]:
```
removed_count = taxi_data.shape[0] - cleaned_pickup_data.shape[0]
print(f"Number of removed data points: {removed_count}")
```

```
Number of removed data points: 925
```

In [27]:
```
plt.figure(figsize=(10, 6))
plt.scatter(cleaned_pickup_data['pickup_longitude'], cleaned_pickup_da


plt.hist2d(cleaned_pickup_data['pickup_longitude'], cleaned_pickup_dat
plt.colorbar(label='Count')
plt.text(s=f'Number \nof removed\n data points\n({removed_count})', x=
plt.text(s=f'265 W 40th St, New York, NY 10036, United States', x=-98,



plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Cleaned Taxi Pickup Locations')
plt.axis([-130,10,-10,50])
plt.show()
```

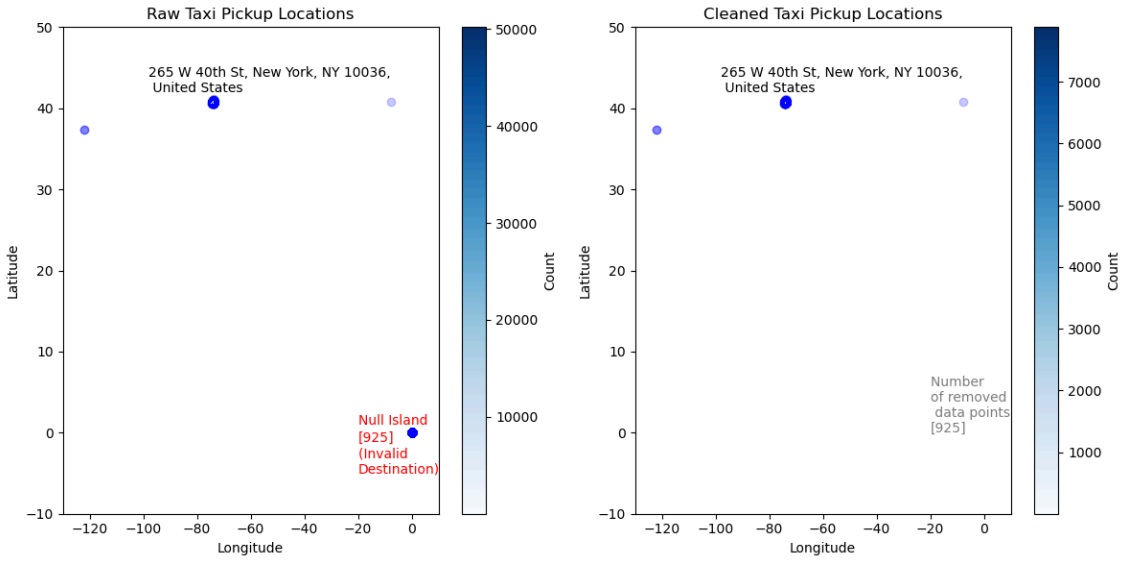


place two plots before after change sideby side & Count of how many (0,0) locations are removed

In [45]:

```
import matplotlib.pyplot as plt

plt.figure(figsize=(12, 6))

plt.subplot(1, 2, 1)
plt.scatter(
    taxi_data['pickup_longitude'],
    taxi_data['pickup_latitude'],
    alpha=0.2,
    label='Pickup',
    color='b'
)
plt.hist2d(
    taxi_data['pickup_longitude'],
    taxi_data['pickup_latitude'],
    bins=(1000, 1000),
    cmap='Blues',
    cmin=1
)
plt.colorbar(label='Count')
plt.text(
    s=f'Null Island',
    x=-20,
    y=1,
    size=10,
    color='red'
)
plt.text(
    s=f'({removed_count})\n Invalid \nDestination',
    x=-20,
    y=-5,
    size=10,
    color='red'
)
plt.text(
    s='265 W 40th St, New York, NY 10036,\n United States',
    x=-98,
    y=42,
    size=10,
    color='black'
)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Raw Taxi Pickup Locations')
plt.axis([-130, 10, -10, 50])


plt.subplot(1, 2, 2)
plt.scatter(
    cleaned_pickup_data['pickup_longitude'],
    cleaned_pickup_data['pickup_latitude'],
    alpha=0.2,
    label='Pickup',
    color='b'
)
plt.hist2d(
    cleaned_pickup_data['pickup_longitude'],
    cleaned_pickup_data['pickup_latitude'],
    bins=(1000, 1000),
    cmap='Blues',
    cmin=1
```

```
)
plt.colorbar(label='Count')
plt.text(
    s=f'Number \nof removed\n data points\n({removed_count})',
    x=-20,
    y=0,
    size=10,
    color='gray'
)
plt.text(
    s='265 W 40th St, New York, NY 10036,\n United States',
    x=-98,
    y=42,
    size=10,
    color='black'
)
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Cleaned Taxi Pickup Locations')
plt.axis([-130, 10, -10, 50])

plt.tight_layout()
plt.show()

```



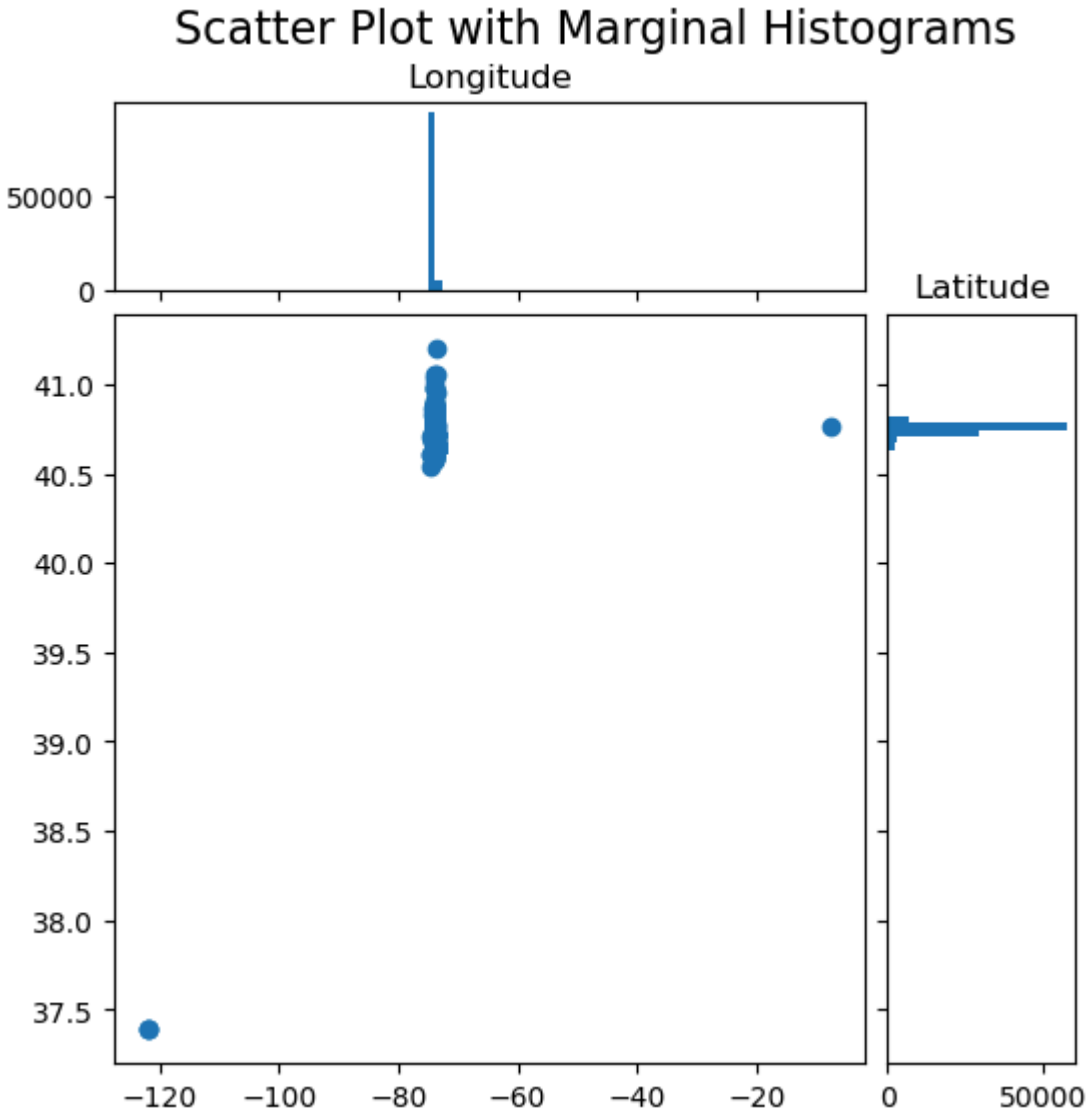
In [20]:
```
import plotly.express as px
import pandas as pd

fig = px.scatter(
    cleaned_pickup_data,
    x='pickup_longitude',
    y='pickup_latitude',
    title="Scatterplot of Valid Pickup Locations",
    labels={'pickup_longitude': 'Longitude', 'pickup_latitude': 'Latit
    template="plotly"
)
fig.show()
```

```
---------------------------------------------------------------------------
--
NameError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_3632\3814009198.py in <module>
      3
      4 fig = px.scatter(
----> 5     cleaned_pickup_data,
      6     x='pickup_longitude',
      7     y='pickup_latitude',

NameError: name 'cleaned_pickup_data' is not defined
```

In [24]:
```
x = cleaned_pickup_data['pickup_longitude']
y = cleaned_pickup_data['pickup_latitude']


def scatter_hist(x, y, ax, ax_histx, ax_histy):
    # no labels
    ax_histx.tick_params(axis="x", labelbottom=False)
    ax_histy.tick_params(axis="y", labelleft=False)

    # the scatter plot:
    ax.scatter(x, y)

    # now determine nice limits by hand:
    binwidth = 0.25
    xymax = max(np.max(np.abs(x)), np.max(np.abs(y)))
    lim = (int(xymax/binwidth) + 1) * binwidth

#     bins = np.arange(-lim, lim + binwidth, binwidth)
    ax_histx.hist(x, bins=100)
    ax_histy.hist(y, bins=100, orientation='horizontal')
```

In [27]:

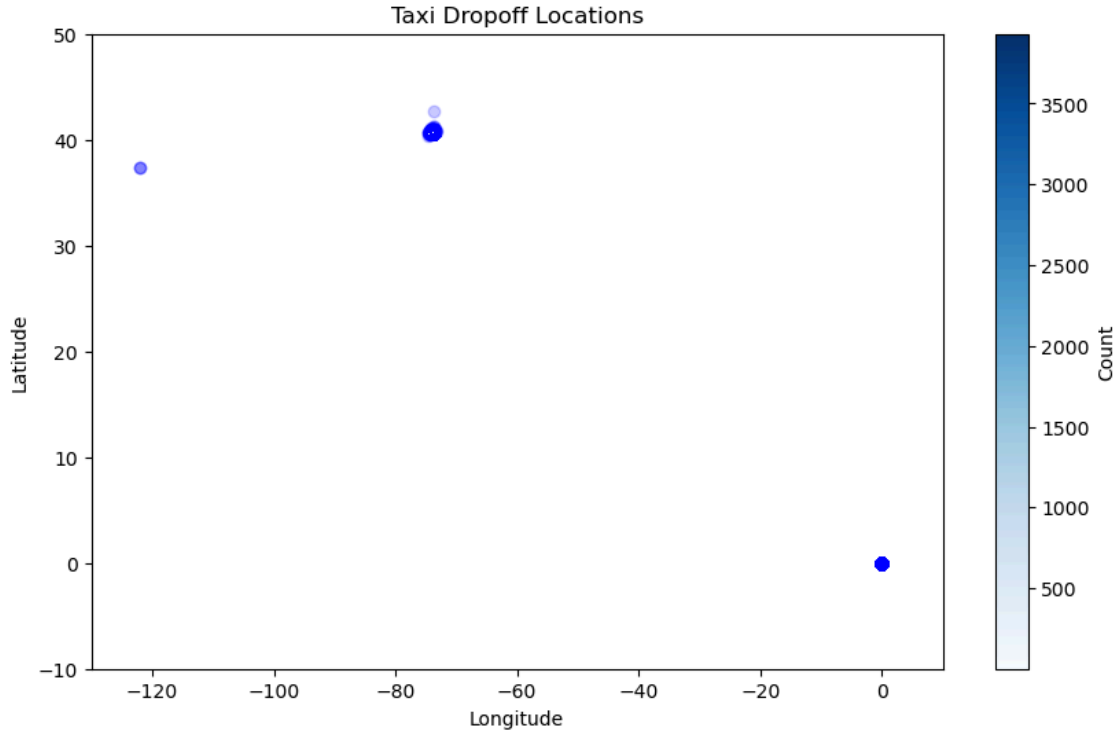
```
# Start with a square Figure.
fig = plt.figure(figsize=(6, 6))
# Add a gridspec with two rows and two columns and a ratio of 1 to 4 b
# the size of the marginal Axes and the main Axes in both directions.
# Also adjust the subplot parameters for a square plot.
gs = fig.add_gridspec(2, 2,  width_ratios=(4, 1), height_ratios=(1, 4)
                      left=0.1, right=0.9, bottom=0.1, top=0.9,
                      wspace=0.05, hspace=0.05)
# Create the Axes.
ax = fig.add_subplot(gs[1, 0])
ax_histx = fig.add_subplot(gs[0, 0], sharex=ax)
ax_histy = fig.add_subplot(gs[1, 1], sharey=ax)
ax_histx.set_title('Longitude')  # Label for the x-axis histogram
ax_histy.set_title('Latitude')   # Label for the y-axis histogram
# Draw the scatter plot and marginals.
fig.suptitle('Scatter Plot with Marginal Histograms', fontsize=16)
scatter_hist(x, y, ax, ax_histx, ax_histy)
```



In [ ]:

```
1
```

In [28]:
```
plt.figure(figsize=(10, 6))
plt.scatter(taxi_data['dropoff_longitude'], taxi_data['dropoff_latitud

plt.hist2d(taxi_data['dropoff_longitude'], taxi_data['dropoff_latitude
plt.colorbar(label='Count')

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Taxi Dropoff Locations')
plt.axis([-130,10,-10,50])
plt.show()
```

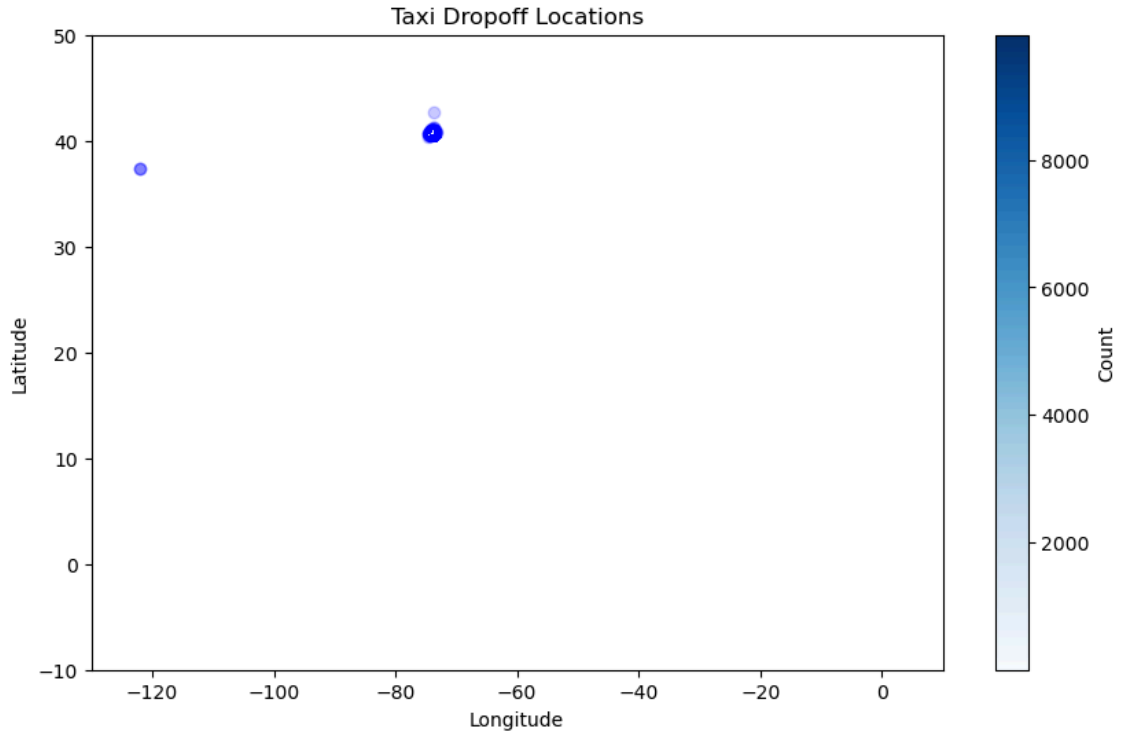


In [9]:
```
cleaned_dropoff_data = taxi_data[(taxi_data['dropoff_longitude'] != 0)
```

In [10]:
```
cleaned_dropoff_data.shape
```

Out[10]: (99107, 19)

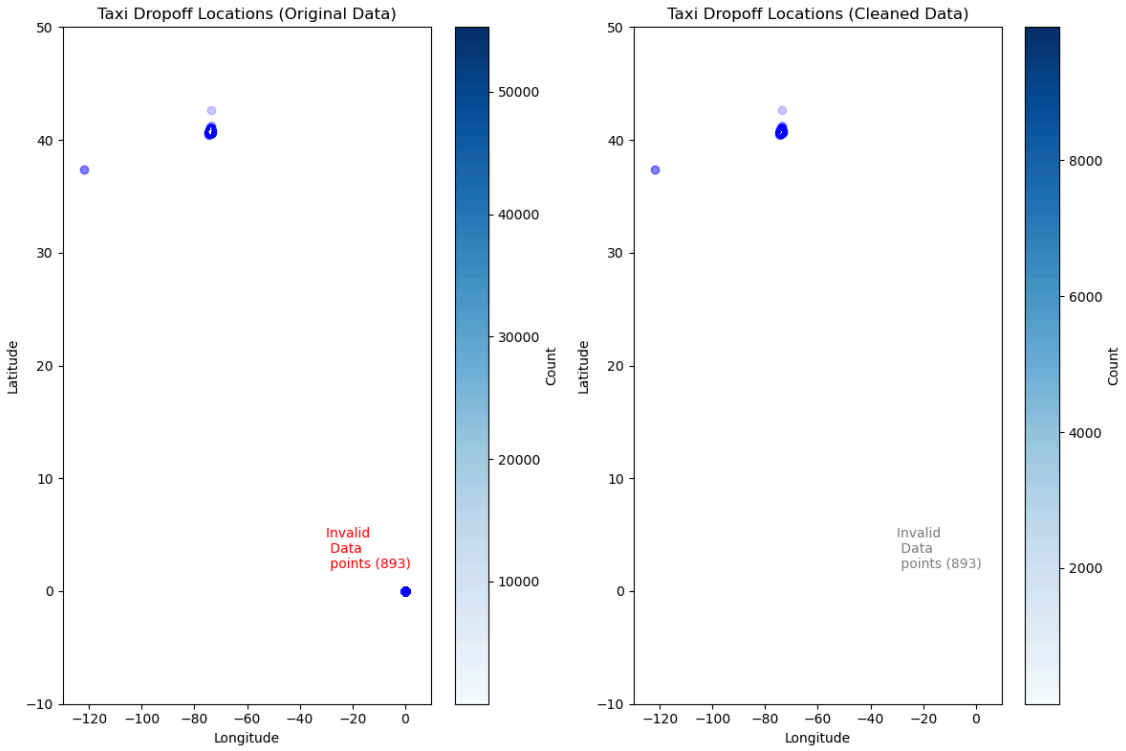
In [12]:
```
plt.figure(figsize=(10, 6))
plt.scatter(cleaned_dropoff_data['dropoff_longitude'], cleaned_dropoff


plt.hist2d(cleaned_dropoff_data['dropoff_longitude'], cleaned_dropoff_
plt.colorbar(label='Count')

plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Taxi Dropoff Locations')
plt.axis([-130,10,-10,50])
plt.show()
```



In [ ]:
```
fig = px.scatter(
    cleaned_dropoff_data,
    x='dropoff_longitude',
    y='dropoff_latitude',
    title="Scatterplot of Valid Dropoff Locations",
    labels={'dropoff_longitude': 'Longitude', 'dropoff_latitude': 'Lat
    template="plotly"
)
fig.show()
```

In [26]:

```
plt.figure(figsize=(12, 8))

plt.subplot(1, 2, 1)
plt.scatter(taxi_data['dropoff_longitude'], taxi_data['dropoff_latitud
plt.hist2d(taxi_data['dropoff_longitude'], taxi_data['dropoff_latitude
plt.colorbar(label='Count')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Taxi Dropoff Locations (Original Data)')
plt.text(-30, 2, 'Invalid \n Data\n points (893)', size=10, color='red
plt.axis([-130, 10, -10, 50])


plt.subplot(1, 2, 2)
plt.scatter(cleaned_dropoff_data['dropoff_longitude'], cleaned_dropoff
plt.hist2d(cleaned_dropoff_data['dropoff_longitude'], cleaned_dropoff_
plt.colorbar(label='Count')
plt.xlabel('Longitude')
plt.ylabel('Latitude')
plt.title('Taxi Dropoff Locations (Cleaned Data)')
plt.text(-30, 2, 'Invalid \n Data\n points (893)', size=10, color='gra
plt.axis([-130, 10, -10, 50])

plt.tight_layout()
plt.show()

```



In [13]:

```
100000-99107
```

Out[13]: 893

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

A sizable number of rows show zero lat, long as dropoff location. This need to be investigated.

In [4]:
```
1 zero_lat_long_rows = taxi_data[(taxi_data['pickup_longitude'] == 0) &
2 valid_taxi_data = taxi_data[(taxi_data['pickup_longitude'] != 0) | (ta
3 # print("Removed data points:")
4 # print(zero_lat_long_rows)
5
```

In [5]:
```
1 zero_lat_long_rows
```

Out[5]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
116	2	01-03-2016 00:00	01-03-2016 00:11	1	2.9
365	2	10-03-2016 07:08	10-03-2016 07:32	2	8.8
434	2	10-03-2016 07:08	10-03-2016 07:09	6	0.0
478	2	10-03-2016 07:09	10-03-2016 07:40	1	18.0
491	2	10-03-2016 07:09	10-03-2016 07:17	5	1.7
...	...	...	...	...	
99620	2	10-03-2016 14:27	10-03-2016 14:30	1	0.5
99641	1	01-03-2016 06:14	01-03-2016 06:18	1	0.8
99653	1	01-03-2016 06:14	01-03-2016 07:02	1	18.2
99655	1	01-03-2016 06:14	01-03-2016 06:21	1	2.6
99990	2	01-03-2016 06:17	01-03-2016 06:25	5	2.1

925 rows × 19 columns

In [6]:
```
np.vstack(zero_lat_long_rows.columns)
```

Out[6]:
```
array([['VendorID'],
       ['tpep_pickup_datetime'],
       ['tpep_dropoff_datetime'],
       ['passenger_count'],
       ['trip_distance'],
       ['pickup_longitude'],
       ['pickup_latitude'],
       ['RatecodeID'],
       ['store_and_fwd_flag'],
       ['dropoff_longitude'],
       ['dropoff_latitude'],
       ['payment_type'],
       ['fare_amount'],
       ['extra'],
       ['mta_tax'],
       ['tip_amount'],
       ['tolls_amount'],
       ['improvement_surcharge'],
       ['total_amount']], dtype='<U21')
```

In [7]:
```
zero_lat_long_rows.iloc[:,[3,4,12,5,6,9,10]]
```

Out[7]:

	passenger_count	trip_distance	fare_amount	pickup_longitude	pickup_latitude	drop
116	1	2.91	12.0	0.0	0.0	
365	2	8.81	28.0	0.0	0.0	
434	6	0.02	2.5	0.0	0.0	
478	1	18.05	66.0	0.0	0.0	
491	5	1.73	8.0	0.0	0.0	
...	...	...	...	...	...	
99620	1	0.53	4.0	0.0	0.0	
99641	1	0.80	5.5	0.0	0.0	
99653	1	18.20	52.0	0.0	0.0	
99655	1	2.60	9.5	0.0	0.0	
99990	5	2.11	8.5	0.0	0.0	

925 rows × 7 columns

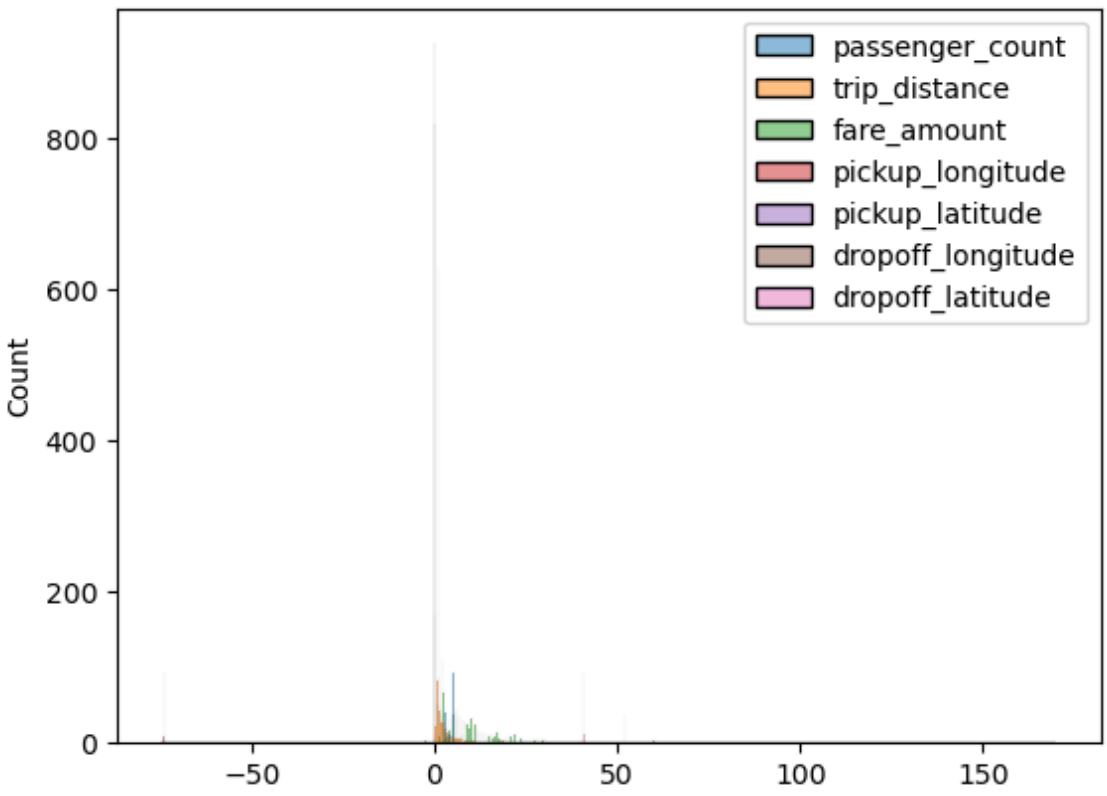
In [8]:
```
1 zero_lat_long_rows.iloc[:,[3,4,12,5,6,9,10]].describe()
```

Out[8]:

	passenger_count	trip_distance	fare_amount	pickup_longitude	pickup_latitude	drop
count	925.000000	925.000000	925.000000	925.0	925.0	
mean	1.878919	2.431405	15.161849	0.0	0.0	
std	1.559928	3.300437	17.893160	0.0	0.0	
min	0.000000	0.000000	-2.500000	0.0	0.0	
25%	1.000000	0.590000	5.500000	0.0	0.0	
50%	1.000000	1.280000	9.500000	0.0	0.0	
75%	2.000000	2.930000	17.000000	0.0	0.0	
max	6.000000	21.920000	170.000000	0.0	0.0	

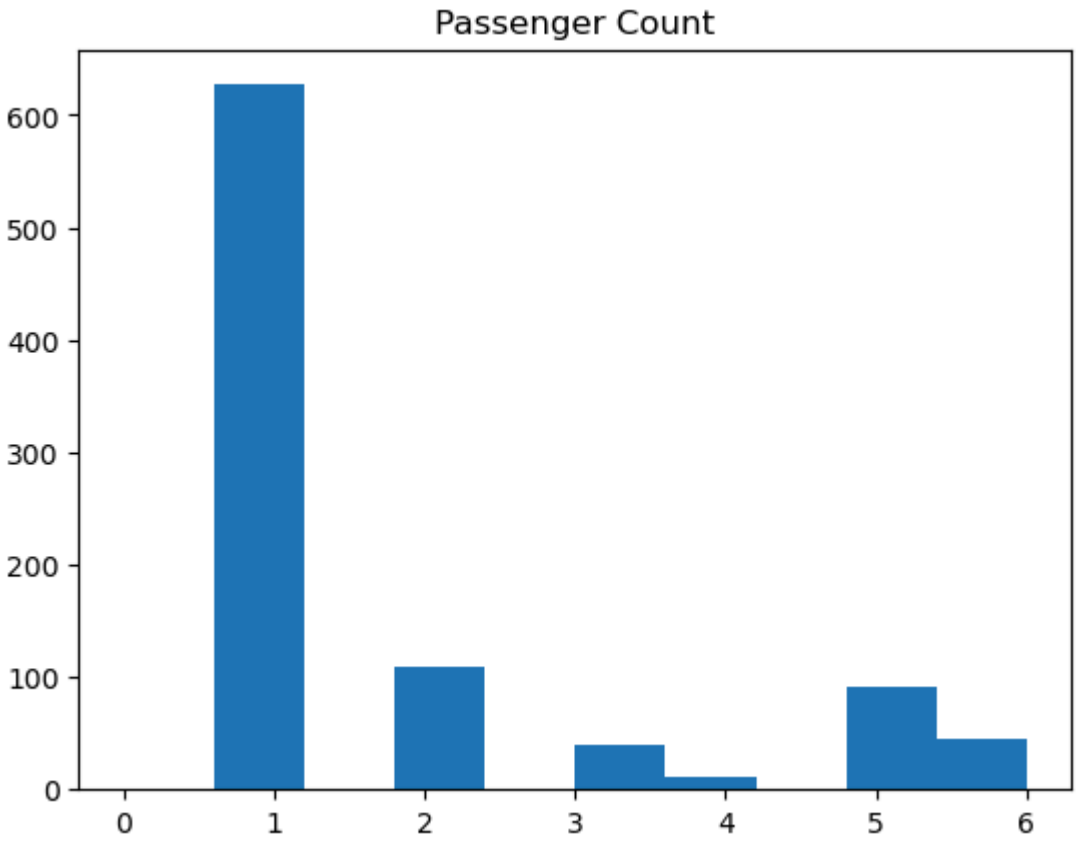
In [64]:
```
1 sns.histplot(zero_lat_long_rows.iloc[:,[3,4,12,5,6,9,10]])
```

Out[64]: <AxesSubplot:ylabel='Count'>



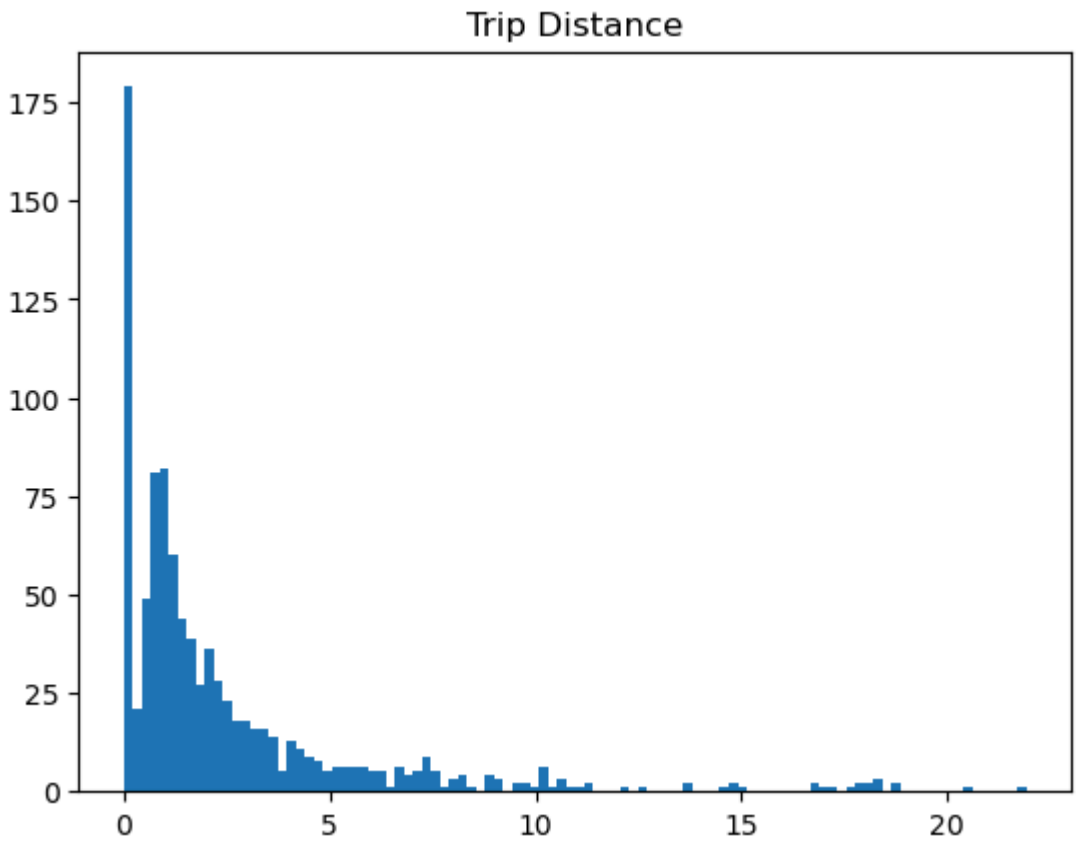
In [66]:
```
1 plt.hist(zero_lat_long_rows.iloc[:,[3]])
2 plt.title("Passenger Count")
```

Out[66]: Text(0.5, 1.0, 'Passenger Count')



In [10]:
```
count,bins,hist_fig =plt.hist(zero_lat_long_rows.iloc[:,[4]],bins = 10
# plt.ylim(30)
plt.title("Trip Distance")
```

Out[10]: Text(0.5, 1.0, 'Trip Distance')



In [102]:
```
# np.hstack(dir(zero_lat_long_rows.iloc[:,[4]]))
set((zero_lat_long_rows.iloc[:,[4]]).__array__)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_13676\1050776519.py in <module>
      1 # np.hstack(dir(zero_lat_long_rows.iloc[:,[4]]))
----> 2 set((zero_lat_long_rows.iloc[:,[4]]).__array__)

TypeError: 'method' object is not iterable
```

In [11]:
```
bins.shape
low_bins = bins[:-1]
low_bins
```

Out[11]:
```
array([ 0.    ,  0.2192,  0.4384,  0.6576,  0.8768,  1.096 ,  1.3152,
        1.5344,  1.7536,  1.9728,  2.192 ,  2.4112,  2.6304,  2.8496,
        3.0688,  3.288 ,  3.5072,  3.7264,  3.9456,  4.1648,  4.384 ,
        4.6032,  4.8224,  5.0416,  5.2608,  5.48  ,  5.6992,  5.9184,
        6.1376,  6.3568,  6.576 ,  6.7952,  7.0144,  7.2336,  7.4528,
        7.672 ,  7.8912,  8.1104,  8.3296,  8.5488,  8.768 ,  8.9872,
        9.2064,  9.4256,  9.6448,  9.864 , 10.0832, 10.3024, 10.5216,
       10.7408, 10.96  , 11.1792, 11.3984, 11.6176, 11.8368, 12.056 ,
       12.2752, 12.4944, 12.7136, 12.9328, 13.152 , 13.3712, 13.5904,
       13.8096, 14.0288, 14.248 , 14.4672, 14.6864, 14.9056, 15.1248,
       15.344 , 15.5632, 15.7824, 16.0016, 16.2208, 16.44  , 16.6592,
       16.8784, 17.0976, 17.3168, 17.536 , 17.7552, 17.9744, 18.1936,
       18.4128, 18.632 , 18.8512, 19.0704, 19.2896, 19.5088, 19.728 ,
       19.9472, 20.1664, 20.3856, 20.6048, 20.824 , 21.0432, 21.2624,
       21.4816, 21.7008])
```

In [12]:
```
up_bins = bins[1:]
up_bins
```

Out[12]:
```
array([ 0.2192,  0.4384,  0.6576,  0.8768,  1.096 ,  1.3152,  1.5344,
        1.7536,  1.9728,  2.192 ,  2.4112,  2.6304,  2.8496,  3.0688,
        3.288 ,  3.5072,  3.7264,  3.9456,  4.1648,  4.384 ,  4.6032,
        4.8224,  5.0416,  5.2608,  5.48  ,  5.6992,  5.9184,  6.1376,
        6.3568,  6.576 ,  6.7952,  7.0144,  7.2336,  7.4528,  7.672 ,
        7.8912,  8.1104,  8.3296,  8.5488,  8.768 ,  8.9872,  9.2064,
        9.4256,  9.6448,  9.864 , 10.0832, 10.3024, 10.5216, 10.7408,
       10.96  , 11.1792, 11.3984, 11.6176, 11.8368, 12.056 , 12.2752,
       12.4944, 12.7136, 12.9328, 13.152 , 13.3712, 13.5904, 13.8096,
       14.0288, 14.248 , 14.4672, 14.6864, 14.9056, 15.1248, 15.344 ,
       15.5632, 15.7824, 16.0016, 16.2208, 16.44  , 16.6592, 16.8784,
       17.0976, 17.3168, 17.536 , 17.7552, 17.9744, 18.1936, 18.4128,
       18.632 , 18.8512, 19.0704, 19.2896, 19.5088, 19.728 , 19.9472,
       20.1664, 20.3856, 20.6048, 20.824 , 21.0432, 21.2624, 21.4816,
       21.7008, 21.92  ])
```

In [13]:
```
bins
```

Out[13]:
```
array([ 0.    ,  0.2192,  0.4384,  0.6576,  0.8768,  1.096 ,  1.3152,
        1.5344,  1.7536,  1.9728,  2.192 ,  2.4112,  2.6304,  2.8496,
        3.0688,  3.288 ,  3.5072,  3.7264,  3.9456,  4.1648,  4.384 ,
        4.6032,  4.8224,  5.0416,  5.2608,  5.48  ,  5.6992,  5.9184,
        6.1376,  6.3568,  6.576 ,  6.7952,  7.0144,  7.2336,  7.4528,
        7.672 ,  7.8912,  8.1104,  8.3296,  8.5488,  8.768 ,  8.9872,
        9.2064,  9.4256,  9.6448,  9.864 , 10.0832, 10.3024, 10.5216,
       10.7408, 10.96  , 11.1792, 11.3984, 11.6176, 11.8368, 12.056 ,
       12.2752, 12.4944, 12.7136, 12.9328, 13.152 , 13.3712, 13.5904,
       13.8096, 14.0288, 14.248 , 14.4672, 14.6864, 14.9056, 15.1248,
       15.344 , 15.5632, 15.7824, 16.0016, 16.2208, 16.44  , 16.6592,
       16.8784, 17.0976, 17.3168, 17.536 , 17.7552, 17.9744, 18.1936,
       18.4128, 18.632 , 18.8512, 19.0704, 19.2896, 19.5088, 19.728 ,
       19.9472, 20.1664, 20.3856, 20.6048, 20.824 , 21.0432, 21.2624,
       21.4816, 21.7008, 21.92  ])
```

In [15]:
```
dist_df=pd.Series({
        'low_bins': low_bins,
        'up_bins': up_bins,
        'count': count})
pd.DataFrame(dist_df).T
```

Out[15]:

	low_bins	up_bins	count
0	[0.0, 0.2192, 0.4384, 0.6576, 0.8768, 1.096, 1...	[0.2192, 0.4384, 0.6576, 0.8768, 1.096, 1.3152...	[179.0, 21.0, 49.0, 81.0, 82.0, 60.0, 44.0, 39...

In [125]:
```
type(dist_df)
```

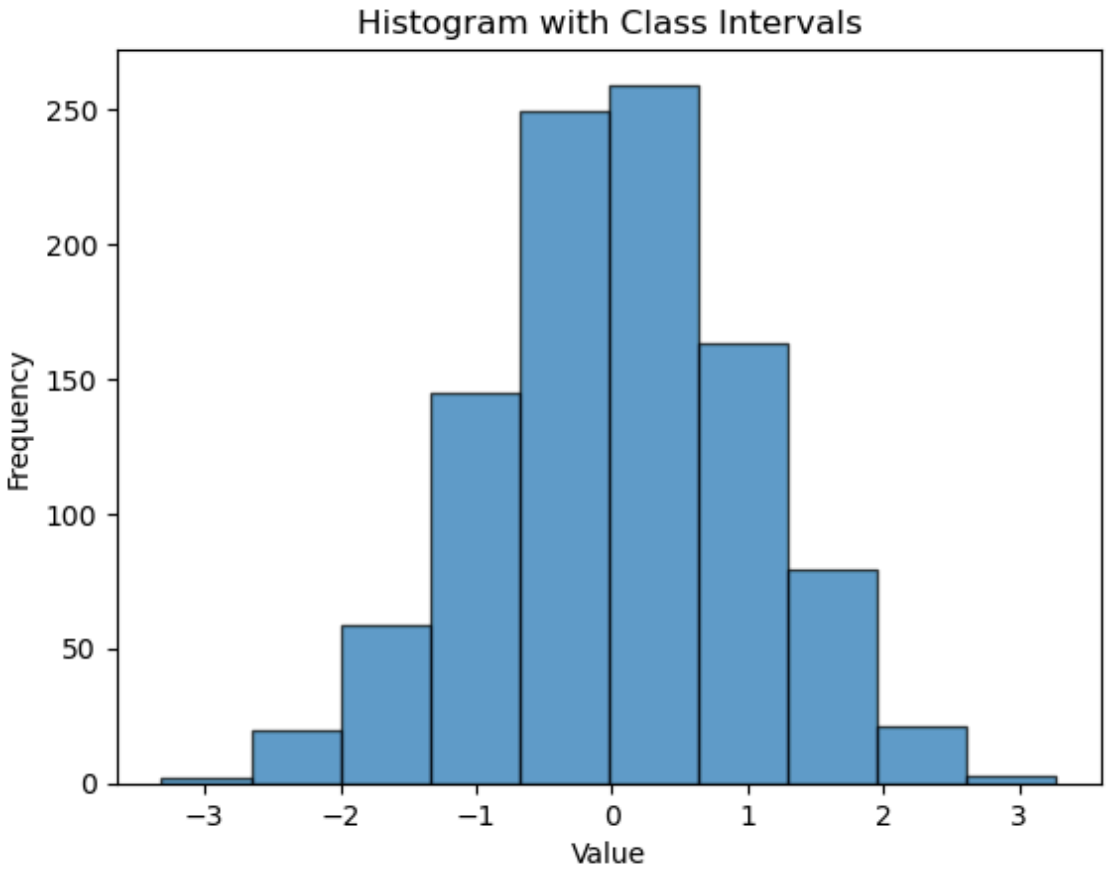
Out[125]:
```
dict
```

How to show class interval and Count from plt.hist output

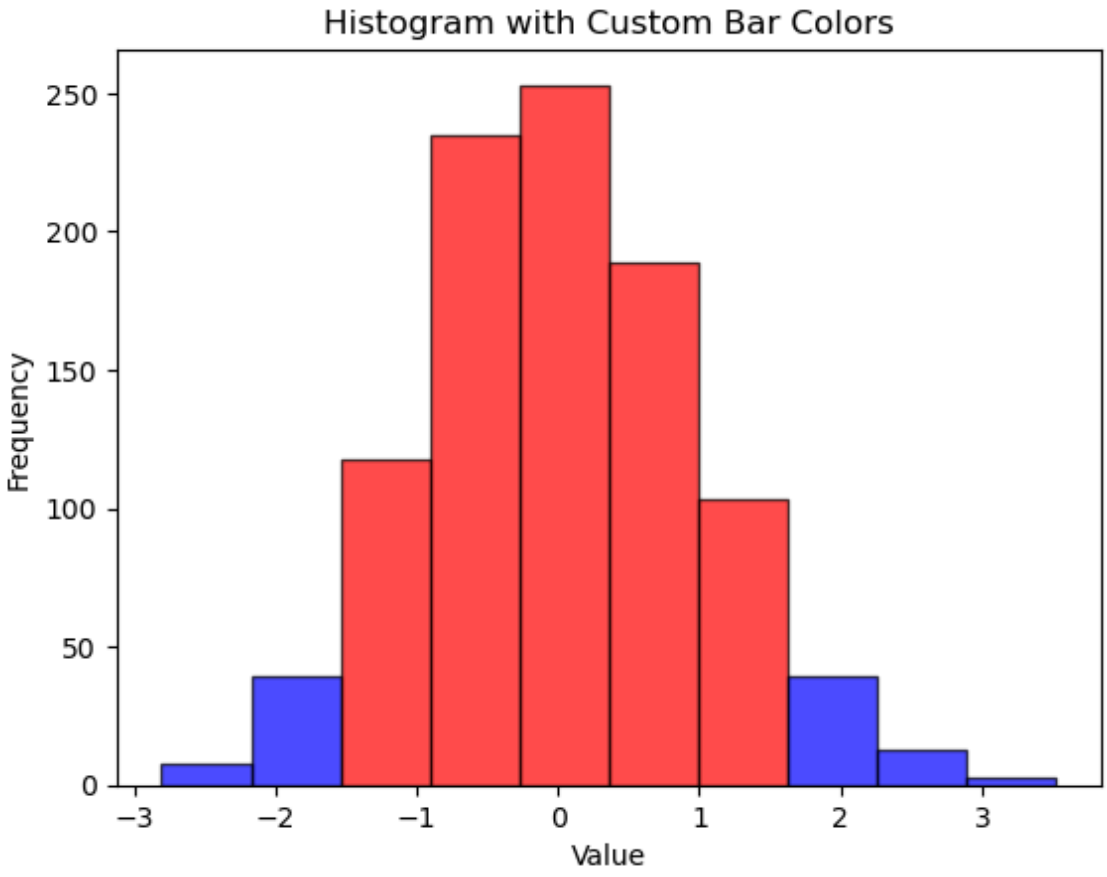
In [16]:

```
import matplotlib.pyplot as plt
import numpy as np

# Example data
data = np.random.randn(1000)  # 1000 random points from a normal distr

# Create a histogram and get the outputs
n, bins, patches = plt.hist(data, bins=10, edgecolor='black', alpha=0.

# Print class intervals and counts
for i in range(len(bins) - 1):
    print(f"Class Interval: [{bins[i]:.2f}, {bins[i+1]:.2f}) - Count:

# Display the plot
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram with Class Intervals')
plt.show()

```

```
Class Interval: [-3.32, -2.66) - Count: 2
Class Interval: [-2.66, -2.00) - Count: 20
Class Interval: [-2.00, -1.34) - Count: 59
Class Interval: [-1.34, -0.68) - Count: 145
Class Interval: [-0.68, -0.02) - Count: 249
Class Interval: [-0.02, 0.64) - Count: 259
Class Interval: [0.64, 1.30) - Count: 163
Class Interval: [1.30, 1.95) - Count: 79
Class Interval: [1.95, 2.61) - Count: 21
Class Interval: [2.61, 3.27) - Count: 3
```


Histogram with Class Intervals
Frequency
Value
0
50
100
150
200
250
−3
−2
−1
0
1
2
3

In [17]:
```
import matplotlib.pyplot as plt
import numpy as np

# Example data
data = np.random.randn(1000)  # 1000 random points from a normal distr

# Create a histogram and get patches
n, bins, patches = plt.hist(data, bins=10, edgecolor='black', alpha=0.

# Change the color of the bars dynamically
for i, patch in enumerate(patches):
    if n[i] > 50:  # Example condition: highlight bins with counts > 5
        patch.set_facecolor('red')  # Set bar color to red
    else:
        patch.set_facecolor('blue')  # Set bar color to blue

# Display the plot
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.title('Histogram with Custom Bar Colors')
plt.show()

```



In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [18]:
```
1 count
```

Out[18]:
```
array([179.,  21.,  49.,  81.,  82.,  60.,  44.,  39.,  27.,  36.,  28.,
        23.,  18.,  18.,  16.,  16.,  14.,   5.,  13.,  11.,   9.,   8.,
         5.,   6.,   6.,   6.,   6.,   5.,   5.,   1.,   6.,   4.,   5.,
         9.,   5.,   1.,   3.,   4.,   1.,   0.,   4.,   3.,   0.,   2.,
         2.,   1.,   6.,   1.,   3.,   1.,   1.,   2.,   0.,   0.,   0.,
         1.,   0.,   1.,   0.,   0.,   0.,   0.,   2.,   0.,   0.,   0.,
         1.,   2.,   1.,   0.,   0.,   0.,   0.,   0.,   0.,   0.,   2.,
         1.,   1.,   0.,   1.,   2.,   2.,   3.,   0.,   2.,   0.,   0.,
         0.,   0.,   0.,   0.,   0.,   1.,   0.,   0.,   0.,   0.,   0.,
         1.])
```

In [109]:
```
1 zero_lat_long_rows["trip_distance"].unique
```

Out[109]:
```
<bound method Series.unique of 116       2.91
365       8.81
434       0.02
478      18.05
491       1.73
         ...
99620     0.53
99641     0.80
99653    18.20
99655     2.60
99990     2.11
Name: trip_distance, Length: 925, dtype: float64>
```

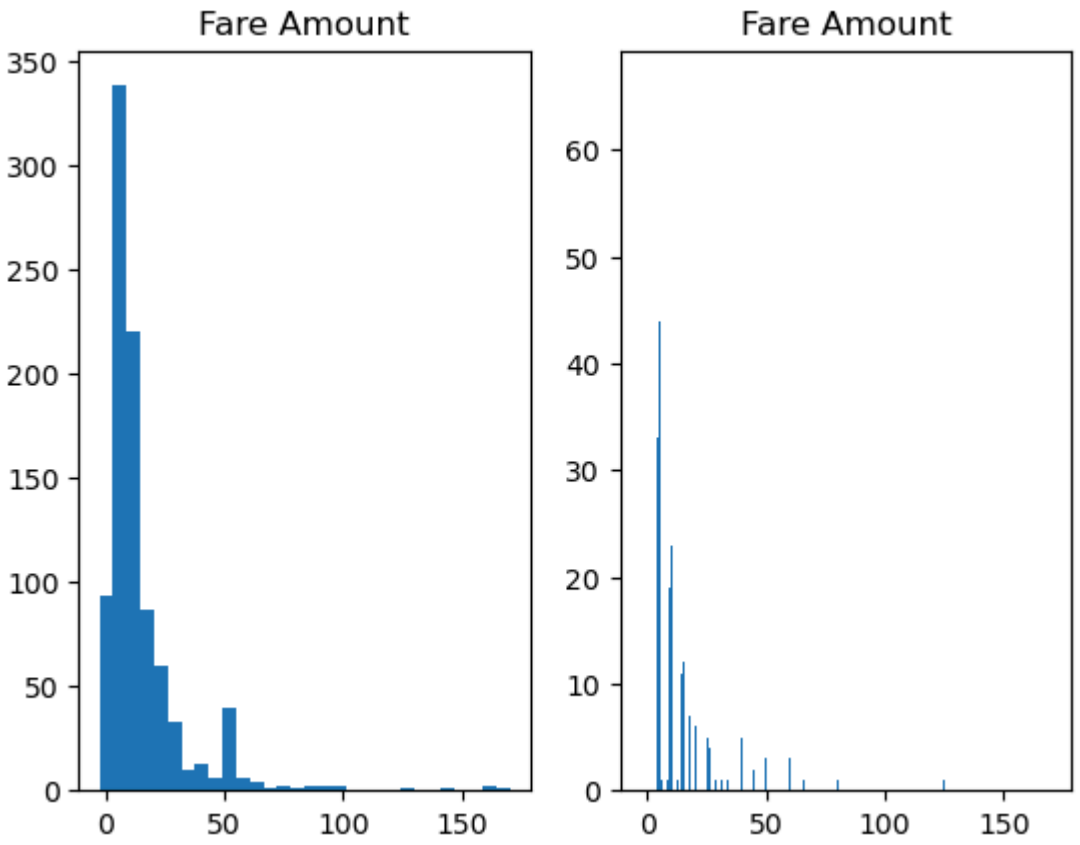
In [76]:
```
1 count.max()
```

Out[76]:
```
165.0
```

In [77]:
```
1 count
```

In [87]:
```
plt.subplot(1,2,1)
plt.hist(zero_lat_long_rows.iloc[:,[12]],bins=30)
plt.title("Fare Amount")

plt.subplot(1,2,2)
plt.hist(zero_lat_long_rows.iloc[:,[12]],bins=925)
plt.title("Fare Amount")
```

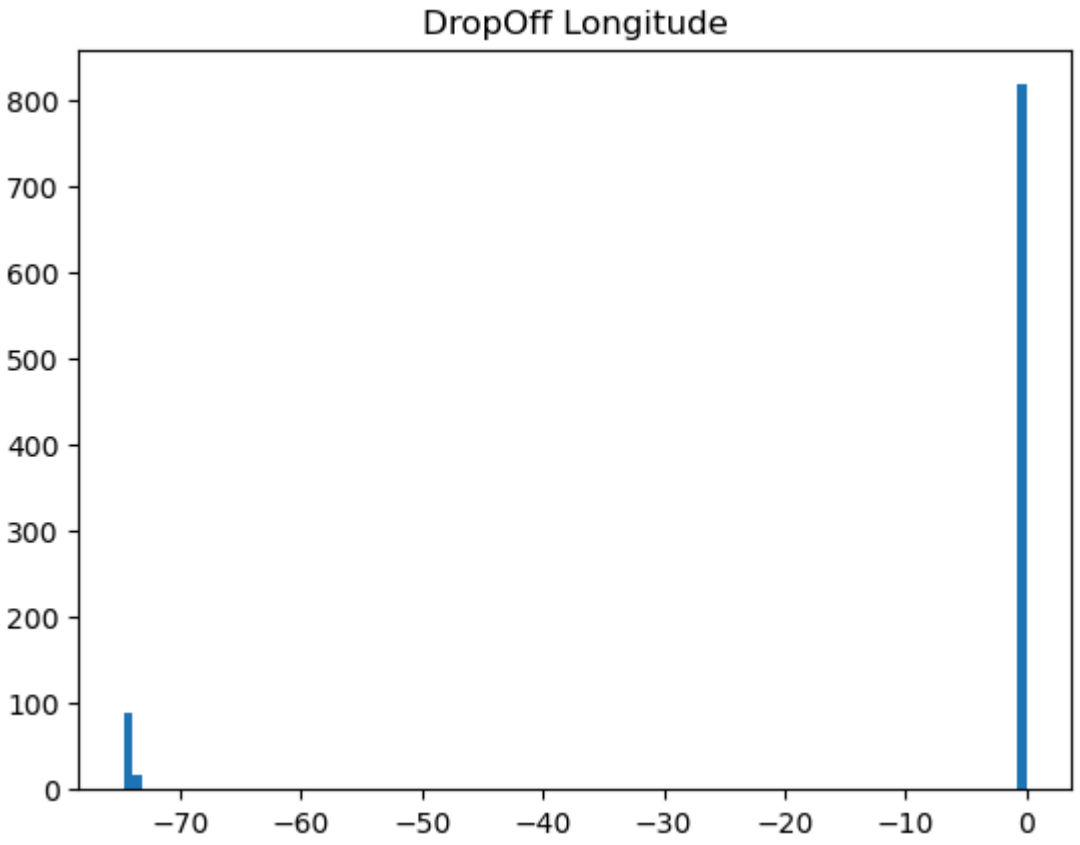
Out[87]: Text(0.5, 1.0, 'Fare Amount')



In [ ]:
```
1
```

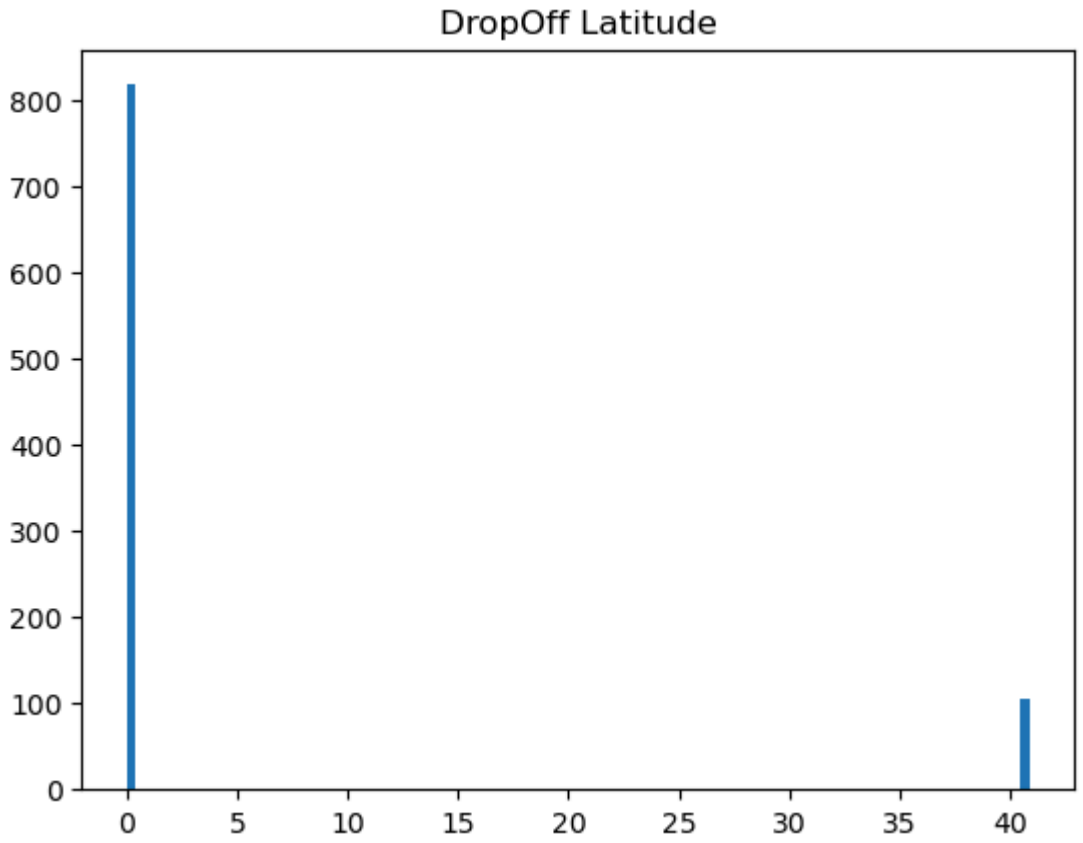
In [92]:
```
1 plt.hist(zero_lat_long_rows.iloc[:,[9]],bins=100)
2 plt.title("DropOff Longitude")
```

Out[92]: Text(0.5, 1.0, 'DropOff Longitude')



In [90]:
```
plt.hist(zero_lat_long_rows.iloc[:,[10]],bins=100)
plt.title("DropOff Latitude")
```

Out[90]: Text(0.5, 1.0, 'DropOff Latitude')



In [ ]:
```
1
```

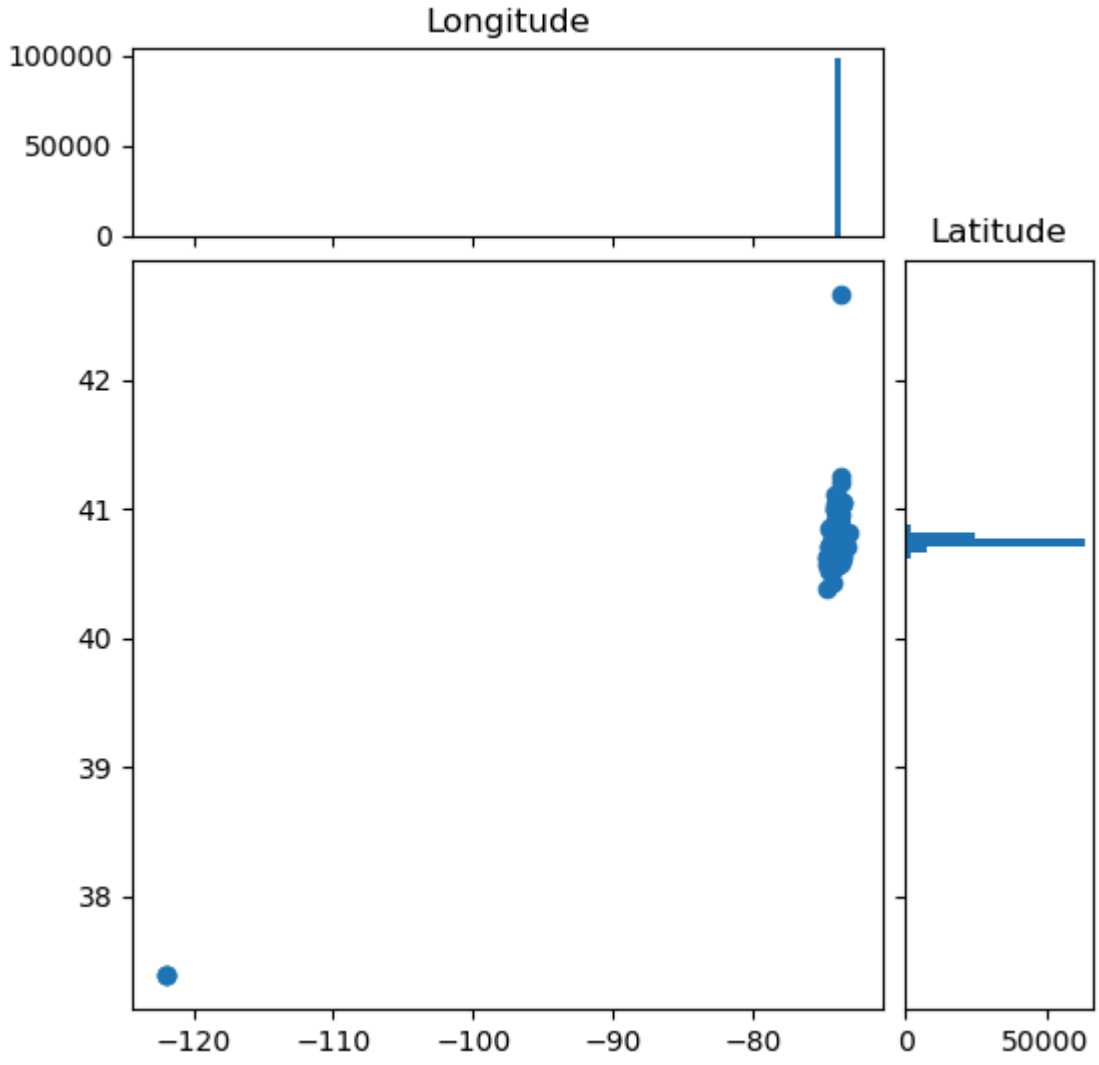
In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:

```
x1 = cleaned_dropoff_data['dropoff_longitude']
y1 = cleaned_dropoff_data['dropoff_latitude']


def scatter_hist(x, y, ax, ax_histx1, ax_histy1):
    # no labels
    ax_histx1.tick_params(axis="x", labelbottom=False)
    ax_histy1.tick_params(axis="y", labelleft=False)

    # the scatter plot:
    ax.scatter(x1, y1)

    # now determine nice limits by hand:
    binwidth = 0.25
    xymax = max(np.max(np.abs(x1)), np.max(np.abs(y1)))
    lim = (int(xymax/binwidth) + 1) * binwidth

#     bins = np.arange(-lim, lim + binwidth, binwidth)
    ax_histx1.hist(x1, bins=100)
    ax_histy1.hist(y1, bins=100, orientation='horizontal')
```

In [33]:
```
# Start with a square Figure.
fig = plt.figure(figsize=(6, 6))
# Add a gridspec with two rows and two columns and a ratio of 1 to 4 b
# the size of the marginal Axes and the main Axes in both directions.
# Also adjust the subplot parameters for a square plot.
gs = fig.add_gridspec(2, 2,  width_ratios=(4, 1), height_ratios=(1, 4)
                      left=0.1, right=0.9, bottom=0.1, top=0.9,
                      wspace=0.05, hspace=0.05)
# Create the Axes.
ax = fig.add_subplot(gs[1, 0])
ax_histx1 = fig.add_subplot(gs[0, 0], sharex=ax)
ax_histy1 = fig.add_subplot(gs[1, 1], sharey=ax)
ax_histx1.set_title('Longitude')  # Label for the x-axis histogram
ax_histy1.set_title('Latitude')   # Label for the y-axis histogram
# Draw the scatter plot and marginals.
scatter_hist(x1, y1, ax, ax_histx1, ax_histy1)
```



Add choroplot to show the coordinates

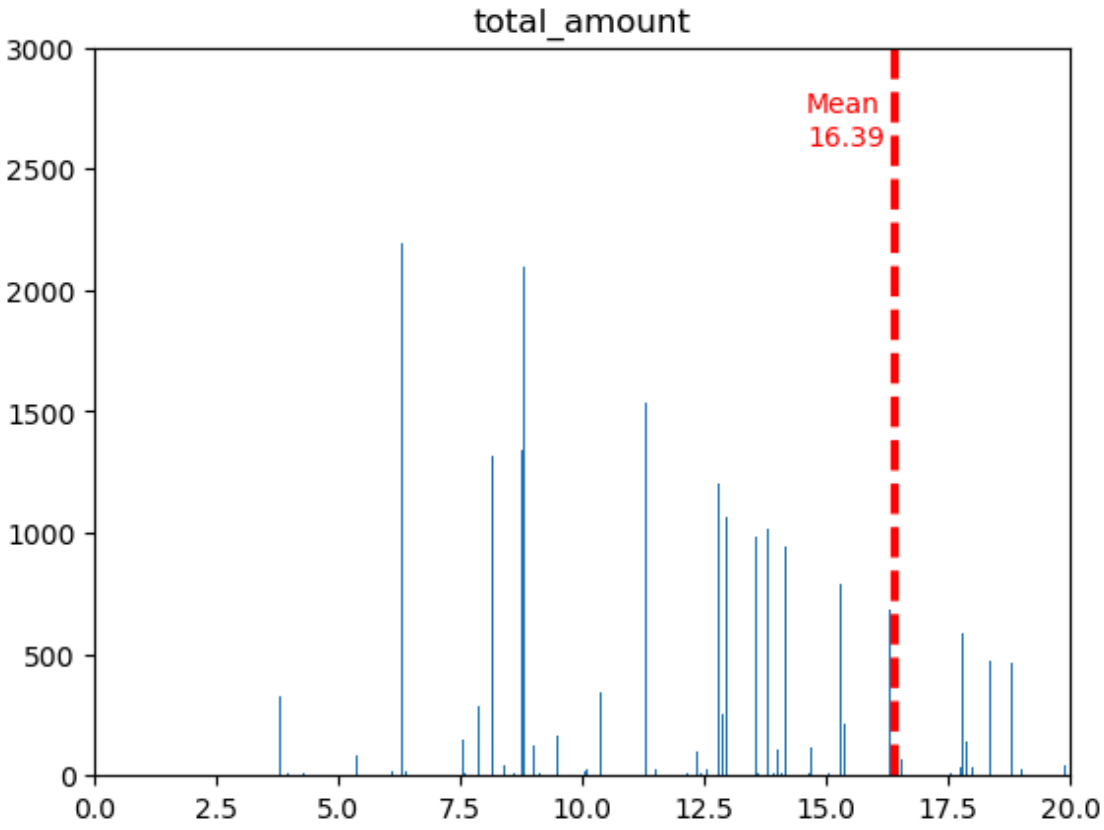
In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
plt.hist(taxi_data["total_amount"],bins = 100000);
plt.axis([0,100, 0,500])
# plt.ylim(0,1000)
plt.show();
```

In [ ]:
```
plt.hist(taxi_data["total_amount"],bins = 100000);
plt.axis([0,78, 0,2000])
# plt.ylim(0,1000)
plt.show();
```

In [37]:
```
plt.hist(taxi_data["total_amount"], bins=100000)
plt.axvline(x=taxi_data["total_amount"].mean(), color="red", linestyle
mean_value = round(taxi_data["total_amount"].mean(), 2)
plt.text(s=f'Mean\n{mean_value}', x=0.89 * taxi_data["total_amount"].m
plt.axis([0, 20, 0, 3000])
plt.xlabel('total_amount')
plt.title("total amount")

# plt.ylim(0, 1000)
plt.show()

```



Meaning of Mean:

Mode in python . How many max passengers

In [38]:
```
max_total_amount = taxi_data["total_amount"].max()
print(f"The maximum total amount is ${max_total_amount:.2f}")
```

```
The maximum total amount is $832.80
```

In [39]:
```
min_total_amount = taxi_data["total_amount"].min()
print(f"The minimum total amount is ${min_total_amount:.2f}")
```

```
The minimum total amount is $-47.30
```

In [40]:
```
mean_total_amount = round(taxi_data["total_amount"].mean(),2)
mean_total_amount
```

Out[40]: 16.39

In [54]:
```
std_total_amount = round(taxi_data["total_amount"].std(),2)
std_total_amount
```

Out[54]: 14.44

In [41]:
```
median_total_amount = statistics.median(taxi_data["total_amount"])
median_total_amount
```

Out[41]: 11.8

In [52]:
```
Q1, Q2, Q3 = np.quantile(taxi_data["total_amount"], [.25, .5, .75] ).r
W1 = (Q1 - 1.5*(Q3-Q1)).round(2)
W2 = (Q3 + 1.5*(Q3-Q1)).round(2)

```

In [70]:
```
# print(f'Computed whiskers {(W1, W2)}')
x_adj = 0.05 # adj factor used for improving display

# my_arr[my_arr < W1]  # outliers on the lower (left) side - None
# my_arr[my_arr > W2]  # outliers on the upper (right) side - Only One

plt.boxplot(taxi_data["total_amount"], vert = False, showmeans = True)

# The lines below annotate the boxplot with Q1 Q2 Q3
plt.text(s=f'Left_Whisker \n{W1}', x= W1 - 5*x_adj, y = 1.1, color = '
plt.text(s=f'Right_Whisker \n{W2}', x= W2 - 5*x_adj, y = 1.1, color =
plt.text(s=f'Q1 \n{Q1}', x= Q1 - x_adj, y = 1.1, color = 'red', size =
plt.text(s=f'Q2 \n{Q2}', x= Q2  - x_adj, y = 1.1, color = 'red', size
plt.text(s=f'Q3 \n{Q3}', x= Q3  - x_adj, y = 1.1, color = 'red', size


## Further annotation adjusts the x-, y-positions

plt.text(s=f'Left_Whisker: \nlast extreme pt.\n before which \n outlie
plt.text(s=f'Right_Whisker: \nlast extreme pt.\n after which \n outlie
plt.text(s=f'<-------------IQR--------------->', x= Q2 -1.2, y = .8, c
# plt.text(s=f'ISSUE \n : Why does right whisker \n appear at ~ 2.5\n
plt.title('Box-Whisker plot of `total_amount` showing median, IQR \n E

plt.text(s=f"total_amount : Random normal values\n mean {mean_total_am
```

...

In [56]:
```
import seaborn as sns
```

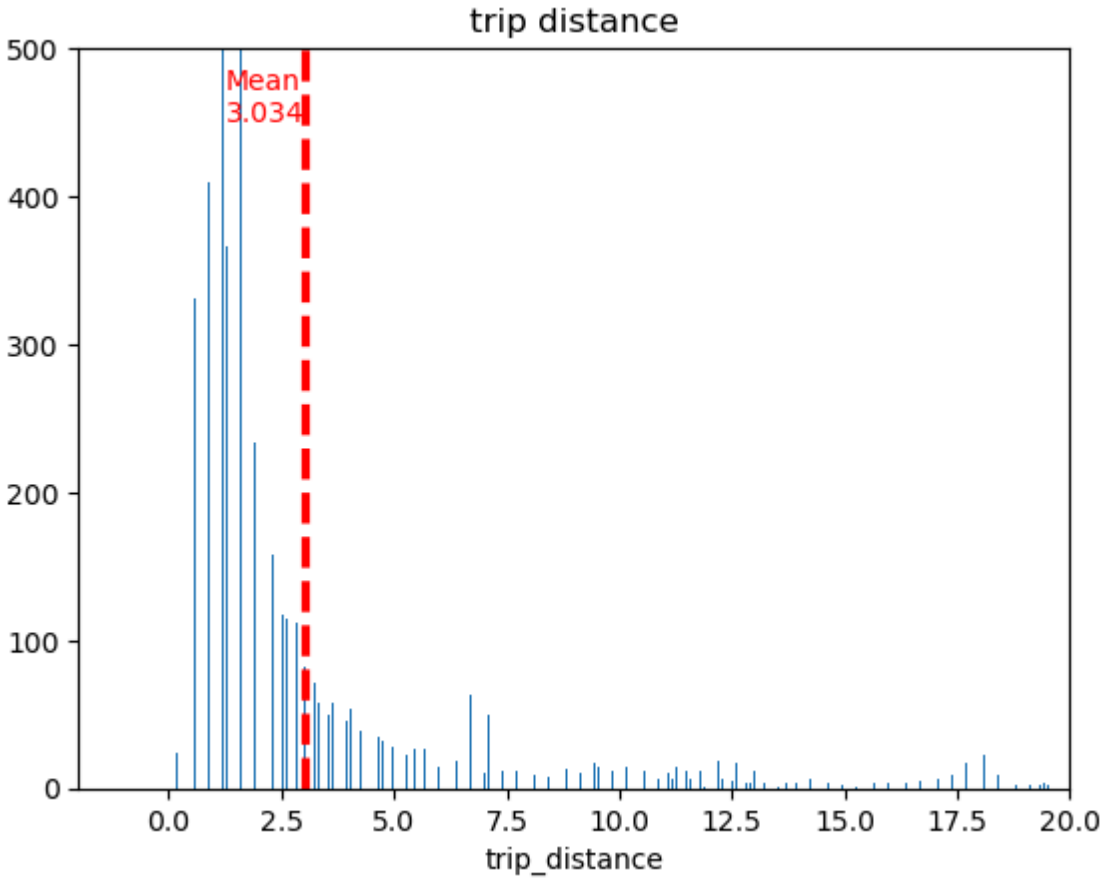
In [57]:
```
plt.figure(figsize=(8, 5))
plt.boxplot(taxi_data['total_amount'], vert=False, showmeans=True)

# Add title and labels
plt.title('Boxplot of Total Amount', fontsize=16)
plt.xlabel('Total Amount', fontsize=12)

# Display the plot
plt.show()
```

...

In [ ]:
```

```

In [ ]:
```

```

In [44]:
```
plt.hist(taxi_data["trip_distance"],bins = 100000);
plt.axvline(x=mean_trip_distance, color="red",linestyle="--",linewidth
# plt.axvline(x=taxi_data["trip_distance"].median(), color="orange",li
plt.text(s=f'Mean\n{mean_trip_distance}', x=0.41 * taxi_data["trip_dis
plt.axis([-2,20, 0,500])
# plt.ylim(0,1000)
plt.xlabel('trip_distance')
plt.title("trip distance")
plt.show();
```



Scatterplot of trip_distance vs fare amount , all histogram above scatterplot

In [36]:
```
max_trip_distance = taxi_data["trip_distance"].max()
max_trip_distance
```

Out[36]: 184.4

In [37]:
```
min_trip_distance = taxi_data["trip_distance"].min()
min_trip_distance
```

Out[37]: 0.0

In [42]:
```
mean_trip_distance = round(taxi_data["trip_distance"].mean(),3)
mean_trip_distance
```

Out[42]: 3.034

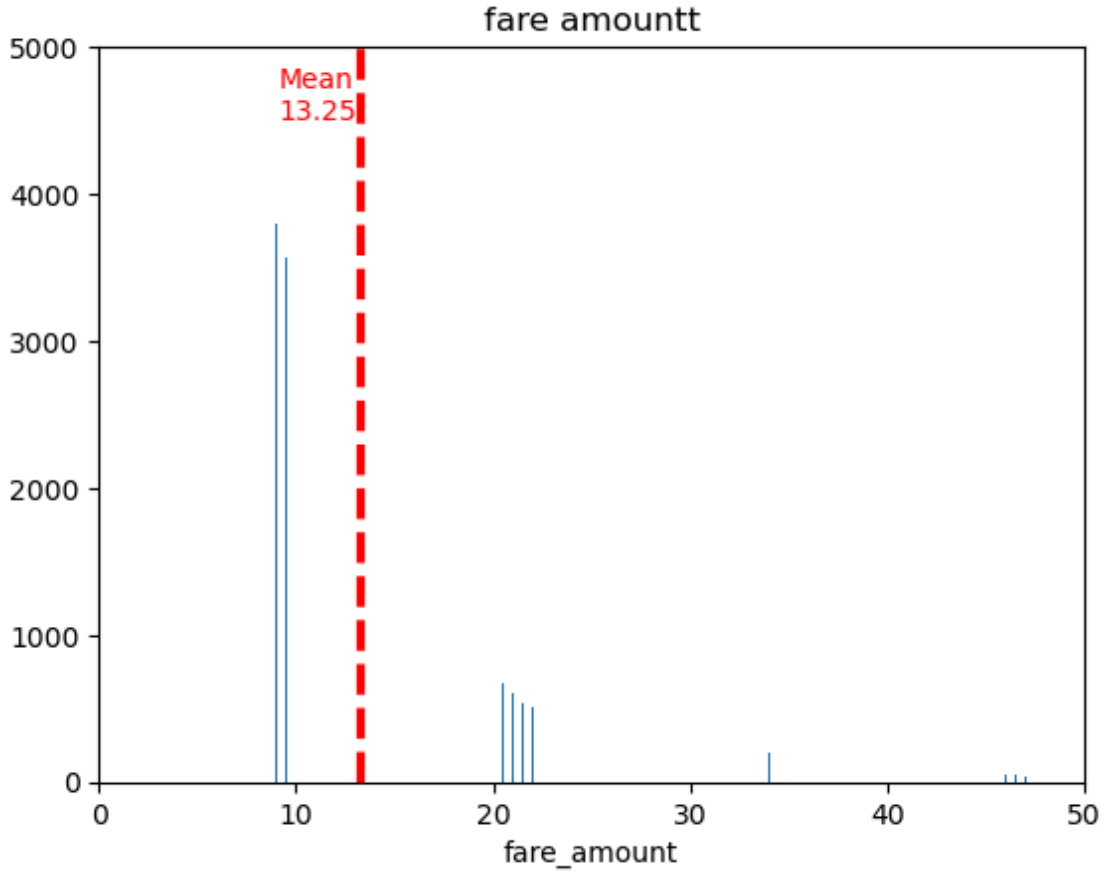
In [77]:
```
median_trip_distance = taxi_data["trip_distance"].median()
median_trip_distance
```

Out[77]: 1.67

In [ ]:
```

```

Valid and Invalid(negative/zero)

In [51]:
```
plt.hist(taxi_data["fare_amount"],bins = 100000);
plt.axvline(x=mean_fare_amount, color="red",linestyle="--",linewidth=3
# plt.axvline(x=taxi_data["fare_amount"].median(), color="orange",line
plt.text(s=f'Mean\n{mean_fare_amount}', x=0.69 * taxi_data["fare_amoun
# plt.axis([-50,50, 0,5000])# try this
plt.axis([0,50, 0,5000])
plt.xlabel('fare_amount')
plt.title("fare amountt")
# plt.ylim(0,1000)
plt.show();
```



In [45]:
```
max_fare_amount = taxi_data["fare_amount"].max()
max_fare_amount
```

Out[45]: 819.5

In [46]:
```
min_fare_amount = taxi_data["fare_amount"].min()
min_fare_amount
```

Out[46]: -47.0

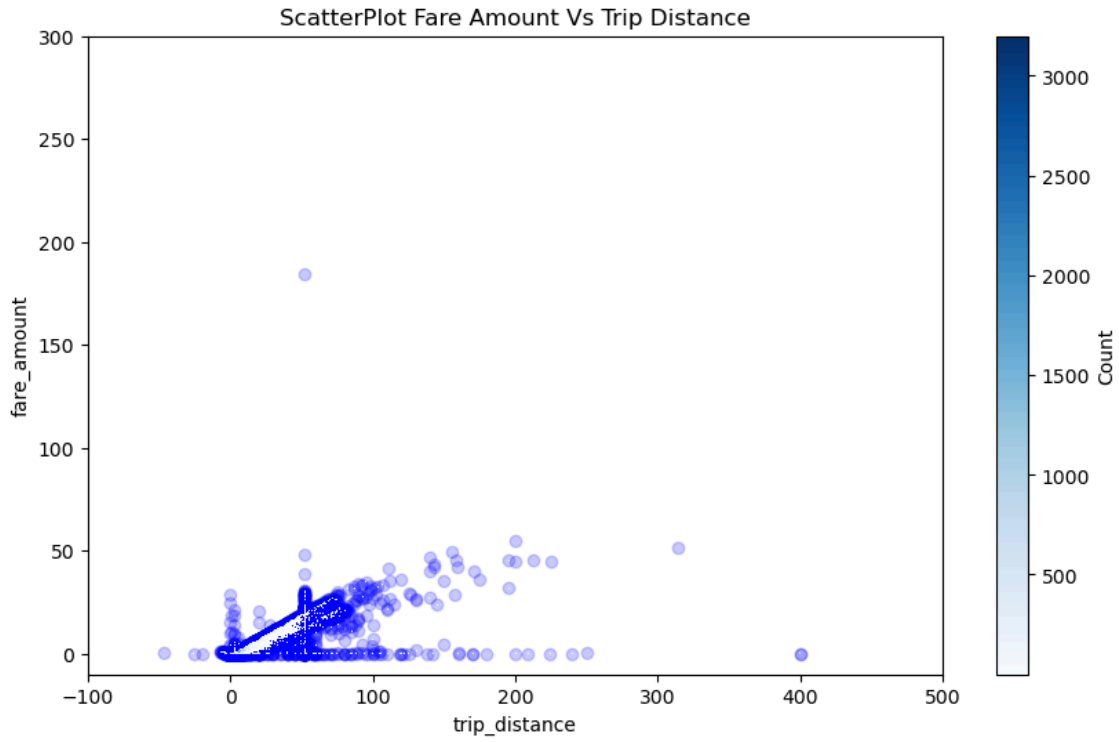
In [50]:
```
mean_fare_amount = round(taxi_data["fare_amount"].mean(),2)
mean_fare_amount
```

Out[50]: 13.25

In [ ]:
```
median_fare_amount = taxi_data["fare_amount"].median()
median_fare_amount
```

In [ ]:
```

```

In [4]:
```
plt.figure(figsize=(10, 6))
plt.scatter(taxi_data['fare_amount'], taxi_data['trip_distance'], alph


plt.hist2d(taxi_data['fare_amount'], taxi_data['trip_distance'], bins=
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.title('ScatterPlot Fare Amount Vs Trip Distance')
plt.axis([-100,500,-10,300])
plt.show()
```



In [42]:
```
rmd_invalid_data = taxi_data[
    (taxi_data['trip_distance'] >= 0) &
    (taxi_data['fare_amount'] >= 0) &
    (taxi_data['trip_distance'] <= 400)
]
```

In [43]:
```
rmd_invalid_data.shape

```

Out[43]: (99930, 19)

In [44]:
```
100000-99930
```

Out[44]: 70

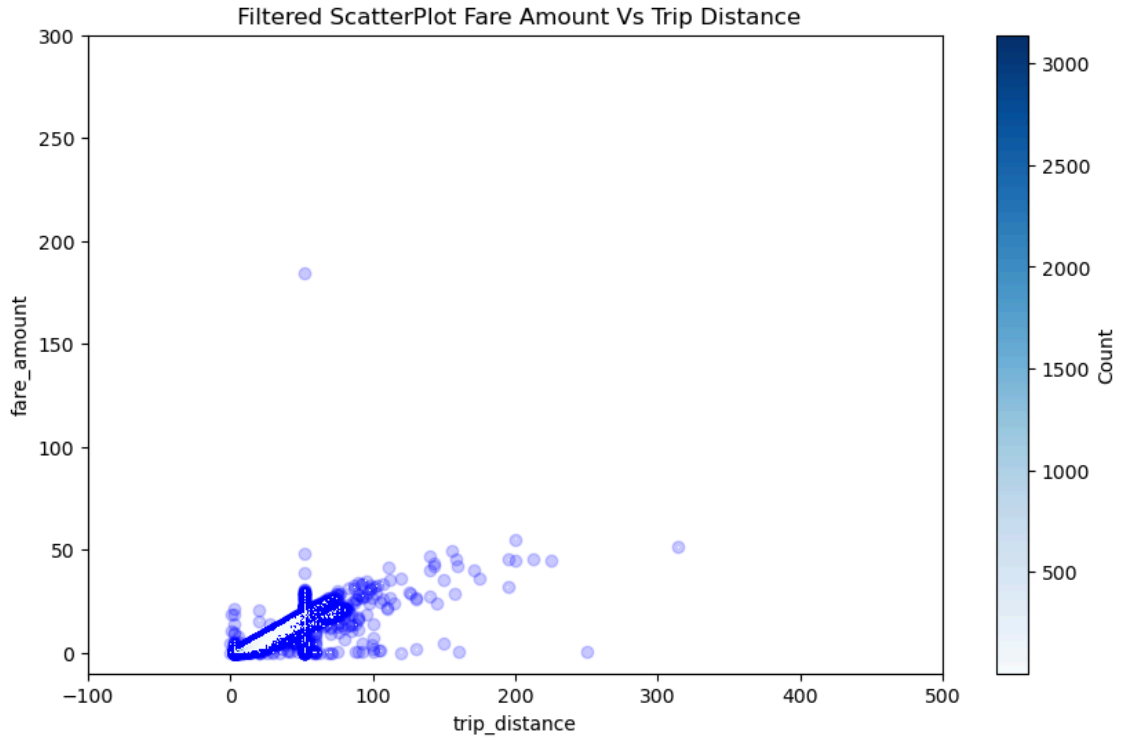
In [30]:
```
#
filter_condition = (
    (taxi_data['trip_distance'] >= 0) & (taxi_data['trip_distance'] <=
    (taxi_data['fare_amount'] >= 0)
)



```

In [37]:
```
# filtered_fare_trip_dist = taxi_data[filter_condition]

fare_trip_rows_removed = taxi_data.shape[0] - rmd_invalid_data.shape[0
rmd_invalid_data.reset_index(drop=True, inplace=True)

# print(f"Number of rows removed: {fare_trip_rows_removed}")
print(f"Filtered dataset shape: {filtered_fare_trip_dist.shape}")
```

```
Filtered dataset shape: (99930, 19)
```

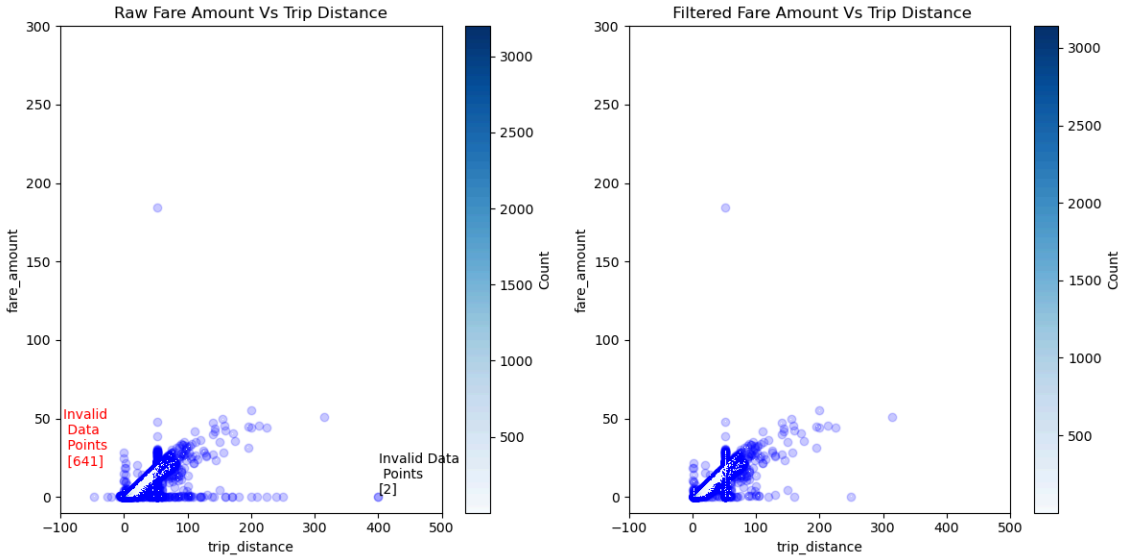
In [91]:

```
plt.figure(figsize=(10, 6))
plt.scatter(filtered_fare_trip_dist['fare_amount'], filtered_fare_trip


plt.hist2d(filtered_fare_trip_dist['fare_amount'], filtered_fare_trip_
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.title(' Filtered ScatterPlot Fare Amount Vs Trip Distance')
plt.axis([-100,500,-10,300])
plt.show()
```



In [104]:

```
plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
plt.scatter(taxi_data['fare_amount'], taxi_data['trip_distance'], alph


plt.hist2d(taxi_data['fare_amount'], taxi_data['trip_distance'], bins=
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.text(s='# Invalid Data\n Points \n:2',x=401,y=2,size=10,color='bla
plt.text(s='# Invalid \n Data\n Points \n :641',x=-95,y=20,size=10,col
plt.title('Raw Fare Amount Vs Trip Distance')
plt.axis([-100,500,-10,300])

plt.subplot(1,2,2)
plt.scatter(filtered_fare_trip_dist['fare_amount'], filtered_fare_trip


plt.hist2d(filtered_fare_trip_dist['fare_amount'], filtered_fare_trip_
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.title(' Filtered Fare Amount Vs Trip Distance')
plt.axis([-100,500,-10,300])
plt.tight_layout()
plt.show()
```

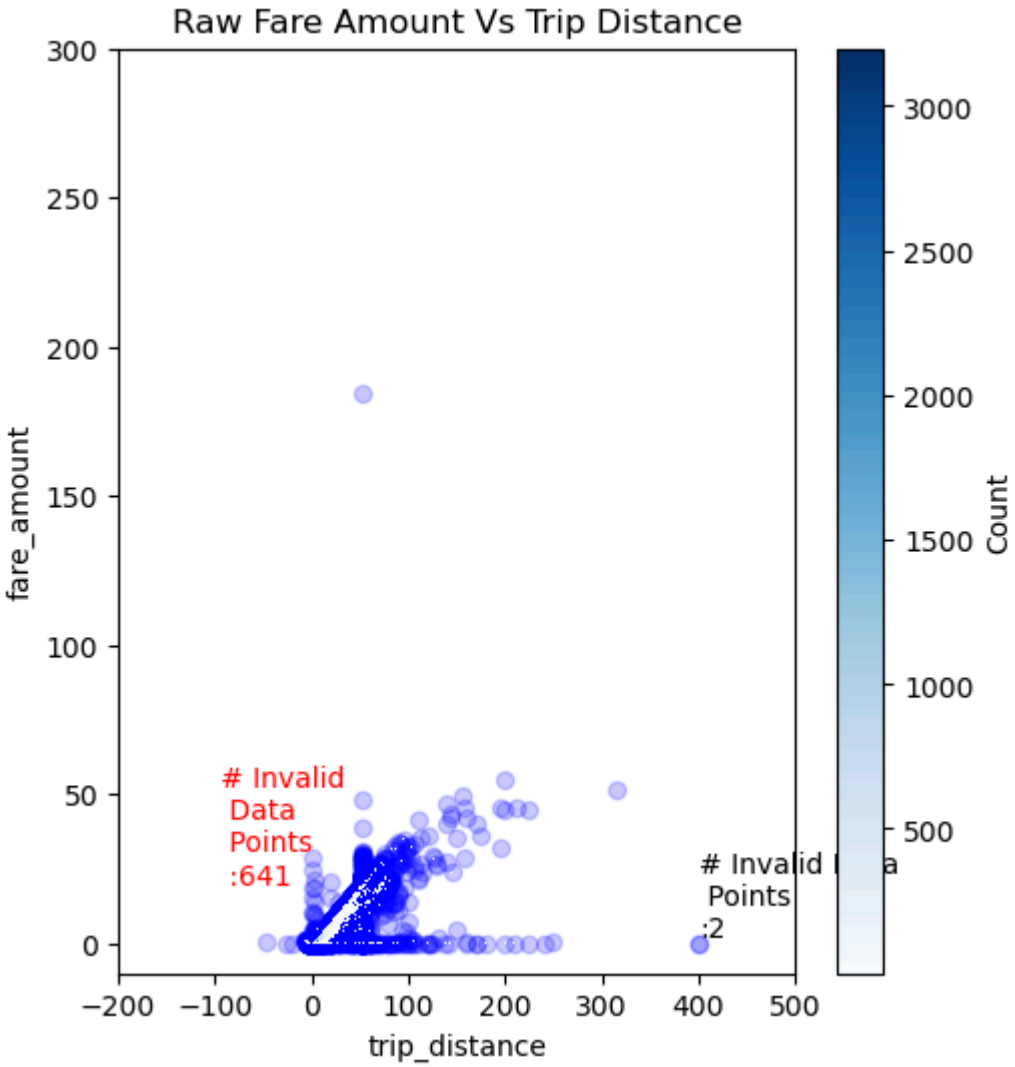


In [ ]:

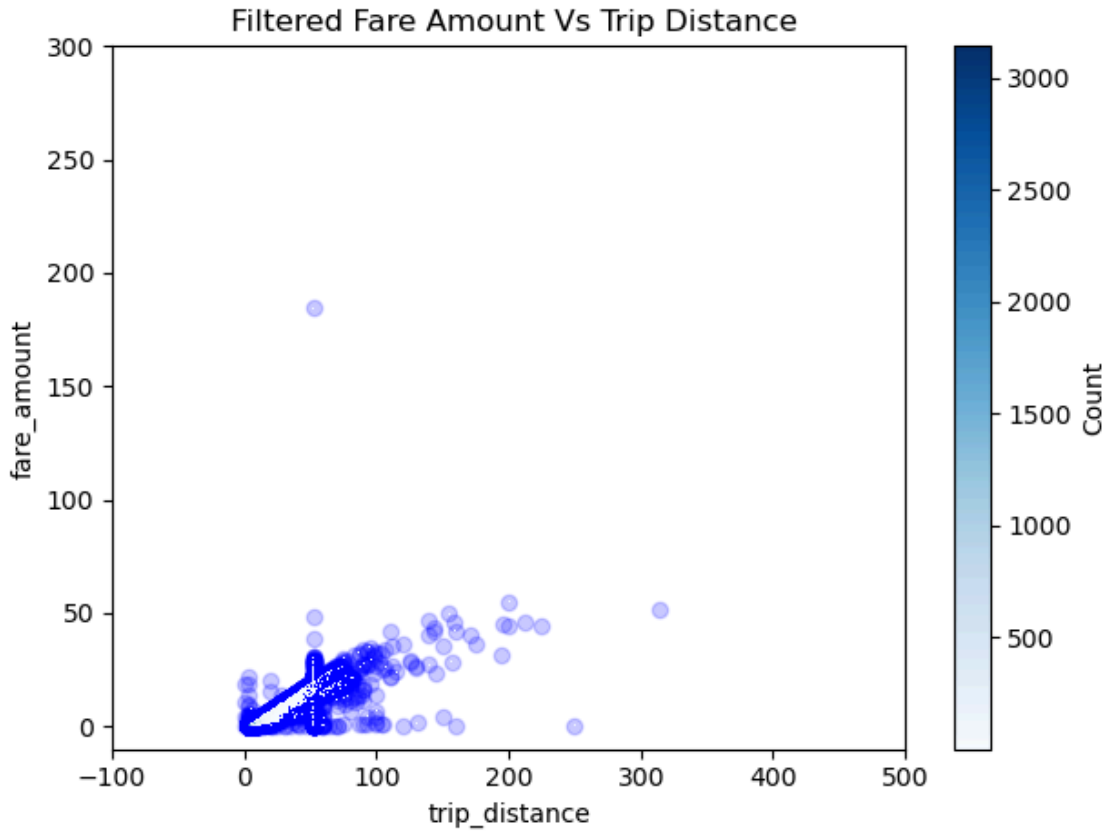
```
1
```

In [58]:
```
plt.figure(figsize=(12, 6))
plt.subplot(1,2,1)
plt.scatter(taxi_data['fare_amount'], taxi_data['trip_distance'], alph


plt.hist2d(taxi_data['fare_amount'], taxi_data['trip_distance'], bins=
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.text(s='# Invalid Data\n Points \n:2',x=401,y=2,size=10,color='bla
plt.text(s='# Invalid \n Data\n Points \n :641',x=-95,y=20,size=10,col
plt.title('Raw Fare Amount Vs Trip Distance')
plt.axis([-200,500,-10,300])

```

Out[58]: (-200.0, 500.0, -10.0, 300.0)



In [57]:
```
plt.scatter(new_taxi_data['fare_amount'], new_taxi_data['trip_distance


plt.hist2d(new_taxi_data['fare_amount'], new_taxi_data['trip_distance'
plt.colorbar(label='Count')

plt.xlabel('trip_distance')
plt.ylabel('fare_amount')
plt.title(' Filtered Fare Amount Vs Trip Distance')
plt.axis([-100,500,-10,300])
plt.tight_layout()
plt.show()
```



In [51]:
```
rmd_invalid_data = taxi_data[
    (taxi_data['trip_distance'] != 0) &
    (taxi_data['fare_amount'] !=0)]
```

In [52]:
```
rmd_invalid_data.shape
```

Out[52]: (99409, 19)

In [54]:
```
new_taxi_data = taxi_data[
    (taxi_data['fare_amount'] > 0) &  # Remove rows with 0 or negative
    (taxi_data['trip_distance'] > 0) &  # Remove rows with 0 or negati
    (taxi_data['trip_distance'] < 400)  # Remove rows with trip_distan
]
```

In [55]:
```
1 new_taxi_data.shape
```

Out[55]: (99359, 19)

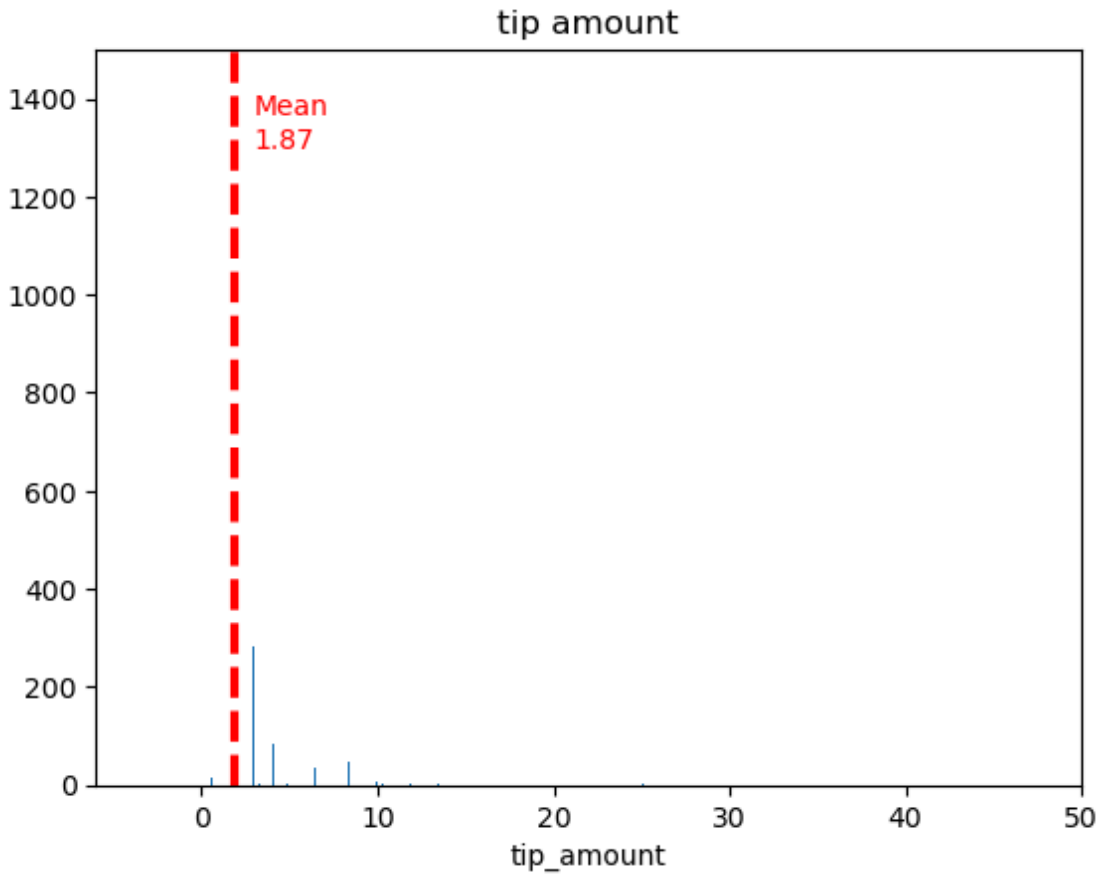
In [56]:
```
1 100000-99359
```

Out[56]: 641

In [ ]:
```
1
```

In [ ]:
```
1
```

In [29]:
```
1 plt.hist(taxi_data["tip_amount"],bins = 100000);
2 plt.axvline(x=mean_tip_amount, color="red",linestyle="--",linewidth=3.
3 # plt.axvline(x=taxi_data["tip_amount"].median(), color="orange",lines
4 plt.text(s=f'Mean\n{mean_tip_amount}', x=3, y=1300, size=10, color='re
5 plt.axis([-6,50, 0,1500])
6 # plt.ylim(0,1000)
7 plt.xlabel('tip_amount')
8 plt.title("tip amount")
9 plt.show();
```



In [25]:
```
1 max_tip_amount = taxi_data["tip_amount"].max()
2 max_tip_amount
```

Out[25]: 125.88

In [26]:
```
min_tip_amount = taxi_data["tip_amount"].min()
min_tip_amount
```

Out[26]: -2.7

In [27]:
```
mean_tip_amount = round(taxi_data["tip_amount"].mean(),2)
mean_tip_amount
```

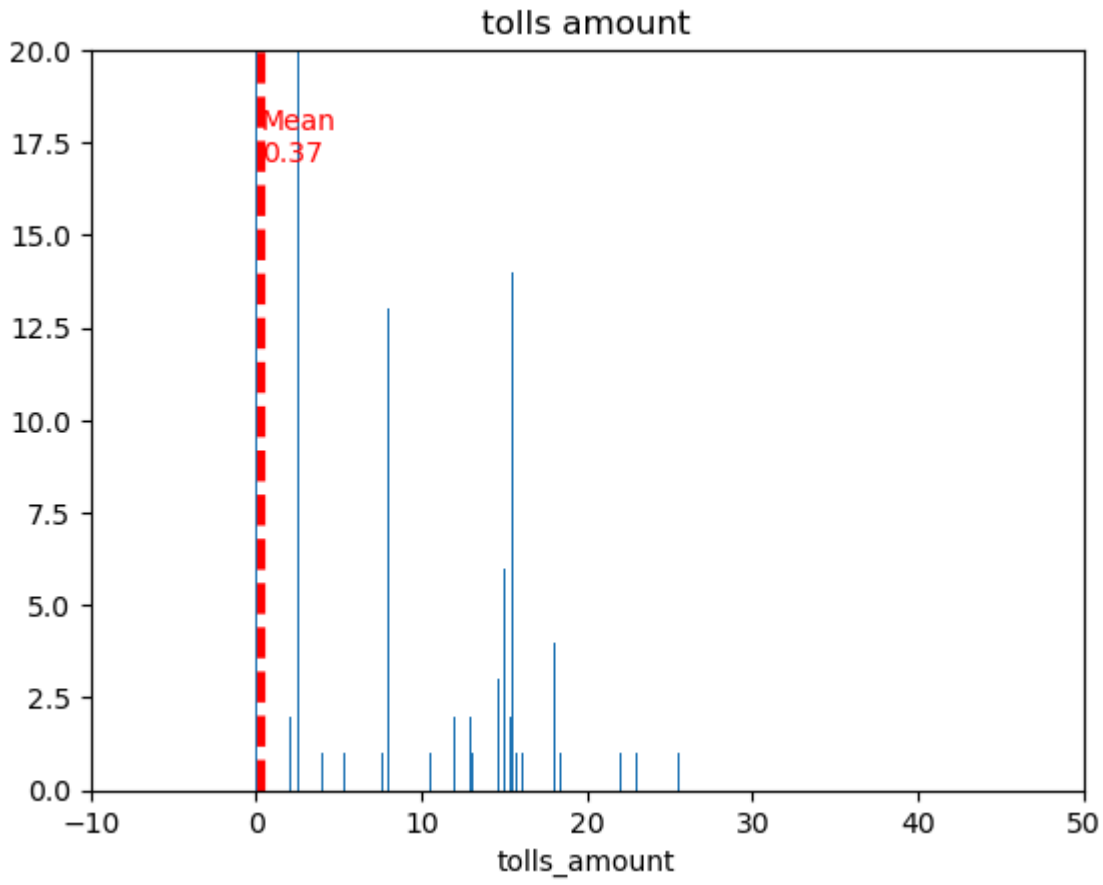
Out[27]: 1.87

In [28]:
```
median_tip_amount = taxi_data["tip_amount"].median()
median_tip_amount
```

Out[28]: 1.36

In [ ]:
```

```

In [73]:
```
plt.hist(taxi_data["tolls_amount"],bins = 1000);
plt.axvline(x=taxi_data["tolls_amount"].mean(), color="red",linestyle=
# plt.axvline(x=taxi_data["tolls_amount"].median(), color="orange",lin
plt.text(s=f'Mean\n{mean_tolls_amount}', x=0.89 * mean_tolls_amount, y
plt.axis([-10,50, 0,20])
# plt.ylim(0,1000)
plt.xlabel('tolls_amount')
plt.title("tolls amount")
plt.show();
```



In [66]:
```
max_tolls_amount = taxi_data["tolls_amount"].max()
max_tolls_amount
```

Out[66]: 25.54

In [67]:
```
min_tolls_amount = taxi_data["tolls_amount"].min()
min_tolls_amount
```

Out[67]: 0.0

In [72]:
```
mean_tolls_amount = round(taxi_data["tolls_amount"].mean(),2)
mean_tolls_amount
```

Out[72]: 0.37

In [69]:
```
median_tolls_amount = taxi_data["tolls_amount"].median()
median_tolls_amount
```

Out[69]: 0.0

In [75]:
```
zero_toll_data = taxi_data[taxi_data['tolls_amount'] == 0]
non_zero_toll_data = taxi_data[taxi_data['tolls_amount'] != 0]
```

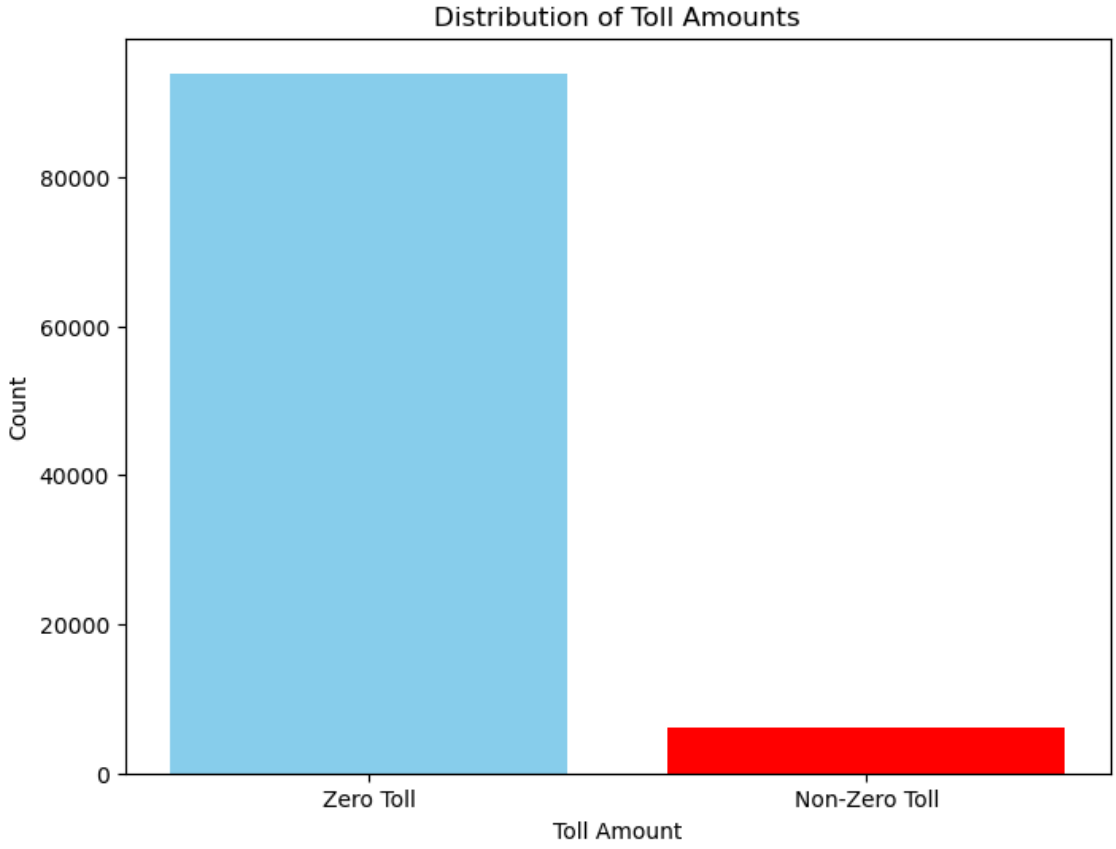
In [78]:
```
zero_toll_data.shape
```

Out[78]: (93880, 19)

In [77]:
```
non_zero_toll_data.shape
```

Out[77]: (6120, 19)

In [76]:
```
plt.figure(figsize=(8, 6))# reduce width of bar
plt.bar(['Zero Toll', 'Non-Zero Toll'], [len(zero_toll_data), len(non_
plt.xlabel('Toll Amount')
plt.ylabel('Count')
plt.title('Distribution of Toll Amounts')
plt.show()
```



Study New York Geography which are the toll highways??

Create a subset of all non-zero toll trips.

Regression in fare_amount = trip_distance, tolls, all others variables)

In [ ]:
```
1
```

separate data and time

In [10]:
```
taxi_data['tpep_dropoff_datetime'] = pd.to_datetime(taxi_data['tpep_dr

taxi_data['date'] = taxi_data['tpep_dropoff_datetime'].dt.date
taxi_data['time'] = taxi_data['tpep_dropoff_datetime'].dt.time

# dates_table = taxi_data[['date']].drop_duplicates().reset_index(drop
# times_table = taxi_data[['time']].drop_duplicates().reset_index(drop

# dates_table.to_csv("dates_table.csv", index=False)
# times_table.to_csv("times_table.csv", index=False)

# print("Dates Table:")
# print(dates_table)

# print("\nTimes Table:")
# print(times_table)
```

In [5]:
```
taxi_data['time']
```

Out[5]:
```
0        00:07:00
1        00:11:00
2        00:31:00
3        00:00:00
4        00:00:00
           ...
99995    06:22:00
99996    06:32:00
99997    06:37:00
99998    06:22:00
99999    06:22:00
Name: time, Length: 100000, dtype: object
```

In [83]:
```
taxi_data['date']
```

Out[83]:
```
0        2016-01-03
1        2016-01-03
2        2016-01-03
3        2016-01-03
4        2016-01-03
           ...
99995    2016-01-03
99996    2016-01-03
99997    2016-01-03
99998    2016-01-03
99999    2016-01-03
Name: date, Length: 100000, dtype: object
```

In [84]:
```
unique_dates = taxi_data['date'].unique()

print(unique_dates)
```

```
[datetime.date(2016, 1, 3) datetime.date(2016, 10, 3)
 datetime.date(2016, 11, 3) datetime.date(2016, 2, 3)]
```

In [9]:
```
dates_table = taxi_data[['date']].drop_duplicates().reset_index(drop=T
dates_table
```

Out[9]:

	date
0	2016-01-03
1	2016-10-03
2	2016-11-03
3	2016-02-03

In [85]:
```
times_table = taxi_data[['time']].drop_duplicates().reset_index(drop=T
```

In [ ]:
```

```

In [16]:
```
dates_table
```

Out[16]:

	date
0	2016-01-03
1	2016-10-03
2	2016-11-03
3	2016-02-03

In [17]:
```
times_table
```

Out[17]:

	time
0	00:07:00
1	00:11:00
2	00:31:00
3	00:00:00
4	00:16:00
...	...
942	15:57:00
943	15:42:00
944	16:09:00
945	07:07:00
946	06:55:00

947 rows × 1 columns

In [11]:
```
taxi_data['tpep_dropoff_datetime'] = pd.to_datetime(taxi_data['tpep_dr

taxi_data['date'] = taxi_data['tpep_dropoff_datetime'].dt.date
taxi_data['time'] = taxi_data['tpep_dropoff_datetime'].dt.time

separate_time_date = taxi_data.copy()
separate_time_date
```

Out[11]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
0	1	01-03-2016 00:00	2016-01-03 00:07:00	1	2.5
1	1	01-03-2016 00:00	2016-01-03 00:11:00	1	2.9
2	2	01-03-2016 00:00	2016-01-03 00:31:00	2	19.9
3	2	01-03-2016 00:00	2016-01-03 00:00:00	3	10.7
4	2	01-03-2016 00:00	2016-01-03 00:00:00	5	30.4
...	...	...	...	...	
99995	1	01-03-2016 06:17	2016-01-03 06:22:00	1	0.5
99996	1	01-03-2016 06:17	2016-01-03 06:32:00	1	3.4
99997	1	01-03-2016 06:17	2016-01-03 06:37:00	1	9.7
99998	2	01-03-2016 06:17	2016-01-03 06:22:00	1	0.9
99999	1	01-03-2016 06:17	2016-01-03 06:22:00	1	1.0

100000 rows × 21 columns

In [34]:
```
1 taxi_data.head(10)
```

Out[34]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	p
0	1	01-03-2016 00:00	2016-01-03 00:07:00	1	2.50	
1	1	01-03-2016 00:00	2016-01-03 00:11:00	1	2.90	
2	2	01-03-2016 00:00	2016-01-03 00:31:00	2	19.98	
3	2	01-03-2016 00:00	2016-01-03 00:00:00	3	10.78	
4	2	01-03-2016 00:00	2016-01-03 00:00:00	5	30.43	
5	2	01-03-2016 00:00	2016-01-03 00:00:00	5	5.92	
6	2	01-03-2016 00:00	2016-01-03 00:00:00	6	5.72	
7	1	01-03-2016 00:00	2016-01-03 00:16:00	1	6.20	
8	1	01-03-2016 00:00	2016-01-03 00:05:00	1	0.70	
9	2	01-03-2016 00:00	2016-01-03 00:24:00	3	7.18	

10 rows × 21 columns

In [12]:
```
1 taxi_data = taxi_data.sort_values(by=['date', 'time'], ascending=True)
```

In [21]:
```
1 taxi_data.head(20)
```

Out[21]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance
0	2	2016-01-03 00:00:00	2016-01-03 00:00:00	3	10.78
1	2	2016-01-03 00:00:00	2016-01-03 00:00:00	5	30.43
2	2	2016-01-03 00:00:00	2016-01-03 00:00:00	5	5.92
3	2	2016-01-03 00:00:00	2016-01-03 00:00:00	6	5.72
4	1	2016-01-03 00:00:00	2016-01-03 00:01:00	2	0.20
5	2	2016-01-03 00:01:00	2016-01-03 00:01:00	1	0.03
6	2	2016-01-03 00:00:00	2016-01-03 00:02:00	2	0.54
7	2	2016-01-03 00:00:00	2016-01-03 00:02:00	2	0.80
8	2	2016-01-03 00:00:00	2016-01-03 00:02:00	2	0.49
9	2	2016-01-03 00:00:00	2016-01-03 00:02:00	1	0.95
10	2	2016-01-03 00:00:00	2016-01-03 00:02:00	1	0.42
11	2	2016-01-03 00:01:00	2016-01-03 00:02:00	1	0.35
12	2	2016-01-03 00:01:00	2016-01-03 00:02:00	1	0.00
13	2	2016-01-03 00:02:00	2016-01-03 00:02:00	1	0.00
14	1	2016-01-03 00:02:00	2016-01-03 00:02:00	1	0.00
15	1	2016-01-03 00:00:00	2016-01-03 00:03:00	1	1.10
16	1	2016-01-03 00:00:00	2016-01-03 00:03:00	2	0.50
17	1	2016-01-03 00:00:00	2016-01-03 00:03:00	1	1.00
18	2	2016-01-03 00:00:00	2016-01-03 00:03:00	1	1.21
19	2	2016-01-03 00:00:00	2016-01-03 00:03:00	1	0.48

20 rows × 21 columns

In [22]:
```
1 taxi_data.tail(10)
```

Type *Markdown* and LaTeX: α^2

In [13]:
```
1 date_time_mapping = taxi_data.groupby('date')['time'].apply(list)
```

In [37]:
```
1 date_time_mapping.head
```

Out[37]:
```
<bound method NDFrame.head of date
2016-01-03    [00:00:00, 00:00:00, 00:00:00, 00:00:00, 00:01...
2016-02-03    [00:00:00, 00:00:00, 00:00:00, 00:00:00, 00:00...
2016-10-03    [07:08:00, 07:09:00, 07:09:00, 07:09:00, 07:09...
2016-11-03    [00:00:00, 00:00:00, 00:00:00, 00:00:00, 00:00...
Name: time, dtype: object>
```

In [14]:
```
1 date_time_range = (
2     taxi_data.groupby('date')['time']
3     .apply(lambda times: f"[{min(times)} - {max(times)}]")
4     .sort_index()
5 )
```

In [9]:
```
1 date_time_range
```

Out[9]:
```
date
2016-01-03    [00:00:00 - 23:59:00]
2016-02-03    [00:00:00 - 06:14:00]
2016-10-03    [07:08:00 - 18:32:00]
2016-11-03    [00:00:00 - 14:19:00]
Name: time, dtype: object
```

datetime.date(2016, 1, 3) datetime.date(2016, 10, 3) datetime.date(2016, 11, 3) datetime.date(2016, 2, 3)]

In [91]:
```
1 dates_table = pd.DataFrame({"date": ["2016-01-03", "2016-02-03  ", "20
2 dates_table['date'] = pd.to_datetime(dates_table['date'].str.strip()).
3 dates_table['day_name'] = dates_table['date'].dt.day_name()
```

In [92]:
```
1 dates_table
```

Out[92]:

	date	day_name
0	2016-01-02 19:00:00-05:00	Saturday
1	2016-02-02 19:00:00-05:00	Tuesday
2	2016-10-02 20:00:00-04:00	Sunday
3	2016-11-02 20:00:00-04:00	Wednesday

In [83]:
```
1 type(taxi_data['tpep_dropoff_datetime'])
2 np.hstack(dir(taxi_data['tpep_dropoff_datetime']))
```

...

In [93]:
```
new_order_dataset= [1,19,20,2]
new_order_dataset = taxi_data.iloc[:, new_order_dataset]
```

In [94]:
```
new_order_dataset.head(10)
```

Out[94]:

	tpep_pickup_datetime	time_frame	date	tpep_dropoff_datetime
0	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:07:00
1	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:11:00
2	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:31:00
3	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:00:00
4	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:00:00
5	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:00:00
6	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:00:00
7	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:16:00
8	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:05:00
9	2016-01-03	2016-01-03	2016-01-03	2016-01-03 00:24:00

Note: Extracted date format is yyyy-mm-dd

In [95]:
```
# Convert 'tpep_pickup_datetime' to datetime if not already
taxi_data['tpep_pickup_datetime'] = pd.to_datetime(taxi_data['tpep_pic

# Create 15-minute time frames
taxi_data['time_frame'] = taxi_data['tpep_pickup_datetime'].dt.floor('

# Extract the date for grouping and sorting
taxi_data['date'] = taxi_data['time_frame'].dt.date

# Group by date and time_frame, then count pickups
pickup_counts = (
    taxi_data.groupby(['date', 'time_frame'])
    .size()
    .reset_index(name='pickup_count')
)

print(pickup_counts)

```

. . .

In [15]:
```
# Convert 'tpep_pickup_datetime' to datetime if not already
taxi_data['tpep_pickup_datetime'] = pd.to_datetime(taxi_data['tpep_pic

# Create 15-minute time frames
taxi_data['time_frame'] = taxi_data['tpep_pickup_datetime'].dt.floor('

# Extract the date for grouping
taxi_data['date'] = taxi_data['time_frame'].dt.date

# Group by date and time_frame, count pickups, and sum passenger_count
pickup_summary = (
    taxi_data.groupby(['date', 'time_frame'])
    .agg(
        pickup_count=('time_frame', 'size'),
        total_passenger_count=('passenger_count', 'sum')
    )
    .reset_index()
)

print(pickup_summary)

```

	date	time_frame	pickup_count	total_passenger_coun t
0	2016-01-03	2016-01-03 00:00:00	2189	3577
1	2016-01-03	2016-01-03 00:15:00	1802	2908
2	2016-01-03	2016-01-03 00:30:00	1636	2593
3	2016-01-03	2016-01-03 00:45:00	1452	2397
4	2016-01-03	2016-01-03 01:00:00	1262	2030
5	2016-01-03	2016-01-03 01:15:00	1103	1816
6	2016-01-03	2016-01-03 01:30:00	943	1566
7	2016-01-03	2016-01-03 01:45:00	840	1320
8	2016-01-03	2016-01-03 02:00:00	760	1260

In [11]:
```
unique_dates = separate_time_date['date'].unique()

print(unique_dates)
```

```
[datetime.date(2016, 1, 3) datetime.date(2016, 10, 3)
 datetime.date(2016, 11, 3) datetime.date(2016, 2, 3)]
```

In [10]:
```
separate_time_date.head(2)
```

Out[10]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distance	p
0	1	01-03-2016 00:00	2016-01-03 00:07:00	1	2.5	
1	1	01-03-2016 00:00	2016-01-03 00:11:00	1	2.9	

2 rows × 21 columns

In [16]:
```
# Convert 'tpep_pickup_datetime' to datetime if not already
separate_time_date['date'] = pd.to_datetime(separate_time_date['date']

# Filter rows for the specific date and time range
data_2016_01_03 = separate_time_date[
    (separate_time_date['date'].dt.date == pd.Timestamp('2016-01-03').
]

data_2016_01_03

```

Out[16]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
0	1	01-03-2016 00:00	2016-01-03 00:07:00	1	2.5
1	1	01-03-2016 00:00	2016-01-03 00:11:00	1	2.9
2	2	01-03-2016 00:00	2016-01-03 00:31:00	2	19.9
3	2	01-03-2016 00:00	2016-01-03 00:00:00	3	10.7
4	2	01-03-2016 00:00	2016-01-03 00:00:00	5	30.4
...	...	...	...	...	
99995	1	01-03-2016 06:17	2016-01-03 06:22:00	1	0.5
99996	1	01-03-2016 06:17	2016-01-03 06:32:00	1	3.4
99997	1	01-03-2016 06:17	2016-01-03 06:37:00	1	9.7
99998	2	01-03-2016 06:17	2016-01-03 06:22:00	1	0.9
99999	1	01-03-2016 06:17	2016-01-03 06:22:00	1	1.0

23192 rows × 21 columns

In [8]:
```
1 data_2016_01_03.shape
```

Out[8]: (23192, 21)

In [17]:
```
1  # Convert 'tpep_pickup_datetime' to datetime if not already
2  taxi_data['date'] = pd.to_datetime(taxi_data['date'])
3
4  # Filter rows for the specific date '2016-10-03'
5  data_2016_10_03 = taxi_data[
6      taxi_data['date'].dt.date == pd.Timestamp('2016-10-03 ').date()
7  ]
8
9  data_2016_10_03
10
```

Out[17]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
23220	2	2016-10-03 07:08:00	2016-10-03 07:08:00	5	0.(
23221	2	2016-10-03 07:07:00	2016-10-03 07:09:00	1	0.4
23222	2	2016-10-03 07:07:00	2016-10-03 07:09:00	1	0.4
23223	2	2016-10-03 07:07:00	2016-10-03 07:09:00	5	0.4
23224	2	2016-10-03 07:07:00	2016-10-03 07:09:00	1	0.3
...	...	...	...	...	
99995	2	2016-10-03 14:08:00	2016-11-03 14:00:00	2	0.7
99996	2	2016-10-03 14:09:00	2016-11-03 14:00:00	1	0.(
99997	2	2016-10-03 14:11:00	2016-11-03 14:03:00	1	0.3
99998	2	2016-10-03 14:15:00	2016-11-03 14:10:00	1	20.9
99999	2	2016-10-03 14:22:00	2016-11-03 14:19:00	1	0.8

76780 rows × 22 columns

In [10]:
```
1 data_2016_10_03.shape
```

Out[10]: (76780, 22)

In [18]:
```
# Convert 'tpep_pickup_datetime' to datetime if not already
separate_time_date['date'] = pd.to_datetime(separate_time_date['date']

# Filter rows for the specific date '2016-11-03'
data_2016_11_03 = taxi_data[
    separate_time_date['date'].dt.date == pd.Timestamp('2016-11-03 ').
]

data_2016_11_03
```

Out[18]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
927	1	2016-01-03 00:02:00	2016-01-03 00:16:00	1	9.2
2135	1	2016-01-03 00:11:00	2016-01-03 00:26:00	1	5.2
2273	1	2016-01-03 00:17:00	2016-01-03 00:27:00	1	4.2
2705	1	2016-01-03 00:27:00	2016-01-03 00:30:00	1	0.4
3132	1	2016-01-03 00:18:00	2016-01-03 00:34:00	1	3.7
...	...	...	...	...	
97127	2	2016-10-03 14:23:00	2016-10-03 14:28:00	2	0.8
97487	2	2016-10-03 14:02:00	2016-10-03 14:31:00	1	4.8
98716	2	2016-10-03 14:06:00	2016-10-03 14:42:00	1	15.1
98741	2	2016-10-03 14:20:00	2016-10-03 14:42:00	4	2.1
99391	2	2016-10-03 14:21:00	2016-10-03 14:54:00	1	7.4

164 rows × 22 columns

In [12]:
```
data_2016_11_03.shape
```

Out[12]: (164, 22)

In [19]:
```
taxi_data=pd.read_csv(r"C:\Users\Sanjoy\Desktop\Bittu's File\SEM V\AI\
```

In [20]:
```
taxi_data['tpep_dropoff_datetime'] = pd.to_datetime(taxi_data['tpep_dr

taxi_data['date'] = taxi_data['tpep_dropoff_datetime'].dt.date
taxi_data['time'] = taxi_data['tpep_dropoff_datetime'].dt.time

separate_time_date = taxi_data.copy()
separate_time_date
```

Out[20]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
0	1	01-03-2016 00:00	2016-01-03 00:07:00	1	2.5
1	1	01-03-2016 00:00	2016-01-03 00:11:00	1	2.9
2	2	01-03-2016 00:00	2016-01-03 00:31:00	2	19.9
3	2	01-03-2016 00:00	2016-01-03 00:00:00	3	10.7
4	2	01-03-2016 00:00	2016-01-03 00:00:00	5	30.4
...	...	...	...	...	
99995	1	01-03-2016 06:17	2016-01-03 06:22:00	1	0.5
99996	1	01-03-2016 06:17	2016-01-03 06:32:00	1	3.4
99997	1	01-03-2016 06:17	2016-01-03 06:37:00	1	9.7
99998	2	01-03-2016 06:17	2016-01-03 06:22:00	1	0.9
99999	1	01-03-2016 06:17	2016-01-03 06:22:00	1	1.0

100000 rows × 21 columns

In [21]:

```
# Convert 'tpep_pickup_datetime' to datetime if not already
taxi_data['date'] = pd.to_datetime(taxi_data['date'])

# Filter rows for the specific date '2016-02-03'
data_2016_02_03 = taxi_data[
    taxi_data['date'].dt.date == pd.Timestamp('2016-02-03 ').date()
]

data_2016_02_03
```

Out[21]:

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
65630	2	01-03-2016 00:07	2016-02-03 00:04:00	3	2.2
70985	2	01-03-2016 00:18	2016-02-03 00:05:00	6	0.7
71992	2	01-03-2016 00:27	2016-02-03 00:02:00	2	6.8
73078	2	01-03-2016 00:36	2016-02-03 00:23:00	1	2.3
73540	2	01-03-2016 00:40	2016-02-03 00:37:00	1	0.6
73677	2	01-03-2016 00:42	2016-02-03 00:00:00	1	17.4
76283	2	01-03-2016 01:10	2016-02-03 00:54:00	1	1.3
79440	2	01-03-2016 01:15	2016-02-03 01:03:00	1	2.1
79849	2	01-03-2016 01:20	2016-02-03 00:00:00	1	1.5
80908	2	01-03-2016 01:35	2016-02-03 00:47:00	1	1.0
81371	2	01-03-2016 01:43	2016-02-03 00:00:00	1	1.4
82102	2	01-03-2016 01:55	2016-02-03 00:00:00	1	6.5
84565	2	01-03-2016 02:50	2016-02-03 02:14:00	1	2.8
84920	2	01-03-2016 03:00	2016-02-03 01:49:00	1	2.3
84947	2	01-03-2016 03:01	2016-02-03 00:00:00	1	1.2
85351	2	01-03-2016 03:13	2016-02-03 02:40:00	1	1.1
86431	2	01-03-2016 03:47	2016-02-03 00:00:00	1	7.5
87201	2	01-03-2016 04:12	2016-02-03 04:09:00	2	10.7
87574	2	01-03-2016 04:24	2016-02-03 03:55:00	1	7.0
90009	2	01-03-2016 05:30	2016-02-03 04:51:00	1	9.7
90237	2	01-03-2016 05:34	2016-02-03 00:00:00	4	3.2
91238	2	01-03-2016 05:47	2016-02-03 00:00:00	1	3.4
91331	2	01-03-2016 05:47	2016-02-03 05:45:00	1	2.4
91471	2	01-03-2016 05:49	2016-02-03 05:03:00	2	2.5
91506	2	01-03-2016 05:49	2016-02-03 04:56:00	2	1.6

	VendorID	tpep_pickup_datetime	tpep_dropoff_datetime	passenger_count	trip_distanc
92432	2	01-03-2016 06:00	2016-02-03 00:00:00	2	3.2
99860	2	01-03-2016 06:16	2016-02-03 05:53:00	1	1.8
99923	2	01-03-2016 06:16	2016-02-03 06:14:00	1	2.0

28 rows × 21 columns

In [11]:
```
1 data_2016_02_03.shape
```

Out[11]: (28, 21)

In [22]:
```
1 23192+76616+164+28
```

Out[22]: 100000

In [16]:
```
1 data_2016_02_03.sort_values(by='time')
```

...

In [31]:
```
1 data_2016_01_03.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 23192 entries, 0 to 99999
Data columns (total 21 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   VendorID               23192 non-null  int64
 1   tpep_pickup_datetime   23192 non-null  object
 2   tpep_dropoff_datetime  23192 non-null  datetime64[ns]
 3   passenger_count        23192 non-null  int64
 4   trip_distance          23192 non-null  float64
 5   pickup_longitude       23192 non-null  float64
 6   pickup_latitude        23192 non-null  float64
 7   RatecodeID             23192 non-null  int64
 8   store_and_fwd_flag     23192 non-null  object
 9   dropoff_longitude      23192 non-null  float64
 10  dropoff_latitude       23192 non-null  float64
 11  payment_type           23192 non-null  int64
 12  fare_amount            23192 non-null  float64
 13  extra                  23192 non-null  float64
 14  mta_tax                23192 non-null  float64
 15  tip_amount             23192 non-null  float64
 16  tolls_amount           23192 non-null  float64
 17  improvement_surcharge  23192 non-null  float64
 18  total_amount           23192 non-null  float64
 19  date                   23192 non-null  datetime64[ns]
 20  time                   23192 non-null  object
dtypes: datetime64[ns](2), float64(12), int64(4), object(3)
memory usage: 3.9+ MB
```

data_2016_01_03(Tuesday)

In [22]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_01_03['full_datetime'] = pd.to_datetime(
    data_2016_01_03['date'].astype(str) + ' ' + data_2016_01_03['time'
)

# Create 15-minute time frames
data_2016_01_03['time_frame'] = data_2016_01_03['full_datetime'].dt.fl

# Group by date and time_frame, sum passenger_count, and get mean of l
pickup_summary_data_2016_01_03 = (
    data_2016_01_03.groupby(['date', 'time_frame'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        avg_pickup_longitude=('pickup_longitude', 'mean'),
        avg_pickup_latitude=('pickup_latitude', 'mean'),
        total_trip_distance=('trip_distance', 'sum'),  # Sum of trip d
        total_amount=('total_amount', 'sum')  # Sum of total amounts i
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_01_03)

```

	date	time_frame	total_passenger_count	avg_pickup_longitude	avg_pickup_latitude	total_trip_distance
0	2016-01-03	2016-01-03 00:00:00	1346	-72.513299	39.942912	1267.58
1	2016-01-03	2016-01-03 00:15:00	2890	-73.232010	40.338644	4978.20
2	2016-01-03	2016-01-03 00:30:00	2778	-72.506499	39.935352	7291.48
3	2016-01-03	2016-01-03 00:45:00	2702	-72.829122	40.115409	6350.84
4	2016-01-03	2016-01-03 01:00:00	2182	-72.862679	40.131547	5648.68
5	2016-01-03	2016-01-03 01:15:00	2014	-73.195071	40.315332	5396.70
6	2016-01-03	2016-01-03 01:30:00	1837	-72.914801	40.161534	4873.57
7	2016-01-03	2016-01-03 01:45:00	1521	-73.248420	40.343580	3530.26
8	2016-01-03	2016-01-03 02:00:00	1253	-72.776640	40.082619	2813.28
9	2016-01-03	2016-01-03 02:15:00	1174	-72.861096	40.129602	2567.38
10	2016-01-03	2016-01-03 02:30:00	1034	-72.636912	40.005990	2567.82
11	2016-01-03	2016-01-03 02:45:00	973	-73.485380	40.474177	2146.34
12	2016-01-03	2016-01-03 03:00:00	859	-73.161410	40.297644	1866.60
13	2016-01-03	2016-01-03 03:15:00	852	-73.262725	40.353065	1912.92
14	2016-01-03	2016-01-03 03:30:00	749	-73.471843	40.464190	1728.65
15	2016-01-03	2016-01-03 03:45:00	700	-73.084642	40.254508	1881.30
16	2016-01-03	2016-01-03 04:00:00	744	-73.000728	40.206164	1888.70
17	2016-01-03	2016-01-03 04:15:00	814	-73.209871	40.324610	1954.83
18	2016-01-03	2016-01-03 04:30:00	735	-72.815591	40.109604	2086.72
19	2016-01-03	2016-01-03 04:45:00	765	-72.368237	39.866564	2334.78
20	2016-01-03	2016-01-03 05:00:00	857	-72.532742	39.957702	2421.76
21	2016-01-03	2016-01-03 05:15:00	984	-73.043951	40.240386	2723.19
22	2016-01-03	2016-01-03 05:30:00	1262	-72.696701	40.048725	3111.18
23	2016-01-03	2016-01-03 05:45:00	1910			
24	2016-01-03	2016-01-03 06:00:00	2040			
25	2016-01-03	2016-01-03 06:15:00	1689			
26	2016-01-03	2016-01-03 06:30:00	286			
27	2016-01-03	2016-01-03 06:45:00	97			
28	2016-01-03	2016-01-03 07:00:00	35			
29	2016-01-03	2016-01-03 07:15:00	1			
30	2016-01-03	2016-01-03 07:30:00	1			
31	2016-01-03	2016-01-03 08:00:00	1			
32	2016-01-03	2016-01-03 08:45:00	1			
33	2016-01-03	2016-01-03 20:15:00	5			
34	2016-01-03	2016-01-03 23:45:00	8			

23	-72.804001	40.109563	4149.95
24	-72.874639	40.147171	4489.32
25	-73.291099	40.377586	4728.13
26	-73.516642	40.502296	2306.72
27	-72.883303	40.151500	1394.70
28	-69.919372	38.517345	477.76
29	-73.781860	40.644691	16.20
30	-73.789597	40.642818	38.70
31	-73.955399	40.819908	5.55
32	-73.776680	40.645386	15.60
33	-73.780495	40.645245	18.62
34	-73.974515	40.723671	14.03

	total_amount
0	8368.33
1	25267.07
2	32397.80
3	28575.70
4	24579.98
5	23334.92
6	21148.45
7	16357.86
8	12963.83
9	11548.41
10	11428.28
11	9521.48
12	8465.51
13	8369.66
14	7402.51
15	8488.10
16	8123.42
17	8425.88
18	8622.90
19	9856.98
20	10144.68
21	12283.31
22	13892.27
23	19442.44
24	21103.69
25	20163.04
26	8765.57
27	4902.88
28	1122.22
29	65.32
30	100.88
31	24.36
32	62.16
33	67.09
34	54.96

```
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\3688602739.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['full_datetime'] = pd.to_datetime(
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\3688602739.py:7: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['time_frame'] = data_2016_01_03['full_datetime'].dt.flo
or('15T')
```

In [23]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_01_03['full_datetime'] = pd.to_datetime(data_2016_01_03['dat

# Convert 'time_frame' to datetime format and extract only the time
data_2016_01_03['time_frame'] = pd.to_datetime(data_2016_01_03['time_f
data_2016_01_03['separate_time'] = data_2016_01_03['time_frame'].dt.ti

# Create 15-minute time frames
data_2016_01_03['time_frame'] = data_2016_01_03['full_datetime'].dt.fl

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_01_03 = (
    data_2016_01_03.groupby(['date', 'time_frame','time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_01_03)
```

```
          date          time_frame      time  total_passenger_count  \
0   2016-01-03 2016-01-03 00:00:00  00:00:00                     19
1   2016-01-03 2016-01-03 00:00:00  00:01:00                      3
2   2016-01-03 2016-01-03 00:00:00  00:02:00                     12
3   2016-01-03 2016-01-03 00:00:00  00:03:00                     24
4   2016-01-03 2016-01-03 00:00:00  00:04:00                     40
..         ...                 ...       ...                    ...
435 2016-01-03 2016-01-03 08:45:00  08:51:00                      1
436 2016-01-03 2016-01-03 20:15:00  20:18:00                      5
437 2016-01-03 2016-01-03 23:45:00  23:49:00                      1
438 2016-01-03 2016-01-03 23:45:00  23:57:00                      6
439 2016-01-03 2016-01-03 23:45:00  23:59:00                      1

     median_pickup_longitude  median_pickup_latitude  total_trip_distance
\
0                -73.983162               40.748831                52.85
1                -73.986351               40.754251                 0.23
2                -73.988426               40.757118                 3.55
3                -73.972672               40.755989                 9.95
4                -73.983017               40.753368                14.84
..                      ...                     ...                  ...
435              -73.776680               40.645386                15.60
436              -73.780495               40.645245                18.62
437              -73.988701               40.720154                 9.33
438              -73.987915               40.728355                 2.76
439              -73.946930               40.722504                 1.94

     total_amount
0          210.08
1            8.10
2          268.99
3          133.95
4          143.26
..            ...
435         62.16
436         67.09
437         29.80
438         15.36
439          9.80

[440 rows x 8 columns]
```

```
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\2673600815.py:2: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['full_datetime'] = pd.to_datetime(data_2016_01_03['dat
e'].astype(str) + ' ' + data_2016_01_03['time'].astype(str))
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\2673600815.py:5: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['time_frame'] = pd.to_datetime(data_2016_01_03['time_fr
ame'])
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\2673600815.py:6: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['separate_time'] = data_2016_01_03['time_frame'].dt.tim
e
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\2673600815.py:9: Settin
gWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
  data_2016_01_03['time_frame'] = data_2016_01_03['full_datetime'].dt.flo
or('15T')
```

Range of latt and long

In [33]:
```
1 pickup_summary_data_2016_01_03
```

Out[33]:

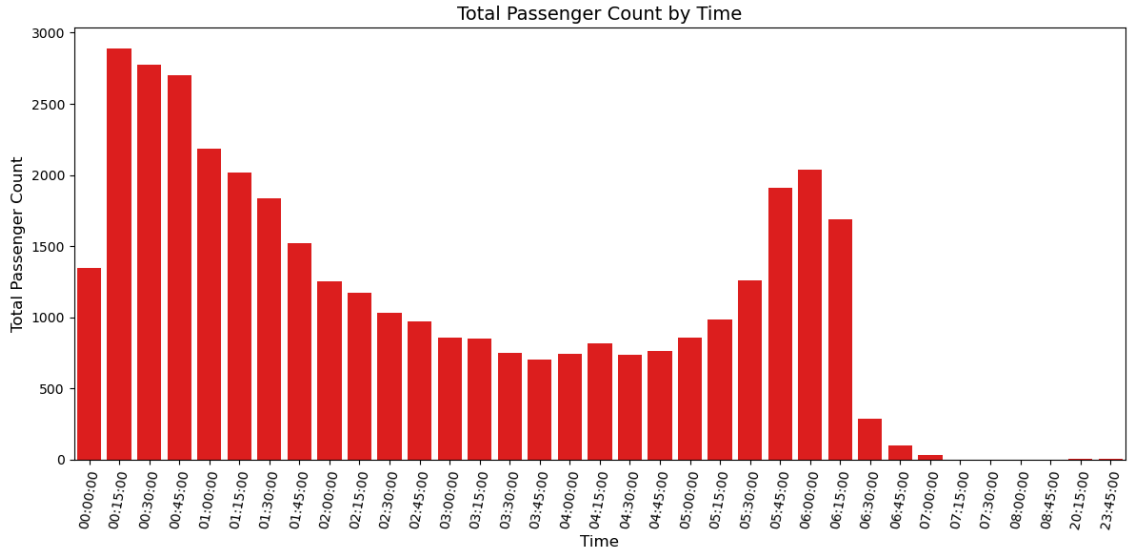
	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pi
0	2016-01-03	2016-01-03 00:00:00	00:00:00	1346	-73.982285	
1	2016-01-03	2016-01-03 00:15:00	00:15:00	2890	-73.982414	
2	2016-01-03	2016-01-03 00:30:00	00:30:00	2778	-73.983223	
3	2016-01-03	2016-01-03 00:45:00	00:45:00	2702	-73.983269	
4	2016-01-03	2016-01-03 01:00:00	01:00:00	2182	-73.984047	
5	2016-01-03	2016-01-03 01:15:00	01:15:00	2014	-73.983940	
6	2016-01-03	2016-01-03 01:30:00	01:30:00	1837	-73.982868	
7	2016-01-03	2016-01-03 01:45:00	01:45:00	1521	-73.984177	
8	2016-01-03	2016-01-03 02:00:00	02:00:00	1253	-73.984863	
9	2016-01-03	2016-01-03 02:15:00	02:15:00	1174	-73.985565	
10	2016-01-03	2016-01-03 02:30:00	02:30:00	1034	-73.985710	
11	2016-01-03	2016-01-03 02:45:00	02:45:00	973	-73.986038	
12	2016-01-03	2016-01-03 03:00:00	03:00:00	859	-73.985153	
13	2016-01-03	2016-01-03 03:15:00	03:15:00	852	-73.984062	
14	2016-01-03	2016-01-03 03:30:00	03:30:00	749	-73.986645	
15	2016-01-03	2016-01-03 03:45:00	03:45:00	700	-73.985607	
16	2016-01-03	2016-01-03 04:00:00	04:00:00	744	-73.985783	
17	2016-01-03	2016-01-03 04:15:00	04:15:00	814	-73.985752	
18	2016-01-03	2016-01-03 04:30:00	04:30:00	735	-73.983948	
19	2016-01-03	2016-01-03 04:45:00	04:45:00	765	-73.982193	
20	2016-01-03	2016-01-03 05:00:00	05:00:00	857	-73.981422	
21	2016-01-03	2016-01-03 05:15:00	05:15:00	984	-73.980675	
22	2016-01-03	2016-01-03 05:30:00	05:30:00	1262	-73.982338	
23	2016-01-03	2016-01-03 05:45:00	05:45:00	1910	-73.979080	
24	2016-01-03	2016-01-03 06:00:00	06:00:00	2040	-73.982025	

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pi
25	2016-01-03	2016-01-03 06:15:00	06:15:00	1689	-73.980011	
26	2016-01-03	2016-01-03 06:30:00	06:30:00	286	-73.959751	
27	2016-01-03	2016-01-03 06:45:00	06:45:00	97	-73.787888	
28	2016-01-03	2016-01-03 07:00:00	07:00:00	35	-73.785339	
29	2016-01-03	2016-01-03 07:15:00	07:15:00	1	-73.781860	
30	2016-01-03	2016-01-03 07:30:00	07:30:00	1	-73.789597	
31	2016-01-03	2016-01-03 08:00:00	08:00:00	1	-73.955399	
32	2016-01-03	2016-01-03 08:45:00	08:45:00	1	-73.776680	
33	2016-01-03	2016-01-03 20:15:00	20:15:00	5	-73.780495	
34	2016-01-03	2016-01-03 23:45:00	23:45:00	8	-73.987915	

In [ ]:
```
1
```

Bar plot(x axis -- time slot, y axis - total passenger count)

In [59]:
```
plt.figure(figsize=(12, 6))

# Create the bar plot
sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_01_03,
    color='red'
)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by 15 min Interval', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()
```

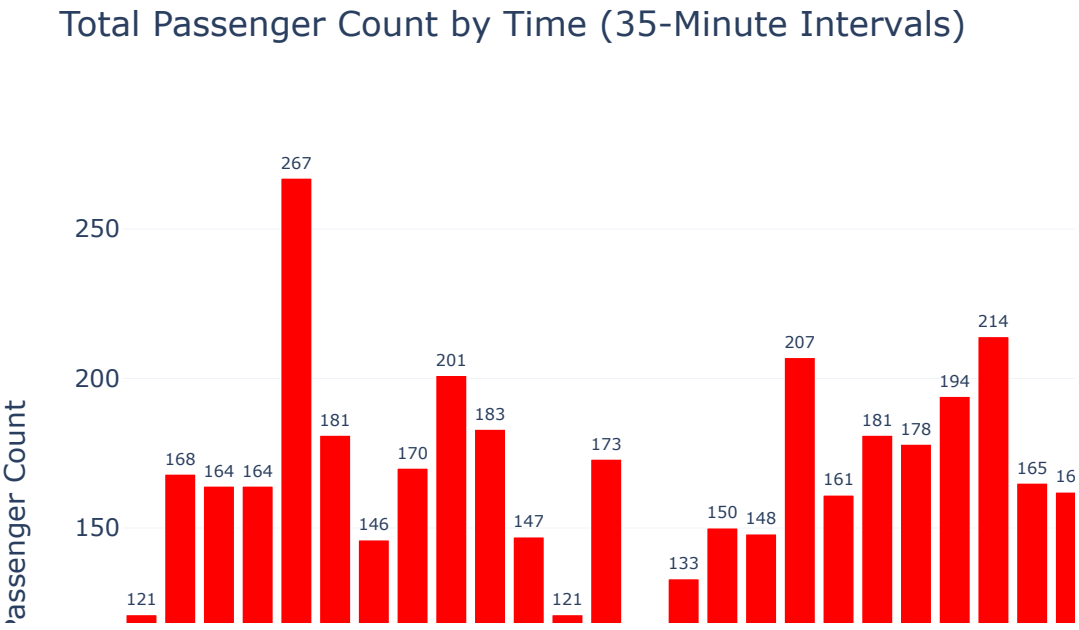


In [ ]:
```
1
```

In [24]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from ipywidgets import interact

# Sample data preparation
pickup_summary_data_2016_01_03
time_intervals = pd.date_range(start='00:00', end='23:55', freq='5T')
pickup_summary_data_2016_01_03 = pd.DataFrame({
    'time': time_intervals,
    'total_passenger_count': np.random.randint(0, 50, len(time_interva
})

# Function to aggregate data based on time interval
def plot_dynamic_barplot(interval_minutes):

    df = pickup_summary_data_2016_01_03.copy()
    df['time'] = pd.to_datetime(df['time'])
    df.set_index('time', inplace=True)
    df_resampled = df['total_passenger_count'].resample(f'{interval_mi

    # Create bar plot
    fig = go.Figure()

    # Add bar trace
    fig.add_trace(go.Bar(
        x=df_resampled['time'].dt.strftime('%H:%M'),
        y=df_resampled['total_passenger_count'],
        text=df_resampled['total_passenger_count'],
        textposition='outside',  # Show text above bars
        marker=dict(color='red')
    ))

    # Update layout
    fig.update_layout(
        title=f'Total Passenger Count by Time ({interval_minutes}-Minu
        xaxis_title='Time',
        yaxis_title='Total Passenger Count',
        xaxis=dict(tickangle=45),  # Rotate x-axis labels
        template='plotly_white',
        height=600
    )

    # Show plot
    fig.show()

# Interactive slider
interact(plot_dynamic_barplot, interval_minutes=(5, 60, 5))

```

interval_mi… 35



Out[24]: <function __main__.plot_dynamic_barplot(interval_minutes)>

In [19]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from ipywidgets import interact

# Sample data preparation (replace this with your dataset)
time_intervals = pd.date_range(start='00:00', end='23:55', freq='5T')
pickup_summary_data_2016_01_03 = pd.DataFrame({
    'time': time_intervals,
    'total_passenger_count': np.random.randint(0, 50, len(time_interva
})

# Function to create a dynamic bar plot
def plot_dynamic_barplot(interval_minutes):
    # Resample data
    df = pickup_summary_data_2016_01_03.copy()
    df['time'] = pd.to_datetime(df['time'])
    df.set_index('time', inplace=True)
    df_resampled = df['total_passenger_count'].resample(f'{interval_mi

    # Create the plot
    plt.figure(figsize=(12, 6))
    bars = plt.bar(
        x=df_resampled['time'].dt.strftime('%H:%M'),
        height=df_resampled['total_passenger_count'],
        color='red'
    )

    # Add labels above each bar
    for bar in bars:
        plt.text(
            bar.get_x() + bar.get_width() / 2,  # X coordinate
            bar.get_height() + 1,  # Y coordinate (slightly above the
            f'{int(bar.get_height())}',  # Text (count value)
            ha='center', va='bottom', fontsize=10
        )

    # Rotate x-axis labels for better readability
    plt.xticks(rotation=80)

    # Add labels and title
    plt.xlabel('Time', fontsize=12)
    plt.ylabel('Total Passenger Count', fontsize=12)
    plt.title(f'Total Passenger Count by Time ({interval_minutes}-Minu

    # Adjust layout and display
    plt.tight_layout()
    plt.show()

# Create an interactive slider
interact(plot_dynamic_barplot, interval_minutes=(5, 60, 5))

```

```
interactive(children=(IntSlider(value=30, description='interval_minutes',
max=60, min=5, step=5), Output()), _…
```

Out[19]: <function __main__.plot_dynamic_barplot(interval_minutes)>

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

data_2016_10_03(Thursday)

In [22]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_10_03['full_datetime'] = pd.to_datetime(data_2016_10_03['dat

# Convert 'time_frame' to datetime format and extract only the time
data_2016_10_03['time'] = pd.to_datetime(data_2016_10_03['time'])
data_2016_10_03['separate_time'] = data_2016_10_03['time'].dt.time

# Create 15-minute time frames
data_2016_10_03['time'] = data_2016_10_03['full_datetime'].dt.floor('1

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_10_03_15_min = (
    data_2016_10_03.groupby(['date', 'time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_10_03_15_min)
```

```
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_13676\425297506.py:2: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
```

```
---------------------------------------------------------------------------
--
TypeError                                 Traceback (most recent call las
t)
~\AppData\Local\Temp\ipykernel_13676\425297506.py in <module>
      3
      4 # Convert 'time_frame' to datetime format and extract only the ti
me
----> 5 data_2016_10_03['time'] = pd.to_datetime(data_2016_10_03['time'])
      6 data_2016_10_03['separate_time'] = data_2016_10_03['time'].dt.tim
e
      7

~\anaconda\lib\site-packages\pandas\core\tools\datetimes.py in to_datetim
e(arg, errors, dayfirst, yearfirst, utc, format, exact, unit, infer_datet
ime_format, origin, cache)
   1045                 result = arg.tz_localize(tz)
   1046     elif isinstance(arg, ABCSeries):
-> 1047         cache_array = _maybe_cache(arg, format, cache, convert_li
stlike)
   1048         if not cache_array.empty:
   1049             result = arg.map(cache_array)

~\anaconda\lib\site-packages\pandas\core\tools\datetimes.py in _maybe_cac
he(arg, format, cache, convert_listlike)
    195         unique_dates = unique(arg)
    196         if len(unique_dates) < len(arg):
--> 197             cache_dates = convert_listlike(unique_dates, format)
    198             cache_array = Series(cache_dates, index=unique_dates)
    199             # GH#39882 and GH#35888 in case of None and NaT we ge
t duplicates

~\anaconda\lib\site-packages\pandas\core\tools\datetimes.py in _convert_l
istlike_datetimes(arg, format, name, tz, unit, errors, infer_datetime_for
mat, dayfirst, yearfirst, exact)
    400     assert format is None or infer_datetime_format
    401     utc = tz == "utc"
--> 402     result, tz_parsed = objects_to_datetime64ns(
    403         arg,
    404         dayfirst=dayfirst,

~\anaconda\lib\site-packages\pandas\core\arrays\datetimes.py in objects_t
o_datetime64ns(data, dayfirst, yearfirst, utc, errors, require_iso8601, a
llow_object, allow_mixed)
   2222     order: Literal["F", "C"] = "F" if flags.f_contiguous else "C"
   2223     try:
-> 2224         result, tz_parsed = tslib.array_to_datetime(
   2225             data.ravel("K"),
   2226             errors=errors,

~\anaconda\lib\site-packages\pandas\_libs\tslib.pyx in pandas._libs.tsli
b.array_to_datetime()

~\anaconda\lib\site-packages\pandas\_libs\tslib.pyx in pandas._libs.tsli
b.array_to_datetime()

~\anaconda\lib\site-packages\pandas\_libs\tslib.pyx in pandas._libs.tsli
b._array_to_datetime_object()

~\anaconda\lib\site-packages\pandas\_libs\tslib.pyx in pandas._libs.tsli
b.array_to_datetime()
```

```
TypeError: <class 'datetime.time'> is not convertible to datetime
```

In [25]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_10_03['full_datetime'] = pd.to_datetime(data_2016_10_03['dat

# Create 15-minute time frames
data_2016_10_03['time_frame'] = data_2016_10_03['full_datetime'].dt.fl

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_10_03_15_min = (
    data_2016_10_03.groupby(['date', 'time_frame'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_10_03_15_min)

```

```
8  2016-10-03 2016-10-03 08:15:00                5827
9  2016-10-03 2016-10-03 08:30:00                5611
10 2016-10-03 2016-10-03 08:45:00                6001
11 2016-10-03 2016-10-03 09:00:00                6079
12 2016-10-03 2016-10-03 09:15:00                5602
13 2016-10-03 2016-10-03 09:30:00                5217
14 2016-10-03 2016-10-03 09:45:00                5651
15 2016-10-03 2016-10-03 10:00:00                5207
16 2016-10-03 2016-10-03 10:15:00                4962
17 2016-10-03 2016-10-03 10:30:00                4756
18 2016-10-03 2016-10-03 10:45:00                5044
19 2016-10-03 2016-10-03 11:00:00                4629
20 2016-10-03 2016-10-03 11:15:00                4422
21 2016-10-03 2016-10-03 11:30:00                4670
22 2016-10-03 2016-10-03 11:45:00                5139
23 2016-10-03 2016-10-03 12:00:00                5022
24 2016-10-03 2016-10-03 12:15:00                5268
25 2016-10-03 2016-10-03 12:30:00                5063
26 2016-10-03 2016-10-03 12:45:00                5721
```

In [ ]:

```
1
```

In [ ]:

```
1
```

In [28]:
```
print(data_2016_10_03.columns)
```

```
Index(['VendorID', 'tpep_pickup_datetime', 'tpep_dropoff_datetime',
       'passenger_count', 'trip_distance', 'pickup_longitude',
       'pickup_latitude', 'RatecodeID', 'store_and_fwd_flag',
       'dropoff_longitude', 'dropoff_latitude', 'payment_type', 'fare_amo
unt',
       'extra', 'mta_tax', 'tip_amount', 'tolls_amount',
       'improvement_surcharge', 'total_amount', 'date', 'time',
       'full_datetime'],
      dtype='object')
```

In [40]:
```
pickup_summary_data_2016_10_03_15_min
```

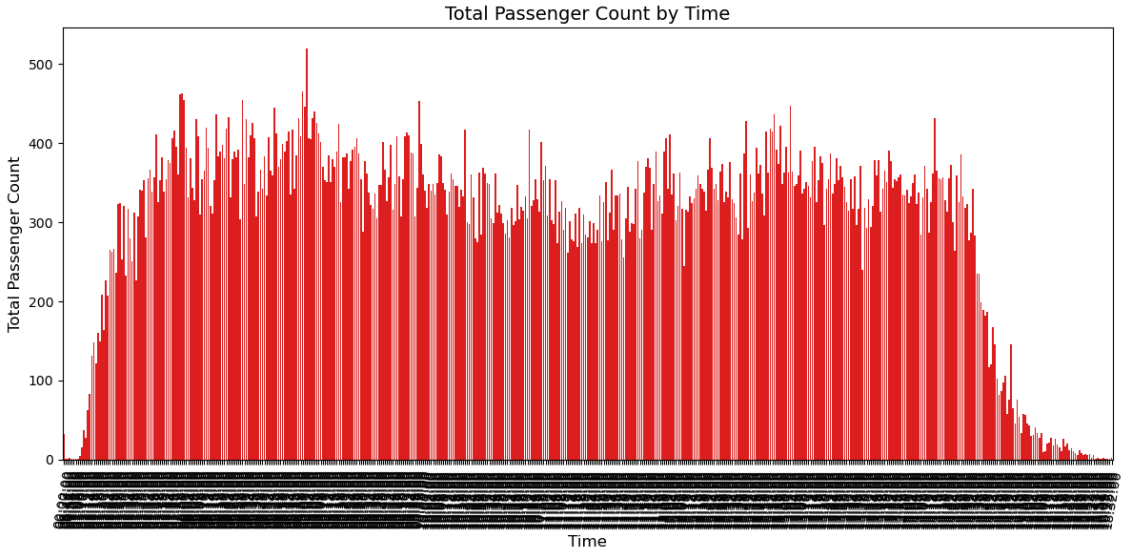
Out[40]:

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_p
0	2016-10-03	2016-10-03 00:00:00	00:00:00	32	-73.923458	
1	2016-10-03	2016-10-03 06:30:00	06:37:00	1	-74.005104	
2	2016-10-03	2016-10-03 06:45:00	06:46:00	1	-73.981483	
3	2016-10-03	2016-10-03 06:45:00	06:47:00	2	-73.908352	
4	2016-10-03	2016-10-03 06:45:00	06:54:00	1	-73.986641	
...	...	...	...	...	...	
518	2016-10-03	2016-10-03 16:00:00	16:03:00	2	-73.973396	
519	2016-10-03	2016-10-03 16:00:00	16:09:00	1	-73.784805	
520	2016-10-03	2016-10-03 16:00:00	16:12:00	1	-73.863464	
521	2016-10-03	2016-10-03 16:15:00	16:25:00	1	-73.981239	
522	2016-10-03	2016-10-03 18:30:00	18:32:00	2	-73.947281	

523 rows × 8 columns

In [58]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_10_03_15_min,
    color='red'
)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()
```



In [26]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_10_03['full_datetime'] = pd.to_datetime(data_2016_10_03['dat

# Create 1-hour time frames
data_2016_10_03['time_frame'] = data_2016_10_03['full_datetime'].dt.fl

# Separate the time component from the time_frame
data_2016_10_03['time'] = data_2016_10_03['time_frame'].dt.time

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_10_03_1_hr = (
    data_2016_10_03.groupby(['date', 'time_frame', 'time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_10_03_1_hr)

```

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pickup_latitude	total_trip_distance	total_amount
0	2016-10-03	2016-10-03 00:00:00	00:00:00	32	-73.923458	40.769064	141.38	605.43
1	2016-10-03	2016-10-03 06:00:00	06:00:00	5	-73.981483	40.764221	23.69	114.83
2	2016-10-03	2016-10-03 07:00:00	07:00:00	14084	-73.978401	40.757420	14637.19	85436.01
3	2016-10-03	2016-10-03 08:00:00	08:00:00	22988	-73.978806	40.756466	29342.26	169831.29
4	2016-10-03	2016-10-03 09:00:00	09:00:00	22549	-73.979507	40.755848	28997.59	173165.62
5	2016-10-03	2016-10-03 10:00:00	10:00:00	19969	-73.979637	40.756401	26234.00	155748.16
6	2016-10-03	2016-10-03 11:00:00	11:00:00	18860	-73.979607	40.756248	24992.05	148070.57
7	2016-10-03	2016-10-03 12:00:00	12:00:00	21074	-73.979111	40.756973	27104.42	164852.30
8	2016-10-03	2016-10-03 13:00:00	13:00:00	20945	-73.979729	40.756310	29754.89	173261.37
9	2016-10-03	2016-10-03 14:00:00	14:00:00	14625	-73.978844	40.755991	26596.15	144052.02
10	2016-10-03	2016-10-03 15:00:00	15:00:00	637	-73.951378	40.741959	4329.96	17243.48
11	2016-10-03	2016-10-03 16:00:00	16:00:00	5	-73.918430	40.760752	116.79	468.41
12	2016-10-03	2016-10-03 18:00:00	18:00:00	2	-73.947281	40.783939	51.27	315.30

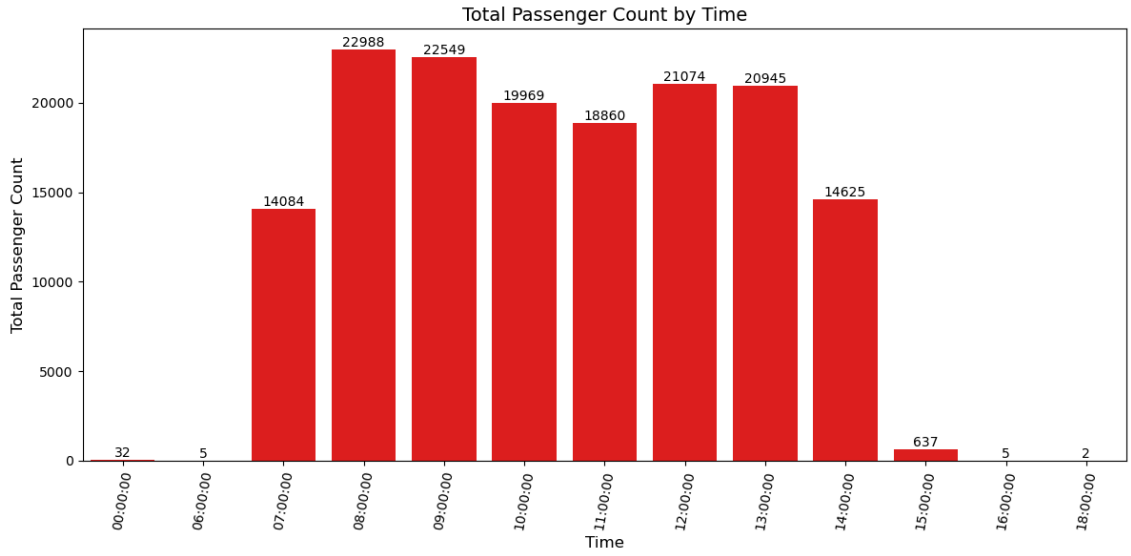
In [22]:
```
1 pickup_summary_data_2016_10_03_1_hr
```

Out[22]:

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pi
0	2016-10-03	2016-10-03 00:00:00	00:00:00	32	-73.923458	
1	2016-10-03	2016-10-03 06:00:00	06:00:00	5	-73.981483	
2	2016-10-03	2016-10-03 07:00:00	07:00:00	14084	-73.978401	
3	2016-10-03	2016-10-03 08:00:00	08:00:00	22988	-73.978806	
4	2016-10-03	2016-10-03 09:00:00	09:00:00	22549	-73.979507	
5	2016-10-03	2016-10-03 10:00:00	10:00:00	19969	-73.979637	
6	2016-10-03	2016-10-03 11:00:00	11:00:00	18860	-73.979607	
7	2016-10-03	2016-10-03 12:00:00	12:00:00	21074	-73.979111	
8	2016-10-03	2016-10-03 13:00:00	13:00:00	20945	-73.979729	
9	2016-10-03	2016-10-03 14:00:00	14:00:00	14625	-73.978844	
10	2016-10-03	2016-10-03 15:00:00	15:00:00	637	-73.951378	
11	2016-10-03	2016-10-03 16:00:00	16:00:00	5	-73.918430	
12	2016-10-03	2016-10-03 18:00:00	18:00:00	2	-73.947281	

In [24]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
ax = sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_10_03_1_hr,
    color='red'
)

# Annotate the bar plot with total passenger counts
for bar in ax.patches:
    # Get the height of the bar
    bar_height = bar.get_height()
    if bar_height > 0:  # To avoid displaying labels for zero-height b
        ax.text(
            bar.get_x() + bar.get_width() / 2,  # X-coordinate of the
            bar_height,                         # Y-coordinate of the t
            f'{int(bar_height)}',               # Text (convert to inte
            ha='center',                        # Horizontal alignment
            va='bottom',                        # Vertical alignment
            fontsize=10,                        # Font size
            color='black'                       # Text color
        )

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()

```

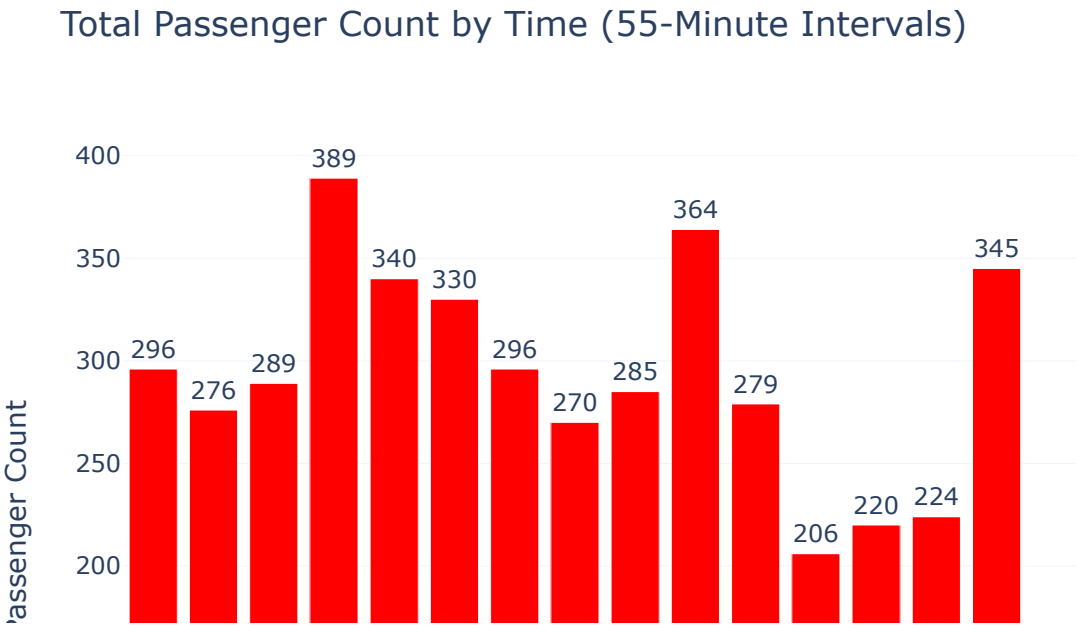


In [ ]:
```
1
```

In [27]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from ipywidgets import interact

# Sample data preparation (replace this with your dataset)
pickup_summary_data_2016_10_03_15_min
time_intervals = pd.date_range(start='00:00', end='23:55', freq='5T')
pickup_summary_data_2016_10_03_15_min = pd.DataFrame({
    'time': time_intervals,
    'total_passenger_count': np.random.randint(0, 50, len(time_interva
})

# Function to aggregate data based on time interval
def plot_dynamic_barplot(interval_minutes):
    # Resample data
    df = pickup_summary_data_2016_10_03_15_min.copy()
    df['time'] = pd.to_datetime(df['time'])
    df.set_index('time', inplace=True)
    df_resampled = df['total_passenger_count'].resample(f'{interval_mi

    # Create bar plot
    fig = go.Figure()

    # Add bar trace
    fig.add_trace(go.Bar(
        x=df_resampled['time'].dt.strftime('%H:%M'),
        y=df_resampled['total_passenger_count'],
        text=df_resampled['total_passenger_count'],
        textposition='outside',  # Show text above bars
        marker=dict(color='red')
    ))

    # Update layout
    fig.update_layout(
        title=f'Total Passenger Count by Time ({interval_minutes}-Minu
        xaxis_title='Time',
        yaxis_title='Total Passenger Count',
        xaxis=dict(tickangle=45),  # Rotate x-axis labels
        template='plotly_white',
        height=600
    )

    # Show plot
    fig.show()

# Interactive slider
interact(plot_dynamic_barplot, interval_minutes=(5, 60, 5))

```

interval_mi… 55



Out[27]: <function __main__.plot_dynamic_barplot(interval_minutes)>

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

show slot numbers

In [ ]:
```
1
```

data_2016_11_03(Friday)

In [28]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_11_03['full_datetime'] = pd.to_datetime(data_2016_11_03['dat

# Convert 'time_frame' to datetime format and extract only the time
data_2016_11_03['time_frame'] = pd.to_datetime(data_2016_11_03['time_f
data_2016_11_03['separate_time'] = data_2016_11_03['time_frame'].dt.ti

# Create 15-minute time frames
data_2016_11_03['time_frame'] = data_2016_11_03['full_datetime'].dt.fl

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_11_03_15_min = (
    data_2016_11_03.groupby(['date', 'time_frame','time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_11_03_15_min)
```

```
          date          time_frame      time  total_passenger_count  \
0   2016-01-03 2016-01-03 00:15:00  00:16:00                      1
1   2016-01-03 2016-01-03 00:15:00  00:26:00                      1
2   2016-01-03 2016-01-03 00:15:00  00:27:00                      1
3   2016-01-03 2016-01-03 00:30:00  00:30:00                      1
4   2016-01-03 2016-01-03 00:30:00  00:34:00                      1
..         ...                 ...       ...                    ...
139 2016-10-03 2016-10-03 14:15:00  14:27:00                      1
140 2016-10-03 2016-10-03 14:15:00  14:28:00                      2
141 2016-10-03 2016-10-03 14:30:00  14:31:00                      1
142 2016-10-03 2016-10-03 14:30:00  14:42:00                      5
143 2016-10-03 2016-10-03 14:45:00  14:54:00                      1

     median_pickup_longitude  median_pickup_latitude  total_trip_distance
\
0                -73.870789               40.773788                 9.20
1                -73.985565               40.727383                 5.20
2                -73.992165               40.721573                 4.20
3                -73.942139               40.754311                 0.40
4                -73.955696               40.779705                 3.70
..                      ...                     ...                  ...
139              -73.983917               40.746998                 0.54
140              -74.105141               40.681492                 0.85
141              -74.013863               40.709518                 4.85
142              -73.973373               40.753534                17.29
143              -73.983032               40.677299                 7.40

     total_amount
0           34.34
1           21.60
2           18.35
3            5.80
4           20.37
..            ...
139          9.36
140          6.80
141         27.88
142         92.04
143         34.56

[144 rows x 8 columns]
```

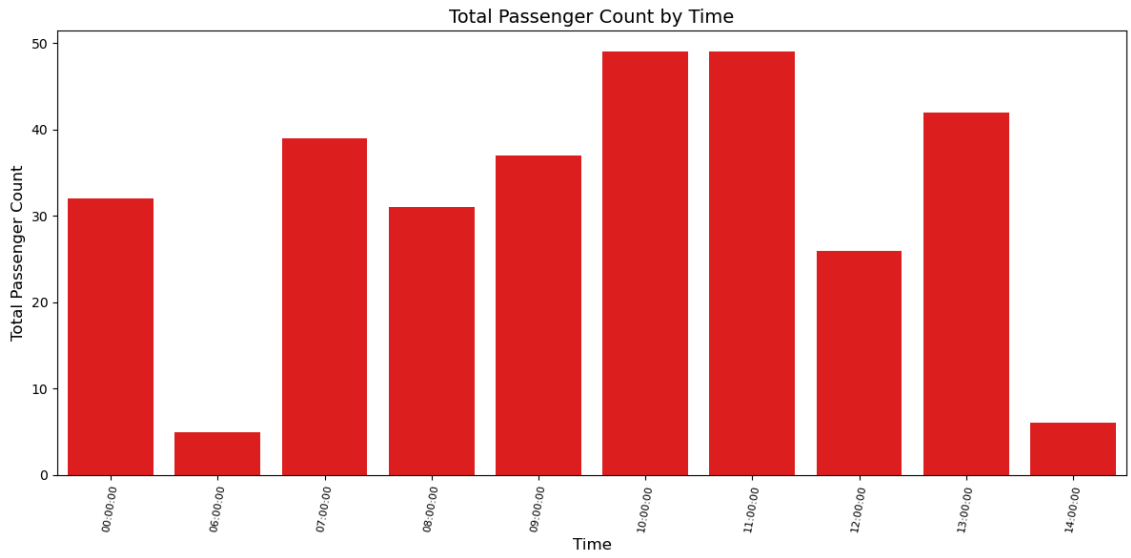
In [29]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_11_03['full_datetime'] = pd.to_datetime(data_2016_11_03['dat

# Create 1-hour time frames
data_2016_11_03['time_frame'] = data_2016_11_03['full_datetime'].dt.fl

# Separate the time component from the time_frame
data_2016_11_03['time'] = data_2016_11_03['time_frame'].dt.time

# Group by date and time_frame, calculate summary statistics
pickup_summary_data_2016_11_03_1_hr = (
    data_2016_11_03.groupby(['date', 'time_frame', 'time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_11_03_1_hr)

```

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pickup_latitude	total_trip_distance	total_amount
0	2016-01-03	2016-01-03 00:00:00	00:00:00	17	-73.976643	40.740847	65.93	275.04
1	2016-01-03	2016-01-03 01:00:00	01:00:00	21	-73.987000	40.740646	38.38	182.84
2	2016-01-03	2016-01-03 02:00:00	02:00:00	16	-73.989731	40.729832	31.97	137.41
3	2016-01-03	2016-01-03 03:00:00	03:00:00	6	-73.989433	40.759693	13.97	71.50
4	2016-01-03	2016-01-03 04:00:00	04:00:00	8	-73.991062	40.729145	42.90	173.75
5	2016-01-03	2016-01-03 05:00:00	05:00:00	12	-73.930511	40.746601	30.36	113.90
6	2016-01-03	2016-01-03 06:00:00	06:00:00	5	-73.978111	40.745644	6.12	39.76
7	2016-10-03	2016-10-03 07:00:00	07:00:00	17	-73.980408	40.751814	18.47	127.59
8	2016-10-03	2016-10-03 08:00:00	08:00:00	39	-73.978844	40.752266	45.93	326.55
9	2016-10-03	2016-10-03 09:00:00	09:00:00	54	-73.982079	40.756630	87.98	447.91
10	2016-10-03	2016-10-03 10:00:00	10:00:00	21	-73.977806	40.752407	42.44	242.33
11	2016-10-03	2016-10-03 11:00:00	11:00:00	24	-73.983147	40.759567	36.87	222.52
12	2016-10-03	2016-10-03 12:00:00	12:00:00	16	-73.979149	40.756119	11.68	89.79
13	2016-10-03	2016-10-03 14:00:00	14:00:00	26	-73.983917	40.743450	63.80	358.18

In [56]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_11_03_1_hr,
    color='red'
)

# Rotate x-axis labels for better readability
plt.xticks(rotation=80, fontsize=8)  # Adjust the font size of x-axis

# Add labels and title with correct font sizes
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()

```



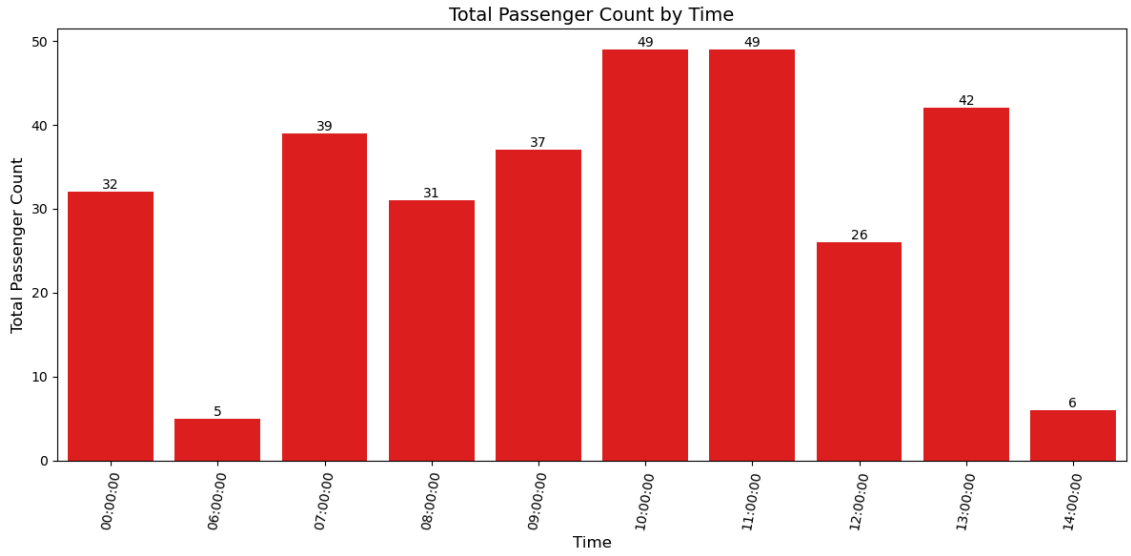
In [47]:
```
1 pickup_summary_data_2016_11_03_1_hr
```

Out[47]:

	date	time_frame	time	total_passenger_count	median_pickup_longitude	median_pic
0	2016-10-03	2016-10-03 00:00:00	00:00:00	32	-73.923458	
1	2016-10-03	2016-10-03 06:00:00	06:00:00	5	-73.981483	
2	2016-10-03	2016-10-03 07:00:00	07:00:00	39	-73.981064	
3	2016-10-03	2016-10-03 08:00:00	08:00:00	31	-73.981201	
4	2016-10-03	2016-10-03 09:00:00	09:00:00	37	-73.979565	
5	2016-10-03	2016-10-03 10:00:00	10:00:00	49	-73.975109	
6	2016-10-03	2016-10-03 11:00:00	11:00:00	49	-73.977470	
7	2016-10-03	2016-10-03 12:00:00	12:00:00	26	-73.983784	
8	2016-10-03	2016-10-03 13:00:00	13:00:00	42	-73.981369	
9	2016-10-03	2016-10-03 14:00:00	14:00:00	6	-73.980431	

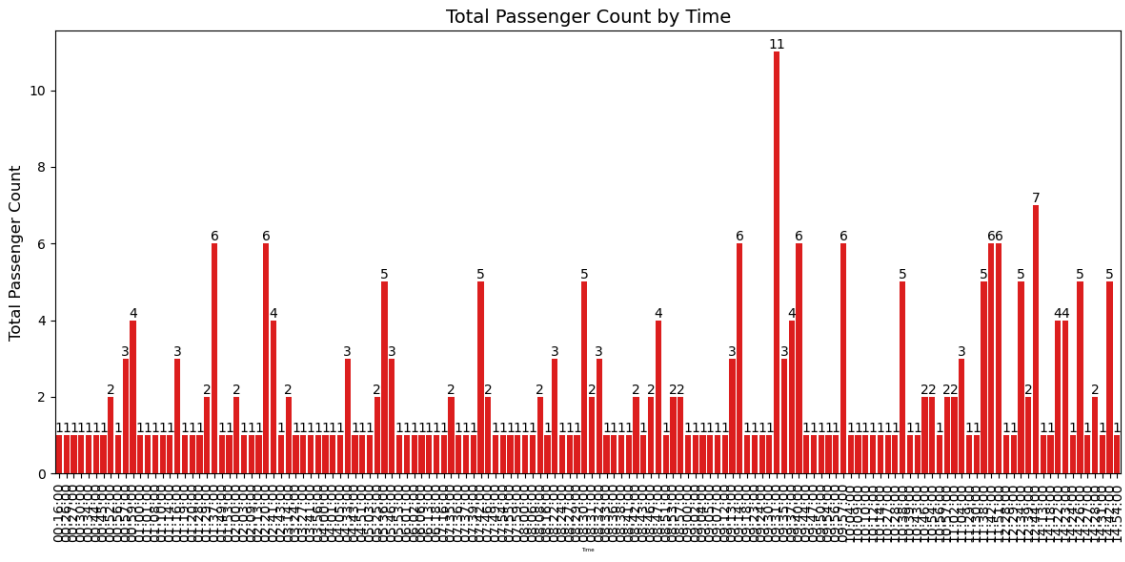
In [58]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
ax = sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_11_03,
    color='red'
)

# Annotate the bar plot with total passenger counts
for bar in ax.patches:
    # Get the height of the bar
    bar_height = bar.get_height()
    if bar_height > 0:  # To avoid displaying labels for zero-height b
        ax.text(
            bar.get_x() + bar.get_width() / 2,  # X-coordinate of the
            bar_height,                         # Y-coordinate of the t
            f'{int(bar_height)}',               # Text (convert to inte
            ha='center',                        # Horizontal alignment
            va='bottom',                        # Vertical alignment
            fontsize=10,                        # Font size
            color='black'                       # Text color
        )

# Rotate x-axis labels for better readability
plt.xticks(rotation=80)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()

```



In [31]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
ax = sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_11_03,
    color='red'
)

# Annotate the bar plot with total passenger counts
for bar in ax.patches:
    # Get the height of the bar
    bar_height = bar.get_height()
    if bar_height > 0:  # To avoid displaying labels for zero-height b
        ax.text(
            bar.get_x() + bar.get_width() / 2,  # X-coordinate of the
            bar_height,                         # Y-coordinate of the t
            f'{int(bar_height)}',               # Text (convert to inte
            ha='center',                        # Horizontal alignment
            va='bottom',                        # Vertical alignment
            fontsize=10,                        # Font size
            color='black'                       # Text color
        )

# Rotate x-axis labels for better readability
plt.xticks(rotation=90)

# Add labels and title
plt.xlabel('Time', fontsize=4)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()

```

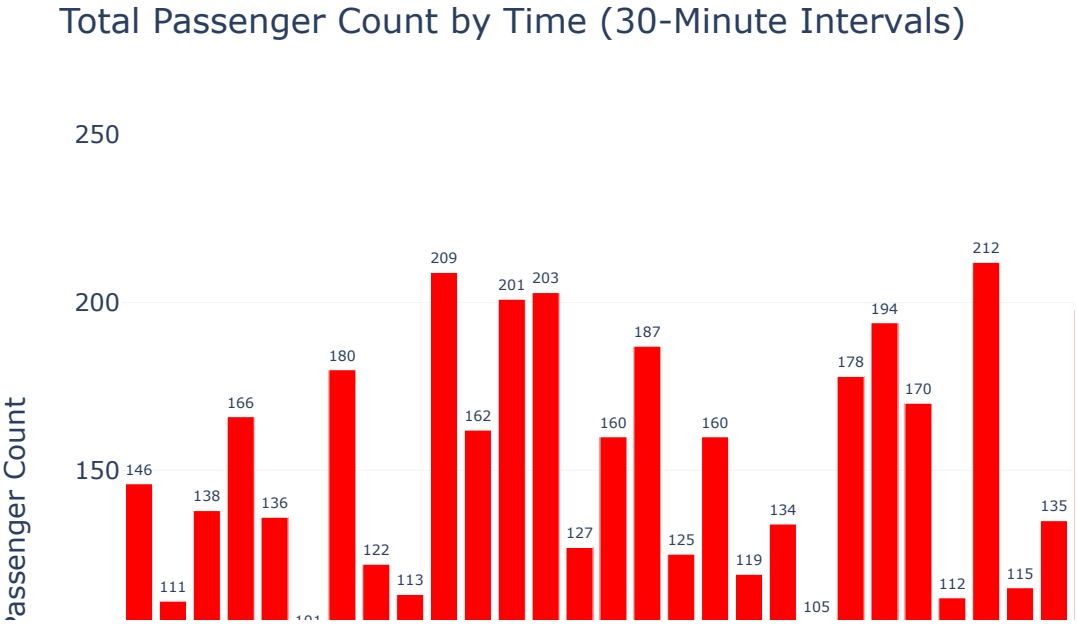


In [ ]:
```
1
```

In [30]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from ipywidgets import interact

# Sample data preparation (replace this with your dataset)
pickup_summary_data_2016_11_03_15_min
time_intervals = pd.date_range(start='00:00', end='23:55', freq='5T')
pickup_summary_data_2016_11_03_15_min = pd.DataFrame({
    'time': time_intervals,
    'total_passenger_count': np.random.randint(0, 50, len(time_interva
})

# Function to aggregate data based on time interval
def plot_dynamic_barplot(interval_minutes):
    # Resample data
    df = pickup_summary_data_2016_11_03_15_min.copy()
    df['time'] = pd.to_datetime(df['time'])
    df.set_index('time', inplace=True)
    df_resampled = df['total_passenger_count'].resample(f'{interval_mi

    # Create bar plot
    fig = go.Figure()

    # Add bar trace
    fig.add_trace(go.Bar(
        x=df_resampled['time'].dt.strftime('%H:%M'),
        y=df_resampled['total_passenger_count'],
        text=df_resampled['total_passenger_count'],
        textposition='outside',  # Show text above bars
        marker=dict(color='red')
    ))

    # Update layout
    fig.update_layout(
        title=f'Total Passenger Count by Time ({interval_minutes}-Minu
        xaxis_title='Time',
        yaxis_title='Total Passenger Count',
        xaxis=dict(tickangle=45),  # Rotate x-axis labels
        template='plotly_white',
        height=600
    )

    # Show plot
    fig.show()

# Interactive slider
interact(plot_dynamic_barplot, interval_minutes=(5, 60, 5))

```

interval_mi… 30



Out[30]: <function __main__.plot_dynamic_barplot(interval_minutes)>

In [ ]:
```
1
```

In [ ]:
```
1
```

Text size kamm karnah hai

How many taxi are required at particulat median locations(median_latt, median_long)

If there are multiple taxi request from a median location in a 15 minute slot, how to handle those requests??

How many taxi are required near the median locations . so that a cab can reach customer in the minimum amount of time?? If time to reach customer destination is large (>15 mins),customer may be anxious or may cancel trip ?? -- revenue loss

range of latitude and range of latitude (min , max)

data_2016_02_03(Wednesday)

In [31]:

```
# Combine 'date' and 'time' to create a full datetime column
data_2016_02_03['full_datetime'] = pd.to_datetime(data_2016_02_03['dat

# Create 15-minute time frames
data_2016_02_03['time_frame'] = data_2016_02_03['full_datetime'].dt.fl

# Extract separate 'time' and 'date' columns for clarity
data_2016_02_03['time'] = data_2016_02_03['time_frame'].dt.time
data_2016_02_03['date'] = data_2016_02_03['time_frame'].dt.date

# Group by 'date' and 'time_frame', calculate summary statistics
pickup_summary_data_2016_02_03 = (
    data_2016_02_03.groupby(['date', 'time'])
    .agg(
        total_passenger_count=('passenger_count', 'sum'),
        median_pickup_longitude=('pickup_longitude', 'median'),  # Med
        median_pickup_latitude=('pickup_latitude', 'median'),    # Med
        total_trip_distance=('trip_distance', 'sum'),           # Sum
        total_amount=('total_amount', 'sum')                    # Sum
    )
    .reset_index()
)

# Print the summary
print(pickup_summary_data_2016_02_03)
```

```
         date      time  total_passenger_count  median_pickup_longitude
\
0  2016-02-03  00:00:00                     24               -73.975002
1  2016-02-03  00:15:00                      1               -73.994186
2  2016-02-03  00:30:00                      1               -74.008232
3  2016-02-03  00:45:00                      2               -73.993198
4  2016-02-03  01:00:00                      1               -73.962166
5  2016-02-03  01:45:00                      1               -74.002480
6  2016-02-03  02:00:00                      1               -74.002029
7  2016-02-03  02:30:00                      1               -73.987839
8  2016-02-03  03:45:00                      1               -73.929840
9  2016-02-03  04:00:00                      2               -74.005028
10 2016-02-03  04:45:00                      3               -73.980015
11 2016-02-03  05:00:00                      2               -73.976974
12 2016-02-03  05:45:00                      2               -73.998528
13 2016-02-03  06:00:00                      1               -73.990753

    median_pickup_latitude  total_trip_distance  total_amount
0                40.745703                55.53        222.55
1                40.738789                 2.30         10.80
2                40.741253                 0.65          5.30
3                40.738693                 2.34         16.56
4                40.760437                 2.14         11.88
5                40.733379                 2.36         10.80
6                40.715256                 2.85         14.80
7                40.749619                 1.19          7.14
8                40.756527                 7.01         27.88
9                40.737541                10.73         46.61
10               40.738716                11.38         46.34
11               40.742111                 2.50         13.30
12               40.747707                 4.34         20.60
13               40.758060                 2.04         10.80
```

```
C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\1435495900.py:2: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)

C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\1435495900.py:5: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)

C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\1435495900.py:8: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)

C:\Users\Sanjoy\AppData\Local\Temp\ipykernel_6016\1435495900.py:9: Settin
gWithCopyWarning:


A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-do
cs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (http
s://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returni
ng-a-view-versus-a-copy)
```

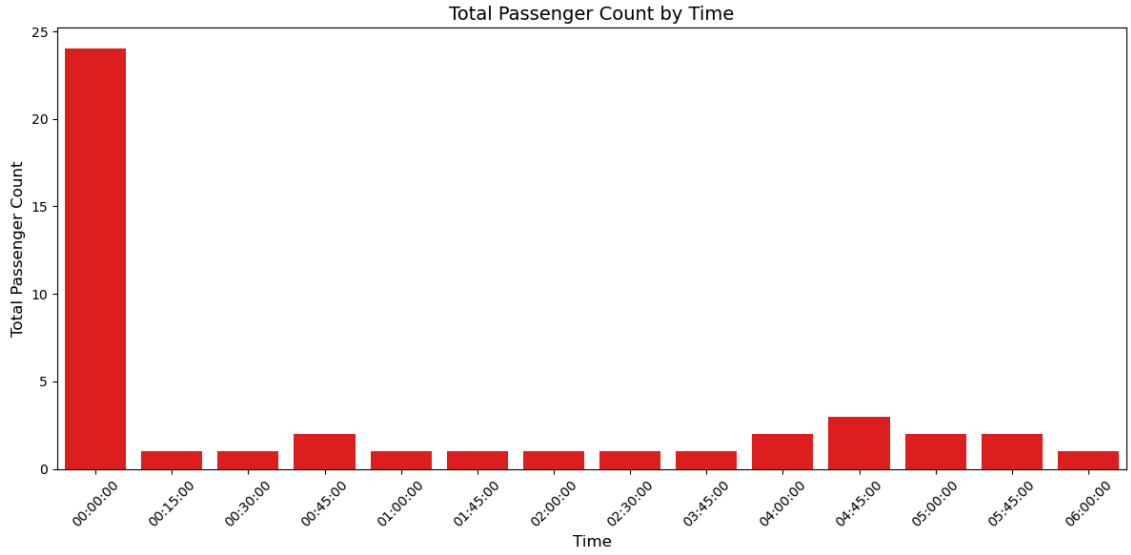
In [53]:
```
1 pickup_summary_data_2016_02_03
```

Out[53]:

	date	time	total_passenger_count	median_pickup_longitude	median_pickup_latitude
0	2016-02-03	00:00:00	24	-73.975002	40.745703
1	2016-02-03	00:15:00	1	-73.994186	40.738789
2	2016-02-03	00:30:00	1	-74.008232	40.741253
3	2016-02-03	00:45:00	2	-73.993198	40.738693
4	2016-02-03	01:00:00	1	-73.962166	40.760437
5	2016-02-03	01:45:00	1	-74.002480	40.733379
6	2016-02-03	02:00:00	1	-74.002029	40.715256
7	2016-02-03	02:30:00	1	-73.987839	40.749619
8	2016-02-03	03:45:00	1	-73.929840	40.756527
9	2016-02-03	04:00:00	2	-74.005028	40.737541
10	2016-02-03	04:45:00	3	-73.980015	40.738716
11	2016-02-03	05:00:00	2	-73.976974	40.742111
12	2016-02-03	05:45:00	2	-73.998528	40.747707
13	2016-02-03	06:00:00	1	-73.990753	40.758060

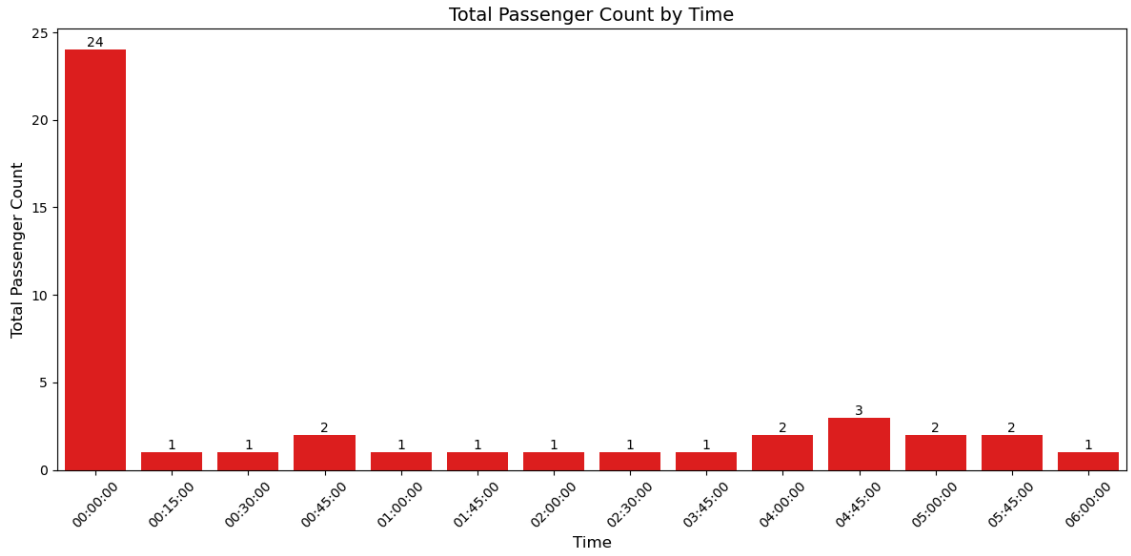
In [56]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_02_03,
    color='red'
)

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()
```



In [52]:

```
plt.figure(figsize=(12, 6))

# Create the bar plot
ax = sns.barplot(
    x='time',
    y='total_passenger_count',
    data=pickup_summary_data_2016_02_03,
    color='red'
)

# Annotate the bar plot with total passenger counts
for bar in ax.patches:
    # Get the height of the bar
    bar_height = bar.get_height()
    if bar_height > 0:  # To avoid displaying labels for zero-height b
        ax.text(
            bar.get_x() + bar.get_width() / 2,  # X-coordinate of the
            bar_height,                         # Y-coordinate of the t
            f'{int(bar_height)}',               # Text (convert to inte
            ha='center',                        # Horizontal alignment
            va='bottom',                        # Vertical alignment
            fontsize=10,                        # Font size
            color='black'                       # Text color
        )

# Rotate x-axis labels for better readability
plt.xticks(rotation=45)

# Add labels and title
plt.xlabel('Time', fontsize=12)
plt.ylabel('Total Passenger Count', fontsize=12)
plt.title('Total Passenger Count by Time', fontsize=14)

# Display the plot
plt.tight_layout()  # Adjust layout for better fit
plt.show()

```

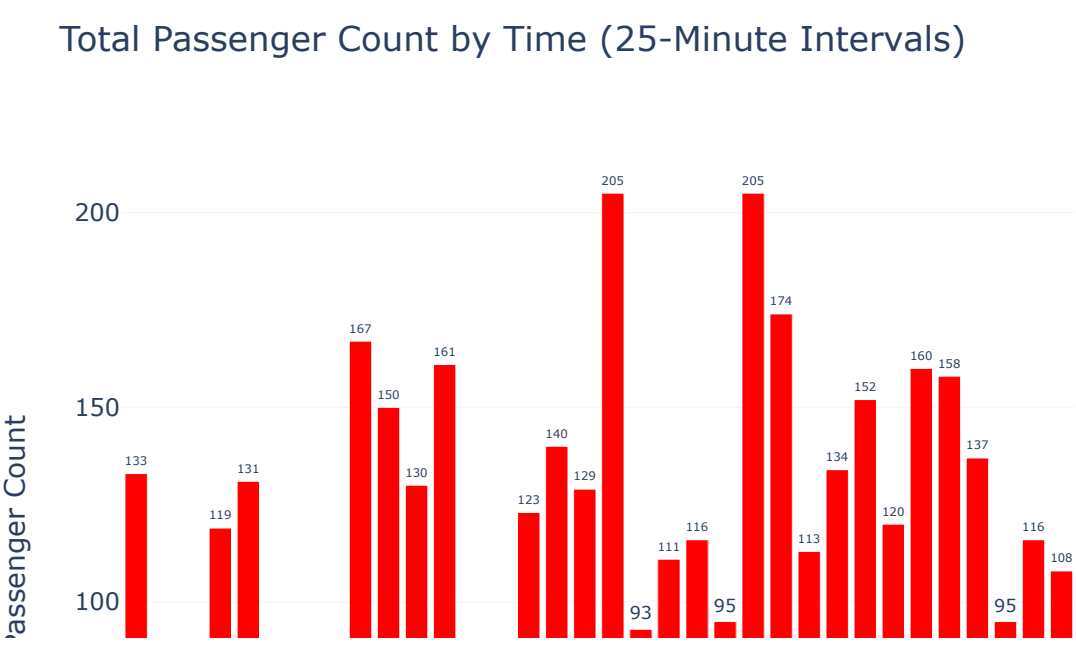


In [ ]:
```
1
```

In [32]:

```
import pandas as pd
import numpy as np
import plotly.graph_objects as go
from ipywidgets import interact


pickup_summary_data_2016_02_03
time_intervals = pd.date_range(start='00:00', end='23:55', freq='5T')
pickup_summary_data_2016_02_03 = pd.DataFrame({
    'time': time_intervals,
    'total_passenger_count': np.random.randint(0, 50, len(time_interva
})

# Function to aggregate data based on time interval
def plot_dynamic_barplot(interval_minutes):
    # Resample data
    df = pickup_summary_data_2016_02_03.copy()
    df['time'] = pd.to_datetime(df['time'])
    df.set_index('time', inplace=True)
    df_resampled = df['total_passenger_count'].resample(f'{interval_mi

    # Create bar plot
    fig = go.Figure()

    # Add bar trace
    fig.add_trace(go.Bar(
        x=df_resampled['time'].dt.strftime('%H:%M'),
        y=df_resampled['total_passenger_count'],
        text=df_resampled['total_passenger_count'],
        textposition='outside',  # Show text above bars
        marker=dict(color='red')
    ))

    # Update layout
    fig.update_layout(
        title=f'Total Passenger Count by Time ({interval_minutes}-Minu
        xaxis_title='Time',
        yaxis_title='Total Passenger Count',
        xaxis=dict(tickangle=45),  # Rotate x-axis labels
        template='plotly_white',
        height=600
    )

    # Show plot
    fig.show()

# Interactive slider
interact(plot_dynamic_barplot, interval_minutes=(5, 60, 5))

```

interval_mi… 25



Out[32]: <function __main__.plot_dynamic_barplot(interval_minutes)>

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [58]:
```
1 print(taxi_data.corr())
```

...

In [59]:
```
1 taxi_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100000 entries, 0 to 99999
Data columns (total 19 columns):
 #   Column                 Non-Null Count   Dtype  
---  ------                 --------------   -----  
 0   VendorID               100000 non-null  int64  
 1   tpep_pickup_datetime   100000 non-null  object 
 2   tpep_dropoff_datetime  100000 non-null  object 
 3   passenger_count        100000 non-null  int64  
 4   trip_distance          100000 non-null  float64
 5   pickup_longitude       100000 non-null  float64
 6   pickup_latitude        100000 non-null  float64
 7   RatecodeID             100000 non-null  int64  
 8   store_and_fwd_flag     100000 non-null  object 
 9   dropoff_longitude      100000 non-null  float64
 10  dropoff_latitude       100000 non-null  float64
 11  payment_type           100000 non-null  int64  
 12  fare_amount            100000 non-null  float64
 13  extra                  100000 non-null  float64
 14  mta_tax                100000 non-null  float64
 15  tip_amount             100000 non-null  float64
 16  tolls_amount           100000 non-null  float64
 17  improvement_surcharge  100000 non-null  float64
 18  total_amount           100000 non-null  float64
dtypes: float64(12), int64(4), object(3)
memory usage: 14.5+ MB
```

OLS Regression

In [13]:
```
1 num_cols = [3,4,5,6,7,9,10,13,14,15,16,17] ## apply index numbers of n
2
3 features = taxi_data[taxi_data.columns[num_cols]]
4 type(features)  # check type of object created - it sis also a datafra
5 features.head() # check
6 features = sm.add_constant(features)  # adding a_0
```

In [14]:
```
1 fare =  taxi_data[taxi_data.columns[12]]
2 fare
3 type(fare)  # when one column is extrcated fm a dataframe, a series is
4 taxi_data_linear_model = sm.OLS(fare, features) # Ordinary Least Squar
5 taxi_data_result = taxi_data_linear_model.fit()
```

In [15]:

```
# dir(taxi_data_result) # shows associated methods with this result ob
print(taxi_data_result.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            fare_amount   R-squared:                       0.848
Model:                            OLS   Adj. R-squared:                  0.848
Method:                 Least Squares   F-statistic:                 4.632e+04
Date:                Sun, 24 Nov 2024   Prob (F-statistic):               0.00
Time:                        23:06:56   Log-Likelihood:            -2.9369e+05
No. Observations:              100000   AIC:                         5.874e+05
Df Residuals:                  99987   BIC:                         5.875e+05
Df Model:                         12
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -5.0452      0.317    -15.909      0.000      -5.667      -4.424
passenger_count  0.0097      0.009      1.058      0.290      -0.008       0.028
trip_distance    2.3920      0.006    425.024      0.000       2.381       2.403
pickup_longitude -0.1280     0.065     -1.964      0.049      -0.256      -0.000
pickup_latitude -0.1681      0.119     -1.418      0.156      -0.400       0.064
RatecodeID       6.4740      0.072     89.578      0.000       6.332       6.616
dropoff_longitude 0.3835     0.079      4.881      0.000       0.230       0.537
dropoff_latitude 0.5963      0.143      4.176      0.000       0.316       0.876
extra           -3.5269      0.073    -48.177      0.000      -3.670      -3.383
mta_tax         -22.5820     0.585    -38.634      0.000     -23.728     -21.436
tip_amount       0.4122      0.007     59.116      0.000       0.399       0.426
tolls_amount    -0.2196      0.014    -16.166      0.000      -0.246      -0.193
improvement_surcharge 55.4808 1.263     43.945      0.000      53.006      57.955
==============================================================================
Omnibus:                   202843.068   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      19899758503.792
Skew:                          15.482   Prob(JB):                         0.00
Kurtosis:                    2188.174   Cond. No.                     1.11e+04
```

```
================================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.11e+04. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [16]:

```
dir(taxi_data_result)
print(taxi_data_result.summary2())
```

```
                     Results: Ordinary least squares
================================================================================
Model:                 OLS               Adj. R-squared:      0.848
Dependent Variable:    fare_amount       AIC:                 587397.7515
Date:                  2024-11-24 23:07  BIC:                 587521.4195
No. Observations:      100000            Log-Likelihood:      -2.9369e+05
Df Model:              12                F-statistic:         4.632e+04
Df Residuals:          99987             Prob (F-statistic):  0.00
R-squared:             0.848             Scale:               20.821
--------------------------------------------------------------------------------
                      Coef.   Std.Err.     t     P>|t|   [0.025   0.975]
--------------------------------------------------------------------------------
const                -5.0452   0.3171 -15.9086 0.0000  -5.6668  -4.4236
passenger_count       0.0097   0.0091   1.0579 0.2901  -0.0082   0.0275
trip_distance         2.3920   0.0056 425.0244 0.0000   2.3810   2.4031
pickup_longitude     -0.1280   0.0652  -1.9645 0.0495  -0.2557  -0.0003
pickup_latitude      -0.1681   0.1185  -1.4181 0.1562  -0.4004   0.0642
RatecodeID            6.4740   0.0723  89.5784 0.0000   6.3323   6.6156
dropoff_longitude     0.3835   0.0786   4.8814 0.0000   0.2295   0.5375
dropoff_latitude      0.5963   0.1428   4.1759 0.0000   0.3164   0.8761
extra                -3.5269   0.0732 -48.1768 0.0000  -3.6704  -3.3834
mta_tax             -22.5820   0.5845 -38.6337 0.0000 -23.7277 -21.4364
tip_amount            0.4122   0.0070  59.1156 0.0000   0.3986   0.4259
tolls_amount         -0.2196   0.0136 -16.1659 0.0000  -0.2462  -0.1930
improvement_surcharge 55.4808   1.2625  43.9450 0.0000  53.0063  57.9553
--------------------------------------------------------------------------------
Omnibus:              202843.068     Durbin-Watson:         1.970
Prob(Omnibus):        0.000          Jarque-Bera (JB):      19899758503.792
Skew:                 15.482         Prob(JB):              0.000
Kurtosis:             2188.174       Condition No.:         11076
================================================================================
* The condition number is large (1e+04). This might indicate
strong multicollinearity or other numerical problems.
```

In [17]:

```
print(f"The first thing to examine in the output is  Prob (F-Statistic
```

```
The first thing to examine in the output is  Prob (F-Statistic)
 We need to ask: Is it less than 0.05 ?
 Here it appears as : 0.00
 Hence the model's Adj R-Square 0.85 is statistically significant
```

In [18]:

```
type(taxi_data_linear_model.fit()) # this is a wrapper object


print(str(taxi_data_result.summary())) # to print regression model nea
# dir(taxi_data_result) This shows what methods are available without
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:            fare_amount   R-squared:                       0.848
Model:                            OLS   Adj. R-squared:                  0.848
Method:                 Least Squares   F-statistic:                 4.632e+04
Date:                Sun, 24 Nov 2024   Prob (F-statistic):               0.00
Time:                        23:07:28   Log-Likelihood:            -2.936e+05
No. Observations:              100000   AIC:                         5.874e+05
Df Residuals:                  99987   BIC:                         5.875e+05
Df Model:                         12
Covariance Type:            nonrobust
================================================================================
                   coef    std err          t      P>|t|      [0.025      0.975]
--------------------------------------------------------------------------------
const           -5.0452      0.317    -15.909      0.000      -5.667      -4.424
passenger_count  0.0097      0.009      1.058      0.290      -0.008       0.028
trip_distance    2.3920      0.006    425.024      0.000       2.381       2.403
pickup_longitude -0.1280     0.065     -1.964      0.049      -0.256      -0.000
pickup_latitude  -0.1681     0.119     -1.418      0.156      -0.400       0.064
RatecodeID        6.4740     0.072     89.578      0.000       6.332       6.616
dropoff_longitude 0.3835     0.079      4.881      0.000       0.230       0.537
dropoff_latitude  0.5963     0.143      4.176      0.000       0.316       0.876
extra            -3.5269     0.073    -48.177      0.000      -3.670      -3.383
mta_tax         -22.5820     0.585    -38.634      0.000     -23.728     -21.436
tip_amount        0.4122     0.007     59.116      0.000       0.399       0.426
tolls_amount     -0.2196     0.014    -16.166      0.000      -0.246      -0.193
improvement_surcharge 55.4808 1.263    43.945      0.000      53.006      57.955
==============================================================================
Omnibus:                   202843.068   Durbin-Watson:                   1.970
Prob(Omnibus):                  0.000   Jarque-Bera (JB):      19899758503.792
Skew:                          15.482   Prob(JB):                         0.00
Kurtosis:                    2188.174   Cond. No.                     1.11e+04
```

```
==========================================================================
=====

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is co
rrectly specified.
[2] The condition number is large, 1.11e+04. This might indicate that the
re are
strong multicollinearity or other numerical problems.
```

In [19]:

```
print(f'Model Fit is assessed \n using Adjusted R Squared {100*taxi_da
taxi_data_ols_coeff = taxi_data_result.params.round(2).copy()

taxi_data_ols_coeff
taxi_data_ols_coeff = taxi_data_ols_coeff[1:] # drop const or intercep
print('----------')
print('Coeff     Values')
print('-----     ------')
print(taxi_data_ols_coeff)
```

```
Model Fit is assessed
 using Adjusted R Squared 84.75 %
----------
Coeff     Values
-----     ------
passenger_count           0.01
trip_distance             2.39
pickup_longitude         -0.13
pickup_latitude          -0.17
RatecodeID                6.47
dropoff_longitude         0.38
dropoff_latitude          0.60
extra                    -3.53
mta_tax                 -22.58
tip_amount                0.41
tolls_amount             -0.22
improvement_surcharge    55.48
dtype: float64
```

In [20]:
```
1 taxi_data_result.conf_int(alpha = 0.05).round(2)
```

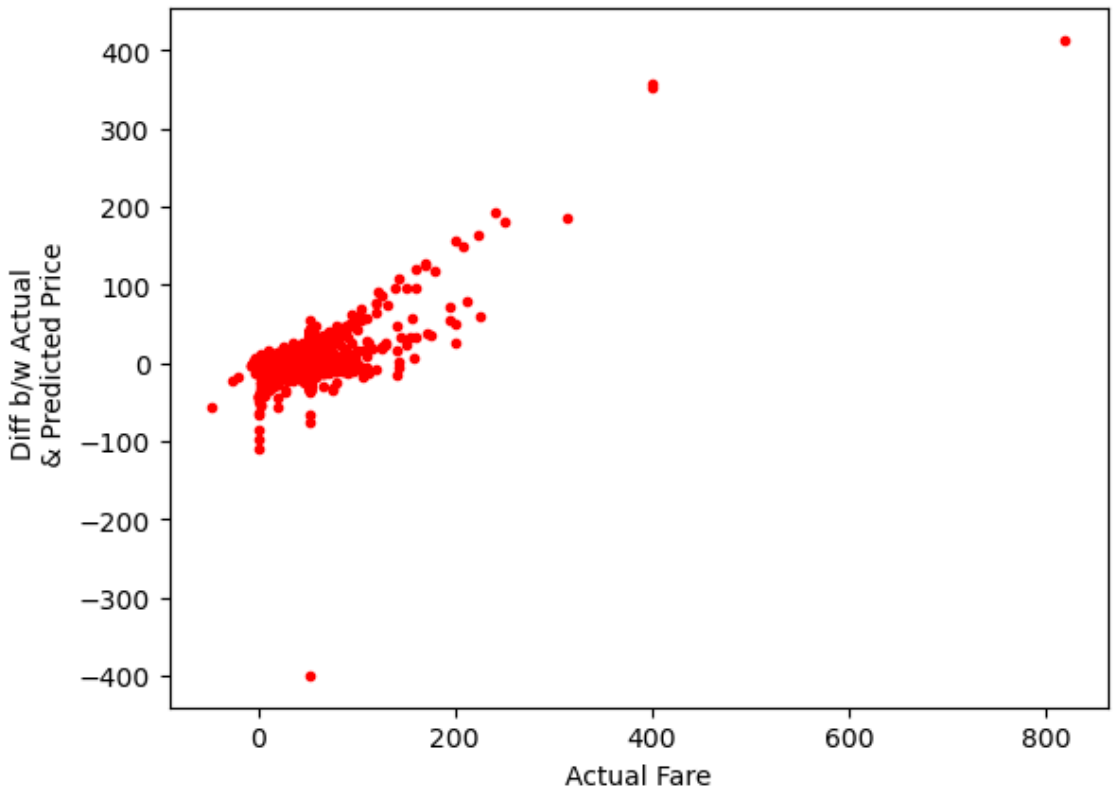
Out[20]:

	0	1
const	-5.67	-4.42
passenger_count	-0.01	0.03
trip_distance	2.38	2.40
pickup_longitude	-0.26	-0.00
pickup_latitude	-0.40	0.06
RatecodeID	6.33	6.62
dropoff_longitude	0.23	0.54
dropoff_latitude	0.32	0.88
extra	-3.67	-3.38
mta_tax	-23.73	-21.44
tip_amount	0.40	0.43
tolls_amount	-0.25	-0.19
improvement_surcharge	53.01	57.96

In [21]:
```
1 taxi_data_result.ess.sum() + taxi_data_result.resid.sum()
```

Out[21]:
```
11573279.620580472
```

In [22]:
```
1 plt.plot(taxi_data.fare_amount, taxi_data_result.resid,'.r')
2 plt.xlabel("Actual Fare")
3 plt.ylabel( "Diff b/w Actual \n & Predicted Fare")
4 plt.show()
```



In [23]:
```
1 taxi_data_result.resid
```

Out[23]:
```
0       -1.385854
1       -0.746923
2       -0.125933
3        0.023719
4       -0.885933
             ...
99995   -1.529312
99996   -0.306455
99997   -2.002751
99998   -0.844012
99999   -1.734828
Length: 100000, dtype: float64
```

Boxplot of fare_amount , total amount, trip_distance

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```

In [ ]:
```
1
```