In [1]:
```
import pandas as pd
```

In [2]:
```
#file_location = "DataSet (1).xlsx"

#dfs = pd.read_excel(file_location, sheet_name="Sheet1")
dfs = pd.read_excel(r'C:\Users\ayanthg\Documents\D\ds.xlsx')
```

In [3]:
```
dfs
```

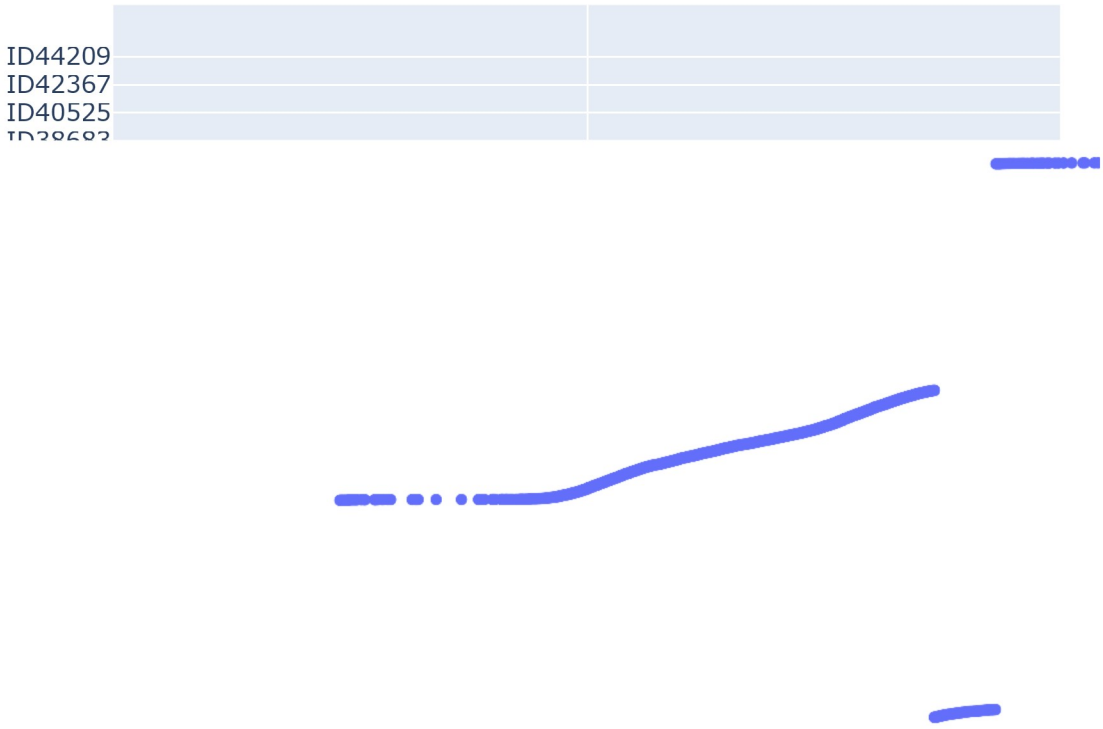
Out[3]:

	trip_id	customer_id	timestamp	pick_lat	pick_lng	drop_lat	drop_lng	travel_di
0	ID001	CUST_001	1546709270211	17.442705	78.387878	17.457829	78.399056	
1	ID002	CUST_002	1546709309524	17.490189	78.415512	17.450548	78.367294	
2	ID003	CUST_003	1546709331857	17.370108	78.515045	17.377041	78.517921	
3	ID004	CUST_004	1546709358403	17.439314	78.443001	17.397131	78.516586	
4	ID005	CUST_005	1546709386884	17.432325	78.381966	17.401625	78.400032	
...	...	...	...	...	...	...	...	
44582	ID44583	CUST_19137	1546531491044	17.443661	78.391968	17.451042	78.371658	
44583	ID44584	CUST_19138	1546531505869	17.439289	78.396118	17.449976	78.389160	
44584	ID44585	CUST_5061	1546531522103	17.363689	78.535194	17.374418	78.529823	
44585	ID44586	CUST_19139	1546531536429	17.401539	78.570076	17.416904	78.591362	
44586	ID44587	CUST_15562	1546531549938	17.385243	78.479896	17.394102	78.499550	

44587 rows × 10 columns

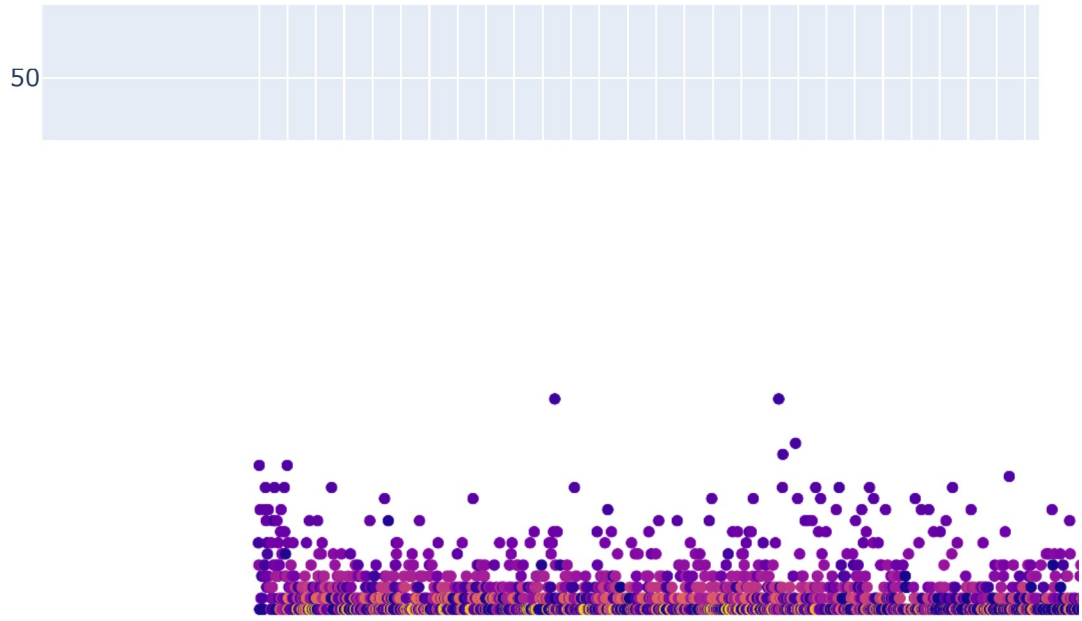
In [4]:
```
import plotly.express as px
import plotly.graph_objects as go
```

In [5]:
```
# timestamp vs trips...helps to analyse trips in a given timefrmae ....zoom-in to s
fig = px.scatter(dfs, x='timestamp', y='trip_id')
fig.show()
```



In [6]:
```
# returnig customers at timpestamps
fig = px.scatter(dfs, x='customer_id', y='timestamp')
fig.show()
```

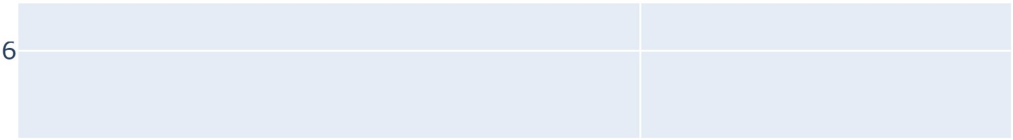
In [7]:
```
# frequency of returnig customers with color scale demonstrating avreage time betwe
import numpy as np
customers_grp = dfs.groupby(['customer_id']).groups
cus_grp = []
for k,v in customers_grp.items():
    if len(v)>=2:# atleast 2 trips
        trip_timestamps = [dfs["timestamp"][i] for i in v]
        avg_time_2_trips = ((np.max(trip_timestamps) - np.min(trip_timestamps))/len
    #     if avg_time_2_trips==0:
    #         avg_time_2_trips = 40000
        cus_grp.append([k,len(v),avg_time_2_trips])

z=pd.DataFrame(cus_grp)
z.columns=["customer", "trip_freq","avg_time_2_trips"]
fig = px.scatter(z, x="customer", y="trip_freq",color='avg_time_2_trips')
fig.show()
```



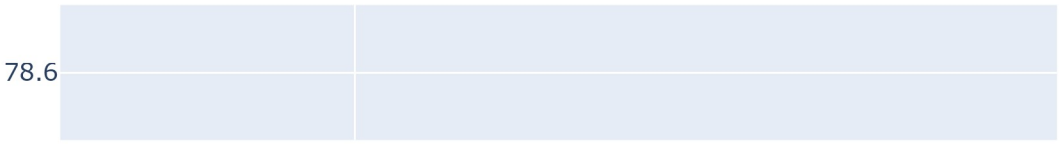
In [8]:
```
# Trips' geographic reagion of coverage

pickupLat_max = dfs['pick_lat'].max()
pickupLat_min = dfs['pick_lat'].min()
dropLat_max = dfs['drop_lat'].max()
dropLat_min = dfs['drop_lat'].min()
pickupLng_max = dfs['pick_lng'].max()
pickupLng_min = dfs['pick_lng'].min()
dropLng_max = dfs['drop_lng'].max()
dropLng_min = dfs['drop_lng'].min()
print ("Max pick-up reagio geographic spread latitude: "+ str(pickupLat_max)+", lon
print ("Min pick-up reagio geographic spread latitude: "+ str(pickupLat_min)+", lon
print ("Max dropping reagio geographic spread latitude: "+ str(dropLat_max)+", long
print ("Min dropping reagio geographic spread latitude: "+ str(dropLat_min)+", long

# range_pLat = (pickupLat_max - pickupLat_min)/10
# range_dLat = (dropLat_max - dropLat_min)/10
# range_pLng = (pickupLng_max - pickupLng_min)/10
# range_dLng = (dropLng_max - dropLng_min)/10
```

```
Max pick-up reagio geographic spread latitude: 17.5297909, longitude: 78.600647
Min pick-up reagio geographic spread latitude: 17.3303394, longitude: 78.3082581
Max dropping reagio geographic spread latitude: 17.7361546, longitude: 78.6348038
Min dropping reagio geographic spread latitude: 12.9216957, longitude: 77.5481033
```

In [9]:
```
# geographic region spread of pickup loactions
fig = px.scatter(dfs, x='pick_lat', y='pick_lng')
fig.show()
```



In [10]:
```
# geographic region spread of droping loactions
fig = px.scatter(dfs, x='drop_lat', y='drop_lng')
fig.show()
```



In [11]:
```
# 4-Dimensional representation of trips (used in deciding trip routes and also for
import plotly.express as px
fig = px.scatter_3d(dfs, x='pick_lat', y='pick_lng', z='timestamp',
                    color='travel_distance')
fig.show()
```

In [ ]:
```
### PART 2  Top 5 pairs of hex (resolution=8) clusters where most of the trips happ
# You can refer to the library listed below to get hexid for a given latitude and l
```

In [ ]:
```
#!pip install h3
```

In [12]:
```
import h3

given_grid_resolution = 8

def getHexID(location=[0, 0], resolution = given_grid_resolution):
    return h3.geo_to_h3(location[0], location[1], resolution)
```

In [13]:
```
### calculating hexa pairs between pickup and drop

number_of_Customer = len(dfs)
trip_dict = {}
for i in range(number_of_Customer):
    # pickup location
    pickup_loc = [dfs["pick_lat"][i], dfs["pick_lng"][i]]
    # droping location
    drop_loc = [dfs["drop_lat"][i], dfs["drop_lng"][i]]
    # pickup hexa ID
    pickup_HexaId = getHexID(location = pickup_loc)
    # droping hexa ID
    drop_HexaId = getHexID(location = drop_loc)
    # dictionary to store hexa pairs and keep counter for number of trips
#     print(pickup_HexaId, drop_HexaId)
    trip_key = pickup_HexaId +"||"+drop_HexaId
    try:
        trip_dict[trip_key] += 1
    except:
        trip_dict[trip_key] = 1
```

In [ ]:
```
# print(trip_dict)
```

In [14]:
```
### printing top 5 pairs

# sorting on number of trips
sorted_trip_list = sorted(trip_dict.items(), key=lambda item: item[1])
sorted_trip_list.reverse()

sorted_trip_output = [[i+1,k.replace("||"," to "), v ] for i,(k,v) in enumerate(sor

df_sorted_trip_output = pd.DataFrame(sorted_trip_output)
df_sorted_trip_output.columns =['Rank', 'Hex pair (source_hexid, destination_hexid)

df_sorted_trip_output.head(5)
```

Out[14]:

	Rank	Hex pair (source_hexid, destination_hexid)	Totaltrips
0	1	8860a259b9fffff to 8860a25995fffff	194
1	2	8860a25995fffff to 8860a259b9fffff	90
2	3	8860a259b9fffff to 8860a259bbfffff	82
3	4	8860a259b9fffff to 8860a24a6dfffff	80
4	5	8860a24a6dfffff to 8860a24b51fffff	79

In [15]:
```
### saving pairs into xlxs file
df_sorted_trip_output.to_excel("output_part_2.xlsx")
```

In [ ]:
```
# Part 3. Metric calculation
# What is the average duration between the 1st trip and the 2nd trip of customers?
# Note: Consider only the customers who have done 2 or more trips.
```

In [16]:
```
import numpy as np
# grouping data by customer id in dictionary using pandas
customers_grp = dfs.groupby(['customer_id']).groups

# selecting customer who have done 2 or more trips.
list_cust_time_taken = []

for k,v in customers_grp.items():
    # k is customerID
    # v is list of index of rows of  customerID in file
    if len(v)>=2:
        dic = {}
        # retriving all timestamp and travel_time for the customerID
        for i in v:
            dic[dfs["timestamp"][i]] = dfs["travel_time"][i]
        # taking first 2 trips by sorting
        sort_li = sorted(dic.items())
        # calculating duration between the 1st trip and the 2nd trip
        time_taken = sort_li[1][0] - sort_li[0][0] #- sort_li[0][1]
#         print(sort_li)
#         print(k, time_taken)
        list_cust_time_taken.append(time_taken)
# finding average
average = np.mean(list_cust_time_taken)


print("Average duration between the 1st trip and the 2nd trip of customers", averag
```

```
Average duration between the 1st trip and the 2nd trip of customers 89229691.580722
9
```

In [ ]:
```
# Part 4. Model building
# Build a model to predict trip_fare using travel_distance and travel_time.
# Measure the accuracy of the model and use the model to
# predict trip_fare for a trip with travel_distance of 3.5 kms and travel_time of 1
```

In [17]:
```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn import metrics
```

In [18]:
```
X = dfs[["travel_distance", "travel_time"]]
```

In [19]:
```
Y = dfs["trip_fare"]
```

In [20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_sta
```

In [21]:
```
regressor = LinearRegression()
regressor.fit(X_train, y_train)
```

Out[21]:
```
LinearRegression()
```

In [22]:
```
coeff_df = pd.DataFrame(regressor.coef_, X.columns, columns=['Coefficient'])
print(coeff_df)
```

```
                  Coefficient
travel_distance      8.668994
travel_time          0.057518
```

In [23]:
```
# running preiction on test data
y_pred = regressor.predict(X_test)
```

In [24]:
```
# chekcing results from model
df_res = pd.DataFrame({'Actual': y_test, 'Predicted': y_pred})
print(df_res.head(25))
```

```
       Actual   Predicted
38585      28   22.037448
9379       35   32.976831
43928      38   38.524921
28255      43   40.015022
23615      49   48.133081
28406      24   19.022837
41266      57   64.644247
19761     103  100.893141
28624      58   56.958910
23881      38   35.957075
36304      42   41.292215
4198       27   21.268335
44275      32   29.461554
33453      20    9.041120
16087      60   67.233138
2330       57   58.235315
15447      53   49.894947
35424      40   40.206023
36656      41   37.618022
14863      37   34.667475
25519      54   47.301952
27626      43   43.806892
22655      53   60.015781
19985      57   62.703115
19962      27   20.745130
```

In [25]:
```
# calculating predictions

print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, y_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, y_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test, y_pred
```

```
Mean Absolute Error: 5.3811275957755464
Mean Squared Error: 336.0989313861474
Root Mean Squared Error: 18.33300115600682
```

In [26]:
```
regressor.score(X,Y)
```

Out[26]:
```
0.7908637401272512
```

In [27]:
```
query = [[3.5, 15]]
ans_query = regressor.predict(query)
print("predicted trip_fare for a trip with travel_distance of 3.5 kms and travel_ti
```

```
predicted trip_fare for a trip with travel_distance of 3.5 kms and travel_time of 1
5 minutes: 40.033956197004706
```

```
C:\Users\ayanthg\Anaconda3\lib\site-packages\sklearn\base.py:450: UserWarning:

X does not have valid feature names, but LinearRegression was fitted with feature n
ames
```