Connecting MySQL Database to Jupyter Notebooks

Step 1. Download mysql-connector-python

You can do this through commands such as: pip install mysql-connector-python

Step 2. Connect to your database

Add your password and uncomment out the others

In [1]:
```
import pandas as pd
import mysql.connector

db = mysql.connector.connect(
    user="root",
    password="{Password}", #angy, comment/un-comment for yours
    #password = "ilo5517WD",
    host="localhost",
    port = 3306,
    database = 'rideshare'
)
```

Checking connection

In [2]:
```
print(db)
```

```
<mysql.connector.connection_cext.CMySQLConnection object at 0x0000022461C1ED60>
```

Setting cursor that will allow us to execute commands like in MySQL queries

In [3]:
```
cursor = db.cursor()
```

Step 3. Import Tables

Listing all tables in current database = rideshare

In [4]:
```
cursor.execute("SHOW TABLES")
for table in [tables[0] for tables in cursor.fetchall()]:
    print(table)
```

```
april_green
april_yellow
feb_green_one_rider
feb_green_two_rider
february_green
february_yellow
jan_green_one_rider
jan_green_two_rider
january_green
january_yellow
june_green
june_yellow
mar_green_one_rider
mar_green_two_rider
```

```
march_green
march_yellow
may_green
may_yellow
```

Selecting the january green taxi data that was split with passenger count = 1

In [5]:
```
jan_green_one_rider = pd.read_sql("select * from jan_green_one_rider", db);
```

In [6]:
```
jan_green_one_rider.head()
```

Out[6]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	14	2014-01-01 00:00:07	2014-01-01 00:08:28	-73.916992	40.771004	-73.888550	
1	16	2014-01-01 00:00:13	2014-01-01 00:07:26	-73.844109	40.721107	-73.816330	
2	17	2014-01-01 00:00:13	2014-01-01 00:18:35	-73.911644	40.767956	-73.946960	
3	18	2014-01-01 00:00:16	2014-01-01 00:04:05	-73.954727	40.800072	-73.946892	
4	19	2014-01-01 00:00:16	2014-01-01 00:08:32	-73.924431	40.743668	-73.934021	

In [7]:
```
jan_green_one_rider.describe()
```

Out[7]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_
count	640964.000000	640964.000000	640964.000000	640964.000000	640964.000000	64(
mean	403753.981295	-73.929510	40.768245	-73.930421	40.764315	
std	232045.540272	0.041355	0.056710	0.049958	0.057734	
min	14.000000	-74.331802	40.552734	-74.478737	40.441864	
25%	202493.750000	-73.955269	40.721291	-73.963234	40.724201	
50%	403116.500000	-73.940575	40.766838	-73.941170	40.763699	
75%	604872.250000	-73.909353	40.810234	-73.903580	40.807499	
max	803852.000000	-72.633492	41.183037	-72.633492	41.367435	

In [8]:
```
jan_green_one_rider.dtypes
```

Out[8]:
```
ride_id                     int64
pickup_datetime      datetime64[ns]
dropoff_datetime     datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
dropoff_latitude            float64
passenger_count               int64
```

```
trip_distance              float64
dtype: object
```

In [9]:
```
print(round(jan_green_one_rider['trip_distance'].sum()), "total miles for ONE RIDER gre
```

```
1837331 total miles for ONE RIDER green taxi data in january 2014 from NYC
```

Selecting the january green taxi data that was split with passenger count = 2

In [10]:
```
jan_green_two_rider = jan_green_two_rider = pd.read_sql("select * from jan_green_two_ri
```

In [11]:
```
jan_green_two_rider.head()
```

Out[11]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	24	2014-01-01 00:00:43	2014-01-01 00:07:44	-73.924568	40.861614	-73.942627	
1	34	2014-01-01 00:01:05	2014-01-01 00:14:43	-73.929337	40.703571	-73.957573	
2	36	2014-01-01 00:01:09	2014-01-01 00:03:59	-73.981842	40.666687	-73.985321	
3	44	2014-01-01 00:01:36	2014-01-01 00:12:24	-73.953888	40.806480	-73.937546	
4	67	2014-01-01 00:02:50	2014-01-01 00:05:51	-73.925217	40.761761	-73.917320	

In [12]:
```
jan_green_two_rider.describe()
```

Out[12]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_
count	56464.000000	56464.000000	56464.000000	56464.000000	56464.000000	5
mean	391011.861717	-73.930385	40.754977	-73.930964	40.752764	
std	232746.447478	0.043352	0.055280	0.051819	0.056788	
min	24.000000	-74.079437	40.573067	-74.484711	40.324463	
25%	189143.750000	-73.957954	40.713748	-73.966669	40.712924	
50%	397935.500000	-73.942066	40.747704	-73.942165	40.750050	
75%	592494.250000	-73.904423	40.804214	-73.900877	40.792474	
max	803851.000000	-73.247894	40.972569	-73.192268	41.233383	

In [13]:
```
jan_green_two_rider.dtypes
```

Out[13]:
```
ride_id                     int64
pickup_datetime      datetime64[ns]
dropoff_datetime     datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
```

```
dropoff_latitude          float64
passenger_count             int64
trip_distance             float64
dtype: object
```

In [14]:
```
print(round(jan_green_two_rider['trip_distance'].sum()), "total miles for TWO RIDER gre
```

```
170068 total miles for TWO RIDER green taxi data in january 2014 from NYC
```

Selecting the february green taxi data that was split with passenger count = 1

In [15]:
```
feb_green_one_rider = pd.read_sql("select * from feb_green_one_rider", db);
```

In [16]:
```
feb_green_one_rider.head()
```

Out[16]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	36	2014-02-01 00:00:02	2014-02-01 00:08:29	-73.959465	40.716217	-73.965782	
1	40	2014-02-01 00:00:08	2014-02-01 00:15:13	-73.990730	40.694611	-73.957352	
2	42	2014-02-01 00:00:13	2014-02-01 00:06:56	-73.953674	40.790752	-73.959160	
3	43	2014-02-01 00:00:24	2014-02-01 00:08:06	-73.944794	40.727329	-73.941681	
4	44	2014-02-01 00:00:27	2014-02-01 00:19:34	-73.977577	40.678616	-73.988533	

In [17]:
```
feb_green_one_rider.describe()
```

Out[17]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	8.151410e+05	815141.000000	815141.000000	815141.000000	815141.000000	815
mean	5.045571e+05	-73.929019	40.767307	-73.929428	40.764263	
std	2.904891e+05	0.041702	0.058114	0.049842	0.059347	
min	3.600000e+01	-74.728561	40.294209	-75.989021	39.357197	
25%	2.524790e+05	-73.955582	40.719196	-73.961983	40.721554	
50%	5.046100e+05	-73.940231	40.765774	-73.939995	40.763062	
75%	7.565040e+05	-73.906891	40.810978	-73.902313	40.810101	
max	1.005242e+06	-73.054497	41.584133	-72.091438	41.585518	

In [18]:
```
feb_green_one_rider.dtypes
```

Out[18]:
```
ride_id                    int64
pickup_datetime     datetime64[ns]
dropoff_datetime    datetime64[ns]
pickup_longitude           float64
```

```
pickup_latitude            float64
dropoff_longitude          float64
dropoff_latitude           float64
passenger_count              int64
trip_distance              float64
dtype: object
```

In [19]:
```
print(round(feb_green_one_rider['trip_distance'].sum()), "total miles for ONE RIDER gre
```

```
2320251 total miles for ONE RIDER green taxi data in february 2014 from NYC
```

Selecting the february green taxi data that was split with passenger count = 2

In [20]:
```
feb_green_two_rider = pd.read_sql("select * from feb_green_two_rider", db);
```

In [21]:
```
feb_green_two_rider.head()
```

Out[21]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	38	2014-02-01 00:00:08	2014-02-01 00:20:33	-73.961670	40.714012	-73.917099	
1	39	2014-02-01 00:00:08	2014-02-01 00:28:26	-73.951134	40.748909	-73.980003	
2	69	2014-02-01 00:00:59	2014-02-01 00:16:54	-73.960213	40.715313	-73.979050	
3	126	2014-02-01 00:02:28	2014-02-01 00:30:48	-73.983704	40.676292	-73.983566	
4	133	2014-02-01 00:02:36	2014-02-01 00:17:17	-73.935623	40.833282	-73.865150	

In [22]:
```
feb_green_two_rider.describe()
```

Out[22]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	7.101700e+04	71017.000000	71017.000000	71017.000000	71017.000000	71
mean	4.983607e+05	-73.930757	40.754266	-73.930883	40.752525	
std	2.871839e+05	0.043510	0.056829	0.051334	0.058684	
min	3.800000e+01	-74.177330	40.574635	-74.728554	39.346779	
25%	2.548690e+05	-73.958336	40.711933	-73.965775	40.710632	
50%	5.020950e+05	-73.942413	40.747322	-73.941162	40.749107	
75%	7.474800e+05	-73.904327	40.804947	-73.900833	40.796864	
max	1.005237e+06	-73.685745	40.983341	-73.306358	41.067734	

In [23]:
```
feb_green_two_rider.dtypes
```

Out[23]:
```
ride_id                     int64
pickup_datetime      datetime64[ns]
```

```
dropoff_datetime     datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
dropoff_latitude            float64
passenger_count               int64
trip_distance               float64
dtype: object
```

In [24]:
```
print(round(feb_green_two_rider['trip_distance'].sum()), "total miles for TWO RIDER gre
```

```
211835 total miles for TWO RIDER green taxi data in february 2014 from NYC
```

Selecting the march green taxi data that was split with passenger count = 1

In [25]:
```
mar_green_one_rider = pd.read_sql("select * from mar_green_one_rider", db);
```

In [26]:
```
mar_green_one_rider.head()
```

Out[26]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	15	2014-03-01 00:00:00	2014-03-01 00:11:44	-73.807571	40.700371	-73.759422	
1	34	2014-03-01 00:00:01	2014-03-01 00:04:27	-73.951355	40.809841	-73.937584	
2	35	2014-03-01 00:00:03	2014-03-01 00:39:11	-73.958801	40.716785	-73.908257	
3	36	2014-03-01 00:00:03	2014-03-01 00:14:32	-73.938881	40.681664	-73.956787	
4	37	2014-03-01 00:00:03	2014-03-01 00:08:42	-73.941376	40.818493	-73.935242	

In [27]:
```
mar_green_one_rider.describe()
```

Out[27]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	1.058404e+06	1.058404e+06	1.058404e+06	1.058404e+06	1.058404e+06	1058
mean	6.467195e+05	-7.393054e+01	4.076543e+01	-7.393090e+01	4.076220e+01	
std	3.729006e+05	4.134149e-02	5.825813e-02	4.988082e-02	5.936801e-02	
min	1.500000e+01	-7.436404e+01	4.031835e+01	-7.582375e+01	3.958759e+01	
25%	3.236348e+05	-7.395638e+01	4.071749e+01	-7.396370e+01	4.071945e+01	
50%	6.474105e+05	-7.394131e+01	4.076383e+01	-7.394112e+01	4.076123e+01	
75%	9.706468e+05	-7.391082e+01	4.081035e+01	-7.390433e+01	4.080824e+01	
max	1.293471e+06	-7.304129e+01	4.225634e+01	-7.241040e+01	4.225480e+01	

In [28]:
```
mar_green_one_rider.dtypes
```

Out[28]:
```
ride_id                    int64
pickup_datetime      datetime64[ns]
dropoff_datetime     datetime64[ns]
pickup_longitude           float64
pickup_latitude            float64
dropoff_longitude          float64
dropoff_latitude           float64
passenger_count              int64
trip_distance              float64
dtype: object
```

In [29]:
```
print(round(mar_green_one_rider['trip_distance'].sum()), "total miles for ONE RIDER gre
```

```
3094100 total miles for ONE RIDER green taxi data in march 2014 from NYC
```

Selecting the march green taxi data that was split with passenger count = 2

In [30]:
```
mar_green_two_rider = pd.read_sql("select * from mar_green_two_rider", db);
```

In [31]:
```
mar_green_two_rider.head()
```

Out[31]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	41	2014-03-01 00:00:07	2014-03-01 00:11:19	-73.964630	40.712296	-73.947220	
1	53	2014-03-01 00:00:16	2014-03-01 00:18:50	-73.917015	40.761211	-73.850166	
2	62	2014-03-01 00:00:25	2014-03-01 00:05:02	-73.941803	40.798351	-73.941078	
3	66	2014-03-01 00:00:28	2014-03-01 00:04:31	-73.962364	40.682636	-73.961952	
4	92	2014-03-01 00:01:05	2014-03-01 00:04:24	-73.925407	40.761841	-73.935768	

In [32]:
```
mar_green_two_rider.describe()
```

Out[32]:

	ride_id	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	passenger_c
count	9.111900e+04	91119.000000	91119.000000	91119.000000	91119.000000	91
mean	6.524365e+05	-73.932978	40.752387	-73.933037	40.750524	
std	3.772053e+05	0.042651	0.056229	0.051104	0.057751	
min	4.100000e+01	-74.438080	40.366768	-74.656998	40.354107	
25%	3.275905e+05	-73.959290	40.709984	-73.968376	40.708351	
50%	6.512600e+05	-73.944229	40.746983	-73.943207	40.748341	
75%	9.737920e+05	-73.911263	40.803995	-73.904236	40.792187	
max	1.293468e+06	-73.559784	41.034164	-73.040619	41.056427	

In [33]:
```
mar_green_two_rider.dtypes
```

Out[33]:
```
ride_id                      int64
pickup_datetime      datetime64[ns]
dropoff_datetime     datetime64[ns]
pickup_longitude            float64
pickup_latitude             float64
dropoff_longitude           float64
dropoff_latitude            float64
passenger_count               int64
trip_distance               float64
dtype: object
```

In [34]:
```
print(round(mar_green_two_rider['trip_distance'].sum()), "total miles for TWO RIDER gre
```

```
281419 total miles for TWO RIDER green taxi data in march 2014 from NYC
```

Algorithm with single and double riders

Function to get distance between coordinates

In [35]:
```
import geopy.distance

def get_distance(lat_origin, lon_origin, lat_end, lon_end):
    coords_1 = (lat_origin, lon_origin)
    coords_2 = (lat_end, lon_end)

    distance = geopy.distance.distance(coords_1, coords_2).mi
    #print("Distance: ", distance, " mi")
    return distance

get_distance(52.2296756, 21.0122287,52.406374, 16.9251681)
```

Out[35]:
```
173.5818455248231
```

In [36]:
```
def time_difference(time_1, time_2):
    elapsedTime = time_2 - time_1
    #print("Elapsed time: ", elapsedTime)
    #print("Total seconds: ", elapsedTime.total_seconds())

    return (elapsedTime.total_seconds())

print("time difference from jan one riders : " + str(time_difference(jan_green_one_ride
print("time difference from jan two riders : " + str(time_difference(jan_green_two_ride

print("time difference from feb one riders : " + str(time_difference(feb_green_one_ride
print("time difference from feb two riders : " + str(time_difference(feb_green_two_ride

print("time difference from mar one riders : " + str(time_difference(mar_green_one_ride
print("time difference from mar two riders : " + str(time_difference(mar_green_two_ride
```

```
time difference from jan one riders : 501.0
time difference from jan two riders : 421.0
time difference from feb one riders : -2677894.0
time difference from feb two riders : -2677944.0
time difference from mar one riders : -5097092.0
time difference from mar two riders : -5097143.0
```

In [37]:
```
def add_speed_to_trip(trip):
    time_1 = trip['pickup_datetime']
    time_2 = trip['dropoff_datetime']

    distance = trip['trip_distance']
    time = time_difference(time_1, time_2)/3600
    speed = distance/time

    #print("Speed: ", speed, "mi/hr")
    return speed
```

In [38]:
```
jan_green_one_rider['speed'] = jan_green_one_rider.apply(add_speed_to_trip, axis = 1)
jan_green_two_rider['speed'] = jan_green_two_rider.apply(add_speed_to_trip, axis = 1)

feb_green_one_rider['speed'] = feb_green_one_rider.apply(add_speed_to_trip, axis = 1)
feb_green_two_rider['speed'] = feb_green_two_rider.apply(add_speed_to_trip, axis = 1)

mar_green_one_rider['speed'] = mar_green_one_rider.apply(add_speed_to_trip, axis = 1)
mar_green_two_rider['speed'] = mar_green_two_rider.apply(add_speed_to_trip, axis = 1)
```

In [39]:
```
def calcSequence(p1_lat, p1_long, p2_lat, p2_long, p3_lat, p3_long, p4_lat, p4_long, sp
                 trip1_do_time, trip2_do_time, delay):

    ## p1 - p2 - p3 - p4 / point1 - point2 - point3 - point4
    #within 5 minutes before -2 pickup, and withing 5minutes of after p3 and p4 dropoff

    #Keeps track of the total distance everytime distance is added
    totalDistance = 0

    # distance between o1 and o2
    distance = get_distance(p1_lat,p1_long,p2_lat,p2_long)
    # time it will take to go from o1 to o2
    time = speed/distance
    # time is in hours so convert to minutes to seconds
    time_seconds = time * 60 * 60
    # delay in seconds
    delay_seconds = delay * 60
    # trip 2 pickupd time range edge

    # [trip2_pickup_edge, trip2_pickup]
    # [12:55 pm, 1:00 pm]
    # trip1_pickup + time seconds = some time that needs to fall in this range
    if (trip1_pu_time + time_seconds >= (trip2_pu_time - delay_seconds)) or (trip1_pu_t
        # Get the trip1 drop off edge
        # p2 - p3
        totalDistance += distance
        distance = get_distance(p2_lat,p2_long,p3_lat,p3_long)

        # time it will take to go from o1 to o2
        time = speed/distance
        # time is in hours so convert to minutes to seconds
        time_seconds = time * 60 * 60
        if (trip2_pickup + time_seconds >= (trip1_do_time)) or (trip2_pu_time + time_se
            totalDistance += distance
            distance = get_distance(p3_lat,p3_long,p4_lat,p4_long)
            # time it will take to go from o1 to o2
            time = speed/distance
            # time is in hours so convert to minutes to seconds
            time_seconds = time * 60 * 60
            if (trip1_do_time + time_seconds >= (trip2_do_time)) or (trip1_do_time + ti
```

```
                totalDistance += distance
                return totalDistance
    return -1



    # get time range permissible for each origin and dropoff
    # start at o1 check if taxi can make it to o2 within delay

    # i.e. if 5 minute delay then make it within 5 min before o2
    # then have to make it within 5 min after d1 and d2 drop off

    # calculate savings if trip is permissible
    # savings = [distance(trip1) + distance(trip2)] - distance(combined trip1 & trip2)

    # o1 - o2 - d2 - d1

    # o2 - o1 - d1 - d2

    # o2 - o1 - d2 - d1



def prototype_algorithm(single_trips, double_trips, delay):
    # columns- 'ride_id', 'pickup_datetime', 'dropoff_datetime', 'pickup_longitude',
    #          'pickup_latitude', 'dropoff_longitude', 'dropoff_latitude',
    #          'passenger_count', 'trip_distance', 'speed'
    combined_trips = {}

    for index, trip1 in single_trips.iterrows():

        for index, trip2 in double_trips.iterrows():
            # average speed of two trips
            speed3 = (trip1['speed'] + trip2['speed'])/2

            # possible sequences:

            # o1 - o2 - d1 - d2

            # o1 - o2 - d2 - d1

            # o2 - o1 - d1 - d2

            # o2 - o1 - d2 - d1
            o1_lon = trip1['pickup_longitude']
            o1_lat = trip1['pickup_latitude']
            o2_lon = trip2['pickup_longitude']
            o2_lat = trip2['pickup_latitude']

            d1_lon = trip1['pickup_longitude']
            d1_lat = trip1['pickup_latitude']
            d2_lon = trip2['pickup_longitude']
            d2_lat = trip2['pickup_latitude']


            pu1 = trip1['pickup_datetime'].total_seconds()
            pu2 = trip2['pickup_datetime'].total_seconds()


            do1 = trip1['dropoff_datetime'].total_seconds()
            do2 = trip2['dropoff_datetime'].total_seconds()
```

```
            # o1 - o2 - d1 - d2
            seq1 = calcSequence(o1_lat, o1_long, o2_lat, o2_long, d1_lat, d1_long, d2_l

            # o1 - o2 - d2 - d1
            seq2 = calcSequence(o1_lat, o1_long, o2_lat, o2_long, d2_lat, d2_long, d1_l

            # o2 - o1 - d2 - d1
            seq3 = calcSequence( o2_lat, o2_long, o1_lat, o1_long, d2_lat, d2_long, d1_

            # o2 - o1 - d1 - d2
            seq4 = calcSequence( o2_lat, o2_long, o1_lat, o1_long, d1_lat, d1_long, d2_

            totalDistance = trip1['trip_distance'] + trip2['trip_distance']


            if seq1 < seq2 and seq1 < seq3 and seq1 < seq4 and seq1 != -1:
                combined_trips[(trip1['ride_id'], trip2['ride_id'])] = totalDistance -
            elif seq2 < seq1 and seq2 < seq3 and seq2 < seq4 and seq2 != -1:
                combined_trips[(trip1['ride_id'], trip2['ride_id'])] = totalDistance -
            elif seq3 < seq1 and seq3 < seq4 and seq3 < seq2 and seq3 != -1:
                combined_trips[(trip1['ride_id'], trip2['ride_id'])] = totalDistance -
            elif seq4 < seq1 and seq4 < seq2 and seq4 < seq3 and seq4 != -1:
                combined_trips[(trip1['ride_id'], trip2['ride_id'])] = totalDistance -

    return combined_trips
```

Perhaps Limit the Longitude and Latitude ranges to make algorithm faster

These values are chosen based on the 25% to 50% quartile ranges for the latitude and longitude to get 25% of the data isntead of 100%

-73.959290 < pu_lon < -73.944229

40.709984 < pu_lat < 40.746983

-73.968376 < do_lon < -73.943207

40.708351 < do_lat < 40.748341

In [40]:
```
jan_one = jan_green_one_rider
jan_two = jan_green_two_rider
```

In [41]:
```
jan_one.shape
```

Out[41]:
```
(640964, 10)
```

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

Out[42]:
```
(56464, 10)
```

In [43]:
```
jan_one = jan_one[(jan_one['pickup_latitude'] > 40.709984) & (jan_one['pickup_latitude'
                  (jan_one['pickup_longitude'] > -73.959290) & (jan_one['pickup_longitu
                  (jan_one['dropoff_latitude'] > 40.708351) & (jan_one['dropoff_latitud
                  (jan_one['dropoff_longitude'] > -73.968376) & (jan_one['dropoff_longi
                 ]
```

In [44]:
```
jan_two = jan_two[(jan_two['pickup_latitude'] > 40.709984) & (jan_two['pickup_latitude'
                  (jan_two['pickup_longitude'] > -73.959290) & (jan_two['pickup_longitu
                  (jan_two['dropoff_latitude'] > 40.708351) & (jan_two['dropoff_latitud
                  (jan_two['dropoff_longitude'] > -73.968376) & (jan_two['dropoff_longi
                 ]
```

In [45]:
```
jan_one.shape
```

Out[45]:
```
(13676, 10)
```

In [46]:
```
jan_two.shape
```

Out[46]:
```
(1813, 10)
```

In [47]:
```
jan_one = jan_one.sort_values(by = ['pickup_datetime', 'pickup_latitude', 'pickup_longi
jan_one.head()
```

Out[47]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	38	2014-01-01 00:01:16	2014-01-01 00:07:05	-73.949501	40.714016	-73.947380	
1	59	2014-01-01 00:02:10	2014-01-01 00:04:54	-73.951622	40.714329	-73.950470	
2	60	2014-01-01 00:02:11	2014-01-01 00:06:53	-73.957886	40.717773	-73.957207	
3	64	2014-01-01 00:02:39	2014-01-01 00:06:04	-73.952599	40.726860	-73.945587	
4	76	2014-01-01 00:03:16	2014-01-01 00:13:14	-73.951637	40.713833	-73.957687	

In [48]:
```
jan_two = jan_two.sort_values(by = ['pickup_datetime', 'pickup_latitude', 'pickup_longi
jan_two.head()
```

Out[48]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	86	2014-01-01 00:03:43	2014-01-01 00:11:39	-73.957870	40.721825	-73.943550	
1	568	2014-01-01 00:16:16	2014-01-01 00:24:40	-73.957764	40.722015	-73.946861	
2	954	2014-01-01 00:24:27	2014-01-01 00:32:28	-73.950150	40.718609	-73.955276	
3	1117	2014-01-01 00:28:14	2014-01-01 00:32:58	-73.945168	40.719082	-73.954559	
4	1373	2014-01-01 00:33:45	2014-01-01 00:39:54	-73.955292	40.736786	-73.956200	

Algorithm

Select one rider

In [49]:
```
# Algorithm
# select one ride
ride_one = jan_one.loc[[0]]
ride_one
```

Out[49]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	38	2014-01-01 00:01:16	2014-01-01 00:07:05	-73.949501	40.714016	-73.94738	

Select two rider

In [50]:
```
ride_two = jan_two.loc[[0]]
ride_two
```

Out[50]:

	ride_id	pickup_datetime	dropoff_datetime	pickup_longitude	pickup_latitude	dropoff_longitude	dro
0	86	2014-01-01 00:03:43	2014-01-01 00:11:39	-73.95787	40.721825	-73.94355	

Check if combinable with delay 5 minutes

In [51]:
```
delay = 5
print("Delay:", delay, "minutes")
```

```
Delay: 5 minutes
```

1. get speed of combined trips

In [52]:
```
speed = (ride_one['speed'][0] + ride_two['speed'][0])/2
print(speed, "mi/hr")
```

```
10.097505477835835 mi/hr
```

2a. get distance between pickup points

In [53]:
```
pu_distance = get_distance(ride_one['pickup_latitude'][0], ride_one['pickup_longitude']
                           ride_two['pickup_latitude'][0], ride_two['pickup_longitude']
print(pu_distance, "mi")
```

```
0.6952721673692104 mi
```

2b. get distance between dropoff points

In [54]:
```
do_distance = get_distance(ride_one['dropoff_latitude'][0], ride_one['dropoff_longitude
                           ride_two['dropoff_latitude'][0], ride_two['dropoff_longitude
print(do_distance, "mi")
```

```
0.2160517717605075 mi
```

3a. get time it will take to travel distance between PU points

In [55]:
```
pu_travel_time = (pu_distance/speed) * 60 # in hours, change to minute by multiplying 6
print(pu_travel_time, "minutes")
```

```
4.131350077870277 minutes
```

3b. get time it will take to travel distance between DO points

In [56]:
```
do_travel_time = (do_distance/speed) * 60 # in hours, change to minute by multiplying 6
print(do_travel_time, "minutes")
```

```
1.28379294609789 minutes
```

4a. compare PU times to delay + time from 3a to see if trip is likely valid

In [57]:
```
from datetime import datetime as dt
import pandas as pandas
```

In [58]:
```
# get pickup times for both rides

ride_one_pu_time = (pandas.to_datetime(ride_one['pickup_datetime'][0]).second +
                    pandas.to_datetime(ride_one['pickup_datetime'][0]).minute*60 +
                    pandas.to_datetime(ride_one['pickup_datetime'][0]).hour*60*60)

ride_two_pu_time = (pandas.to_datetime(ride_two['pickup_datetime'][0]).second +
                    pandas.to_datetime(ride_two['pickup_datetime'][0]).minute*60 +
                    pandas.to_datetime(ride_two['pickup_datetime'][0]).hour*60*60)

distance = -1
trip_sequence = []
# combo: o1 -> o2
# have to make it by 5 minutes before pickup 2
if (ride_one_pu_time + pu_travel_time) < (ride_two_pu_time - delay):
    distance = pu_distance
    combo = (ride_one['ride_id'][0], ride_two['ride_id'][0])
# combo: o2 -> o1
# have to make it by 5 minutes before pickup 1
if (ride_two_pu_time + pu_travel_time) < (ride_one_pu_time - delay):
    distance = pu_distance
    combo = (ride_two['ride_id'][0], ride_one['ride_id'][0])

if distance == -1:
    print("FAIL")
else:
    print("Ride 1 id: ", ride_one['ride_id'][0])
    print("Ride 2 id: ", ride_two['ride_id'][0])
    print()
    print("Trip combinable for pickup segment with distance", distance)
    print("Combinable sequence is: ", combo)
```

```
Ride 1 id:  38
Ride 2 id:  86

Trip combinable for pickup segment with distance 0.6952721673692104
Combinable sequence is:  (38, 86)
```

4b. compare DO times to delay + time from 3b to see if trip is likely valid

In [59]:
```
do_time_difference = time_difference(ride_one['dropoff_datetime'][0], ride_two['dropoff
print(do_time_difference, "seconds")
```

```
274.0 seconds
```

In [60]:
```
# get dropoff times for both rides

ride_one_do_time = (pandas.to_datetime(ride_one['dropoff_datetime'][0]).second +
```

```
                    pandas.to_datetime(ride_one['dropoff_datetime'][0]).minute*60 +
                    pandas.to_datetime(ride_one['dropoff_datetime'][0]).hour*60*60)

ride_two_do_time = (pandas.to_datetime(ride_two['dropoff_datetime'][0]).second +
                    pandas.to_datetime(ride_two['dropoff_datetime'][0]).minute*60 +
                    pandas.to_datetime(ride_two['dropoff_datetime'][0]).hour*60*60)

distance = -1
trip_sequence = []
# combo: d1 -> d2
# have to make it by 5 minutes before pickup 2
if (ride_one_do_time + do_travel_time) < (ride_two_do_time + delay):
    distance = do_distance
    combo = (ride_one['ride_id'][0], ride_two['ride_id'][0])

# combo: d2 -> d1
# have to make it by 5 minutes before pickup 1
if (ride_two_do_time + do_travel_time) < (ride_one_do_time + delay):
    distance = do_distance
    combo = (ride_two['ride_id'][0], ride_one['ride_id'][0])

if distance == -1:
    print("FAIL")
else:
    print("Ride 1 id: ", ride_one['ride_id'][0])
    print("Ride 2 id: ", ride_two['ride_id'][0])
    print()
    print("Trip combinable for pickup segment with distance", distance)
    print("Combinable sequence is: ", combo)
```

```
Ride 1 id:  38
Ride 2 id:  86

Trip combinable for pickup segment with distance 0.2160517717605075
Combinable sequence is:  (38, 86)
```

Analyzing results

For combining trip in the pickup segment the order of trips (ride_one id, ride_two id) = (38, 86) also can be said to be (o1, o2) is a valid order for delay given

For combining tip in the dropoff segment the order of trips (ride_one id, ride_two id) = (38, 86) also can be said to be (d1, d2) is a valid order for delay given

Hypothesis:

If the pickup segment and dropoff segment of ridesharing is valid then the center part of the trip that varies in distance should be valid

If a valid solution is found then there is an optimal combination for the rideshare.

5. Checking if sequence will result in saved distance for combining ride

In [61]:
```
# Sequence is o1, o2, d1, d2

# get distance between o2, d1
distance = get_distance(ride_two['pickup_latitude'][0], ride_two['pickup_longitude'][0]
                        ride_one['dropoff_latitude'][0], ride_one['dropoff_longitude'][
print("Distance between o2, d1: ", distance, "mi")
```

```
# get the time it will take to travel in the center part of the trip from o2 to d1
center_travel_time = distance/speed

# check if will make it within delay
if (ride_two_pu_time + center_travel_time) < (ride_one_do_time + delay):
    rideshare_distance = pu_distance + distance + do_distance

    print("Rideshare total trip distance: ", rideshare_distance, "mi")
```

```
Distance between o2, d1:  0.9805547868624445 mi
Rideshare total trip distance:  1.8918787259921623 mi
```

6. Seeing if there was a saving in distance

In [62]:
```
original_distance = ride_one['trip_distance'][0] + ride_two['trip_distance'][0]
print("Original total distance of the individual trips: ", original_distance, "mi")
```

```
Original total distance of the individual trips:  2.35 mi
```

Results:

The combined rideshare trip was about 1.89 mi

The total of the individual rides is 2.35 mi

7. Getting distance savings

In [64]:
```
print("The distance saved by combining ride 1 with id", ride_one['ride_id'][0],
      "and combining ride 2 with id", ride_two['ride_id'][0], "is",
      original_distance - rideshare_distance, "mi", "\nwithin a delay of", delay, "minu
      ride_one['passenger_count'][0] + ride_two['passenger_count'][0], "passengers"
     )
```

```
The distance saved by combining ride 1 with id 38 and combining ride 2 with id 86 is 0.4
5812127400783775 mi
within a delay of 5 minutes and a total of 3 passengers
```