A2

March 3, 2019

0.0.1 Name: Binben Wang

0.0.2 NSID: biw123

```
In [375]: import os
          import folium
          import numpy as np
          import pandas as pd
          import pyproj as pj
          import math
          from folium.plugins import HeatMap
          from collections import Counter
          import matplotlib
          import matplotlib.pyplot as plt
          from scipy.ndimage.interpolation import shift
          import scipy.stats as stats
          import pylab as pl
          %matplotlib inline
          %config InlineBackend.figure_format = 'retina'
```

1 STEP 1: Preparation

```
In [2]: # gps.csv contains five features: user_id, record_time, latitude, longitude, accu
        gps_df = pd.read_csv("../A1/gps.csv")
        # battery.csv contains two features: user_id, record_time
        battery_df = pd.read_csv("../A1/battery.csv")
```

```
In [3]: # latitude and longitude of saskatoon center
        lat_center_saskatoon = (52.058367 + 52.214608) / 2
        lon_center_saskatoon = (-106.7649138128 - 106.52225318) / 2
```

```
In [4]: # calculate number of total possible battery records per particiants
        battery_df.record_time = battery_df.record_time.astype('datetime64')
        time_interval = battery_df.record_time.max() - battery_df.record_time.min()
        num_total_possible_records = np.ceil(time_interval.total_seconds() / 300)
```

```
In [5]: # calculate the number of all battery records per user.
        battery_counts_df = battery_df.groupby(['user_id']).size().reset_index(name='counts')
        battery_counts_df.describe()
```

Out[5]:
```
            user_id       counts
count   108.000000   108.000000
mean   1084.518519  3276.287037
std     274.204286  2718.917432
min     264.000000     1.000000
25%     939.750000   646.750000
50%    1077.500000  2233.000000
75%    1315.250000  6155.750000
max    1364.000000  7655.000000
```

In [6]:
```python
# filter battery dataframe in 50% threshold
fifty_user_df = battery_counts_df[battery_counts_df.counts >= num_total_possible_recor
fifty_user_df.count()
```

Out[6]:
```
user_id    43
counts     43
dtype: int64
```

In [9]:
```python
analysis_gps_df = gps_df[(gps_df.lat <= 52.214608) & (gps_df.lat >= 52.058367) & (gps_
```

In [11]:
```python
# save the gps records that we will analyze into csv file
analysis_gps_df.to_csv('analysis_gps.csv', index=False)
```

In [12]:
```python
analysis_gps_df = pd.read_csv("analysis_gps.csv")
```

In [14]:
```python
# convert the datatype of record_time from object to datatime64
analysis_gps_df.record_time = analysis_gps_df.record_time.astype('datetime64')
# calculate the minimum timestamp of all records
min_timestamp = analysis_gps_df.record_time.min()
```

In [15]:
```python
# convert record_time to relative_time by duty cycle
analysis_gps_df.loc[:, 'relative_time'] = np.floor((analysis_gps_df.record_time - min_
# We don't need the 'accu' column in further analysis
analysis_gps_df = analysis_gps_df.drop(columns=['accu'])
```

In [19]:
```python
# group the dataframe by user_id, and relative_time, then calculate the average latit
analysis_gps_df_aggregated = analysis_gps_df.groupby(['user_id', 'relative_time'], as_
```

In [21]:
```python
# transfer latitude and longitude coordinates to UTM coordinates
p1 = pj.Proj(init='epsg:32613')
x, y = p1(analysis_gps_df_aggregated.lon.values.tolist(), analysis_gps_df_aggregated.l
analysis_gps_df_aggregated.loc[:, 'x'] = x
analysis_gps_df_aggregated.loc[:, 'y'] = y
```

In [22]:
```python
# drop lat and lon columns, we don't need that
analysis_gps_df_aggregated = analysis_gps_df_aggregated.drop(columns=['lat', 'lon'])
```

In [23]:
```python
# save gps records in UTM coordinates into csv file
analysis_gps_df_aggregated.to_csv('aggregated_x_y_location.csv', index=False)
```

```
In [24]: # calculate the start and end coordinates in UTM for the greater Saskatoon area
         start_x, start_y = p1(-106.7649138128, 52.058367)
         end_x, end_y = p1(-106.52225318, 52.214608)
```

```
In [25]: # convert UTM coordinates into grid coordinates
         def which_grid(x, y, start_x, start_y, step):
             which_grid_x = np.floor((x - start_x) / step)
             which_grid_y = np.floor((y - start_y) / step)

             # in case UTM coordinates in the boundary of the greater Saskatoon area
             # there are some -1 value appearing
             # set all -1 to 0
             which_grid_x[which_grid_x < 0] = 0
             which_grid_y[which_grid_y < 0] = 0

             return(which_grid_x, which_grid_y)
```

2 STEP 2: Trip Definition

For N-times definition, I include firt N repeated grids into the following trip but include only 1 grid when the trip ends due to more than or equal to N repeated grids.

For example, given grid sequence: AAAABBCDDDDDEFGG

AAABBCD is a trip. And DDDEFGG is another trip.

```
In [365]: def operationalize(file_path, lat_center_saskatoon, lon_center_saskatoon, start_x, er
              # load the data we want to operationalize
              df = pd.read_csv(file_path)

              # convert UTM coordinates into grid coordinates
              df.loc[:, 'grid_x'], df.loc[:, 'grid_y'] = which_grid(df.x.values, df.y.values, s
              df = df.drop(columns=['x', 'y'])

              #convert datatype from float64 to int
              df.grid_x = df.grid_x.astype('int')
              df.grid_y = df.grid_y.astype('int')

              # sort dataframe
              df = df.sort_values(by=['user_id', 'relative_time'])

              # whether an observation is in a trip
              not_in_trip = ((df.grid_x == shift(df.grid_x, -1, cval=0)) & (df.grid_y == shift
              for i in range(1, N):
                  not_in_trip = not_in_trip & ((shift(df.grid_x, -1 * i, cval=0) == shift(df.gr
              df.loc[:, 'not_in_trip'] = not_in_trip
              df.not_in_trip = df.not_in_trip.astype('int')

              # cumsum of not_in_trip
              df.loc[:, 'trip'] = df.not_in_trip.cumsum()
```

```
        # calculate trip numbers


        # calculate trip duration and trip numbers
        temp = df
        durations_per_participant = temp.groupby(['user_id', 'trip']).size().reset_index
        # filter all durations less than or equal N, they are not trips
        durations_per_participant = durations_per_participant[durations_per_participant.
        num_of_trips_per_participant = durations_per_participant.groupby(['user_id']).si
        tn = num_of_trips_per_participant[['user_id', 'num_of_trip']].values
        durations = temp.groupby(['trip']).size().reset_index(name='duration')
        counts_of_durations = durations.groupby(['duration']).size().reset_index(name='nu
        td = counts_of_durations[['duration', 'num_of_duration']].values

        # calculate trip duration
        temp = df
        temp.loc[:, 'change_grid'] = ((temp.grid_x != shift(temp.grid_x, -1, cval=0)) |
        temp.change_grid = temp.change_grid.astype('int')
        lengths = temp.groupby(['trip']).change_grid.sum().reset_index(name='length')
        counts_of_lengths = lengths.groupby(['length']).size().reset_index(name = 'num_o
        tl = counts_of_lengths[['length', 'num_of_length']].values

        # Convert grid coordinates back to lat and lon coordinates
        temp_x = (df.grid_x + 0.5) * step + start_x
        temp_y = (df.grid_y + 0.5) * step + start_y
        temp_x[temp_x > end_x] = end_x
        temp_y[temp_y > end_y] = end_y
        df.loc[:, 'center_lon'], df.loc[:, 'center_lat'] = p1(temp_x.tolist(), temp_y.to

        # render the heatmap including only trips
        m1 = folium.Map([lat_center_saskatoon, lon_center_saskatoon], tiles='stamentoner
        HeatMap(df.loc[df.not_in_trip == 0, ['center_lat', 'center_lon']].values.tolist(
        filename = str(N) + "_times_trip_only_heatmap.html"

        # render the heatmap including only non-trips
        m2 = folium.Map([lat_center_saskatoon, lon_center_saskatoon], tiles='stamentoner
        HeatMap(df.loc[df.not_in_trip == 1, ['center_lat', 'center_lon']].values.tolist(
        filename = str(N) + "_times_non_trip_only_heatmap.html"

        return (td, tl, tn, m1, m2)
```

3 STEP 3: Presentation

```
In [366]: td_1, tl_1, tn_1, m_trip_1, m_nontrip_1 = operationalize('aggregated_x_y_location.cs
```

```
In [367]: m_trip_1
```

```
Out[367]: <folium.folium.Map at 0xb19851f98>
```

```
In [368]: m_nontrip_1
```

```
Out[368]: <folium.folium.Map at 0xb274746d8>
```

```
In [369]: td_3, tl_3, tn_3, m_trip_3, m_nontrip_3 = operationalize('aggregated_x_y_location.csv
```

```
In [370]: m_trip_3
```

```
Out[370]: <folium.folium.Map at 0xb1b4d0ef0>
```

```
In [371]: m_nontrip_3
```

```
Out[371]: <folium.folium.Map at 0xb1b4d0ac8>
```

```
In [372]: td_5, tl_5, tn_5, m_trip_5, m_nontrip_5 = operationalize('aggregated_x_y_location.csv
```

```
In [381]: m_trip_5
```

```
Out[381]: <folium.folium.Map at 0xb314f9f60>
```

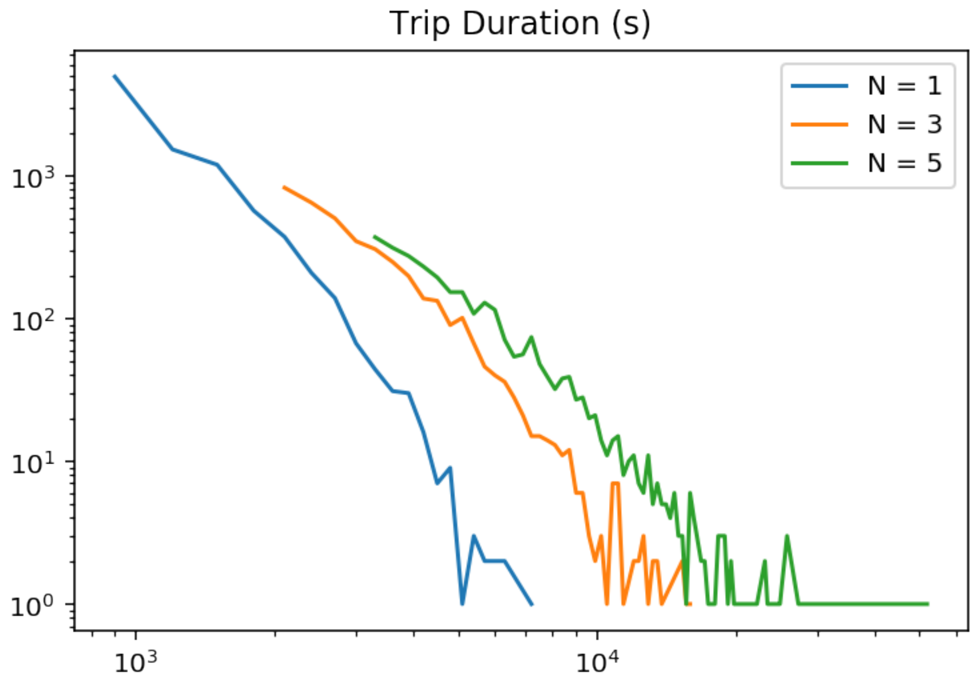
```
In [374]: m_nontrip_5
```

```
Out[374]: <folium.folium.Map at 0xb1bc22128>
```

3.0.1 Choice of Distribution for Trip Duration

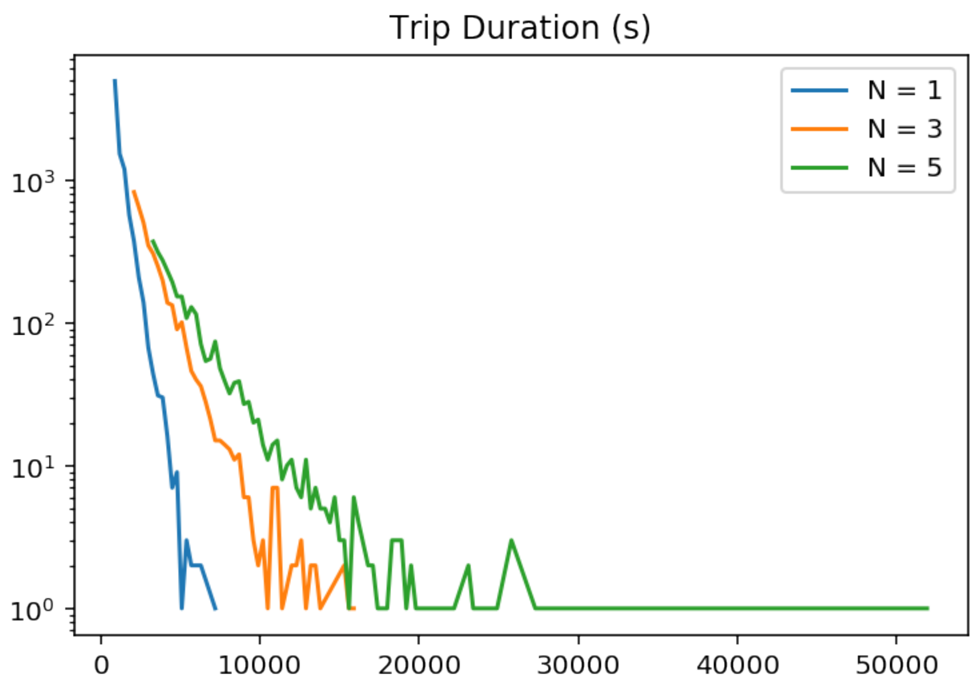
Try 1: power law

```
In [376]: # No trip duration can be less than N + 1 when using N-times definition
          plt.loglog(td_1[2:,0] * 5 * 60, td_1[2:,1])
          plt.loglog(td_3[4:,0] * 5 * 60, td_3[4:,1])
          plt.loglog(td_5[6:,0] * 5 * 60, td_5[6:,1])
          plt.legend(["N = 1", "N = 3", "N = 5"])
          plt.title("Trip Duration (s)")
          plt.show()
```



Try 2: exponential

```
In [377]: plt.semilogy(td_1[2:,0] * 5 * 60, td_1[2:,1])
          plt.semilogy(td_3[4:,0] * 5 * 60, td_3[4:,1])
          plt.semilogy(td_5[6:,0] * 5 * 60, td_5[6:,1])
          plt.legend(["N = 1", "N = 3", "N = 5"])
          plt.title("Trip Duration (s)")
          plt.show()
```

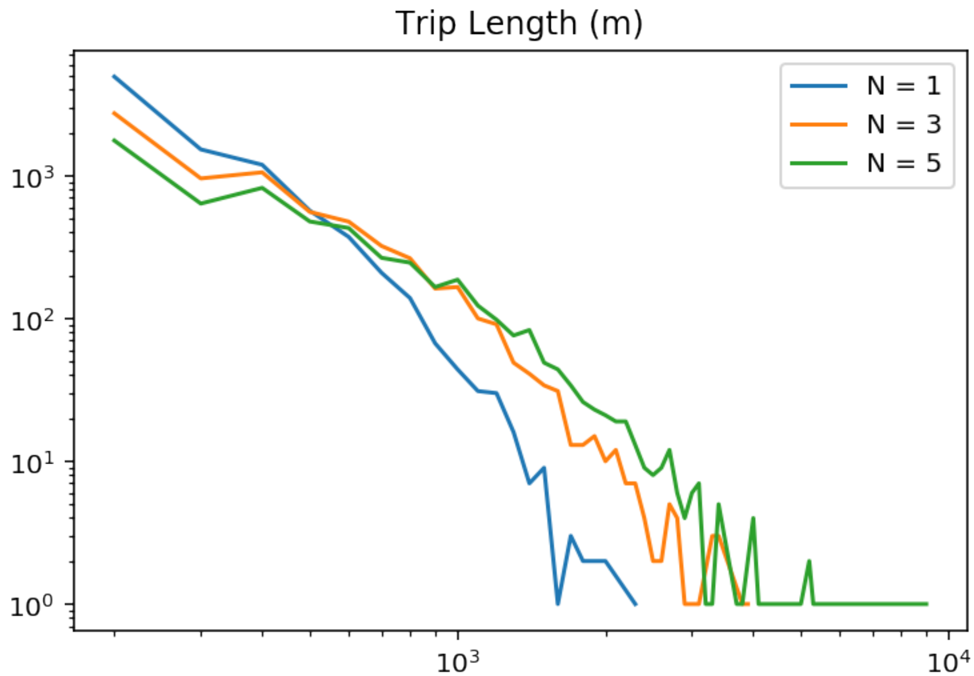


As we can see, both power law distribtions and expoential distribtions fit trip duration pretty well. One drawback of expoential distribution in this case is that the slope of lines in the above figure is too large. When we try to fit this distribution, it may cause large errors. And gaussian distributions are bell curves. They definitely won't fit trip durations.

3.0.2 Choice of Distribution for Trip Length

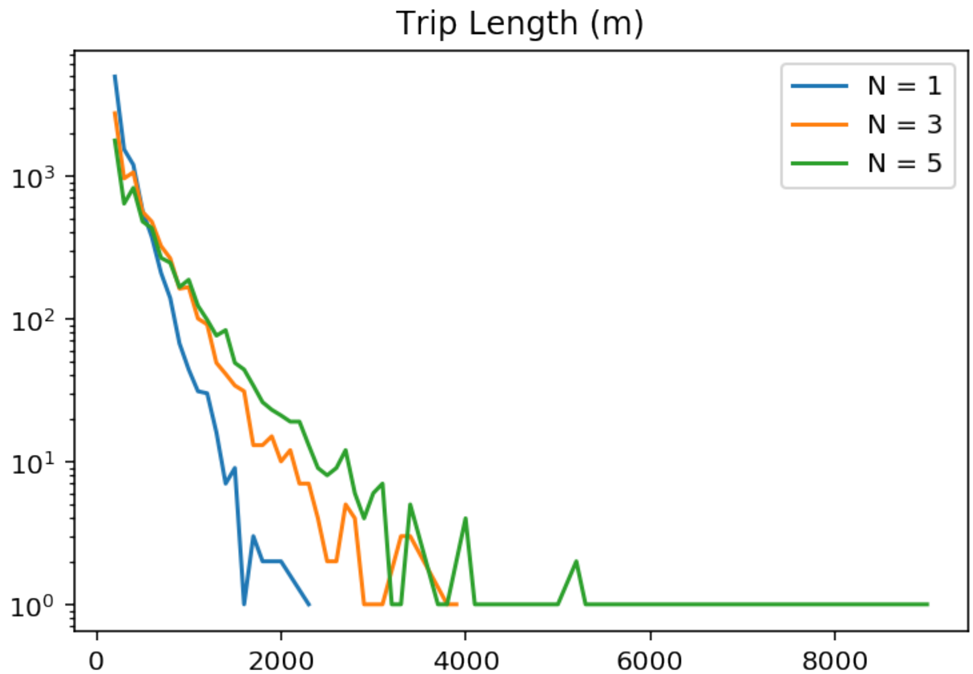
Try 1: power law

```
In [378]: # No trip length can be less than 2
          plt.loglog(tl_1[2:,0] * 100, tl_1[2:,1])
          plt.loglog(tl_3[2:,0] * 100, tl_3[2:,1])
          plt.loglog(tl_5[2:,0] * 100, tl_5[2:,1])
          plt.legend(["N = 1", "N = 3", "N = 5"])
          plt.title("Trip Length (m)")
          plt.show()
```



Try 2: exponential

```
In [379]: # No trip length can be less than 2
          plt.semilogy(tl_1[2:,0] * 100, tl_1[2:,1])
          plt.semilogy(tl_3[2:,0] * 100, tl_3[2:,1])
          plt.semilogy(tl_5[2:,0] * 100, tl_5[2:,1])
          plt.legend(["N = 1", "N = 3", "N = 5"])
          plt.title("Trip Length (m)")
          plt.show()
```



Reasons are the same as reasons for trip durations.

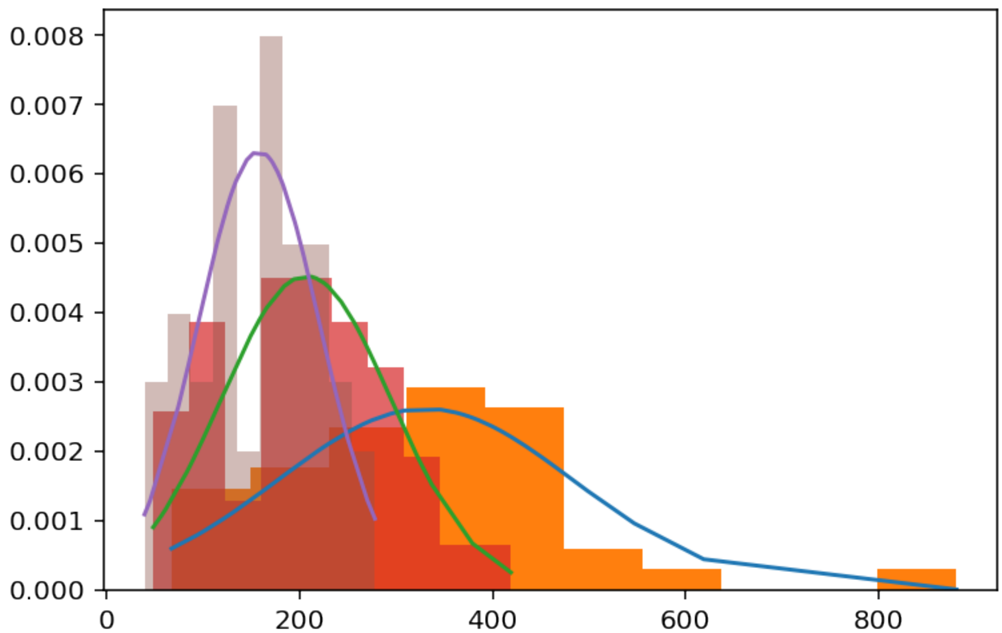
3.0.3 Choice of Distribution for Trip Number

In [380]:
```
h_1 = sorted(tn_1[:, 1])
fit = stats.norm.pdf(h_1, np.mean(h_1), np.std(h_1))
pl.plot(h_1,fit)
pl.hist(h_1,density=True, alpha=1)

h_3 = sorted(tn_3[:, 1])
fit = stats.norm.pdf(h_3, np.mean(h_3), np.std(h_3))
pl.plot(h_3,fit)
pl.hist(h_3,density=True, alpha=0.7)

h_5 = sorted(tn_5[:, 1])
fit = stats.norm.pdf(h_5, np.mean(h_5), np.std(h_5))
pl.plot(h_5,fit)
pl.hist(h_5,density=True, alpha=0.4)

pl.show()
```



From all three histogram graphs, we know that the probability (counts) of a trip number first increases then decreases as trip numbers increase. So the distribution of trip numbers is a bell curve, which is the shape of gaussian distributions. Whereas the probability of exponential distributions and power law distribtions decreases all the time as x increases. So these two distributions will not fit trip numbers.

4 STEP 4: Interpretation

4.1 What is the impact of changing N on the distribution of trip number, length and duration? What trips are being captured and which are being ignored? Given an example of a research question where the differences would be important, and an example of a question where they would be unimportant.

- When increasing N, distributions of trip number, trip length and trip duration keep the same. However, the distribtion of trip number has larger mean and larger standard deviation, which is shown as more flat bell curves. Distributions of trip length and trip duration decrease slower as N increases.
- Trips from one grid to another grid will be captured depending on N. Trips within a single grid will be ignored
- When comparing trip behaviors of people from different areas, this will be important.
- When studying trip behaviors of certain group of people from same area, this would be unimportant.

4.2 What distinguishing features did you see in the heatmaps? Where there points included in either map (trip, not trip) at any N that seemed out of place? How would you change the operationalization to eliminate these points?

- As N becoming larger, there are more data on trip heatmaps and less data on non-trip heatmaps. Because more and more GPS will be categorized as trips. And we also can see that people are often in the building when they are in non-trips and are often in the road when they are in trips.
- Many GPS records in shopping malls are categorized as non-trips. I think this is uncorrect.
- I can decrease the time period used to down-sampling, we are more likely to see people moving in shopping malls when the time period is small.

4.3 Using the DADeP as headings, describe the entire process of this assignment as a series of algorithms. Assuming you have access to a function probability_of_location(location) describe the process using the dynamic location formalism described in class.

- Filter: first of all, I remove all unvalid data including GPS data with low accuracy, GPS data out of Saskatoon, and participants without enough records.
- Stratify: Then I separate GPS data into two types, one is in trips and the another is in non-trips.
- Aggregate: I aggregate data by duty cycles and grids.
- Model: I use different distributions to fit trip number, trip length, and trip duration.