An Analysis of Exercise Performance

11/25/2019

In this notebook, we will explore the exercise data of Professor Christopher Brooks. We will look at through the lens of statistical relationship exploration, performance trends and improvements, and performance patterns. We will look at a dataset comprised of his running and cycling performance and statistics over the course of July to October 2019 and see what information we can glean.

Allison Bergmann
MADS Candidate

In [1]:
```
import pandas as pd
import numpy as np
import calendar
import warnings
```

In [2]:
```
data = pd.read_csv('assets/strava.csv')
#data.head()
```

Cleaning the Data

Upon further analysis of the dataset, there is a lot going on with this data. The dataset represents two different types of data: running and cycling. Since the data here has the potential to be drastically different, we will label each activity, grouped by datafile, as either a running or cycling activity.

Per discussions with the client, it would seem he did not get a bike until September. So we can rule out any activity before that date as running. Additionally, we want to be able to sort. Now, as much of an athlete as Prof. Brooks is, I doubt he would run faster than the world record of the 1000m sprint, which is at 7.45m/s. That said, we will make the threshold dynamic.

In [3]:
```
def define_activity(df, threshold=7.45):
    df0 = df.groupby('datafile').agg({"enhanced_speed":(np.max,np.min)
})
    df0.columns =['trip_max', 'trip_min']
    df = pd.merge(df, df0, on=['datafile'])
    df['activity'] = np.where((df['trip_max'] > threshold) & (df['time
stamp'] > '2019-09-01'), 'biking', 'running')
    return df

df = define_activity(data, threshold=7.45)

#df['mph'] = df['enhanced_speed'] * 2.23694                          # Im
plemented for sanity check, as Americans
#df2 = df.groupby(['activity','datafile']).agg({'mph':np.mean})     # me
asure in dreams instead of logic like
#df2                                                                 # ev
eryone else.
```

```
# And ensure datatypes of targeted columns
df[['cadence', 'enhanced_altitude', 'enhanced_speed', 'heart_rate', 'd
istance']].astype(float)
# We will also break out dates and times separately for analysis.
df['date'] = pd.to_datetime(df['timestamp']).apply(lambda x: x.date())
df['time'] = pd.to_datetime(df['timestamp']).apply(lambda x: x.time())

df.head()
```

Out[3]:

	Air Power	Cadence	Form Power	Ground Time	Leg Spring Stiffness	Power	Vertical Oscillation	altitude	cadence	
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	activities
1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	0.0	activities
2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	54.0	activities
3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3747.0	77.0	activities
4	NaN	NaN	NaN	NaN	NaN	NaN	NaN	3798.0	77.0	activities

5 rows × 27 columns

With this "define activity" function and a little tweaking of the threshold, we find that there is only one day in particular that has running data near that threshold - July 24th. This indicates that there may be some outlier data we want to normalize for. Or that a bear came up behind him.

There are a multitude of gadgets Chris is using to track his health data. There are certain columns tracking data from different devices, but ultimately it appears the devices can override data in certain categories, such as heart rate, speed, etc.

Due to the fact that the combination of all of this data makes column usage inconsistent outside of the original Strava dataset, e.g. there is no Vertical Oscillation in cycling, the Cadence is only in running (sometimes) and overrides the cadence (lowercase c), while similarly the speed is only sometimes accounted for, but "enhanced speed" is always accounted for, we will use the Strava data to compare, as it is consistent and "apples to apples."

Finding Relationships

Let's see what we can gather from the data thus far by comparing what, if any, trends exist in the consistent dataset by charting them against each other to see if any obvious trends can be identified.

We will limit our splom dataset to altitutde, cadence, speed, and heart rate. This is to see if there is any relationship between any of these factors, i.e. can Professor Brooks increase his heart rate to get to a certain target level by trying to maintain a certain speed, altitude, or cadence. Similarly, do his cadence and speed have anything to do with each other, and so on.

Additionally, we will color the activities differently to see if there is any decipherable pattern that exists in running or cycling that may not exist in the other.

In [4]:
```
import seaborn as sns

# Let's limit the dataset we are passing through to the sploms.
splom = df[['enhanced_altitude', 'cadence', 'enhanced_speed', 'heart_r
ate','activity']].copy()

# And clean out any NaN values
splom.dropna(inplace=True)

# Finally, let's plot using Seaborn.
sns.pairplot(splom,
             hue='activity',
             palette='husl',
             #kind='reg',           # Attempted to incorporate a regre
ssion, but it did not provide any insight.
             diag_kind='hist',
             plot_kws={'alpha':0.5}
             #plot_kws={'line_kws':{'color':'blue'},
             #          'scatter_kws':{'alpha':0.5}
             #         }
            )

warnings.simplefilter(action='ignore', category=FutureWarning)
```

Analysis

Alright, so this is a little difficult to read. Despite a few attempts at deciphering some sort of regression, there doesn't seem to be much useful information here. We have over 40,000 datapoints for this dataset, nearly every few seconds of each activity, for an entire summer. There are a few interesting shapes in here:

- *Cadence is consistently in the ~75-80 range for running vs. enhance altitude, but is all over the board for biking. This indicates a steadier pace while running, regardless of altitude, but a more inconsistent one while biking, even though the same general altitude is maintained.*
- *Speed changes are very drastic, though altitude stays much the same for biking, meanwhile running sees a greater range in altitude, but a relatively consistent speed.*
- *Heart rate for biking seems to be in teh same general range, regardless of cadence, speed, or altitude. However, for running, it seems to have a wider range and more interesting phenomena, especially with enhanced altitude, where an increase seems to lead to more sporadic heart rate changes.
*

Consolidating Data for Readability

Let's see if we can decipher more by rolling up each activity into it's own datapoint (row) and analyzing them from a different perspective.

In [5]:
```
def create_activity_datapoint(df):

    # Create a list of the individual activities logged.
    datafiles = set()
    for file in df['datafile'].unique():
```

```
        datafiles.add(file)
    datafiles = list(datafiles)

    # Sort through the dataframe to isolate a single logged activity.
    activities = {}
    for record in datafiles:
        df0 = df[df['datafile'] == record]
        df0.sort_values(by='timestamp')
        df0.reset_index()

        # Calculate the elapsed time of the activity
        start = df0['timestamp'].iloc[0]
        end = df0['timestamp'].iloc[-1]
        elapsed_time = pd.to_datetime(end) - pd.to_datetime(start)

        # Create a dictionary of the stats from the activity.
        info = {'date' : df0['date'].iloc[0],
                'datafile' : df0['datafile'].iloc[0],
                'activity' : df0['activity'].iloc[0],
                'length' : elapsed_time,
                'distance' : df0['distance'].iloc[-1],
                'max_hr' : df0['heart_rate'].max(),
                'min_hr' : df0['heart_rate'].min(),
                'avg_hr' : df0['heart_rate'].mean(),
                'max_speed' : df0['enhanced_speed'].max(),
                'min_speed' : df0['enhanced_speed'].min(),
                'avg_speed' : df0['enhanced_speed'].mean(),
                'max_altitude' : df0['enhanced_altitude'].max(),
                'min_altitude' : df0['enhanced_altitude'].min(),
                'avg_altitude' : df0['enhanced_altitude'].mean(),
                'max_cadence' : df0['cadence'].max(),
                'min_cadence' : df0['cadence'].min(),
                'avg_cadence' : df0['cadence'].mean()
               }

        # Append the activity stats to the activities dictionary with
key of the activity code.
        activities[record] = info

    # Create a dataframe of the activities and their corresponding sta
ts.
    df = pd.DataFrame(activities)
    df = df.T
    df['date'] = pd.to_datetime(df['date'])
    df['month'] = df['date'].dt.month
    df['month'] = df['month'].apply(lambda x: calendar.month_name[x])

    return df
```

In [6]:
```
ind_activities = create_activity_datapoint(df)
ind_activities.head()
```

Out[6]:

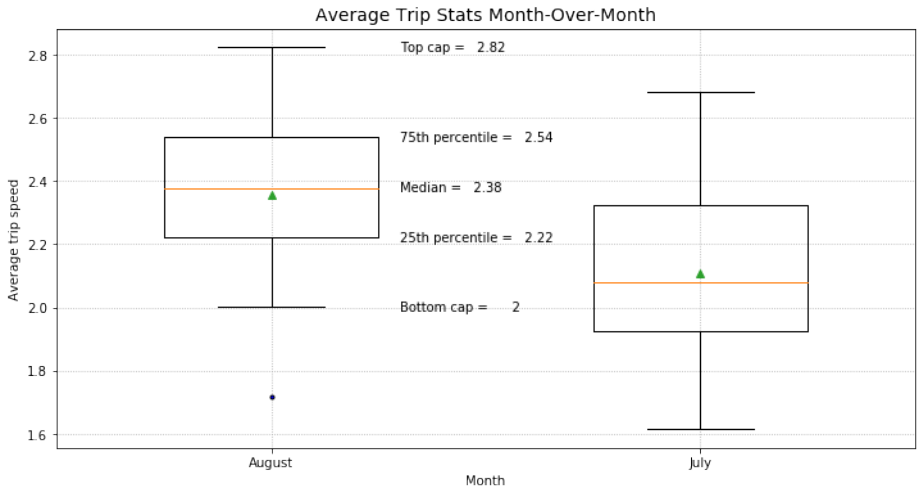
	activity	avg_altitude	avg_cadence	avg_hr	avg_speed	
activities/2796644542.fit.gz	running	257.404	76.9589	138.103	2.41177	activities/27
activities/2700298434.fit.gz	running	276.317	65.1359	109.853	1.82802	activities/27
activities/2694503073.fit.gz	running	269.759	76.7564	110.897	2.30041	activities/26
activities/2728413477.fit.gz	running	240.815	76.4921	118.137	2.16228	activities/27
activities/2729896401.fit.gz	running	294.445	72.8319	99.1877	1.67604	activities/27

Okay, cool, so now that we have data about each of the activities, let's see how Prof. Brooks' stats compare month-over-month to see if there is any change. Since running has a dataset over more than one month (biking only has September and one day in October), we will focus on this, but the code will be dynamic to include biking data if the dataset were to expand to include future months of cycling.

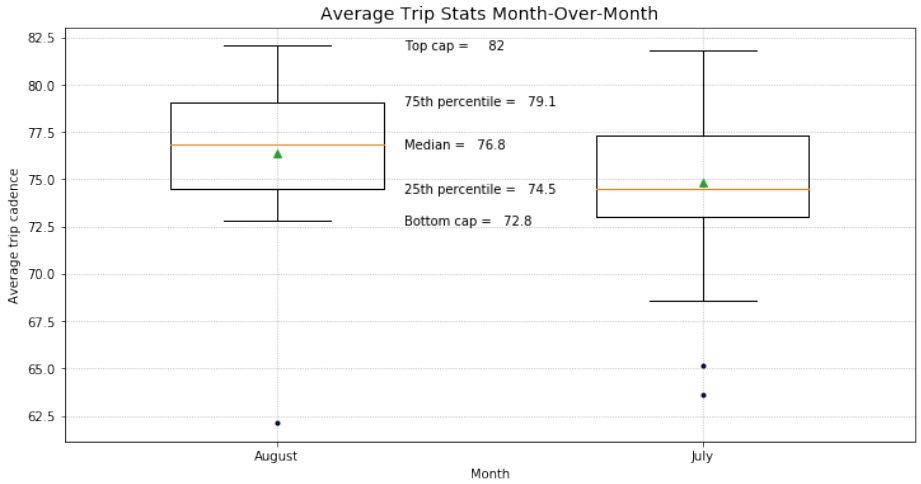
In [7]:
```
import matplotlib.pyplot as plt

def make_labels(ax, boxplot):
    # With help from https://stackoverflow.com/questions/55648729/pyth
on-how-to-print-the-box-whiskers-and-outlier-values-in-box-and-whisker
-plo

    # Retrieve relevant Line2d instances from boxplot dictionary
    iqr = boxplot['boxes'][0]
    caps = boxplot['caps']
    med = boxplot['medians'][0]

    # Get the x position of the median line
    xpos = med.get_xdata()

    # Give text a horizontal offset of some fraction of width of box
    xoff = 0.10 * (xpos[1] - xpos[0])

    # Set x position of the labels
    xlabel = xpos[1] + xoff

    # The median is the y-position of the median line
    median = med.get_ydata()[1]

    # the 25th and 75th percentiles are found from the top and bottom
(min and max) of the box
    pc25 = iqr.get_ydata().min()

    pc75 = iqr.get_ydata().max()

    # The caps give the vertical position of the ends of the whiskers
    capbottom = caps[0].get_ydata()[0]
    captop = caps[1].get_ydata()[0]

    # Make some labels on the figure using the values derived above
    ax.text(xlabel, median,
            'Median = {:6.3g}'.format(median), va='center')
    ax.text(xlabel, pc25,
            '25th percentile = {:6.3g}'.format(pc25), va='center')
    ax.text(xlabel, pc75,
            '75th percentile = {:6.3g}'.format(pc75), va='center')
    ax.text(xlabel, capbottom,
            'Bottom cap = {:6.3g}'.format(capbottom), va='center')
    ax.text(xlabel, captop,
            'Top cap = {:6.3g}'.format(captop), va='center')


def plot_stats(df1, activity='running', stat='speed'):
    plt.close()

    # Separate out by which activity we are measuring
    stat_data = df1[df1['activity'] == activity].copy()

    stat_data.set_index('month', inplace=True)
    month_list = list(stat_data.index.unique())

    # Create Datasets for each of the months
    month_data = []
    for m in month_list:
        df = stat_data[stat_data.index == m]
        if stat == 'speed':
            df = df['avg_speed']
        elif stat == 'cadence':
            df = df['avg_cadence']
```

```
        elif stat == 'altitude':
            df = df['avg_altitude']
        elif stat == 'heart_rate':
            df = df['avg_hr']
        df.reset_index(drop=True,inplace=True)
        m = df
        month_data.append(m)

    month_data = tuple(month_data)

    # Plot Boxplot
    fig = plt.figure(figsize=(12,6))
    ax = fig.add_subplot(111)
    # Set x-axis
    #ax.set_xticks(x, month_list)
    ax.set_xlabel("Month")
    # Set y-axis

    ax.set_ylabel("Average trip {}".format(stat.replace("_"," ")))
    # Set title and draw box plot
    ax.set_title("Average Trip Stats Month-Over-Month", fontsize=14)

    # Set a few other parameters
    ax.grid(True, linestyle='dotted')

    outlier_format = dict(markerfacecolor='b', marker='.')
    boxplot = plt.boxplot(month_data,
                          labels=month_list,
                          widths=.5,
                          flierprops=outlier_format,
                          showmeans=True);

    # Annotate
    make_labels(ax, boxplot)

    plt.show()
```

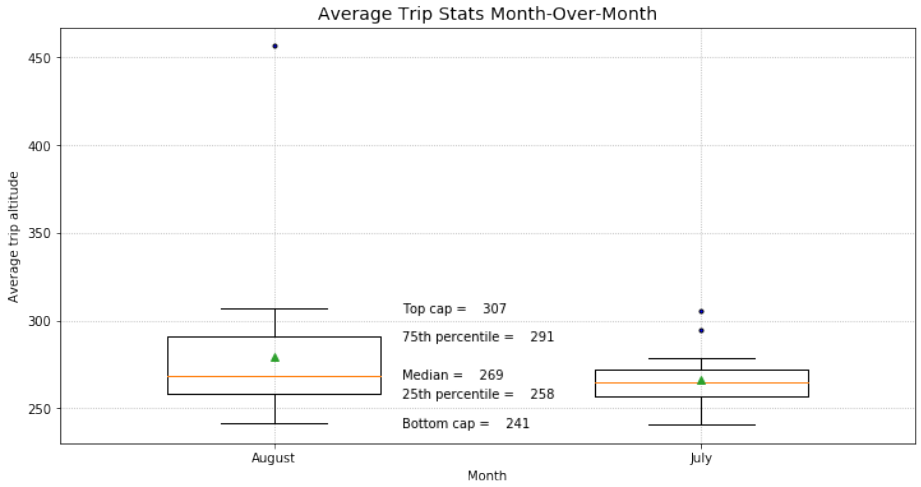
In [8]:
```
plot_stats(ind_activities,
           activity='running',  # Options: 'running', 'biking'
           stat='speed'         # Options: 'speed', 'cadence', 'altitu
de', 'heart_rate'
          )
```



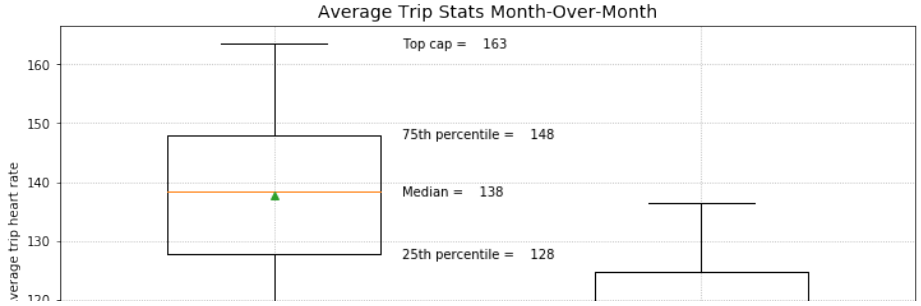
In [9]:
```
plot_stats(ind_activities,
           activity='running',  # Options: 'running', 'biking'
           stat='cadence'       # Options: 'speed', 'cadence', 'altitu
de', 'heart_rate'
          )
```

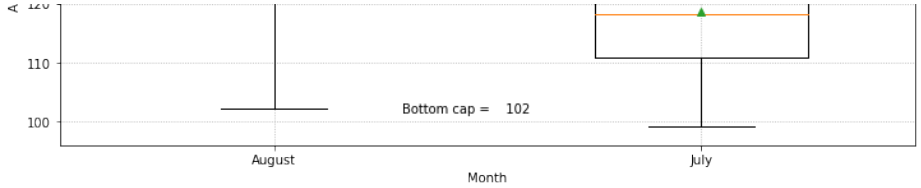


In [10]:
```
plot_stats(ind_activities,
           activity='running',  # Options: 'running', 'biking'
           stat='altitude'      # Options: 'speed', 'cadence', 'altitu
de', 'heart_rate'
          )
```



In [11]:
```
plot_stats(ind_activities,
           activity='running',  # Options: 'running', 'biking'
           stat='heart_rate'    # Options: 'speed', 'cadence', 'altitu
de', 'heart_rate'
          )
```





Analysis
What is especially interesting here is that, while the majority of the boxplots overlap month-over-month, they seem to have generally higher stats in the second month across the board. Now, one month of this is not indicative of a definted trend, but it is compelling enough that I'd be curious, had Professor Brooks run in October too, if this would have held up. The two that did not see drastic changes were altitude (which is only so controllable) and cadence, which to some degree makes sense because you can only have so many steps per minute, and with this dataset you'd actually want consolidation of the box to show less variation in the cadence (more consistently timed strides), which is what we see here!

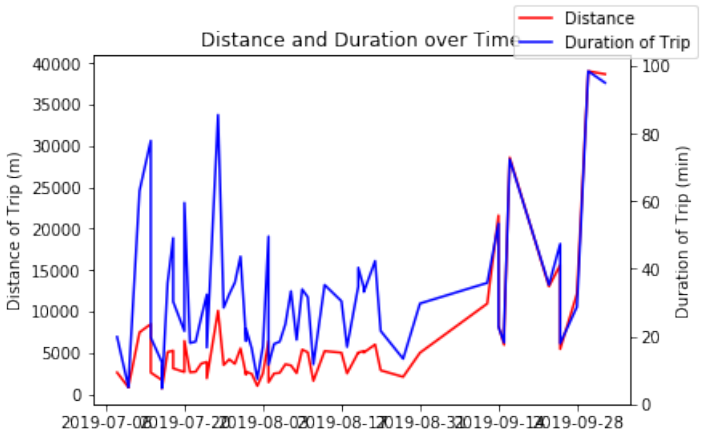
Comparing Performance Stats and Improvements

Let's look at how the distance and duration of Professor Brooks' trips changes over time to see if he is getting any more efficient at covering longer distances in shorter spans of time.

In [12]:
```
import datetime

def plot_len_dist(df, dataset='both'):

    if dataset == 'running':
        df = df[df['activity'] == 'running']
    elif dataset == 'biking':
        df = df[df['activity'] == 'biking']
    elif dataset == 'both':
        pass
    else:
        raise UnboundLocalError('Please select a valid dataset: runnin
g, biking, or both')

    df = df.set_index(df['date'])
    df.sort_values('date', inplace=True)
    df['length'] = df['length']

    df['len_mins'] = df['length'].apply(lambda x: x.total_seconds() /
60)

    # Plot the original line for the distance of the activities over t
ime.
    plt.plot(df['date'],
             df['distance'],
             "r-",
             label='Distance')

    # Create a second floating y axis
    plt.gca().twinx()

    # Plot the length of the trips
    plt.plot(df['date'],
             df['len_mins'],
             'b-',
```

```
            label='Duration of Trip')

    # Label the Plot
    plt.gcf().legend(loc='upper right') # There is nowhere that this l
ooks good and doesn't block something, I'm sorry.
    plt.gcf().axes[0].set_ylabel('Distance of Trip (m)')
    plt.gcf().axes[1].set_ylabel('Duration of Trip (min)')
    plt.title("Distance and Duration over Time")
    plt.xlabel(df['date'], rotation=45)
```

In [13]:
```
plot_len_dist(ind_activities)
```



Analysis

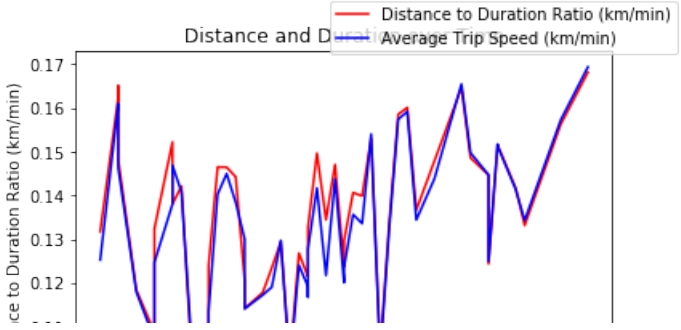
From a preliminary standpoint, this doesn't seem to show much. Since we are combining datasets, it also gets a little convoluted and difficult to compare apples to oranges. Let's see if we can create a ratio of distance over duration to see if this helps matters. We will compare it to the average speed recorded for that trip and see if there are any variations of note.

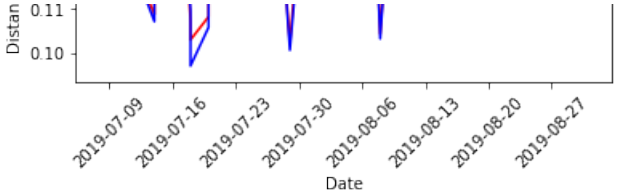
Transforming the Data for Additional Charting

To ensure that we can compare these two datasets, we will translate them both in to km/min. The distance-vs-duration ratio should be the distance divided by 1000 (to km), then divided by the length in minutes. The average speed will be multiplied by 60 (to get to minutes) and then divided by 1000.

In [14]:
```
def plot_dur_dist(df, dataset='both'):

    if dataset == 'running':
        df = df[df['activity'] == 'running']
    elif dataset == 'biking':
        df = df[df['activity'] == 'biking']
    elif dataset == 'both':
        pass
    else:
        raise UnboundLocalError('Please select a valid dataset: runnin
g, biking, or both')

    df = df.set_index(df['date'])
    df.sort_values('date', inplace=True)
    df['length'] = df['length']

    # Apply the new transformations of the data
    df['len_mins'] = df['length'].apply(lambda x: x.total_seconds() /
60)
    df['dist_dur'] = (df['distance'] / 1000) / df['len_mins']
    df['avg_km_min'] = df['avg_speed'] * 60 / 1000

    # Plot the distance vs duration ratio
    plt.plot(df['date'],
             df['dist_dur'],
             "r-",
             label='Distance to Duration Ratio (km/min)')

    # Plot the average speed recorded.
    plt.plot(df['date'],
             df['avg_km_min'],
             'b-',
             label='Average Trip Speed (km/min)')

    # Label the Plot
    plt.gcf().legend(loc='upper right') # There is nowhere that this l
ooks good and doesn't block something, I'm sorry.
    plt.gcf().axes[0].set_ylabel('Distance to Duration Ratio (km/min)'
)
    plt.xticks(rotation=45)
    plt.xlabel('Date')
    #plt.gcf().axes[1].set_ylabel('Trip Average Speed (km/min)')
    plt.title("Distance and Duration over Time")
```

In [15]:
```
########################################
# Let's plot for our running activities.#
########################################
plot_dur_dist(ind_activities, dataset='running')
```

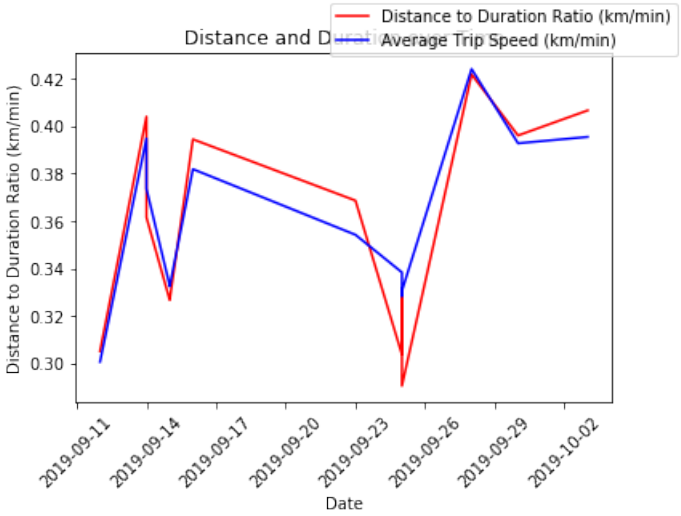




Analysis

So, what can we garner from this data? Professor Brooks' running distance-to-duration ratio is pretty varied, but we see a slowly upward moving trend. This indicates that he is gradually able to run longer distances per amount of time allotted to the travel. This indicates an small increase in running stamina. If Prof. Brooks continues to run, I think it would be interesting to see if this upward trend continues.

In [16]:
```
##########################################
# And let's plot for our biking activities.#
##########################################
plot_dur_dist(ind_activities, dataset='biking')
```



Analysis

This plot, while not an abundance of datapoints, shows some interesting data. For example, on both 9-17-2019 and 9-23-2019 we see a reasonable deviation between the Distance to Duration ratio and the Average Trip Speed. This indicates that more of the datapoints picked up by the devices may have been logged at lower speeds and may have missed some bursts. While I find it difficult to believe that there would be occurrences of extreme deviation without device failure, it is interesting to see that there may be some variation. Granted, these are both rough averages and not particularly granular at this level, so there is room

for interpretation and potential for more detailed analysis.
Further exploration of the more granular data would be interesting to see. For example, on the lower ratio days, was he stopped at stoplights/intersections more? Alternatively, comparing to some sort of psychological/physical metric, are the lower ratio days days where he felt worse? Did he eat better on higher-ratio days, or drink coffee, or have the day off of work? Some cross examination of lifestyle here would be interesting to see whether the variations here can be attributed to any particular behavior.</i>

Mapping Exploration

Let's map out what Prof. Brooks' runs have looked like over time. We will start with some basic calculations and data cleaning to ensure that the datapoints can be mapped properly.

In [17]:
```
import folium

# Let's convert the longitude and latitude data to degrees
def lat_long_deg(df):
    df["position_lat_degrees"] = df["position_lat"] * ( 180 / 2**31 )
    df["position_long_degrees"] = df["position_long"] * ( 180 / 2**31
)
    return df

df = define_activity(data, threshold=7.45)

# Call function to adjust lat/long
mapping_df = lat_long_deg(df)

# Add a date column for endpoint labels later
mapping_df['date'] = pd.to_datetime(mapping_df['timestamp']).apply(lam
bda x: x.date())

# Drop any NaN values in the coordinates, as they cause errors.
mapping_df = mapping_df.dropna(subset=['position_lat_degrees', 'positi
on_long_degrees'])
#mapping_df.head()
```

We will use the Folium library with callouts for start and end points, as well as route tracing. We will create separate route colors for each activity, and develop a function that allows for the manipulation of all these factors, as well as condensing to specific desired timeframes.

In [18]:
```
# Create a function to add callouts for start and end of each activity
.
def add_callouts(df, fol_map):

    # Create a list of the individual activities logged.
    datafiles = set()
    for file in df['datafile'].unique():
        datafiles.add(file)
    datafiles = list(datafiles)

    # Sort through the dataframe to isolate the dataset of a single lo
gged activity.
    for record in datafiles:
        df0 = df[df['datafile'] == record]
        df0.sort_values(by='timestamp')
        df0.reset_index()

        # Add starting marker
        folium.Marker([df0['position_lat_degrees'].iloc[0],
                       df0['position_long_degrees'].iloc[0]],
                      popup='Start {}'.format(df0['date'].iloc[0])
                     ).add_to(fol_map)
```

```
        # Add ending marker
        folium.Marker([df0['position_lat_degrees'].iloc[-1],
                       df0['position_long_degrees'].iloc[-1]],
                      popup='Stop {}'.format(df0['date'].iloc[-1])
                     ).add_to(fol_map)

# Create a mapped line for each activity.
def add_route(df, fol_map):

    # Create a list of the individual routes logged.
    datafiles = set()
    for file in df['datafile'].unique():
        datafiles.add(file)
    datafiles = list(datafiles)

    # Sort through the dataframe to isolate the dataset of a single lo
gged activity.
    for record in datafiles:
        df0 = df[df['datafile'] == record]
        df0.sort_values(by='timestamp')
        df0.reset_index()

        # Create route for each activity record, color coding for acti
vity type
        if df0['activity'].iloc[0] == 'running':
            route=folium.PolyLine(locations=zip(df0['position_lat_degr
ees'],
                                                df0['position_long_deg
rees']),
                                  weight=5,
                                  opacity=1,
                                  color='blue'
                                 ).add_to(m)

        elif df0['activity'].iloc[0] == 'biking':
            route=folium.PolyLine(locations=zip(df0['position_lat_degr
ees'],
                                                df0['position_long_deg
rees']),
                                  weight=5,
                                  opacity=1,
                                  color='red'
                                 ).add_to(m)
```

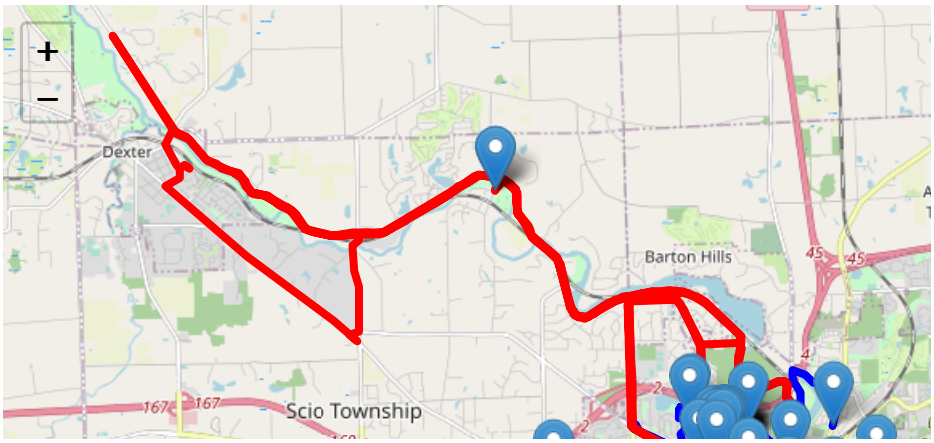
In [19]:

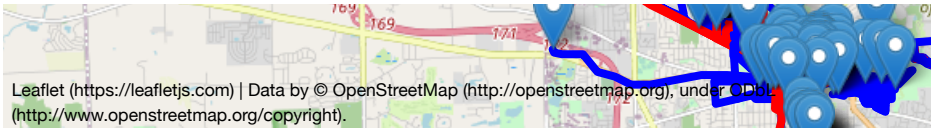
```
# Let's bring these functions together within a "create_map" function
that allows us to toggle
# the activities we are tracking, the timeframe we are looking at, and
the corresponding data we show.
def create_map(df, fol_map, endpoints='on', routes='on', start_date='2
019-01-01', end_date='2019-12-01', activity='both'):
    df = df.copy()

    # Define dataset by activity
    if activity == 'running':
        df = df[df['activity'] == 'running']
    elif activity == 'biking':
        df = df[df['activity'] == 'biking']
    elif activity == 'both':
        pass
    else:
        raise UnboundLocalError('Please select a valid dataset: runnin
```

```
g, biking, or both')

    # Define dataset by date range
    start_date = pd.Timestamp(start_date)
    end_date = pd.Timestamp(end_date)
    df['timestamp'] = pd.to_datetime(df['timestamp'])
    mask = (df['timestamp'] >= start_date) & (df['timestamp'] <= end_d
ate)
    df = df.loc[mask]

    # Decide which datapoints to show
    if endpoints == 'on':
        add_callouts(df, fol_map=m)
    else:
        pass

    if routes == 'on':
        add_route(df, fol_map=m)
    else:
        pass

    # Display chart
    display(m)
```

In [20]:

```
#####################
# Now let's map it! #
#####################

# Generate a map in the desired area
m=folium.Map(location=[42.281, -83.743],
             #tiles='Stamen Terrain',
             zoom_start=12) # Starting at Ann Arbor coordinates

create_map(mapping_df,
           fol_map=m,
           endpoints='on',           # Options: 'on' or 'off'
           routes='on',              # Options: 'on' or 'off'
           start_date='2019-07-01',
           end_date='2019-11-01',
           activity='both'           # Options: 'running', 'biking', '
both'
          )
```





Analysis

It looks like Professor Brooks sticks to similar areas. Interestingly enough, he very rarely runs on campus, but around the edges and particularly through a nearby cemetary. The center of the running seems to be centered around North Main Street and East Huron Street, with an extreme propensity to run along Catherine Street, which seems to be a heavily-traveled area up to Second Baptist Church, all the way down the Gallup Park Pathway. I find it compelling that Professor Brooks often travels in the same general areas, but rarely travels northeast or southwest. This is additionally true for biking, as he mostly stays in the same general region, with his main points being in the downtown area by campus (I am not very familiar with Ann Arbor) and up near Second Baptist Church.

I was curious what might be propelling this odd phenomenon, since there seem to be some residential areas to the west and southwest that would be ideal for exercising, and thought potentially the untraveled areas were less safe, or something to that extent. I ended up comparing my rendering to a crime rates map of Ann Arbor, and found that Professor Brooks is actually running and cycling in what seem to be some of the more dangerous areas of the city. If he were to explore a little bit more of Ann Arbor on his runs, I'd recommend branching out to some other areas of town that may even be a little safer!</i>

Crime Rates Map Source: https://www.neighborhoodscout.com/mi/ann-arbor/crime

Rule et al's Heuristics for Computational Narrative Analysis

Rule 2: *Document the process*

I feel this rule was adhered to quite well. The point o fthis is to "document all your explorations, even (or perhaps especially) those that lead to dead ends," for example the first splom plot I had made. The original intent was to find if there were any relationships further into which we could delve to analyze, but really nothing seemed to turn up. Similarly, I left in the original line chart under "Comparing Performance Stats and Improvement" even though it showed very little in order to justify my modification of the data in the following cells.

Additionally, I tried to document any edits I made, or why I left something out. For example, in the first chart, the splom, I tried to put in a regression line to see if anything could be made from it. The line was even further distracting than the chaotic charts themselves and did not provide any additional information, so it was commented out. However, I left the notes in, because should more data be added to this dataset, it is conceivable that it could be useful at that time.

Rule 3: *Use cell divisions to make steps clear*

I feel this rule was pretty adequately adhered to. I did my best to separate the function creation from call, and make clear what the end goal of each cell was. I broke up some of the longer codes, such as in the mapping functions, so that no cell ran for too long. This also helped clarify what each cell was doing.

My knowledge of connecting notebooks and creating libraries is relatively limited at this juncture. I feel this particular project is bordering on length and complexity to where I should create multiple notebooks. In future assignments, I hope to better adhere to this particular aspect of the rule with an increased knowledge base.

Rule 4: *Modularize code*

I think this rule was fairly well adhered to in this particular demonstration. I did my best to create a function for nearly every major process that could be edited, and any subsequent calls of the plots would be changed accordingly. This is especially evident in the Box Plot demonstration under "Consolidating for Readability" where changes to the module change 4 of the following plots. This was great, as the annotations were added later, and it did not involve redundant edits to each plot. I also make a point to create arguments within each module to edit which datasets were being explored, what parameters were desired for the outputs, etc. so that they could retain functionality in a multitude of settings, potentially outside of the notebook.

Personal Self-Critiques

- *"Biking" should be "Cycling," sorry if I offended any hardcore cyclists out there. I put biking too early and did not have time to go back and change everything.*
- *I should have done some smoothing of speeds, etc. and think my analysis could have been far more robust had I done so, but was having trouble with getting a df.resample("#min").apply(np.mean) to work properly and decided to move on. This, unfortunately, meant all the data following was in it's original form instead of some sort of transformed set. I found this frustrating, but was running out of time and patience to find alternative solutions and had to make do. This is something I would like to continue to practice outside of homework assignments.*
- *Could not get box plots to annotate on the other box plots. Tried variations of incorporating multiple subplots, etc., but realized probably too late that I should have built it differently from the top to not zip everything as a tuple, as that made it difficult to isolate the datasets for extraction. There may be a way*

to wrap it into a for-loop to be dynamic for each dataset, but I did not have time to troubleshoot this. Again, I will probably mess around with this after this assignment and try to solve it, if for no other reason but pride.

- *I realize the first visualization was kind of a bust. I spent quite a bit of time before that trying to define "running" vs "biking", including unsuccessfully smoothing the speeds, and was thus frustrated by the time I got to the dataset, and upon having realized it showed little of value was quite discouraged. I think the premise was sound, as there could have potentially been some sort of relationship with some of these sets that could have used further exploration. I actually found it more odd that there was little to none. I contemplated going back and trying to figure out if the dataset had anything else to glean using the by-activity data.*
- *My narrative could be better. I know this. I honestly got tripped up when I tried to overclean data and then the first graph was pretty uneventful, and it set the pace for a clunky remainder. I'm not super into fitness and know nothing about running and cycling, so I had trouble turning this into something of a story/sales pitch/etc. to discuss and convey it to a "client", which I understand is the entire point of this project, I just really had trouble connecting to it. I understand the point of a narrative, and think my Assignment 2 was a better example of my understanding than this one was.*