A4 Part 3 - Advanced

April 2, 2025

1 Part 3: A Demonstration of a More Advanced Visualization Technique

1.1 Discussion on Rule et al.'s Rules

1. **Rule 1**: Tell a story for an audience
   - From the outset, I aimed to craft a narrative tailored to Professor Brooks, my client, transforming raw Strava data into insights he could use—whether for cardio gains or workout tweaks. My simple plots (line, scatter, violin) in Part 2 built a foundation: a typical workout arc, rising intensity over months, and HR zone shifts —- all narrated with specific commentary to keep the project engaging and personal. The advanced ridgeline plot, with its blue-to-orange gradient and heart rate zones, evolved this story, revealing monthly Cycling patterns (e.g., August's Aerobic/Cardio peak). Initially, I assumed a Running-Cycling mix, but debugging showed all 59 datafiles were Cycling —- a twist I discussed frankly in Part 3. This pivot, documented in my "Last Remarks" and "So what did we learn?" sections, reflects Rule 1: I adapted the story to the data's truth, keeping Brooks as the audience, ensuring he'd find value in his cycling abilities and next steps.
2. **Rule 2**: Document the process, not just the results

- This project wasn't a straight shot to polished plots by any means -— it was a messy and filled with debugging, and I documented my stumbles to honor Rule 2. In Part 2, I found myself disappointed with the histogram and continued on in an effort to make a better, more informative chart. In Part 3, I used enhanced_speed to split Running and Cycling, but the cadence column's arrival flipped that assumption. My reflection—"I questioned myself quite a lot, especially my underlying assumptions"—lives in the "So what did we learn?" section, admitting I should've dug into cadence in Part 1. This isn't just a "here's the plot" story; it's a log of how I wrestled with NaNs, missteps, and revelations, giving a transparent view of the process behind the final Cycling-only insight.

3. **Rule 3**: Use cell divisions to make steps clear
4. **Rule 8**: Share and explain your data

- Rule 8 guided me to make the data's quirks and limits clear, not just its outputs. I started with the file_path, timestamp conversions, and filters —-laying out the dataset's scope. I printed NaN percentages, debugs, stats on the data to expose its intricacies. I sought to explained these in context and tied them to visuals like the ridgeline's zones (106–126 bpm Fat-Burning). This transparency would allow Professor Brooks (or any reader) rerun my code, see the same output, and understand how we achieved the results we did. It's not a simple "trust me" exercise, but rather "here's the data, and here's why."

5. **Rule 9**: Design your notebooks to be read, run, and explored

1.2 Explanation of Rule et al.

1.3 A Comparison of Running & Cycling

One aspect of the data that we have yet to explore is the type of exercise Professor Brooks was doing throughout the summer. - Deepti Wilson (Tuesday at 10:41 PM) stated that, "The only type of recorded exercises are biking and running."

With this in mind, it would be interesting to see any discrepancies in Professor Brooks' heart rate with respect to the type of exercise. For this graph, I will be working under the assumption that each datafile (i.e., each individually recorded workout) will be *either* running *or* cycling. The reason being that it would be somewhat strange for one to bike, leave their bike somewhere, and run away from their bike (unless, perhaps, someone was training for a triathlon?

If we were to work under the mutually exclusive assumption, then we could categorize individual workouts (i.e., datafiles) as running or cycling by calculating the **maximum** `enhanced_speed` of a datafile. The reason we would use the `enhanced_speed` column over the `speed` column is simply due to the significant lack of NaN values. - As seen in Part 1, `speed` had an NaN percentage of approximately 63.28%, while `enhanced_speed` had an NaN percentage of approximately 0.025%. To clarify the difference between the two, I attended office hours with Erik Lang on March 29, 2025, where we discussed possible differences. - `enhanced_speed` typically refers to a more precise measurement of one's speed compared to basic speed measurements (i.e., `speed`). This is because `enhanced_speed` takes into account GPS data, accelerometer data, elevation changes, and even heart rate.

After some research, it appears that, for an average person: - a typical running speed is around 5-8 mph (~ 2.2-3.8 m/s) - a typical cycling speed for beginners starts around 10-15 mph (~ 4.47-6.7 m/s) - We will also be making the assumption that, because Professor Brooks is cycling for exercise, that the bicycle will not be powered electrically [thereby increasing cycling speed]

However, there can be a lot of overlap between the two speeds, so it becomes crucial to find a speed in which it is nearly impossible to run, but easily attainable for cyclists. With some research, I believe this number to be approximately 13 mph. - Elite marathon runners can average speeds around 13 mph long-distance, but this is very difficult to achieve for the average person. - 13 mph is very easily attainable cycling

We were given information in the assignment that the units are in m/s, thus our cut-off will be 5.81152 m/s.

1.3.1 Import Libraries & Load Data

[1]:
```
import os
import numpy as np
import pandas as pd
import plotly.express as px
import plotly.graph_objects as go
from plotly.colors import n_colors   # We can use this so that we can create␣
 ↪discrete colors for a given range!
```

```
file_path = os.path.realpath("/home/jovyan/work/resources/course_assignments/
 ↪assets/strava.csv")
df = pd.read_csv(file_path)

df['timestamp'] = pd.to_datetime(df['timestamp'])
df['timestamp'] = pd.to_datetime(df['timestamp'], utc=True)
df['timestamp'] = df['timestamp'].dt.tz_convert('America/Detroit')

df = df.dropna(subset=['heart_rate'])
```

1.3.2 A New Type of Chart

As I read up on violin plots on `Plotly`, I came across a type of graph I'd never seen before called a **"Ridgeline Plot"** [https://plotly.com/python/violin/]. Ridgeline plots are essentially violin plots but flat on one side, and resembling a ridge or mound. They're useful in showing the distribution of several groups (e.g., across time, salaries, test scores). They're also compact – they can "stack" distributions, which allow them to partially overlap when there are numerous groups.

Because we didn't analyze the differences between running and cycling in our work with the simple charts, it would be interesting to see whether the distributions of the professor's heart rate

[2]:
```
# Define the speed threshold for running and cycling (5.81152 m/s threshold)
df['activity_type'] = df.groupby('datafile')['enhanced_speed'].transform('max').
 ↪apply(
    lambda x: 'Running' if x < 5.81152 else 'Cycling')     # We can replace 5.
 ↪81152 with other values we see fit, such as 4

# Extract Month
df['month'] = df['timestamp'].dt.strftime('%B')
months = ['July', 'August', 'September', 'October']

# Define Activity Types
activities = ['Running', 'Cycling']

# Generate 8 traces (4 months × 2 activities)
labels = [f"{month} {activity}" for month in months for activity in activities]

# Create Colors from `plotly.colors` package (8 distinct shades from blue to␣
 ↪orange)
colors = n_colors('rgb(0, 120, 180)', 'rgb(200, 100, 0)', 8, colortype='rgb')

# Create Figure
fig = go.Figure()

# Loop to add Ridgeline Plots for each month and activity
for i, label in enumerate(labels):
```

```
    month, activity = label.split(' ', 1)
    hr_data = df[(df['month'] == month) & (df['activity_type'] ==␣
 ↪activity)]['heart_rate'].dropna()
    if len(hr_data) > 10:               # Ensure enough data for a ridgeline
        fig.add_trace(go.Violin(
            x=hr_data,                  # HR on x-axis
            y=[label] * len(hr_data),   # Label on y-axis
            line_color=colors[i],       # Distinct color per trace
            orientation='h',            # Horizontal
            side='positive',            # Extend rightward
            width=1.5,                  # Narrower width for clarity
            points=False,               # No individual points
            box_visible=True,           # Show quartiles
            meanline_visible=True       # Show mean
        ))

# Update layout
fig.update_layout(
    title="Heart Rate Distributions by Activity (July-October 2019)",
    xaxis_title="Heart Rate (bpm)",
    yaxis_title="Month & Activity",
    xaxis_showgrid=False,
    xaxis_zeroline=False,
    yaxis=dict(
        tickmode='array',
        tickvals=labels,
        ticktext=labels
    ),
    showlegend=False,                   # No legend needed with y-axis labels
    plot_bgcolor='white',
    paper_bgcolor='white',
    height=600                          # Taller for readability
)

fig.show()
```

1.3.3 Well that's interesting!

Given that we have 4 months and 2 types of exercises, we would expect 8 "ridges" (or 7, given the one workout in October). But it's clear from the graph above that the professor was only running and cycling during the month of July. During the other months, it appears he was either **only** running *or* cycling.

However, it's important to keep in mind that my assumption of using 13 mph (or 5.81152 m/s) as a "cutoff" between running and cycling may very well be an incorrect one. There is a possibility that the professor may have been cycling at low speeds during this activity.

1.3.4 AHA!

It is at this moment that I realized that there is a column titled `cadence`, which is likely from the "Garmin Cadence Sensor" mentioned in the assignment instructions. I had to Google this device to understand what it does; and it is specific to bikes. This means that, as long as there is `cadence` data (perhaps something that is non-zero and non-NaN), the data pertains to cycling!

Therefore, I do not need to categorize the data based on `enhanced_speed`, but rather the presence of `cadence`!

[3]:
```
## Debugging Cadence: I ran into some issues because ALL my ridges became␣
 ↪cycling! More on this in the discussion below
print("Rows with cadence > 0:", len(df[df['cadence'] > 0]))
print("Datafiles with any non-zero, non-NaN cadence:", df[df['cadence'] >␣
 ↪0]['datafile'].nunique())
print("Datafiles with all zero/NaN cadence:", df.groupby('datafile')['cadence'].
 ↪max().eq(0).sum())

# Categorize: Cycling if any non-zero, non-NaN cadence; else Running
has_cadence = df.groupby('datafile')['cadence'].max().apply(
    lambda x: 'Cycling' if pd.notna(x) and x > 0 else 'Running')
df['activity_type'] = df['datafile'].map(has_cadence)
print("Activity split:", df['activity_type'].value_counts())

# Extract Month
df['month'] = df['timestamp'].dt.strftime('%B')
months = ['July', 'August', 'September', 'October']

# Define Activity Types
activities = ['Running', 'Cycling']

# Generate 8 traces
labels = [f"{month} {activity}" for month in months for activity in activities]

# Create Colors using n_colors for discrete values in a given range
colors = n_colors('rgb(0, 120, 180)', 'rgb(200, 100, 0)', 8, colortype='rgb')

# Create figure
fig = go.Figure()

# Loop to Create Ridgeline Plots
for i, label in enumerate(labels):
    month, activity = label.split(' ', 1)
    hr_data = df[(df['month'] == month) & (df['activity_type'] ==␣
 ↪activity)]['heart_rate'].dropna()
    if len(hr_data) > 10:
        fig.add_trace(go.Violin(
            x=hr_data,
```

```
            y=[label] * len(hr_data),
            line_color=colors[i],
            orientation='h',
            side='positive',
            width=1.5,
            points=False,
            box_visible=True,
            meanline_visible=True
        ))

# Define HR Zones
zones = [
    (0, 90, "Resting", "cornflowerblue"),
    (90, 106, "Lower-Intensity", "green"),
    (106, 126, "Fat-Burning", "gold"),
    (126, 180, "Aerobic/Cardio", "red")
]

# Add HR Zones
for lower, upper, label, color in zones:
    fig.add_vrect(
        x0=lower, x1=upper,
        fillcolor=color,
        opacity=0.15,
        layer="below",
        line_width=0
    )

# Update Layout
fig.update_layout(
    title="Heart Rate Distributions by Activity (July-October 2019)",
    xaxis_title="Heart Rate (bpm)",
    yaxis_title="Month & Activity",
    xaxis_showgrid=False,
    xaxis_zeroline=False,
    yaxis=dict(
        tickmode='array',
        tickvals=labels,
        ticktext=labels
    ),
    showlegend=False,
    plot_bgcolor='white',
    paper_bgcolor='white',
    height=600,
    xaxis_range=[0, 200]
)
```

```
fig.show()
```

```
Rows with cadence > 0: 37158
Datafiles with any non-zero, non-NaN cadence: 59
Datafiles with all zero/NaN cadence: 0
Activity split: activity_type
Cycling    38355
Name: count, dtype: int64
```

1.4 WHEW!

I realized, after much effort, that my attempts to generate 8 ridges may have been based on an incorrect assumption – that perhaps all of the 59 datafiles I have been working with were **all** cycling data. I skimmed through the Strava data and, unless Professor Brooks had strapped the Garmin device on himself while he ran (highly unlikely), it appears that the data indicates that all the heart rate data we have is cycling data!

This being the case, I found myself somewhat disappointed not being able to make such a comparison for heart rate zones and activity type.

Before realizing this, I had actually also created a template faceted violin plot in preparation for a comparison between the two. I also wanted to add the "points" in the violin plot as a cool way to see the HR data. (As it turns out, because HR is recorded as discrete values, we end up with these cool "bands" of dots.)

[4]:
```
# Create Ridgeline Plot using Plotly Express
fig = px.violin(df,
                x="heart_rate",
                color="activity_type",
                box=True,
                points="all",
                #hover_data=["datafile"],
                facet_col="activity_type",  # Create small multiples for␣
 ↪Running and Cycling, which does not apply now!
                title="Heart Rate Distribution by Activity Type")

# Customize Layout
fig.update_layout(
    violingap=0.2,                          # Gap between Violins
    xaxis_title="Heart Rate (bpm)",
    yaxis_title="Density",
    title="Heart Rate Distribution by Activity Type",
    showlegend=False                        # if facet_col used, no legend is␣
 ↪necessary; but this does not apply now either!
)

fig.show()
```

1.5 So what did we learn from these more complex plots?

In one way, it was a *very* trying process because it required a lot of self-reflection and self-evaluation along the way. I questioned myself quite a lot, especially my underlying assumptions—like whether enhanced_speed or cadence could cleanly split Running and Cycling. Each debug step forced me to rethink: why was I categorizing this way? Was the data lying to me? It helped me better understand not just what I was doing, but *why* and *how*. In hindsight, I learned I should've tackled this deeper dive earlier, perhaps in Part 1, instead of chasing a "big picture" at the end. Starting small and iterating would've saved some late-night head-scratching!

In terms of using this information to generate useful information to the professor: 1. **Cycling-Only Revelation**: Discovering that all the HR data from July–October 2019 reflects Cycling workouts reframes our simple charts from Part 2. The line graph's warmup-to-surge profile, the scatter plot's 120–145 bpm rise with an August spike, and the violin plots' monthly zone shifts (July in Fat-Burning, August in Aerobic/Cardio)—these all describe your cycling prowess, not a mix of activities. This sharpens their relevance: you're a cycling master! But this is a double-edged sword: these insights don't touch running, leaving a gap in understanding your full fitness picture. 2. **Data Expansion Needed**: To dig deeper into your workouts and cardio improvement, we'd need more measurements. The Garmin cadence sensor nailed your cycling, but Running data is missing—perhaps untracked or lost in Strava's merge. Adding run-specific metrics (e.g., via a watch or Garmin Foot Pod for step cadence) would let us compare effort across activities. 3. **Cycling Strengths & Next Steps**: With cycling as your go-to workout, the ridgeline plot with heart rate zones highlights your cycling capabilities: July's steady Fat-Burning rides (106–126 bpm) built endurance; August's Aerobic/Cardio peaks (140+ bpm) pushed intensity (likely heat-driven); September eased off, and October came roaring back.

1.6 Just For Fun

Below is a map of Professor Brooks' workout paths that I've traced around Ann Arbor using his longitude and latitude data. I did use the `activity_type` data from earlier, in that we could update this data in the future and, if there is running data, the map below would reflect this! Anyway, I just thought this would be a cool interactive visual I could add to this project that would provide another dimension to our study – it just kind of ties it altogether :)

I'm an Ann Arbor native (my family and I immigrated here in 1999), so it was fun moving the slider around and seeing where the professor's paths took him (e.g., near Gallup Park/Huron High (my alma mater), or all the way south near Briarwood Mall). Anyways, I had a lot of fun making this project. I hope you enjoyed reading it!

[5]:
```
# Converts Semicircle Lat/Long to Degrees, only if needed! (Can re-run cell␣
 ↪without worrying about re-calculating again)
if df['position_lat'].abs().max() > 90:
    df['position_lat'] = df['position_lat'] * 180 / 2**31
    df['position_long'] = df['position_long'] * 180 / 2**31

# Ensure Valid Lat/Long Values
df_filtered = df.dropna(subset=['position_lat', 'position_long'])
df_filtered = df_filtered[(df_filtered['position_lat'].between(-90, 90)) &
                          (df_filtered['position_long'].between(-180, 180))]
```

```
# Extract Date from Timestamp
df_filtered['date'] = df_filtered['timestamp'].dt.date.astype(str)   # For use␣
 ↪on slider

# Get Activity Types for Legend
activity_types = df_filtered['activity_type'].unique()

# Create Map with Animation!
fig = px.scatter_mapbox(df_filtered,
                        lat="position_lat",
                        lon="position_long",
                        color="activity_type",                       # This␣
 ↪would have changed color if we had running data!
                        hover_data=["date", "enhanced_speed"],
                        animation_frame="date",                      # Uses the␣
 ↪date instead of datafile for ease of understanding
                        title="Professor Brooks' Workouts: Running & Cycling␣
 ↪(but really just cycling)",
                        zoom=12,
                        height=600,
                        category_orders={"activity_type":␣
 ↪sorted(activity_types)}
)

# Update Layout
fig.update_layout(
    mapbox_style="carto-positron",
    mapbox_center={"lat": df_filtered["position_lat"].median(), "lon":␣
 ↪df_filtered["position_long"].median()},
    margin={"r":0, "t":50, "l":0, "b":0}
)

# Update Marker Size
fig.update_traces(marker=dict(size=7))

fig.show()
```

[ ]: