A4 Part 1 - Data Prep

April 2, 2025

1 Assignment 4: A Computational Narrative with Strava Data

2 Part 1: Preparing the Data

2.1 Rule et al.'s Rules (Discussion in Part 3)

1. **Rule 2**: Document the process, not just the results
2. **Rule 3**: Use cell divisions to make steps clear
3. **Rule 8**: Share and explain your data
4. **Rule 9**: Design your notebooks to be read, run, and explored

1. (20%) Are you making a compelling computational narrative, judged in part by **Rule et al's ten rules for computational analyses**?
   - You don't need to follow all of the rules all of the time, but you must **explicitly indicate at the header** of each notebook which rules you adhered to and what the evidence was.
   - While there are no hard limits on the number of rules you should address, I expect **at least three rules** would be able to be discussed for a notebook of this size, and that discussion and evidence of how you aligned with those rules would be on the order of **1-2 paragraphs per rule**.
2. (35%) Have you demonstrated that you have a solid grasp of **at least three** of the basic visual analysis techniques in this class **(scatter, box, line, violin, histograms, heatmaps, probability plots, treemaps, sploms)** and that they were appropriate for the analysis/data you were investigating?

   You get equal grades for **each plot type (15% each: total 45% for 3?)**, and grades for a given plot will be broken down into three equal categories (5% each): [Strava]

   - i. The mechanics of generating a reasonable plot from the data you are working with.
   - ii. The justification for the plot and the insight as a result, as described by your computational narrative.
   - iii. Making the plot rock visually, by embedding advanced features ranging from the aesthetic (color, form) to the informational (callouts, annotations).
3. (15%) Have you demonstrated that you have a solid grasp of at l east one of the more advanced visual analysis techniques in this class?

   (Don't use any visualizations listed as basic plots above. You can explore a new visualization technique which the lecture didn't teach you, or you can even come up with a combination of multiple types of plots to generate an advanced plot) and that it was appropriate for the analysis/data you were investigating?

The grading rubric is the same as the basic plots. You may use other advanced plots with permission in this category (ask first to ensure they seem reasonably advanced).

4. (20%) Are you able to provide an interesting and defensible analysis?

   For the both data, your visualization should help Professor Brooks understand what this data **means**. If your data science discovery will make the client happy then this part of the overall grade tilts up towards 20%. If there are obvious things you should have looked at then it tilts down towards 0%.

5. (10%) Are the final results displayed in a dashboard that tells your story?

2.1.1 Additional Notes

The data are in a variety of different units. A previous student noted the following units: | Data | Units | | ——— | ——— | | Cadence | rpm | | Ground Time | milliseconds | | Vertical Oscillation | centimeters | | Distance, Altitude, & Enhanced Altitude | meters | | Longitude & Latitude | semicircles (radians) | | Air & Form Power | watts | | Leg Spring Stiffness | kN/m | | Speed | m/s |

3 Preparing the Data

3.0.1 Importing Libraries

I will be importing the libraries I expect I will need. I may import more later on in the assignment.

[2]:
```
import os
import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
import seaborn as sns
```

3.0.2 Uploading Strava Data File

The data I used for this assignment was readily available in our course materials on Jupyter. The data was provided to us with permission from Professor Brooks with the understanding that we would honor his privacy and not distribute it without his permission.

[3]:
```
file_path = os.path.realpath("/home/jovyan/work/resources/course_assignments/
 ↪assets/strava.csv")
df = pd.read_csv(file_path)
df.head()
#df.shape (40649, 22)
```

[3]:
```
   Air Power  Cadence  Form Power  Ground Time  Leg Spring Stiffness  Power  \
0        NaN      NaN         NaN          NaN                   NaN    NaN
1        NaN      NaN         NaN          NaN                   NaN    NaN
```

```
2        NaN       NaN          NaN           NaN                      NaN     NaN
3        NaN       NaN          NaN           NaN                      NaN     NaN
4        NaN       NaN          NaN           NaN                      NaN     NaN

   Vertical Oscillation  altitude  cadence                     datafile  …  \
0                   NaN       NaN      0.0  activities/2675855419.fit.gz  …
1                   NaN       NaN      0.0  activities/2675855419.fit.gz  …
2                   NaN       NaN     54.0  activities/2675855419.fit.gz  …
3                   NaN    3747.0     77.0  activities/2675855419.fit.gz  …
4                   NaN    3798.0     77.0  activities/2675855419.fit.gz  …

   enhanced_speed  fractional_cadence  heart_rate  position_lat  \
0           0.000                 0.0        68.0           NaN
1           0.000                 0.0        68.0           NaN
2           1.316                 0.0        71.0           NaN
3           1.866                 0.0        77.0   504432050.0
4           1.894                 0.0        80.0   504432492.0

   position_long   speed           timestamp  unknown_87 unknown_88  \
0            NaN     0.0  2019-07-08 21:04:03         0.0      300.0
1            NaN     0.0  2019-07-08 21:04:04         0.0      300.0
2            NaN  1316.0  2019-07-08 21:04:07         0.0      300.0
3   -999063637.0  1866.0  2019-07-08 21:04:14         0.0      100.0
4   -999064534.0  1894.0  2019-07-08 21:04:15         0.0      100.0

   unknown_90
0         NaN
1         NaN
2         NaN
3         NaN
4         NaN

[5 rows x 22 columns]
```

3.0.3 Gaining an Understanding of the Data

[4]:
```
df.columns
```

[4]:
```
Index(['Air Power', 'Cadence', 'Form Power', 'Ground Time',
       'Leg Spring Stiffness', 'Power', 'Vertical Oscillation', 'altitude',
       'cadence', 'datafile', 'distance', 'enhanced_altitude',
       'enhanced_speed', 'fractional_cadence', 'heart_rate', 'position_lat',
       'position_long', 'speed', 'timestamp', 'unknown_87', 'unknown_88',
       'unknown_90'],
      dtype='object')
```

[5]:

```python
# Let's see what types of values of in each column
# I interchanged the "column" parameter to get an idea of each column

def get_unique_values(df, column):
    unique_values = {}
    unique_values[column] = df[column].unique().tolist()
    return unique_values

get_unique_values(df, 'unknown_88')

# The code below calculates how many unique values are in the column
# len(get_unique_values(df, 'unknown_88')['unknown_88'])
```

[5]:

```
{'unknown_88': [300.0, 100.0, nan]}
```

[6]:

```python
nan_percentage = (df.isnull().sum() / len(df)) * 100
nan_percentage
```

[6]:

```
Air Power               56.107161
Cadence                 56.094861
Form Power              56.107161
Ground Time             56.094861
Leg Spring Stiffness    56.107161
Power                   56.094861
Vertical Oscillation    56.094861
altitude                63.332431
cadence                  0.054122
datafile                 0.000000
distance                 0.000000
enhanced_altitude        0.125464
enhanced_speed           0.024601
fractional_cadence       0.054122
heart_rate               5.643435
position_lat             0.472336
position_long            0.472336
speed                   63.275849
timestamp                0.000000
unknown_87               0.054122
unknown_88               5.643435
unknown_90              54.198135
dtype: float64
```

There are a LOT of `NaN`s! In fact, some columns (e.g., `'Air Power'`, `'speed'`) have more `NaN` values than actual values. Because of the huge variance across the data types, I think that the best way to deal with this would be to create charts that take this into account. Thus, instead of dropping all NaNs (which would clear out a substantial amount of data), I'd like to keep as much of it as possible, and only drop the necessary pieces of data per chart.

This means that I will be creating each chart and dropping or replacing values **dynamically** based on what I believe best suits the needs of the visualization and also best represents the data.

[7]:
```
print(df.dtypes)
```

```
Air Power                 float64
Cadence                   float64
Form Power                float64
Ground Time               float64
Leg Spring Stiffness      float64
Power                     float64
Vertical Oscillation      float64
altitude                  float64
cadence                   float64
datafile                   object
distance                  float64
enhanced_altitude         float64
enhanced_speed            float64
fractional_cadence        float64
heart_rate                float64
position_lat              float64
position_long             float64
speed                     float64
timestamp                  object
unknown_87                float64
unknown_88                float64
unknown_90                float64
dtype: object
```

3.0.4 Type Conversions

As shown above, `timestamp` is currently an `object` dtype. I will like to convert the `timestamp` entries to the correct datetime format.

[8]:
```
df['timestamp'] = pd.to_datetime(df['timestamp'])
print(df['timestamp'].dtype)
```

```
datetime64[ns]
```

[9]:
```
df['hour'] = df['timestamp'].dt.hour
unique_hours = df['hour'].unique()
print(sorted(unique_hours))
```

```
[0, 1, 2, 3, 11, 12, 14, 15, 17, 18, 19, 20, 21, 22, 23]
```

3.0.5 Considering the Correct Time Zone

Looking at the unique hours above, it appears that the professor's workouts typically take place: - Around noon - In the evenings - Around midnight - Sometimes 2-3 in the morning?!

While this might be true, let's consider if an alternate theory would give us more reasonable data. In other words, could the fitness tracker be recording the professor's workouts using UTC times, even though the professor might not live in a UTC+00:00 time zone?

Let's dive in.

A quick Google search turns up the professor's office on the University of Michigan's Ann Arbor campus at the School of Information [https://www.si.umich.edu/people/christopher-brooks]. Thus, let us work under the assumption that it is highly likely that Professor Brooks lives in the Ann Arbor area (or, at the very least, in the Eastern Time Zone). This means that the times should be converted into **Eastern Time**.

EDT is the time zone used in summer and spring, while EST is the time zone used in winter and autumn. - In 2019, "Spring Forward" occurred on March 10, 2019 - "Fall Back" occurred on November 3, 2019

[10]:
```
df['month'] = df['timestamp'].dt.month
unique_months = df['month'].unique()
print(sorted(unique_months))
```

```
[7, 8, 9, 10]
```

Luckily for us, the data takes place across July through October 2019, so we do not need to take Daylight Savings into account.

So let's continue trying to convert our data from UTC to EDT. A quick Google search turns up that EDT is **UTC-04:00**.

[11]:
```
import pytz         # Time Zone Package: https://pypi.org/project/pytz/
print(type(pytz.all_timezones))  # Used pytz.all_timezones; but it was a very␣
 ↪large list
                                 # Noticed that time zones were categorized by␣
 ↪continents
                                 # I used this to find the an Eastern Time Zone␣
 ↪with a city close to Ann Arbor

for timezone in pytz.all_timezones:
    if "America" in timezone:
        print(timezone)
```

```
<class 'pytz.lazy.LazyList.__new__.<locals>.LazyList'>
America/Adak
America/Anchorage
America/Anguilla
America/Antigua
America/Araguaina
America/Argentina/Buenos_Aires
America/Argentina/Catamarca
America/Argentina/ComodRivadavia
America/Argentina/Cordoba
America/Argentina/Jujuy
```

```
America/Argentina/La_Rioja
America/Argentina/Mendoza
America/Argentina/Rio_Gallegos
America/Argentina/Salta
America/Argentina/San_Juan
America/Argentina/San_Luis
America/Argentina/Tucuman
America/Argentina/Ushuaia
America/Aruba
America/Asuncion
America/Atikokan
America/Atka
America/Bahia
America/Bahia_Banderas
America/Barbados
America/Belem
America/Belize
America/Blanc-Sablon
America/Boa_Vista
America/Bogota
America/Boise
America/Buenos_Aires
America/Cambridge_Bay
America/Campo_Grande
America/Cancun
America/Caracas
America/Catamarca
America/Cayenne
America/Cayman
America/Chicago
America/Chihuahua
America/Ciudad_Juarez
America/Coral_Harbour
America/Cordoba
America/Costa_Rica
America/Creston
America/Cuiaba
America/Curacao
America/Danmarkshavn
America/Dawson
America/Dawson_Creek
America/Denver
America/Detroit
America/Dominica
America/Edmonton
America/Eirunepe
America/El_Salvador
America/Ensenada
```

```
America/Fort_Nelson
America/Fort_Wayne
America/Fortaleza
America/Glace_Bay
America/Godthab
America/Goose_Bay
America/Grand_Turk
America/Grenada
America/Guadeloupe
America/Guatemala
America/Guayaquil
America/Guyana
America/Halifax
America/Havana
America/Hermosillo
America/Indiana/Indianapolis
America/Indiana/Knox
America/Indiana/Marengo
America/Indiana/Petersburg
America/Indiana/Tell_City
America/Indiana/Vevay
America/Indiana/Vincennes
America/Indiana/Winamac
America/Indianapolis
America/Inuvik
America/Iqaluit
America/Jamaica
America/Jujuy
America/Juneau
America/Kentucky/Louisville
America/Kentucky/Monticello
America/Knox_IN
America/Kralendijk
America/La_Paz
America/Lima
America/Los_Angeles
America/Louisville
America/Lower_Princes
America/Maceio
America/Managua
America/Manaus
America/Marigot
America/Martinique
America/Matamoros
America/Mazatlan
America/Mendoza
America/Menominee
America/Merida
```

America/Metlakatla
America/Mexico_City
America/Miquelon
America/Moncton
America/Monterrey
America/Montevideo
America/Montreal
America/Montserrat
America/Nassau
America/New_York
America/Nipigon
America/Nome
America/Noronha
America/North_Dakota/Beulah
America/North_Dakota/Center
America/North_Dakota/New_Salem
America/Nuuk
America/Ojinaga
America/Panama
America/Pangnirtung
America/Paramaribo
America/Phoenix
America/Port-au-Prince
America/Port_of_Spain
America/Porto_Acre
America/Porto_Velho
America/Puerto_Rico
America/Punta_Arenas
America/Rainy_River
America/Rankin_Inlet
America/Recife
America/Regina
America/Resolute
America/Rio_Branco
America/Rosario
America/Santa_Isabel
America/Santarem
America/Santiago
America/Santo_Domingo
America/Sao_Paulo
America/Scoresbysund
America/Shiprock
America/Sitka
America/St_Barthelemy
America/St_Johns
America/St_Kitts
America/St_Lucia
America/St_Thomas

```
America/St_Vincent
America/Swift_Current
America/Tegucigalpa
America/Thule
America/Thunder_Bay
America/Tijuana
America/Toronto
America/Tortola
America/Vancouver
America/Virgin
America/Whitehorse
America/Winnipeg
America/Yakutat
America/Yellowknife
```

[12]:
```
# Confirm Times are in UTC
df['timestamp'] = pd.to_datetime(df['timestamp'], utc=True)

# Convert to EDT (pytz takes into account Daylight Savings automatically)
# 'America/Detroit' was the closest one to Ann Arbor
# Converting using pytz & Pandas: https://pandas.pydata.org/docs/reference/api/
 ↪pandas.Series.dt.tz_convert.html
df['timestamp'] = df['timestamp'].dt.tz_convert('America/Detroit')

df['timestamp']
```

[12]:
```
0        2019-07-08 17:04:03-04:00
1        2019-07-08 17:04:04-04:00
2        2019-07-08 17:04:07-04:00
3        2019-07-08 17:04:14-04:00
4        2019-07-08 17:04:15-04:00
                     …
40644    2019-10-03 19:04:54-04:00
40645    2019-10-03 19:04:56-04:00
40646    2019-10-03 19:04:57-04:00
40647    2019-10-03 19:05:02-04:00
40648    2019-10-03 19:05:05-04:00
Name: timestamp, Length: 40649, dtype: datetime64[ns, America/Detroit]
```

Now let's repeat the process and see the unique hours in which the professor was working out, in EDT

[13]:
```
df['hour'] = df['timestamp'].dt.hour
unique_hours = df['hour'].unique()
print(sorted(unique_hours))
```

```
[7, 8, 10, 11, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23]
```

Ah, these numbers make a lot more sense! The professor is now working out in the mornings (7-11

AM), afternoons (1-4 PM), and evenings (5-11 PM). These times span more typical times a person is awake throughout the day. Thus, I would like to proceed with the EDT times we've converted from the data.

3.0.6 Separating Data into Workouts

I wanted to consider how many workouts were recorded in total. After manually looking through the data, I thought about the names of datafiles. If Professor Brooks were to manually press 'Start' and 'Stop' on his fitness tracker each time he began and ended his workouts, the workout would be saved as a new file. I checked this below this below.

[14]:
```
len(get_unique_values(df, 'datafile')['datafile'])
```

[14]:
```
64
```

Thus I am led to believe that there are 64 separate files/workouts in the Strava CSV file.