1. About me

My name is **Reza Farzad**, and I am currently a third-year Ph.D. candidate in **Civil Engineering** at the **University of Notre Dame**. Alongside my doctoral studies, I am also pursuing a master's degree in **Applied and Computational Mathematics and Statistics**. My academic journey began in Iran, where I was ranked among the top 200 out of 38,000 candidates in the national entrance exam for civil engineering graduate programs, earning me admission to Amirkabir University of Technology, one of the top three universities in the country. In 2022, I was honored to receive a fully funded Ph.D. position at Notre Dame.

I have a strong passion for programming and data analysis, with **Python** being my primary language. I also work with **SQL** within RStudio environments. Beyond academics, I enjoy reading books on personal growth and spending time in the gym doing weight training. I took this course to strengthen my Python skills, recognizing its foundational importance in the field of data science. Please feel free to contact me at **rfarzad@nd.edu**, or connect with me on LinkedIn.

2. Homeworks

2.1. HW1 - Part D: Functions, lists, and dictionaries

In this part of Homework 1, I worked with data representing two groups of sheep: those that received treatment and those that did not. The goal was to store these data points in a dictionary and then create a function that computes the mean of each group. This task is useful in data science when summarizing group-based data and calculating key statistics across labeled categories. It strengthens the concept of using dictionaries for structured data and looping over dictionary items to apply operations.

The function `get_mean_dict()` iterates through the dictionary, computes the mean for each list of values, and returns a new dictionary with the same keys but averaged values. This is a common practice in data processing.

```
#############################
### Data & Previous Codes ###
#############################

# The followings are the lists we have
treated = [18, 43, 28, 50, 16, 32, 13, 35, 38, 33, 6, 7]
untreated = [40, 54, 26, 63, 21, 37, 39, 23, 48, 58, 28, 39, 42]

# Create a function to calculate the mean of a list input called num_list and return the mean
def get_mean(num_list):
    return sum(num_list)/len(num_list)


###########################
### Dictionary Creation ###
###########################

# Problem 1 - Create a dictionary called 'data_dict' containing the sheep experiment data
```

```python
data_dict = {'treated': treated, 'untreated':untreated}


###########################
### Function and Result ###
###########################

# Problem 2 - Create get_mean_dict() function that
#             takes a dictionary and iterates over the keys,
#             and calculates the mean of the values
def get_mean_dict(input_dict):

    # Initializing a dictionary to store
    means = {}

    # Define a loop over input dictionary
    for key_i in input_dict.keys():

        # Compute the mean value of each key and value in the dictionary
        means[key_i] = get_mean(input_dict[key_i])

    return means

get_mean_dict(data_dict)
```

```
Out[ ]: {'treated': 26.583333333333332, 'untreated': 39.84615384615385}
```

Output Summary

This result shows the average number of worms in each sheep in each group. The `treated` group had a lower mean than the `untreated` group, indicating that the average number of tapeworms in the treated lambs is lower than untreated lambs. This shows a possible effect of the treatment. The output is a simple dictionary with the same keys as the input but transformed values.

Self-Evaluation:

This solution is effective because it uses a clear function that separates list mean calculation (`get_mean`) from computations for dictionary (`get_mean_dict`). I used a for loop to iterate over the dictionary, which made the code simple and readable. One area I could improve upon is on printing and rounding the output up to two digits. However, for a beginner-level task (as the first HW), I believe the structure is clear, and the code correctly meets the assignment's requirements.

2.2. HW2 - Part A: Geocoding and Data Exploration with Pandas

In this assignment, I worked with a CSV file containing data on over 51,000 cities, aiming to explore and clean the dataset before geocoding. The tasks focused on understanding the structure of the dataset, identifying missing values, and analyzing location name patterns such as repeated city names. This process is particularly important in geographic data science where repetition of similar location names can lead to incorrect results if not properly handled.

The dataset was loaded using `pandas.read_csv` with the `geonameid` as the index. From there, I calculated row/column dimensions, memory usage, and performed value counts on the `name` column to investigate frequency and repetition patterns of city names.

```python
# Problem 1 - Mount Google drive, import pandas, and load the dataset using pd.read_csv
# Mount Google drive
from google.colab import drive
drive.mount('/content/drive')
# Import Pandas using the usual alias
import pandas as pd
# Import top_cities data with geonameid as the row index
top_cities = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/top_cities_clean.csv', index_col='geonameid')
print(top_cities.shape)
# top_cities


# Problem 2 - Check missing values and memory usage
top_cities.info()
# ######################################################################################
# ### This is a better investigation of missing values based on Session 4 lessons ####
# print(top_cities.shape, '\n')
# print('Datatype of each column:\n', top_cities.dtypes, '\n') # A goodway to see dimensions
# print('Number of null values of each column:\n', top_cities.isnull().sum(), '\n')
# print('Proportion of null values of each column:\n', top_cities.isnull().mean(), '\n')
# print('The frequency distribution of the number of missing values per row:\n',
# top_cities.isnull().sum(axis = 'columns').value_counts(), '\n')
# print('The relative frequency distribution of the number of missing values per row:\n',
# top_cities.isnull().sum(axis = 'columns').value_counts(normalize=True), '\n')

# # Create Heatmap of missingness to see how missing value are located
```

```python
# import seaborn as sns
# sns.set(rc = {'figure.figsize':(12,12)})
# sns.heatmap(top_cities.isnull(), cbar=False)
# #####################################################################################


# Problem 3 - Count how many times each city name appears
print(top_cities['name'].value_counts().head(10))


# Problem 4 - How many locations are there with a shared location names
new_Series = top_cities['name'].value_counts()
name_count = 0
loc_count = 0
i=new_Series.iloc[name_count]
while i>1:
  # print(new_Series.iloc[name_count])
  name_count += 1
  loc_count += i
  i=new_Series.iloc[name_count]
print(f'\nThe number of shared names are {name_count}, involving {loc_count} cities totally.')


# Problem 5 - What are the top 10 most common city names
top_ten_city_names_list = []
for i in range(10):
  top_ten_city_names_list.append(new_Series.index[i])
print(f'\nThe top ten cities include {top_ten_city_names_list}.')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
(51603, 20)
<class 'pandas.core.frame.DataFrame'>
Index: 51603 entries, 3039163 to 1106542
Data columns (total 20 columns):
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   name                   51602 non-null  object
 1   asciiname              51601 non-null  object
 2   alternatenames         45253 non-null  object
 3   latitude               51603 non-null  float64
 4   longitude              51603 non-null  float64
 5   feature class          51603 non-null  object
 6   feature code           51603 non-null  object
 7   country code           51568 non-null  object
 8   alternate country code  605 non-null    object
 9   admin1 code            51595 non-null  object
 10  admin2 code            41486 non-null  object
 11  admin3 code            20141 non-null  object
 12  admin4 code            6745 non-null   object
 13  population             51603 non-null  int64
 14  elevation              9968 non-null   float64
 15  dem                    51603 non-null  int64
 16  timezone               51603 non-null  object
 17  modification date      51603 non-null  object
 18  exp_feature_code       51603 non-null  object
 19  elev_dif               9968 non-null   float64
dtypes: float64(4), int64(2), object(14)
memory usage: 8.3+ MB
The relative frequency distribution of the number of missing values per row:
 5    0.295119
6    0.233436
3    0.201326
4    0.177780
2    0.083871
7    0.005581
1    0.002810
0    0.000078
Name: proportion, dtype: float64

name
Richmond       14
Santa Ana      13
Santa Cruz     13
San Pedro      13
Springfield    12
Victoria       12
Greenville     12
Clinton        12
Santa Rosa     12
San Luis       12
Name: count, dtype: int64

The number of shared names are 2457, involving 6528 cities totally.

The top ten cities include ['Richmond', 'Santa Ana', 'Santa Cruz', 'San Pedro', 'Springfield', 'Victoria', 'Greenville', 'Clint
on', 'Santa Rosa', 'San Luis'].
```

Output Summary:

- The dataset has **51603 rows and 20 columns**.
- It uses around **8.3 MB** of memory, and seven columns (e.g., `elevation`, `admin3 code`) contain missing values.
- The most common city name is "Richmond" with **14** repetitions. Other frequently repeated names include "Santa Ana", "Santa Cru", "San Pedro", and "Springfield".
- There are **2457** city names shared by more than one city, involving a total of **6528** cities.
- The top 10 repeated city names are: `['Richmond', 'Santa Ana', 'Santa Cruz', 'San Pedro', 'Springfield', 'Victoria', 'Greenville', 'Clinton', 'Santa Rosa', 'San Luis']`.

Self-Evaluation:

This assignment helped me improve foundational `pandas` skills like reading data, handling missing values, and analyzing value frequency. I used `value_counts()` to identify repeated city names and implemented a loop to summarize the extent of duplication. For detecting missing values, the commented code (learned in session 4) could give a better understanding about the location of missing values. The solution was effective, but the logic for counting shared names could be simplified using boolean filtering. Also, visualizing the results (e.g., bar chart of top 10 names) could further improve readability. Overall, I believe the code is clear and functional, and it serves its purpose in this context.

2.3. HW3 - Part C: groupby, pivot_table, and crosstab

In this assignment, I worked with a dataset of user film reviews to explore the time lag between when movies were released and when they were reviewed. I calculated the average number of years between release and review (`rel_rev`) for each movie entry and analyzed the distribution by users and genres. This is a valuable exercise in using pandas to clean and group time-related data.

The task involved using `groupby` and `pivot_table` methods, as well as `crosstab` function in pandas to assess which users tend to review older movies and which genres have higher review delays. These skills are foundational for time-based user analytics in real-world recommender systems.

```
#############################
### Data & Previous Codes ###
#############################

# Import necessary libraries
import pandas as pd

# Mount your Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Import top users dataset
rat_pow = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/rat_pow.csv')

# Split out movie release date and add as a new column called 'release_date'
rat_pow['release_date'] = (rat_pow['title'].str.extract(r'(\(\d{4}\))'))
rat_pow.dropna(inplace = True)

# Remove parenthesis and convert to integer for release year
rat_pow['rel_year'] = rat_pow['release_date'].str.lstrip('(').str.rstrip(')').astype('int64')

# Convert review timestamp to datetime variable for day in a new column
rat_pow['day'] = pd.to_datetime(rat_pow['timestamp'], unit = 's').dt.day_name()

# Convert review timestamp to datetime variable for year in a new column
rat_pow['year'] = (pd.to_datetime(rat_pow['timestamp'], unit = 's').dt.year)


#################
### Questiosn and Solutions ###
#################
# Problem 1 - Add a column of lag between when a film was released and when it was reviewed
rat_pow['rel_rev'] = (rat_pow['year'] - rat_pow['rel_year'])
mean_value = rat_pow['rel_rev'].mean()
print('\nProblem 1: ')
print(f'The average time difference in years between when a film was released and when the film was reviewed is {mean_value.round(2)} yea


# Problem 2 - For each top user, compute the average time difference between when a film was released and when it was reviewed
print('\nProblem 2: ')
print(rat_pow.groupby('userId').agg({'rel_rev':'mean'}))


# Problem 3 - For each top user, show the average lag between film release year and review year by genre
print('\nProblem 3: ')
print(rat_pow.pivot_table(index='userId',
                    columns='genres',
                    values='rel_rev',
                    aggfunc='mean'))

# Problem 4 - For each top user, show, by genre, the average rating, the average lag between film release year and review year, and the n
# Hint: Adding title count helps see which numbers might be outliers
rat_pow_prb4 = rat_pow.groupby(['userId','genres']).agg({'rating':'mean','rel_rev':'mean','title':'count'}).rename(columns={'title':'rev_
print('\nProblem 4: ')
print(rat_pow_prb4.head(10))

# Problem 5 - What is the proportion of films reviewed by each top user for each day?
print('\nProblem 5: ')
print(pd.crosstab(index=rat_pow['userId'],
                  columns=rat_pow['day'],
                  normalize=True))
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).

Problem 1:
The average time difference in years between when a film was released and when the film was reviewed is 13.71 years


Problem 2:
          rel_rev
userId
111      9.741294
274     10.894495
307     12.043103
318     15.328358
414      6.721453
448      6.716146
474     19.834471
599     22.470464
603     10.260700
606     19.153226

Problem 3:
genres      Comedy  Documentary      Drama     Horror   Thriller
userId
111       9.589595     7.900000  12.625000        NaN   9.000000
274       9.215385     3.400000  11.955556  18.300000   9.125000
307      10.852113     5.500000  13.576271  17.181818   9.000000
318      15.117647     9.802817  20.430380        NaN        NaN
414       8.319328     2.836735   6.042802  11.444444   4.400000
448       7.166052     2.923077   6.063291  37.500000   2.368421
474      20.644737     5.962500  22.137931  33.812500  21.315789
599      24.137168    16.944444  21.763441  21.285714  20.000000
603       8.800000     5.291667  11.287582  17.909091   6.444444
606      22.328125     4.000000  17.784431  24.375000  24.000000

Problem 4:
                      rating    rel_rev  rev_count
userId genres
111    Comedy       3.404624   9.589595        173
       Documentary  4.050000   7.900000         10
       Drama        3.781250  12.625000         16
       Thriller     4.000000   9.000000          2
274    Comedy       3.042308   9.215385        130
       Documentary  3.500000   3.400000          5
       Drama        3.300000  11.955556         45
       Horror       3.083333  18.300000         30
       Thriller     3.500000   9.125000          8
307    Comedy       2.464789  10.852113        142

Problem 5:
day       Friday    Monday  Saturday    Sunday  Thursday   Tuesday  Wednesday
userId
111     0.002072  0.000000  0.002368  0.000000  0.000296  0.012134   0.042616
274     0.005031  0.004143  0.002072  0.017757  0.006511  0.014797   0.014205
307     0.024268  0.002368  0.012430  0.000296  0.022196  0.002368   0.004735
318     0.008878  0.005031  0.018645  0.009174  0.004439  0.007399   0.005919
414     0.018645  0.054750  0.005919  0.015093  0.009470  0.048831   0.018349
448     0.007991  0.015093  0.028411  0.018349  0.031074  0.006215   0.006511
474     0.030778  0.045872  0.007991  0.014797  0.027819  0.023676   0.022492
599     0.012430  0.054158  0.002664  0.000296  0.007991  0.043208   0.019532
603     0.039065  0.001184  0.003255  0.027819  0.000000  0.004735   0.000000
606     0.001776  0.005327  0.010358  0.017757  0.007695  0.026043   0.004439
```

Output Summary:

- The average lag between a movie's release year and its review year is **13.71** years.
- User **599** had the highest average delay (about 22.47 years), while user **448** had the lowest (about 6.72 years).
- When analyzing by genre, users showed the longest lag times in **Horror** films, but the lowest lag times in **Documentary** films.
- The crosstab revealed which days of the week users are more active. For instance, user **414** did most reviews on **Monday**, while user **603** leaned toward **Friday**.

These summaries helped identify patterns in reviewing habits and preferences across different users and genres.

Self-Evaluation:

This task required strong understanding of pandas' aggregation and reshaping tools. I effectively used `groupby`, `pivot_table`, and `crosstab` to explore how users differ in their review behaviors. The most challenging part was aligning the data types correctly and ensuring that the aggregation function does what we want. I'm satisfied with the clarity and modularity of the code, but it could be enhanced by adding visualizations or handling missing genre data.

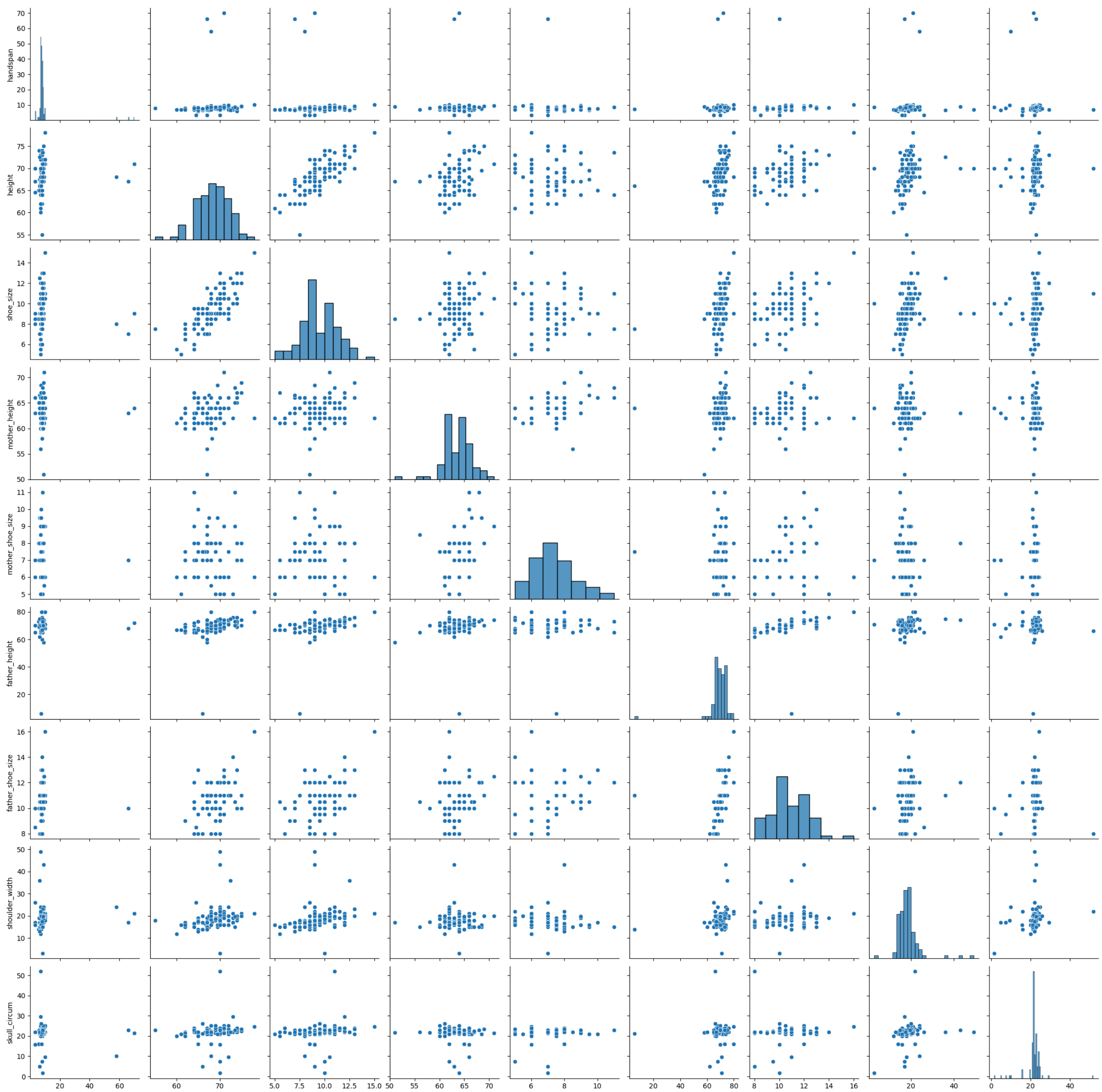
2.4. HW4 - Part B: Data Cleaning and Visualization

In this assignment, I analyzed body measurement data using various visual tools and data cleaning steps. The initial goal was to locate anomalies using `sns.pairplot()` and remove meaningless entries such as extremely short heights or large skull circumferences. Then, I cleaned the gender column by removing entries labeled "Prefer not to say" and explored gender differences in height and shoe size.

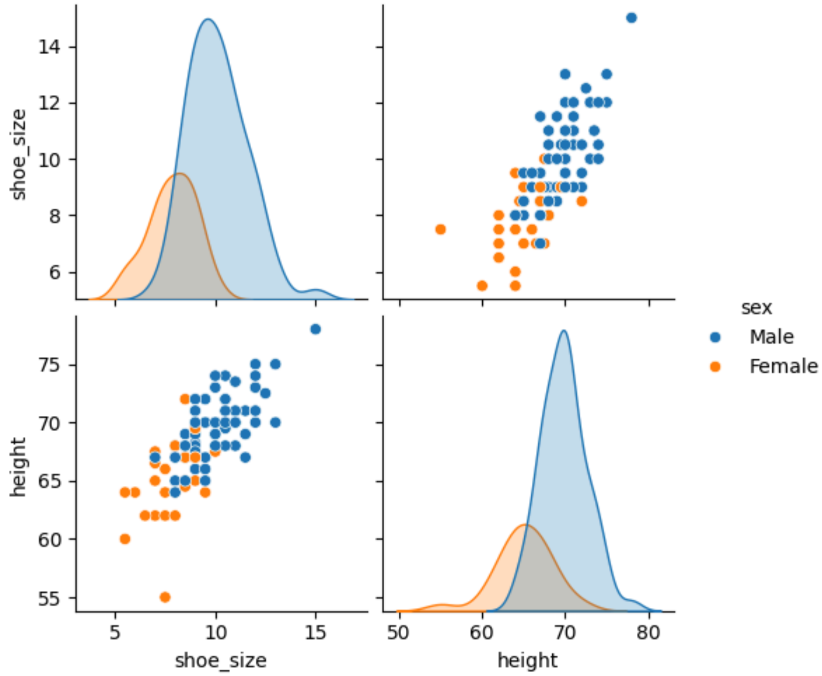
Using grouping and plotting techniques, I visualized trends and estimated average heights by gender and shoe size. I also explored how height distributions differ between athletes and non-athletes. The exercise enhanced essential skills in data visualization, grouping, and basic anomaly detection.

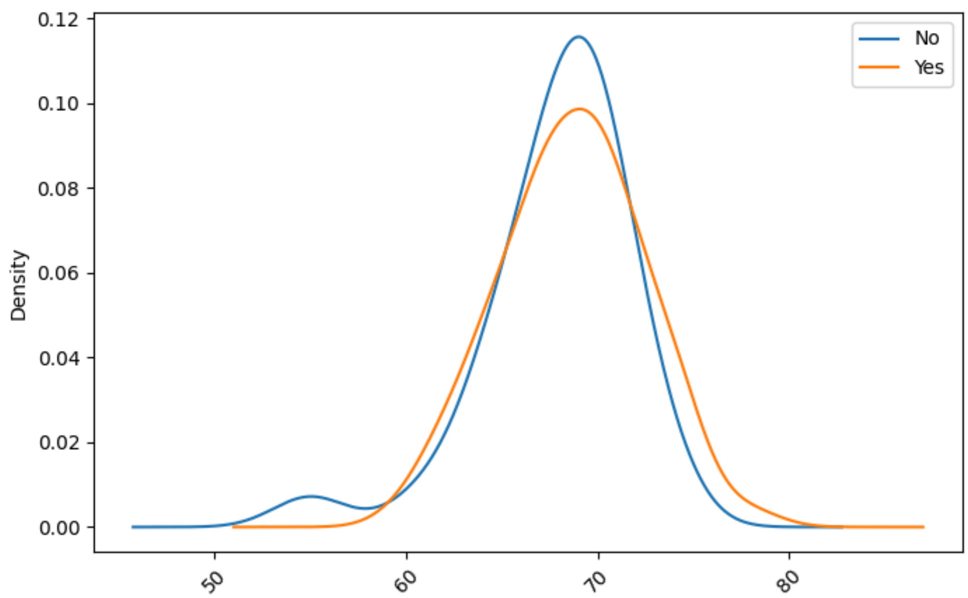
```
# Import libraries and data
import pandas as pd
import seaborn as sns

body_data = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/class_survey_body_measurements.csv')
col_names = ['timestamp', 'sex', 'handspan', 'height',
             'shoe_size','hair_color', 'mother_height', 'mother_shoe_size',
             'mother_hair_color', 'father_height', 'father_shoe_size',
             'father_hair_color','athlete','shoulder_width','skull_circum']
body_data.columns = col_names
body_data.head()


# Problem 1 - Create a scatterplot matrix of the entire data set and determine at least two variables that appear to have suspect values.
sns.pairplot(data=body_data)


# Problem 2 - How many observations were there for each level of sex (Male, Female, or Prefer not to say)?
#             Because there are very few who prefer not to say, permanently remove them from the data using the drop(index = XXXX) method
#             Take appropriate steps to verify that these observations have been removed.
print(body_data['sex'].value_counts())
ind_prefer_not_to_say = body_data[body_data['sex'] == 'Prefer not to say'].index[0]
print(f'\nThe index of prefer_not_to_say is: {ind_prefer_not_to_say}')
print('\nThe number of rows before removing the data points: ', body_data.shape)
body_data = body_data.drop(index=ind_prefer_not_to_say)
print('The number of rows after removing the data points: ', body_data.shape)
print(body_data['sex'].value_counts())


# Problem 3 - Create a DataFrame called avg_heights containing 3 columns, 1.) sex, 2.) shoe_size, 3.) the average height
#            (by sex and shoe_size). Print out the DataFrame.
avg_heights = body_data[['sex', 'shoe_size', 'height']].groupby(['sex','shoe_size']).mean()
print(avg_heights.head(10))


# Problem 4 - Create a plot using Seaborn's pairplot() on the avg_heights data, setting the hue = sex.
plt.figure(figsize=(10, 10))
sns.pairplot(data=body_data[['sex', 'shoe_size', 'height']], hue='sex')
plt.show()


# Problem 5 - Create overlapping kernel density estimates of height by athlete and height by sex. Interpret what you see in the plot.
# (Height by Athelete)
plt.figure(figsize=(8, 5))
body_data.groupby('athlete')['height'].plot(kind='kde',
                                            legend = True,
                                            rot=45)
plt.show()

# (Height by Sex)
plt.figure(figsize=(8, 5))
body_data.groupby('sex')['height'].plot(kind='kde',
                                        legend = True,
                                        rot=45)
plt.show()
```

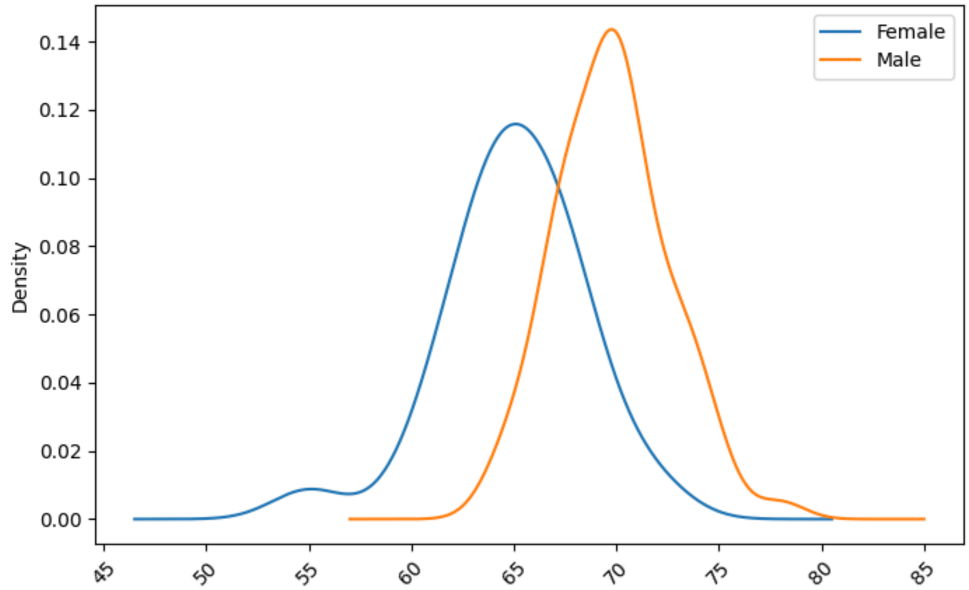
```
sex
Male                 70
Female               27
Prefer not to say     1
Name: count, dtype: int64

The index of prefer_not_to_say is: 28

The number of rows before removing the data points:  (102, 15)
The number of rows after removing the data points:  (101, 15)
sex
Male      70
Female    27
Name: count, dtype: int64
                  height
sex    shoe_size
Female 5.5        62.000000
       6.0        64.000000
       6.5        62.000000
       7.0        65.250000
       7.5        61.750000
       8.0        64.666667
       8.5        67.125000
       9.0        67.600000
       9.5        64.000000
       10.0       67.500000
```



<Figure size 1000x1000 with 0 Axes>







Output Summary:

- Problem 1: The values of variable father_height less than 20 inches does not make any sense. The reason could be that the person filling the form did not understand the question or filled the form carelessly. Also, skull circumference greater than 50 does not have any meaning. Perhaps, the person filling the form entered the value in centimeter instead of inches because 50cm = 20cm approximately.
- Problem 2: I removed 1 entry from the dataset where the sex was marked as "Prefer not to say".
- Problem 3: The `avg_heights` table shows how average height varies across shoe sizes and between males and females. Males generally have higher shoe sizes and heights.
- Problem 4: The shoe_size of 9 and the height of 68 could be a reasonable cut-off differentiating Female and Male. However, this is not a perfect predictor, especially if the values are close to these cut-off values.
- Problem 5: Based on average, there is no difference. However, based on distribution, athletes range a wider variety of heights than non-athletes. Also, people who are extremely tall are athletes while people who are extremely short are not. Men are on average taller than female, while the variability of them is similar. Also, extreme tall cases relate to male, while extreme short cases relate to female.

Self-Evaluation:

This section strengthened my understanding of how visualization can help get overall idea about datasets. I was able to clean data based on logic and used pairplots and density plots to explore patterns in height and shoe size by sex. The biggest challenge was interpreting the plots meaningfully and deciding if we can define justifiable thresholds. I could improve this work by finding outliers. Overall, this task helped me practice combining visual analysis with structured logic.

2.5. HW5 - Part B: Pattern Matching & String Methods

In this assignment, I worked with airline flight data to identify patterns in company names, aircraft models, and tail numbers using string methods and regular expressions. First, I counted flights operated by companies whose names ended with a period (e.g., "Co."). Then, I used a regex to filter Boeing aircraft that follows a specific model format (###-###). Finally, I analyzed Delta Airlines' fleet to determine what proportion of their planes still had tail numbers ending in "NW" — inherited from the Northwest Airlines merger in 2010.

This exercise provided practice with `str.endswith()`, `str.match()`, as well as left and outer merge.

```
#############################
### Data & Previous Codes ###
#############################

# Load Libraries
import pandas as pd
import matplotlib.pyplot as plt
import geopandas

# Mount your Google Drive
from google.colab import drive
drive.mount('/content/drive')

# Make sure we can see all the columns in our dataframes
pd.set_option('display.max_colwidth', None)
```

```python
df_flights = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/nycflights/flights.csv')
print(df_flights.shape)
df_airlines = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/nycflights/airlines.csv')
print(df_airlines.shape)
df_planes = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/nycflights/planes.csv')
print(df_planes.shape)
df_airports = pd.read_csv('drive/MyDrive/Python Resources/notebooks/data-sets/nycflights/airports.csv')
print(df_airports.shape)

# Data is too large. We filter it to make it more manageable.
# First, filter your flights data frame to just include flights from January of 2013, then remove any rows that contain missing values.
flights1= df_flights[(df_flights['month']==1) & (df_flights['year']==2013)].dropna()

# Create a subset data frame from the planes data frame that includes the tailnum, the model and the manufacturer.
# Join this data frame with flights1 so that all of the rows in flights1 are utilized.
# The rows of the planes data frame should only be used if they correspond to a row in flights1.
df_planes_subset = df_planes[['tailnum', 'model', 'manufacturer']]
flights2 = flights1.merge(df_planes_subset, on='tailnum', how='left')

# merge flights2 with the airlines data frame. We would like to keep information from both data frames so that
# we can have the full carrier name in the current data set for the flights, as well as see any airlines that do not fly in and out of Ne
flights3 = flights2.merge(df_airlines, on='carrier', how='outer')

# Add latitude and longitude of the destination airport to our flights3 dataframe
df_airports_subset = df_airports[['faa', 'lat', 'lon']].rename(columns={'faa':'dest', 'lat': 'dest_lat', 'lon':'dest_lon'})
flights_final = flights3.merge(df_airports_subset, on='dest', how='left')


#################
### Questiosn and Solutions ###
#################
# Problem 1 - Some of the carriers include American Airlines Inc., Southwest Airlines Co. and JetBlue Airways.
#             How many of our flights were from companies that ended with a period (such as Inc. or Co.)?
print('\nProblem 1: ', flights_final['name'].str.endswith('.').sum())


# Problem 2 - Boeing is the only airline that gives model numbers to their aircraft as ###-###.
#             Use regular expressions to identify the number of flights in flights_final that were on Boeing planes.
print('\nProblem 2: ', flights_final['model'].str.match(r'^\d{3}-\d{3}$').sum())

# Problem 3 - Northwest planes have tail names that end in "NW". What proportion of Delta's fleet, in January of 2013, were still Northwe

delta_flights = flights_final[flights_final['carrier']=='DL']
unique_delta_tailnum = delta_flights['tailnum'].unique()
print('\nProblem 3: ', sum(tail.endswith('NW') for tail in unique_delta_tailnum) / len(unique_delta_tailnum))
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=Tru
e).
(336776, 19)
(16, 2)
(3322, 9)
(1458, 8)

Problem 1:  19144

Problem 2:  5052

Problem 3:  0.12162162162162163
```

Output Summary:

- There were **19,144** flights operated by companies whose names ended in a period (like "Co." or "Inc.").
- **5,052** flights involved Boeing aircraft with a model pattern of ###-###.
- Out of all unique Delta Airlines tail numbers in January 2013, **12.16%** ended in "NW", indicating that they were originally part of Northwest Airlines' fleet before the 2010 merger.

These string-based searches provide meaningful insights into data (airline identity and fleet history) using simple operations.

Self-Evaluation:

This task helped me improve my understanding of Python string operations, especially `.str.endswith()` and `.str.match()` with regular expressions. I found the Boeing model pattern matching to be the most interesting, as it used real-world formatting to filter data. The logic for determining inherited tail numbers from Northwest Airlines was straightforward but meaningful in interpreting company mergers. One area I could improve would be using pandas chaining for shorter filtering.

3. Growth Summary

Over the past six weeks, my understanding of programming in Python has evolved from simply writing code to deeply analyzing data with purpose. Initially, I approached problems by thinking about syntax or finding a function that worked. Now, I think more in terms of **how** the data flows through the program and **why** a certain structure or pattern is meaningful. Now, I start with significant data preprocessing to get insight about the data. Whether it's grouping values by categories, filtering outliers using visualizations, dealing carefully with missing values, or reshaping datasets with pivot tables, I've developed a more intuitive understanding of what it means to "work with data" rather than just manipulate it.

This course has also influenced how I think about the role of programming in my broader academic and research goals. As a Ph.D. student working on engineering problems, I now see how Python can help me systematically track trends and explain results clearly. For example, visualizations are no longer just plots, but they are tools for identifying inconsistencies and communicating insights. I've come to appreciate how programming supports critical thinking, and this shift in mindset has been the most valuable growth I've experienced.