DS 3000 HW 2

Due: Friday July 12th @ 11:59 PM EST

Submission Instructions

Submit this `ipynb` file to Gradescope (this can also be done via the assignment on Canvas). To ensure that your submitted files represent your latest code, make sure to give a fresh `Kernel > Restart & Run All` just before uploading the files to gradescope.

Tips for success

- **Start early**
- **Make use of Piazza**
- **Make use of Office hour**
- **Remember to use cells and headings to make the notebook easy to read (if a grader cannot find the answer to a problem, you will receive no points for it)**
- **Under no circumstances may one student view or share their ungraded homework or quiz with another student (see also), though you are welcome to talk about (not show each other) the problems.**

Part 1: Arrays

Part 1.1: (10 points: 5 pts each)

Create the following two arrays using the NumPy library. Call the first array `array_a` and the second array `array_b`. Use .linspace and .reshape to create `array_b`:

$$\text{array\_a} = \begin{bmatrix} 3 & -8 & -2 & 3 \\ .5 & 2 & 6 & 14 \\ 9 & 7 & 69 & 2 \end{bmatrix}$$

$$\text{array\_b} = \begin{bmatrix} 42 & 38 & 34 & 30 & 26 \\ 22 & 18 & 14 & 10 & 6 \\ 2 & -2 & -6 & -10 & -14 \end{bmatrix}$$

In [69]:

```
import numpy as np


array_a = np.array([[3, -8, -2, 3],
                    [0.5, 2, 6, 14],
                    [9, 7, 69, 2]])

array_b = np.linspace(42, -14, 15).reshape(3, 5)

print('array a:')
print(array_a)
print('array b:')
print(array_b)
```

```
array a:
[[ 3.   -8.   -2.    3. ]
 [ 0.5   2.    6.   14. ]
 [ 9.    7.   69.    2. ]]
array b:
[[ 42.  38.  34.  30.  26.]
 [ 22.  18.  14.  10.   6.]
 [  2.  -2.  -6. -10. -14.]]
```

Part 1.2: (15 points: 5 pts each)

1. **Give the shape, size, ndim, and nbytes for each of the two arrays.**
2. **Take the transpose of both arrays. Call these `t_array_a` and `t_array_b`.**
3. **Try to add `array_a` and `array_b`, then remove the last column of `array_b` and try to add them again. In a markdown cell, explain what happened.**

In [70]:

```python
print('1.')
print('ARRAY A DETAILS')
print("Shape of array_a:", array_a.shape)
print("Size of array_a:", array_a.size)
print("Number of dimensions of array_a (ndim):", array_a.ndim)
print("Number of bytes of array_a (nbytes):", array_a.nbytes)

print(' ')

print('ARRAY B DETAILS')
print("Shape of array_b:", array_b.shape)
print("Size of array_b:", array_b.size)
print("Number of dimensions of array_b:", array_b.ndim)
print("Number of bytes of array_b:", array_b.nbytes)

print('')

print('2.')
t_array_a = array_a.T
t_array_b = array_b.T
print('array_a transposed')
print(t_array_a)
print('')
print('array_b transposed')
print(t_array_b)
```

```
1.
ARRAY A DETAILS
Shape of array_a: (3, 4)
Size of array_a: 12
Number of dimensions of array_a (ndim): 2
Number of bytes of array_a (nbytes): 96

ARRAY B DETAILS
Shape of array_b: (3, 5)
Size of array_b: 15
Number of dimensions of array_b (ndim): 2
Number of bytes of array_b (nbytes): 120

2.
array_a transposed
[[ 3.   0.5  9. ]
 [-8.   2.   7. ]
 [-2.   6.  69. ]
 [ 3.  14.   2. ]]

array_b transposed
[[ 42.  22.   2.]
 [ 38.  18.  -2.]
 [ 34.  14.  -6.]
 [ 30.  10. -10.]
 [ 30.  10.  10.]
 [ 26.   6. -14.]]
```

In [13]:

```
'''
print('3.')
array_ab = array_a + array_b
print("Combining array_a and array_b:")
print(array_ab)
'''
# The code above produces an error and does not run


array_b_edited = array_b[:, :-1]
print(array_b_edited)

print('')

array_ab = array_a + array_b_edited
print("Combining array_a and array_b:")
print(array_ab)
```

```
[[ 42.  38.  34.  30.]
 [ 22.  18.  14.  10.]
 [  2.  -2.  -6. -10.]]

Combining array_a and array_b:
[[45.   30.   32.   33. ]
 [22.5 20.   20.   24. ]
 [11.    5.   63.   -8. ]]
```

1.2 Explaination

The first chunk of code did not work and produced an error because in order to add two arrays they must have the same dimentions, which they didn't. After removing the last column in array_b, then both arrays had the same dimentions and were able to be combined using '+'.

Part 2: Bike Data

Part 2.1: DataFrame Construction (10 points)

Recreate the following table of bicycle race data as a dataframe (do not write a csv and read it in to accomplish this; use pandas and dictionary). Use the `Bike ID` as the index column and save the resulting dataframe as a csv (you need not submit this csv, but be sure to include the `DataFrame.to_csv()` command in your submission).

Bike ID	Rider ID	Make	Color	Bike Type	Weight (g)	Time Trial 1 (s)	Time Trial 2 (s)
037	3	Bianchi	Celeste	Road	8200	450	205
379	1	Duratec	\	Cyclocross	9500	510	222
398	7	Trek	Red	Road	9000	432	211
37B	3	Trek	Black	Mountain	13607	561	301
BRG	7	Canondale	Black	Mountain	15005	524	299

In [71]:

```
import pandas as pd

bikes = {
    'Bike ID': ['037', '379', '398', '37B', 'BRG'],
    'Rider ID': [3, 1, 7, 3, 7],
    'Make': ['Bianchi', 'Duratec', 'Trek', 'Trek', 'Canondale'],
    'Color': ['Celeste', '<no paint>', 'Red', 'Black', 'Black'],
    'Bike Type': ['Road', 'Cyclocross', 'Road', 'Mountain', 'Mountain'],
    'Weight (g)': [8200, 9500, 9000, 13607, 15005],
    'Time Trial 1 (s)': [450, 510, 432, 561, 524],
    'Time Trial 2 (s)': [205, 222, 211, 301, 299]
}
df = pd.DataFrame(bikes).set_index('Bike ID')
df.to_csv('bicycle_data.csv')
print(df)
```

```
         Rider ID       Make       Color   Bike Type  Weight (g)  \
Bike ID
037             3    Bianchi     Celeste        Road        8200
379             1    Duratec  <no paint>  Cyclocross        9500
398             7       Trek         Red        Road        9000
37B             3       Trek       Black    Mountain       13607
BRG             7  Canondale       Black    Mountain       15005

         Time Trial 1 (s)  Time Trial 2 (s)
Bike ID
037                   450               205
379                   510               222
398                   432               211
37B                   561               301
BRG                   524               299
```

Part 2.2: Manipulating DataFrames (30 points: 10 pts each)

For each of the questions below:

- **Provide a few (1 to 3) code cells which construct a series or dataframe object which is sufficient to answer each question**
    - **one shouldn't have to look at the full dataframe or otherwise as reference**
        - **we practice this way as real datasets are too big for this to be easily done!**
- **Provide a markdown cell which contains a one sentence response to each question**
    - **In effect, you're interpretting the code cell(s) so one who knows no python can understand how your code answers the question**

Questions:

1. **Which `Bike ID` has the greatest weight?**
2. **Which `Bike ID` has the fastest average time trial?**
3. **What is the average weight of each bike, per `Bike Type`?**

    - **Hint: `groupby()` and/or `.unique()` might be helpful**

Note that:

- **each time trial records the time taken to complete a given track under similar conditions.**
- **some riders (3 and 7) completed the time trials on two distinct bikes, the data is stored in distinct rows**

Question 1 Markdown Answer

The 'Bike ID' with the greatest weight is 'BRG', which weighs 15005 grams.

Question 1 code below ↓

In [30]:

```
print('Question 1')
heaviest_bike = df['Weight (g)'].idxmax()
print("The heaviest bike is:", heaviest_bike)
```

```
Question 1
The heaviest bike is: BRG
```

Question 2 Markdown Answer

The Bike ID with the fastest average time trial is 398.

Question 2 code below ↓

In [31]:

```
print('Question 2')
df['Average Time Trial'] = df[['Time Trial 1 (s)', 'Time Trial 2 (s)']].mean(axis=1)

fastest_bike = df['Average Time Trial'].idxmin()
print('The fasted bike is:',fastest_bike)
```

```
Question 2
The fasted bike is: 398
```

Question 3 Markdown Answer

The average weight of each bike, per Bike Type, is as follows: Road bikes have an average weight of 8600 grams, Cyclocross bikes 9500 grams, and Mountain bikes 14356 grams.

Question 3 code below ↓

In [34]:

```
print('Question 3')
bike_types = df['Bike Type'].unique()

average_weight = {}

for bike_type in bike_types:
    bikes = df[df['Bike Type'] == bike_type]
    avg_weight = bikes['Weight (g)'].mean()
    avg_weight = float(avg_weight)
    average_weight[bike_type] = avg_weight

average_weight
```

```
Question 3
```

Out[34]:

```
{'Road': 8600.0, 'Cyclocross': 9500.0, 'Mountain': 14306.0}
```

Part 3: Pokémon Data

Part 3.1: Reading in Data (5 points)

On Canvas is the `pokedata.csv` file. Read this data set in, using the `Pokedex` as the index column, and print the first few rows of the data.

In [42]:

```
import pandas as pd
df = pd.read_csv('pokedata.csv', index_col='Pokedex')
print(df.head())
```

```
            Pokemon MainType SecondaryType  Height  Weight  Damage  BaseSpeed  \
Pokedex
1         Bulbasaur    Grass        Poison      24    25.2      45         45
2           Ivysaur    Grass        Poison      39    28.7      60         60
3          Venusaur    Grass        Poison      79   220.5      80         80
4        Charmander     Fire           NaN      24    18.7      39         65
5        Charmeleon     Fire           NaN      43    41.9      58         80

         Attack  Defense
```

```
Pokedex
1             49        49
2             62        63
3             82        83
4             52        43
5             64        58
```

Part 3.2: More Manipulation (30 points: 10 pts each)

1. **Add a new column to the data set that calculates the BMI of the Pokémon. The formula for BMI using Imperial Units (such as these data contain) is = $703 \times \frac{Weight}{Height^2}$. Print the first few rows.**
   - **Find out which Pokémon have (a) the highest BMI and (b) the lowest BMI.**
2. **Create a subset of the Pokémon Data that you created in (1) which (a) includes only Pokémon that have `BaseSpeed >= 60` and (b) excludes all Pokémon with `MainType == Fire`. Make sure to save this subset as a new data frame and print the first few rows of the data.**
3. **Use the `.describe()` function to produce summary statistics for the Pokémon Data from the data from (1). Create a markdown cell and explain:**
   - **What Series did the `.describe()` function run on? What Series did it not run on? What is the difference, and what does this mean the `.describe()` function is used for?**

In [68]:

```
print('Part 1')
df['BMI'] = 703 * df['Weight'] / (df['Height'] ** 2)
print(df.head())

highest_bmi = df.loc[df['BMI'].idxmax()]
lowest_bmi = df.loc[df['BMI'].idxmin()]

print('')
print(f"Pokémon with the highest BMI: {highest_bmi['Pokemon']} with a BMI of {highest_bmi
['BMI']} ")
print(f"Pokémon with the lowest BMI: {lowest_bmi['Pokemon']} with a BMI of {lowest_bmi['B
MI']}")

print('')

print('Part2')
subset = df[(df['BaseSpeed'] >= 60) & (df['MainType'] != 'Fire')]

print('')
print("Subset of Pokémon Data:")
print(subset.head())

print('')

print('Part 3')

summary_stats = df.describe()

print("Summary Statistics:")
print(summary_stats)
```

```
Part 1
            Pokemon MainType SecondaryType  Height  Weight  Damage  BaseSpeed  \
Pokedex
1         Bulbasaur    Grass        Poison      24    25.2      45         45
2           Ivysaur    Grass        Poison      39    28.7      60         60
3          Venusaur    Grass        Poison      79   220.5      80         80
4        Charmander     Fire           NaN      24    18.7      39         65
5        Charmeleon     Fire           NaN      43    41.9      58         80

         Attack  Defense        BMI
Pokedex
1            49       49  30.756250
```

```
2             62       63  13.265023
3             82       83  24.837606
4             52       43  22.823090
5             64       58  15.930611

Pokémon with the highest BMI: Golem with a BMI of 153.707173553719
Pokémon with the lowest BMI: Haunter with a BMI of 0.03542454018644495

Part2

Subset of Pokémon Data:
            Pokemon MainType SecondaryType  Height  Weight  Damage  BaseSpeed  \
Pokedex
2           Ivysaur    Grass        Poison      39    28.7      60         60
3          Venusaur    Grass        Poison      79   220.5      80         80
9         Blastoise    Water           NaN      63   188.5      79         78
12       Butterfree      Bug        Flying      43    70.5      60         70
15         Beedrill      Bug        Poison      39    65.0      65         75

         Attack  Defense        BMI
Pokedex
2            62       63  13.265023
3            82       83  24.837606
9            83      100  33.387629
12           45       50  26.804489
15           90       40  30.042735

Part 3
Summary Statistics:
           Height       Weight      Damage   BaseSpeed      Attack  \
count  151.000000   151.000000  151.000000  151.000000  151.000000
mean    46.894040   100.766954   64.218543   68.933775   72.913907
std     37.880717   131.299070   28.585519   26.746880   26.755421
min      8.000000     0.200000   10.000000   15.000000    5.000000
25%     28.000000    21.800000   45.000000   46.500000   51.000000
50%     39.000000    66.100000   60.000000   70.000000   70.000000
75%     59.000000   124.250000   80.000000   90.000000   92.000000
max    346.000000  1014.100000  250.000000  140.000000  134.000000

          Defense         BMI
count  151.000000  151.000000
mean    68.225166   32.764506
std     26.916704   23.189403
min      5.000000    0.035425
25%     50.000000   20.856877
50%     65.000000   30.042735
75%     84.000000   38.245600
max    180.000000  153.707174
```

.describe() Analysis

When I used the .describe function, it ran on Height, Weight, Damage, BaseSpeed, Attack, Defense and BMI. In addition, the function found the count, mean, std, min , max, and the 1st, 2nd, and 3rd quartiles. The function however, did not run on the pokemon names or types because it is not included in the summary. All in all, the function calculates basic numerical statistics which can be used when doing quick data analysis on numerical data sources. It is an easy was to understand a dataset