Data Visualization

Lecture 2

Introduction to Pandas Library

Data Exploration and Manipulation

NumPy Random

- NumPy offers the random module to work with random numbers.
  - Generate a Random integer from 0 to 100:

```
from numpy import random
x = random.randint(100)
print(x)
```

  - Generate a Random float from 0 to 1:

```
from numpy import random
x = random.rand()
print(x)
```

NumPy Random

- NumPy offers the random module to work with random numbers.
  - Generate a 2-D array with 3 rows, each row containing 5 random integers from 0 to 100:

```
from numpy import random
x = random.randint(100, size=(3, 5))
print(x)
```

Random Data Distribution

- Data Distribution is a list of all possible values, and how often each value occurs.

- Such lists are important when working with statistics and data science.

- The random module offer methods that returns randomly generated data distributions.

Random Data Distribution

- A random distribution is a set of random numbers that follow a certain probability density function.

- Probability Density Function: A function that describes a continuous probability. i.e. probability of all values in an array.

- We can generate random numbers based on defined probabilities using the **choice()** method of the random module.

Random Data Distribution

- Example:
  - Generate a 1-D array containing 100 values, where each value has to be 3, 5, 7 or 9.
  - The probability for the value to be 3 is set to be 0.1
  - The probability for the value to be 5 is set to be 0.3
  - The probability for the value to be 7 is set to be 0.6
  - The probability for the value to be 9 is set to be 0

Random Data Distribution

- Example:

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0],
size=(100))

print(x)
```

Random Data Distribution

- Exercise:
  - Generate a 2-D array with 3 rows, each containing 5 values:
  - The probability for the value to be 3 is set to be 0.1
  - The probability for the value to be 5 is set to be 0.3
  - The probability for the value to be 7 is set to be 0.6
  - The probability for the value to be 9 is set to be 0

Random Data Distribution

- Exercise:

```
from numpy import random

x = random.choice([3, 5, 7, 9], p=[0.1, 0.3, 0.6, 0.0], 
size=(3,5))

print(x)
```

Random Normal Data Distribution

- Generate 2D array with 3 rows and 5 columns where it is normally distributed with mean = 4 & standard deviation = 10
- Exercise:

```
from numpy import random

x = random.normal(loc = 4, scale = 10, size = (3, 5))

print(x)
```

Seaborn Module

- Seaborn is a library that uses Matplotlib underneath to plot graphs. It will be used to visualize random distributions.

- How to install Seaborn while using Jupyter?
  - C:\Users\Your name> !pip install seaborn

Seaborn Module

- Distplots
  - Distplot stands for distribution plot, it takes as input an array and plots a curve corresponding to the distribution of points in the array.

```python
import matplotlib.pyplot as plt
import seaborn as sns

sns.distplot([0, 1, 2, 3, 4, 5])

plt.show()
```

Pandas Library

- A python library used to analyze data.
- It has functions for analyzing, cleaning, exploring, and manipulating data.
- Why use Pandas?
  - Pandas allows us to analyze big data and make conclusions based on statistical theories.
  - Pandas can clean messy data sets and make them readable and relevant which is very important in data science.
  - Pandas are also able to delete rows that are not relevant, or contains wrong values, like empty or NULL values. This is called cleaning the data.

Pandas Library

- Installation of Pandas:
  - If you have Python and PIP already installed on system, then install it using this command:
    - C:\Users\your name> pip install pandas
  - Note: Anaconda python distribution has already Pandas installed.

- Importing Pandas
  - import pandas OR import pandas as pd

Pandas Library

- Series:
  - A Pandas Series is like a column in a table.
  - It is a one-dimensional array holding data of any type.

  - Example: Create a simple Pandas series from a list:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Pandas Library

- Create Labels:
  - By default, a series is labeled starting from 0 till end of list unless something else is specified.
  - Example:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
```

Pandas Library

- Key/Value objects as series:
    - Exercise: Create a simple Pandas series from a dictionary:

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

    - Note: Dictionary keys are considered the label.

Pandas Library

- Data Frame:
  - A 2D data structure, like a 2D array, or a table with rows and columns.
  - Example: Create a data frame from two series:

```
import pandas as pd
data = {
   "calories": [420, 380, 390],
   "duration": [50, 40, 45]
}
#load data into a DataFrame object:
df = pd.DataFrame(data)

print(df)
```

Pandas Library

- Locate Row:
  - Example: Return row 0:

```
print(df.loc[0])
```

  - Example: Return row 0 and 1:

```
print(df.loc[[0, 1]])
```

Pandas Library

- Load Files into a DataFrame :
  - Example: Load a comma separated file (CSV file) into a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv’)
print(df) or print(df. to_string())
```

  - Note: to_string() is used to print the entire DataFrame.

Pandas Library

- Read JSON:
  - Big data sets are often stored or extracted as JSON.
  - JSON is plain text but has the format of an object.

  - Example: Load the JSON file into a DataFrame:

```
import pandas as pd
df = pd.read_json('data.json’)
print(df.to_string())
```

Pandas Library

- Analyzing DataFrames:
  - Viewing the data:
  - Example: Print the first 5 rows of a DataFrame:

```
import pandas as pd
df = pd.read_csv('data.csv')
print(df.head())
```

  - Example: Print the last 5 rows of a DataFrame:

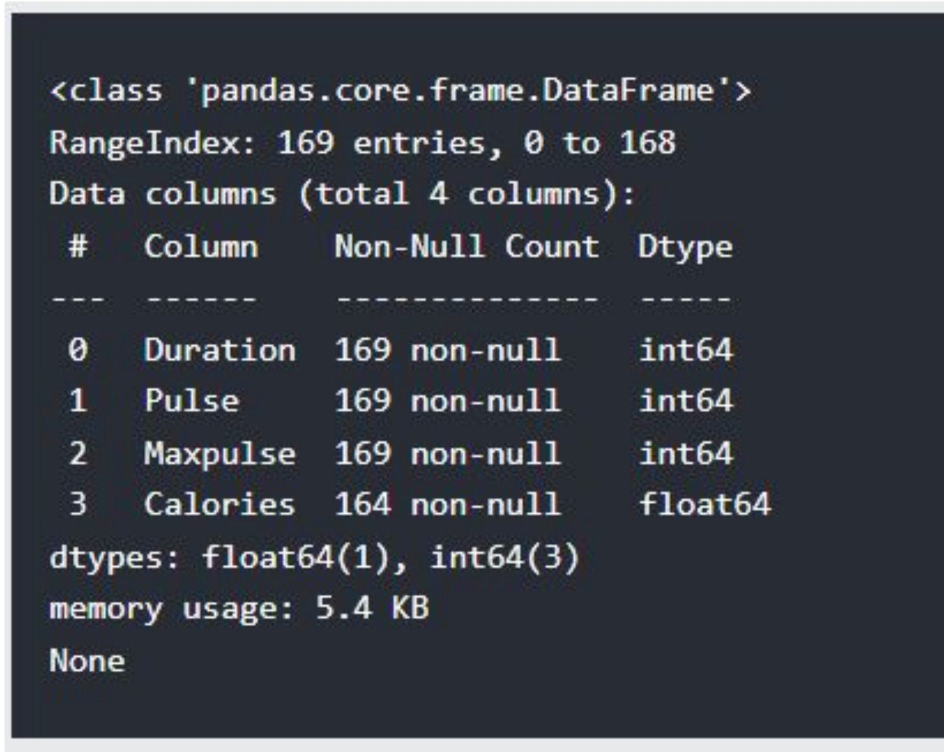
```
print(df.tail())
```

  - What will happen if we `print(df.head(10))`?

Pandas Library

- Analyzing DataFrames:
  - Information about the data:
  - Example:

```
print(df.info())
```



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 169 entries, 0 to 168
Data columns (total 4 columns):
 #   Column    Non-Null Count  Dtype  
---  ------    --------------  -----  
 0   Duration  169 non-null    int64  
 1   Pulse     169 non-null    int64  
 2   Maxpulse  169 non-null    int64  
 3   Calories  164 non-null    float64
dtypes: float64(1), int64(3)
memory usage: 5.4 KB
None
```

Data Cleaning

- Fixing bad data in your data set.
- Bad data could be:
  - Empty cells
  - Data in wrong format
  - Wrong data
  - Duplicates
- Open data_new.csv and find bad data.

Data Cleaning

- Empty Cells:
  - Null values can give wrong results when analyzing the data.
  - How can we handle them?
    - Remove rows that contain empty cells.
    - Replace empty cells with predefined value.
    - Replace only for specific columns.
    - Replace using mean, median or mode.
  - Example:

```
import pandas as pd
df = pd.read_csv('data.csv’)
new_df = df.dropna() or df.dropna(inplace = True)
print(new_df.to_string())
```

Data Cleaning

- Empty Cells:
  - Exercise: Clean the file file data_new.csv by replacing empty cells with their mean.

```
import pandas as pd
df = pd.read_csv('data.csv')
x = df["Calories"].mean()
df["Calories"].fillna(x, inplace = True)
```

  - Note: Students should implement the other methods for dealing with empty cells, this lecture only shows the implementation of two of them.

Data Cleaning

- Wrong Format:
  - It makes it difficult, or even impossible, to analyze data.
  - How can we deal with them?
    - Remove rows
    - convert all cells in the column into the same format
  - Example: In data_new.csv in the ‘Date’ column row 24 is null and row 28 contains a wrong format.
  - Solution: Convert all cells in the ‘Date’ column into dates.

```
import pandas as pd
df = pd.read_csv('data.csv')
df['Date'] = pd.to_datetime(df['Date’], format=‘mixed’)
print(df.to_string())
```

Data Cleaning

- Wrong Data:
  - We can spot wrong data by looking at the data set, because we have an expectation of what it should be.
  - Example: ‘Duration’ in row 9 from data_new.csv, It doesn't have to be wrong, but taking in consideration that this is the data set of someone's workout sessions, we conclude with the fact that this person did not work out in 450 minutes.
  - We can deal with wrong data by:
    - Replacing wrong values.
    - Removing rows.

Data Cleaning

- Wrong Data:
  - Example: Loop through all values in the "Duration" column. If the value is higher than 120, set it to 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.loc[x, "Duration"] = 120
```

  - Exercise: Delete rows where "Duration" is higher than 120:

```
for x in df.index:
  if df.loc[x, "Duration"] > 120:
    df.drop(x, inplace = True)
```

Data Cleaning

- Removing Duplicates:
  - Rows that have been registered more than one time.

  - Example: Return true for every row that is a duplicate, otherwise false:

    ```
    print(df.duplicated())
    ```

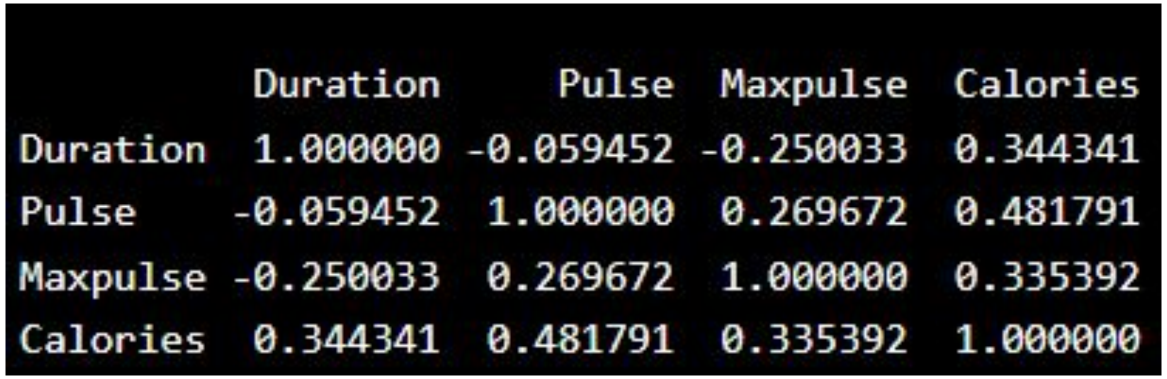
  - Exercise: Remove all duplicates:

    ```
    df.drop_duplicates(inplace = True)
    ```

Pandas Correlations

- Finding relationships between each column in the data set.

```
import pandas as pd
df = pd.read_csv('data.csv’)
print(df.corr())
```



```
          Duration     Pulse  Maxpulse  Calories
Duration  1.000000 -0.059452 -0.250033  0.344341
Pulse    -0.059452  1.000000  0.269672  0.481791
Maxpulse -0.250033  0.269672  1.000000  0.335392
Calories  0.344341  0.481791  0.335392  1.000000
```

- Note: The corr() method ignores "not numeric" columns.

Pandas Correlations

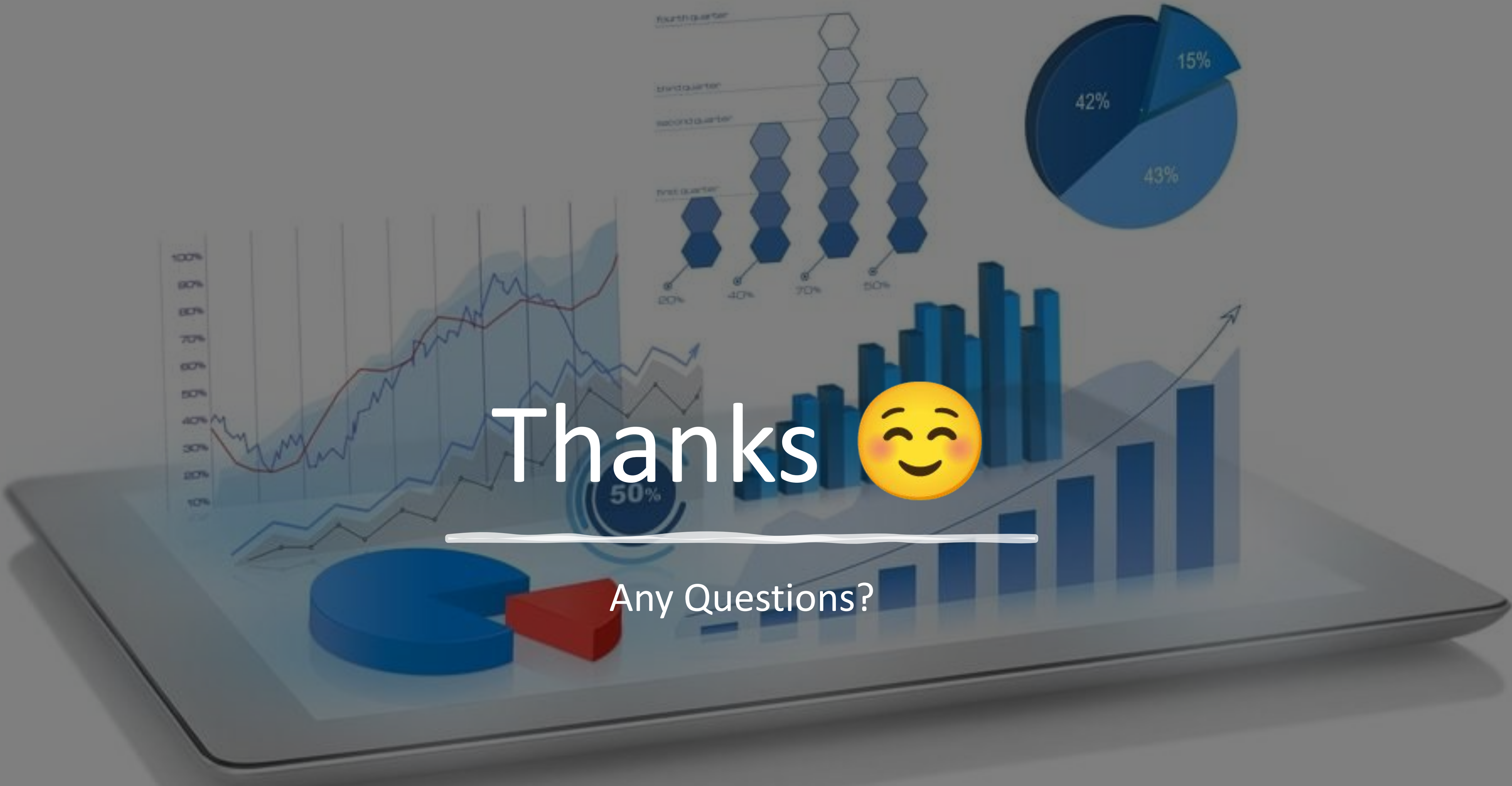
- Results Explanation:
  - The correlation value varies from -1 to 1.
  - 1 means that there is a 1 to 1 relationship (a perfect correlation), and for this data set, each time a value went up in the first column, the other one went up as well.
  - 0.9 is also a good relationship, and if you increase one value, the other will probably increase as well.
  - -0.9 would be just as good relationship as 0.9, but if you increase one value, the other will probably go down.
  - 0.2 means NOT a good relationship, meaning that if one value goes up does not mean that the other will.

Pandas Correlations

- Using the data from ‘correlations_data’ file.
- Apply the correlation for the data
- Then submit your interpretation

Pandas Correlations

- Perfect Correlation
  - "Duration" and "Duration" got the number 1.000000, which makes sense, each column always has a perfect relationship with itself.
- Good Correlation
  - "Duration" and "Calories" got a 0.922721 correlation, which is a very good correlation, and we can predict that the longer you work out, the more calories you burn, and the other way around: if you burned a lot of calories, you probably had a long work out.
- Bad Correlation
  - "Duration" and "Maxpulse" got a 0.009403 correlation, which is a very bad correlation, meaning that we can not predict the max pulse by just looking at the duration of the work out, and vice versa.



Thanks ☺️

Any Questions?