# Youtube Video 2: Best of Psych Series 2

Tuesday, November 16, 2021 8:31 AM

Shot 1:
- video screen: Quick scroll through of Google Colab document for shortened version of Psych Python data science
- Voice:
  - Introduction from the Brock University Digital Scholarship lab on using the MBTI 16 personality theory to learn how to use Python for Data science.
  - Remind viewers that they can pause the video or hit the j k to rewind 10 seconds if the video is too fast paced.

Shot 2:
- Video screen: navigating to a blank Google colab document
- Voice: describing how viewers can follow along with the video by launching a blank google colab document.

Shot 3:
- video screen:
  - Title: intro to Python libraries
- Voice:
  - none

Shot 4:
- video screen:

The `str` library is so important that it is included all the time Python runs.

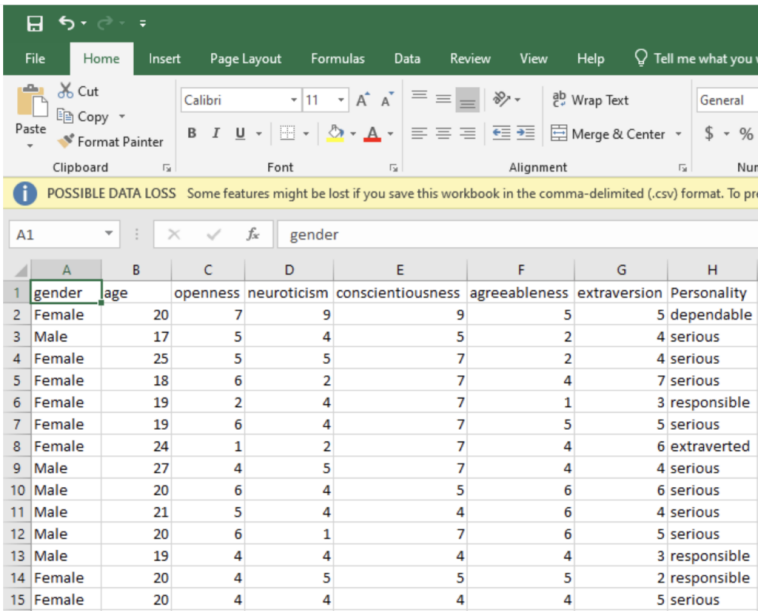
```
all_caps = "HELLO PYTHON USER"

# add .lower() to the following line so that the variable represented by all_caps prints in all lowercase
print(all_caps)
# add .title() to the following line to capitalize the first letter of each word, and the rest lowercase
print(all_caps)
# add .capitalize() to the following line to capitalize only the first letter of the sentence, and the rest lowercase
print(all_caps)
```

- Voice:
  - Description of the string library that loads automatically each time you load Python, and how you add them onto the end of a variable to do different capitalizations.

Shot 5:
- video screen:
  - Title: using functions in libraries to analyze data from a Big 5 personality quiz
- Voice:
  - None

Shot 6:
- video screen:




| | A | B | C | D | E | F | G | H |
|---|---|---|---|---|---|---|---|---|
| 1 | gender | age | openness | neuroticism | conscientiousness | agreeableness | extraversion | Personality |
| 2 | Female | 20 | 7 | 9 | 9 | 5 | 5 | dependable |
| 3 | Male | 17 | 5 | 4 | 5 | 2 | 4 | serious |
| 4 | Female | 25 | 5 | 5 | 7 | 2 | 4 | serious |
| 5 | Female | 18 | 6 | 2 | 7 | 4 | 7 | serious |
| 6 | Female | 19 | 2 | 4 | 7 | 1 | 3 | responsible |
| 7 | Female | 19 | 6 | 4 | 7 | 5 | 5 | serious |
| 8 | Female | 24 | 1 | 2 | 7 | 4 | 6 | extraverted |
| 9 | Male | 27 | 4 | 5 | 7 | 4 | 4 | serious |
| 10 | Male | 20 | 6 | 4 | 5 | 6 | 6 | serious |
| 11 | Male | 21 | 5 | 4 | 4 | 6 | 4 | serious |
| 12 | Male | 20 | 6 | 1 | 7 | 6 | 5 | serious |
| 13 | Male | 19 | 4 | 4 | 4 | 4 | 3 | responsible |
| 14 | Female | 20 | 4 | 5 | 5 | 5 | 2 | responsible |
| 15 | Female | 20 | 4 | 4 | 4 | 4 | 5 | serious |

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 16 | Female | 19 | 7 | 2 | 4 | 3 | 4 lively |

- Voice:
  - A description of what the data looks like in an Excel file, and advantages to using Python instead. Definition of each variable.

Shot 7:

- video screen:
  - Title – Loading the required Python libraries
- Voice:
  - None

Shot 8:

- video screen:

## Loading the required Python Libraries

```
#Load the statistics library
import statistics

#Load the math library
import math

#Load the Library Pandas, that works with data
import pandas as pd

#Load the Library Numpy, that works with numerical calculations
import numpy as np

#These two libraries are often used together!
```

- Voice:
  - Description of import function, description of each of the 4 libraries

Shot 9:

- video screen:

## Uploading the csv file from your computer

```
from google.colab import files
uploaded = files.upload()

#pandas is already loaded, so you can skip importing it

import io

df = pd.read_csv(io.StringIO(uploaded['psyc.csv'].decode('utf-8')))

```

Choose Files psyc.csv

- **psyc.csv**(text/csv) - 9411 bytes, last modified: 11/16/2021 - 100% done

```
Saving psyc.csv to psyc (3).csv
```

- Voice:
  - Describing how to upload a csv file from your computer

Printing the first 10 rows of data to the screen

```
#List the column names so we can use them as variables
df.columns = ["gender","age","openness","neuroticism","conscientiousness","agreeableness","extraversion","Personality"]

df.columns = df.columns.str.title()

df.head(10)
```

| | Gender | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion | Personality |
|---|---|---|---|---|---|---|---|---|
| 0 | Female | 20 | 7 | 9 | 9 | 5 | 5 | dependable |
| 1 | Male | 17 | 5 | 4 | 5 | 2 | 4 | serious |
| 2 | Female | 25 | 5 | 5 | 7 | 2 | 4 | serious |
| 3 | Female | 18 | 6 | 2 | 7 | 4 | 7 | serious |
| 4 | Female | 19 | 2 | 4 | 7 | 1 | 3 | responsible |
| 5 | Female | 19 | 6 | 4 | 7 | 5 | 5 | serious |
| 6 | Female | 24 | 1 | 2 | 7 | 4 | 6 | extraverted |
| 7 | Male | 27 | 4 | 5 | 7 | 4 | 4 | serious |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 8 | Male | 20 | 6 | 4 | 5 | 6 | 6 | serious |
| 9 | Male | 21 | 5 | 4 | 4 | 6 | 4 | serious |

Shot 10:

- video screen:
  - ▾ Printing the first 10 rows of data to the screen

```
#List the column names so we can use them as variables
df.columns = ["gender","age","openness","neuroticism","conscientiousness","agreeableness","extraversion","Personality"]

df.columns = df.columns.str.title()

df.head(10)
```

| | Gender | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion | Personality |
|---|---|---|---|---|---|---|---|---|
| 0 | Female | 20 | 7 | 9 | 9 | 5 | 5 | dependable |
| 1 | Male | 17 | 5 | 4 | 5 | 2 | 4 | serious |
| 2 | Female | 25 | 5 | 5 | 7 | 2 | 4 | serious |
| 3 | Female | 18 | 6 | 2 | 7 | 4 | 7 | serious |
| 4 | Female | 19 | 2 | 4 | 7 | 1 | 3 | responsible |
| 5 | Female | 19 | 6 | 4 | 7 | 5 | 5 | serious |
| 6 | Female | 24 | 1 | 2 | 7 | 4 | 6 | extraverted |
| 7 | Male | 27 | 4 | 5 | 7 | 4 | 4 | serious |
| 8 | Male | 20 | 6 | 4 | 5 | 6 | 6 | serious |
| 9 | Male | 21 | 5 | 4 | 4 | 6 | 4 | serious |

- Voice:
  - Describing how to print the first 10 rows to the screen, and showing how to apply the .title() to this scenario.

Shot 11:

- video screen:
  - ▾ The describe function

```
df.describe()
```

| | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion |
|---|---|---|---|---|---|---|
| count | 315.000000 | 315.000000 | 315.000000 | 315.000000 | 315.000000 | 315.000000 |
| mean | 20.244444 | 4.850794 | 4.584127 | 4.812698 | 4.844444 | 4.926984 |
| std | 2.616811 | 1.537211 | 1.818623 | 1.786315 | 1.718555 | 1.466527 |
| min | 5.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 |
| 25% | 18.000000 | 4.000000 | 3.000000 | 4.000000 | 4.000000 | 4.000000 |
| 50% | 20.000000 | 5.000000 | 5.000000 | 5.000000 | 5.000000 | 5.000000 |
| 75% | 22.000000 | 6.000000 | 6.000000 | 6.000000 | 6.000000 | 6.000000 |
| max | 28.000000 | 8.000000 | 9.000000 | 9.000000 | 8.000000 | 8.000000 |

- Voice:
  - Description of the describe function, like standard deviation, mean, etc.

Shot 12:

- video screen:
  - ▾ Grouping and Counting
    - We also need to gather the entries we need by grouping them together with the `.groupby()` function. We can chain these things together to ask very specific questions of the data.
    - We pass what column we'd like to group the data by
    - We add `.count()` if we are just interested int the counts and not the dataframe

```
[11] df.groupby('Personality')
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x7fd22dd328d0>
```

```
df.groupby("Personality").count()
```

| Personality | Gender | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion |
|---|---|---|---|---|---|---|---|
| dependable | 21 | 21 | 21 | 21 | 21 | 21 | 21 |
| extraverted | 77 | 77 | 77 | 77 | 77 | 77 | 77 |
| lively | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
| responsible | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| serious | 153 | 153 | 153 | 153 | 153 | 153 | 153 |

- Voice:
  - Description of grouping and counting

Shot 13:

- video screen:
  - Example 2: How many people are female in this dataset?

```
1 df.groupby("Gender").count()
```

| Gender | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion | Personality |
|---|---|---|---|---|---|---|---|
| Female | 156 | 156 | 156 | 156 | 156 | 156 | 156 |
| Male | 159 | 159 | 159 | 159 | 159 | 159 | 159 |

- Voice:
  - Explanation that once the code runs, everything is grouped by the gender column, so you can just look to see the row for Female.

Shot 14:

- video screen:

Example 3: How many different ages are in the dataset?

```
1 df.groupby("Age").count()
```

| Age | Gender | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion | Personality |
|---|---|---|---|---|---|---|---|
| 5 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 17 | 29 | 29 | 29 | 29 | 29 | 29 | 29 |
| 18 | 52 | 52 | 52 | 52 | 52 | 52 | 52 |
| 19 | 73 | 73 | 73 | 73 | 73 | 73 | 73 |
| 20 | 41 | 41 | 41 | 41 | 41 | 41 | 41 |
| 21 | 29 | 29 | 29 | 29 | 29 | 29 | 29 |
| 22 | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
| 23 | 24 | 24 | 24 | 24 | 24 | 24 | 24 |
| 24 | 17 | 17 | 17 | 17 | 17 | 17 | 17 |
| 25 | 15 | 15 | 15 | 15 | 15 | 15 | 15 |
| 26 | 7 | 7 | 7 | 7 | 7 | 7 | 7 |
| 27 | 2 | 2 | 2 | 2 | 2 | 2 | 2 |
| 28 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

- Voice:
  - Description of a differently worded question whereby you find the answer the same way.

Shot 15:

- video screen:

## Grouping and applying functions

- If we want to do some math on the data we need to cluster it together a bit. We use `.groupby()` and then apply our mathematical functions to the result
- Here we'll use the following 3 functions:
  - `mean()` finds the arithmetic mean of the data
  - `max()` finds the largest occurence of data in that column
  - `min()` finds the smallest occurennce of data in that column

What is the average extroversion score of people with a personality label of 'extroverted'?

```
1 df.groupby("Personality")["Extraversion"].mean()
```

```
Personality
dependable     5.333333
extraverted    4.636364
lively         3.208333
responsible    4.125000
serious        5.496732
Name: Extraversion, dtype: float64
```

- Voice:
  - Description of how to apply other functions to the groupby function

Shot 16:

- video screen:
  - What is the average Age of people of each Neuroticism score?

```
1 df.groupby("Neuroticism")["Age"].mean()
```

```
Neuroticism
1    19.954545
2    20.678571
3    20.000000
4    20.533333
5    20.031250
6    20.078125
7    20.342857
8    20.888889
9    20.000000
Name: Age, dtype: float64
```

- Voice:
  - Explanation of another example

Shot 17:

- video screen:

## Sorting & Multi line commands

- We can apply sorting to our dataframe actions by using the funciton `.sort_values()`
- We need to give what column we'd like to sort it with `by =`
- We also need to tell it to display it in an increase way `ascending = False`

What Agreeableness score has the most people assigned to it? Here we do it in two steps

```
[ ] 1 by_Agreeableness = df.groupby("Agreeableness").count()
    2
    3 sorted_Agreeableness = by_Agreeableness.sort_values(by = "Personality",ascending = False)
    4
    5 sorted_Agreeableness
```

We could also do it in one step:

```
1 df.groupby("Agreeableness").count().sort_values(by = "Personality",ascending = False)
```

- Voice:
  - Description of how to apply sorting

Shot 18:

- video screen:

## Unique entries & values counts

- Here we use `.unique()` to only give the first instances of the item. Results are returned as a list, which is useful for us later
- This is useful for seeing how many values are in a categorical column

```
[ ] 1 df["Openness"].unique()
```

What are unique values for the Age field?

```
[ ] 1 df["Age"].unique()
```

- To get total number of unique values and frequency in the data we use `value_counts()'

```
[ ] 1 df["Age"].value_counts()
```

- Voice:
  - Explanation of .unique function and value_count

Shot 19:

- video screen:
  -

## Selecting subsets of data

- To make life easier we can create dataframes that just have the values we are interested in

- To make life easier we can create dataframes that just have the values we are interested in
- This is a bit more complicated but follows this type of pattern:

```
dataframe[dataframe[search criteria]]
```

- We are basically creating a subset of the dataframe by matching all entries that match `search criteria`
- That search criteria can be anything that is a conditional
- Doing this gives you a new dataframe

- Voice:
  - Explanation of how you can filter out subsets of data

Shot 20:

- video screen:

EG. A new dataframe of people with an Extroversion score over 6

```
1 over_6 = df[df["Extraversion"] > 6]
2
3 over_6.head(10)
```

| | Gender | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion | Personality |
|---|---|---|---|---|---|---|---|---|
| **3** | Female | 18 | 6 | 2 | 7 | 4 | 7 | serious |
| **17** | Male | 18 | 5 | 3 | 5 | 6 | 7 | serious |
| **38** | Female | 17 | 3 | 7 | 3 | 3 | 7 | extraverted |
| **45** | Male | 24 | 5 | 6 | 3 | 3 | 8 | serious |
| **48** | Female | 25 | 2 | 5 | 4 | 4 | 7 | serious |
| **50** | Male | 17 | 6 | 5 | 6 | 2 | 7 | serious |
| **63** | Male | 19 | 5 | 5 | 4 | 2 | 7 | serious |
| **64** | Female | 18 | 3 | 5 | 4 | 8 | 7 | serious |
| **68** | Male | 21 | 3 | 4 | 6 | 4 | 7 | serious |
| **71** | Female | 24 | 6 | 4 | 4 | 1 | 7 | serious |

- Voice:
  - Explanation of example of creating a subset of data

Shot 21:

- video screen:

EG. If we want the count of participants with an extraversion score greater than 6, we apply the .count() function to what we selected

```
[ ]  1 over_6.count()
```

This can be done in 1 line as well

```
[ ]  1 df[df["Extraversion"] > 6].count()
```

- Voice:
  - Explanation of applying the count function to the subset of data

Shot 22:

- video screen:

Can you 'describe' the newly created dataframe, to get some basic information on the columns in the dataframe?

```
1 over_6.describe()
```

| | Age | Openness | Neuroticism | Conscientiousness | Agreeableness | Extraversion |
|---|---|---|---|---|---|---|
| **count** | 40.000000 | 40.000000 | 40.000000 | 40.000000 | 40.000000 | 40.000000 |
| **mean** | 19.800000 | 4.625000 | 4.825000 | 4.525000 | 4.450000 | 7.200000 |
| **std** | 2.244081 | 1.719981 | 1.737778 | 1.867227 | 1.920871 | 0.405096 |
| **min** | 17.000000 | 1.000000 | 1.000000 | 1.000000 | 1.000000 | 7.000000 |
| **25%** | 18.000000 | 3.000000 | 4.000000 | 3.000000 | 3.000000 | 7.000000 |
| **50%** | 19.000000 | 5.000000 | 5.000000 | 5.000000 | 4.000000 | 7.000000 |
| **75%** | 21.000000 | 6.000000 | 6.000000 | 6.000000 | 6.000000 | 7.000000 |

| | | | | | | |
|---|---|---|---|---|---|---|
| **max** | 25.000000 | 7.000000 | 8.000000 | 8.000000 | 8.000000 | 8.000000 |

- Voice:
  - Explanation of adding the describe function to the subset

Shot 22:

- video screen:

How can we sort our new datafeame?

```
1 over_6.sort_values(by="Agreeableness")
```

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| **231** | Male | 17 | 3 | 7 | 6 | 3 | 7 | serious |
| **223** | Male | 19 | 2 | 5 | 5 | 3 | 7 | serious |
| **311** | Male | 18 | 2 | 5 | 8 | 3 | 7 | dependable |
| **142** | Female | 19 | 6 | 1 | 7 | 3 | 7 | serious |
| **38** | Female | 17 | 3 | 7 | 3 | 3 | 7 | extraverted |
| **45** | Male | 24 | 5 | 6 | 3 | 3 | 8 | serious |
| **101** | Female | 18 | 2 | 7 | 1 | 4 | 8 | extraverted |
| **304** | Female | 19 | 6 | 3 | 2 | 4 | 8 | serious |
| **48** | Female | 25 | 2 | 5 | 4 | 4 | 7 | serious |
| **178** | Male | 23 | 3 | 3 | 5 | 4 | 7 | responsible |
| **68** | Male | 21 | 3 | 4 | 6 | 4 | 7 | serious |
| **3** | Female | 18 | 6 | 2 | 7 | 4 | 7 | serious |
| **139** | Female | 19 | 6 | 5 | 5 | 4 | 7 | serious |
| **213** | Female | 20 | 5 | 8 | 2 | 5 | 7 | extraverted |
| **154** | Male | 19 | 6 | 5 | 4 | 5 | 7 | responsible |
| **167** | Male | 22 | 7 | 7 | 2 | 5 | 7 | responsible |
| **246** | Male | 18 | 6 | 5 | 5 | 5 | 7 | serious |

- Voice:
  - Explanation of how one can sort the new dataset

Shot 23:

- Video screen:
  - Title – Visualizing Data
- Voice:
  - none

Shot 24:

- Video screen:

## Another Library, MatplotLib

Let's take a look at graphing our results. We can use the `matplotlib` library to generate some graphs of our results. We always gives lists as parameters for the graphs

```
1 #This line is for Jupyter's benefit
2 %matplotlib inline
3 #Import MayPlotLib to graph some results
4 import matplotlib.pyplot as plt
```

- Voice:
  - Explanation of the matplotlib library

Shot 25:

- Video screen:
  - Title – Donut charts
- Voice:
  - none

Shot 26:

- Video screen:

+ Code + Text

Let's draw a donut chart of the number of people of each personality label.

```
1 #All of the dependable people
2 Total_dependable = df[df['Personality'] == 'dependable']['Personality'].count()
3 print("Dependable People: " + str(Total_dependable))
4
5 #All of the serious people
6 Total_serious = df[df['Personality'] == "serious"]["Personality"].count()
7 print("Serious People: "+ str(Total_serious))
8
9 #All of the extraverted people
10 Total_extraverts = df[df["Personality"] == "extraverted"]["Personality"].count()
11 print("Extraverts: "+ str(Total_extraverts))
```

```
#All of the lively people
Total_lively = df[df["Personality"] == "lively"]["Personality"].count()
print("Lively People: "+ str(Total_lively))

#All of the responsible people
Total_responsible = df[df["Personality"] == "responsible"]["Personality"].count()
print("Responsible People: "+ str(Total_responsible))

# Matplot lib always wants data in a list, so we'll make one
pie_data = [Total_dependable,Total_serious,Total_extraverts,Total_lively,Total_responsible]
pie_labels = ["Dependable People", "Serious People", "Extraverts", "Lively People", "Responsible People"]
plt.pie(pie_data,labels=pie_labels, colors = ("red", "pink", "blue", "cyan", "purple"))

# Add a circle to create a hole in the pie chart
centre_circle = plt.Circle((0, 0), 0.70, fc='white')
fig = plt.gcf()
fig.gca().add_artist(centre_circle)
```

- Voice:
  - Explanation of all of the code lines to create the donut chart

Shot 27:

- Video screen:
  - Title - Histograms
- Voice:
  - none

Shot 28:

- Video screen:

## Histograms

Say we wanted to plot the personality distribution of our data

```
# bins is the number of containers we'll split our x-axis values into
bins = 5

plt.hist(df["Personality"],bins, color=('red'), alpha=(0.9), hatch="x", edgecolor='white')

plt.title("Personality Distribution", color=(0.2,0.6,0.4,0.6), size=30)
plt.xlabel("Personality", size=20)
plt.ylabel("Occurrences", size=20)

#Set Background colour
plt.gca().set_facecolor('lightblue')
plt.gca().set_axis_on()

#Change the color of the x and y values
ax = plt.gca()
ax.tick_params(axis='x', colors='brown')
ax.tick_params(axis='y', colors='blue')

plt.show()
```

- Voice:
  - Explanation of how to create histograms using matplotlib

Shot 29:

- Video screen:
  - Title – Seaborn Library
- Voice:
  - none

Shot 30:

- Video screen:

## Another Library: Seaborn

Seaborn can do the same charts as Matplotlib, along with correlational charts that visualize correlations between variables.

Install Seaborn

```
pip install seaborn
```

Import Seaborn library, and ensure that numpy, matplotlib, and pandas are also imported

```
#we already imported numpy, so we can skip importing it again
#we already have imported matplotlib, so we can skip importing it again
#we already have pandas imported, so we can skip that step as well!
```

```
#Now all is left to import is Seaborn
import seaborn as sns
```

- Voice:
  - Explanation of installing Seaborn and then importing it, and that if you haven't done so already, import its supporting libraries (numpy, pandas, matplotlib)

Shot 31:

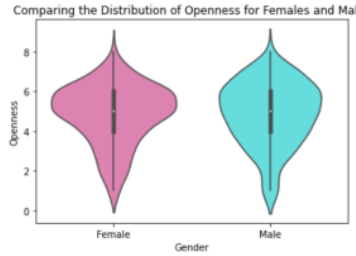
- Video screen:

Create a violin plot to compare distribution between two variables

```
#Create a violin chart
sns.violinplot("Gender", "Openness", data=df,
               palette=["hotpink", "cyan"]);

#Create a title
plt.title('Comparing the Distribution of Openness for Females and Males')
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variables as keyword args: x, y. From version 0.12, the o
  FutureWarning
Text(0.5, 1.0, 'Comparing the Distribution of Openness for Females and Males')
```




- Voice:
  - Explanation of how to create a violin chart using Seaborn library