Demo - Week 1

Imports

```
#imports
import os
import numpy as np   #Numerical Python - For processing numpy arrays
import pandas as pd  #Panel Data - For processing columns and dataframes
import seaborn as sns #just to get the penguin data

# Use pd.set_option to set some configuations:
# Syntax:  pd.set_option(option_name, value)

# Show all columns - otherwise they could be "ellipsed out"
pd.set_option('display.max_columns', None)

# I use the following often to supress scientific notation or control numeric displays
pd.set_option('display.float_format', '{:.3f}'.format)
```

Prep for the csv

This is a bit wonky, but in order to practice using the `pd.read_csv` function you learned, here we load the Palmer Penguins Dataset and save it as a csv into our data folder.

We'll use this famous dataset for today's demo because most of you have likely seen it before and we only have 30 min together.

```
penguins = sns.load_dataset("penguins")
penguins.to_csv("data/penguins.csv")
```

Read in Penguins

Here we use `pd.read_csv` but set an index col since the csv has one.

```
penguins = pd.read_csv("data/penguins.csv", index_col = 0)
penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.100	18.700	181.000	3750.000	Male
1	Adelie	Torgersen	39.500	17.400	186.000	3800.000	Femal
2	Adelie	Torgersen	40.300	18.000	195.000	3250.000	Femal
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.700	19.300	193.000	3450.000	Femal
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.800	14.300	215.000	4850.000	Femal
341	Gentoo	Biscoe	50.400	15.700	222.000	5750.000	Male
342	Gentoo	Biscoe	45.200	14.800	212.000	5200.000	Femal
343	Gentoo	Biscoe	49.900	16.100	213.000	5400.000	Male

To work with the working directory in RStudio, can use the `os` package.

```
#See os functions by commenting in this help call
#help(os)

#Check out the current working dirctory
print(os.getcwd())
```

```
/home/u55d55330-6954-406c-b5b0/covid_01_quarto_python-basics
```

value_counts

There's 2 ways to get the value counts. One way will be deprecated for and gives a warning.

Here we use the function to be deprecated:

```
#Use species column to demo this.
pd.value_counts(penguins['species'])  #This way will  be deprecated
```

```
/tmp/ipykernel_5651/3913864214.py:2: FutureWarning: pandas.value_counts is deprecated and wi
  pd.value_counts(penguins['species'])  #This way will  be deprecated

species
Adelie       152
Gentoo       124
Chinstrap     68
Name: count, dtype: int64
```

Now we use the value_counts method on a column.

So this method is built into a Series object.

```
penguins['species'].value_counts()   #The future way to do value_counts
```

```
species
Adelie       152
Gentoo       124
Chinstrap     68
Name: count, dtype: int64
```

The tutorial uses a list (not a series) to demo `pd.value_counts`.

You cannot call `list_name.value_counts()` because `value_counts` is not a method built into Python's `list` class.

For now, you can still call `pd.value_counts(list_name)`, but that functionality is going away.

Since we don't usually do data analysis on a list, this is likely not a problem that will pop up in your futures.

```
#Basic list - what is a list in Python?
#Processing lists tends to be SLOW....But they can contain different types.

#create a list of multiple types
penguins_list = ["Chinstrap", 3, "Chinstrap", 5.5, "Adelie", "Adelie", "Adelie"]

#try value_counts in our 2 ways
pd.value_counts(penguins_list) #deprecated warning

#penguins_list.value_counts() #comment in and you'll see this line of code
                              #will crash - can only call
                              #value_counts on a column (Series)
```

```
/tmp/ipykernel_5651/2466625501.py:8: FutureWarning: pandas.value_counts is deprecated and wi
  pd.value_counts(penguins_list) #deprecated warning
/tmp/ipykernel_5651/2466625501.py:8: FutureWarning: value_counts with argument that is not n
  pd.value_counts(penguins_list) #deprecated warning
```

```
 Adelie       3
 Chinstrap    2
 3            1
5.500         1
Name: count, dtype: int64
```

```
print("Data type of the list variable:", type(penguins_list))
print("Data type of the column variable:", type(penguins['species']))
```

```
Data type of the list variable: <class 'list'>
Data type of the column variable: <class 'pandas.core.series.Series'>
```

Describe

Practice: how do I get the longest bill_length_mm?

```
penguins['bill_length_mm'].min()
```

```
32.1
```

```
# Describe gives more descriptive stats all at once

penguins['bill_length_mm'].describe()
```

```
count   342.000
mean     43.922
std       5.460
min      32.100
25%      39.225
50%      44.450
75%      48.500
max      59.600
Name: bill_length_mm, dtype: float64
```

np.arange

Time permitting: `np.arange` - helpful

```
np.arange(0,100,4)  #3 parameters:  start, 'the almost end', the skip factor

np.arange(.1,1,.1)
```

```
array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9])
```

Just to demo its helpfulness, we'll use `np.arange` to get different percentiles

```
#np.arange is helpful in many different scenarios
penguins['bill_length_mm'].describe(percentiles=np.arange(.1,1,.1))
```

```
count   342.000
mean     43.922
std       5.460
min      32.100
10%      36.600
20%      38.340
30%      40.200
40%      42.000
50%      44.450
60%      46.000
70%      47.370
80%      49.380
90%      50.800
max      59.600
Name: bill_length_mm, dtype: float64
```

Counting nulls

```
#Selecting just a few cols
penguins = penguins[["species", "island", "bill_length_mm"]]

#Can break onto the next line
(penguins.isna()
         .sum())
```

```
#can run help on a class to find out its methods
#help(pd.Series)
```

```
species           0
island            0
bill_length_mm    2
dtype: int64
```

Group by - For Thurs

```
penguins.groupby("species").agg({"species":"count",
                                 "bill_length_mm":["mean", "max"]})
```

	species	bill_length_mm	
	count	mean	max
species			
Adelie	152	38.791	46.000
Chinstrap	68	48.834	58.000
Gentoo	124	47.505	59.600