week2_demo

```python
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

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

#Load penguins
penguins = sns.load_dataset("penguins")
penguins
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Femal
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Femal
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Femal
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Femal
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Femal
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

Compound Filtering

FYI: Python uses the operators `and` and `or` in many use cases (see below)

But when filtering/subsetting, the operators `&` and `|` are used.

```
day = "Tuesday"
time = 9

if ((day == 'Friday'  and time >= 5) or day == 'Saturday' or day == 'Sunday'):
  print('Sleep In!')

else:
  print('Get up early!')
```

```
Get up early!
```

Also note:

- `.loc` gets brackets because we're indexing, even though it may feel like it should be a method and have parentheses.
- We need to include parentheses in compound expressions. (The tutorial mentions but it's easy to forget.)

Example: The following will crash because it is missing parentheses.

- `penguins.loc[penguins['sex'] == 'Female' &  penguins['flipper_length_mm'] > 190]`

More info: The operator `&` has a higher precedence than `==` so Python interprets the above line of code as the following line of code. Then Python gets confused because it does not know how to compare a String and a Series/column.

- `penguins.loc[penguins['sex'] == ('Female' &  penguins['flipper_length_mm']) > 190]`

```
#one step
penguins.loc[(penguins['sex'] == 'Female') & (penguins['flipper_length_mm'] > 190)]

#This is tough syntax. You can do in 2 steps
#if it makes reading the code easier:

#step 1 create the mask
filter = (penguins['sex'] == 'Female') & (penguins['flipper_length_mm'] > 190)

#Then ask for the filtered table
penguins_f_gt_190 = penguins.loc[filter]
```

Technically, you don't need `loc` to filter rows. It can save you some trouble though.

- Example: `penguins['species']` => returns a column/series (vector)
- Example:
  - `ad_filter = penguins['species'] == 'Adelie'`
  - `penguins[ad_filter]` => performs masking/filtering => very different behavior

```
#Both filter as desired
penguins.loc[penguins['sex'] == 'Female']  #more explicit and safer

penguins[penguins['sex'] == 'Female'] #you'll see this a lot though
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Femal
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Femal
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Femal
6	Adelie	Torgersen	38.9	17.8	181.0	3625.0	Femal
12	Adelie	Torgersen	41.1	17.6	182.0	3200.0	Femal
...	...	...	...	...	...	...	...
332	Gentoo	Biscoe	43.5	15.2	213.0	4650.0	Femal
334	Gentoo	Biscoe	46.2	14.1	217.0	4375.0	Femal
338	Gentoo	Biscoe	47.2	13.7	214.0	4925.0	Femal
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Femal
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Femal

The format for using `.loc[]` is as follows: `data_frame.loc[rows, columns]`.

- The rows parameter is generally a boolean (True/False) expression which evaluates to True for the rows you want to keep
- the columns parameter is generally a list of columns you'd like to keep.
- We can use a colon in either position to indicate we'd like to keep all of the rows/cols

```
#Gets all rows of females penguins and shows all columns from/between species to bill_depth_
penguins.loc[penguins['sex'] == 'Female', 'species':'bill_depth_mm']
```

	species	island	bill_length_mm	bill_depth_mm
1	Adelie	Torgersen	39.5	17.4
2	Adelie	Torgersen	40.3	18.0
4	Adelie	Torgersen	36.7	19.3
6	Adelie	Torgersen	38.9	17.8
12	Adelie	Torgersen	41.1	17.6

	species	island	bill_length_mm	bill_depth_mm
...	...	...	...	...
332	Gentoo	Biscoe	43.5	15.2
334	Gentoo	Biscoe	46.2	14.1
338	Gentoo	Biscoe	47.2	13.7
340	Gentoo	Biscoe	46.8	14.3
342	Gentoo	Biscoe	45.2	14.8

```
#Gets all rows of females penguins and only shows the 2 specified cols
penguins.loc[penguins['sex'] == 'Female', ['species','bill_depth_mm']]
```

	species	bill_depth_mm
1	Adelie	17.4
2	Adelie	18.0
4	Adelie	19.3
6	Adelie	17.8
12	Adelie	17.6
...	...	...
332	Gentoo	15.2
334	Gentoo	14.1
338	Gentoo	13.7
340	Gentoo	14.3
342	Gentoo	14.8

Follow up on loc

Often, folks will try and assign values to a subset of a df without usoing `.loc`. This causes warnings or errors, depending on the version of pandas.

So the following example is a classic pandas gotcha.

```
#create a SD
df = pd.DataFrame(data = {
    'name': ['Alice', 'Bob', 'Charlie', 'David'],
    'age': [25, 30, 22, 28],
    'city': ['New York', 'London', 'Paris', 'Tokyo']
})

#Setting the index col to name
```

```python
df = df.set_index('name')
df
```

	age	city
name		
Alice	25	New York
Bob	30	London
Charlie	22	Paris
David	28	Tokyo

```python
#This correctly gets the rows where age > 25
df[df['age'] > 25]

#This correctly gets the rows where age > 25 and then selects just
#the city col from that subset.
df[df['age'] > 25]['city']

#If we add an equal sign to the above code (like we did below), then
# we're trying to update the city of any row where age is greater than 25 to be DC.
# This gives a pandas warning/error (depending on version of pandas)
df[df['age'] > 25]['city'] = 'DC'
df
```

```
/tmp/ipykernel_60755/4142335128.py:11: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  df[df['age'] > 25]['city'] = 'DC'
```

	age	city
name		
Alice	25	New York
Bob	30	London
Charlie	22	Paris
David	28	Tokyo

However, if we use `loc`, we're essentially telling Python, "I know what I am doing - make the change to the original dataframe please." So no errors/warnings appear.

```
#Not using loc can cause Copy Warnings - Loc gets you out of that

df.loc[df['age'] > 25, 'city'] = 'DC'
df
```

name	age	city
Alice	25	New York
Bob	30	DC
Charlie	22	Paris
David	28	DC

`.iloc`

FYI: `.iloc` ignores index and gets rows via integer locations.

```
#Series
df.iloc[2]
print(type(df.iloc[2]))

print("------------------------------")

#dataframe
df.iloc[[2]]
print(type(df.iloc[[2]]))
```

```
<class 'pandas.core.series.Series'>
------------------------------
<class 'pandas.core.frame.DataFrame'>
```

Can use `iloc` to get the values associated with the max or min.

```
#Get the oldest person by sorting and using iloc
df.sort_values("age", ascending=False).iloc[[0]]
```

	age	city
name		
Bob	30	DC

Discuss: `plotnine` -> do you feel it’s very similar to R?