Getting started with Python!

Sunny Fang
CSC Computing Fellow
yf2610@barnard.edu

Agenda

01. Basic Python syntax

02. Intro to Pandas and data processing

03. Exploratory Data Analysis

Why learn Python?

- **Python**:
  - high-level programming language
  - interpreted language
  - object-oriented
  - free and open source
  - core language for many spatial data tools:

Jupyter Notebook

- **Interactive** IDE for coding
- **Markdown** blocks are great for annotating and commenting longer thoughts
  - Utilize outlines wisely!!!
- Allows conversion to multiple formats:
  - slideshow
  - PDF
  - LaTeX file
  - …and much more

Visualization tells stories

ACTIVITY: Visit **one** of the three links, skim through the article and pay attention to the data visualization. Note one good thing and one thing you'd change from it.

Unequal access to water

tinyurl.com/water-csc

Planet or Plastic?

tinyurl.com/planetorplastic-csc

(email required)

"A Good Life for All"

tinyurl.com/goodlife-csc

Visualization tells stories

ACTIVITY: Visit **one** of the three links, skim through the article and pay attention to the data visualization. Note one good thing and one thing you'd change from it.

Unequal access to water

tinyurl.com/water-csc

Planet or Plastic?

tinyurl.com/planetorplastic-csc

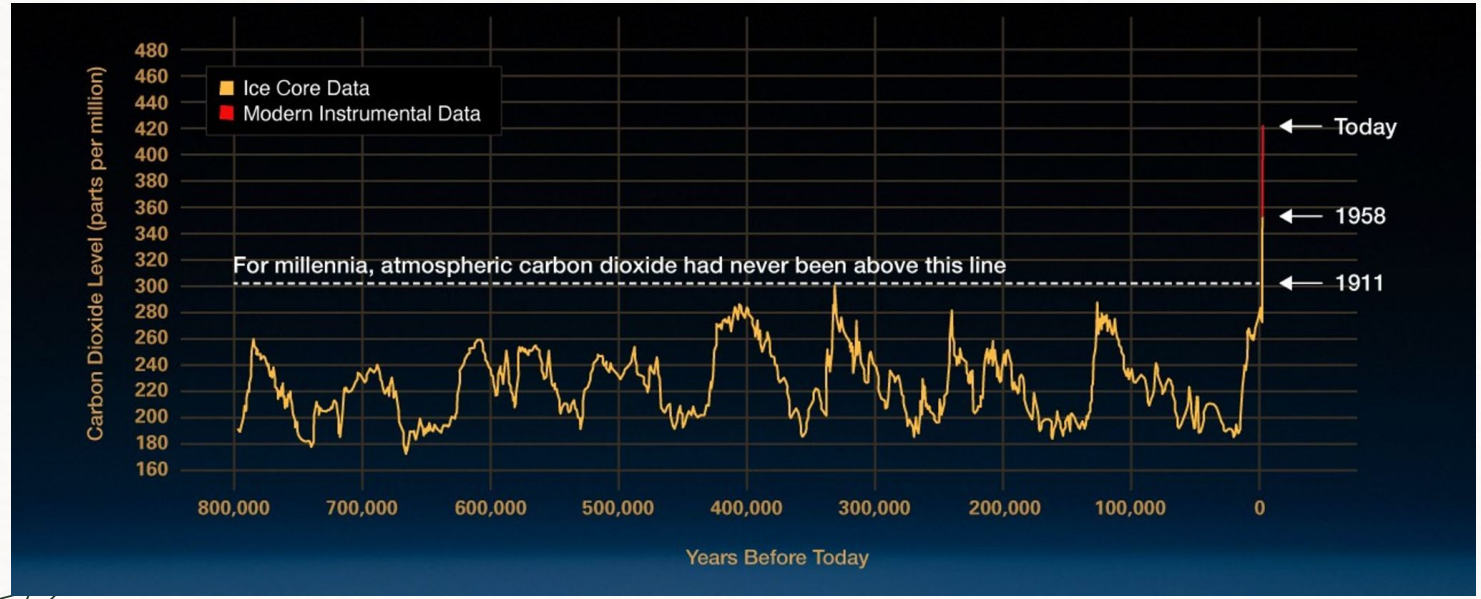
"A Good Life for All"

tinyurl.com/goodlife-csc

YOU CAN DO THIS TOO!!

Power of data

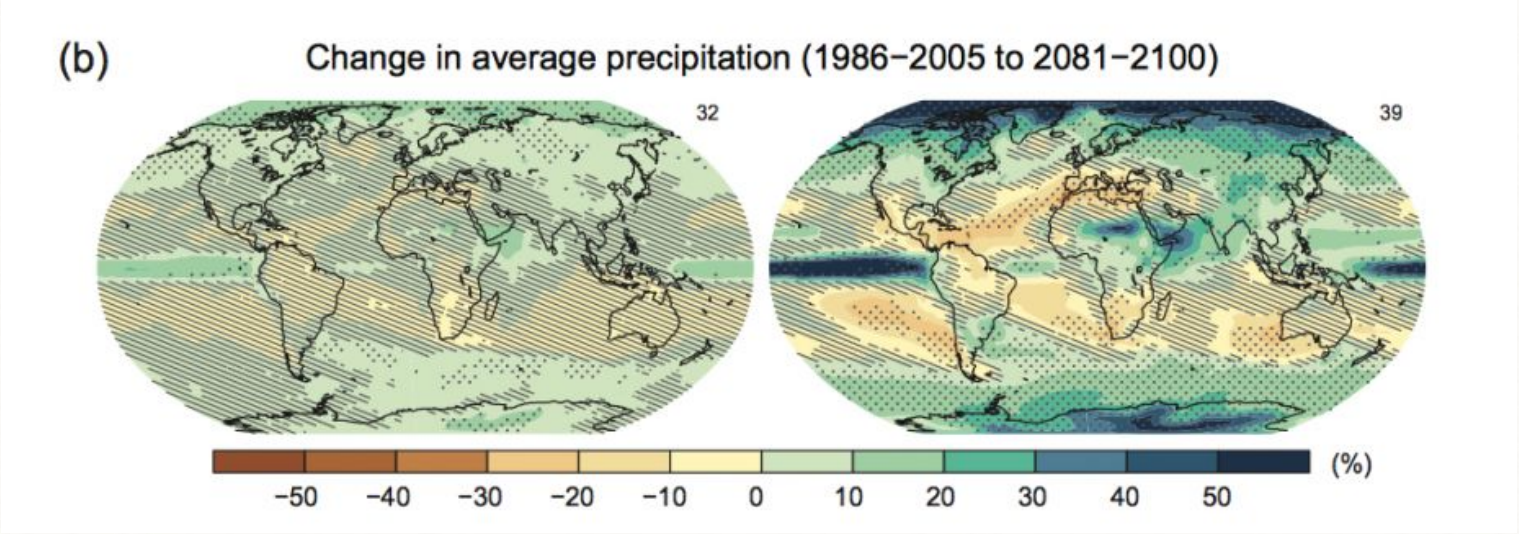
GATHERING INFORMATION: evidence of CO_2 level



Source: https://science.nasa.gov/climate-change/evidence/

Power of data

FORECASTING



Source: https://climatechange.chicago.gov/climate-change-science/future-climate-change

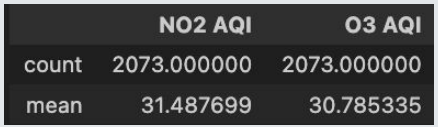
01.

Basic Python Syntax

Variables 101

- **Variable:** used to store information that you want to re-use in your code.
- **Value:** data you want to store in the variable, for example:
  - numeric values
  - filenames
  - paths
  - machine learning models
- **Operator:** used to manipulate or conduct operations on variables

Common data types (non-exhaustive!)

Category	Types	Examples
Text Types	`str`	`"Hello World!"`
Numeric Types	`int`, `float`, `complex`	`2025, 59.4`
Sequence Types	`list`, `tuple`, `range`	`[1, 2, 3]`
Mapping Type	`dict`	`{1: "one", 2: "two"}`
Boolean Type	`bool`	`True/False`
N-dim arrays (e.g., tables)	`numpy.ndarray`, `pandas DataFrame`	

Basic syntax of declaring variables

(what it means)

variable <— ***“assigned to”*** — **value**

variable **“stores”** **value**

(how to code it)

```
temperature = 59
my_name = “Millie”
```

I pawmise: it is not spooky at all!

Let's do some live coding!

tinyurl.com/csc-es-getting-started-colab

Go to `Files` → `Save a copy in Drive`

Print statements

- **“Printing”** in Python means you are displaying the value in your environment
- **syntax:** `print(“some_text”)`
  - `some_text` must be a **string**
- **Example:** I want to print out today’s date and temperature

Print statements

- **Example:** I want to print out today's date and temperature

```
# step 1: define the variables
today_date = "2024-10-31"
# you can also try: today = datetime.today().strftime('%Y-%m-%d')
temperature = 60

# step 2: format the string
# it is crucial to cast “str” data type on any non-string variables
sentence = "Today is " + today_date + ", and the temperature is " +
str(temperature) + " degrees"

# step 3: print!
print(sentence)
```

Print statements

Other ways to format your string!

```
# .format
sentence = 'Today is {}, and the temperature is {} degrees'.format(today_date,
temperature)

# f-strings
sentence = f'Today is {today_date}, and the temperature is {temperature}
degrees'
```

f-strings are

just…meowvelous!

Print statements - Your turn!

ACTIVITY: Print out your name, age, and birthday in the format of **“My name is ___. I am __ years old, and my birthday is ___”**

Print statements - Your turn!

ACTIVITY: Print out your name, age, and birthday in the format of **“My name is ___. I am __ years old, and my birthday is ___”**

```
# step 1: define the variables
name = "Bob"

age = 20

birthday = "2000-01-01"


# step 2: format and print the string
print(f"My name is {name}. I am {age} years old, and my birthday is

{birthday}")
```

Lists and ranges

- **Lists:** collection (sequence) of objects, can contain different data types
  - **mutable,** meaning they can be changed or updated
  - Important: indexing in Python starts at 0
    - (i.e., the **first item** of the list has an **index of 0**)
- **Range:** function that returns a sequence of numbers
  - **syntax:** `range(start, stop, step)`
  - `start`: optional, default 0
  - `stop` : ***required***, last number of your sequence + 1
  - `step` : optional, default 1 step (difference between elements in the sequence)

Operators

Operator	Usage	Example
Arithmetic	to complete mathematical calculations	+, *, /, %, //
Assignment	to assign new values (typically as a result of a arithmetic calculation)	precip_in *= 25.4
Comparison or Relational	to compare operands (e.g., greater than symbol >)	ny_precip_in > nj_precip_in
Identity	to check whether operands are the same	ny_precip_in is not nj_precip_in
Membership	to check whether one operand is contained within another operand	"January" in months
Logical	to check whether operands are true	"January" in months AND "Jan" in months

Operators: Your turn!

ACTIVITY: **Create your own list and calculate the mean**

Operators: Your turn!

ACTIVITY: **Create your own list and calculate the mean**

```
# 1. Create a list called list2 from 4 to 20, with step size = 4
list2 = list(range(4, 21, 4))

# 2. Find the mean of the list using operators only
# Your task is to divide it (/) by the length of the list
temp = 0
for num in list2:
    temp += num
list2_mean = temp / len(list2)
print(list2_mean)
```

02.

Data Processing

Data Analysis Libraries

Pandas provides functionalities that are crucial to data analysis and is suitable for handling multiple file types (e.g., Comma-separated values, or csv, files)

Matplotlib and **Seaborn** are both libraries for data visualization. Seaborn is built on top of matplotlib

Reading file

```
import pandas as pd


df = pd.read_csv("your_file_name.csv")


df.head(5) # first n rows, default 5
```

Reading file

```
import pandas as pd
```

since read_csv is a function in pandas, we need to explicate the library we are using

```
df = pd.read_csv("your_file_name.csv")
```

This is a different case where “df” is your variable, and “head” is one of the many methods you can access from a DataFrame object

```
df.head(5) # first n rows, default 5
```

Understanding the Dataset

```
print(df.shape) # -> rows, columns
print(f"The dataset has {df.shape[0]} rows and
{df.shape[1]} columns\n")


print(df.info()) # -> col_names, values, memory
```

Data Slicing

Syntax:

- `df.loc[row_name, col_name]`
- `df.iloc[row_index, col_index]`

subset by row

```
df.loc[0:4]
```

```
df.iloc[0:4]
```

	State	County	City	Date Local	NO2 Units	NO2 Mean
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333
...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130

Data Slicing

Syntax:

- `df.loc[row_name, col_name]`
- `df.iloc[row_index, col_index]`
- `(col only) df[col_name]`

subset by column

```
df.loc[:,"County"]
```

"all rows"

```
df.iloc[:,2]
```

```
df["County"]
```

	State	County	City	Date Local	NO2 Units	NO2 Mean
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333
...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130

Data Slicing

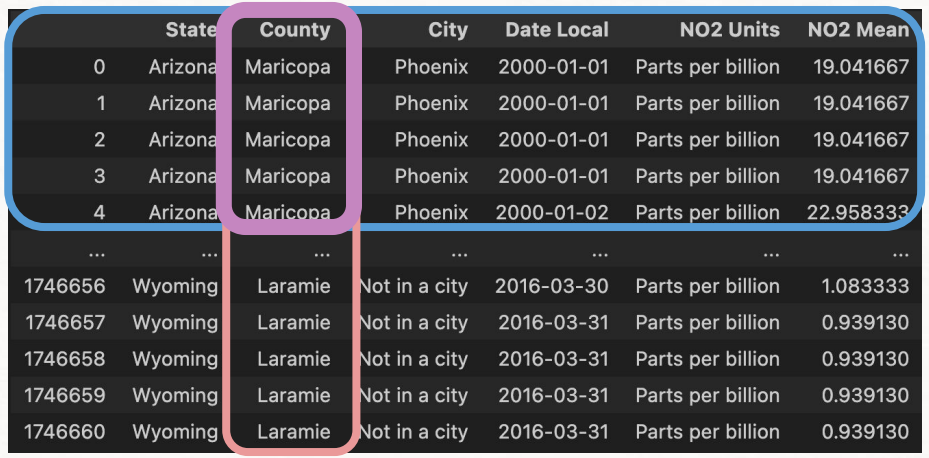
Syntax:

- `df.loc[row_name, col_name]`
- `df.iloc[row_index, col_index]`

subset by **both**

```
df.loc[0:4,"County"]
```

```
df.iloc[0:4, 2]
```



	State	County	City	Date Local	NO2 Units	NO2 Mean
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333
...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130

Data Slicing

Syntax: `df.loc[row_name, [col_names]]`

subset by
multiple columns

```
df.loc[:,["County","Date Local"]]
(or)
df[["County","Date Local"]]
```

this is a list object!!

	State	County	City	Date Local	NO2 Units	NO2 Mean
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333
...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130

	County	Date Local
0	Maricopa	2000-01-01
1	Maricopa	2000-01-01
2	Maricopa	2000-01-01
3	Maricopa	2000-01-01
4	Maricopa	2000-01-02
...	...	...
1746656	Laramie	2016-03-30
1746657	Laramie	2016-03-31
1746658	Laramie	2016-03-31
1746659	Laramie	2016-03-31
1746660	Laramie	2016-03-31

Boolean Masking

Slicing data based on given conditions - recall *operators*!

1. find rows with NO2 mean > 20

```
df[df.loc[:,"NO2 Mean"] > 20]
```

Selects all rows of the column “NO2 Mean”

	State	County	City	Date Local	NO2 Units	NO2 Mean
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333
...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130

Boolean Masking

Slicing data based on given conditions - recall *operators*!

1. find rows with NO2 mean > 20

```
df[df.loc[:,"NO2 Mean"] > 20]
```

Selects the column "NO2 Mean"

Goes through each record and see if it satisfies the condition

	State	County	City	Date Local	NO2 Units	NO2 Mean	
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333	True
...	...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333	False
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False

Boolean Masking

Slicing data based on given conditions - recall *operators*!

1. find rows with NO2 mean > 20

```
df[df.loc[:,"NO2 Mean"] > 20]
```

Only keep the rows that returns True

	State	County	City	Date Local	NO2 Units	NO2 Mean	
0	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
1	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
2	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
3	Arizona	Maricopa	Phoenix	2000-01-01	Parts per billion	19.041667	False
4	Arizona	Maricopa	Phoenix	2000-01-02	Parts per billion	22.958333	True
...	...	...	...	...	...	...	...
1746656	Wyoming	Laramie	Not in a city	2016-03-30	Parts per billion	1.083333	False
1746657	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746658	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746659	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False
1746660	Wyoming	Laramie	Not in a city	2016-03-31	Parts per billion	0.939130	False

Boolean Masking - Your turn!

Slicing data based on given conditions - recall *operators*!

ACTIVITY: Say we are only interested in the **AQI** data in NYC after 2010. How can we go about doing that?

Boolean Masking - Your turn!

Slicing data based on given conditions - recall *operators*!

ACTIVITY: Say we are only interested in the **AQI** data in NYC after 2010. How can we go about doing that?

```
# step 1: create a temporary dataframe with the columns we are interested in
nyc = df[["City", "Date Local", "NO2 AQI", "O3 AQI", "County"]]


# step 2: apply boolean masking
nyc = nyc[(nyc["City"] == "New York") & (nyc["Date Local"] > "2010-12-31")]
```

Save your files!!!!

```
your_df.to_filetype(filename)
```

```
nyc.to_csv('filename.csv')
```

if using colab:

```
from google.colab import files
files.download('filename.csv')
```

03.

Exploratory Data Analysis

Exploratory Data Analysis

- **Exploratory Data Analysis (EDA)** is a process that summarize their main characteristics of a dataset
  - involves *descriptive statistics* (median, mean, mode and measures of variability)
  - often utilizes data visualization techniques
- Data types
  - **Numeric**: magnitude (e.g., temperature)
  - **Categorical**: no order (e.g., states)
  - **Ordinal**: ordered, (e.g., education level)

Before we start…

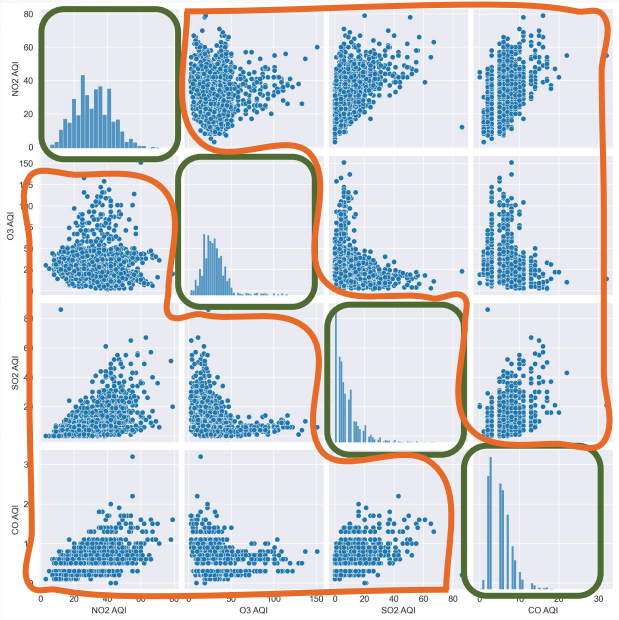
- `df.dropna(inplace=True)` drops all NA values in the dataset
- `df.info()` tells you the data type for each column
  - if int or float ⇒ numerical
  - otherwise ⇒ usually *ordinal* or *categorical*

Exploratory Data Analysis: Numeric

- `df.describe()` returns the count, mean, std, min, 25%, 50%, 75%, and max of **ALL** numerical columns; or you can get individual metrics by:
  - **Get central tendency measures**
    - `df['col_name'].mean()` ⇒ mean
    - `df['col_name'].median()` ⇒ median
    - `df['col_name'].median()` ⇒ mode
    - `df['col_name'].quantile([n])` ⇒ returns the n*100% percentile
  - **Get spread**
    - `df['col_name'].std()` ⇒ standard deviation
    - `df['col_name'].var()` ⇒ variance

Exploratory Data Analysis: Numeric

- `sns.pairplot()` returns (1) the **univariate distribution** and (2) the **bivariate scatterplot** between two variables



Aside: a note on Dots per Inch (DPI)

- Ever feel like your plot looks fuzzy? Try increasing the DPI!
- **caveat**: takes up more memory when saving files

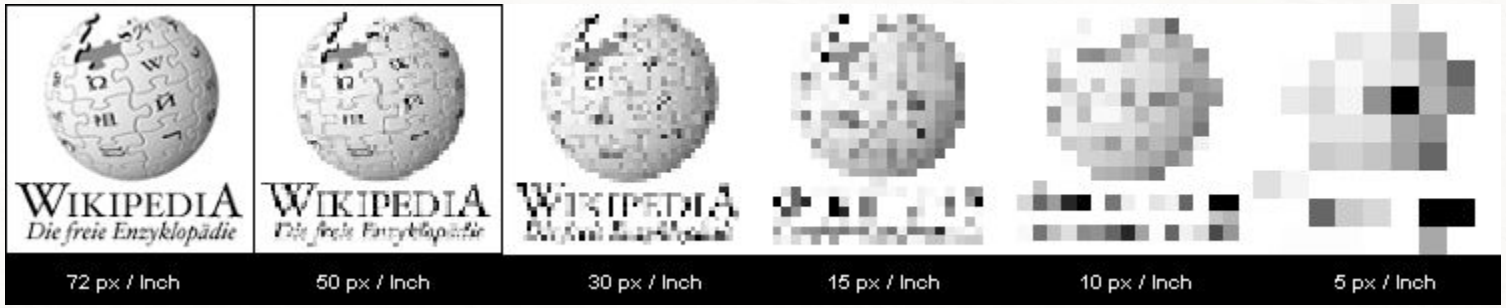


Image attribution: Creative Commons Attribution-Share Alike 3.0 Unported license.

Data Visualization: Numeric

- Before we jump into coding…**pause and think about:**
  1. What is the question I am trying to answer?
  2. What type of visualization suits my question the most?

     ⇒ tinyurl.com/choose-best-graph
  3. What are my x and y-axes?
  4. What do I want to highlight in my graph? (Do I need a caption? Annotations?)

Data Visualization: Numeric

(Generally) good steps to follow on plotting

```
# declare a figure object
fig, ax = plt.subplots(1, 1, figsize=(10,4), dpi = 200)

# plot with Seaborn
sns.boxplot(x="IV", y = "DV",
     data=your_dataset,
     label=legend)

# set title for the plot
ax.set_title("Name of your plot")

# save figure
plt.savefig("your_filename.png")
```

DPI determines the quality of your plot

1. Declare a fig, ax object
2. Plot with plt. or sns.
3. Add elements as needed
4. [optional but recommended] save your figures!

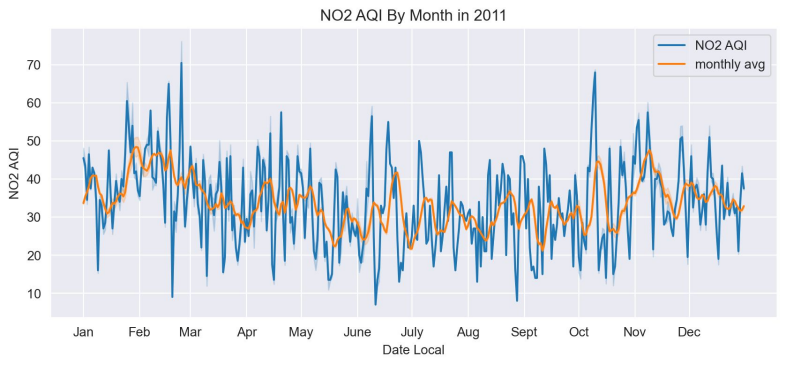
Data Visualization: Numeric

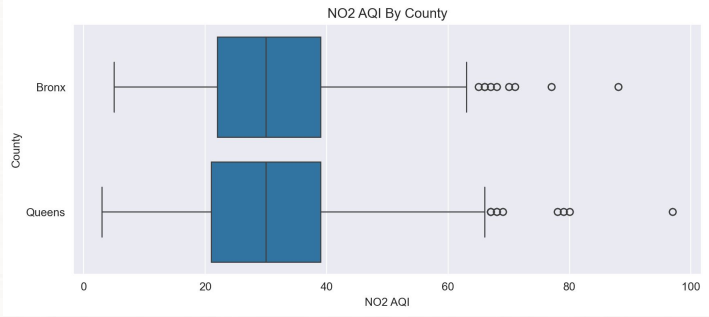
ACTIVITY: Your turn! Let's say we are interested in understanding the **distribution of NO_2** (for example, before doing a t-test against a hypothesized value), how can we create a plot for the purpose?

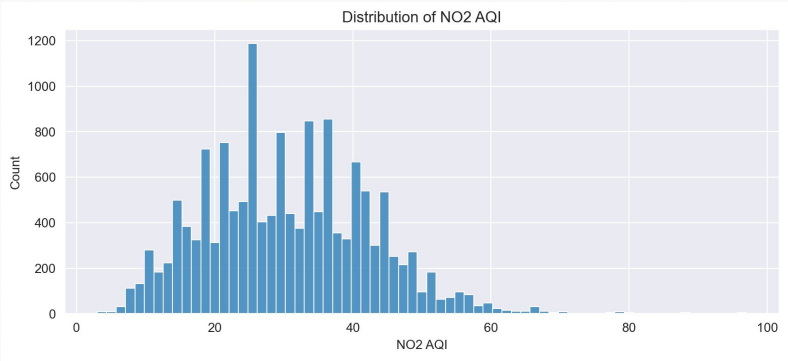
Data Visualization: Numeric

ACTIVITY: Your turn! Let's say we are interested in understanding the **distribution of NO_2** (for example, before doing a t-test against a hypothesized value), how can we create a plot for the purpose?

Data Visualization: Numeric







Some amazing plots we made together today! Are you ready to make more on your own?

What we learned together today

- What is **Python**
- The power of **data visualization**
- Basic Python syntax
  - what are **variables, values, data types**
  - **print** statements
- Data Processing
  - **pandas:** subsetting, slicing, and filtering data
  - **matplotlib:** creating figures
  - **seaborn:** make figures customizable
  - exploratory data analysis on *numeric* data

Before we go...some tips on coding

```
# step 1: create a temporary dataframe with the columns we are interested in
# which are City, Date Local, NO2 AQI, O3 AQI, CO AQI, SO2 AQI, and County
nyc = df[["City", "Date Local", "NO2 AQI", "O3 AQI", "SO2 AQI", "CO AQI","County"]]

# step 2: we can filter by NYC
nyc = nyc[nyc["City"] == "New York"]
display(nyc)

# step 3: Lastly, trim to keep everything after 2010 (i.e., start on 2011-01-01)
nyc = nyc[nyc["Date Local"] > "2010-12-31"]

# always a good idea to check the data
display(nyc.head())

# save file
nyc.to_csv("nyc_since2011.csv")
```

Document, document, document! The MAIN person you are writing comments for is **YOUR FUTURE SELF!**

Before we go...some tips on coding

```
# step 1: create a temporary dataframe with the columns we are interested in
# which are City, Date Local, NO2 AQI, O3 AQI, CO AQI, SO2 AQI, and County
nyc = df[["City", "Date Local", "NO2 AQI", "O3 AQI", "SO2 AQI", "CO AQI","County"]]

# step 2: we can filter by NYC
nyc = nyc[nyc["City"] == "New York"]
display(nyc)

# step 3: Lastly, trim to keep everything after 2010 (i.e., start on 2011-01-01)
nyc = nyc[nyc["Date Local"] > "2010-12-31"]

# always a good idea to check the data
display(nyc.head())

# save file
nyc.to_csv("nyc_since2011.csv")
```

When naming variables, make it intuitive (e.g., would it make sense to name the new df “x”?)

Before we go...some tips on coding

```
# step 1: create a temporary dataframe with the columns we are interested in
# which are City, Date Local, NO2 AQI, O3 AQI, CO AQI, SO2 AQI, and County
nyc = df[["City", "Date Local", "NO2 AQI", "O3 AQI", "SO2 AQI", "CO AQI","County"]]

# step 2: we can filter by NYC
nyc = nyc[nyc["City"] == "New York"]
display(nyc)

# step 3: Lastly, trim to keep everything after 2010 (i.e., start on 2011-01-01)
nyc = nyc[nyc["Date Local"] > "2010-12-31"]

# always a good idea to check the data
display(nyc.head())

# save file
nyc.to_csv("nyc_since2011.csv")
```

Print statements/ Display are your best friend for sanity checks!

Before we go...some tips on coding

```
# step 1: create a temporary dataframe with the columns we are interested in
# which are City, Date Local, NO2 AQI, O3 AQI, CO AQI, SO2 AQI, and County
nyc = df[["City", "Date Local", "NO2 AQI", "O3 AQI", "SO2 AQI", "CO AQI","County"]]

# step 2: we can filter by NYC
nyc = nyc[nyc["City"] == "New York"]
display(nyc)

# step 3: Lastly, trim to keep everything after 2010 (i.e., start on 2011-01-01)
nyc = nyc[nyc["Date Local"] > "2010-12-31"]

# always a good idea to check the data
display(nyc.head())

# save file
nyc.to_csv("nyc_since2011.csv")
```

After you make edits to a dataframe (e.g., subset, merge, etc), save it as a new file for easy access!

Open Discussion

- What makes a good data visualization?
- When making visualizations, how do I balance aesthetics and readability?

Questions?

To review, check out the repo here:
tinyurl.com/
csc-es-getting-started-github