



# Data Wrangling (DSC 540)



DR. BRIDGETTE TOWNSEND

MARCH 11 – MAY 4, 2024

8-WEEK COURSE

---

---

# Week 1:

## Data Wrangling Introduction

# Course Resources

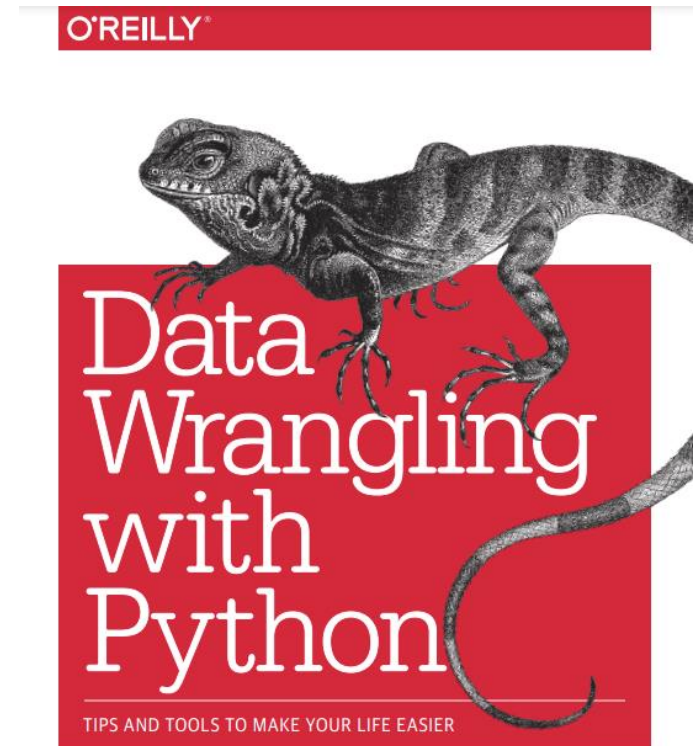
---

## Textbook

- Data Wrangling with Python by J. Kazil & K. Jarmul

## Development Environment

- Python Programming Language
- Jupyter Notebook



Jacqueline Kazil & Katharine Jarmul

# Assignments

---

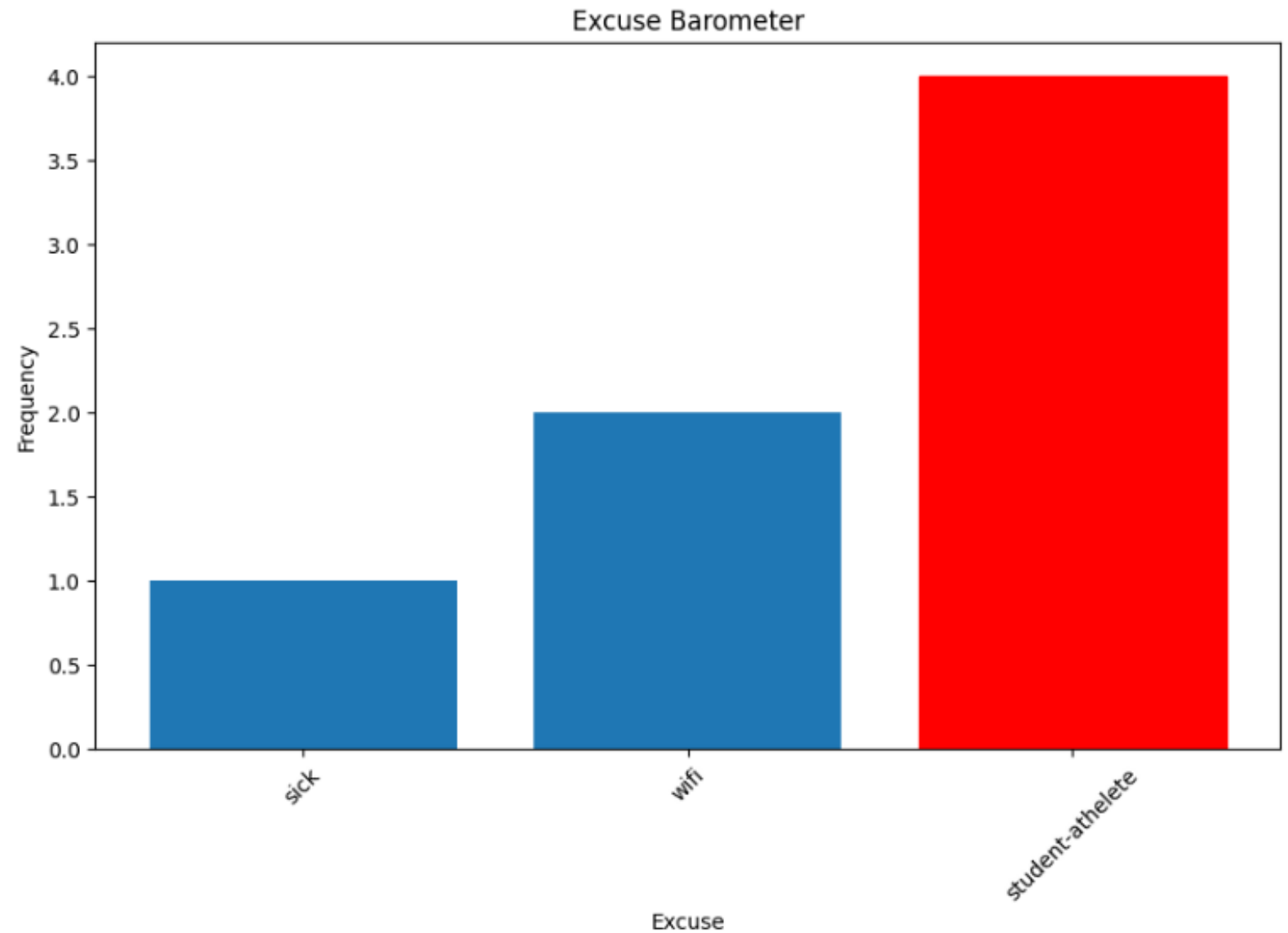
Case Studies

Programming Assignments

Exams (mid-term and final)

“Student-Athletes are responsible for communicating with their professors prior to their absences from class and upon their return to class. Student-Athletes are also responsible for any missed class work or assignments”

**Coaching Staff**



# Keep in mind

---

You are currently a student preparing yourself to be a software engineer, data scientist, information technology professional, machine learning engineer, artificial intelligence engineer, etc.

Do not take your education for granted, you are an exception to the rule.

**IF IT IS IMPORTANT  
TO YOU, YOU WILL  
FIND A WAY.**

**IF NOT  
YOU'LL FIND  
AN EXCUSE**

# Why Code

Start with the full context

The thing you are doing, you are doing properly

Deliberate practice

As a result:

- Writing code
- Not theory
- Practical applications

Think when you starting talking. You learned about nouns, pronouns, and adjectives later...well after you learned to talk.



When [Harvard Graduate School of Education](#) Professor [David Perkins](#) looked back at his childhood little league and "backyard baseball" experiences, he found the perfect metaphor for the set of teaching concepts presented in his 2008 book, [Making Learning Whole: How Seven Principles of Teaching Can Transform Education](#).



# What is Data Wrangling

---

Also called data cleaning, data remediation, or data munging—refers to the process of cleaning, transforming, and organizing raw data into a more structured and usable format for analysis

Examples:

- Merging multiple data sources into a single dataset
- Identify gaps in data (i.e., empty cells in a excel) then fix (i.e. fill, delete)
- Remove any unnecessary data
- Identify outliers in data (i.e. keep, remove, explain)

The goal is to make the data reliable and accessible for exploration



# Andrej Karpathy famous blog post “A Recipe for Training Neural Networks”

---

Becoming “one with the data” is an essential step for training great models!

Recommended reading: <https://karpathy.github.io/2019/04/25/recipe/>

# Data Wrangling Process

---

## Clean

- Remove inaccurate or inconsistent data

## Structure

- Transform raw data

## Missing Values

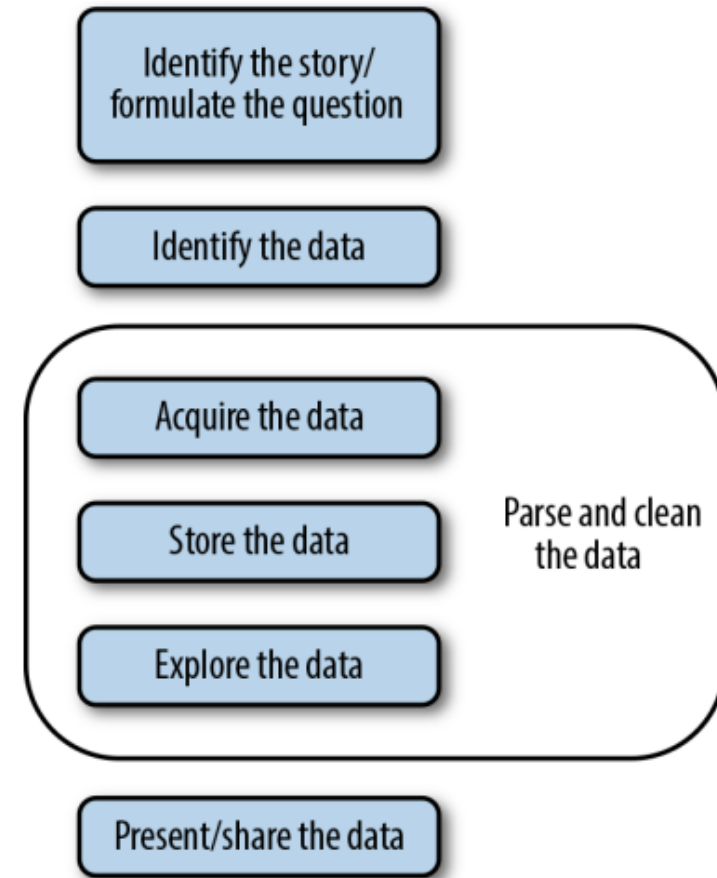
- Remove, reformat or correct

## Merge/Join

- Combine data from multiple source

## Duplicates

- Remove duplicate data



# Data Formats

---

There are various formats and file types.

Machine-readable file formats

- designed for easy interpretation by machines
- commonly known as machine-readable formats

Selecting a format depends on the use case and/or application requirements..

Common Machine Readable Formats:

- CSV (Comma-Separated Values):
  - plain text format, each line represents a row of data, and values within a row are separated by commas
  - Ex., Similar to Excel Spreadsheets
    - Files will have a .csv extension
- JSON (JavaScript Object Notation)
  - key-value pair structure
  - Ex., web applications, configuration files, data transfer files
    - Files will have a .json extension
- XML (eXtensible Markup Language)
  - uses tags to define elements and relationships
  - Ex., web applications
    - Files will have a .xml extension

# Creating a home for code and files

Save the code and files locally and in your working folder/directory

- Ex. /home/jupy/dataWrangler
  - Note the .ipynb, .csv, .json, .xml are in the same folder/directory
  - Dataset files will then be accessible to your development environment
  - In Google Colab
    - Save file to google drive. They are not saved when you instance ends.



The screenshot shows the JupyterLab interface with the address bar displaying 'localhost:8888/tree/dataWrangler'. The 'jupyter' logo is visible, along with 'Quit' and 'Logout' buttons. Below the logo, there are tabs for 'Files', 'Running', and 'Clusters'. A message says 'Select items to perform actions on them.' with 'Upload', 'New', and a refresh icon. The file browser shows a directory structure with a dropdown menu set to '0' and a folder icon for 'dataWrangler'. The table below lists the contents of the 'dataWrangler' directory:

	Name	Last Modified	File size
	..	seconds ago	
<input type="checkbox"/>	week_1_intro.ipynb	Running 18 minutes ago	590 B
<input type="checkbox"/>	data-text.csv	18 minutes ago	635 kB
<input type="checkbox"/>	data-text.json	18 minutes ago	1.65 MB
<input type="checkbox"/>	data-text.xml	18 minutes ago	2.5 MB

# Remember: Data is not saved in Google Colab

---

## Warning

Ensure that your files are saved elsewhere. This runtime's files will be deleted when this runtime is terminated. [More info](#)

OK

# CSV Data

CSV files separate data columns with commas.

The dataset we will be using comes from the World Health Organization (WHO)

- contains life expectancy rates worldwide by country

Best way to open a csv file is in excel or Google Spreadsheets

Another option is to open the csv file in a text editor, you will see something like the following:

```
1 "Indicator","PUBLISH STATES","Year","WHO region","World Bank income group","Country","Sex","Display Value","Numeric","Low","High","Comments"
2 "Life expectancy at birth (years)","Published","1990","Europe","High-income","Andorra","Both sexes","77","77.00000","","",""
3 "Life expectancy at birth (years)","Published","2000","Europe","High-income","Andorra","Both sexes","80","80.00000","","",""
4 "Life expectancy at age 60 (years)","Published","2012","Europe","High-income","Andorra","Female","28","28.00000","","",""
5 "Life expectancy at age 60 (years)","Published","2000","Europe","High-income","Andorra","Both sexes","23","23.00000","","",""
6 "Life expectancy at birth (years)","Published","2012","Eastern Mediterranean","High-income","United Arab Emirates","Female","78","78.00000","","",""
7 "Life expectancy at birth (years)","Published","2000","Americas","High-income","Antigua and Barbuda","Male","72","72.00000","","",""
8 "Life expectancy at age 60 (years)","Published","1990","Americas","High-income","Antigua and Barbuda","Male","17","17.00000","","",""
9 "Life expectancy at age 60 (years)","Published","2012","Americas","High-income","Antigua and Barbuda","Both sexes","22","22.00000","","",""
10 "Life expectancy at birth (years)","Published","2012","Western Pacific","High-income","Australia","Male","81","81.00000","","",""
11 "Life expectancy at birth (years)","Published","2000","Western Pacific","High-income","Australia","Both sexes","80","80.00000","","",""
12 "Life expectancy at birth (years)","Published","2012","Western Pacific","High-income","Australia","Both sexes","83","83.00000","","",""
13 "Life expectancy at birth (years)","Published","2012","Europe","High-income","Austria","Female","83","83.00000","","",""
14 "Life expectancy at age 60 (years)","Published","2012","Europe","High-income","Austria","Female","25","25.00000","","",""
15 "Life expectancy at birth (years)","Published","2012","Europe","High-income","Belgium","Female","83","83.00000","","",""
16 "Life expectancy at birth (years)","Published","2000","Eastern Mediterranean","High-income","Bahrain","Male","73","73.00000","","",""
17 "Life expectancy at birth (years)","Published","1990","Eastern Mediterranean","High-income","Bahrain","Female","74","74.00000","","",""
18 "Life expectancy at age 60 (years)","Published","1990","Eastern Mediterranean","High-income","Bahrain","Male","17","17.00000","","",""
```

# How to Import CSV Data

---

All code is in Python

- There are many programming languages to choose from, so why use Python?
  - Python is easy and handles data wrangling tasks in a simple and straightforward way

Code Explanation:

- **Line 1:** imports the csv library
- **Line 3:** opens the csv file to read the contents
  - This file should be located in the same folder as the .ipynb
- **Line 6:** A for loop to iterate over Python objects
  - A for loop says, “For each thing in this list of things, do something.”
    - In this case, read each row
- **Line 7:** Print the contents in each row

```
In [4]: 1 import csv
        2
        3 with open('data-text.csv', 'r', newline='') as csvfile:
        4     reader = csv.reader(csvfile)
        5
        6     for row in reader:
        7         print(row)
        8
```

# Iterating over Python objects

---

In a for loop, the first word after "for" specifies the variable holding each object in the iterable.

Iterable is an object which can be looped over or iterated over with the help of a for loop.

Objects like lists, tuples, sets, dictionaries, strings, etc. are called iterables.

In this for loop, we store and print each dog's name using the variable 'dog' until all names are processed, completing the code.

In [7]:

```
1 dogs = ['Joker', 'Simon', 'Ellie', 'Lishka', 'Fido']
2 |
3 for dog in dogs:
4     print(dog)
```

## Code Explanations:

- Line 1: creates a list named dogs containing five strings, each representing the name of a dog.
- Line 3: iterates over each element (dog name)
- Line 4: during each iteration, it prints the name of the current dog.



# JSON Data

---

JSON is widely used for data transfers , it is clean, readable, and easy to parse.

A popular format for websites, often used to transmit data to JavaScript.

Each row consists of a key-value pair separated by :, entries are delimited (separated) by commas, enclosed within opening and closing curly braces ({}).

```
1  [  
2    {  
3      "Indicator":"Life expectancy at birth (years)",  
4      "PUBLISH STATES":"Published",  
5      "Year":1990,  
6      "WHO region":"Europe",  
7      "World Bank income group":"High-income",  
8      "Country":"Andorra",  
9      "Sex":"Both sexes",  
10     "Display Value":77,  
11     "Numeric":77.00000,  
12     "Low":"","  
13     "High":"","  
14     "Comments":""  
15   },
```

# How to Import JSON Data

---

In the CSV file, we did not call *read*.

What is the difference?

In the CSV example, we opened the file in read-only mode, but in the JSON example, we are reading the contents of the file into a variable named `json_data`.

Code Explanations:

- Line 1: Imports the *json* Python library, which we will use to process the JSON data
- Line 3: Open file's *read* method, read the file and store in the variable, *json\_data*
- Line 5: Use `json.loads()` to load JSON into Python, save output in the variable named *data*.
- Line 7: Iterate over the *data* using a for loop
- Line 8: Print each *item*

```
In [15]: 1 import json
          2
          3 json_data = open('data-text.json').read()
          4
          5 data = json.loads(json_data)
          6
          7 for item in data:
          8     print(item)
          9
```

# XML Data

---

XML is both human and machine readable.

However, CSV and JSON examples are easier to preview and understand in comparison to XML data.

Metadata, Display and Attribute are all examples of tags.

- Tags (or nodes) store data in a hierarchical and structured way
- Data values are stored between tags:
  - Ex. <Display>IOC</Display>
  - Display tags, where the value for the tag is IOC

```
-<Metadata>
  -<Dataset Label="CYCU">
    <Display>COUNTRY_YEARLY_CORE_UNITS</Display>
  </Dataset>
  -<Attribute Label="DS" EntityType="CORE_DIMENSION" Entity="COUNTRY">
    <Display>DS</Display>
  </Attribute>
  -<Attribute Label="FIPS" EntityType="CORE_DIMENSION" Entity="COUNTRY">
    <Display>FIPS</Display>
  </Attribute>
  -<Attribute Label="IOC" EntityType="CORE_DIMENSION" Entity="COUNTRY">
    <Display>IOC</Display>
  </Attribute>
  -<Attribute Label="ISO2" EntityType="CORE_DIMENSION" Entity="COUNTRY">
    <Display>ISO2</Display>
  </Attribute>
  -<Attribute Label="ISO" EntityType="CORE_DIMENSION" Entity="COUNTRY">
    <Display>ISO</Display>
  </Attribute>
```

# How to Import XML Data

Importing XML files is slightly more complex than the CSV and JSON.

## Code Explanation:

- Line 1: imports the ElementTree library for parsing XML
- Line 3: Parse the XML file using ET.parse method
- Line 4: Get at the root of the XML tree
- Line 5: Find the 'Data' element within the XML
- Line 6: Initialize an empty list all\_data to store the extracted data
- Line 8: Iterate through each 'observation' element within 'Data'
- Lines 9-19: For each 'item' element within the 'observation':
  - Get the first attribute key, list(item.attrib.keys())[0].
  - Assign key and value based on whether or not the lookup key is Numeric.
  - Add the key-value pair to the record dictionary.
- Line 21: print the final list stored in 'all\_data'

```
In [16]: 1 from xml.etree import ElementTree as ET
2
3 tree = ET.parse('data-text.xml')
4 root = tree.getroot()
5 data = root.find('Data')
6 all_data = []
7
8 for observation in data:
9     record = {}
10    for item in observation:
11        lookup_key = list(item.attrib.keys())[0]
12        if lookup_key == 'Numeric':
13            rec_key = 'NUMERIC'
14            rec_value = item.attrib['Numeric']
15        else:
16            rec_key = item.attrib[lookup_key]
17            rec_value = item.attrib['Code']
18        record[rec_key] = rec_value
19    all_data.append(record)
20
21 print(all_data)
22
```

# Summary

---

The ability to handle machine-readable data formats is a critical skill for data wrangling (aka data cleaning, data remediation, or data munging)

CSV, JSON, and XML are data formats that have different structures, parsing methods, and use cases.

- CSV is simple and suitable for tabular data
- JSON is versatile and supports complex data structures
- XML is customizable, you can define tags according to a specific needs
  - Extensible and Hierarchical

The choice of format depends on the specific requirements and characteristics of the data

We also were introduced to Python concepts such as:

- Importing files using import
- How to read and open files
- Iterating over for loops
- Running Python code in Jupyter Notebook

# Homework #1

---

## Programming Assignment

- Importing CSV, JSON and XML
- Each program was covered in class, code is provided
- Submit into canvas
  - code and output
  - save as a PDF

All Homework is Due Sunday at 11:59pm

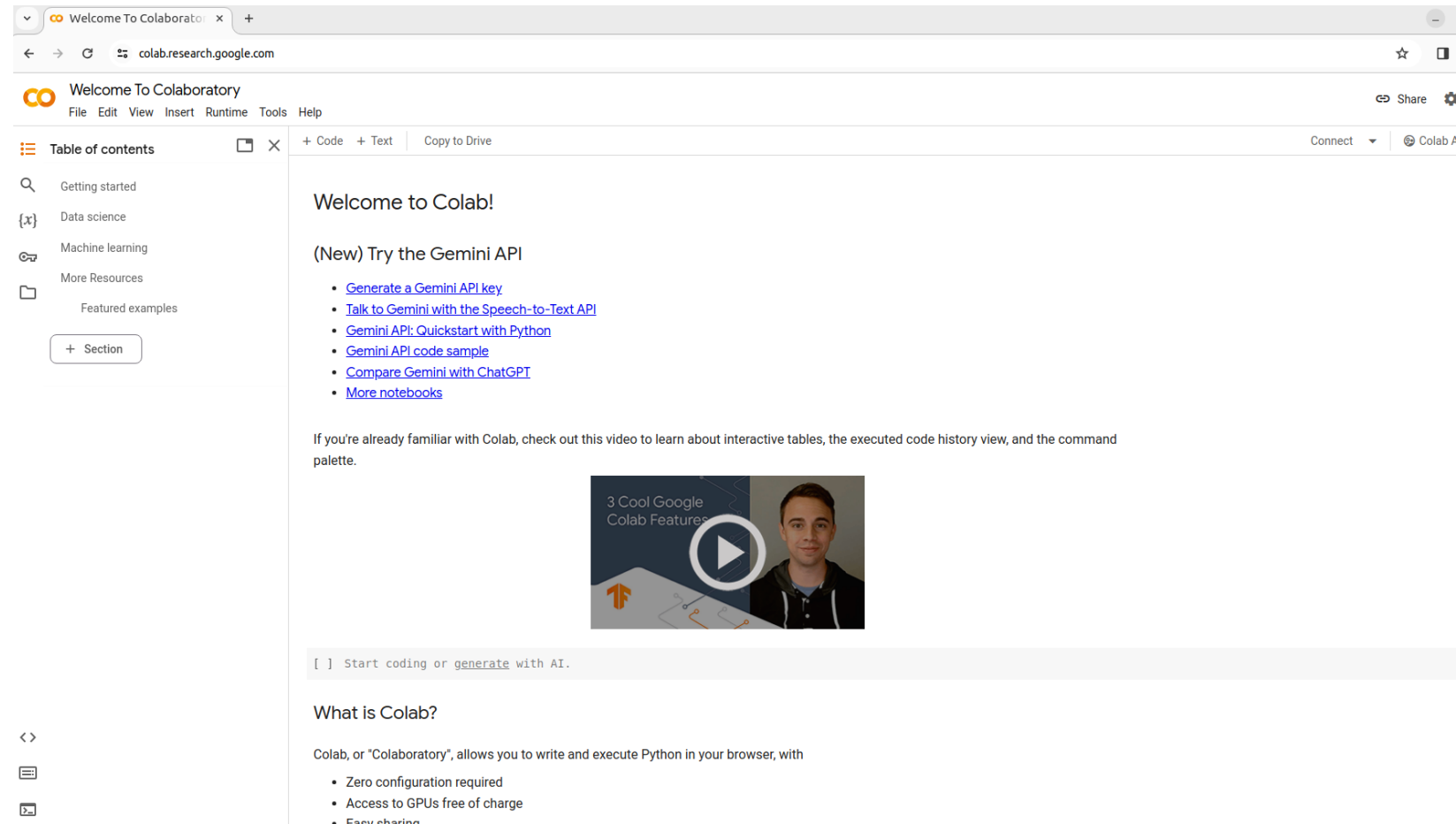
Set up google colab account to use jupyter notebook

# Getting Started with Python and Jupyter Notebook

---

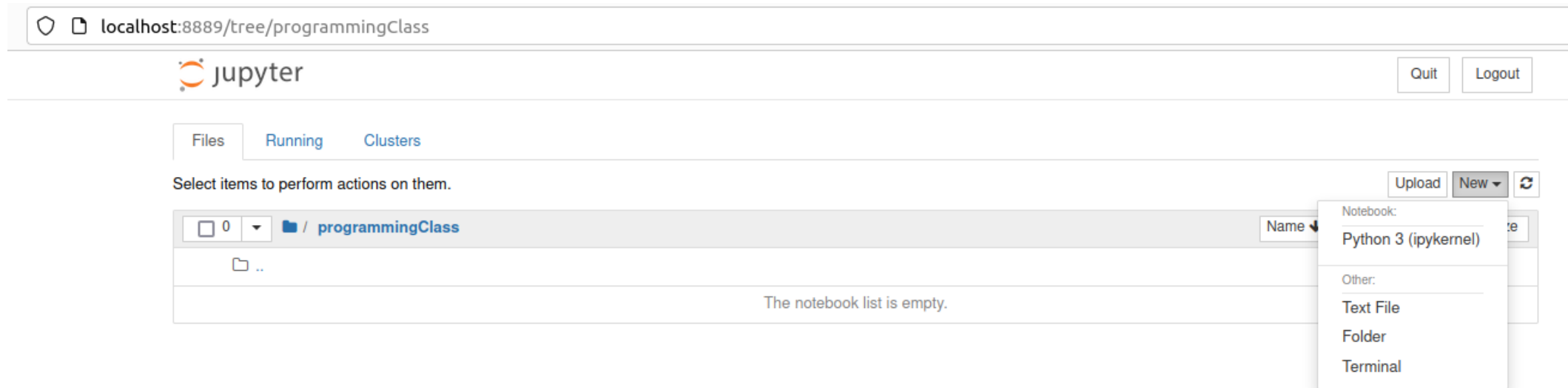
- Python is a general-purpose programming language
  - Not only is it simple and easy to learn, it has a large collection of libraries to support data science projects
- Jupyter Notebook is an interactive web-based computing environment

# Setting up Google Colab – submit screenshot of dashboard






# Starting a notebook



# Opening Notebook

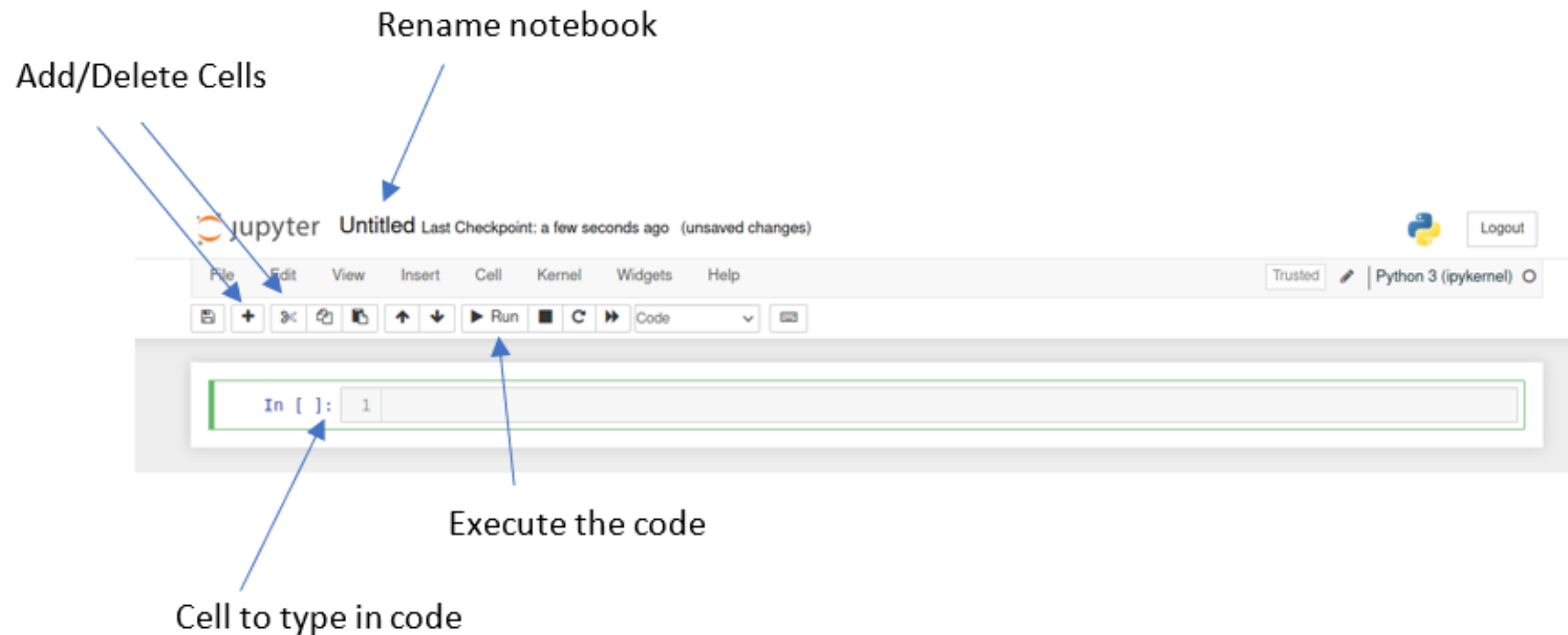
 Quit Logout

Files Running Clusters

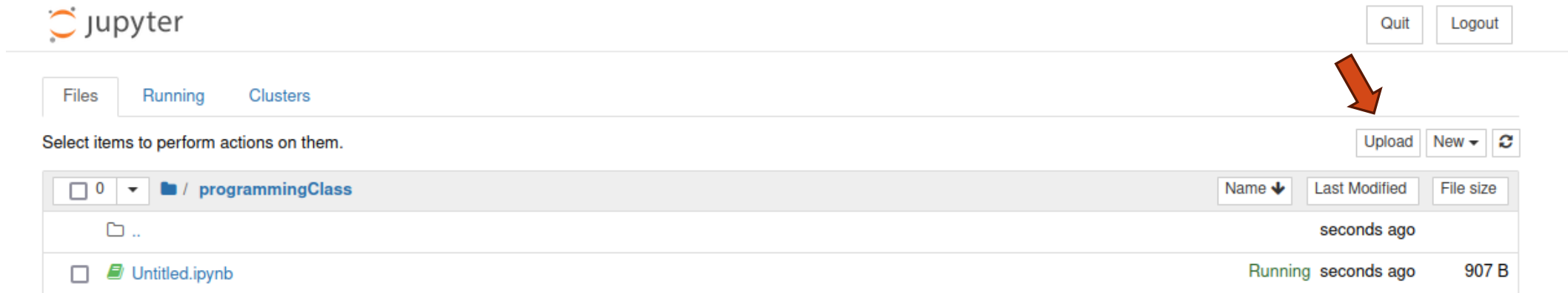
Select items to perform actions on them. Upload New ▾ ↻

<input type="checkbox"/> 0 ▾	📁 / programmingClass	Name ▾	Last Modified	File size
	📁 ..		seconds ago	
<input type="checkbox"/>	📄 Untitled.ipynb		Running seconds ago	907 B

# Rename the notebook



# Upload Files



The image shows the JupyterLab web interface. At the top left is the Jupyter logo. At the top right are 'Quit' and 'Logout' buttons. Below the logo is a tab bar with 'Files', 'Running', and 'Clusters'. The 'Files' tab is active. Below the tab bar is a message: 'Select items to perform actions on them.' To the right of this message are buttons for 'Upload', 'New', and a refresh icon. A red arrow points to the 'Upload' button. Below the message is a table of files and folders. The table has columns for 'Name', 'Last Modified', and 'File size'. The first row is a folder named '/ programmingClass'. The second row is a file named 'Untitled.ipynb' which is in a 'Running' state.

jupyter

Quit Logout

Files Running Clusters

Select items to perform actions on them.

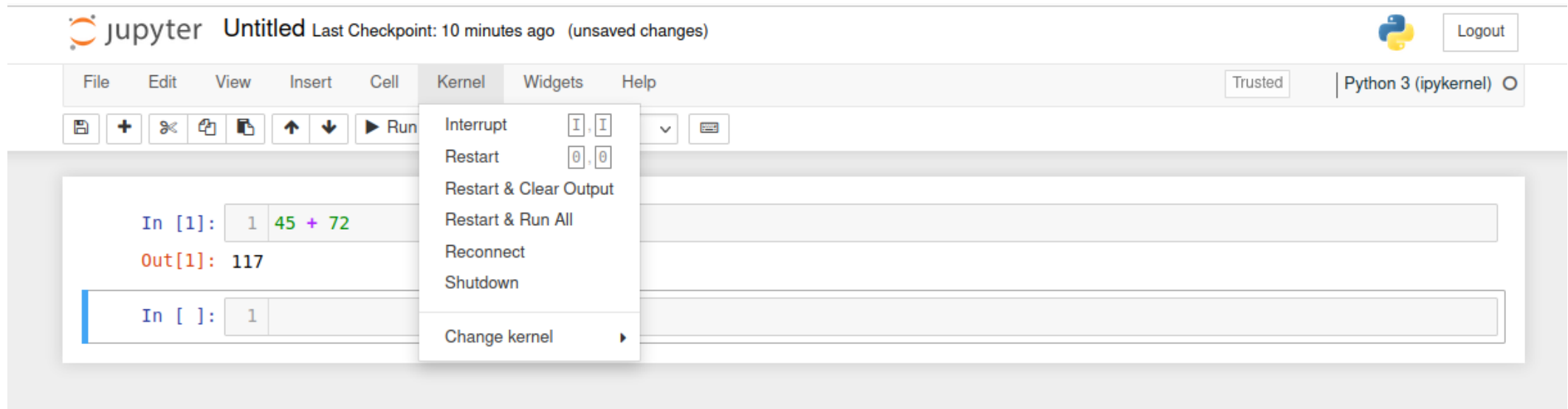
Upload New ↕

	Name ↓	Last Modified	File size
<input type="checkbox"/> 0	/ programmingClass		
	..	seconds ago	
<input type="checkbox"/>	Untitled.ipynb	Running seconds ago	907 B

# Create Folders/Create New Notebook/Launch Terminal

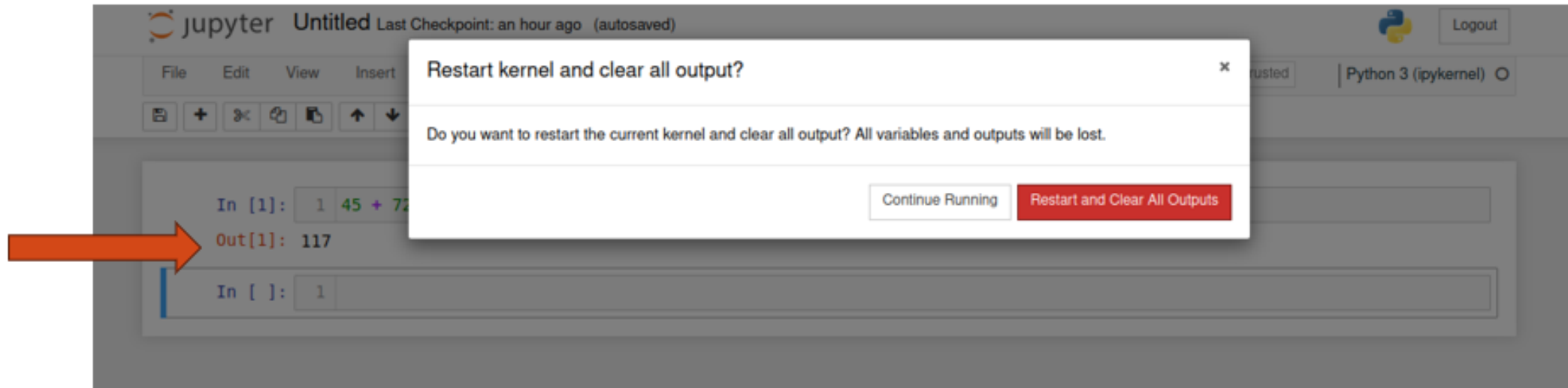


# Reset/Return the state of the notebook



# Restart and Clear all output

---



---

# Week 2:

## Working with Excel



# Working with data

---

Extract useful information and put it into a usable format

Most of the data is not standardize, hard to play nice when it comes to the extraction process

Common data formats

- Excel
- PDF

# Parsing Excel Data

---

3 main libraries for handling Excel files:

- **xlrd** (book has old info)
  - Reads Excel files
- **xlwt**
  - Writes and formats Excel files
- **xlutils**
  - A set of tools for more advanced operations in Excel (requires xlrd and xlwt)

# Files in the same folder

All files (i.e. xlsx) need to be in the same folder as your notebook (i.e. .ipynb)

Everything in my setup is in a `dataWrangler` folder



The image shows a JupyterLab interface. At the top left is the Jupyter logo and the word "jupyter". At the top right are "Quit" and "Logout" buttons. Below the logo are tabs for "Files", "Running", and "Clusters". The "Files" tab is active. Below the tabs is a message "Select items to perform actions on them." and buttons for "Upload", "New", and a refresh icon. The main area shows a file browser for the "dataWrangler" folder. It has a breadcrumb "0 / dataWrangler" and a table of files. The table has columns for "Name", "Last Modified", and "File size". The files listed are:

	Name	Last Modified	File size
<input type="checkbox"/>	..	seconds ago	
<input type="checkbox"/>	week_1_machineData.ipynb	Running 17 days ago	3.62 MB
<input type="checkbox"/>	week_2_working_with_excel.ipynb	Running 2 minutes ago	1.96 kB
<input type="checkbox"/>	data-text.csv	17 days ago	635 kB
<input type="checkbox"/>	data-text.json	17 days ago	1.65 MB
<input type="checkbox"/>	data-text.xml	17 days ago	2.5 MB
<input type="checkbox"/>	unicef_sowc.xlsx	an hour ago	106 kB

# Import package and open file

---

Input code into a cell in jupyter notebook

Must `run` each cell to pull in the code, other wise expect errors

```
In [8]: 1 # required packages `pip install openpyxl`  
        2 import openpyxl
```

```
In [9]: 1  
        2 # name of the excel workbook  
        3 xlsx_file_path = 'unicef_sowc.xlsx'  
        4  
        5 # load the workbook  
        6 workbook = openpyxl.load_workbook(xlsx_file_path)  
        7  
        8 # print to make sure it loaded - `sanity` test or `debug` test  
        9 print(workbook)  
       10  
       11
```

# Handling Excel Workbook Sheets

---

Unlike CSVs, Excel books can have multiple tabs or sheets

- Need to find out the names of them in order to access to specific sheet with the data you want

```
In [10]: 1 # variable to hold sheet names
          2 sheet_names = workbook.sheetnames
          3
          4 # Iterate through the sheet names and print them
          5 print("Names of the sheets in the workbook:")
          6 for sheet_name in sheet_names:
          7     print(sheet_name)
```

# Access Sheet By Name (Oops!!!)

---

Create a variable to hold the name of the sheet

Call the function to access

Pay attention to spaces in between quotes

```
In [11]: 1 # name of the sheet you want to access
          2 sheet_name = 'Table 9'
          3
          4 # Access the sheet by name
          5 sheet = workbook[sheet_name]
```

-----

```
KeyError                                Traceback (most recent call last)
Cell In[11], line 5
      2 sheet_name = 'Table 9'
      4 # Access the sheet by name
----> 5 sheet = workbook[sheet_name]

File ~/jupy/jup_notebook/lib/python3.10/site-packages/openpyxl/workbook/workbook.py:287, in Workbook.__getitem__(self, key)
    285     if sheet.title == key:
    286         return sheet
--> 287 raise KeyError("Worksheet {0} does not exist.".format(key))

KeyError: 'Worksheet Table 9 does not exist.'
```

# Access Sheet By Name (Correction)

---

The problem lies in the difference between what we see and what actually exists

Change 'Table 9' (no space after 9)

to

'Table 9 ' (there is now a space after the 9, no more error)

```
In [13]: 1 # name of the sheet you want to access
          2 sheet_name = 'Table 9 '
          3
          4 # Access the sheet by name
          5 sheet = workbook[sheet_name]
          6
          7 print(sheet)
```

# Explore what to do with a sheet

print a list of methods and attributes associated with the `sheet` object

Ex. `rows1`

```
In [15]: 1 # show what methods are available
        2 print(dir(sheet))
        3
```

['BREAK\_COLUMN', 'BREAK\_NONE', 'BREAK\_ROW', 'HeaderFooter', 'ORIENTATION\_LANDSCAPE', 'ORIENTATION\_PORTRAIT', 'PAPERSIZE\_A3', 'PAPERSIZE\_A4', 'PAPERSIZE\_A4\_SMALL', 'PAPERSIZE\_A5', 'PAPERSIZE\_EXECUTIVE', 'PAPERSIZE\_LEDGER', 'PAPERSIZE\_LEGAL', 'PAPERSIZE\_LETTER', 'PAPERSIZE\_LETTER\_SMALL', 'PAPERSIZE\_STATEMENT', 'PAPERSIZE\_TABLOID', 'SHEETSTATE\_HIDDEN', 'SHEETSTATE\_VERYHIDDEN', 'SHEETSTATE\_VISIBLE', 'WorkbookChild\_title', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_delitem\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getitem\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_iter\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_setitem\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'add\_cell', 'add\_column', 'add\_row', 'cells', 'cells\_by\_col', 'cells\_by\_row', 'charts', 'clean\_merge\_range', 'comments', 'current\_row', 'default\_title', 'drawing', 'get\_cell', 'hyperlinks', 'id', 'images', 'invalid\_row', 'move\_cell', 'move\_cells', 'parent', 'path', 'pivots', 'print\_area', 'print\_cols', 'print\_rows', 'rel\_type', 'rels', 'setup', 'tables', 'active\_cell', 'add\_chart', 'add\_data\_validation', 'add\_image', 'add\_pivot', 'add\_table', 'append', 'array\_formulae', 'auto\_filter', 'calculate\_dimension', 'cell', 'col\_breaks', 'column\_dimensions', 'columns', 'conditional\_formatting', 'data\_validations', 'defined\_names', 'delete\_cols', 'delete\_rows', 'dimensions', 'encoding', 'evenFooter', 'evenHeader', 'firstFooter', 'firstHeader', 'freeze\_panes', 'insert\_cols', 'insert\_rows', 'iter\_cols', 'iter\_rows', 'legacy\_drawing', 'max\_column', 'max\_row', 'merge\_cells', 'merged\_cell\_ranges', 'merged\_cells', 'mime\_type', 'min\_column', 'min\_row', 'move\_range', 'oddFooter', 'oddHeader', 'page\_margins', 'page\_setup', 'parent', 'path', 'print\_area', 'print\_options', 'print\_title\_cols', 'print\_title\_rows', 'print\_titles', 'protection', 'row\_breaks', 'row\_dimensions', 'rows', 'scenarios', 'selected\_cell', 'set\_printer\_settings', 'sheet\_format', 'sheet\_properties', 'sheet\_state', 'sheet\_view', 'show\_gridlines', 'tables', 'title', 'unmerge\_cells', 'values', 'views']



# What does `row` tell us

---

Expected to see a list of the rows

Instead, returns a generator object not a list of the rows

We need to iterate over the rows using a loop

```
In [18]: 1 # show what methods are available
          2 print(sheet.rows)

<generator object Worksheet._cells_by_row at 0x7f740093c890>
```

# Help functions are helpful 😊

`help()` is a built-in Python function that provides documentation on modules, classes, functions, methods, attributes, etc.

Access information associated with an object

- Ex. `help(sheet.rows)`

Understand how to use it correctly

Relevant methods:

- `__iter__`
- `__next__`

We need to iterate over each row, which means we need a **for** loop

In [17]:

```
1  
2 help(sheet.rows)
```

Help on generator object:

```
cells_by_row = class generator(object)  
  Methods defined here:  
  
    __del__(...)  
  
    __getattr__(self, name, /)  
        Return getattr(self, name).  
  
    __iter__(self, /)  
        Implement iter(self).  
  
    __next__(self, /)  
        Implement next(self).  
  
    __repr__(self, /)  
        Return repr(self).  
  
close(...)  
    close() -> raise GeneratorExit inside generator.  
  
send(...)  
    send(arg) -> send 'arg' into generator,  
    return next yielded value or raise StopIteration.  
  
throw(...)  
    throw(value)  
    throw(type[,value[,tb]])  
  
    Raise exception in generator, return next yielded value or raise  
    StopIteration.
```

# Iterate over each row and each cell

---

We need to find out what's in the excel sheet, we have identified the `rows` method as the logical place to start

Use **for** loop to print values in each row, and subsequent cells

Format so it is easier to read, the end='\t' is used to separate cell values with a tab

```
In [9]: 1 # iterate over each row and cell, then print the values
        2 for row in sheet.rows:
        3     for cell in row:
        4         print(cell.value, end='\t')
        5     print()
```

# Identify the contents in the worksheet

---

enumerate() function will get both the index and values for each row and cell

- takes care of keeping count with each loop iteration
- displays both the counter and value at the same time

```
In [ ]: 1 # iterate over each row
        2 for row_index, row_values in enumerate(sheet.iter_rows(min_row=1, values_only=True), start=1):
        3     row_name = f"Row {row_index}"
        4
        5     print(row_name)
        6
        7     # iterate through each cell in the row
        8     for cell_index, cell_value in enumerate(row_values, start=1):
        9         print(f"    Cell {cell_index}: {cell_value}")
       10
       11     # improve readability by adding a separator between each row
       12     print("-" * 20)
       13
```

# Organize the data into a dictionary

skip to get to the data we are interested in, we are looking to identify which row to start the data collection

data collection starts when “Countries and areas” is found

```
In [11]: 1 # skip to the header string "Countries and areas"
2 start_row = None
3
4 for row_index, row_values in enumerate(sheet.iter_rows(min_row=1, values_only=True), start=1):
5     if "Countries and areas" in row_values:
6         start_row = row_index + 1
7         break
8
9 # dictionary to store extracted data
10 extracted_data = {}
11
12 if start_row is not None:
13     for row_index, row_values in enumerate(sheet.iter_rows(min_row=start_row, values_only=True), start=start_row):
14         country_name = row_values[1]
15         child_labor_data = {
16             'total': row_values[4],
17             'male': row_values[6],
18             'female': row_values[8]
19         }
20         other_data = row_values[10:]
21
22         # store data in the dictionary
23         extracted_data[country_name] = {'child_labor': child_labor_data, 'other_data': other_data}
24
25         # print the data and associated a row number
26         print(f"Row {row_index}: {row_values[1:4]}")
27         print(f"    Child Labor (%): {row_values[4]} (total), {row_values[6]} (male), {row_values[8]} (female)")
28         print(f"    Other Data: {row_values[10:]}")
29         print("-" * 50)
30     else:
31         print("'Countries and areas' not found")
32
```

# Skip to the first country before starting

the data dictionary is populated correctly

- Collection starts and stops at the appropriate country
  - If you look at the spreadsheet, you should note the first row for countries is Afghanistan and the last row is Zimbabwe
  - This matches the output
- Countries are listed alphabetically

data contains the statistics from each country

- Ex.
  - Each's countries reported child labor and other data will be collected

In [12]:

```
1 # start from row 15, the first country
2 start_row = 15
3
4 # stop at row 212, the last country
5 stop_row = 212
6
7 # make sure when have are extracting data based on the countries
8 if 1 <= start_row <= sheet.max_row and 1 <= stop_row <= sheet.max_row and start_row <= stop_row:
9     extracted_data = {}
10
11     for row_index, row_values in enumerate(sheet.iter_rows(min_row=start_row, max_row=stop_row, values_only=True)):
12         country_name = row_values[1]
13
14         # skip rows where country_name is None
15         if country_name is None:
16             continue
17
18         child_labor_data = {
19             'total': row_values[4],
20             'male': row_values[6],
21             'female': row_values[8]
22         }
23         other_data = row_values[10:]
24
25         # store data in the dictionary
26         extracted_data[country_name] = {'child_labor': child_labor_data, 'other_data': other_data}
27
28         # print the names of the country only
29         print("\nExtracted Country Names:")
30         for i, name in enumerate(extracted_data.keys(), start=1):
31             print(f"{i}. {name}")
32
33     else:
34         print("Error with start or stop row values")
35
36
```

# Data collection is organized into nested dictionaries

at this point, our code output matches our end goal

the code is well documented with comments

- it is important to keep all your code commented, so you know what is happening for future reference

the excel format is an odd in-between category that is kind of machine readable

- not meant to be read by programs, but easy to parse with available python packages

```
In [13]: 1 # now that we are extracting the data from the countries
2
3 # iterate the data starting with the first country and stop processing on the last country
4 if 1 <= start_row <= sheet.max_row and 1 <= stop_row <= sheet.max_row and start_row <= stop_row:
5     extracted_data = {}
6
7     # get the headers
8     headers_row = next(sheet.iter_rows(min_row=1, max_row=1, values_only=True))
9     headers = headers_row[1:]
10
11     for row_index, row_values in enumerate(sheet.iter_rows(min_row=start_row, max_row=stop_row, values_only=True)):
12         country_name = row_values[1]
13
14         # skip rows where country_name is None
15         if country_name is None:
16             continue
17
18         # create a dictionary to store data for the current country
19         country_data = {}
20
21         # process child labor data
22         child_labor_labels = ['total', 'male', 'female']
23         child_labor_values = [None if value in ('-', ' ', None) or not isinstance(value, (int, float)) else float(
24             value) for value in row_values[2:]]
25         country_data['child_labor'] = dict(zip(child_labor_labels, child_labor_values))
26
27         # process other data
28         other_data_labels = ['married_by_15', 'married_by_18']
29         other_data_values = [None if value in ('-', ' ', None) or not isinstance(value, (int, float)) else float(
30             value) for value in row_values[2:]]
31         country_data['other_data'] = dict(zip(other_data_labels, other_data_values))
32
33         # add the country to dictionary
34         extracted_data[country_name] = country_data
35
36     # print the extracted or pulled data that we are interested in
37     for country, data in extracted_data.items():
38         print(f"\nCountry: {country}")
39         print("Data:")
40         for category, values in data.items():
41             print(f"    {category}: {values}")
42         print("-" * 50)
43
44     else:
45         print("Error with start or stop row values")
```

---

# Week 3:

PDF's and Problem Solving in Python  
Mid-Term



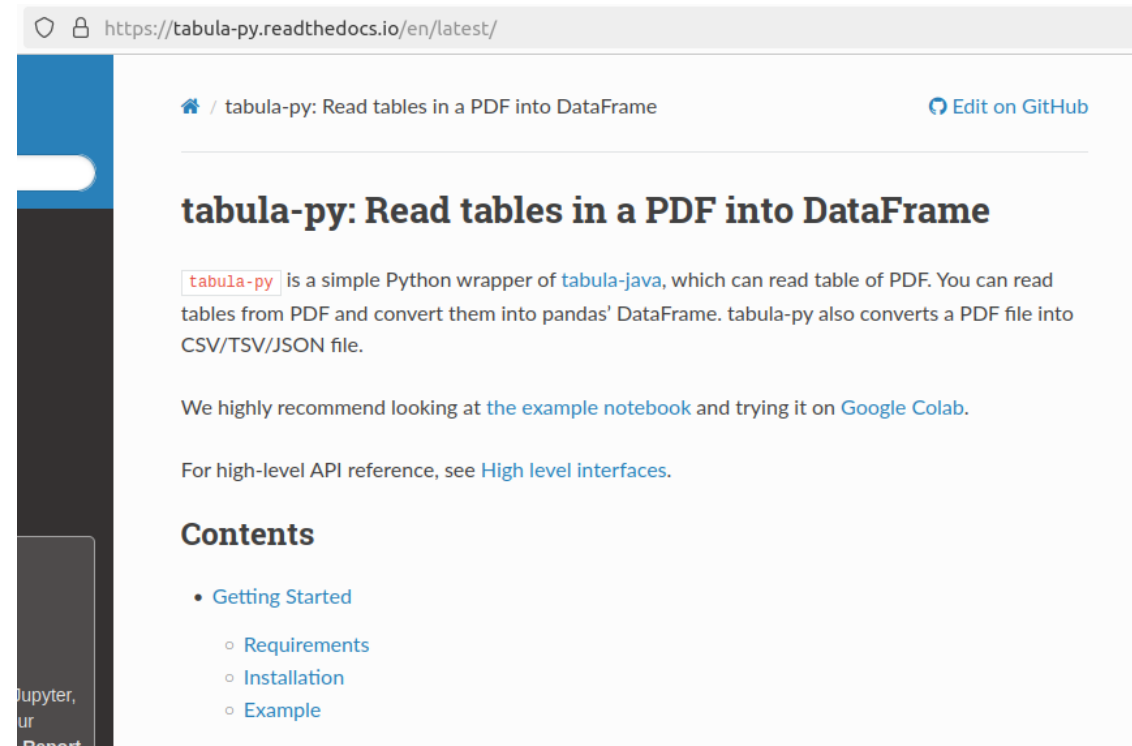
# PDF's as a Data Source

Can be a difficult-to-parse format

- data you need to work with may not always be in the ideal format

More often than not you will need to convert PDF files into some other format

- Can use a PDF tool and a programming language to parse a PDF files
- There is no one-size-fits all method to use, explore the options
  - Which tool is best is a matter of opinion



# Opening and Reading PDFs with Tabula

---

Install tabula python package

Upload FoodList.pdf

- must be uploaded into same folder or local to the Google Colab environment to access

Tables are put into a DataFrame

- read\_pdf()

Inspect the table and understanding its structure

```
[ ] # install tabula python package
!pip install tabula.py
```

```
[ ] # import the necessary libraries
from tabula import read_pdf
from tabulate import tabulate
```

```
[ ]
# filename variable of the pdf file which needs to be uploaded into the folder/environment
pdf_file = 'FoodList.pdf'

# extract data from page 1 of the pd file
page_number = 1

# returns the extracted tables as pandas dataframes
tables_df = read_pdf(pdf_file, pages=page_number)

# print the tables from page 1 of the pdf
print(tables_df)
```

# PDF Source: Food Calories List

Compare the output to what is in the pdf

PDF information is stored in table format

- Not the norm

It is easy to see the pattern of columns and rows in each page

## Food Calories List

From: [www.weightlossforall.com](http://www.weightlossforall.com)

The food calories list is a table of everyday foods listing their calorie content per average portion. The food calories list also gives the calorie content in 100 grams so it can be compared with any other products not listed here. The table can be useful if you want to exchange a food with similar calorie content when following a [weight loss](#) low calorie program.

The food calories list is broken down into sections based on the 5 basic food groups of a [balanced diet](#).

BREADS & CEREALS	Portion size *	per 100 grams (3.5 oz)	energy content
Bagel ( 1 average )	140 cals (45g)	310 cals	Medium
Biscuit digestives	86 cals (per biscuit)	480 cals	High
Jaffa cake	48 cals (per biscuit)	370 cals	Med-High
Bread white (thick slice)	96 cals (1 slice 40g)	240 cals	Medium
Bread wholemeal (thick)	88 cals (1 slice 40g)	220 cals	Low-med
Chapatis	250 cals	300 cals	Medium
Comflakes	130 cals (35g)	370 cals	Med-High
Crackerbread	17 cals per slice	325 cals	Low Calorie
Cream crackers	35 cals (per cracker)	440 cals	<a href="#">Low / portion</a>
Crumpets	93 cals (per crumpet)	198 cals	Low-Med
Flapjacks basic fruit mix	320 cals	500 cals	High
Macaroni (boiled)	238 cals (250g)	95 cals	<a href="#">Low calorie</a>
Muesli	195 cals (50g)	390 cals	Med-high
Naan bread (normal)	300 cals (small plate size)	320 cals	Medium
Noodles (boiled)	175 cals (250g)	70 cals	Low calorie
Pasta ( normal boiled )	330 cals (300g)	110 cals	Low calorie
Pasta (wholemeal boiled )	315 cals (300g)	105 cals	<a href="#">Low calorie</a>
Porridge oats (with water)	193 cals (350g)	55 cals	Low calorie
Potatoes** (boiled)	210 cals (300g)	70 cals	Low calorie
Potatoes** (roast)	420 cals (300g)	140 cals	Medium

# Tabula read\_pdf returns a dataframe object

---

Each DataFrame represents a table extracted from the PDF document

Understanding the format involves accessing individual tables, specific information within tables, and metadata about tables

- You can specify how many pages you want to extract (i.e., one to many, or all of them)

```
# extract tables from all pages
tables = read_pdf(pdf_file, pages='all', multiple_tables=True)
```

## Typical Process

- Extract information, iterate through the data, create a variable to store the results, print the output for inspection

# Use List Comprehension

---

List comprehension can be used with DataFrames to perform operations on each table, extracts specific information efficiently

In this snippet, it is used to clean each DataFrame in `tables_df` by removing columns containing NaN values

- prints the cleaned tables for inspection or further processing
- printed tables do not contain any NaN values

```
# use list comprehension to create a new list, loop through each dataframe, drops any columns that contain NaN (missing) values
cleaned_tables = [table.dropna(axis='columns') for table in tables_df]

# loop through the table and print everything, should not have any NaN values
for idx, table in enumerate(cleaned_tables):
    print(f"Table {idx+1} after dropping NaN values:")
    print(table)
```

# Converting to JSON

---

PDFs can have complex structures, including headers, footers, or other elements that might not be recognized as part of the table

- can lead to inconsistencies in data extraction

The method used to extract data from the PDF might be sensitive to certain formatting styles or content arrangements

- Some extraction methods might not handle certain types of PDFs, leading to missing or incorrectly extracted data

JSON conversion utilizes a different approach to access the data, which can result in different outcomes compared to direct extraction

```
# use list comprehension to convert the dataframe into a JSON string
tables_json = [table.to_json() for table in tables_df]

# loop over each JSON string to print data from the table
for idx, table_json in enumerate(tables_json):
    print(f"Table {idx + 1}:")
    print(table_json)
    # add a space/newline between tables
    print()
```

# Extracting Data from PDF is a Challenge

Problem	Description
Inconsistent Table Structures	Varying data structures, make it challenging to extract data accurately
Missing or Misaligned Rows and Columns	Extraction methods may overlook or misinterpret data
Header and Footer	Elements can interfere with extraction, resulting in incomplete or erroneous data extraction
Complex page layouts	Layouts pose difficulties with extracting data
Text Encoding	Non-standard fonts, encoding schemes, or character sets may cause recognition errors, resulting in garbled or unreadable data
Format conversion	Converting to other formats may introduce inconsistencies or loss of information

# Summary

---

Addressing challenges when PDF is a data source requires careful consideration:

- PDF structure
- selection of appropriate extraction methods and parameters
- preprocessing steps to standardize layouts
- validation of extracted data against the original PDF content



# Pro-Tip: Wrap-around Text in Print Out of Coding Assignment

When text exceeds the width of the cell, it will wrap the remaining text to a new line.

A common convention used in Jupyter Notebook is to denote wrapped text with the wrap-around arrow symbol

- When you see that, the code belongs on one line, do not put on a separate line

Another convention is the box shape 'U', this indicates a space is present before the text wraps

- When you that, the code or comment belongs on one line, do not put on a separate line

```
[ ]: # set flag to process information page by page, performance optimizer
      stream_option = True

      # extract contents from page 4
      page_number = 4

      # extract tables in a rectangular area defined by coordinates (top, left,
      ↪bottom, right)
      area = (270, 13, 790, 900)

      # extract from the specified area using the stream option
      tables_df = read_pdf(pdf_file, pages=page_number, stream=stream_option,
      ↪area=area)

      # loop over the table, print the information
      for idx, table in enumerate(tables_df):
          print(f"Table {idx + 1}:")
          print(table)
```

# Pro-Tip: Indentation in Python

Indentation in Python is crucial for defining the structure and hierarchy of code blocks

Indentations are used to denote blocks of code such as loops, conditional statements, function definitions, and more

Python does not use curly braces {} or keywords like begin and end to delimit blocks of code, it uses indentation

You will get errors if indentation is off, not aligned

```
# correctly aligned, if and else are aligned, print are properly indented
x = 10
if x > 5:
    print("x is greater than 5")
else:
    print("x is not greater than 5")

x is greater than 5

[3] # expect a indentation error
if x > 5:
    print("x is greater than 5")
    # incorrect indentation
    print("Another statement")

File "<ipython-input-3-b9ef6e7d3fcb>", line 3
    print("Another statement") # Incorrect indentation
    ^
IndentationError: unexpected indent

Next steps: Explain error

[4] # expect a logical error
for i in range(5):
    # should be indented
    print(i)

File "<ipython-input-4-12f320cdd30a>", line 4
    print(i)
    ^
IndentationError: expected an indented block after 'for' statement on line 2

Next steps: Explain error
```

# Homework # 3 Working with PDFs

---

As usual, submit screenshot of code and output saved as a PDF

It seems that Google Colab is “wonky” when it comes to saving as a PDF (I don’t know why)

If you experience wonky behavior...save code and corresponding output in a screenshot...put all screenshots in one PDF...DO NOT SEND multiple images...I have no idea what it is. I will not grade it.

Reach out to me if you have questions.

# Mid-Term Grades will reflect current grade

---

No mid-term exam

Canvas is up to date

---

# Week 4:

## Acquiring and Storing Data

# How can I find useful data?

---

Finding your first dataset(s) to investigate might be the most important step because it sets the foundation for your analysis and the insights you can derive from it

- Why?
  - Identifying the right dataset aligns with the questions you want to answer
  - Ensures the data you gather will be relevant and meaningful
  - Significantly impacts the quality of insights you can generate
  - Lays the groundwork for informed decision-making

First spend time refining your question until you have one specific enough to identify good data about but broad enough to be interesting to you and other

Alternatively, you might have a dataset you already find interesting, but no compelling question

If you don't already know and trust the data source, you should spend some time investigating.

Ask yourself:

- Is the data valid?
- Is it updated?
- Can I rely on future or current updates and publications?

# Not All Data Is Created Equal

---

Not all datasets meet expectations; some may be ineffective or inefficient

Perform a data smell test when acquiring new data to assess its trustworthiness

Questions to consider:

- Is the author a reliable source?
- Is the data regularly updated and checked for errors?
- Does the data include acquisition information?
- Can the data be verified by another source?
- Does the data seem plausible based on topic knowledge?

If you answered “yes” to at least three of those questions

- you are on the right track!

If you answered “no” to two or more of them,

- you might need to do some more searching to find data you can reliably defend

# Fact Checking

---

Fact checking your data, although sometimes annoying and exhausting, is paramount to the validity of your reporting

Fact checking may involve:

- Contacting the source(s) and verifying their latest methods and releases
- Determining other good sources for comparison
- Calling an expert and talking with them about good sources and veritable information
- Researching your topic further to determine whether your sources and/or datasets are credible



# Readability, Cleanliness, and Longevity

---

Computer generated data is easy to read; where as data sources like PDFs can be more difficult

Programming languages such as Python can help with unreadable data, but its unreadability may indicate questionable sourcing

Unreadable data may signify issues with cleanliness and reliability

Determine data cleanliness by inquiring about collection, reporting, and updating processes

- You should be able to determine:
  - How clean is the data?
  - Has someone taken the time to show statistical error rates or update erroneous entries or misreported data?
  - Will further updates be published or sent to you?
  - What methods were used in the collection of the data, and how were those methods verified?

Consider the longevity of the data:

- is it regularly collected and updated?

Know the update schedule of the data source to assess its long-term usability.

# Where to Find Data

---

There are many ways to find data, both on and offline

Resources:

- Telephone Books (do these even exist anymore?!)
- US Government
- Government and Civic Open Data Worldwide
- Organization and Non-Government Organization (NGO) Data
- Education and University Data
- Medical and Scientific Data
- Crowdsourced Data and APIs

The amount of data available is enormous, and it's no small task sorting through all of the noise to get a good idea of what questions you can answer and how you should go about answering them

# Storing Your Data: When, Why, and How?

---

Once you've located your data, you need a place to store it

Sometimes, you'll have received data in a clean, easy-to-access, and machine-readable format

Other times, you might want to find a different way to store it

Questions to consider:

- Can you open your dataset in a simple document viewer (like Microsoft Word), without crashing your computer?
- Does the data appear to be properly labeled and organized, so you can easily pluck out each piece of information?
- Is the data easily stored and moved if you need to use more than one laptop or computer while working on it?
- Is the data real-time and accessible via an API, meaning you can get the data you need by requesting it live?

# Data Storage and Backup Best Practices

---

Store datasets on a network, Internet (utilizing cloud computing), or external storage (e.g., hard drive, USB stick) when using them across multiple computers or locations

Consider accessibility needs for teams requiring data access from various locations or computers

Implement a data backup strategy, especially when working on a single computer, to prevent data loss in case of hardware failure or loss.

# Databases: basic concepts

---

Online activities and app usage often involve interactions with databases

Common actions include searching and receiving responses from databases

Two major database types:

- Relational Databases: MySQL and PostgreSQL
  - organize data into structured tables with predefined relationships, suitable for queries and transactions
  - scaling can be challenging, changing the schema may require downtime
  - performance may degrade with highly normalized schemas or complex queries
  - not ideal for handling unstructured or semi-structured data
- Non-Relational Databases: NoSQL
  - store data in flexible, schema-less formats, offering scalability and high performance for handling large volumes of unstructured data
  - data consistency and integrity may be compromised in favor of scalability

Concept/library	Purpose
Relational databases (e.g., MySQL and PostgreSQL)	Storing relational data in an easy way
Non-relational databases (e.g., MongoDB)	Storing data in a flat way
SQLite setup and usage	Easy-to-use SQL-based storage that works well for simple projects
Dataset installation and usage	Easy-to-use Python database wrapper

---

# Code Demonstration

---

Fundamental concepts of storage and retrieval of data, which are essential skills for handling data in various formats and from different sources

- enable you to acquire, store, and prepare data for further analysis or visualization tasks

Create a database, store data in a structured format, retrieve data for analysis, and interact/manipulate data through common operations

Storage:

- Database (sqlite)
- Create Tables
- Insert Data

Retrieval:

- Retrieve data from tables
- View the data

Interaction:

- Modify (i.e., add, update, and delete) data

# Storage Concepts

---

## create\_todo\_table function

- creates a table named todos in the SQLite database if it doesn't already exist.
- task ID, task description, and completion status
- sets up the database schema and ensuring that the required table structure is in place for storing todo list tasks

## add\_task function

- adds a new task to the todo list
- intent is to make sure database remains up-to-date with the latest additions

```
# create a todo list table
def create_todo_table(conn):
    cursor = conn.cursor()
    cursor.execute('''CREATE TABLE IF NOT EXISTS todos (
                        id INTEGER PRIMARY KEY,
                        task TEXT NOT NULL,
                        completed INTEGER DEFAULT 0
                    )''')
    conn.commit()
```

```
# add a new task to the todo list
def add_task(conn, task):
    cursor = conn.cursor()
    cursor.execute('INSERT INTO todos (task) VALUES (?)', (task,))
    conn.commit()
```



# Retrieval Concepts

---

## get\_tasks function

- retrieves all tasks from the todos table
- select all rows from the table
- presents them to the user for viewing or further processing

```
def get_tasks(conn):  
    cursor = conn.cursor()  
    cursor.execute('SELECT * FROM todos')  
    tasks = cursor.fetchall()  
    return tasks
```

# Interaction/Manipulation Concepts

---

## update\_task\_status function

- updates the completion status of a task
- modify task data, keep the database synchronized with the latest status

```
def update_task_status(conn, task_id, completed):  
    cursor = conn.cursor()  
    cursor.execute('UPDATE todos SET completed = ? WHERE id = ?', (completed, task_id))  
    conn.commit()
```

## delete\_task function

- deletes a task from the todos table
- provide task ID and delete the corresponding row in the table
- remove tasks from the todo list, maintain the integrity of the database

```
def delete_task(conn, task_id):  
    cursor = conn.cursor()  
    cursor.execute('DELETE FROM todos WHERE id = ?', (task_id,))  
    conn.commit()
```

# Main Function

Entry point for the todo list application

Establishes a connection to the database

Calls the functions that create the todo table

Presents a menu to the user

Handles user input to perform the various operations as it relates to the todo list

```
def main():  
  
    # connect to the SQLite database  
    conn = sqlite3.connect('todo.db')  
  
    # create the todo list table  
    create_todo_table(conn)  
  
    # loop until you exit  
    while True:  
        # menu interface for tasks  
        print("\nTODO LIST")  
        print("1. Add Task")  
        print("2. View Tasks")  
        print("3. Update Task Status")  
        print("4. Delete Task")  
        print("5. Exit")  
  
        choice = input("Enter your choice: ")  
  
        if choice == '1':  
            task = input("Enter task: ")  
            add_task(conn, task)  
            print("Task added successfully!")  
  
        elif choice == '2':  
            tasks = get_tasks(conn)  
            if not tasks:  
                print("No tasks found.")  
            else:  
                for task in tasks:  
                    print(f"{task[0]}. {task[1]} - {'Completed' if task[2] else 'Incomplete'}")  
  
        elif choice == '3':  
            task_id = int(input("Enter task ID: "))  
            completed = int(input("Enter completion status (1 for completed, 0 for incomplete): "))  
            update_task_status(conn, task_id, completed)  
            print("Task status updated successfully!")
```

# Start the todo list program

---

Effectively starts the execution of the todo list application

```
if __name__ == "__main__":  
    main()
```

# Homework #4 Storage Concepts

---

---

# Week 5:

Data Cleanup: Investigation, Matching and Formatting

# Cleaning Data

Data cleaning is an essential step in the data wrangling process

Goal is to ensure data is accurate, consistent and ready for analysis

By investigating, matching, and formatting data analysts can:

- improve data quality
- uncover hidden insights
- make informed decisions



# Why Clean Data?

---

Properly formatted data is rare (data almost never comes in clean)

- most datasets have inconsistencies or readability issues

Formatting and standardizing data is critical, especially when merging data from multiple sources (prepare data for analysis)

Clean data is easy to import and analyze by others

- Huge benefit when both the cleaned dataset and raw data are available and includes annotations detailing the cleaning steps

Documenting the data cleaning process is essential for defending the dataset's integrity and reproducibility

- ensures the ability to reproduce the cleaning process with new data



# Common Data Problems

---

Inconsistent column names

Missing Data

Outliers

Duplicate rows

Untidy

Need to process columns

Column types can signal unexpected data value

# Unclean Data

Missing Data (NaN)

Unwanted Observations

	created_at	tree_id	block_id	the_geom	tree_dbh	stump_diam	curb_loc	status	health	spc_latn	spc_common	steward	guards	sidewalk	user_type	problems
0	08/27/2015	180,683	348,711	POINT (-73.84421521958048 40.723091773924274)	3	0	OnCurb	Alive	Fair	Acer rubrum	red maple	NaN	NaN	NoDamage	TreesCount Staff	NaN
1	09/03/2015	200,540	315,986	POINT (-73.81867945834878 40.79411066708779)	21	0	OnCurb	Alive	Fair	Quercus palustris	pin oak	NaN	NaN	Damage	TreesCount Staff	Stones
2	09/05/2015	204,026	218,365	POINT (-73.93660770459083 40.717580740099116)	3	0	OnCurb	Alive	Good	Gleditsia triacanthos var. inermis	honeylocust	1or2	NaN	Damage	Volunteer	NaN
3	09/05/2015	204,337	217,969	POINT (-73.93445615919741 40.713537494833226)	10	0	OnCurb	Alive	Good	Gleditsia triacanthos var. inermis	honeylocust	NaN	NaN	Damage	Volunteer	Stones
4	08/30/2015	189,565	223,043	POINT (-73.97597938483258 40.66677775537875)	21	0	OnCurb	Alive	Good	Tilia americana	American linden	NaN	NaN	Damage	Volunteer	Stones
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
683783	08/18/2015	155,433	217,978	POINT (-73.95494401022562 40.7132107823145)	25	0	OnCurb	Alive	Good	Quercus palustris	pin oak	NaN	NaN	Damage	Volunteer	NaN
683784	08/29/2015	183,795	348,185	POINT (-73.85665019989099 40.71519444267162)	7	0	OnCurb	Alive	Good	Cladrastis kentukea	Kentucky yellowwood	1or2	NaN	NoDamage	Volunteer	NaN

# Load Data

---

Read data from a CSV (Comma-Separated Values) file and load it into a DataFrame object (tree\_census)

Pandas library is widely used for data cleaning, manipulation, and analysis

- Built in panda function read\_csv allows you to import data into Python for cleaning and analysis
- pd is an alias for pandas

**\*\*\* large csv file will take some time to load \*\*\***

```
import pandas as pd
```

```
tree_census = pd.read_csv('trees.csv')  
tree_census
```

# Visually Inspect

---

initial exploratory data analysis (EDA) for gaining insights into the dataset's structure, quality, and content

The head function returns the first 5 rows by default

The tail function returns the last 5 rows by default

Often used at the beginning of the cleaning process

```
tree_census.head()
```

```
tree_census.tail()
```

# Yet More Visual Inspection

---

Understand the structure, size, and characteristics of the dataset

Use information to identify potential issues and plan appropriate cleaning strategies

```
# list of column names  
tree_census.columns
```

```
# identify the size, number of rows and columns in the dataset  
tree_census.shape
```

```
# summary of the dataset  
tree_census.info()
```

---

# Exploratory Data Analysis

# Frequency Counts

Contains 683,788 entries (rows) and 42 columns

- Other information: column information and memory usage (cannot see in snippet)

Datatypes of each column:

- integers (int64)
- floating-point numbers (float64)
- strings (object)

```
# summary of the dataset
tree_census.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 683788 entries, 0 to 683787
Data columns (total 42 columns):
#   Column          Non-Null Count  Dtype
---  -
0   created_at      683788 non-null  object
1   tree_id         683788 non-null  object
2   block_id        683788 non-null  object
3   the_geom        683788 non-null  object
4   tree_dbh        683788 non-null  int64
5   stump_diam      683788 non-null  int64
6   curb_loc        683788 non-null  object
7   status          683788 non-null  object
8   health          652172 non-null  object
9   spc_latin       652169 non-null  object
10  spc_common      652169 non-null  object
11  steward         164350 non-null  object
12  guards          79866 non-null   object
13  sidewalk        652172 non-null  object
14  user_type       683788 non-null  object
15  problems        225844 non-null  object
16  root_stone      683788 non-null  object
17  root_grate      683788 non-null  object
18  root_other      683788 non-null  object
19  trnk_wire       683788 non-null  object
20  trnk_light      683788 non-null  object
```

# Frequency Count: Health Status of Trees

---

Distribution of tree health

Health categories of each tree:

- 'Good', 'Fair', and 'Poor'

Presence of missing values (NaN) indicates the health status is unknown or not recorded for some trees

```
# health status of trees  
tree_census.health.value_counts(dropna=False)
```

```
health  
Good      528850  
Fair       96504  
NaN        31616  
Poor       26818  
Name: count, dtype: int64
```



# Frequency Count: Tree Population

---

Current condition of the tree population

Active Categories:

- 'Alive', 'Stump', and 'Dead'

```
# get status on the trees  
tree_census.status.value_counts(dropna=False)
```

```
status  
Alive    652173  
Stump    17654  
Dead     13961  
Name: count, dtype: int64
```

# Remove Unwanted Observations

---

Reduced the number of columns from 42 to 22

Interested in only a subset

Show the first 5 rows in the subset

```
# remove columns not interested in
trees_subset = tree_census[['tree_id', 'tree_dbh',
                             'stump_diam', 'curb_loc', 'status', 'health', 'spc_latin', 'spc_common',
                             'steward', 'guards', 'sidewalk', 'user_type', 'problems', 'root_stone',
                             'root_grate', 'root_other', 'trnk_wire', 'trnk_light', 'trnk_other',
                             'brnch_ligh', 'brnch_shoe', 'brnch_othe']]

trees_subset.head()
```

# Summary Statistics

---

Numeric columns

Missing Values

Outliers

- Considerably higher or lower
- Require further investigation

# Summary Statistics: Missing Values

Each value represents the count of missing values in that column

isna()

- Returns the numbers of values that have NaN (missing)

sum()

- Count them all, add them up

Categories with missing values:

- health, spc\_latin, spc\_common, steward, guards, sidewalk, problems

```
# check for any null values
trees_subset.isna().sum()
```

```
tree_id      0
tree_dbh     0
stump_diam   0
curb_loc     0
status       0
health      31616
spc_latin    31619
spc_common   31619
steward     519438
guards      603922
sidewalk     31616
user_type    0
problems    457944
root_stone   0
root_grate   0
root_other   0
trnk_wire    0
trnk_light   0
trnk_other   0
brnch_ligh   0
brnch_shoe   0
brnch_othe   0
dtype: int64
```

# Data Visualization

---

Great way to spot outliers and obvious errors

More than just looking for patterns

Plan data cleaning steps

# Summary Statistics: Generate summary for the numeric columns

## Compute summary

- Number of non-null values, average, smallest value, middle values, percentiles, largest values

Helps to identify outliers, unusual patterns, or data quality problems that may require further investigation

```
tree_census.describe()
```

	tree_dbh	stump_diam	zipcode	cb_num	borocode	cncldist	st_assem	st_senate	boro_ct	Latitude	longitude
count	683788.000000	683788.000000	683788.000000	683788.000000	683788.000000	683788.000000	683788.000000	683788.000000	6.837880e+05	683788.000000	683788.000000
mean	11.279787	0.432463	10916.246044	343.505404	3.358500	29.943181	50.791583	20.615781	3.404914e+06	40.701261	-73.924060
std	8.723042	3.290241	651.553364	115.740601	1.166746	14.328531	18.966520	7.390844	1.175863e+06	0.090311	0.123583
min	0.000000	0.000000	83.000000	101.000000	1.000000	1.000000	23.000000	10.000000	1.000201e+06	40.498466	-74.254965
25%	4.000000	0.000000	10451.000000	302.000000	3.000000	19.000000	33.000000	14.000000	3.011700e+06	40.631928	-73.980500
50%	9.000000	0.000000	11214.000000	402.000000	4.000000	30.000000	52.000000	21.000000	4.008100e+06	40.700612	-73.912911
75%	16.000000	0.000000	11365.000000	412.000000	4.000000	43.000000	64.000000	25.000000	4.103202e+06	40.762228	-73.834910
max	450.000000	140.000000	11697.000000	503.000000	5.000000	51.000000	87.000000	36.000000	5.032300e+06	40.912918	-73.700488

# Bar Plots and Histograms

---

Bar Plots for Discrete Data Counts

Histograms for Continuous Data Counts

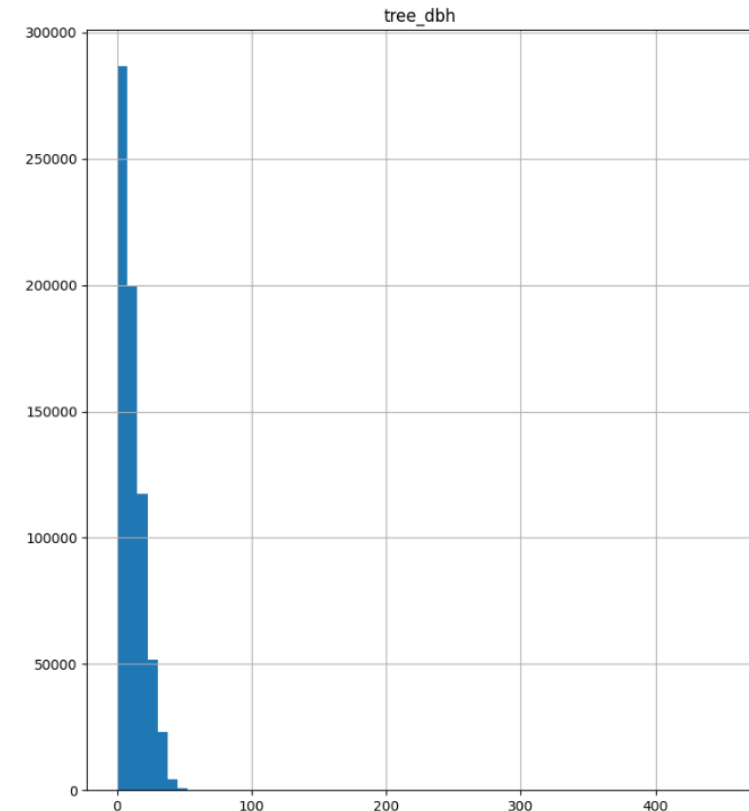
Look at frequencies

# Histogram

Generate histogram, data values are divided into intervals (bins), and plot the frequency count

```
# generate histogram of data distribution
trees_subset.hist(bins=60, figsize=(20,10))

array([[<Axes: title={'center': 'tree_dbh'}>,
        <Axes: title={'center': 'stump_diam'}>]], dtype=object)
```





# Potential Outliers: likely errors

Diameter of tree

Potential outlier is the tree that has a very large diameter size 425 cm (?)

```
# trees larger than 50
big_trees = trees_subset[trees_subset['tree_dbh'] > 50]
big_trees
```

	tree_id	tree_dbh	stump_diam	curb_loc	status	health	spc_lat	spc_common	steward	guards	sidewalk	user_type	problems	ro
	2385	168,583	425	0	OnCurb	Alive	Good	Quercus bicolor	swamp white oak	1or2	NaN	Damage	NYC Parks Staff	NaN
	3724	199,546	51	0	OnCurb	Alive	Good	Acer saccharinum	silver maple	NaN	NaN	NoDamage	TreesCount Staff	Stones
	4874	139,665	72	0	OffsetFromCurb	Alive	Good	Acer saccharinum	silver maple	NaN	NaN	NoDamage	TreesCount Staff	NaN
	6711	209,349	122	0	OnCurb	Alive	Good	Quercus palustris	pin oak	NaN	NaN	Damage	TreesCount Staff	NaN
	10053	215,075	169	0	OnCurb	Alive	Good	Gleditsia triacanthos var. inermis	honeylocust	NaN	NaN	NoDamage	TreesCount Staff	NaN
	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	675215	179,496	52	0	OffsetFromCurb	Alive	Good	Quercus palustris	pin oak	NaN	NaN	Damage	Volunteer	Stones

# Box Plots

---

Visualize basic summary statistics

- Outliers
- Min/Max
- 25<sup>th</sup>, 50<sup>th</sup>, 75<sup>th</sup> percentiles

# Box Plot

Distribution of tree stump diameters

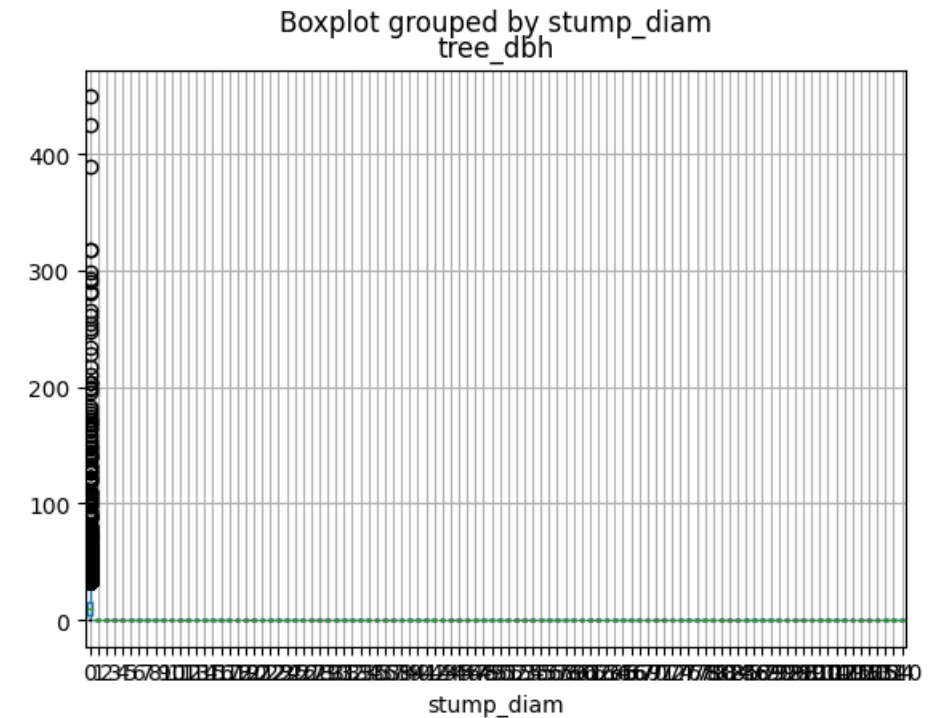
X-axis( stump diameter) is labeled "stump\_diam"

- Stump is 0, not really going to see anything

Y-axis (tree diameter ) is labeled "tree\_dbh".

```
# identify any potential outliers
tree_census.boxplot(column='tree_dbh', by='stump_diam')

<Axes: title={'center': 'tree_dbh'}, xlabel='stump_diam'>
```



# Scatter Plots

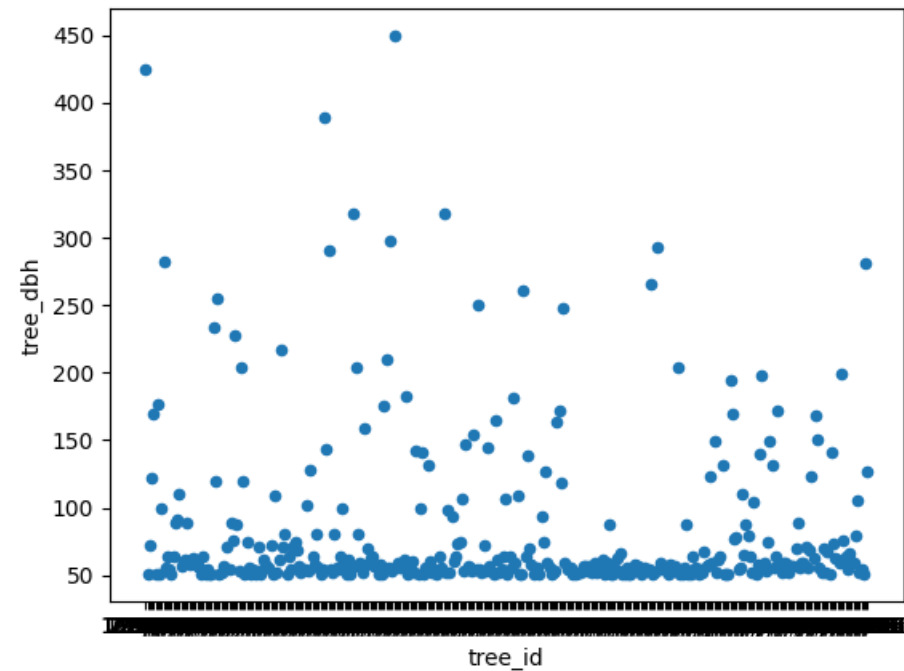
Relationship between 2 numeric variables

Flag potentially bad data

- Errors not found by looking at 1 variable

```
# scatter plot  
big_trees[['tree_id', 'tree_dbh']].plot(kind='scatter', x='tree_id', y='tree_dbh')
```

<Axes: xlabel='tree\_id', ylabel='tree\_dbh'>



# Homework #5 Data Cleaning

---

Be sure to upload tree.csv into the environment

All code and output required

Save as pdf input into canvas

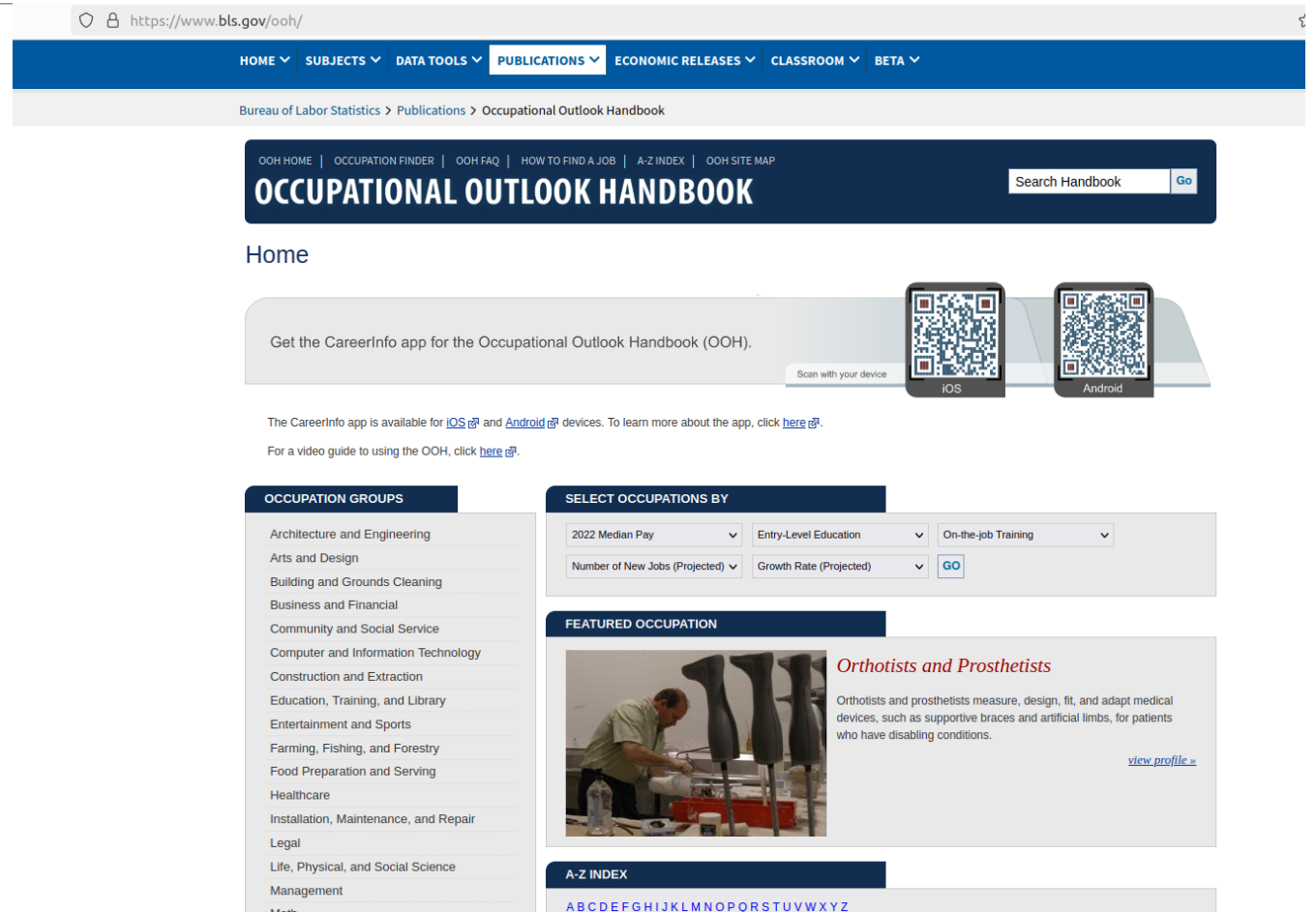
# Research your major

<https://www.bls.gov/ooh/>

Peruse this site for majors, pay, etc.

Share with your friends and loved ones who plan to go off to college soon.

Choose majors wisely!



The screenshot shows the Occupational Outlook Handbook website. The browser address bar displays <https://www.bls.gov/ooh/>. The website has a blue header with navigation links: HOME, SUBJECTS, DATA TOOLS, PUBLICATIONS, ECONOMIC RELEASES, CLASSROOM, and BETA. Below the header, a breadcrumb trail reads "Bureau of Labor Statistics > Publications > Occupational Outlook Handbook". The main content area features a dark blue banner with the text "OCCUPATIONAL OUTLOOK HANDBOOK" and a search bar. Below the banner, there's a section for the CareerInfo app with QR codes for iOS and Android. The main content area is divided into two columns. The left column, titled "OCCUPATION GROUPS", lists various fields of study. The right column, titled "SELECT OCCUPATIONS BY", includes filters for 2022 Median Pay, Entry-Level Education, On-the-job Training, Number of New Jobs (Projected), and Growth Rate (Projected). Below this is a "FEATURED OCCUPATION" section for "Orthotists and Prosthetists", which includes a photo of a person working with prosthetic limbs and a description of their role. At the bottom, there's an "A-Z INDEX" section with a list of letters from A to Z.

https://www.bls.gov/ooh/

HOME SUBJECTS DATA TOOLS PUBLICATIONS ECONOMIC RELEASES CLASSROOM BETA

Bureau of Labor Statistics > Publications > Occupational Outlook Handbook

OCCUPATIONAL OUTLOOK HANDBOOK

Home

Get the CareerInfo app for the Occupational Outlook Handbook (OOH).

Scan with your device

IOS Android

The CareerInfo app is available for [iOS](#) and [Android](#) devices. To learn more about the app, click [here](#).

For a video guide to using the OOH, click [here](#).

OCCUPATION GROUPS

- Architecture and Engineering
- Arts and Design
- Building and Grounds Cleaning
- Business and Financial
- Community and Social Service
- Computer and Information Technology
- Construction and Extraction
- Education, Training, and Library
- Entertainment and Sports
- Farming, Fishing, and Forestry
- Food Preparation and Serving
- Healthcare
- Installation, Maintenance, and Repair
- Legal
- Life, Physical, and Social Science
- Management
- Math

SELECT OCCUPATIONS BY

2022 Median Pay Entry-Level Education On-the-job Training

Number of New Jobs (Projected) Growth Rate (Projected) GO

FEATURED OCCUPATION

**Orthotists and Prosthetists**

Orthotists and prosthetists measure, design, fit, and adapt medical devices, such as supportive braces and artificial limbs, for patients who have disabling conditions.

[view profile »](#)

A-Z INDEX

ABCDEFGHIJKLMNOPQRSTUVWXYZ