Lesson 2 –Sampling And Data Collection

Agenda for Today:

- Understand various data types
- Sampling Methods
- Data Collection for Machine Learning
- Prelude to Data Wrangling
- Python Continuation

Divakaran Liginlal, Ph.D.
Dietrich College & Heinz College of Information Systems
Carnegie Mellon University
liginlal@cmu.edu

Data Analysis & Machine Learning

- **Data analysis** is the process of systematically applying statistical and other computational techniques to process data. Its aim is to extract useful information, and support data-driven decisions.
  - Tools and techniques: **Statistical analysis, machine learning algorithms, and data visualization methods**.

Machine learning is used to automate the data analysis process and aid the workflow to arrive at deeper and more holistic insights.

Overview of the Course

Lec 1: Introduction to Data Analysis
Overview of Data Analysis, Importance of Data Analysis in Business & Scientific research, Python Warmup

Lec 2: Sampling and Data Collection (Today)
Data Types, Sampling Techniques, Data collection methods, Prelude to Data wrangling and data transformation

Lec 3: Feature Engineering and Selection
Data wrangling and data transformation continued; Techniques for Feature Extraction, Criteria for Selecting Features, Impact of Feature Selection on Model Performance

Lec 4: Exploratory Data Analysis
Descriptive statistics, Data visualization tools & techniques, identifying patterns & anomalies in data

Lec 5: Regression Analysis
Linear, Non-linear & Logistic Regression Models; Model Fitting and Assumptions; Evaluating Regression Model Performance

Midterm Exam

Overview of the Course

Lec 6: Unsupervised learning - Clustering Methods
Overview of Clustering Algorithms; Application of K-means; Hierarchical Clustering Techniques and Dendrograms

Lec 7: Classification: Decision Trees
Building and Pruning Decision Trees, Measures of Impurity: Gini Index and Entropy, Advantages and Limitations of Decision Trees

Lec 8: Classification: Nearest Neighbor, Ensembles
Nearest Neighbor Algorithm and Distance Metrics; Introduction to Ensemble Techniques: Boosting, Bagging; Use Cases and Comparison of Ensemble Methods

Lec 9: Neural Networks and Deep Learning
Basics of Neural Networks: Neurons and Layers; Deep Learning Architectures: CNNs, RNNs, Autoencoders; Applications of Deep Learning in Image and Speech Recognition

Lec 10: Machine Learning in Real World Scenarios
Integrating Machine Learning into Business Processes; Ethical Considerations and Bias in Machine Learning Models; Case Studies of Successful Machine Learning Projects

Exam 2

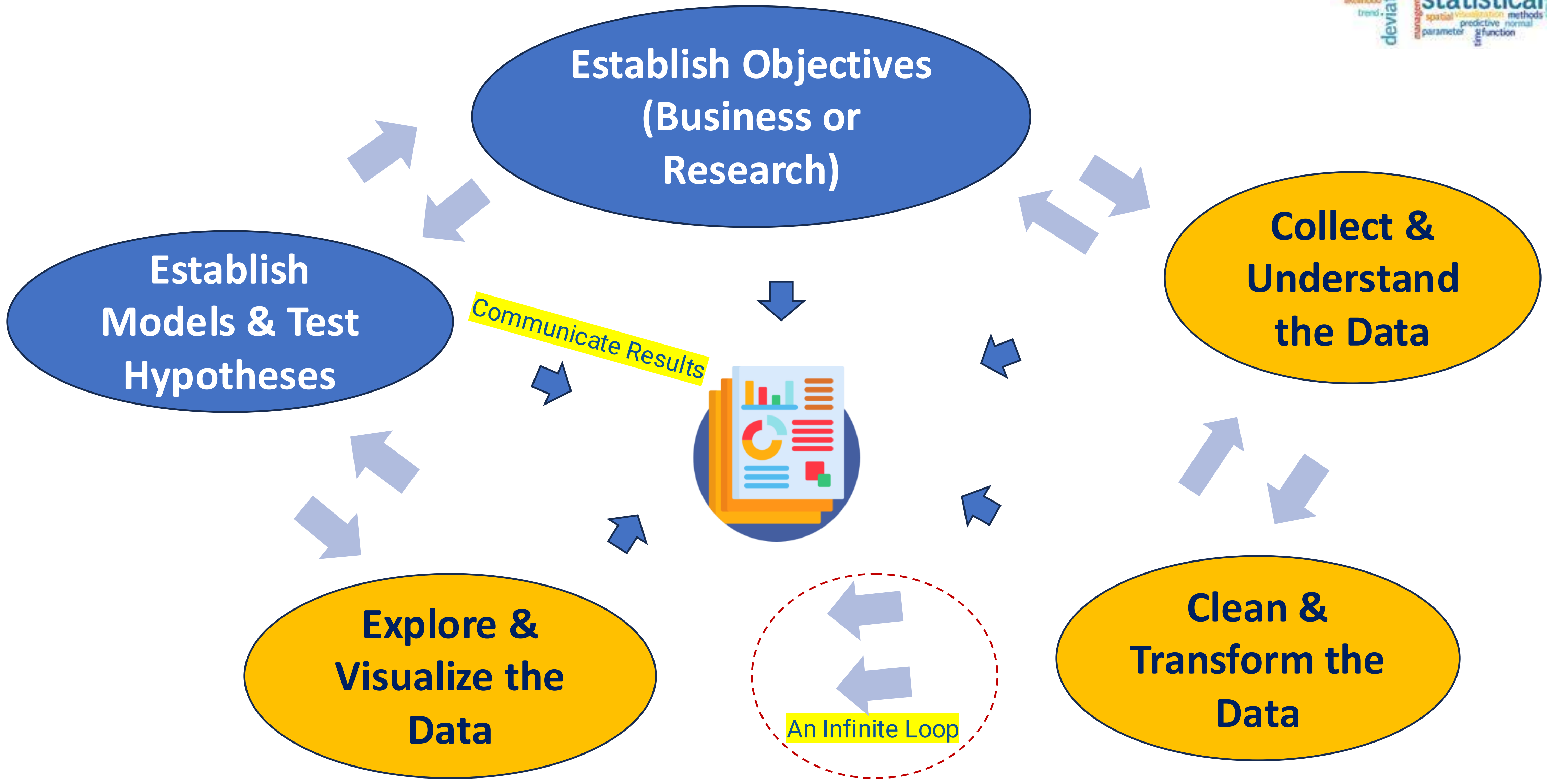
Domains & Applications we'll look at

Domain	Application Case Studies Included in this Course
Sales and Marketing	Hotel booking, real estate pricing, customer segmentation
Human Resources	Analyzing salary data
Finance & Accounting	Approving Credit/Loans, Predicting Credit Card Fraud
Healthcare (Clinical)	Managing Cancer, Predicting heart attacks
Social Welfare	Analyzing World Happiness index

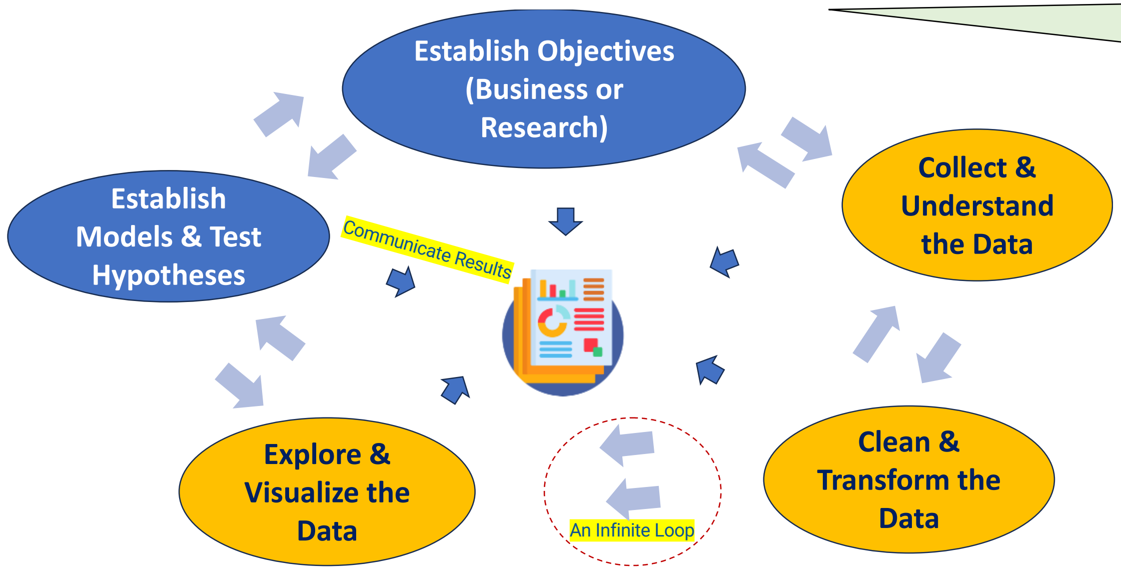
Why is Data Analysis Important Now?

- Much more **operational** data is being created and captured because of the use of technology (structured)
  - Enterprise software
    - ERP
    - CRM
    - SCM
- Much more **unstructured** data is being captured and stored (social media data)
  - Facebook
  - Twitter
- Much **more unstructured** data being captured
  - Web transactions
  - Smart sensors

Data Lifecycle


Establish Objectives (Business or Research)
Collect & Understand the Data
Clean & Transform the Data
Explore & Visualize the Data
Establish Models & Test Hypotheses
Communicate Results
An Infinite Loop

The Data Pipeline



Establish Objectives (Business or Research)
Collect & Understand the Data
Clean & Transform the Data
Explore & Visualize the Data
Establish Models & Test Hypotheses
Communicate Results
An Infinite Loop

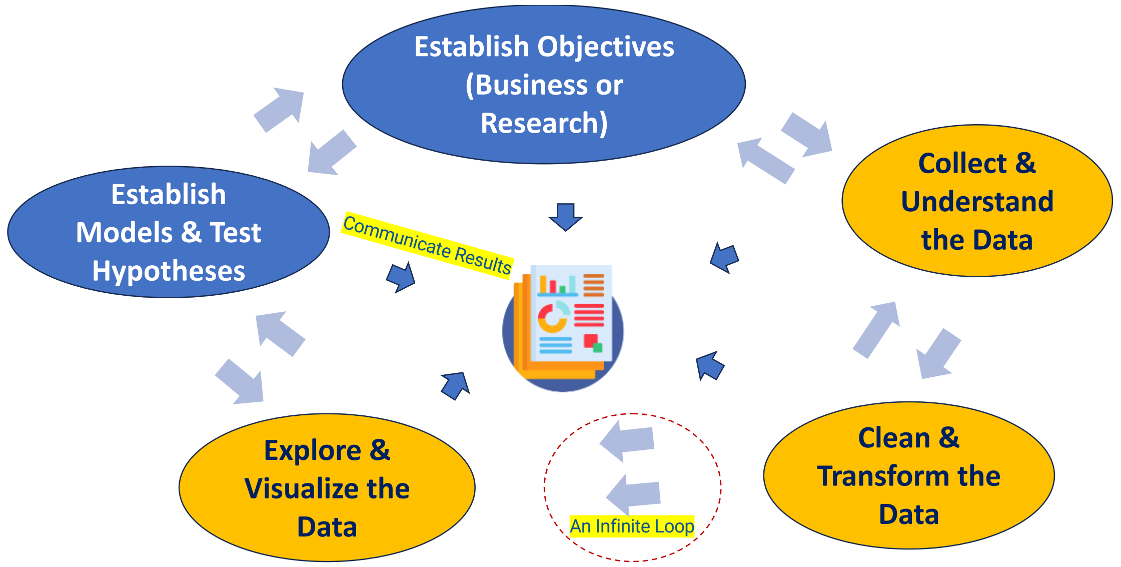
What is the business/scientific question?
What do you want to predict or estimate?
What would you do if you had **all** of the data?

Q: How to stock a store with the products people most want to buy?
Q: How does climate change impact biodiversity in a specific region over time?
Q: How do dietary habits influence the aging process in different populations?

Today's Learning Goals

- Understand data types, sampling and data collection methods
- Examine the steps in data preparation
- Introduce Data Preparation and Data Transformation
- Do Data Acquisition & Data Preparation Exercises in Python

The Data Pipeline


Establish Objectives (Business or Research)
Collect & Understand the Data
Clean & Transform the Data
An Infinite Loop
Explore & Visualize the Data
Establish Models & Test Hypotheses
Communicate Results

How were the data sampled?
Which data are relevant?
What anomalies exist in the data?
Are there privacy issues?

We have two steps now:

1. **Data Acquisition**: acquire the data that you need to answer the question
2. **Data Cleaning**: Investigating the data and cleaning up any problems.

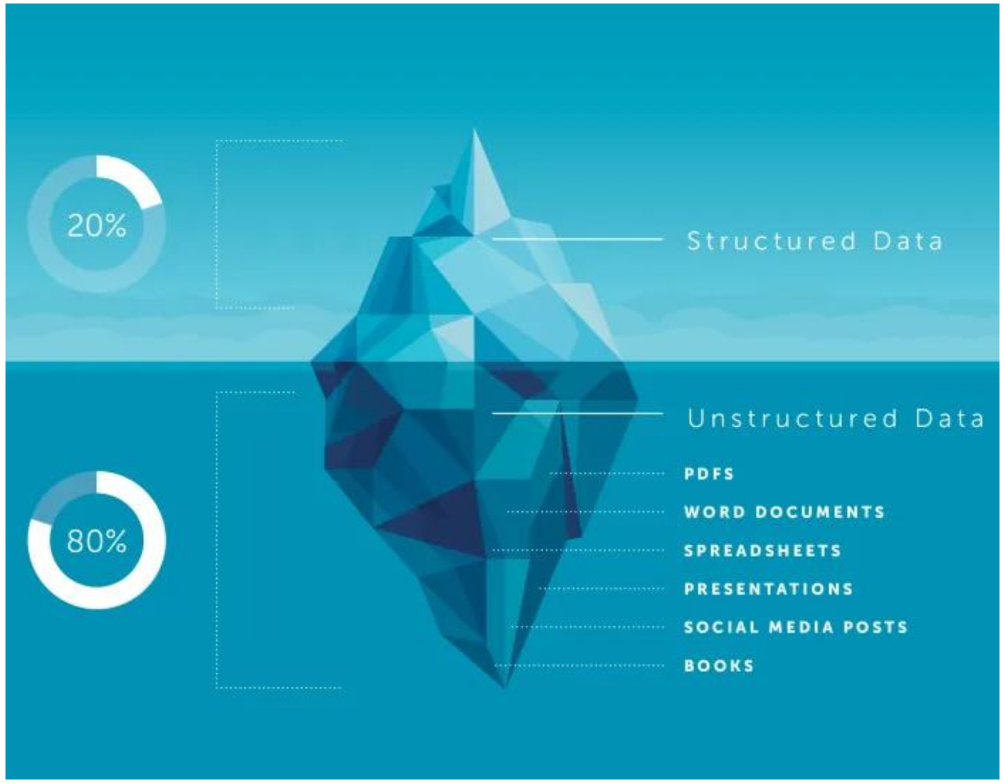
Collect & Understand the Data

1. **Collect initial data:** Use automated tools, forms, surveys, questionnaires etc. to gather data and load it for data preparation.
2. **Examine data:** Examine the data and document its surface properties like data format, number of records, or field identities.
3. **Do preliminary check of data:** Dig deeper into the data. Query it, visualize it, and identify relationships among the data.
4. **Verify data quality:** How clean/dirty is the data? Document any quality issues.

Structured Vs. Unstructured Data

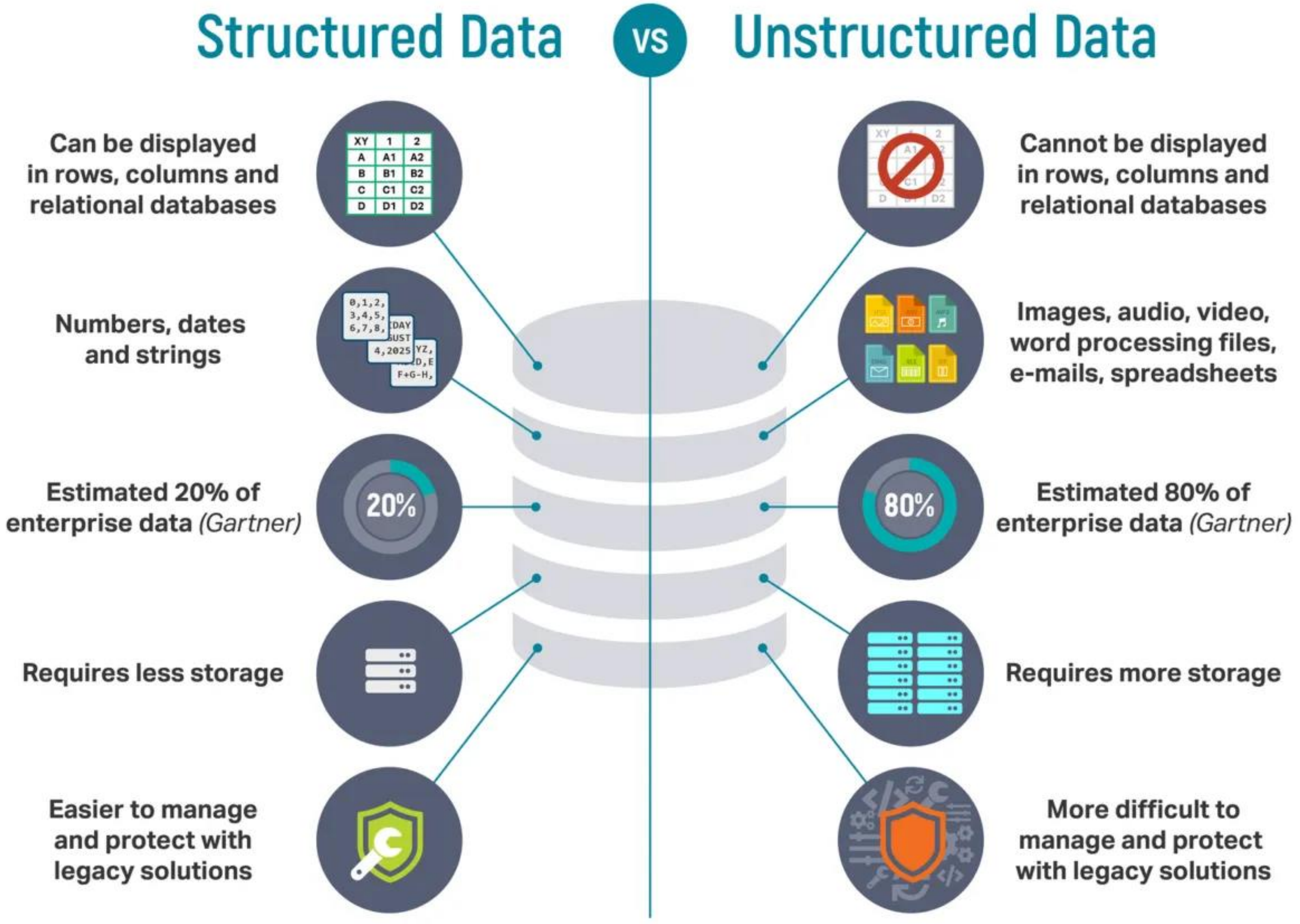
- **Structured Data**
  - Usually resides in a fixed field in a record of file
  - E.g. **Spreadsheet data and relational databases**
  - Based on a data model
  - Data types are defined clearly
  - Data structure is defined

- **Unstructured data**
  - No pre-defined data model
  - Not organized systematically
  - **Text-heavy** but may also contain data such as dates, numbers etc.
  - Video, Audio, Images etc.



https://lawtomated.com

Structured Vs. Unstructured Data



Structured Data	Unstructured Data
Can be displayed in rows, columns and relational databases	Cannot be displayed in rows, columns and relational databases
Numbers, dates and strings	Images, audio, video, word processing files, e-mails, spreadsheets
Estimated 20% of enterprise data *(Gartner)*	Estimated 80% of enterprise data *(Gartner)*
Requires less storage	Requires more storage
Easier to manage and protect with legacy solutions	More difficult to manage and protect with legacy solutions

Source: https://lawtomated.com/structured-data-vs-unstructured-data-what-are-they-and-why-care/

Semi-structured Data

- Structured but does not have a formal data model associated with it
  - Also, some degree of organization
- Adaptable and tolerant of different data formats because it does not have a set schema or data model.
  - **Extensible Markup Language** **(XML)**,
  - JavaScript Object Notation (JSON)
  - Markup languages such as HTML.
- Flexibility in storing and handling different kinds of data (Data Lakes)
- Useful when data may have different levels of organization and structure, such as web scraping, document storage, and **big data** analytics.

Quantitative Vs. Qualitative Data

Quantitative
(number of days stay in hospital, age)

Continuous (age, body weight, temperature)

Discrete (number of children, students in a class, books sold)

Qualitative (color of cars, opinions, types of animals)

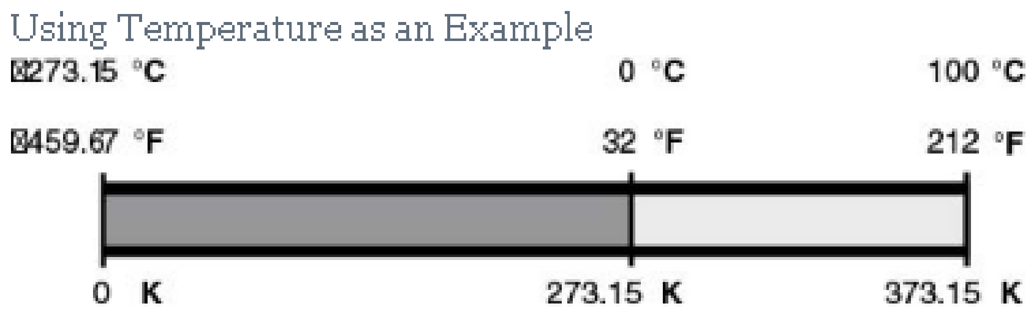
Quantitative Vs. Qualitative Data

- **Quantitative** or numeric or metric
  - Can be **discrete** or **continuous**
    - Number of children in a family, books sold, students in a class
    - Height of students, time to complete a task, today's average temperature
  - Easier to summarize and analyze statistically;
- **Qualitative** or nonnumeric
  - Descriptive in nature
  - E.g., Observations and opinions entered into an electronic medical record
  - Patients' experiences while receiving care
  - Transcribed notes from focus groups
  - Requires more preparation prior to analysis
    - Gives insights that quantitative data may not.

Measuring Quantitative Data

- **Interval**
  - **Difference between two values is meaningful and consistent.**
  - **But No definition of a 0**
  - *Year, Temperature measures (Centigrade, Fahrenheit – 0 does not mean absence of temperature)*
  - Measurable, can be ranked and categorized
- **Ratio.**
  - **Difference between two values is meaningful and consistent**
  - **Clear definition of zero**
  - *Age, height, weight, days inpatient, dosage of medicine, blood pressure, level of calcium*
  - *Temperature (Kelvin scale)*
  - Measurable, can be ranked and categorized
  - E.g. Qty of 20 means twice the value of 10 (ratio)

Interval & Ratio Data



- Celsius scale is an **interval scale** as 0 doesn’t mean absence of temperature
- Kelvin scale is a **ratio scale** as Absolute zero represents total absence of temperature (thermodynamically)

Measuring Qualitative Data

Nominal
(e.g. gender, hair color, mode of transport)

Ordinal
(e.g. level of pain, education level, customer satisfaction rating)

Measuring Qualitative Data

- **Nominal**
  - Non-numeric data (names) that is placed into mutually exclusive, separate, but non-ordered categories
  - *Types of Insurance coverage – Medicare, Medicaid, self-pay, employer-covered etc.,*
  - *Blood types, eye color, mode of transportation*
  - Usually, numeric codes are assigned for convenience
- **Ordinal**
  - May be categorized and ranked in a numerical fashion
  - Difference between values is not meaningful nor consistent.
  - *Likert scale, Level of pain*
  - *Severity of illness (0=none, 4 vital organ failure)*
  - *Economic class (low income, middle income, high income)*

More Data Types

- **Time Series Data**
  - Sequentially indexed at equally spaced intervals, often over time.
  - Economics, finance, and environmental studies.
  - Stock market prices and daily rainfall measurements.
- **Text Data**
  - Unstructured text that can be sourced from books, websites, emails, or social media posts.
  - Sentiment analysis, topic modeling, or language translation.
- **Image Data**
  - Visual data from photographs, videos, or any digital imagery.
  - Image recognition, object detection, and more sophisticated applications like autonomous driving.

More Data Types

- **Audio Data**
  - Data captured from sound recordings, which can include spoken words, music, or ambient noises.
  - Speech recognition, music recommendation systems, and acoustic event detection.
- **Video Data**
  - **S**equence of images (frames) over time, often with audio.
  - Surveillance, motion analysis, and multimedia content analysis.
- **Multimodal Data**
  - Data that combines two or more of these types to provide richer information that can be processed by hybrid machine learning models.
  - Detect and understand human emotions.

Sampling for Machine Learning

- Selecting a subset of data from a larger population
  - Creates datasets that are manageable, representative, and practical for analysis.
  - Not just a technique for reducing data size
  - Strategic approach to enhance model development, ensure robust performance, and maintain practicality in machine learning projects.
- Ensures that the insights drawn from machine learning models are:
  - Accurate
  - Applicable to broader real-world applications.

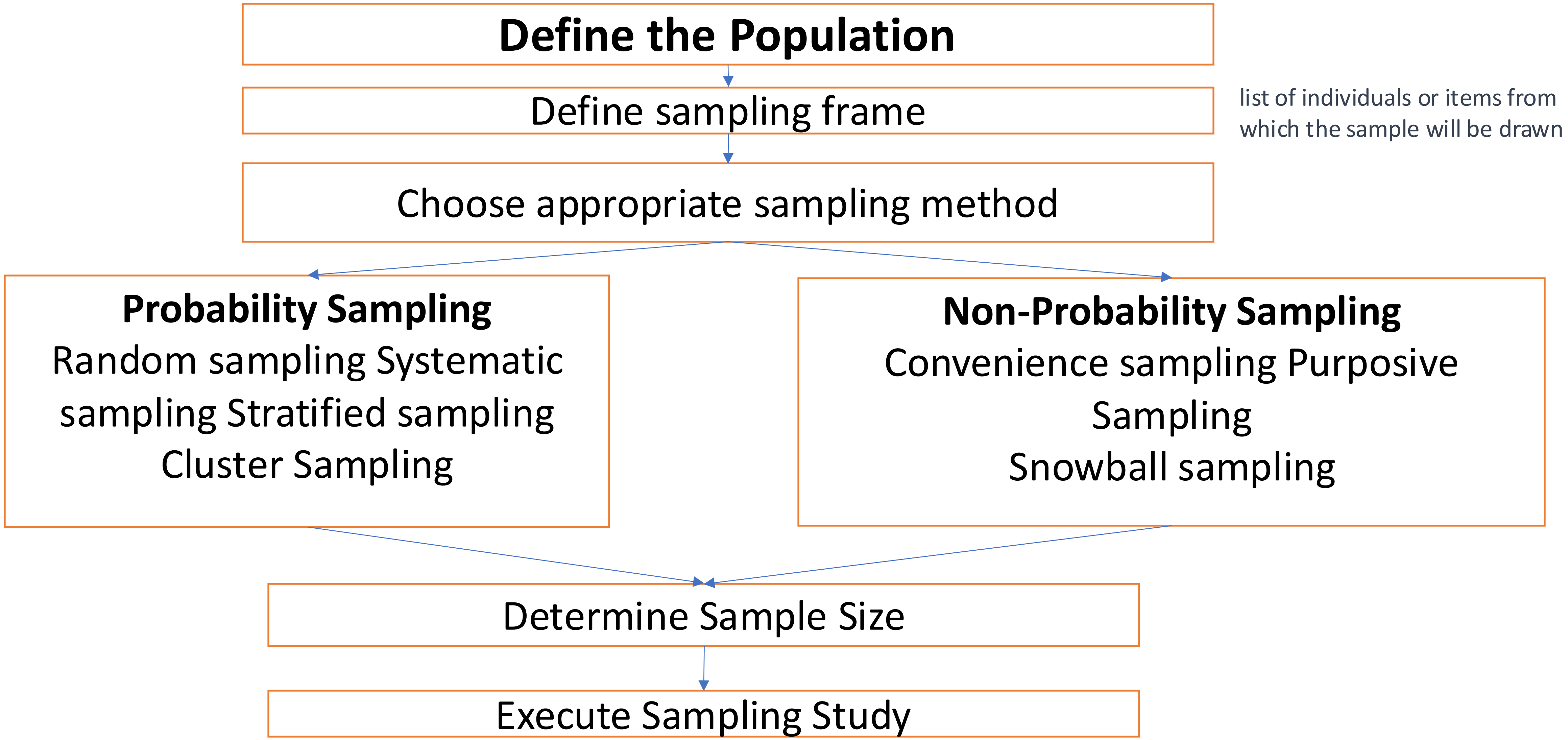
Why Sampling?

- **Feasibility and Cost-Effectiveness**
  - Efficient use of resources by reducing the data volume
- **Speed and Efficiency**
  - Reduce the time needed for training machine learning models.
- **Reduce Overfitting (Training vs Validating/Testing)**
  - Possible to create a dataset that generalizes well on unseen data.
- **Improved Model Performance**
  - Each category is adequately represented

Why Sampling?

- **Handling Skewed Data**
  - Oversampling the minority class or undersampling the majority class helps create more balanced datasets
- **Accessibility and Privacy**
  - Help in creating anonymized and less sensitive datasets while still retaining essential characteristics
- **Testing and Validation**
  - Crucial for dividing data into distinct sets (like training, validation, and test sets)
- **Scalability**
  - Maintaining and processing the entire dataset can become increasingly challenging.
  - Particularly useful in scenarios like streaming data where it's not feasible to store and process every piece of data.

Sampling design


Define the Population
Define sampling frame
list of individuals or items from which the sample will be drawn
Choose appropriate sampling method
Probability Sampling
Random sampling Systematic sampling Stratified sampling Cluster Sampling
Non-Probability Sampling
Convenience sampling Purposive Sampling
Snowball sampling
Determine Sample Size
Execute Sampling Study

Probability versus Nonprobability

- **Probability Sampling:** each member of the population has a known non-zero probability of being selected
  - Methods include random sampling, systematic sampling, stratified sampling and cluster sampling.
- **Nonprobability Sampling:** members are selected from the population in some nonrandom manner
  - Methods include convenience sampling, purposive sampling, and snowball sampling

Random Sampling

- Random sampling is the purest form of probability sampling.
- Each member of the population has an equal and known chance of being selected.
- When there are very large populations, it is often ‘difficult’ to identify every member of the population, so the pool of available subjects becomes biased.
    - You can use software, such as minitab to generate random numbers or to draw directly from the columns

Systematic Sampling

- Systematic sampling is often used instead of random sampling. It is also called an Nth name selection technique.
- After the required sample size has been calculated, every Nth record is selected from a list of population members.
- As long as the list does not contain any hidden order, this sampling method is as good as the random sampling method.
- Its only advantage over the random sampling technique is simplicity (and possibly cost effectiveness).

Stratified Sampling

- Stratified sampling is commonly used probability method that is superior to random sampling because it reduces sampling error.
- A stratum is a subset of the population that share at least one common characteristic; such as males and females.
  - Identify relevant stratums and their actual representation in the population.
  - Random sampling is then used to select a sufficient number of subjects from each stratum.
  - Stratified sampling is often used when one or more of the stratums in the population have a low incidence relative to the other stratums.

Cluster Sampling

- Divide the population into clusters and randomly selecting entire clusters to include in the sample.
- For instance, we want to study the use of AI in Small and Medium Businesses (SME)
    - We can divide SMEs into clusters based on a criterion such as turnover or size of employees and then apply random sampling techniques to choose clusters

Convenience Sampling

- **Convenience sampling** is used in exploratory research where the researcher is interested in getting an inexpensive approximation.
- The sample is selected because they are convenient.
- It is a nonprobability method.
    - Often used during preliminary research efforts to get an estimate without incurring the cost or time required to select a random sample
- Example: If we are studying user response to a newly developed mobile application you can choose anyone who is readily accessible to you as a participant (e.g., maybe people in your organization)

Purposive Sampling

- Selecting individuals or items based on specific criteria or characteristics.
- Example: Understanding the challenges faced by expert artificial intelligence (AI) practitioners in implementing natural language processing (NLP) algorithms in real-world applications.
- Criteria for selection:
  - Years of experience in the field
  - Academic qualifications
  - Track record of successful NLP implementations in real-world applications.

Snowball Sampling

- Utilizing initial participants to refer additional participants for inclusion in the sample.
- Example: Investigating the user experience and challenges faced by individuals with visual impairments while using a newly developed website.
- Contact an organization that exclusively works for people with visual impairments. Try to recruit a few of their members. Then use those contacts to find other participants for your research.

Data Acquisition

The first step of data analysis is …

- The first step in data analysis is to get some data!
- You typically get data following four possible strategies:
  1. Directly download a data file (or files) manually – we'll see dataframes
  2. Query data from a database
  3. Query an API (usually web-based, these days)
  4. Scrape data from a webpage

Data Acquisition

- **Numeric Data Types:**
  - Sensors (e.g., temperature sensors)
  - Surveys with quantitative measures, databases
- **Categorical Data Types**
  - Surveys and questionnaires, transaction logs
  - Data tagging and annotation tools
- **Time Series Data**
  - Financial systems
  - Meteorological data collection, APIs for real-time data feeds

Data Acquisition

- **Text Data**
  - Web scraping, APIs (e.g., Twitter API)
  - Customer feedback forms
- **Image Data**
  - Cameras and imaging devices, medical imaging technologies
  - Satellite imagery services
- **Audio Data**
  - Digital recorders, smartphones and other mobile devices
  - Audio streaming platforms for data capture
- **Video Data**
  - Video cameras, Mobile devices
  - Specialized software for video capture

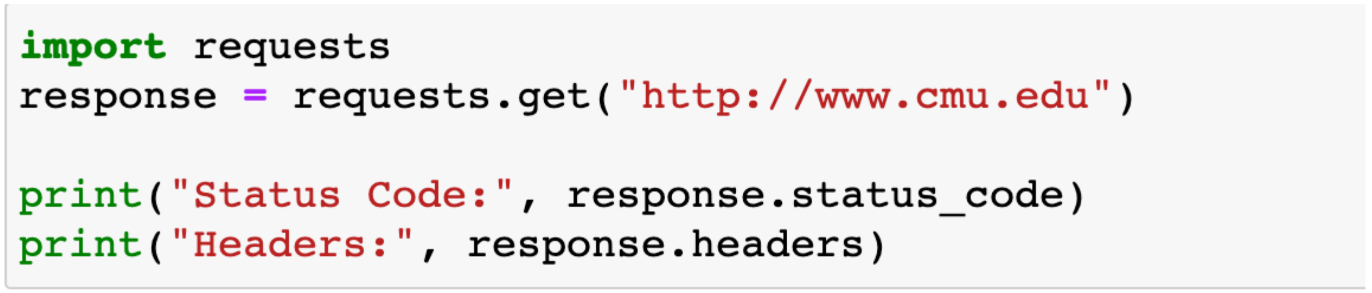
Collecting data from web-based sources

Some scraping knowledge

- HTTP: the communication protocol
- HTML: the language in which web pages are defined
- JS: javascript (code executing in the browser)
- CSS: style sheets, how web pages are styled. Important, but does not contain data.
- JPG, PNG, BMP: images, usually not interesting
- CSV / TXT / JSON / XML: **data, interesting** !!!

Issuing HTTP queries

- Most automated data queries you will run will use **HTTP requests (**It's becoming the dominant protocol for much more than just querying web pages)
- We will not create our own http client!
- We will use the Python **requests** library (http://docs.python-requests.org) to scrape data from the web.



```python
import requests
response = requests.get("http://www.cmu.edu")

print("Status Code:", response.status_code)
print("Headers:", response.headers)
```

```
url = "https://www.someurl"
request_headers = {
   'user-agent': 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko)
Chrome/91.0.4472.124 Safari/537.36'
} #emulates a browser to circumvent blocking of queries made by bots
page =requests.get(url, headers=request_headers)
```

Issuing HTTP queries

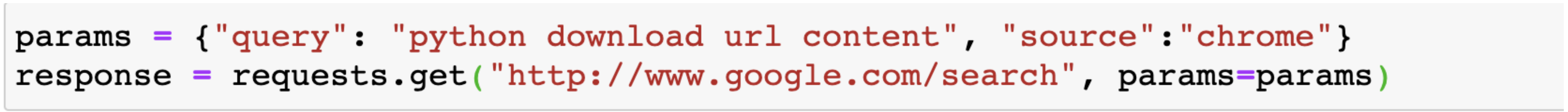
- You’ve seen URLs like these:

https://www.google.com/search?q=python+download+url+content&source=chrome

Composed of two parts:

1. **The URL itself:** “http://www.google.com/search”
2. **After ?**: parameters, each parameter in the form of “parameter=value”

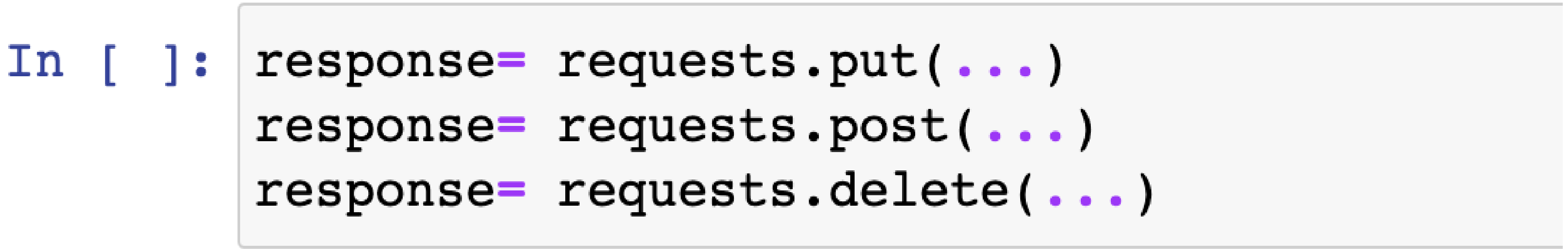
Parameters are provided as follows using the **requests library**



```python
params = {"query": "python download url content", "source":"chrome"}
response = requests.get("http://www.google.com/search", params=params)
```

Issuing HTTP queries

- HTTP GET is the most common method, but there are also PUT, POST, DELETE methods that change some state on the server



```
In [ ]: response= requests.put(...)
        response= requests.post(...)
        response= requests.delete(...)
```

1.GET. The GET method is used to retrieve data on a server. ...
2.POST. The POST method is used to create new resources. ...
3.PUT. The PUT method is used to replace an existing resource with an updated version.

HTML in a few slides

- ***HTML (Hypertext Markup Language)*** is used to format web pages by "marking up" text to display in some special way
  - HTML is (well, should be) hierarchical
  - Elements have begin and end tags

```
<b>This will display as bold and <i>italic</i></b>
```

element, or start tag

end tag

HTML, cont.

- Elements can have ***attributes***: some info related to the tag

```
<a href="https://www.heinz.edu">Heinz College</a>
```

link tag

attribute with info

This will show up as a hyperlink

HTML, cont.

- Elements can be nested

```
<body>
   <h1>Big Heading</h1>
   <p>A paragraph</p>
</body>
```

body: the whole doc

- Block-level elements are for grouping and style within the block

```
<div style="background-color:black;color:white;padding:20px;">
  <h2>Pittsburgh</h2>
   <p>A city in Pennsylvania</p>
</div>
```

div: division

Style specification (CSS)

- Specifying styles for elements

```
<head>
<style>
.cities {
  background-color: black;
  color: white;
  margin: 20px;
  padding: 20px;
}
</style>
</head>
<body>

<div class="cities">
  <h2>Pittsburgh</h2>
  <p> A city in Pennsylvania p>
</div>
</head>
```

Defines .cities style

division using that style

Web Scraping

- You're lucky if the stuff you want is in an html table nicely arranged
- Otherwise, finding data inside a regular web page can be tedious
  - A web page is usually a mix of HTML code and possibly CSS and Javascript and …
  - Disclaimer: web scraping is tedious and ad hoc

Web Scraping

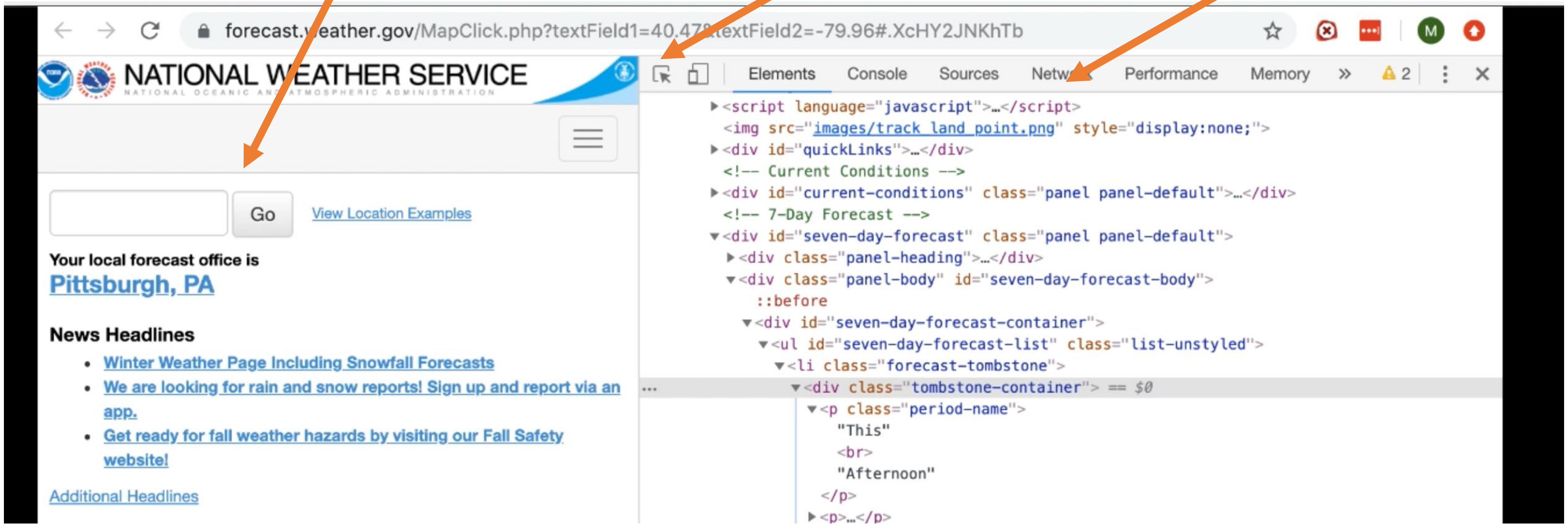
- You have to examine a page to know what to search for programmatically
  - Google **Chrome devTools** helps
- But once you've done that, your program is good for … a while
- Depends on the source not ever changing
- Google Chrome devTools: on a web page, right-click and choose Inspect
- Show rendered page on left, HTML code that produces it on the right
- When you roll the mouse cursor over a rendered element, the HTML code that produces it is highlighted
  - This can be reversed by clicking the square-with-arrow icon

DevTools: examine the html

rendered text and images

toggle switch

HTML code



forecast.weather.gov/MapClick.php?textField1=40.47&textField2=-79.96#.XcHY2JNKhTb
NATIONAL WEATHER SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION
Go
View Location Examples
Elements
Console
Sources
Network
Performance
Memory

Your local forecast office is

Pittsburgh, PA

News Headlines

- Winter Weather Page Including Snowfall Forecasts
- We are looking for rain and snow reports! Sign up and report via an app.
- Get ready for fall weather hazards by visiting our Fall Safety website!

Additional Headlines

```
▶<script language="javascript">…</script>
 <img src="images/track_land_point.png" style="display:none;">
▶<div id="quickLinks">…</div>
 <!-- Current Conditions -->
▶<div id="current-conditions" class="panel panel-default">…</div>
 <!-- 7-Day Forecast -->
▼<div id="seven-day-forecast" class="panel panel-default">
  ▶<div class="panel-heading">…</div>
  ▼<div class="panel-body" id="seven-day-forecast-body">
     ::before
    ▼<div id="seven-day-forecast-container">
      ▼<ul id="seven-day-forecast-list" class="list-unstyled">
        ▼<li class="forecast-tombstone">
          ▼<div class="tombstone-container"> == $0
            ▼<p class="period-name">
               "This"
               <br>
               "Afternoon"
             </p>
            ▶<p>…</p>
```

BeautifulSoup

- ***BeautifulSoup (bs4)*** is a library to help with scraping static web pages (use ***Selenium*** for dynamic web pages)
  - HTML is highly nested with a tree structure
  - Use requests.get(url) first to download a page – import requests
  - *BeautifulSoup* will parse that tree for you and let you search based on HTML tags or known strings

Refer for a simple tutorial: https://www.tutorialspoint.com/beautiful_soup/index.htm

BeautifulSoup, cont.

- Import it with (Make sure bs4 is already installed in your environment)

```
from bs4 import BeautifulSoup
```

- Use ***requests*** library to actually fetch the web page

```
import requests
```

- Also pass a header
- `headers = {"User-Agent":"Mozilla/5.0"}`
- `page = requests.get(<your URL here>, headers)`

BeautifulSoup, cont.

- Useful functions:
  - `find( )`: return reference for a tag/string
  - `find_all( )`: return a list of references for a tag/string
  - `select( )`: return a list of references for elements or CSS selectors
- After downloading the page with the `requests` library -
  - Use `bs4 html.parser` to parse it

Examples

```
obj = soup.find(id="someid") #find elements with specific id
obj = soup.find_all(id="someid", limit=10)
                                        #find 10 elements with specific id
obj = soup.select("#nm") # accepts CSS class selector as argument
```

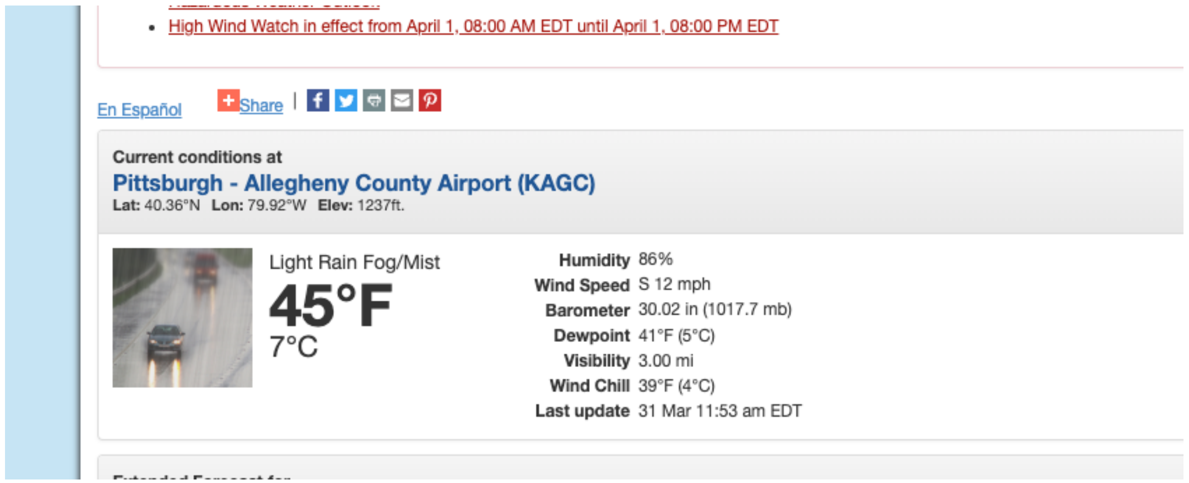
Example: Getting Weather

- Using this web site:

`https://forecast.weather.gov`

- Navigate to that site, search for Pittsburgh
  - Slides will differ from today's data

Part of the page



- High Wind Watch in effect from April 1, 08:00 AM EDT until April 1, 08:00 PM EDT

En Español Share

Current conditions at
Pittsburgh - Allegheny County Airport (KAGC)
Lat: 40.36°N Lon: 79.92°W Elev: 1237ft.

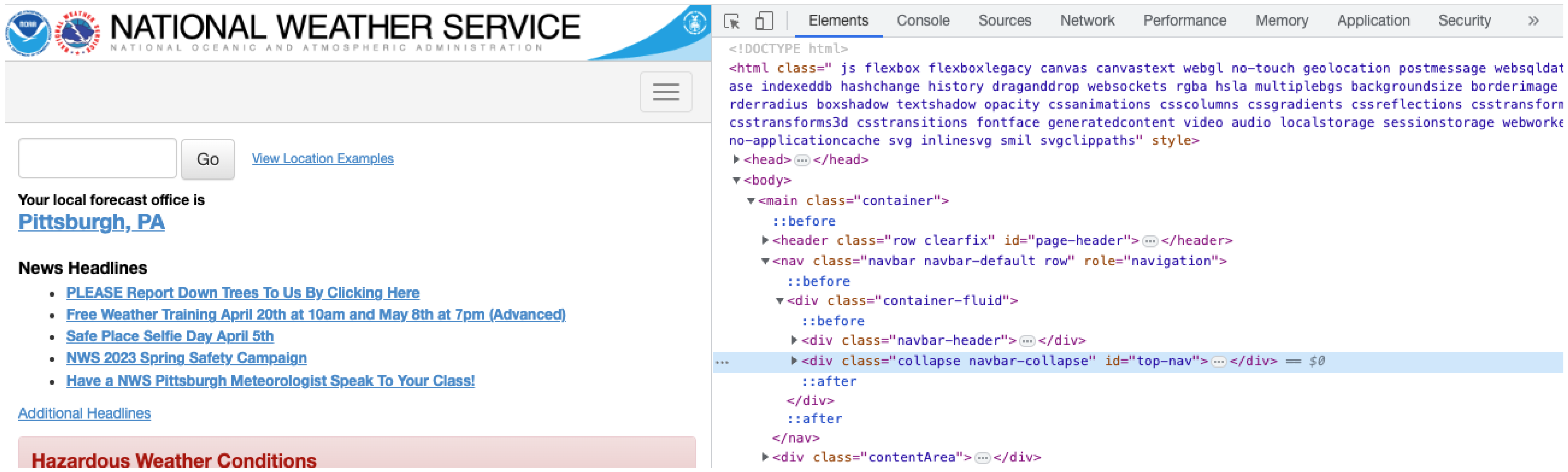
Light Rain Fog/Mist
45°F
7°C

Humidity	86%
Wind Speed	S 12 mph
Barometer	30.02 in (1017.7 mb)
Dewpoint	41°F (5°C)
Visibility	3.00 mi
Wind Chill	39°F (4°C)
Last update	31 Mar 11:53 am EDT

Tutorial: https://www.dataquest.io/blog/web-scraping-tutorial-python/

The Weather Service

- Right-click, choose Inspect to get devTools to get a split screen with the page on the left and the HTML source on the right



NATIONAL WEATHER SERVICE
NATIONAL OCEANIC AND ATMOSPHERIC ADMINISTRATION

Go View Location Examples

Your local forecast office is
Pittsburgh, PA

News Headlines

- PLEASE Report Down Trees To Us By Clicking Here
- Free Weather Training April 20th at 10am and May 8th at 7pm (Advanced)
- Safe Place Selfie Day April 5th
- NWS 2023 Spring Safety Campaign
- Have a NWS Pittsburgh Meteorologist Speak To Your Class!

Additional Headlines

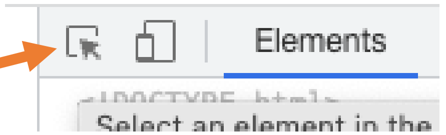
Hazardous Weather Conditions

```
<!DOCTYPE html>
<html class=" js flexbox flexboxlegacy canvas canvastext webgl no-touch geolocation postmessage websqldat
ase indexeddb hashchange history draganddrop websockets rgba hsla multiplebgs backgroundsize borderimage
rderradius boxshadow textshadow opacity cssanimations csscolumns cssgradients cssreflections csstransform
csstransforms3d csstransitions fontface generatedcontent video audio localstorage sessionstorage webworke
no-applicationcache svg inlinesvg smil svgclippaths" style>
 ▶<head>…</head>
 ▼<body>
   ▼<main class="container">
       ::before
     ▶<header class="row clearfix" id="page-header">…</header>
     ▼<nav class="navbar navbar-default row" role="navigation">
         ::before
       ▼<div class="container-fluid">
           ::before
         ▶<div class="navbar-header">…</div>
...      ▶<div class="collapse navbar-collapse" id="top-nav">…</div> == $0
           ::after
         </div>
         ::after
       </nav>
     ▶<div class="contentArea">…</div>
```

The Weather Service

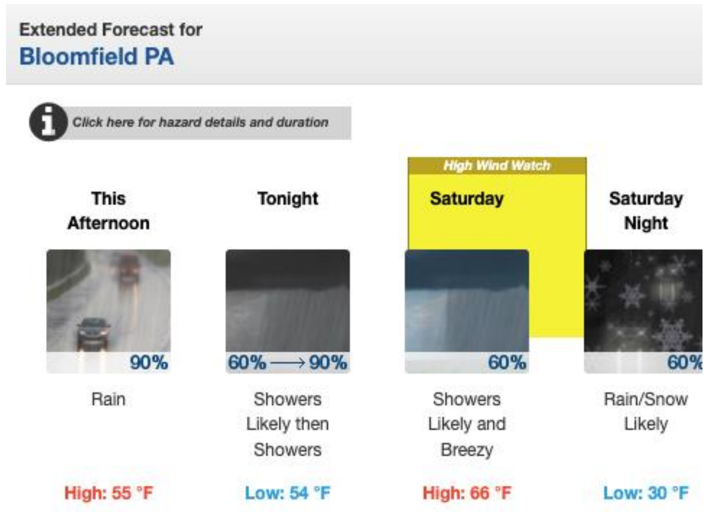
- Roll the mouse over the code to see the associated feature highlighted
- Then click on the toggle icon to switch: then click the feature on the page to see the associated code

feature toggle



- in this case, for the *Extended Forecast*

- Say we want to get the information in today's forecast, the first box in the extended forecast

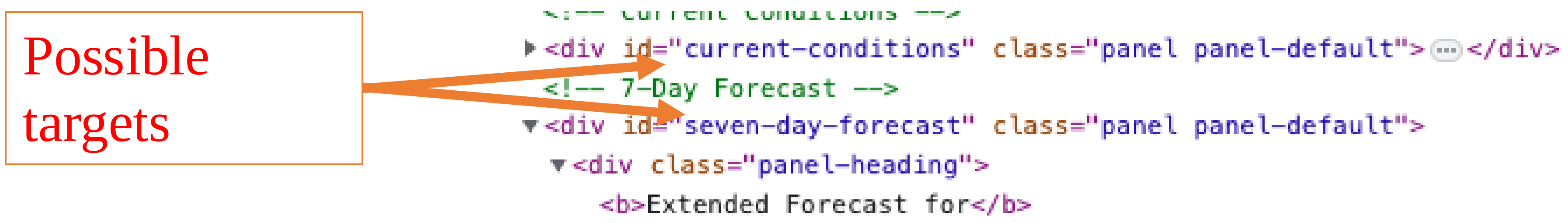

Extended Forecast for
Bloomfield PA

Click here for hazard details and duration

This Afternoon	Tonight	Saturday	Saturday Night
90%	60% ⟶ 90%	High Wind Watch 60%	60%
Rain	Showers Likely then Showers	Showers Likely and Breezy	Rain/Snow Likely
High: 55 °F	Low: 54 °F	High: 66 °F	Low: 30 °F

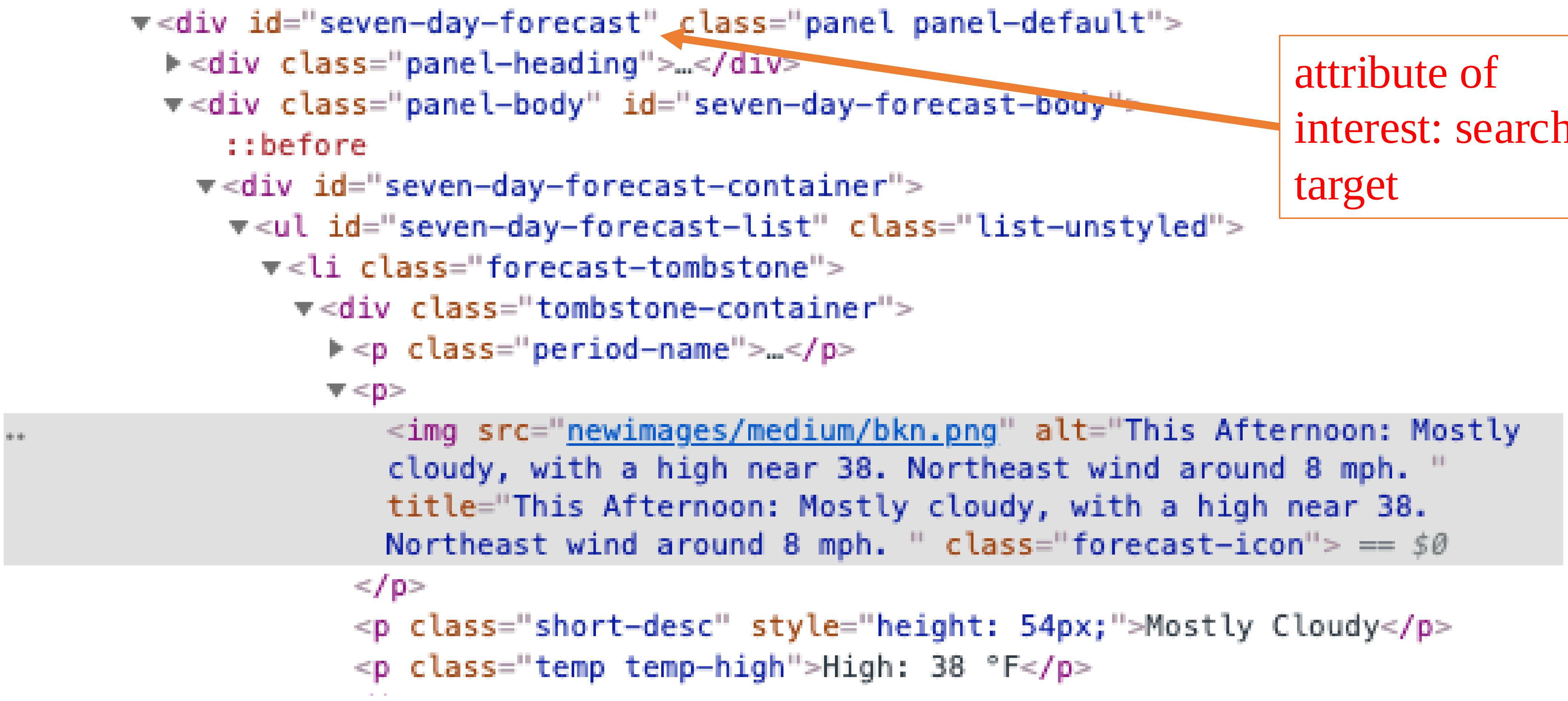
The Weather Service

- Your job is to identify a unique string(s) or attribute(s) that you can use as a string search target in your code
  - HTML tags don't work, they're too common
  - Same with CSS tags
- Typically, you'll use an id or class value – these are like variable names defined by the web developer



```
▸<div id="current-conditions" class="panel panel-default">…</div>
 <!-- 7-Day Forecast -->
▾<div id="seven-day-forecast" class="panel panel-default">
  ▾<div class="panel-heading">
     <b>Extended Forecast for</b>
```

BeautifulSoup Example, cont.



```
▼<div id="seven-day-forecast" class="panel panel-default">
  ▶<div class="panel-heading">…</div>
  ▼<div class="panel-body" id="seven-day-forecast-body">
      ::before
    ▼<div id="seven-day-forecast-container">
      ▼<ul id="seven-day-forecast-list" class="list-unstyled">
        ▼<li class="forecast-tombstone">
          ▼<div class="tombstone-container">
            ▶<p class="period-name">…</p>
            ▼<p>
..              <img src="newimages/medium/bkn.png" alt="This Afternoon: Mostly
                cloudy, with a high near 38. Northeast wind around 8 mph. "
                title="This Afternoon: Mostly cloudy, with a high near 38.
                Northeast wind around 8 mph. " class="forecast-icon"> == $0
             </p>
             <p class="short-desc" style="height: 54px;">Mostly Cloudy</p>
             <p class="temp temp-high">High: 38 °F</p>
```

attribute of interest: search target

Lab 2.3 Demo

BeautifulSoup, cont.

- Getting the right data is (for me) trial and error
- Printing out intermediate data shows where we are inside the page
- Home in on the data we want
- Use find( ) and find_all( ) to locate the target strings using tag names
- Use select() to search the documen tree for given CSS selectors
- Use print( ) to debug

the find*() methods search for the PageElements according to the Tag name and its attributes,
the select() method searches the document tree for the given CSS selector

Example of find_all()

```
# Weather forecast: see devTools for the page
import requests
from bs4 import BeautifulSoup


page =\
requests.get("https://forecast.weather.gov/MapClick.php?textField1=40.4
7&textField2=-79.96#.Xbc395NKhTa")


soup = BeautifulSoup(page.content, 'html.parser')
seven_day = soup.find(id="seven-day-forecast")
forecast_items = seven_day.find_all(class_="tombstone-container")
tonight = forecast_items[0]
print(tonight.prettify())
```

download the page

find( )

find_all( )

https://www.dataquest.io/blog/web-scraping-tutorial-python/

Example output

```
<div class="tombstone-container">
 <p class="period-name">
  This
  <br/>
  Afternoon
 </p>
 <p>
  <img alt="This Afternoon: Sunny, with a high near 66. Calm wind. " class="forecast-icon"
src="newimages/medium/skc.png" title="This Afternoon: Sunny, with a high near 66. Calm wind.
"/>
 </p>
 <p class="short-desc">
  Sunny
 </p>
 <p class="temp temp-high">
  High: 66 °F
 </p>
</div>
```

https://www.dataquest.io/blog/web-scraping-tutorial-python/

Example of find()

```
# Now drill down into 'tonight'
period = tonight.find(class_="period-name").get_text()
short_desc = tonight.find(class_="short-desc").get_text()
temp = tonight.find(class_="temp").get_text()
print(period)
print(short_desc)
print(temp)

# Output:
ThisAfternoon
Sunny
High: 66 °F
```

https://www.dataquest.io/blog/web-scraping-tutorial-python/

Example of select()

select( )

```
# Example of select( )
period_tags = seven_day.select(".tombstone-container .period-name")
print(period_tags)
```

```
[<p class="period-name">This<br/>Afternoon</p>, <p class="period-
name">Tonight<br/><br/></p>, <p class="period-name">Tuesday<br/><br/></p>, <p
class="period-name">Tuesday<br/>Night</p>, <p class="period-
name">Wednesday<br/><br/></p>, <p class="period-name">Wednesday<br/>Night</p>, <p
class="period-name">Thursday<br/><br/></p>, <p class="period-
name">Thursday<br/>Night</p>, <p class="period-name">Friday<br/><br/></p>]
```

```
periods = [pt.get_text() for pt in period_tags]
print(periods)
```

```
['ThisAfternoon', 'Tonight', 'Tuesday', 'TuesdayNight', 'Wednesday', 'WednesdayNight',
'Thursday', 'ThursdayNight', 'Friday']
```

https://www.dataquest.io/blog/web-scraping-tutorial-python/

Example of select()

```
temps = [t.get_text() for t in seven_day.select(".tombstone-container .temp")]
print(temps)
```

```
['High: 66 °F', 'Low: 49 °F', 'High: 71 °F', 'Low: 50 °F', 'High: 68 °F', 'Low: 55 °F',
'High: 67 °F', 'Low: 42 °F', 'High: 51 °F']
```

https://www.dataquest.io/blog/web-scraping-tutorial-python/

Scraping is Fragile

- Pretty tedious, eh? That's web scraping
- Scraping is *fragile* – that is, your scraping script will break if the HTML code for a page is changed
  - Which happens fairly often: web designers re-design pages all the time
  - That has no effect on regular users
- Dynamic web pages require a stronger tool like ***Selenium***
  - For example, Selenium can automate button-pushes and text-field fills
- And some sites will try to block scrapers and web crawlers

BeautifulSoup versus String Matching

- Some data is easier to find than other stuff
- – for example, 'table' is a common thing for data
- BeautifulSoup simplifies the searching
- Alternatively, you can scrape by hand – that is, you store the web page in a str variable, then using index( ) to find the tags you need, which is even more tedious.

Using APIs available for data distribution

Free APIs

https://github.com/public-api-lists/public-api-lists?tab=readme-ov-file

A Web API is **an application programming interface for the Web**

Some API examples

metro Washington Metropolitan Area Transit Authority API

YouTube Data API v3

yelp developers

reddit API

Git API

facebook API

flickr API INTEGRATION

Twitter API

DATA.GOV

DEVELOPERS — APIS

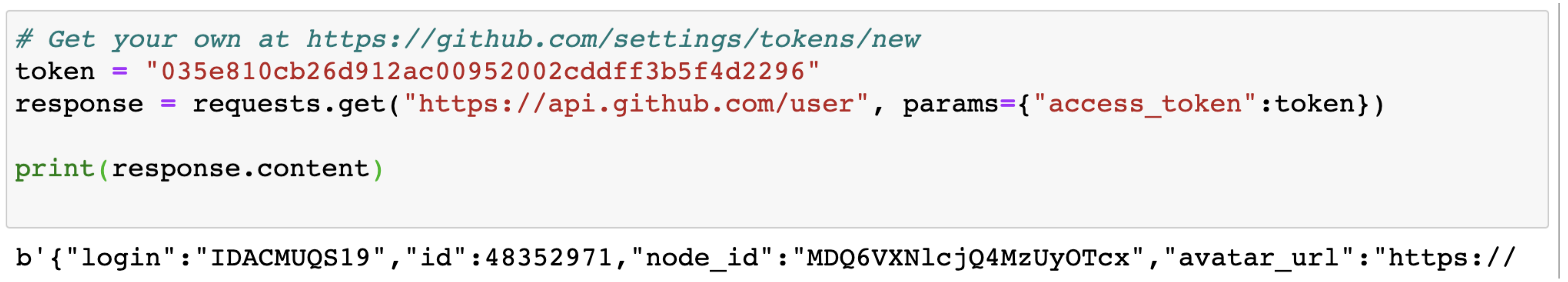
Google Cloud Vision API

REST (Representational State Transfer)ful APIs

- If you move beyond just querying web pages to web APIs such as YouTube API, Twitter API, and GitHub API, you'll most likely encounter **REST APIs** (Representational State Transfer)
- **REST** is a design architecture:
  - Uses standard HTTP interface and methods (GET, PUT, POST, DELETE)
  - **Stateless**: the server doesn't remember what you were doing including any authentication key or token. This means that each time you issue a request, you need to include all relevant information like your account key, etc.
  - REST calls will usually return information in a nice format, typically JSON (more on this later).
    - The `requests` library will automatically parse it to return a Python dictionary with the relevant data.

Querying a RESTful API: example

- To query REST API, we use the **requests** library similarly to standard HTTP requests, but we always need to include extra parameters (usually **authentication** parameters)



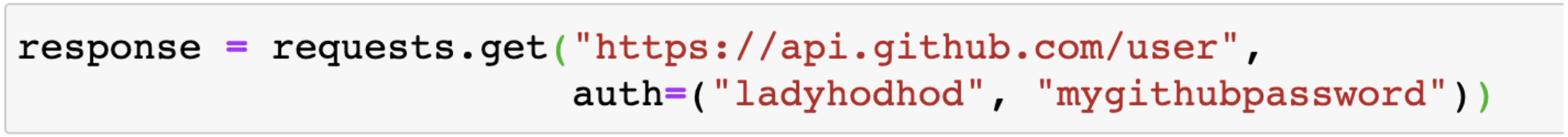
```
# Get your own at https://github.com/settings/tokens/new
token = "035e810cb26d912ac00952002cddff3b5f4d2296"
response = requests.get("https://api.github.com/user", params={"access_token":token})

print(response.content)
```

```
b'{"login":"IDACMUQS19","id":48352971,"node_id":"MDQ6VXNlcjQ4MzUyOTcx","avatar_url":"https://
```

- Get your own access token at https://github.com/settings/tokens/new
- GitHub API uses GET/PUT/DELETE to let you query or update elements in your GitHub account automatically
- **Example of REST**: server doesn't remember your last queries, for instance you always need to include your access token if using it this way

Authentication

- Most APIs use an authentication procedure that is more involved than a simple token.
- The most common approach is the “Basic authentication” , and can be used via the `requests` library as follows:



```
response = requests.get("https://api.github.com/user",
                        auth=("ladyhodhod", "mygithubpassword"))
```

- Most APIs have replaced this with some form of OAuth

API's

- Instead of web scraping, many sites have public API's that return structured data
- Most commonly return either:
  - **JSON** (**J**ava**S**cript **O**bject **N**otation)
  - **XML (**Extensible Markup Language)
- So you need a parser to interpret the data
  - Typically we use the json library

What is JSON?

- JSON stands for **J**ava**S**cript **O**bject **N**otation
- JSON is a lightweight format for storing and transporting data
- JSON is often used when data is sent from a server to a web page
- JSON is "self-describing" and easy to understand

JavaScript Object Notation is the leading data interchange format on the web.

https://www.w3schools.com/whatis/whatis_json.asp

JSON Syntax Rules

- Data is in name/value pairs (`"firstName":"John”`)
- Data is separated by commas
- **Curly braces** hold objects
- **Square brackets** hold arrays

```
"employees":[
    {"firstName":"John", "lastName":"Doe"},
    {"firstName":"Anna", "lastName":"Smith"},
    {"firstName":"Peter", "lastName":"Jones"}
]
```

https://www.w3schools.com/whatis/whatis_json.asp

JSON

Here is another example of how a person's data is structured:

```
{ "firstName": "John",
"lastName": "Smith",
"age": 27,
"address": { "streetAddress": "21 2nd Street", "city": "New York", "state": "NY",
"postalCode": "10021-3100" },
 "phoneNumbers": [ { "type": "home", "number": "212 555-1234" }, { "type":
"office", "number": "646 555-4567" } ],
"children": [ "Catherine", "Thomas", "Trevor" ],
"spouse": null }
```

JSON Data

- If an API returns JSON data, think of it as a dict: find the { }
- The keys will be String data, but the value might be complicated:
  - another dict
  - a list
- So you'll have to drill down to the part(s) that you need, using a mix of `dict['key']` and `list[index]` notation

API Requests

- Easier than scraping, but you still need to understand the data
- Commercial sites often charge real money for access to their API's
  - Some allow a few queries/time period for free
- Either:
  - write a test program and eyeball the data
  - use Chrome devTools
  - or use curl (Linux or Mac in a terminal window; can be downloaded for Windows: https://curl.haxx.se/windows/)

Saving the Data

- It's not required that data downloaded from an API in JSON format must be stored locally as JSON data
- You could use CSV format – we'll see that next
- But to store as JSON, use:

```
mytable.to_json(filename)
```

where mytable is a DataFrame. We'll see a dataframe also next.

- And to read it, use:

```
mytable = pd.read_json(filename)
```

Reading JSON

```
import pandas as pd
import requests
import json
# Using pandas to read a JSON file directly
df = pd.read_json("df.json")

# Reading a JSON response from a web API directly to Python
url = "https://api.example.com/df"
response = requests.get(url)
# Parse the JSON response
df = pd.json_normalize(json.loads(response.text))
```

https://pandas.pydata.org/docs/reference/api/pandas.read_json.html

Example of Parsing JSON

data = [{'Version': 1, 'Key': '1310', 'Type': 'City', 'Rank': 35, 'LocalizedName': 'Pittsburgh', 'EnglishName': 'Pittsburgh', 'PrimaryPostalCode': '15219', 'Region': {'ID': 'NAM', 'LocalizedName': 'North America', 'EnglishName': 'North America'}, 'Country': {'ID': 'US', 'LocalizedName': 'United States', 'EnglishName': 'United States'}, 'AdministrativeArea': {'ID': 'PA', 'LocalizedName': 'Pennsylvania', 'EnglishName': 'Pennsylvania', 'Level': 1, 'LocalizedType': 'State', 'EnglishType': 'State', 'CountryID': 'US'}, 'TimeZone': {'Code': 'EDT', 'Name': 'America/New_York', 'GmtOffset': -4.0, 'IsDaylightSaving': True, 'NextOffsetChange': '2024-11-03T06:00:00Z'}, 'GeoPosition': {'Latitude': 40.441, 'Longitude': -79.996, 'Elevation': {'Metric': {'Value': **219.0**, 'Unit': '**m**', 'UnitType': 5}, 'Imperial': {'Value': 718.0, 'Unit': 'ft', 'UnitType': 0}}}, 'IsAlias': False, 'SupplementalAdminAreas': [{'Level': 2, 'LocalizedName': 'Allegheny', 'EnglishName': 'Allegheny'}], 'DataSets': ['AirQualityCurrentConditions', 'AirQualityForecasts', 'Alerts', 'DailyAirQualityForecast', 'DailyPollenForecast', 'ForecastConfidence', 'FutureRadar', 'MinuteCast', 'ProximityNotification-Lightning', 'Radar']}]

Write code to retrieve the Elevation – Value and Unit

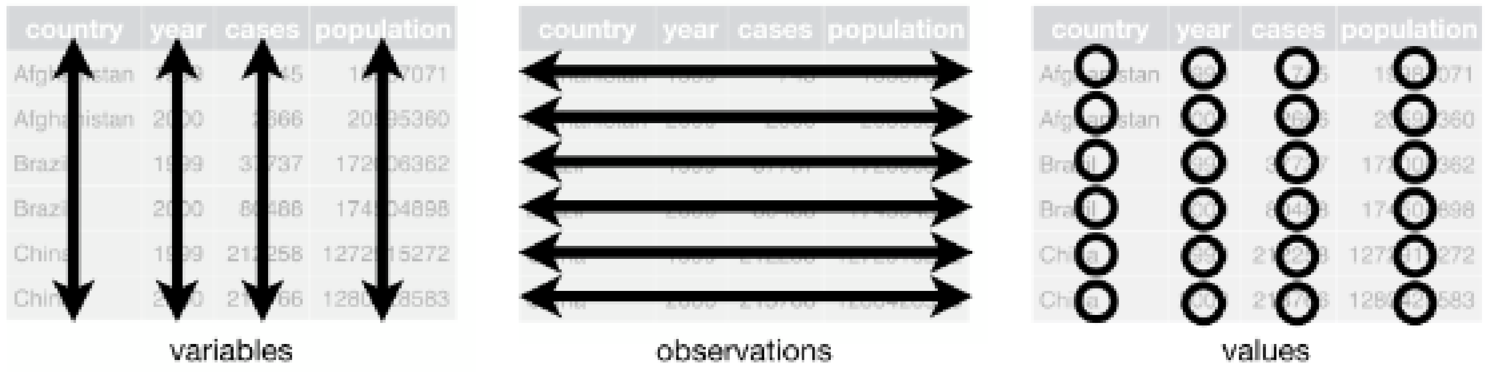
```
print(f"Elevation ={data[0]['GeoPosition']['Elevation']['Metric']['Value']}{data[0]['GeoPosition']['Elevation']['Metric']['Unit']}")
```

DataFrames

Tidy data

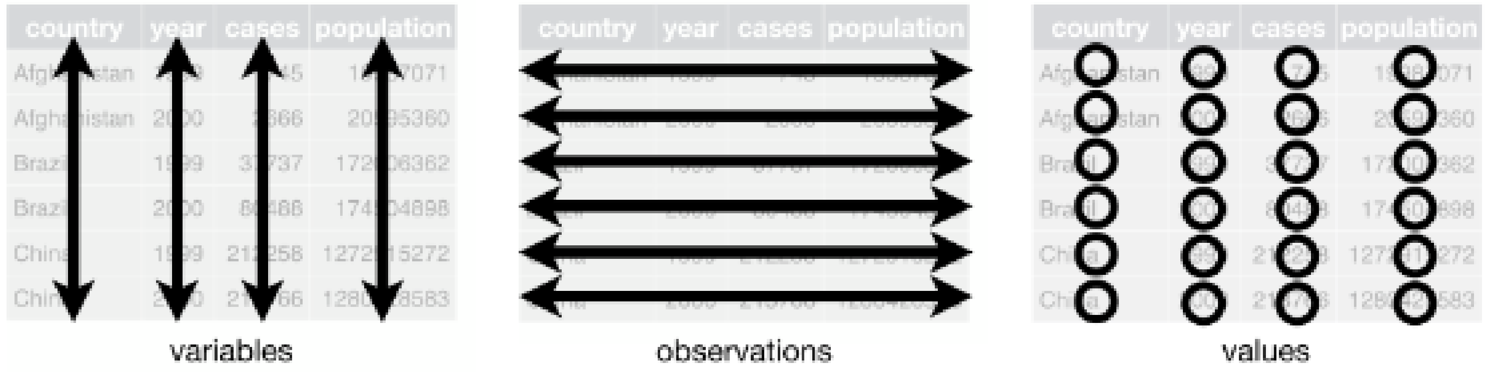
There are three interrelated rules which make a dataset tidy:

1. Each variable must have its own column.
2. Each observation must have its own row.
3. Each value must have its own cell.



Dataframes

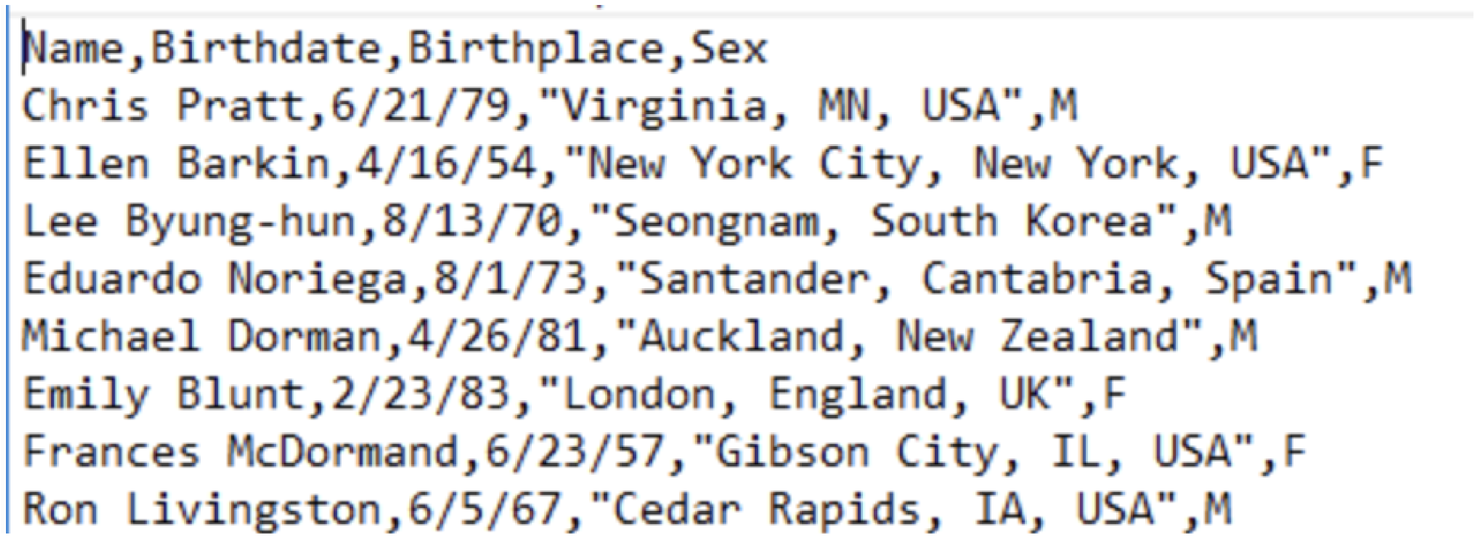
- A data frame is simply a table conforming to tidy data specs
  - Can consider it the same as an Excel spreadsheet
- Each **column** may be of a different type (e.g. numeric, character, etc.)
- The number of elements in each row must be identical



CSV file

CSV is a very common table file format:

- **Records** (rows) are delimited by a newline: `'\n'`, `"\r\n"`
  - `Carriage return followed by line feed`
- **Fields** (columns) are delimited by commas: `','`



```
Name,Birthdate,Birthplace,Sex
Chris Pratt,6/21/79,"Virginia, MN, USA",M
Ellen Barkin,4/16/54,"New York City, New York, USA",F
Lee Byung-hun,8/13/70,"Seongnam, South Korea",M
Eduardo Noriega,8/1/73,"Santander, Cantabria, Spain",M
Michael Dorman,4/26/81,"Auckland, New Zealand",M
Emily Blunt,2/23/83,"London, England, UK",F
Frances McDormand,6/23/57,"Gibson City, IL, USA",F
Ron Livingston,6/5/67,"Cedar Rapids, IA, USA",M
```

Working with CSV files

- Reading a CSV file into a dataframe
  - pd.read_csv('filename.csv'/url, index_col=False)
    - header: int (Default=0; if entries in first row are headers; None if no headers)
    - names=[name1,...] to pass names of columns which sets header=None
    - sep=char (default=, can change to "\t" for tab
    - na_filter: bool (default True) to detect missing values
    - keep_default_na: book (default True)
- Writing a dataframe to a CSV file
  - df.to_csv('filename.csv', index=False/True)

https://pandas.pydata.org/docs/reference/api/pandas.read_csv.html
https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.to_csv.html

Working with DataFrames

```
import pandas as pd
import re

happiness = pd.read_csv('happiness2023.csv') #reads first row as header by default
happiness
#pd.set_option('display.expand_frame_repr', False) #for viewing a whole row
sub_happiness = happiness[['Country name','Ladder score']]
sub_happiness.columns = ['country', 'happiness score']

#Find basic information about this dataframe
happiness.columns
#Get number of countries in the dataset
no_of_countries=len(happiness.axes[0])
no_of_countries
#Get number of columns in the dataset
columns=len(happiness.axes[1])
columns
happiness.info() # Get a snapshot view of the above information
```

Working with DataFrames

```
#Ladder score is the happiness index for a country
#Find the top five happy countries in the dataset
sorted_happiness=happiness.sort_values(by="Ladder score", ascending = False)
sorted_happiness.head(5)

#Create a new dataset with countries sorted by name
sorted_by_country=happiness.sort_values(by="Country name", ascending = True)
print(sorted_by_country)
sorted_by_country.to_csv("sortedhappinessdata.csv", index=False)

#Find all countries with Ladder score between 1 and 5 (higher the more happier)
filtered_countries = happiness[(happiness['Ladder score'] > 1.0) & (happiness['Ladder score'] < 5.0)]
filtered_countries['Country name']
```

Working with DataFrames – apply()

- The apply function in pandas is a powerful tool that allows you to apply a function along an axis of the DataFrame or on values of a Series.
- This function helps in transforming or aggregating data efficiently in a DataFrame.

Usage

- **DataFrame.apply**: Used when you want to apply a function along a specific axis (rows or columns) of a DataFrame.
- **Series.apply**: This is used when you want to apply a function element-wise on a Series.

Parameters

- **func**: The function to apply to each element or to each row/column.
- **axis**: For DataFrame, axis=0 is used to apply the function to each column, while axis=1 applies it to each row.
- **args** and **kwargs**: These are additional arguments and keyword arguments that can be passed to the function.

https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.apply.html

Working with Regular Expressions

- The Python **re** module has powerful features for handling regular expressions
- import re

```
title = "Practical Data Science"
x = re.search("^Practical*Science$", title)
```

String starts with 'Practical'   Any set of characters   Ends with 'Practical'

[findall]()	Returns a list containing all matches
[search]() [match]()	Returns a [Match object]() if there is a match anywhere in the string Checks for a match only at the beginning of the string
[split]()	Returns a list where the string has been split at each match
[sub]()	Replaces one or many matches with a string

https://www.w3schools.com/python/python_regex.asp

Working with Regular Expressions

Character	Description	Example
[]	A set of characters	"[a-m]"
\	Signals a special sequence (can also be used to escape special characters)	"\d"
.	Any character (except newline character)	"he..o"
^	Starts with	"^hello"
$	Ends with	"planet$"
*	Zero or more occurrences	"he.*o"
+	One or more occurrences	"he.+o"
?	Zero or one occurrences	"he.?o"
{}	Exactly the specified number of occurrences	"he.{2}o"
|	Either or	"falls|stays"

https://www.w3schools.com/python/python_regex.asp

Working with Regular Expressions

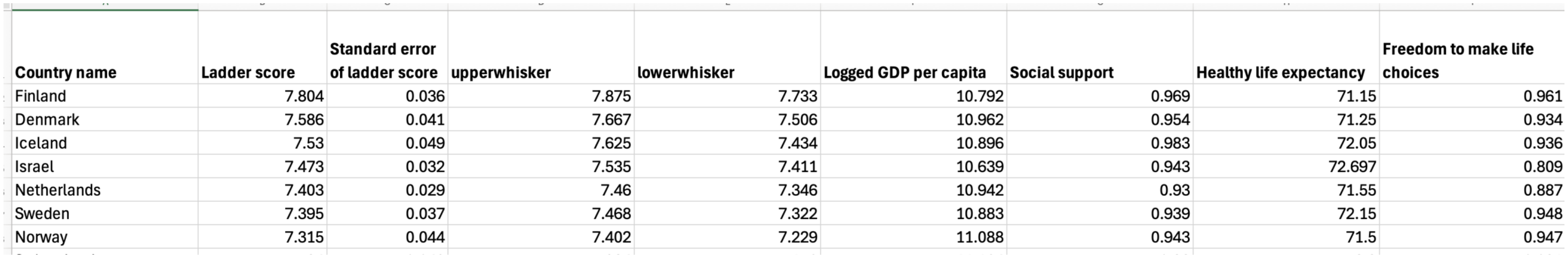
Set	Description
[arn]	Returns a match where one of the specified characters (a, r, or n) is present
[a-n]	Returns a match for any lower case character, alphabetically between a and n
[0123]	Returns a match where any of the specified digits (0, 1, 2, or 3) are present
[0-9]	Returns a match for any digit between 0 and 9
[0-5][0-9]	Returns a match for any two-digit numbers from 00 and 59
[a-zA-Z]	Returns a match for any character alphabetically between a and z, lower case OR upper case
[+]	In sets, +, *, ., |, (), $,{} has no special meaning, so [+] means: return a match for any + character in the string

https://www.w3schools.com/python/python_regex.asp

Working with Regular Expressions

```
def extract_first_three(country):
    # Using regex to find the first three letters
    match = re.search(r'^\w{3}', country) # the r indicates a raw string
    if match:
        return match.group(0)
    return None  # In case no match is found
# Apply the function to create a new column
happiness['Name'] = happiness['Country name'].apply(extract_first_three)
happiness['Name']

#Find all countries whose name starts with P
def starts_with_p(country):
     return re.match(r'^P', country) is not None # the r indicates a raw string
happiness[happiness['Country name'].apply(starts_with_p)]
```

Happiness Data
Demo of DataFrame & Regular Expressions



Country name	Ladder score	Standard error of ladder score	upperwhisker	lowerwhisker	Logged GDP per capita	Social support	Healthy life expectancy	Freedom to make life choices
Finland	7.804	0.036	7.875	7.733	10.792	0.969	71.15	0.961
Denmark	7.586	0.041	7.667	7.506	10.962	0.954	71.25	0.934
Iceland	7.53	0.049	7.625	7.434	10.896	0.983	72.05	0.936
Israel	7.473	0.032	7.535	7.411	10.639	0.943	72.697	0.809
Netherlands	7.403	0.029	7.46	7.346	10.942	0.93	71.55	0.887
Sweden	7.395	0.037	7.468	7.322	10.883	0.939	72.15	0.948
Norway	7.315	0.044	7.402	7.229	11.088	0.943	71.5	0.947

Working with datetime

```
import datetime as dt
x = dt.datetime.now() #returns the current date and time as follows:
print(x)
2024-09-06 10:49:38.852887
```

```
#Extracting information from a datetime string
print(x.strftime("%B")) #returns full month name
%a %A %w returns weekday
%d day of month
%b and %B month name %m month as number
%y %Y returns year
%H (hour 24) %I (hour 12) %p (AM/PM)
%M minute %S second
```

Working with datetime

```
# Create a specific date
d = datetime.date(2024, 9, 8)
print("Specific date:", d)

# Create a specific datetime
dt = datetime.datetime(2024, 9, 8, 15, 30)
print("Specific datetime:", dt)

# Adding days to a date
new_date = today + datetime.timedelta(days=10)
print("Date after 10 days:", new_date)

# Subtracting time
new_datetime = now - datetime.timedelta(hours=5)
print("5 hours before now:", new_datetime)
```

Working with datetime

```
# Format datetime as a string
formatted_datetime = now.strftime("%Y-%m-%d %H:%M:%S")
print("Formatted datetime:", formatted_datetime)

# Parse string to datetime
parsed_datetime = datetime.datetime.strptime("2024-09-08 15:30", "%Y-%m-%d
%H:%M")
print("Parsed datetime:", parsed_datetime)
from datetime import timezone

# Convert to a different timezone
eastern = timezone(datetime.timedelta(hours=-5))  # Eastern Standard Time (no
daylight saving)
eastern_time = utc_now.astimezone(eastern)
print("Eastern Time:", eastern_time)
```

Working with the OS

import os

- os module helps interact with the operating system by creating files and directories, manage files and directories, environment, processes etc.

os.exit() to exit a process with a status

os.abort() to terminate immediately

os.fork() forks a child process and os.pipe() directs output

os.chroot() to change root directory of the current process

os.getenv() returns value of specified environment variable

os.getpid() returns the process id

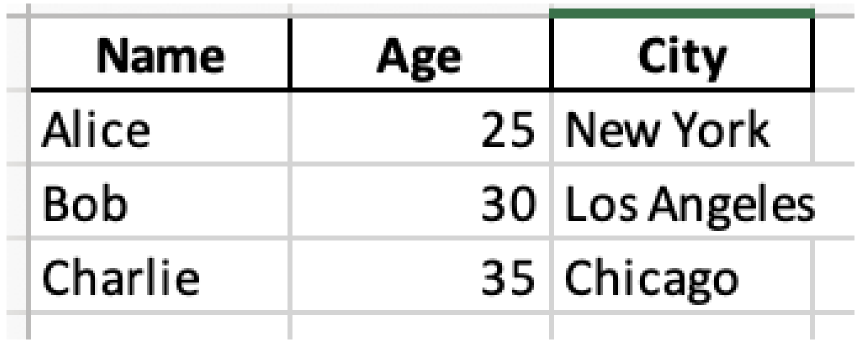
os.getppid() returns parent process id

Working with the File System

```
import os
os.getcwd() #returns the current working directory
os.chdir(path) #will set the current working directory to path
os.listdir(path) #will return the contents of a directory
os.walk(path) #will generate file names in the directory tree
os.stat(file_path) #returns file information
os.path.isfile() and os.path.isdir() #checks file types
os.makedirs() #helps create a directory recursively
shtuil.copy2 #helps copy one directory content to another
shtulil.move() #helps move files
os.remove() #helps delete a file
shutil.remtree() #helps remove a directory tree
```

Writing to an Excel file

```
# Using the pandas library and openpyxl
pip install --upgrade pandas openpyxl
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie'],
    'Age': [25, 30, 35],
    'City': ['New York', 'Los Angeles', 'Chicago']
}
# Convert the dictionary to a DataFrame
df = pd.DataFrame(data)
file_path = 'output.xlsx'
# Write the DataFrame to an Excel file
df.to_excel(file_path, index=False, engine='openpyxl')
print('Data written successfully to', file_path)
```



Name	Age	City
Alice	25	New York
Bob	30	Los Angeles
Charlie	35	Chicago

Reading from Excel file

Reading Excel df

```
# Path to the Excel file
file_path = 'output.xlsx'
# Read the Excel file into a dataframe
df = pd.read_excel(file_path, engine='openpyxl')
print(df)
```

Adidas Vs Nike Dataset

```
import pandas as pd
# For reading Excel file
sales = pd.read_excel("AdidasNike.xlsx", sheet_name="AdidasNike")
# Display the first few rows of the dataset
print(sales.head())
# Number of rows and columns in the dataset
print("Dimensions of the dataset:", sales.shape)
# Structure of the dataset
print("Structure of the dataset:")
sales.info()
#------------------------------------------------
#What are the observations from a preliminary examination of the dataset?
#There are 3268 rows and 10 columns
#Product id, …., Last Visited are the attributes
```

```
Dimensions of the dataset: (3268, 10)
Structure of the dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3268 entries, 0 to 3267
Data columns (total 10 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   Product Name   3268 non-null   object
 1   Product ID     3268 non-null   object
 2   Listing Price  3268 non-null   int64
 3   Sale Price     3268 non-null   int64
 4   Discount       3268 non-null   int64
 5   Brand          3268 non-null   object
 6   Description    3265 non-null   object
 7   Rating         3268 non-null   float64
 8   Reviews        3268 non-null   int64
 9   Last Visited   3268 non-null   object
```

Relational Database Support in Python

- Relational databases (the kind that use SQL) can be accessed from within a Python script to fetch or store data
  - Commercial Databases: Oracle, SQL Server, Access, DB2
  - Freeware Databases: MySQL, Postgres, SQLite, MariaDB
- The advantages of using a database include:
  - very stable program, not yours, is handling the storage/files
  - SQL is standard and powerful
  - Normalized tables improve reliability
- Disadvantages include:
  - (might have) slower access
  - SQL is limited in terms of programming capabilities

SQLite

- An actual relational database management system (RDBMS)
- Unlike most systems, it is a ***serverless* model**, applications directly connect to a file
- Allows for simultaneous connections from many applications to the same database file (but not quite as much concurrency as client-server systems)
- All operations in SQLite will use SQL (Structured Query Language) command issued to the database object
- You can enforce foreign keys in SQLite, but we won't bother for our labs

https://www.sqlite.org/whentouse.html

SQL in Python

- Python has several libraries that help with SQL, pick one
  - We'll be using SQLite, because it's free and included in Python
  - For others, you may need to install the library before import, like Oracle
- To use SQLite:

```
import sqlite3
```

Working with a Database (SQLite)

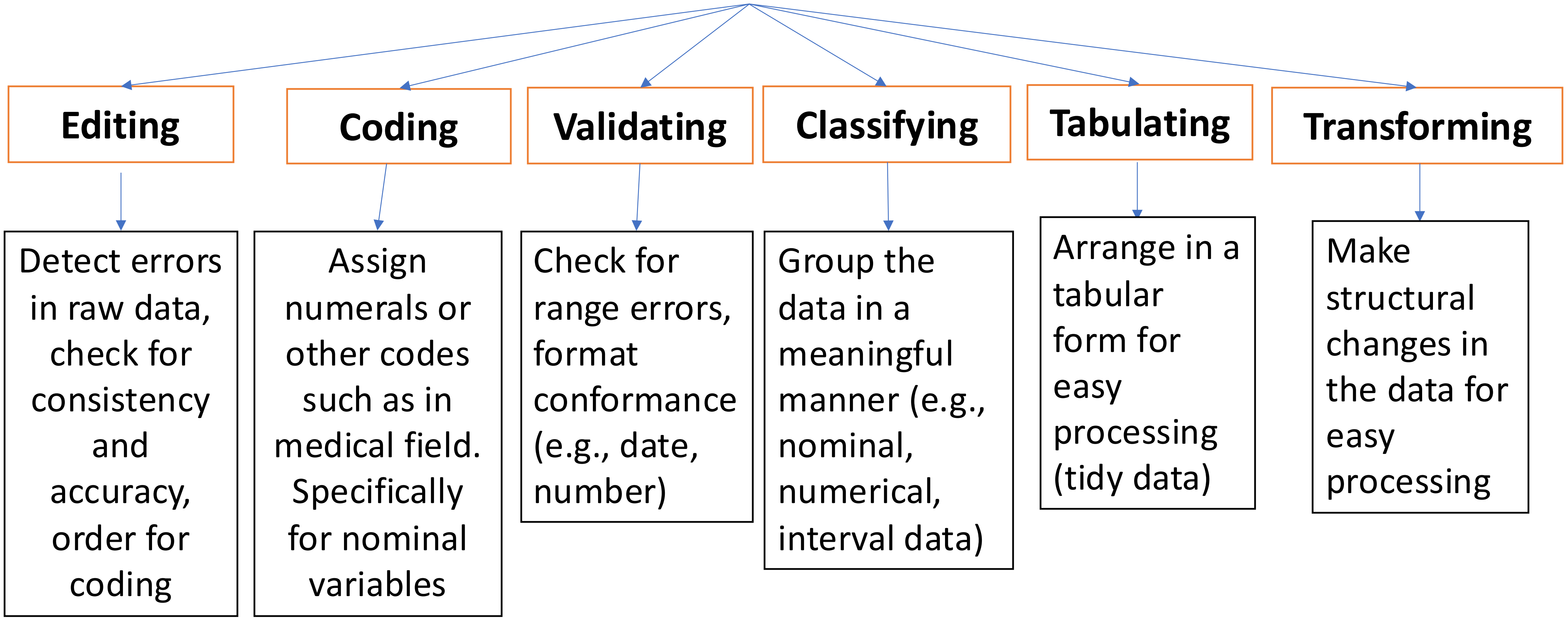
```
import sqlite3
conn = sqlite3.connect("sales.db")
cursor = conn.cursor() #Use cursor for later operations on the db
    # do your stuff – example create a table as shown below
conn.close()
```

```
cursor.execute(""" CREATE TABLE sales (
id INTEGER PRIMARY KEY, product_name TEXT)""")
cursor.execute("INSERT INTO sales VALUES (1, 'T-Shirt')")
conn.commit()
query="SELECT * FROM sales"
cursor.execute(query)
rows = cursor.fetchall()    # possibly many rows
for row in rows:
   print(row)
```

Data Preparation

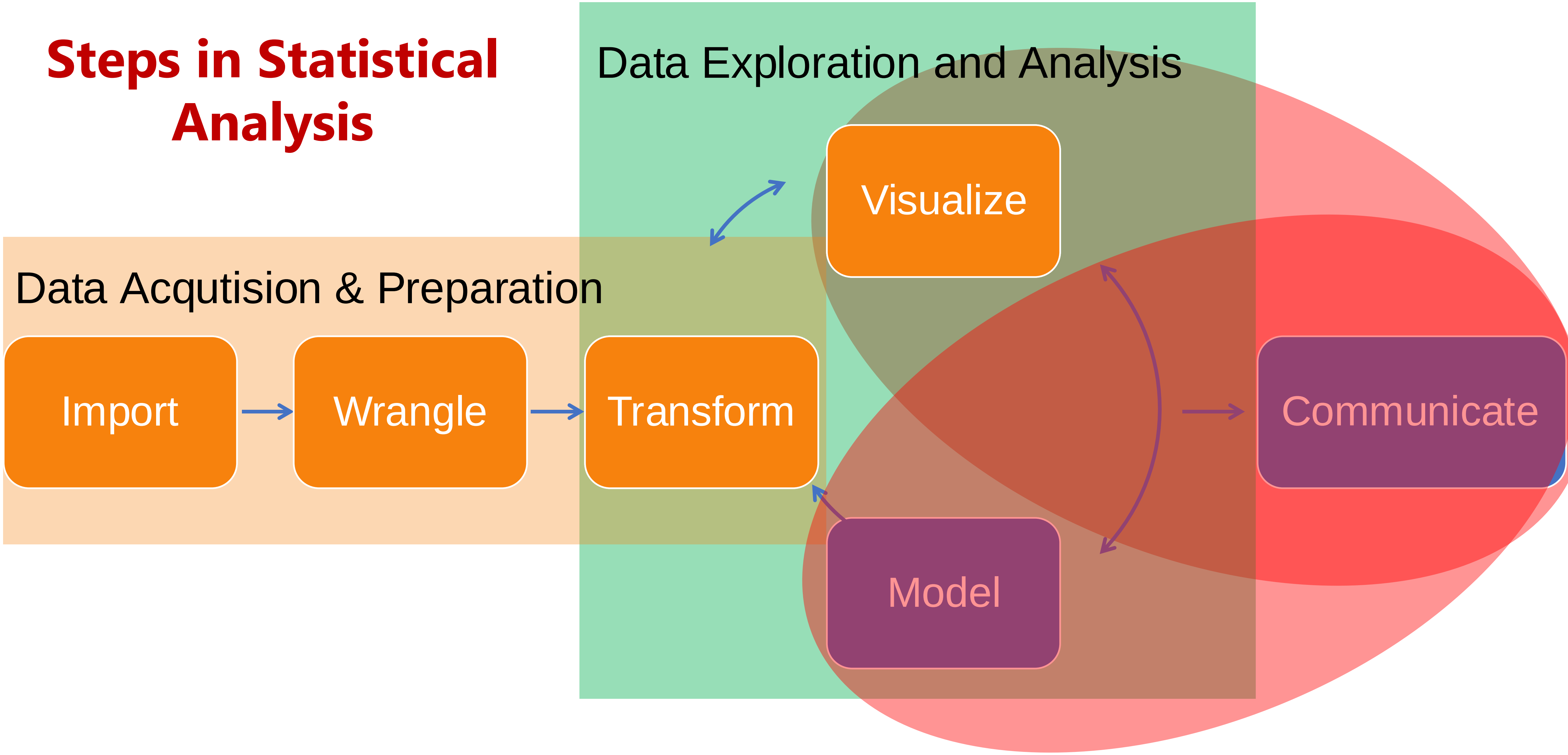
- Data preparation encompasses a broad set of processes that include data cleaning but also extend to more advanced preparation techniques such as data aggregation and transformation for specific analytical purposes.

Data Preparation for Statistical Analysis



- **Editing**: Detect errors in raw data, check for consistency and accuracy, order for coding
- **Coding**: Assign numerals or other codes such as in medical field. Specifically for nominal variables
- **Validating**: Check for range errors, format conformance (e.g., date, number)
- **Classifying**: Group the data in a meaningful manner (e.g., nominal, numerical, interval data)
- **Tabulating**: Arrange in a tabular form for easy processing (tidy data)
- **Transforming**: Make structural changes in the data for easy processing

Steps in Statistical Analysis



R for data science. " O'Reilly Media, Inc.".

Prepare the Data

1. **Select data:** Determine which data sets will be used and document reasons for inclusion/exclusion.
2. **Clean data:** Often this is the lengthiest task. Without it, you'll likely fall victim to garbage-in, garbage-out. A common practice during this task is to correct, impute, or remove erroneous values.
3. **Construct data:** Derive new attributes that will be helpful. For example, derive someone's body mass index from height and weight fields.
4. **Integrate data:** Create new data sets by combining data from multiple sources.
5. **Format data:** Re-format this or as necessary. For example, you might convert string values that store numbers to numeric values so that you can perform mathematical operations.

Note: This process is known as ETL (extract, transform, load). You can set up your own ETL pipeline to deal with all of this or use an integrated customer data platform to handle the task all within a unified environment.

Ensuring Data Quality

- **Validity**
  - Data must reflect real world condition, so conclusions are statistically valid
  - Data accuracy is important prerequisite for validity
  - Minimize error (e.g., recording or interviewer bias, transcription error, sampling error) to a point of being negligible.
- **Reliability**
  - Data are reliable because they are measured and collected consistently.
  - Data generated are based on protocols and procedures.
  - The data should be objectively verifiable.
- **Completeness**
  - The extent to which all required data is available (no missing data)

Data Quality

- **Precision** (Think of taking a patient’s temperature vs a lab measurement)
  - Data should be sufficiently detailed
  - If the measuring equipment is not calibrated properly the data may not be precise (nor reliable)
- **Timeliness** (Think of traffic data)
  - Data is current
  - Data is available on time.
- **Integrity** (Think of voting machines or bank account statements)
  - Data have integrity when the system used to generate them are protected from deliberate bias or manipulation

So, Why is data preprocessing needed?

1. Raw data may not be in a format readily suitable for analysis
2. Raw data often comes from different sources and may be incomplete, inconsistent, or contain errors.
3. The goal is to convert this raw data into a clean, structured format that can be easily analyzed.

E-Commerce Data Preparation

Example - Customer Information

- Customer information might come from **different sources** and may be **incomplete** and might need to be **aggregated**
- **Duplicate** data might have been entered in case of manual data entry.
- There might be a need to **group customer information** based on behavior, demographics & purchase history
- Supporting information might need to be **scraped** from social media sites. Such data are unstructured and incomplete.

E-Commerce Data Preparation

Example - Sales Data

- Data entered about sales might be **incorrect** or have **missing values**
- There might be a need to **merge** sales data **across periods** or seasons
- Identification of extreme values or other indicators of **fraudulent** transactions

Example - Customer Reviews

- Need to cleanse text data for Natural language processing
  - Remove stopwords (‘and’, ‘the’, ‘is’)
  - Do stemming to reduce words to their base form – e.g., fishing, fished -> fish) to convert into suitable format

Lab 2

Python in 5 minutes

1. **Literals** ("hello", 12.3)
2. **Variables** (msg = "statistical computing")
3. **Operators** (+-* != == %% / //)
4. **Control Structures** (if if-else while for continue break)
5. **Functions** (built-in, user-defined and imported)

Built-in Data Structures

- **list** (collection of different data types is the base Python array)

  lcollection = [<item0>, <item1>, …] or [] or list()

  - is mutable unlike strings
  - Is iterable
  - Is sortable
- **tuple is an immutable list (comma separated values)**

  tcollection = (<item0>, <item1>, …)
- **set is an unordered mutable collection of nonduplicated items**

  scollection = {<item0>, <item1>, …}
- **dictionary** is a mutable collection of <unique-key, value> pairs

  dcollection = {<key, value>, <key, value>, …}

https://www.w3schools.com/python/python_lists.asp

NumPy

What is NumPy?

- List has a few drawbacks
  - Slow, partly because it can store different types of data items in the same list
  - Lacks built-in operations for data processing
- Other Python libraries make this easier and faster
  - ***numpy*** is the basic one:
    - easy to create and operate on arrays of data
  - ***scipy*** and ***pandas*** are built on numpy and provide more extensive operations for scientific and statistical data processing

NumPy Features

- Vector and array operations
  - … instead of writing for loops
- Built-in array algorithms
  - … instead of writing your own functions
- Relational and data operations
  - … similar to database operations
- Statistical operations
  - … for easy data processing

Importing numpy

- To use numpy, import the module:

```
import numpy as np    # np is the usual short name
```

- This is a standard practice - use the abbreviation 'np'
- So, functions will be `np.something()` in the rest of these slides

numpy nd-Arrays

- The n-dimensional array type, `numpy.ndarray`, is the main data structure (even though the function is array()):

```
    myarray = np.array(<some list or other data structure>)
```

- np array elements are typically either ***int64*** or ***float64***
  - Note: unlike regular Python int variables, int64 are constrained in size; regular int's can be any size – but this makes nd.array faster
  - Largest int is 2**63, a pretty big number (one bit for +/-)
  - Other types are possible, including ***string_***, a fixed-length string
  - Fixed length data makes memory access easier and faster

1D Array

```
import numpy as np
data1 = [7, 10, 1, 15, 5]# Regular Python list
print(type(data1))        # list
print(data1)              # [7, 10, 1, 15, 5]

array1 = np.array(data1)  # create a 1-d array
print(type(array1))             # <class 'numpy.ndarray'>
print(array1)                   # [7 10 1 15 5]
```

- Note the lack of commas in array1: it's not a list
  - But just typing 'array1' will display with commas, so don't worry about display, just keep in mind that it's *not* a list

np.array -> list

- You can also convert back to list for 1D arrays:

```
mylist = list(array1)
print(mylist)                     # [7, 10, 1, 15, 5]
```

- For 2D and higher, you have to convert row-by-row
  - And ask yourself why you're doing this anyway

```
row0 = list(array2[0])            # Just row 0 of array2
print(row0)                       # prints [-9,0,1]
```

Indexing

- Array elements are accessed like lists, by 0-based indexing
- For 2D and higher, either give the multiple indexes [i][j] .. , or [i, j]

```
array8 = np.arange(6)          # array([0, 1, 2, 3, 4, 5])
array8[3]                     # 3
array10[1][0]                 # 0.79913362, row 1, column 0
array10[1,0]                  # Same thing
```

Slice

- Array slices are also similar to slice in lists
- Specify as *[start:end]* or *[start:end:step]*
  - Include element at *start* index until index *end-1*
- *0* is default for start *dimension* for end and *1* for step

```
array8 = np.arange(6)      # array([0, 1, 2, 3, 4, 5])
array8[3:5]                # array([3,4])
#slice from index 3 to index 5 (not inclusive of 5)
array8[0:3] = -1           # array([-1,-1,-1,3,4,5])
```

Statistical Functions

```
print(A)                          # array([[0, 1], [2, 3], [4, 5]])
print(np.amin(A))                 # 0: overall minimum
print(np.amin(A, axis=0))         # [0, 1]: column 0 min, column 1 min
print(np.amin(A, axis=1))         # [0, 2, 4]
print(np.mean(A, axis=0))         # [2.0, 3.0]
print(np.mean(A,1))               # [0.5, 2.5, 4.5]
                                  # - notice the axis shortcut
print(np.average(A, 1))           # [0.5, 2.5, 4.5] - same thing
                                  # weights default to 1's
print(np.average(A, 1, weights = [1,2]))
# array([0.66666667, 2.66666667, 4.66666667])
#       (0*1+1*2)/(1+2), (2*1+3*2)/(1+2), (4*1+5*2)/(1+2)
```

weights [1, 2] Weighted average of each data item in the 2D array

Do Practice Lab 2 (Not Graded)

See you all next week!!!