Data Analyst

Author: KhoiVN
Github: https://github.com/vnk8071/machine-learning-learning-path

Course 1: Introduction to Data Analysis with Pandas and NumPy

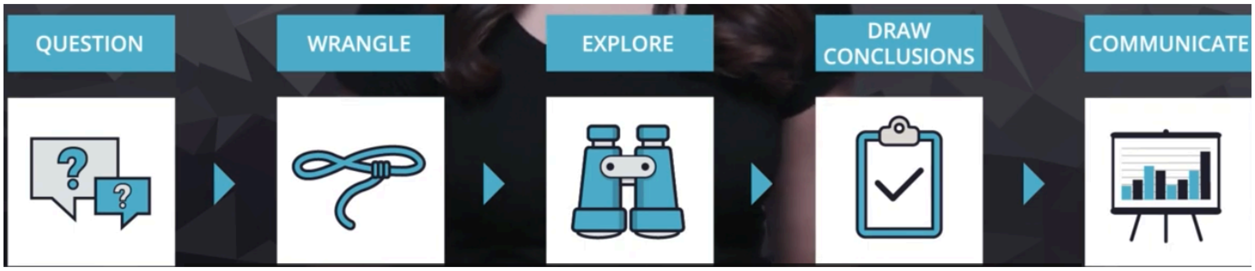
The Data Analysis Process

Problems Solved by Data Analysts

Intelligent analysis of data can result in some awesome stuff. Netflix's personalized movie recommendations, Facebook's news feed ranking algorithm, and OkCupid's predictive matching all rely on the work of data analysts.

But data analysts don't only work at tech companies! Data analysis has applications in a variety of industries.

Data Analysis Process Overview



Step 1: Ask questions
Either you're given data and ask questions based on it, or you ask questions first and gather data based on that later. In both cases, great questions help you focus on relevant parts of your data and direct your analysis toward meaningful insights.

Step 2: Wrangle data
You get the data you need in a form you can work with in three steps: gather, assess, and clean. In other words, you:

- gather the data you need to answer your questions,
- assess your data to identify any problems in your data's quality or structure, and
- clean your data by modifying, replacing, or removing data to ensure that your dataset is of the highest quality and as well-structured as possible.

Step 3: Perform EDA (Exploratory Data Analysis)
You explore and then augment your data to maximize the potential of your analyses, visualizations, and models. Exploring involves finding patterns in your data and visualizing the relationships in your data. After exploring, you can do things like remove outliers and create better features from your data, also known as feature engineering.

Step 4: Draw conclusions
This step is often approached with machine learning or inferential statistics techniques that are beyond the scope of this course. Check out the Data Scientist Nanodegree if you're interested in learning more.
In this course, you will focus on drawing conclusions with descriptive statistics and data visualizations.

Step 5: Communicate your results
You often need to justify and convey meaning in the insights you've found. Or, if your end goal is to build a system, you usually need to share what you've built, explain how you reached design decisions, and report how well it performs. There are many ways to communicate your results: reports, slide decks, blog posts, emails, presentations, or even conversations. Data visualization will always be very valuable, and you will learn data visualization fundamentals in this course.

Packages Overview

One of the biggest reasons that Python is considered the standard language for data science is its powerful packages. NumPy, pandas, and Matplotlib are three core packages for data analysis that we'll be learning to use in this course. If you are new to these packages—or need a refresher—check out the summary below highlighting their top-level features and use cases.

NumPy
- Used in scientific computing
- Provides data structures and mathematical operations
- Provides a set of core operations used in other packages

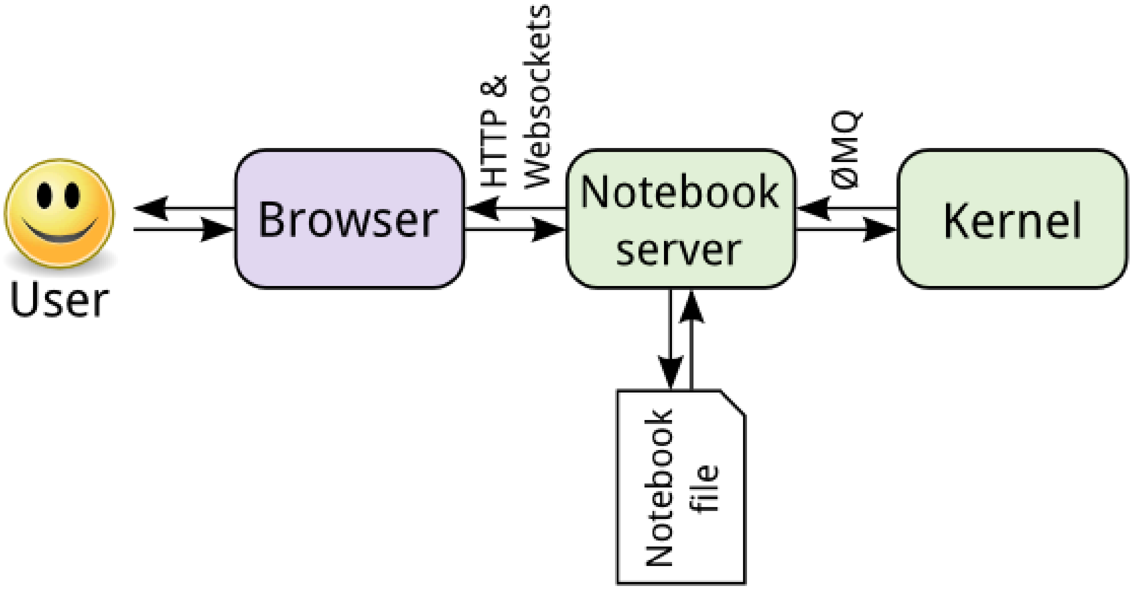
Pandas
- Data manipulation and analysis library
- Provides data structures and tools for working with structured data
- Similar working with spreadsheets using code

Matplotlib
- Plotting library
- Provides a flexible and customizable environment for creating high-quality data visualizations

Jupyter Notebooks

The Notebook is a web application that allows you to combine:
- code,
- explanatory text,
- math equations,
- data tables, and
- visualizations

all in one easily sharable document that is connected to a backend server for executing code.



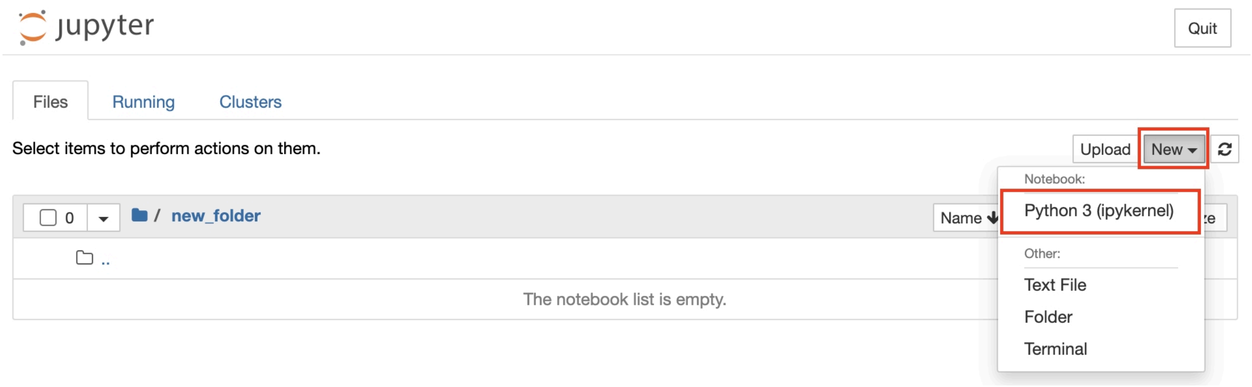
Notebook Server

The central point is the Notebook server. You connect to the server through your browser, and the notebook is rendered as a web app. The code you write in the web app is sent through the server to the kernel. The kernel runs the code and sends it back to the server, then any output is rendered back in the browser. When you save the Notebook, it is written to the server as a JSON file with a .ipynb file extension.

IPython to Jupyter

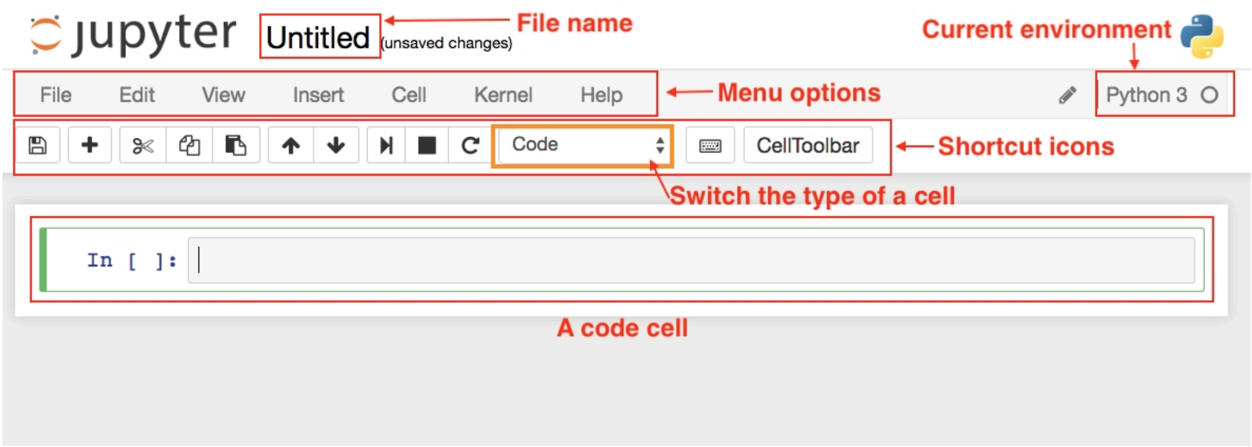
The great part of this architecture is that the kernel doesn't need to run Python. Since the Notebook and the kernel are separate, code in any language can be sent between them. For example, two of the earlier non-Python kernels were for the R and Julia languages. With an R kernel, code written in R will be sent to the R kernel where it is executed, exactly the same as Python code running on a Python kernel. IPython Notebooks were renamed to Jupyter Notebooks because notebooks became language agnostic. The new name Jupyter comes from the combination of Julia, Python, and R.

Notebook Interface



Select items to perform actions on them.

The notebook list is empty.



File name

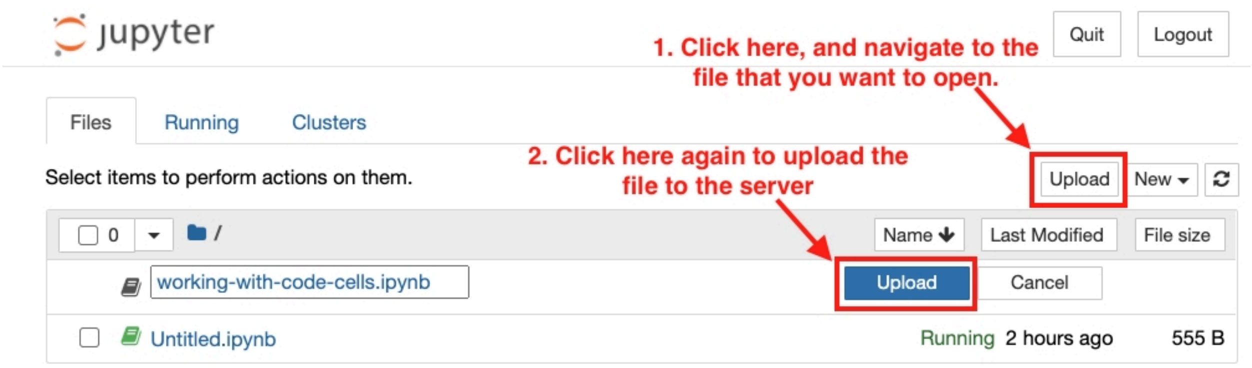
Current environment

Menu options

Shortcut icons

Switch the type of a cell

A code cell

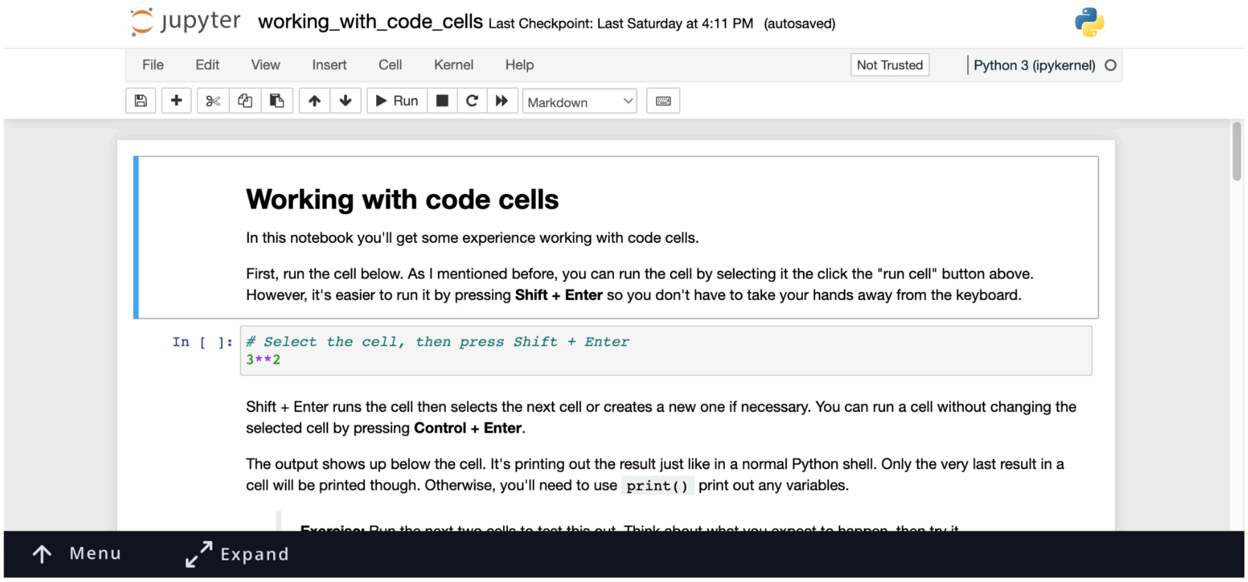


Select items to perform actions on them.

1. Click here, and navigate to the file that you want to open.

2. Click here again to upload the file to the server

Code Cells



Working with code cells

In this notebook you'll get some experience working with code cells.

First, run the cell below. As I mentioned before, you can run the cell by selecting it the click the "run cell" button above. However, it's easier to run it by pressing **Shift + Enter** so you don't have to take your hands away from the keyboard.

```
In [ ]: # Select the cell, then press Shift + Enter
        3**2
```

Shift + Enter runs the cell then selects the next cell or creates a new one if necessary. You can run a cell without changing the selected cell by pressing **Control + Enter**.

The output shows up below the cell. It's printing out the result just like in a normal Python shell. Only the very last result in a cell will be printed though. Otherwise, you'll need to use `print()` print out any variables.

> **Exercise:** Run the next two cells to test this out. Think about what you expect to happen, then try it

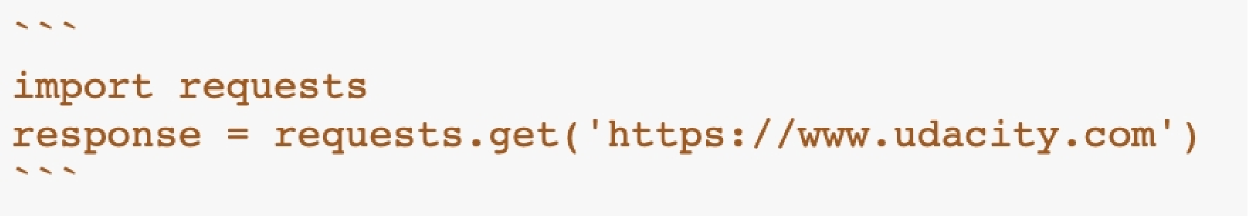
Markdown Cells

As with code cells, you press Shift + Enter or Control + Enter to run the Markdown cell, where it will render the Markdown to formatted text. Including text allows you to write a narrative alongside your code, as well as to document your code and the thoughts that went into it. Below is a brief summary of Markdown concepts.

Links
Linking in Markdown is done by enclosing link text in square brackets and the URL in parentheses, like this [Udacity's home page](https://www.udacity.com) for a link to Udacity's home page.

Emphasis
You can add emphasis through bold or italics with asterisks or underscores (* or _). For italics, wrap the text in one asterisk or underscore, e.g. _gelato_ or *gelato* renders as* gelato*.
Bold text uses two symbols, e.g. **aardvark** or __aardvark__ looks like aardvark.
Either asterisks or underscores are fine as long as you use the same symbol on both sides of the text.

Code
There are two different ways to display code, inline with text and as a code block separated from the text. To format inline code, wrap the text in backticks. A backtick is a character that can be typed in the upper left of the keyboard.



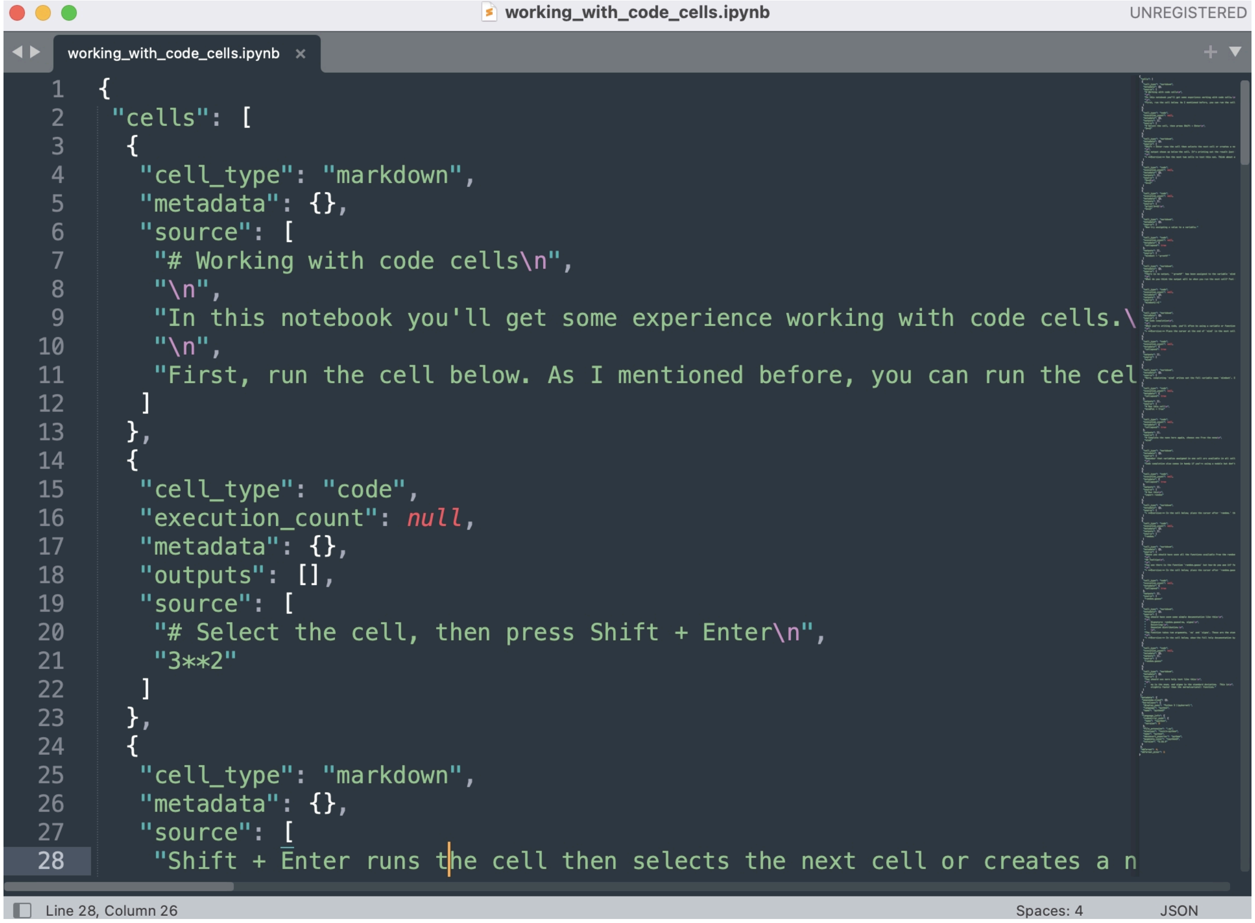
~~~
```
import requests
response = requests.get('https://www.udacity.com')
```
~~~

Math expressions
You can create math expressions in Markdown cells using LaTeX symbols. Notebooks use MathJax to render the LaTeX symbols as math symbols. To start math mode, wrap the LaTeX in dollar signs $y = mx + b$ for inline math. For a math block, use double dollar signs,

$$
y = \frac{a}{b+c}
$$

Images
In order to embed images into your Markdown cells, it is very similar to links. In this case, you are placing text within square brackets and the image location (URL or file path) in parentheses. The only difference is adding a ! at the beginning before the square brackets. The text in the square brackets becomes the alt text.
![GitHub octocat dressed as an astronaut](https://octodex.github.com/images/octonaut.jpg)

Converting Notebooks



```
{
 "cells": [
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "# Working with code cells\n",
    "\n",
    "In this notebook you'll get some experience working with code cells.\
    "\n",
    "First, run the cell below. As I mentioned before, you can run the cel
   ]
  },
  {
   "cell_type": "code",
   "execution_count": null,
   "metadata": {},
   "outputs": [],
   "source": [
    "# Select the cell, then press Shift + Enter\n",
    "3**2"
   ]
  },
  {
   "cell_type": "markdown",
   "metadata": {},
   "source": [
    "Shift + Enter runs the cell then selects the next cell or creates a n
```

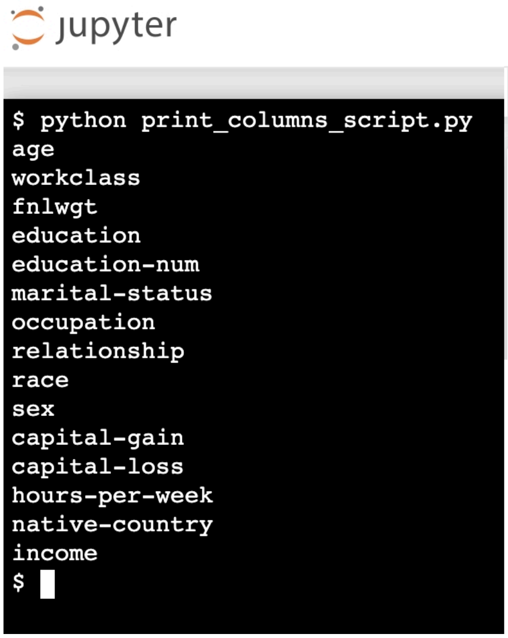
Since notebooks are JSON, it is simple to convert them to other formats. Jupyter comes with a utility called nbconvert for converting to HTML, Markdown, slideshows, etc. The general syntax to convert a given .ipynb file to another FORMAT is:

jupyter nbconvert --to FORMAT jupyter_intro.ipynb

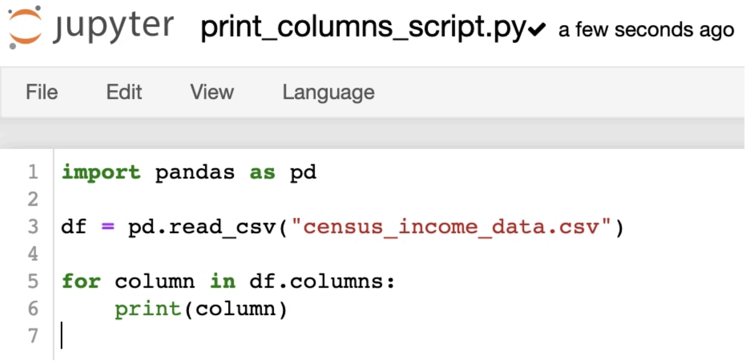
The currently supported output FORMAT could be any of the following:

- HTML (html),
- LaTeX (latex),
- PDF (pdf),
- WebPDF (webpdf),
- Reveal.js HTML slideshow (slides),
- Markdown (markdown),
- ASCII (asciidoc),
- reStructuredText (rst),
- Executable script (script),
- Notebook (notebook).

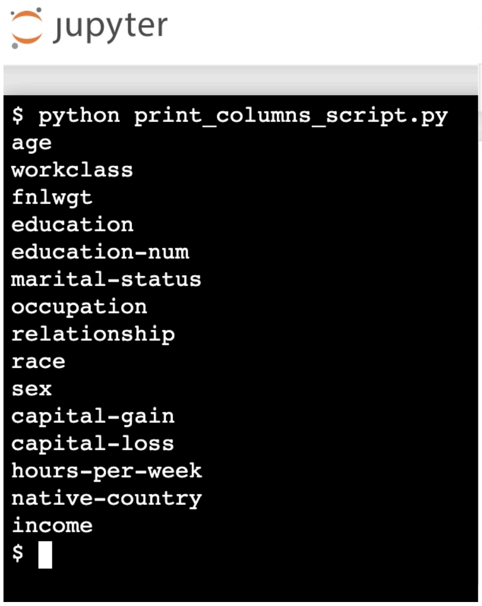
Scripting Your Analysis



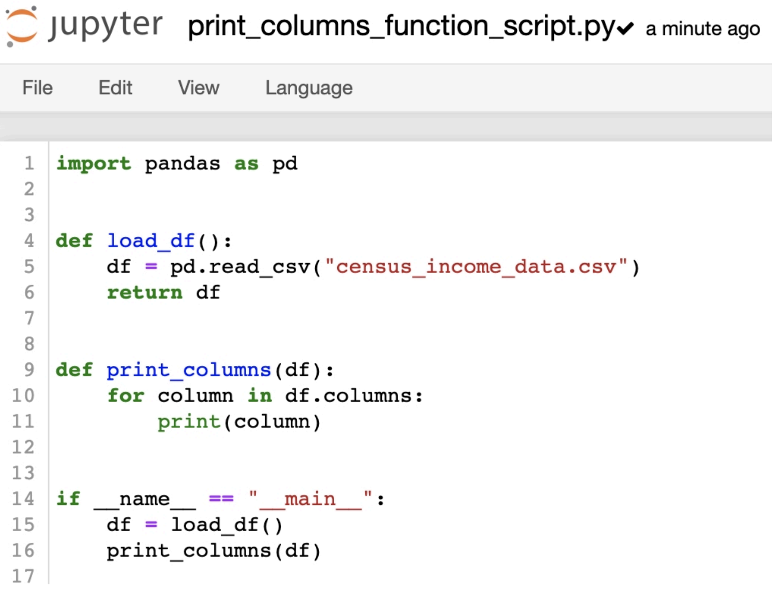
```
$ python print_columns_script.py
age
workclass
fnlwgt
education
education-num
marital-status
occupation
relationship
race
sex
capital-gain
capital-loss
hours-per-week
native-country
income
$ 
```



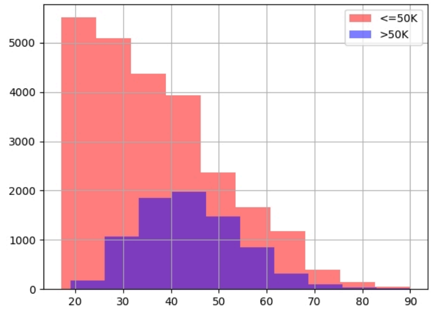
```python
import pandas as pd

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

for column in df.columns:
    print(column)

```

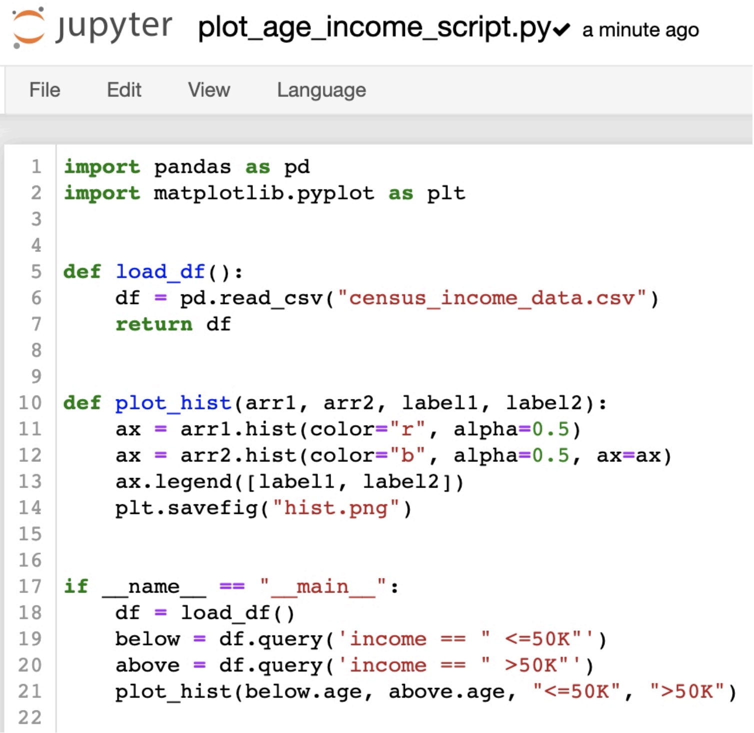


```
$ python print_columns_script.py
age
workclass
fnlwgt
education
education-num
marital-status
occupation
relationship
race
sex
capital-gain
capital-loss
hours-per-week
native-country
income
$ 
```



```python
import pandas as pd


def load_df():
    df = pd.read_csv("census_income_data.csv")
    return df


def print_columns(df):
    for column in df.columns:
        print(column)


if __name__ == "__main__":
    df = load_df()
    print_columns(df)

```





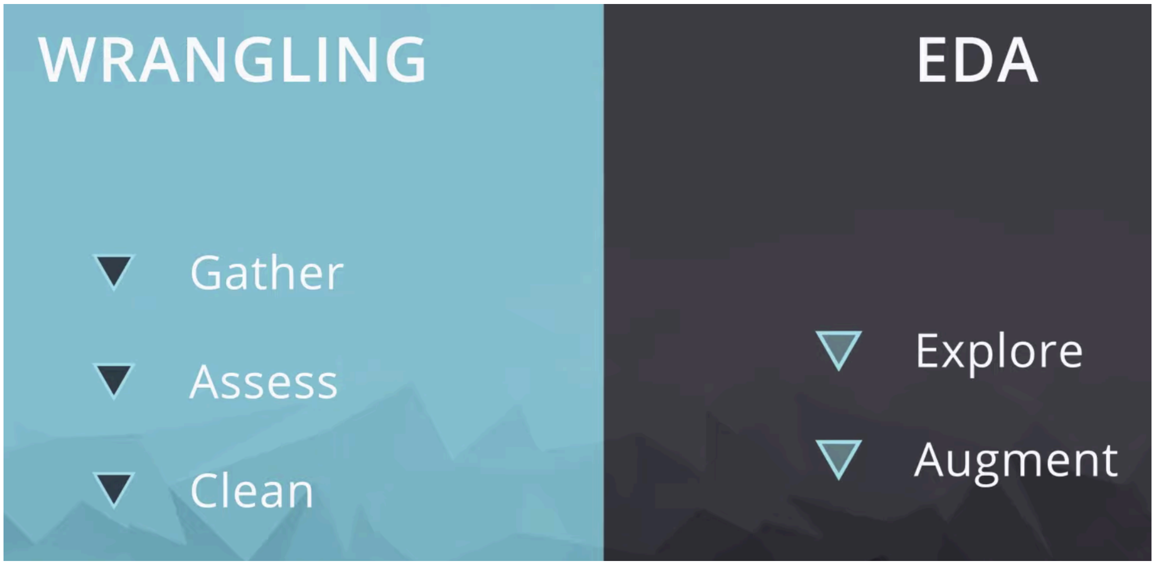
```python
import pandas as pd
import matplotlib.pyplot as plt


def load_df():
    df = pd.read_csv("census_income_data.csv")
    return df


def plot_hist(arr1, arr2, label1, label2):
    ax = arr1.hist(color="r", alpha=0.5)
    ax = arr2.hist(color="b", alpha=0.5, ax=ax)
    ax.legend([label1, label2])
    plt.savefig("hist.png")


if __name__ == "__main__":
    df = load_df()
    below = df.query('income == " <=50K"')
    above = df.query('income == " >50K"')
    plot_hist(below.age, above.age, "<=50K", ">50K")

```

Final thoughts

These were just simple examples to expose you to a different workflow. Writing and running scripts from your terminal is a very flexible and powerful way to program. This is more ideal as a development environment than Jupyter Notebook - which still works and is very useful, but more suited for things like reports.

Exploring and Inspecting Data

Data Wrangling and EDA



WRANGLING

▼ Gather

▼ Assess

▼ Clean

EDA

▽ Explore

▽ Augment

There are plenty of examples of data wrangling and EDA concepts within the industry. For example, you may need to clean large amounts of data before entering it into a database because some fields you collected on a web page are empty. Or augment the data by normalizing and correcting any spelling errors that the user may have put into a field by mistake.

Gathering Data

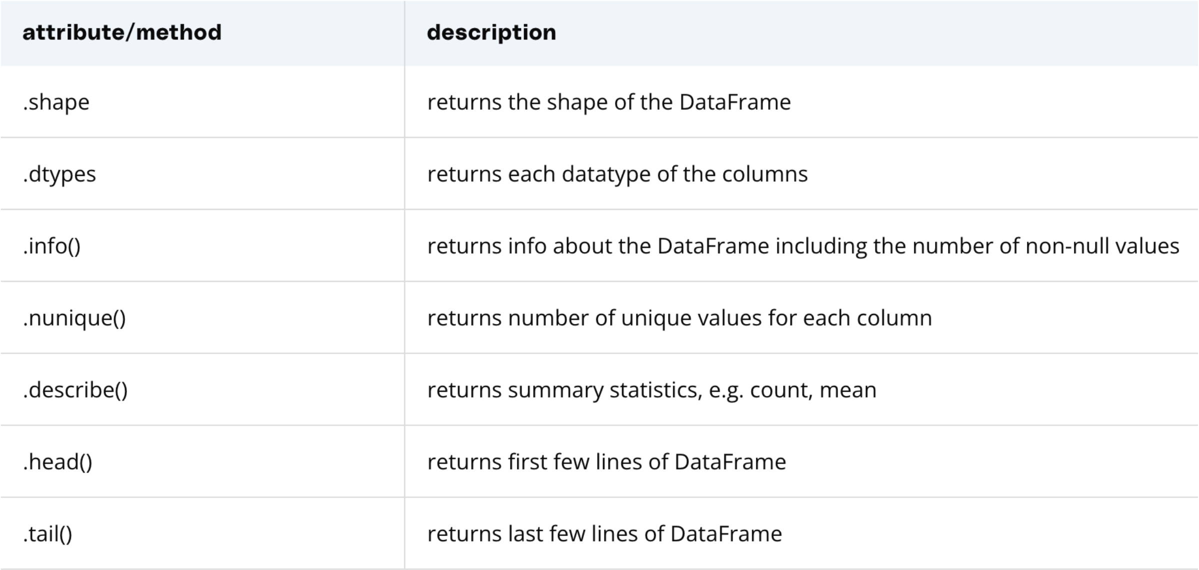
Reading CSV Files

Continuing with the practice of gathering data for analysis, we're introducing several new datasets for you to read into DataFrames. You've seen .read_csv() before, but now let's understand its functionality better.

The .read_csv() method will read any CSV-type file and transform it into a pandas DataFrame. This is incredibly useful, as you can leverage all the power of pandas on the data from the CSV. For comparison, Excel is limit bound by the number of rows it can handle, which makes analyzing large datasets difficult.

Assessing and Building Intuition

Now that we've become comfortable with reading in new and interesting datasets, our next step is to start exploring the data. Previously in the lesson, we spoke about asking questions prior to inspecting the data, or during and after our inspection.



attribute/method	**description**
.shape	returns the shape of the DataFrame
.dtypes	returns each datatype of the columns
.info()	returns info about the DataFrame including the number of non-null values
.nunique()	returns number of unique values for each column
.describe()	returns summary statistics, e.g. count, mean
.head()	returns first few lines of DataFrame
.tail()	returns last few lines of DataFrame

Manipulating Data using Pandas and NumPy

Common Issues in Data

Once you have a DataFrame, it's important to inspect the data and correct any issues that arise. Some of these issues can be:

- Incorrect data types
- Missing data
- Duplicates
- Structural problems, such as Different column names, Mismatched number of records

Basic Data Cleaning with Pandas

Missing Data: .fillna() will fill any null values in your DataFrame with the values used in the method.

Duplicates:

- .duplicated() will return True or False for any rows that are duplicates. You can specify which columns used to compare.
- .drop_duplicates() will drop duplicated given an array the length of your DataFrame with True/False. You can use the inplace=True parameter to modify your current DataFrame.

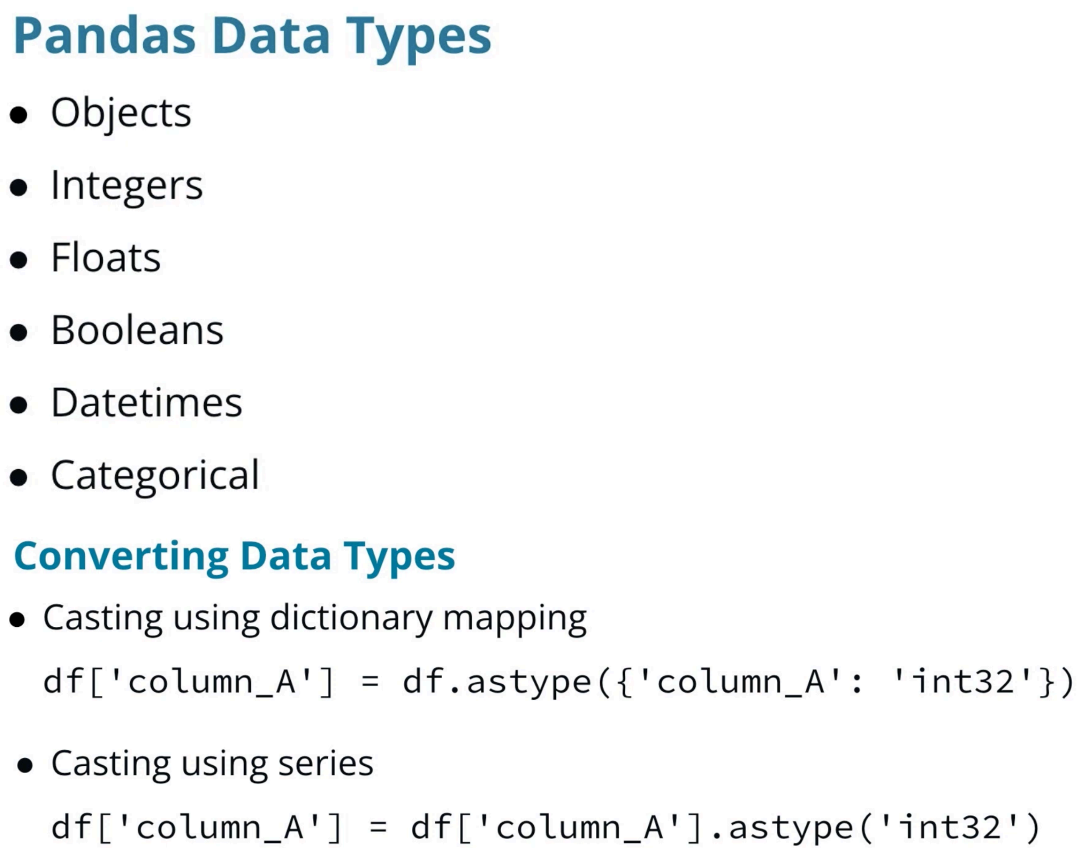
Incorrect Data Types: .to_datetime() will convert a string representation of a datetime into a datetime object. This is important when you want to work with datetimes.

Pandas Query

The first way you typically learn to select rows in a dataframe is by indexing with a mask. This is a very powerful and flexible technique, but it often involves repeating the name of the dataframe like df[df[...]...].

If you are filtering a dataframe based on the contents of that same dataframe, the query() method provides simpler syntax, which looks like this: df.query("...").

Pandas Data Types



Pandas Data Types

- Objects
- Integers
- Floats
- Booleans
- Datetimes
- Categorical

Converting Data Types

- Casting using dictionary mapping

```
df['column_A'] = df.astype({'column_A': 'int32'})
```

- Casting using series

```
df['column_A'] = df['column_A'].astype('int32')
```

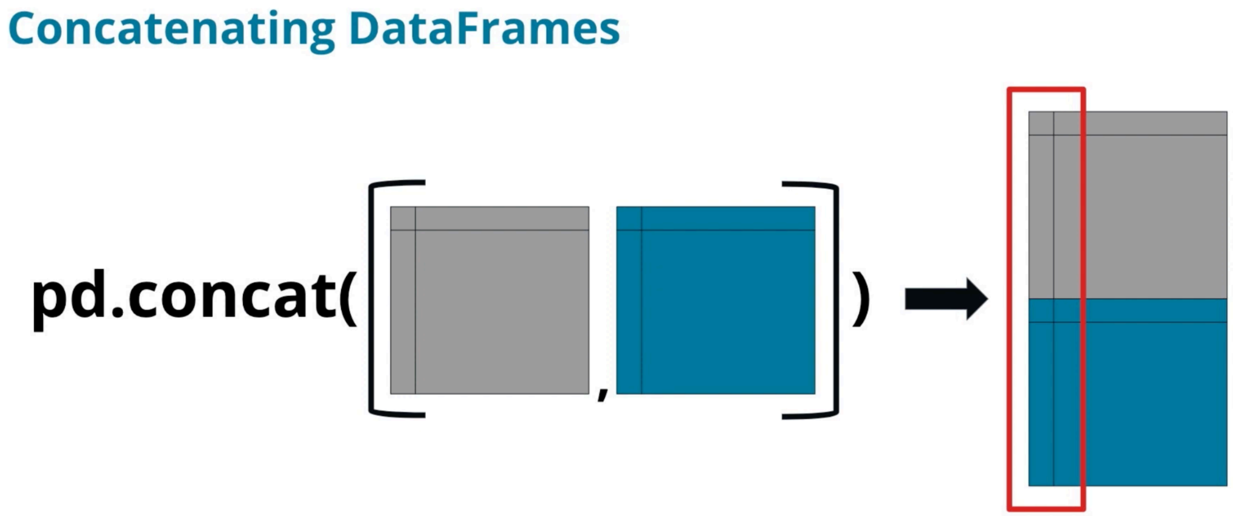
Data Type Optimization

Luckily in pandas, there are some options that we can use to optimize how our DataFrames are stored in memory. Since pandas uses NumPy, and NumPy is based on C, we can leverage what kind of datatypes we use to reduce our memory footprint and give us flexibility in working with larger datasets.

Here is a list of a few techniques that can be used:

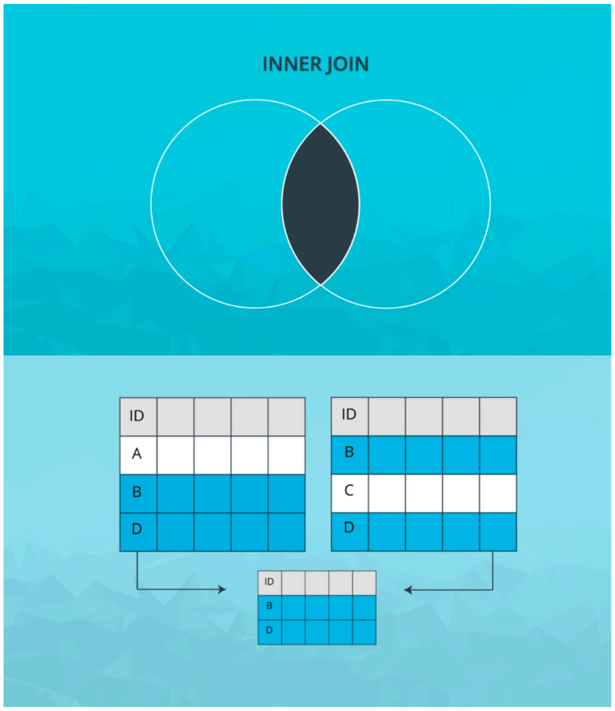
- Ints and floats have different bit sizes. If the range of values in your column fits lower bit sizes, try reducing it — for example, changing an int64 to int16. You can use NumPy iinfo for integers and NumPy finfo for floats to see the minimum and maximum values that can be represented by each size. For example, np.iinfo("int16") will tell you that int16 columns can hold values between -32,768 and +32,767
- If a numeric value is stored as a float, but in reality it is an int, changing the datatype may reduce the memory required to store those values.
- If a string's column is repetitive or not very unique, it can be changed to the category datatype. This reduces space as it only has to store the unique values in memory instead of storing every single character of every string.

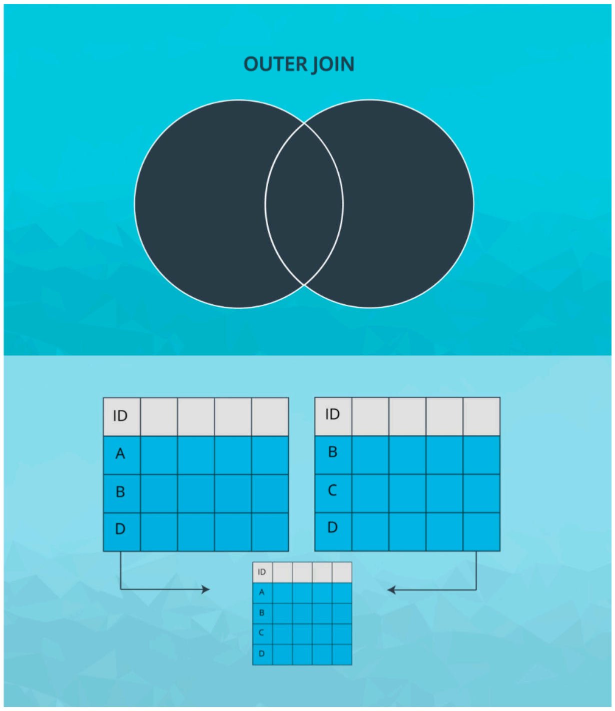
Concatenating Data

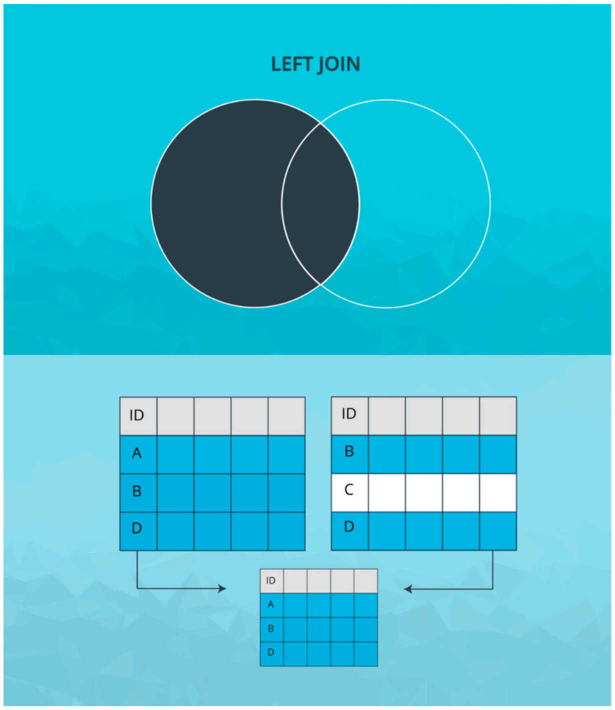


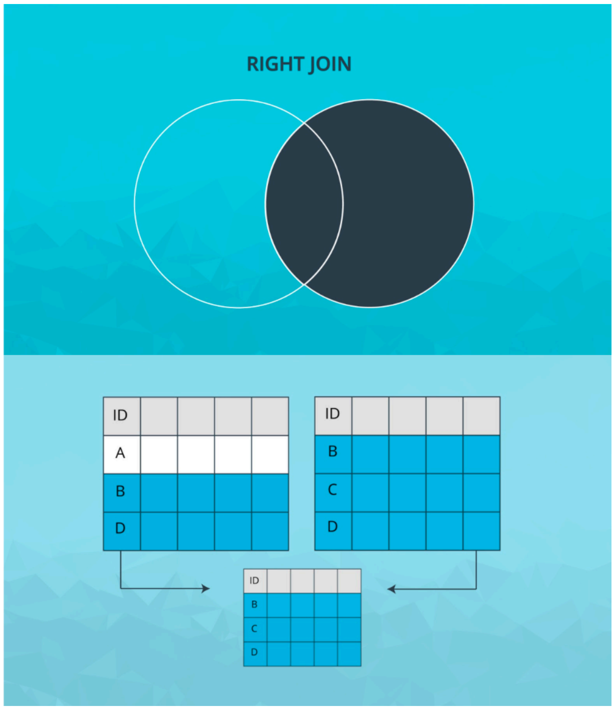
Types of Merges

Inner Join - Use intersection of keys from both frames.
Outer Join - Use union of keys from both frames.
Left Join - Use keys from left frame only.
Right Join - Use keys from right frame only.

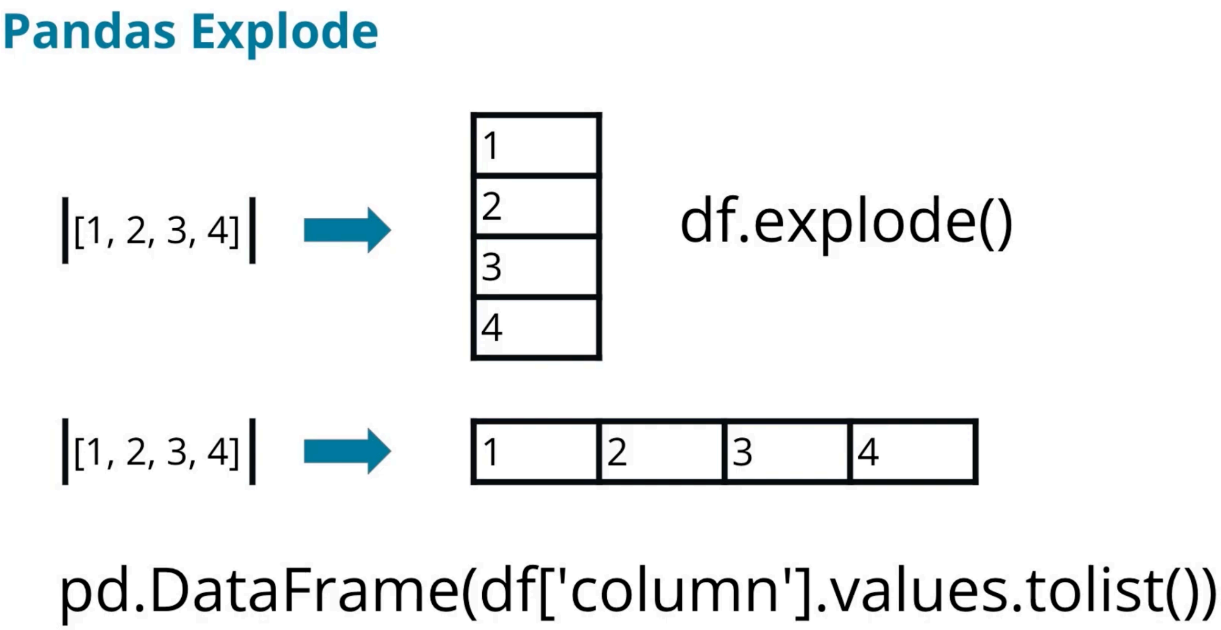








Pandas Explode



Pandas Explode

```
df.explode()
```

```
pd.DataFrame(df['column'].values.tolist())
```

Communicating Results

Optimization Using NumPy

To analyze and communicate our data more quickly, we'll utilize the underlying technology that empowers pandas to be so efficient. NumPy is written in C, and while it is a Python library, it uses C for the majority of its computation. Because of this, it gives us the flexibility and ease of use with Python, and the power of C!

Pandas Groupby

Pandas' answer to this is the .groupby() method, a method that allows you to group your data by a specific column(s) and create aggregate information about those groupings. It also allows for group-specific transformations. In this section's notebook, we will get familiar with the groupby method and use it to get summary statistics about different groups in our data.

Summation

Summation is the process of adding together numerical values to achieve a cumulative total. For pandas, you can do this on a Series, Group, or a DataFrame by using the .sum() method. For DataFrames, one can calculate data either vertically, .sum(axis=0), or horizontally, .sum(axis=1), via the axis parameter. While this process may seem trivial, it is vitally important for all of the downstream methods we'll be talking about in this lesson.

Summing over a range of data has special use cases as well. Any time we need to gather grand totals for some comparison, summation will be the main method. Examples could be summing reports based on financial data, or counting errors that trigger an alert for monitoring APIs.

Measures of Center

Measures of center, which include mean, median, and mode, are essential in data analytics for providing a summary of a dataset focused on its center. These three measurements help analysts gain a deeper understanding of the data by defining a value that it determines is a representation of the dataset.

Mean - .mean(): Otherwise known as average. Mean is the sum of all numbers in a set, divided by the number of values in the set.
Example: [1, 2, 3, 4, 5] has an average of 3 (15 divided by 5).

Median - .median(): Median is the center value in a set. In order to find the center value, you must always sort your set of values first. If the number of values in the set is even, take the midpoint between the two center values.
Example:
[2, 1, 4, 5, 3] sorts to [1, 2, 3, 4, 5], 3 is the median
[2, 1, 4, 5, 3, 6] sorts to [1, 2, 3, 4, 5, 6], 3.5 is the median

Mode - .mode(): Mode is the value that has the highest frequency in a set.
Example: [1, 2, 2, 3, 3, 3, 4, 4, 4, 4], 4 is the mode

Why are these important?
For one, measurements of center provide a quick and easy way to know basic characteristics of your dataset. But they also are fundamental for advanced analysis, including measures of spread and identifying outliers. Overall, finding the mean and median will be so prevalent in your future data analysis work, it will become second nature.

Measures of Spread

Measures of spread include quantiles, standard deviation, and variance. Without getting too deep into the statistical details, it's useful to have a broad understanding of these measures and what they mean in your data.

Quantiles - .quantile(): Quantiles are values that split a group of data into equal parts, helping to show how the data is spread out.
Example:

- [1, 2, 3, 4, 5]
- 25% quantile = 2
- 50% quantile = 3 (commonly referred to as the median)
- 75% quantile = 4

Standard Deviation - .std(): Standard deviation is a measure that shows how much the individual numbers in a group differ from the average, giving an idea of how spread out the data is. The reason we can compare individual numbers to the group is because S.D. is measured in the same unit of measurement as our data.
Below is the general equation to calculate the standard deviation. You'll notice two equations here. The top calculates the S.D. for a population, the bottom calculates the S.D for a sample. Typically you will never be able to measure a "true" population, so pandas by default calculates S.D. for a sample.

$$\sigma = \sqrt{\frac{\sum (X - \mu)^2}{N}}$$

$$s = \sqrt{\frac{\sum (x_i - \bar{x})^2}{n - 1}}$$

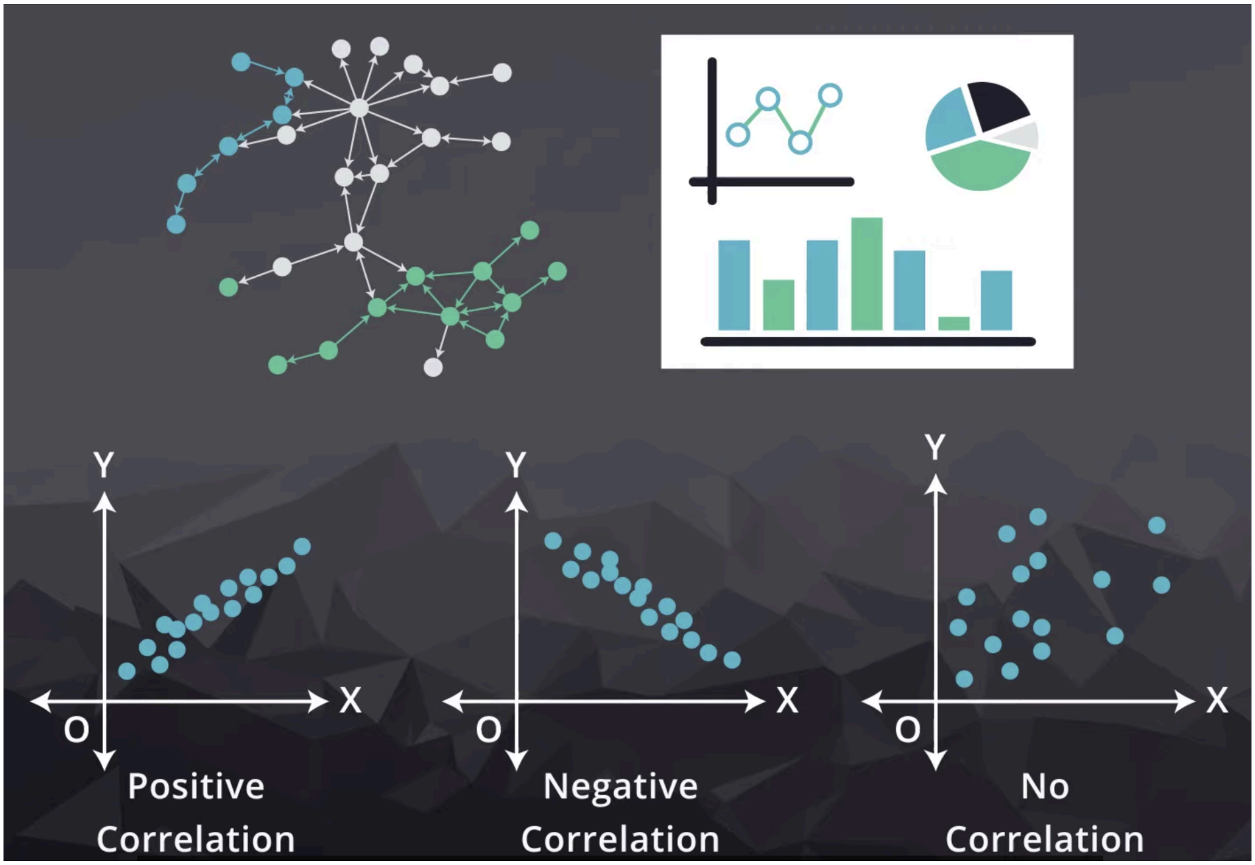
Example:

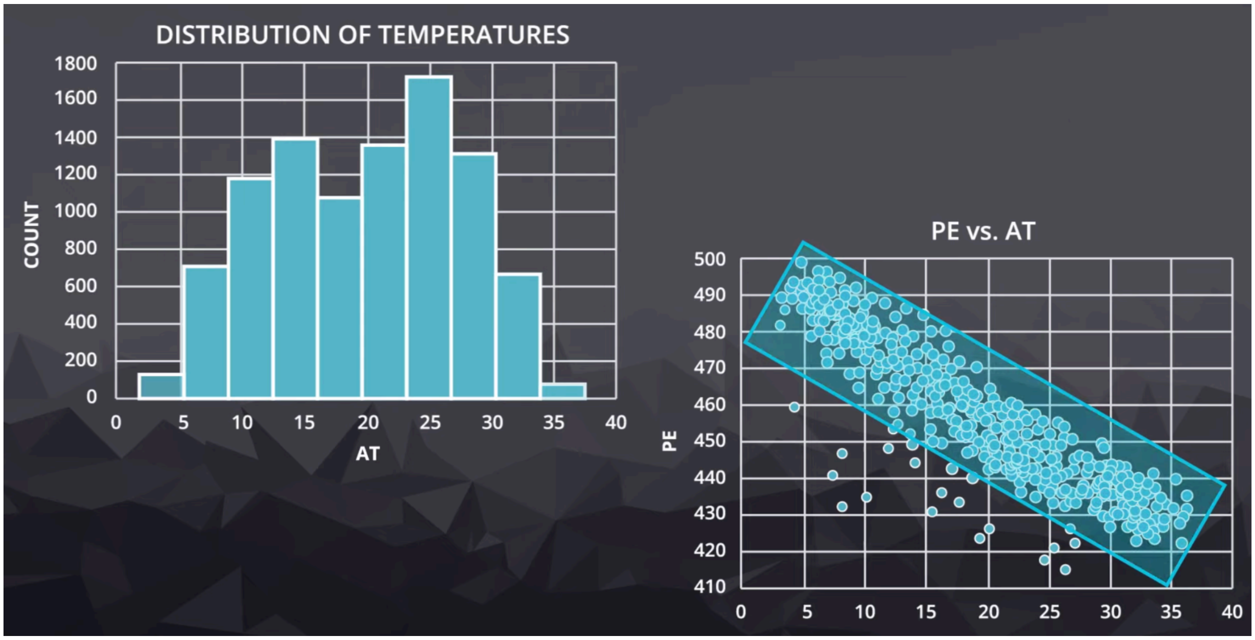
- [1, 2, 3, 4, 5]
- Mean = 3
- [1 - 3, 2 - 3, 3 - 3, 4 - 3, 5 - 3] (subtract mean from each value)
- [-2, -1, 0, 1, 2]
- [4, 1, 0, 1, 4] (square each value)
- 10 / 4 = 2.5 (sum values in list, divide by list length - 1, for sample)
- 2.5 ^ 0.5 = 1.581 (take the square root)

Variance - .var(): Variance is a measure that helps us understand how the numbers in a group differ from their average, giving a sense of how scattered or clustered the data is.
Variance is calculated by squaring standard deviation value.
Example:

- [1, 2, 3, 4, 5]
- Standard deviation = 1.581
- 1.581 ^ 2 = 2.5

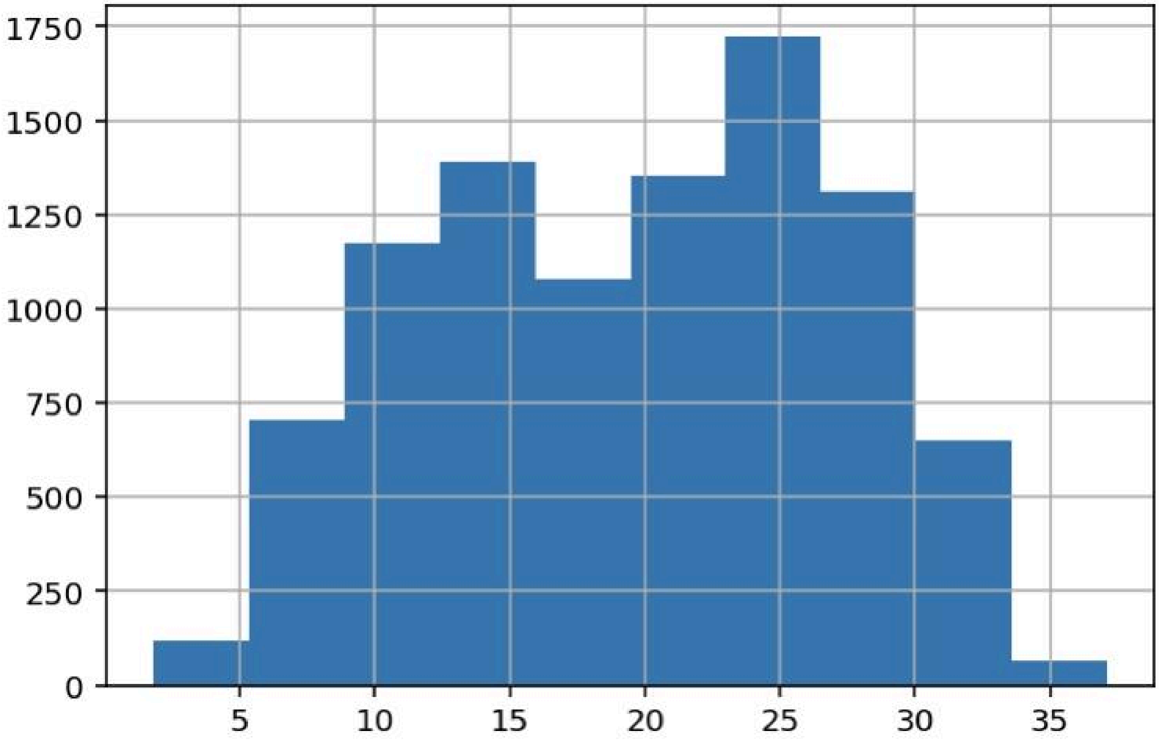
Exploring Data with Visuals





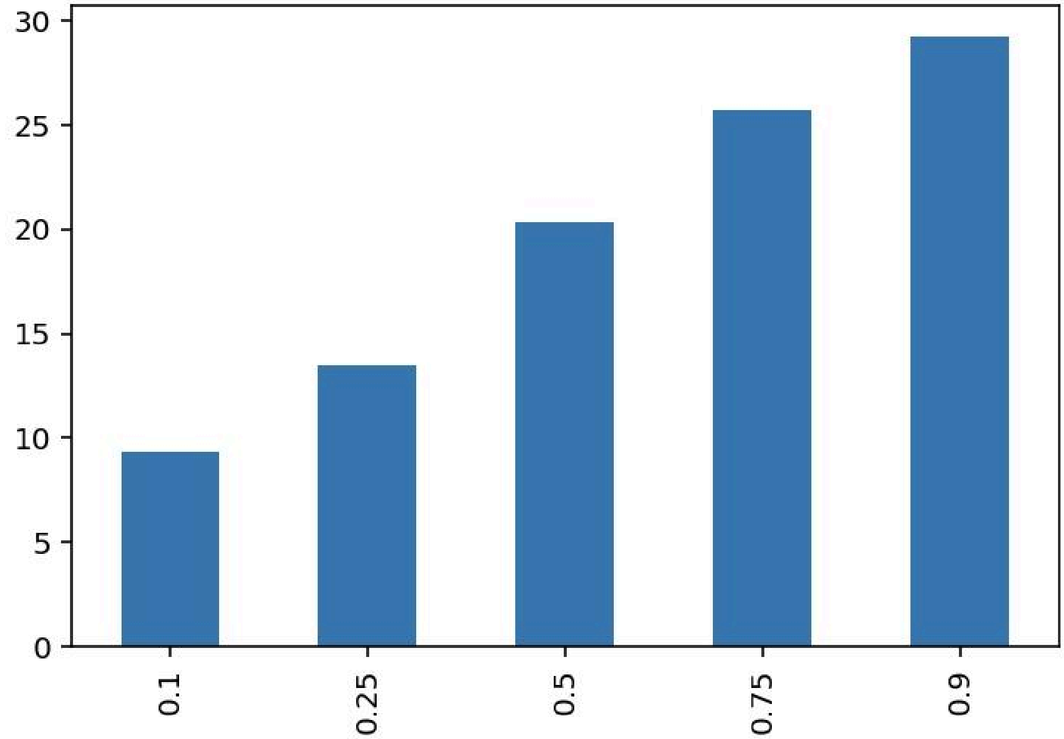
Histogram

- Displays the distribution of numerical data.
- Divides data into bins and shows frequency of observations in each bin.



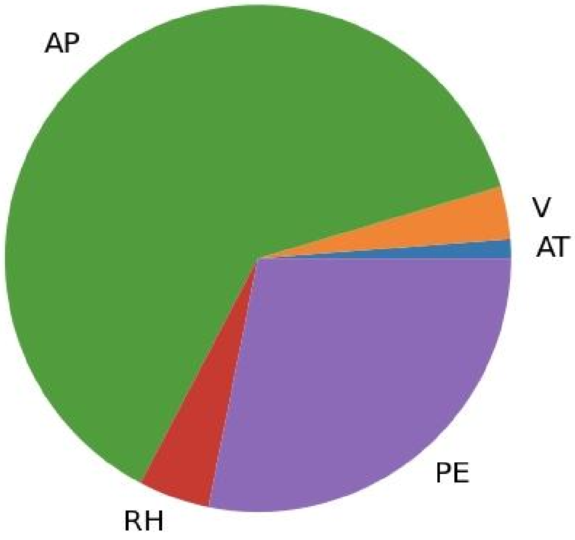
Bar Chart

- Compares different categories.
- Shows values as bars of different lengths.



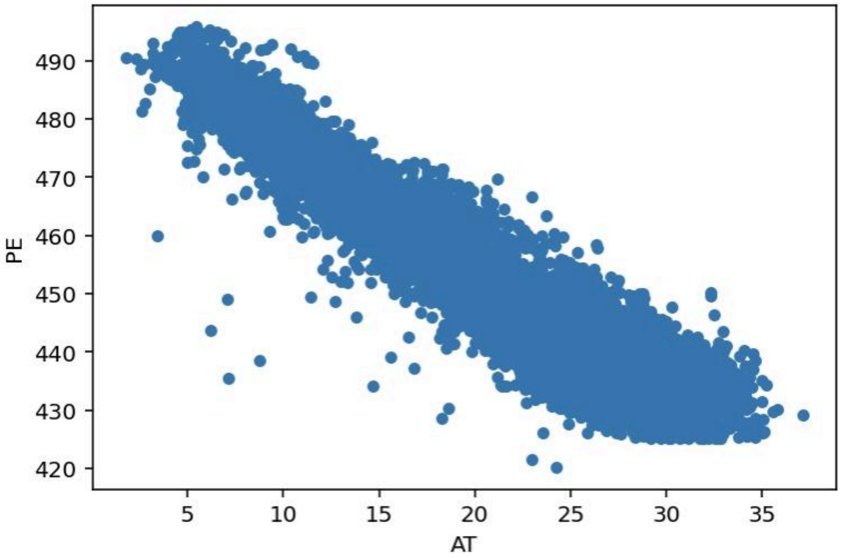
Pie Chart

- Shows the proportion of different categories.
- Displays as slices of a pie.



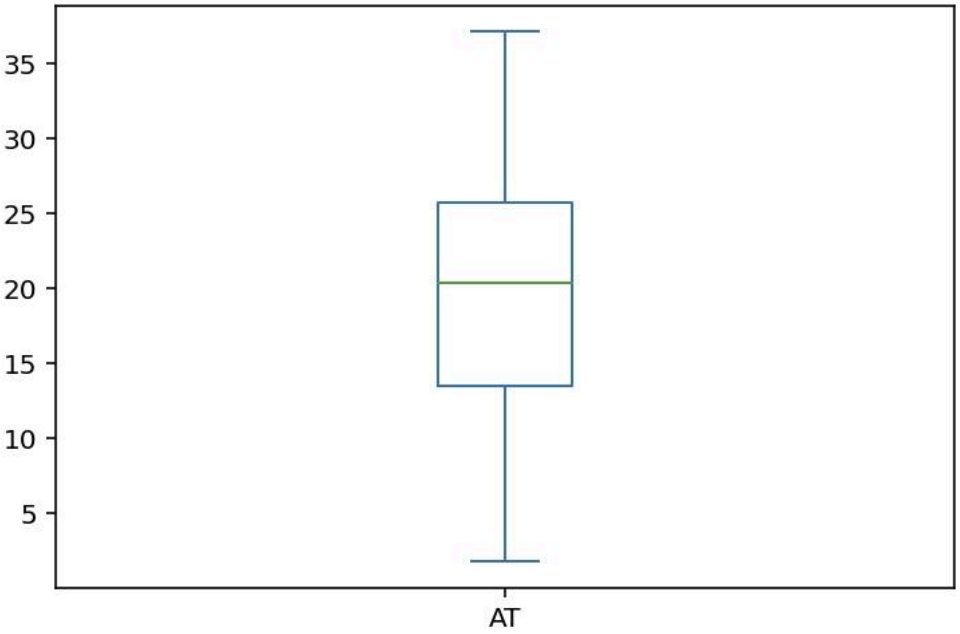
Scatter Plot

- Displays the relationship between two numerical variables.
- Plots points on a two-dimensional graph.
- Positive correlation of the two variables if the slope is increasing from left to right (positive slope).
- Negative correlation of the two variables if the slope is decreasing from left to right (negative slope).



Box Plot

- Displays the distribution of numerical data.
- Shows five key values: minimum, first quartile, median, third quartile, and maximum.



Communicating Results

Visuals can be used to efficiently communicate conclusions drawn from your analysis. Note the labeling, color, size, and data selection arguments for histograms, bar charts, and pie charts in the examples on the following pages. These can be used to customize your visualizations.

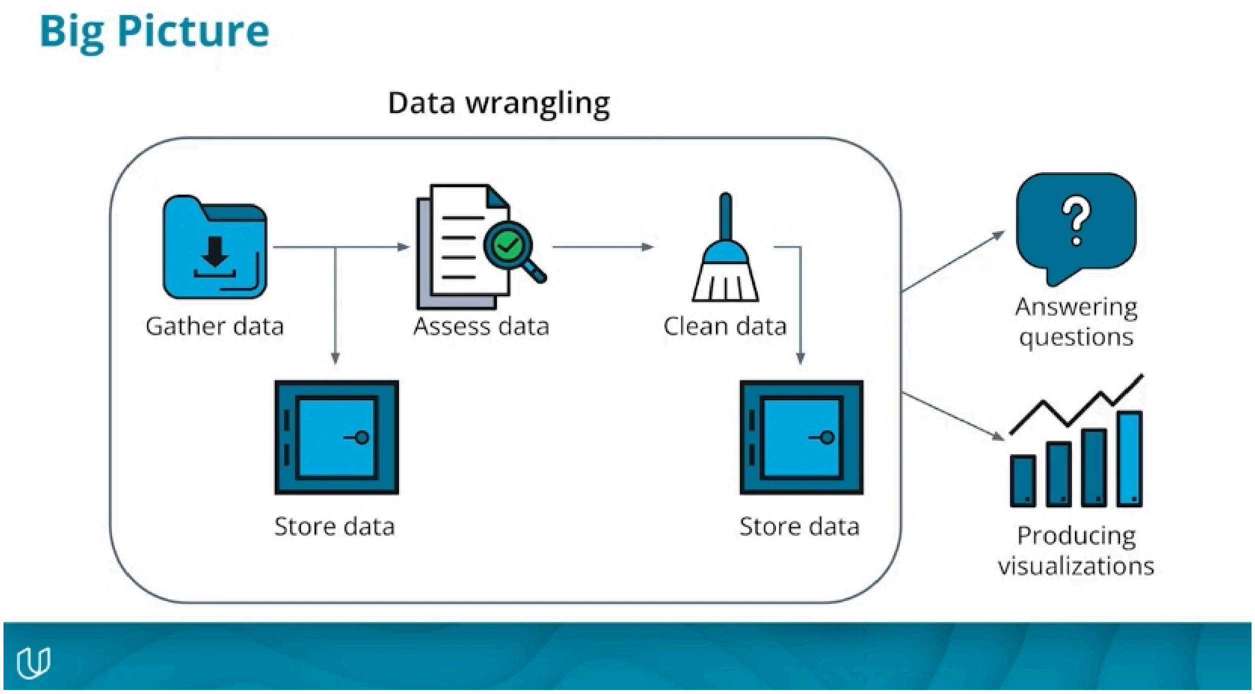
These are a few of the parameters you can change in pandas' .plot() method.

- Figsize: This parameter in the .plot() method of Pandas allows you to set the size of the figure or plot. The syntax for setting this parameter is df.plot(figsize=(width, height)), where width and height are the desired dimensions in inches. For example, df.plot(figsize=(10, 5)) will create a plot with a width of 10 inches and a height of 5 inches.
- Title: You can add titles to your Pandas plot by using the title parameter in the .plot() method. The syntax is df.plot(title='Title of Plot'). This will add a title to your plot with the specified text.
- xlabel, ylabel, and ylim: You can customize the labels and other properties of the x-axis and y-axis of your Pandas plot by using the xlabel, ylabel, and ylim parameters. The syntax for setting these parameters is as follows:

  df.plot(xlabel='X Axis Label') # sets the label for the x-axis
  df.plot(ylabel='Y Axis Label') # sets the label for the y-axis
  df.plot(ylim=(min_value, max_value)) # sets the minimum and maximum values for the y-axis
- Color: You can color your Pandas plot using the color parameter in the .plot() method. The syntax for setting this parameter is df.plot(color='color_name'). You can use any valid color name (e.g. 'blue', 'red', 'green') or a hexadecimal color code (e.g. '#0026ff') to set the color.
- Legend: The legend in a Pandas plot provides a key to identify the different lines or plots in the figure. You can add a legend to your plot by using the legend parameter in the .plot() method. The syntax for setting this parameter is df.plot(legend=True). If you set legend to True, the default legend will be added to your plot.

Course 2: Advanced Data Wrangling

Introduction to Data Wrangling



Big Picture

Data wrangling

Gather data

Assess data

Clean data

Store data

Store data

Answering questions

Producing visualizations

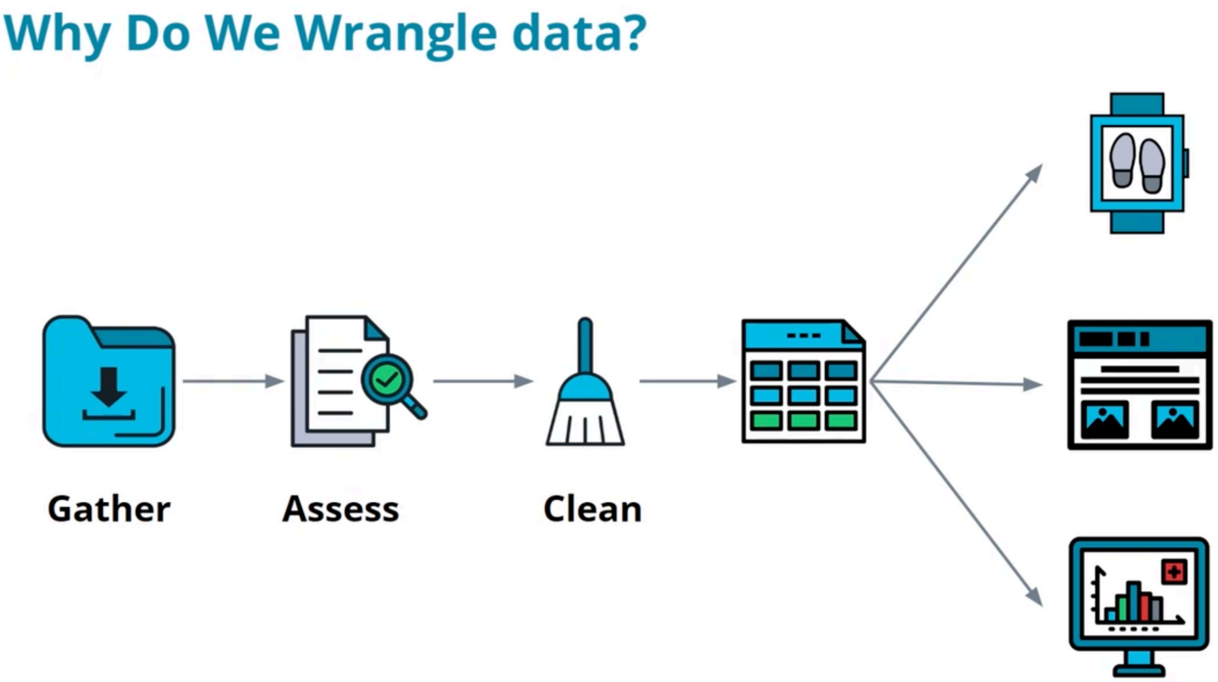
Compare data wrangling with other data processes
While with basic data inspection, you may have data just handed to you, data wrangling involves actively gathering data and strategizing around implementing your wrangling workflows.

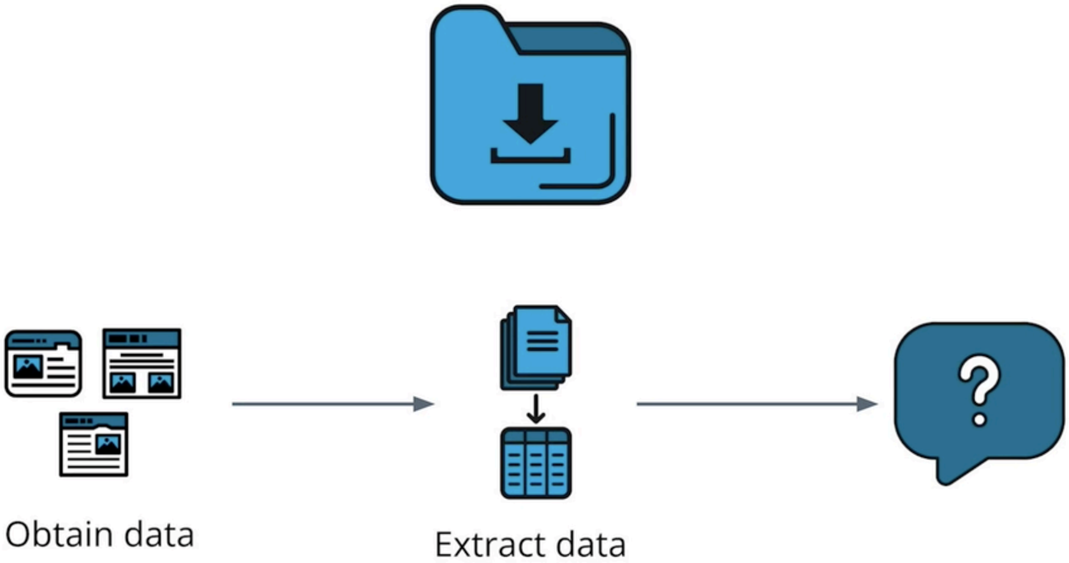
Data wrangling is a more formal-structured process compared to the exploratory efforts used in Exploratory Data Analysis (EDA).

Wrangling data is also often done by analysts and can be used in conjunction with Extract, Transform, and Load (ETL) done by IT teams to ensure a company has a concrete strategy to handle the data.

Note that data wrangling involves several different stakeholders, typically including data analysts, business leaders, data providers, IT teams, and more, who influence how data is generated and moves throughout an organization or even an entire industry.

Gathering Data





Data is everywhere - think wearable devices, hospital equipment, and interactive kiosks in a shopping mall!
The first step in the data wrangling process is grabbing this data and converting it into a format you can work with as a data analyst! In this sense, data gathering contains two sub-steps:

- Obtaining the data from different sources, like websites or Applications Programming Interfaces (APIs).
- Extracting the obtained data into a format suitable for assessment and cleaning.

With this data, we could then use the data to answer questions after assessing and cleaning it.

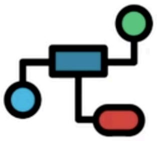
Files on Hand

- Check file safety
- Check file extension and open via right app

.Zip to Store Multiple Files

.zip (Windows, Mac, Unix)

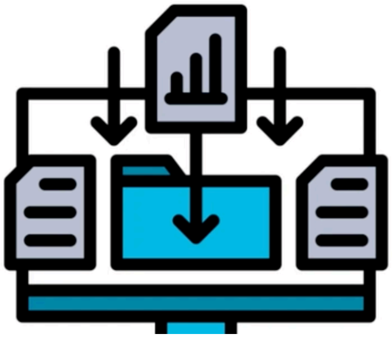
.tar.gz (Unix)



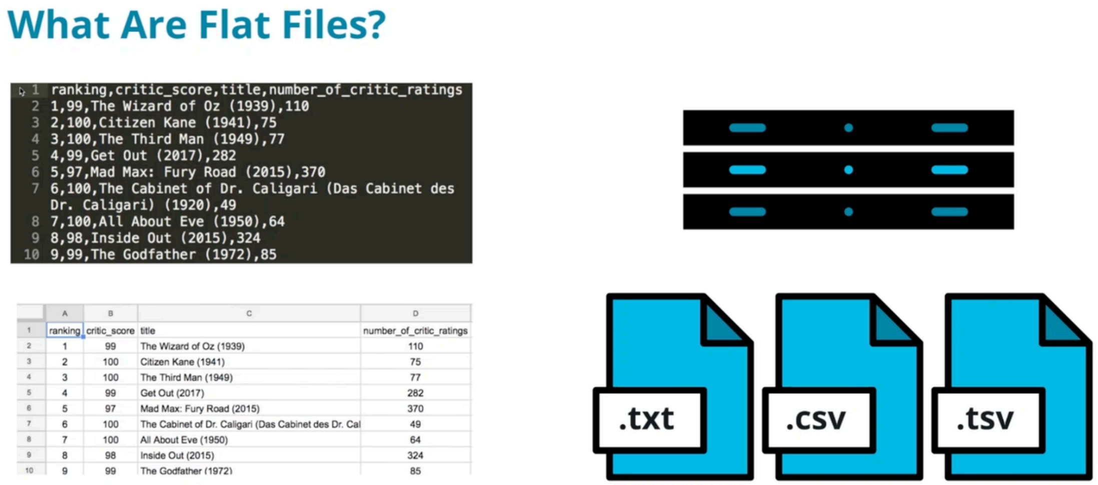
- Programmatically unzipping files helps with reproducibility

Reproducible Workflows

- Name your folder descriptively and consistently
- Name these subfiles constructively
- Create a clear structure of data files within your .zip folder



Flat File Structure



What Are Flat Files?

```
ranking,critic_score,title,number_of_critic_ratings
1,99,The Wizard of Oz (1939),110
2,100,Citizen Kane (1941),75
3,100,The Third Man (1949),77
4,99,Get Out (2017),282
5,97,Mad Max: Fury Road (2015),370
6,100,The Cabinet of Dr. Caligari (Das Cabinet des
Dr. Caligari) (1920),49
7,100,All About Eve (1950),64
8,98,Inside Out (2015),324
9,99,The Godfather (1972),85
```

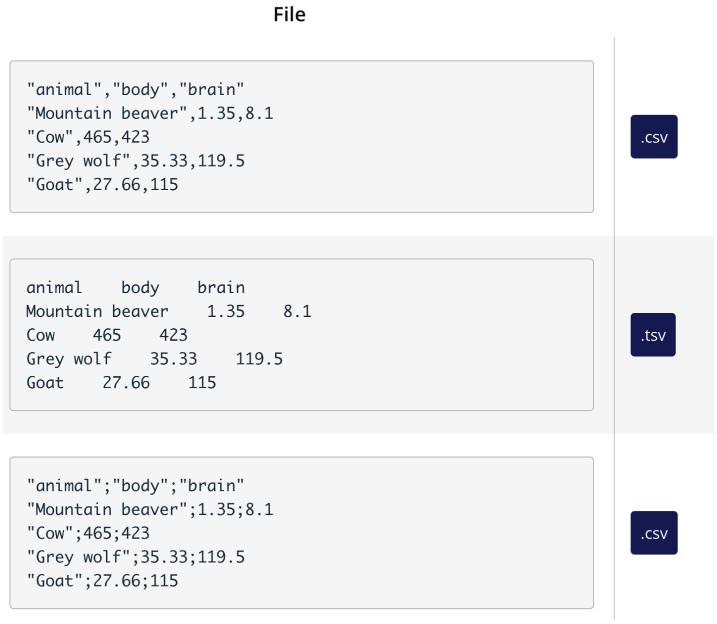
	A	B	C	D
1	ranking	critic_score	title	number_of_critic_ratings
2	1	99	The Wizard of Oz (1939)	110
3	2	100	Citizen Kane (1941)	75
4	3	100	The Third Man (1949)	77
5	4	99	Get Out (2017)	282
6	5	97	Mad Max: Fury Road (2015)	370
7	6	100	The Cabinet of Dr. Caligari (Das Cabinet des Dr. Cal	49
8	7	100	All About Eve (1950)	64
9	8	98	Inside Out (2015)	324
10	9	99	The Godfather (1972)	85

Flat files are simple data files that have a uniform format, such as some text files (.txt), comma-separated values (.csv), and tab-separated values (.tsv).

Let's recap the key properties of flat files:

- They are tabular data in plain text
- Each line is a single data record
- Each record has one or more values or fields
- These values/fields are separated by delimiters or some sort of “character”, like tab spaces, commas, or semicolons.

Tab-separated values use tabs as delimiters, whereas comma-separated value files use commas or sometimes semicolons as the delimiter. Note that not all .txt files are flat files. And Microsoft Excel files (.xlsx) files are not flat files, because .xls often contains more advanced features like formatting, formulas, and multiple sheets within a single file. Flat files simply store tabular data in plain text without additional features or structures.



File

```
"animal","body","brain"
"Mountain beaver",1.35,8.1
"Cow",465,423
"Grey wolf",35.33,119.5
"Goat",27.66,115
```

.csv

```
animal    body    brain
Mountain beaver    1.35    8.1
Cow    465    423
Grey wolf    35.33    119.5
Goat    27.66    115
```

.tsv

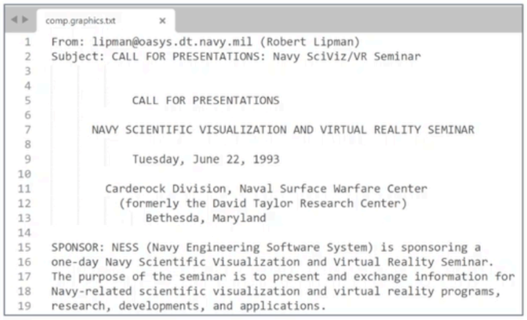
```
"animal";"body";"brain"
"Mountain beaver";1.35;8.1
"Cow";465;423
"Grey wolf";35.33;119.5
"Goat";27.66;115
```

.csv

Text File Structure

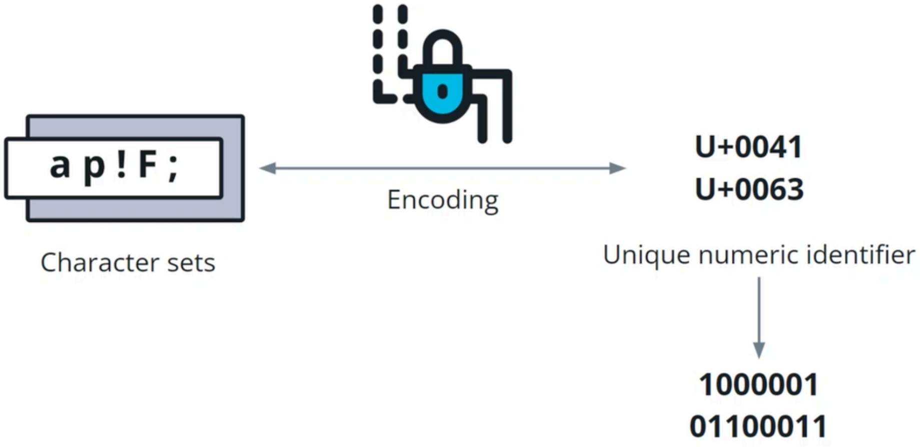
What is a text file?

- Text files (.txt) are files with text that:
  - Use a specific character set
  - No rules for structure and no formatting (e.g., italics or bolding)
  - No media (e.g., image and video)
  - Lines of text that are separated by newline characters or `\` in Python
  - Can be viewed and edited in simple text editors



```
From: lipman@oasys.dt.navy.mil (Robert Lipman)
Subject: CALL FOR PRESENTATIONS: Navy SciViz/VR Seminar


            CALL FOR PRESENTATIONS

      NAVY SCIENTIFIC VISUALIZATION AND VIRTUAL REALITY SEMINAR

            Tuesday, June 22, 1993

        Carderock Division, Naval Surface Warfare Center
          (formerly the David Taylor Research Center)
              Bethesda, Maryland

SPONSOR: NESS (Navy Engineering Software System) is sponsoring a
one-day Navy Scientific Visualization and Virtual Reality Seminar.
The purpose of the seminar is to present and exchange information for
Navy-related scientific visualization and virtual reality programs,
research, developments, and applications.
```

Encoding and Character Sets



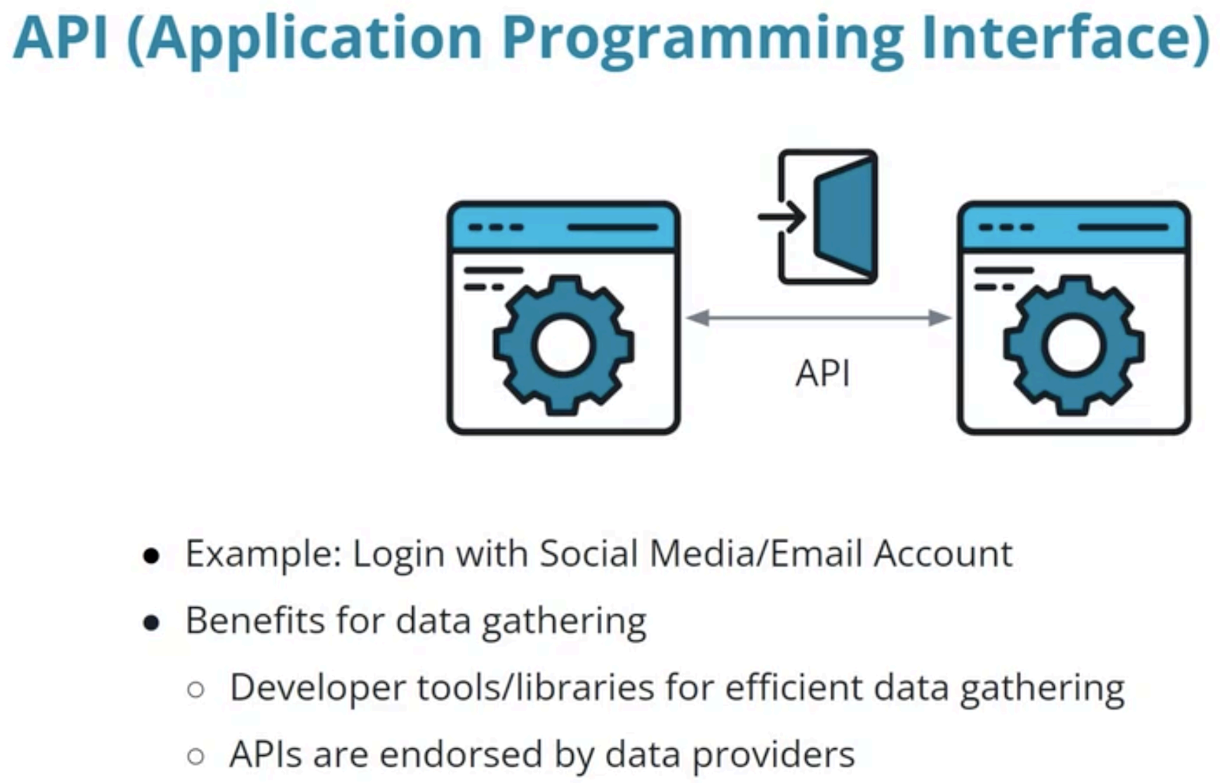
Text vs. Byte Types in Python

- .encode(): turn a *character string* into a *byte string*
- decode(): turn a *byte string* into a *character string*
- Python string type: *str* - holds Unicode data.
- Python bytes types: *bytes* and *bytearray*

`Bytes` is immutable - once created, the object cannot be modified

`Bytesarray` is mutable - it can be modified after creation

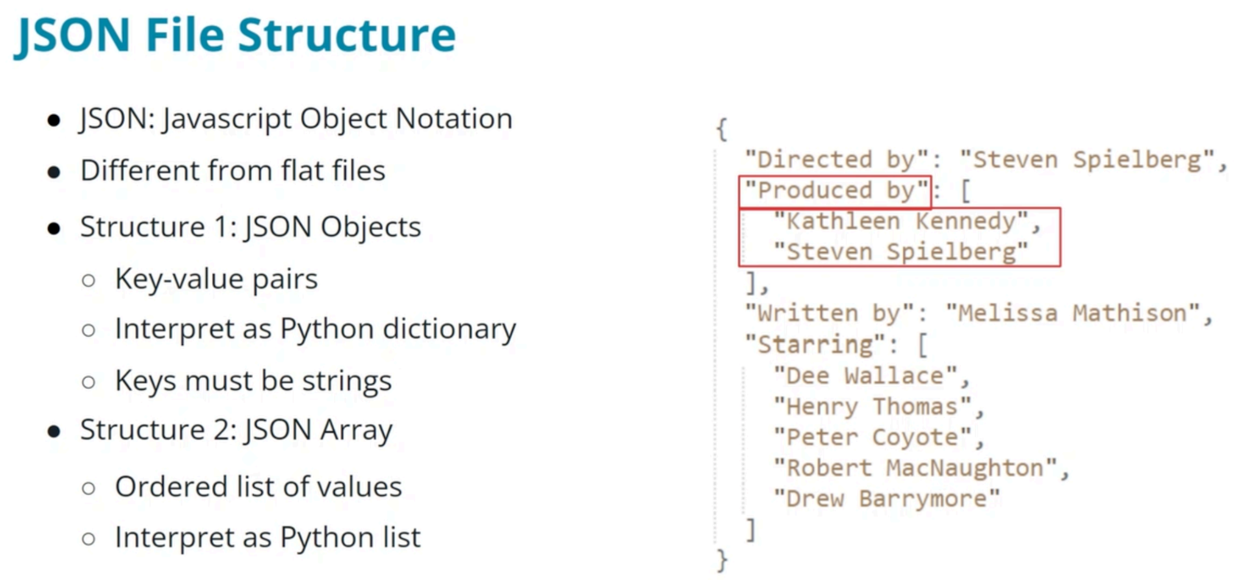
Gathering Data with APIs



API (Application Programming Interface)

- Example: Login with Social Media/Email Account
- Benefits for data gathering
  - Developer tools/libraries for efficient data gathering
  - APIs are endorsed by data providers

JSON File Structure



JSON File Structure

- JSON: Javascript Object Notation
- Different from flat files
- Structure 1: JSON Objects
  - Key-value pairs
  - Interpret as Python dictionary
  - Keys must be strings
- Structure 2: JSON Array
  - Ordered list of values
  - Interpret as Python list

```
{
  "Directed by": "Steven Spielberg",
  "Produced by": [
    "Kathleen Kennedy",
    "Steven Spielberg"
  ],
  "Written by": "Melissa Mathison",
  "Starring": [
    "Dee Wallace",
    "Henry Thomas",
    "Peter Coyote",
    "Robert MacNaughton",
    "Drew Barrymore"
  ]
}
```

JavaScript Object Notation (JSON) is a popular format for representing data! JSON is built on two key structures:

- JSON Objects: A collection of key-value pairs. In Python, JSON objects are interpreted as dictionaries, and you can access them like you would a standard Python Dictionary. Note that JSON object keys must be strings.
- JSON Arrays: A JSON array is an ordered list of values. In Python, JSON arrays are interpreted and accessed like Python Lists. The values for JSON objects and arrays can be any valid JSON data type: string, number, object, array, Boolean or null.

When objects and arrays are combined, it is called nesting. Through nesting, JSON becomes great for representing and accessing complicated data hierarchies. For example, in the JSON file below, the collection attribute is neatly nestled within the metadata attribute, indicating the book collection data comes under metadata

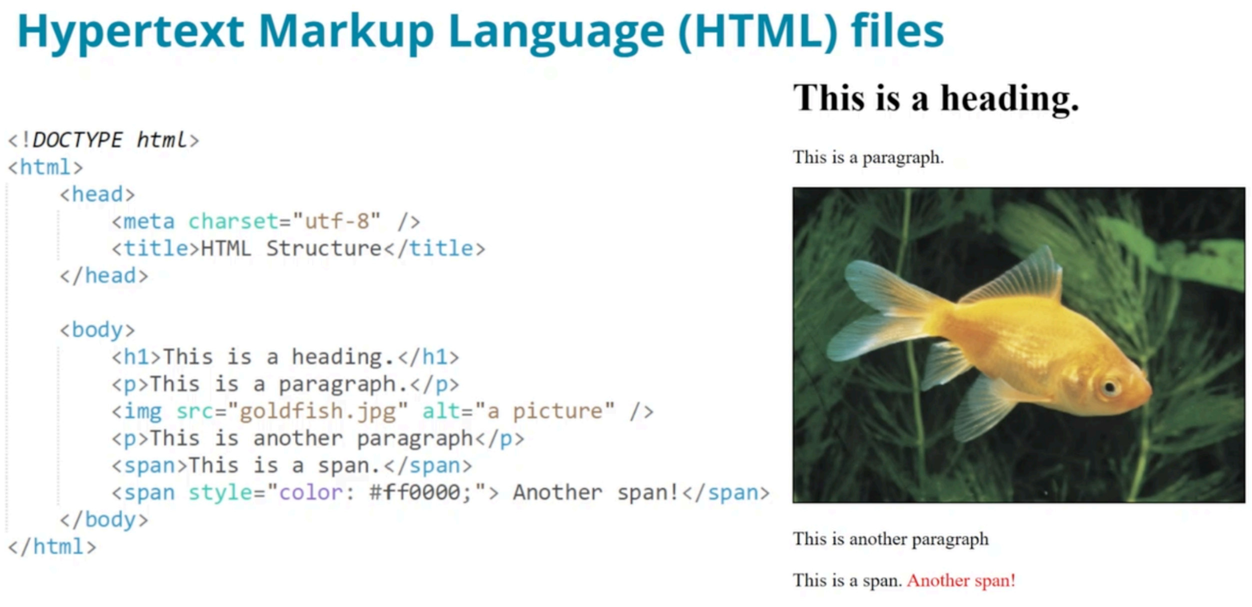
Web Scraping

Data is oftentimes not easily accessible from websites. To get this data, data wranglers sometimes need to use web scraping, which allows us to extract data from websites using code.

Website data is written in HTML (HyperText Markup Language), which uses tags to structure the page. Because HTML and its tags are just text, the text can be accessed using parsers.

We can gather HTML data by downloading HTML files manually or accessing them programmatically.

HTML File Structure



Hypertext Markup Language (HTML) files

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8" />
        <title>HTML Structure</title>
    </head>

    <body>
        <h1>This is a heading.</h1>
        <p>This is a paragraph.</p>
        <img src="goldfish.jpg" alt="a picture" />
        <p>This is another paragraph</p>
        <span>This is a span.</span>
        <span style="color: #ff0000;"> Another span!</span>
    </body>
</html>
```

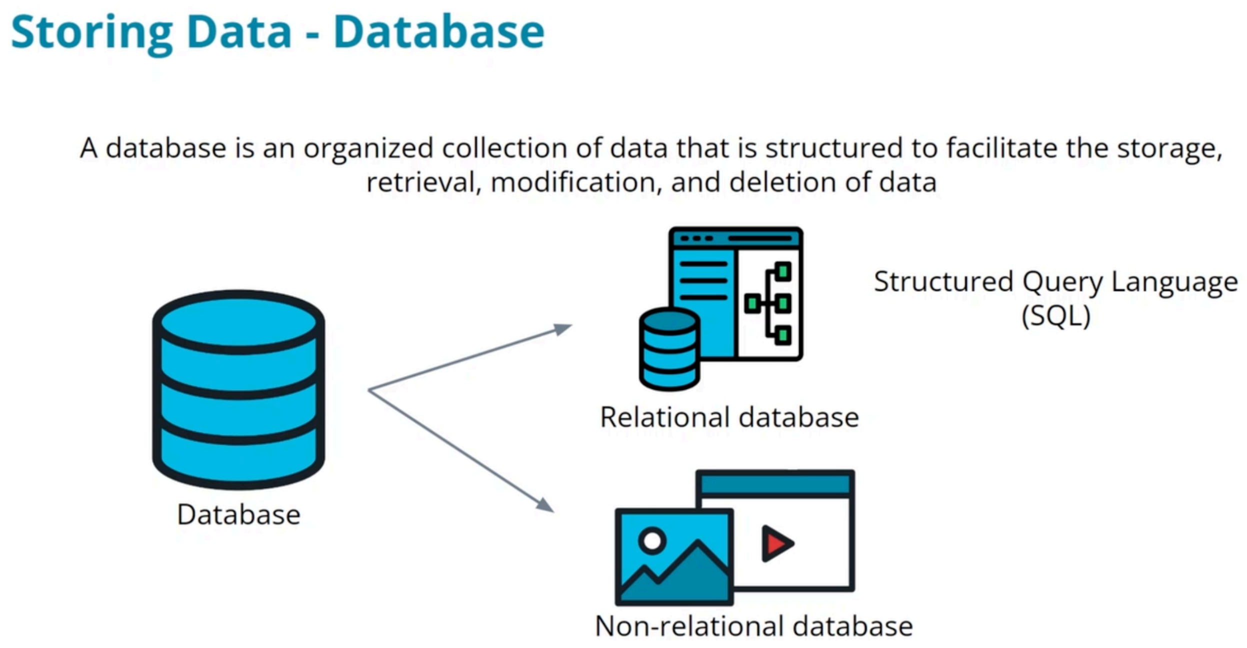
This is a heading.

This is a paragraph.

This is another paragraph

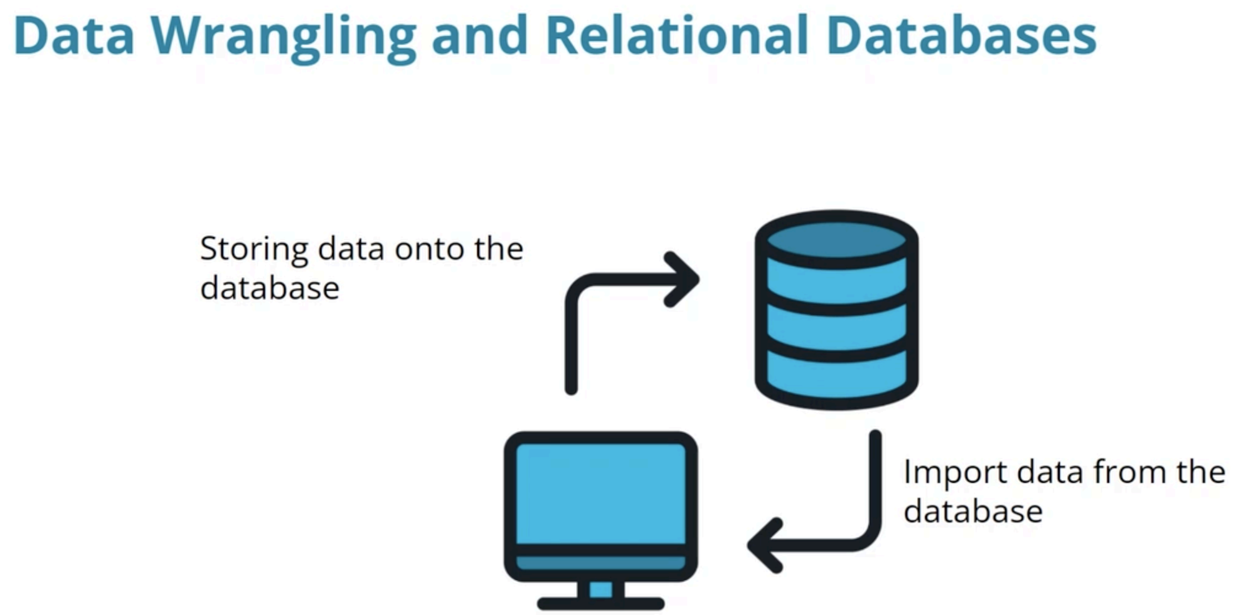
This is a span. Another span!

Relational Databases for Data Wrangling



Storing Data - Database

A database is an organized collection of data that is structured to facilitate the storage, retrieval, modification, and deletion of data



Data Wrangling and Relational Databases

Databases primer
A database is an organized collection of structured data to facilitate data storage, retrieval, modification, and deletion.

Databases have many advantages for data wrangling, including :

- They're fast and can be optimized for speed
- They have administrative features like access controls
- They can check for data integrity to ensure data is entered in the appropriate format, which is critical for data wrangling.

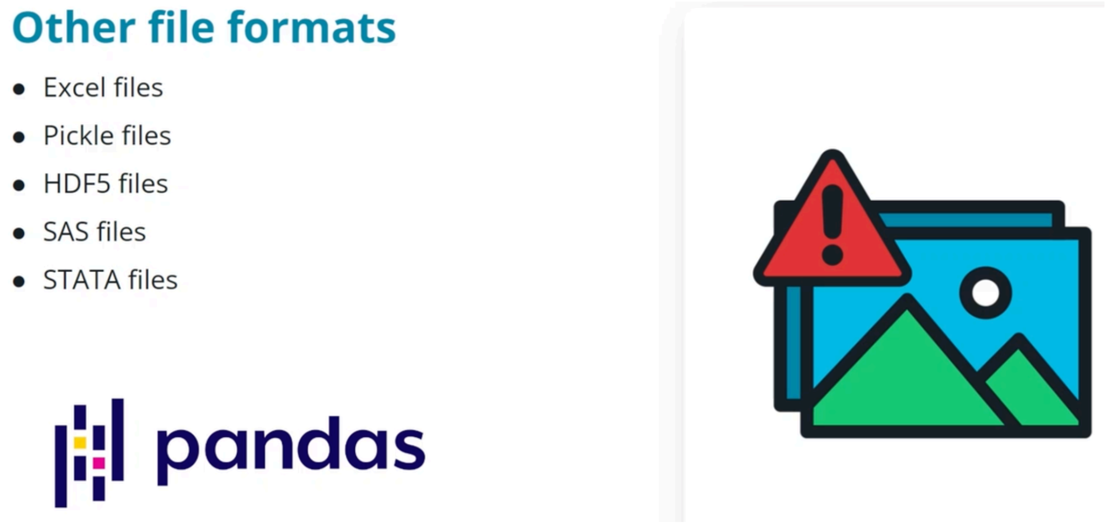
There are two main types of databases: relational and non-relational, with relational being the most popular.

Relational databases
Data in a relational database is organized by column, very similar to tabular data like an Excel spreadsheet but with rigid rules:

- Each column has a unique name
- All the data in a column must be the same data type
- Descriptive column names are important

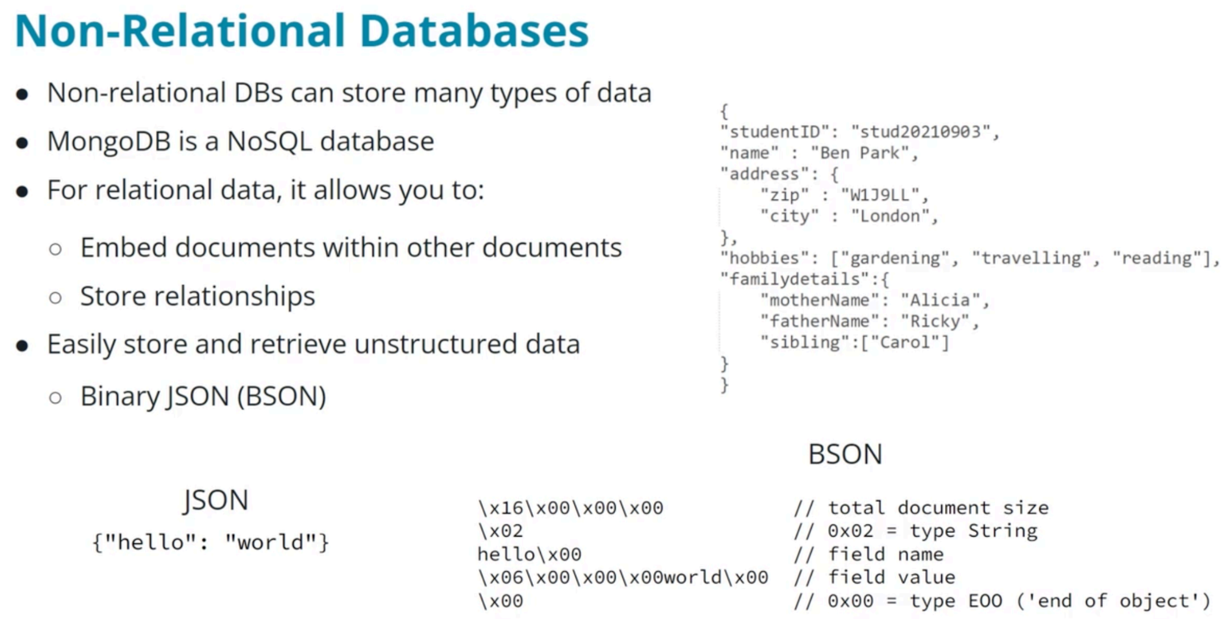
SQL, or Structured Query Language, is a standard and popular language for communicating with relational databases.

Other File Formats



Other file formats

- Excel files
- Pickle files
- HDF5 files
- SAS files
- STATA files



Non-Relational Databases

- Non-relational DBs can store many types of data
- MongoDB is a NoSQL database
- For relational data, it allows you to:
    - Embed documents within other documents
    - Store relationships
- Easily store and retrieve unstructured data
    - Binary JSON (BSON)

```
{
"studentID": "stud20210903",
"name" : "Ben Park",
"address": {
    "zip" : "W1J9LL",
    "city" : "London",
},
"hobbies": ["gardening", "travelling", "reading"],
"familydetails":{
    "motherName": "Alicia",
    "fatherName": "Ricky",
    "sibling":["Carol"]
}
}
```

JSON

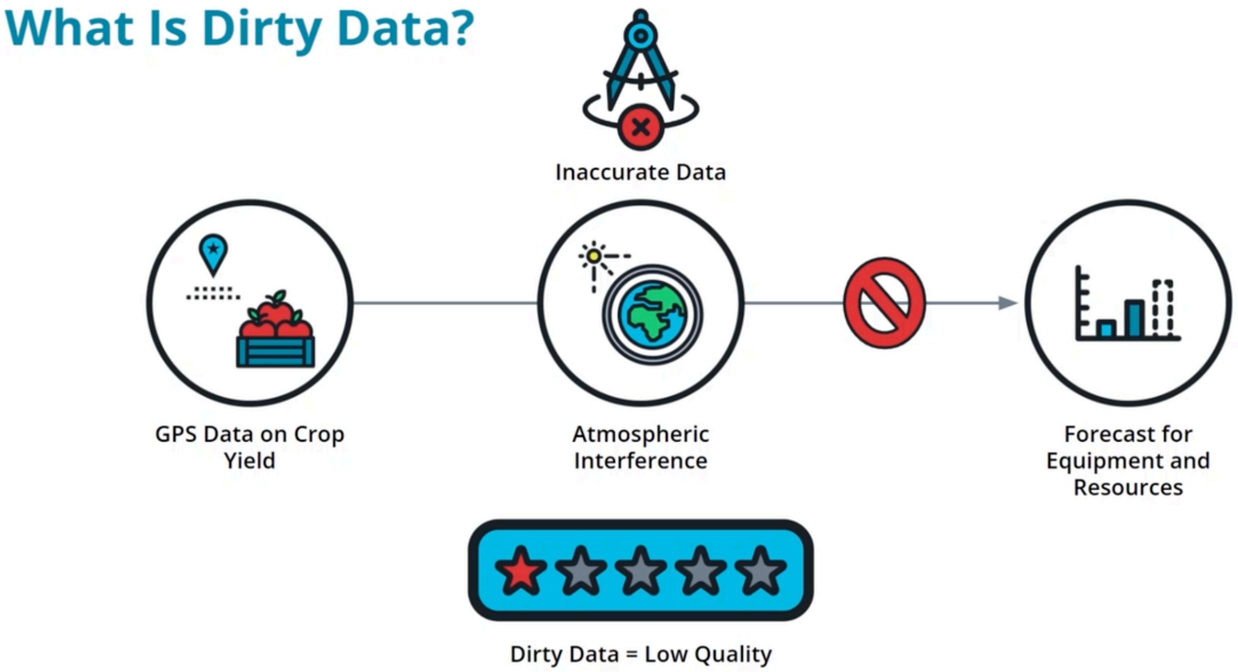
```
{"hello": "world"}
```

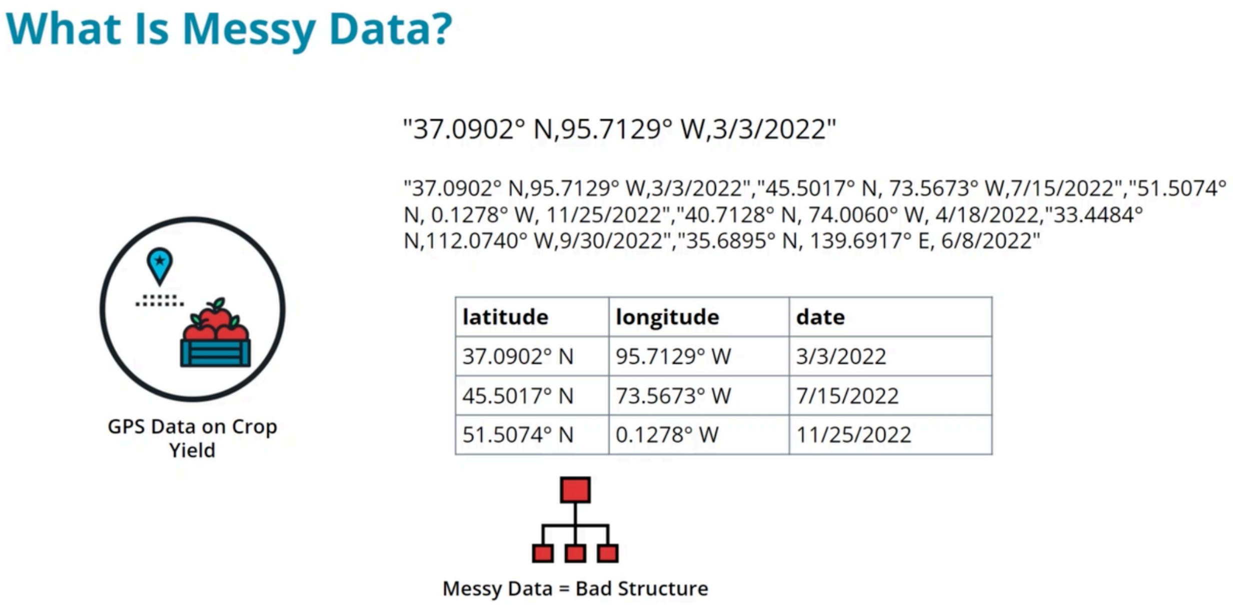
BSON

```
\x16\x00\x00\x00           // total document size
\x02                       // 0x02 = type String
hello\x00                  // field name
\x06\x00\x00\x00world\x00  // field value
\x00                       // 0x00 = type EOO ('end of object')
```

Assessing Data

Unclean Data: Dirty vs. Messy



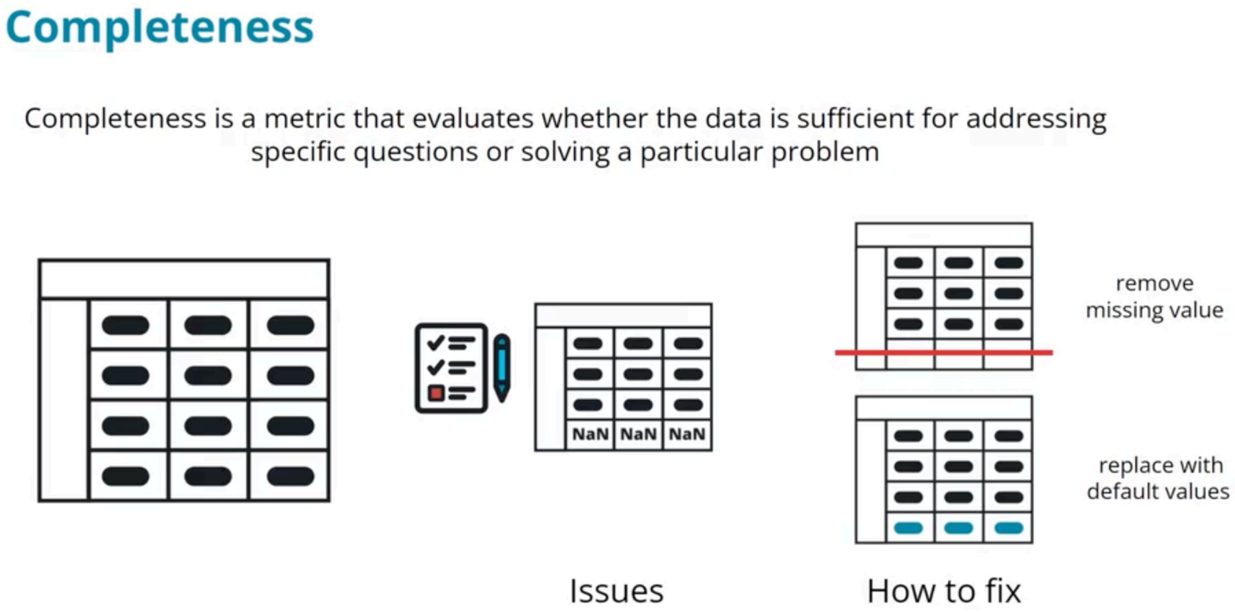


"37.0902° N,95.7129° W,3/3/2022"

"37.0902° N,95.7129° W,3/3/2022","45.5017° N, 73.5673° W,7/15/2022","51.5074° N, 0.1278° W, 11/25/2022","40.7128° N, 74.0060° W, 4/18/2022,"33.4484° N,112.0740° W,9/30/2022","35.6895° N, 139.6917° E, 6/8/2022"

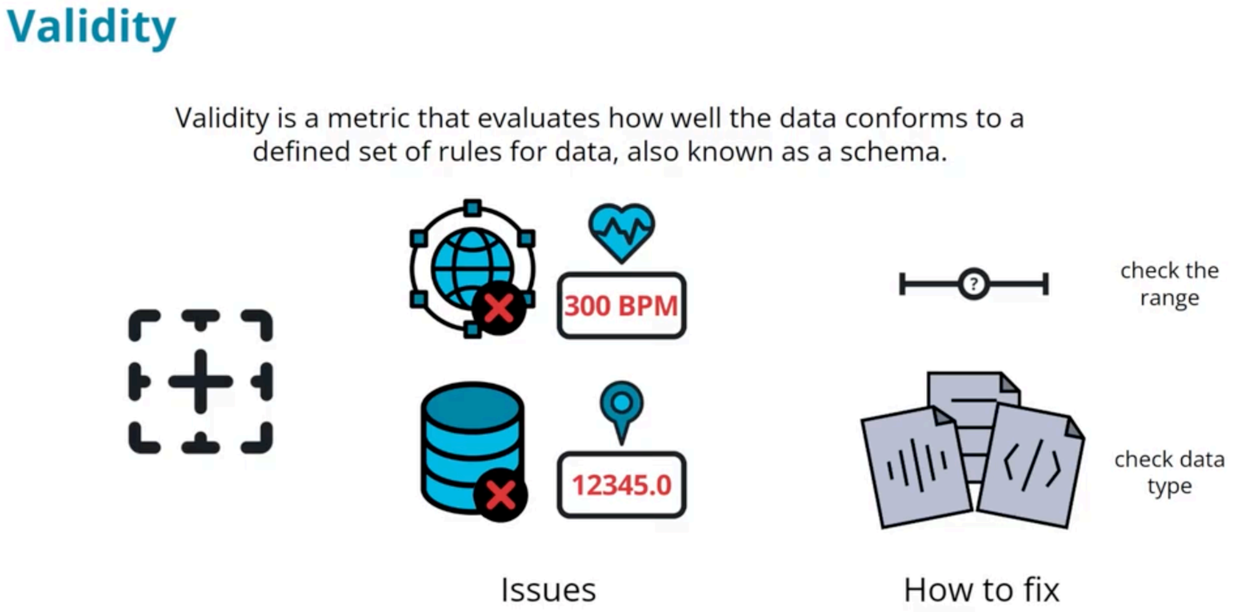
latitude	longitude	date
37.0902° N	95.7129° W	3/3/2022
45.5017° N	73.5673° W	7/15/2022
51.5074° N	0.1278° W	11/25/2022

Dimensions of Data Quality



Completeness

Completeness is a metric that evaluates whether the data is sufficient for addressing specific questions or solving a particular problem

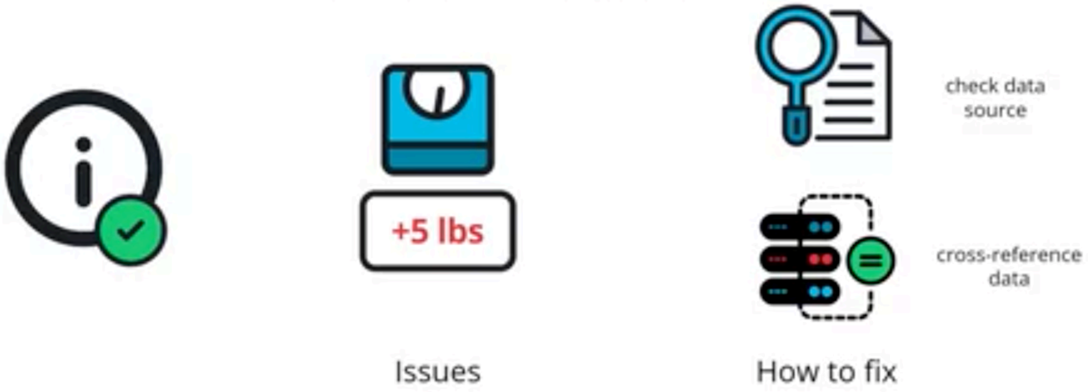


Validity

Validity is a metric that evaluates how well the data conforms to a defined set of rules for data, also known as a schema.

Accuracy

Accuracy is a metric that evaluates whether the data accurately represents the reality it aims to depict.



+5 lbs

check data source

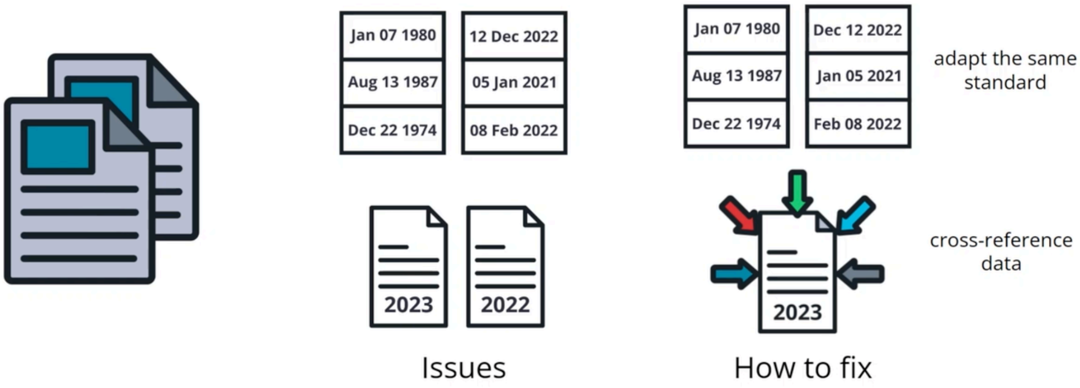
cross-reference data

Issues

How to fix

Consistency

Consistency is a metric that evaluates whether the data is following a standard format, and whether its information matches with information from other data sources.



Jan 07 1980	12 Dec 2022
Aug 13 1987	05 Jan 2021
Dec 22 1974	08 Feb 2022

2023 2022

Jan 07 1980	Dec 12 2022
Aug 13 1987	Jan 05 2021
Dec 22 1974	Feb 08 2022

adapt the same standard

2023

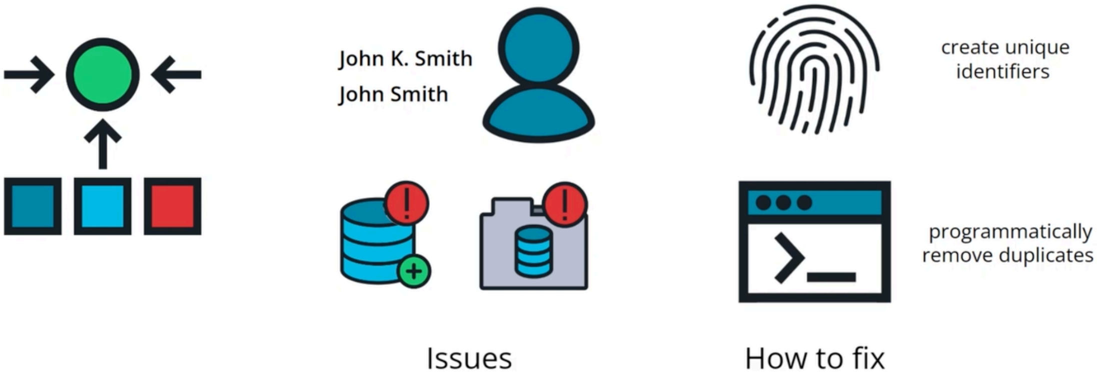
cross-reference data

Issues

How to fix

Uniqueness

Uniqueness is a metric that evaluates whether there is duplicate or overlapping values in the data.



John K. Smith

John Smith

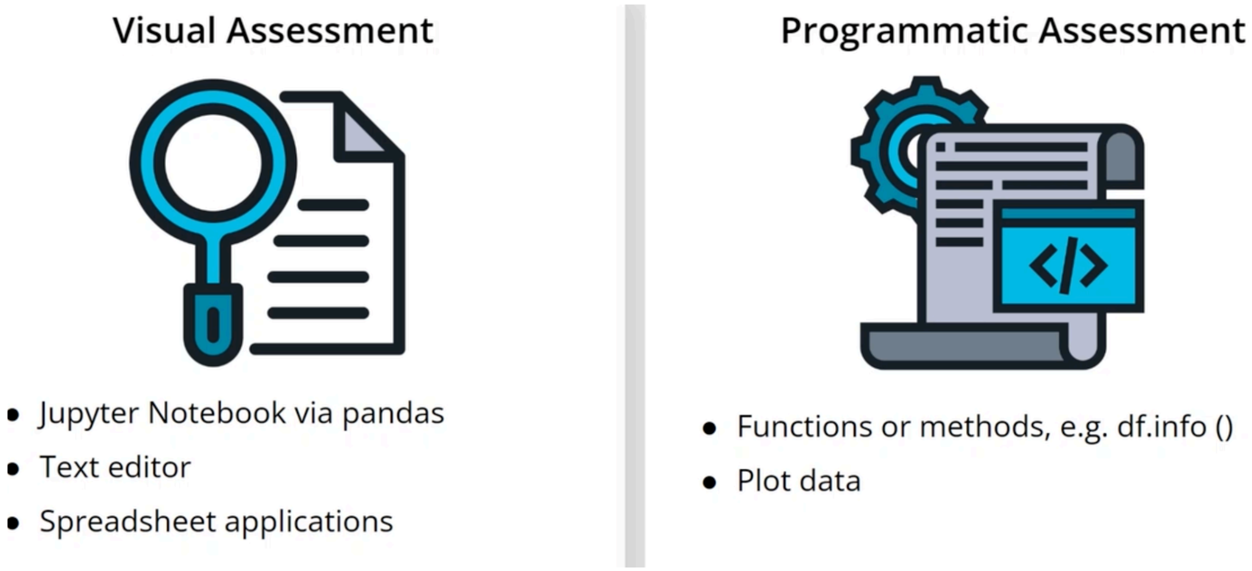
create unique identifiers

programmatically remove duplicates

Issues

How to fix

Assess Data Quality Visually



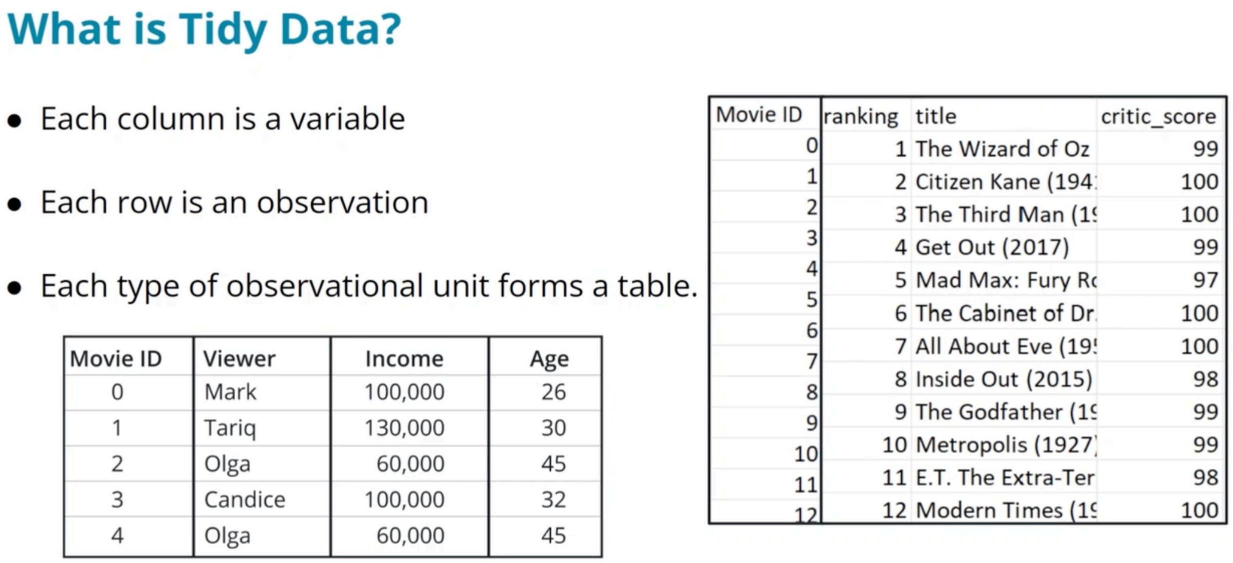
Visual Assessment

- Jupyter Notebook via pandas
- Text editor
- Spreadsheet applications

Programmatic Assessment

- Functions or methods, e.g. df.info ()
- Plot data

Tidy Data



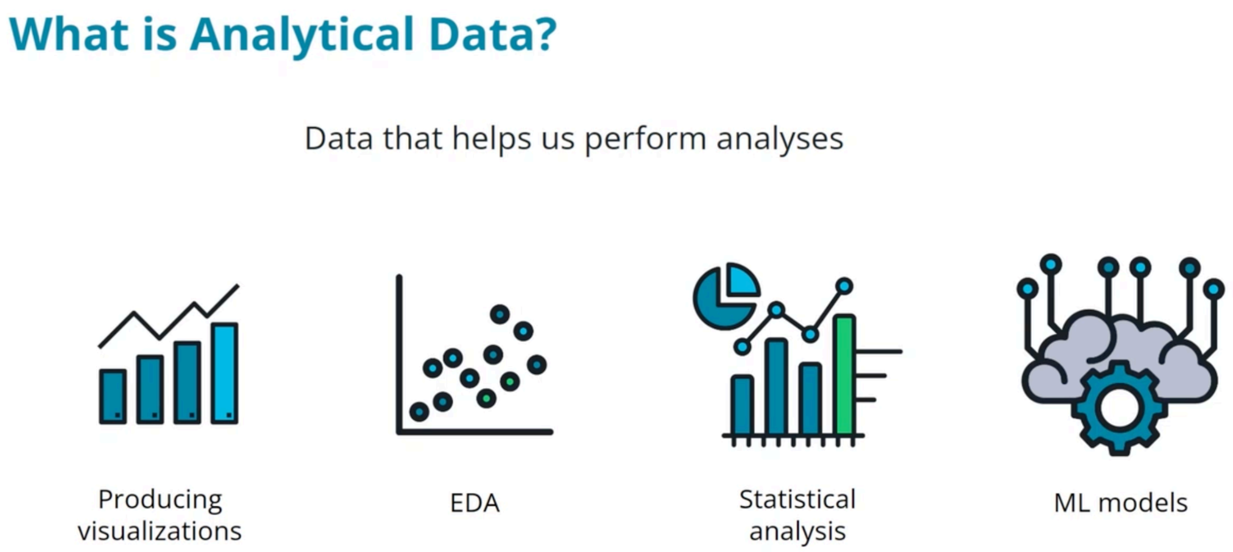
What is Tidy Data?

- Each column is a variable
- Each row is an observation
- Each type of observational unit forms a table.

Movie ID	Viewer	Income	Age
0	Mark	100,000	26
1	Tariq	130,000	30
2	Olga	60,000	45
3	Candice	100,000	32
4	Olga	60,000	45

Movie ID	ranking	title	critic_score
0	1	The Wizard of Oz	99
1	2	Citizen Kane (194	100
2	3	The Third Man (1	100
3	4	Get Out (2017)	99
4	5	Mad Max: Fury R	97
5	6	The Cabinet of Dr.	100
6	7	All About Eve (19	100
7	8	Inside Out (2015)	98
8	9	The Godfather (1	99
9	10	Metropolis (1927)	99
10	11	E.T. The Extra-Ter	98
11	12	Modern Times (1	100
12			

Designing Analytical Datasets



What is Analytical Data?

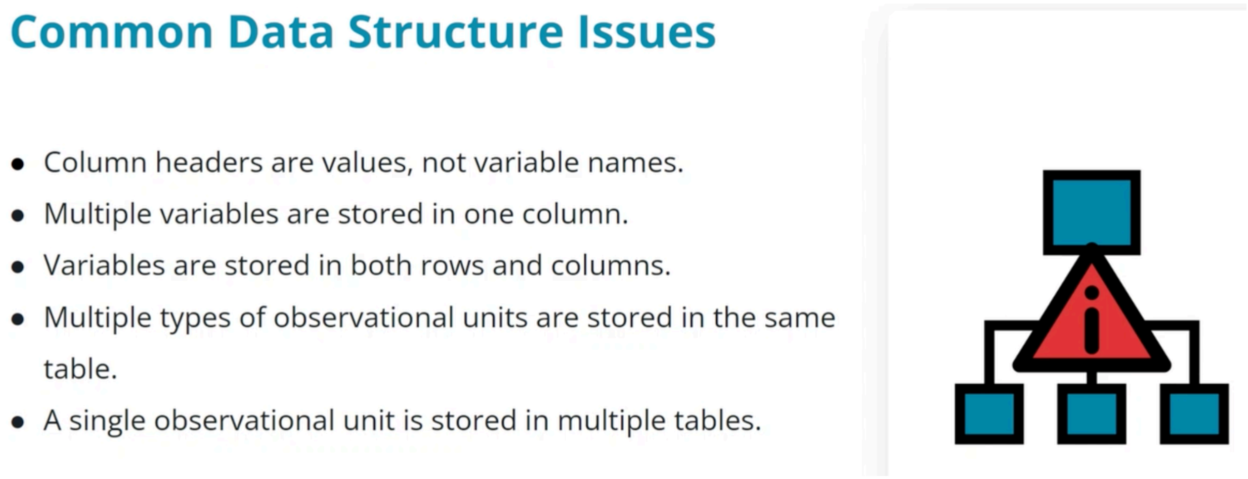
Data that helps us perform analyses

Unit of Analysis: Merging data from additional sources by defining a clear analysis unit can help enrich your dataset's value. Units of analysis can be individuals, groups, artifacts (i.e., items like books), and geographical units (e.g., states).

Note that while a unit of analysis is the item you want to structure your data around, a unit of observation is the item you can actually observe.

Unique keys: Unique keys can help with merging multiple datasets/data tables. Note that both primary keys and unique keys can be used to uniquely identify records in a table; however, a primary key does not allow NULL values, whereas a unique key can allow one NULL value per column and can be used for additional uniqueness constraints on columns other than the primary key.

Assessing Data Structure Visually



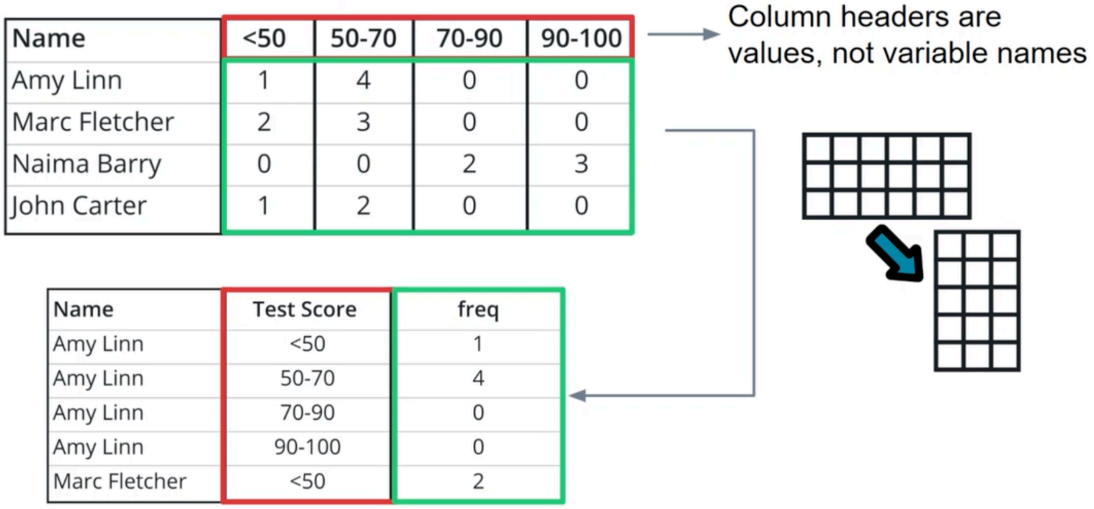
Common Data Structure Issues

- Column headers are values, not variable names.
- Multiple variables are stored in one column.
- Variables are stored in both rows and columns.
- Multiple types of observational units are stored in the same table.
- A single observational unit is stored in multiple tables.

Cleaning Data

Data Structuring Issues and Techniques

Unpivoting or Melting

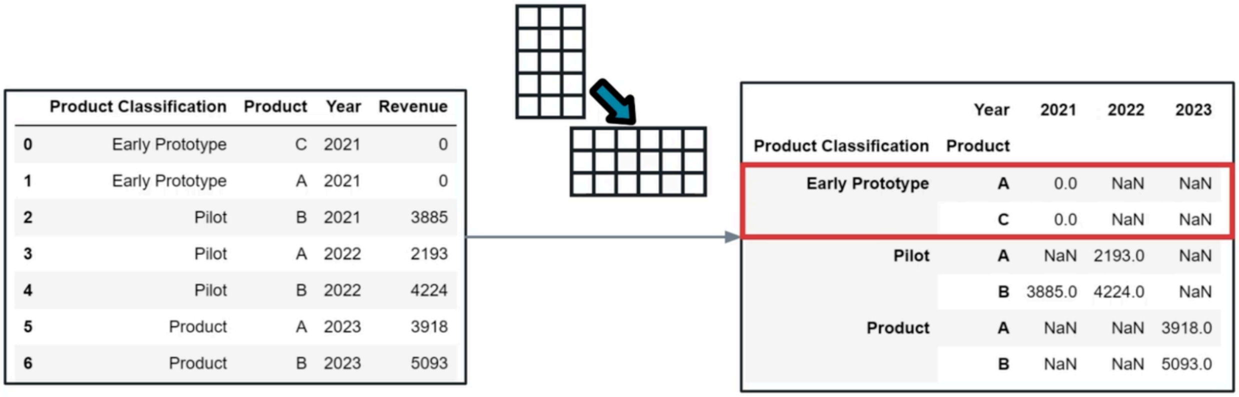


Name	<50	50-70	70-90	90-100
Amy Linn	1	4	0	0
Marc Fletcher	2	3	0	0
Naima Barry	0	0	2	3
John Carter	1	2	0	0

Column headers are values, not variable names

Name	Test Score	freq
Amy Linn	<50	1
Amy Linn	50-70	4
Amy Linn	70-90	0
Amy Linn	90-100	0
Marc Fletcher	<50	2

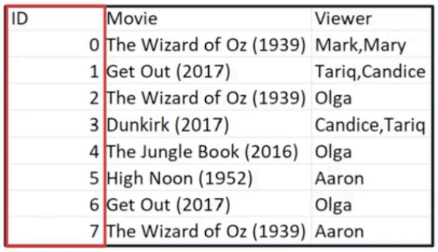
Pivoting



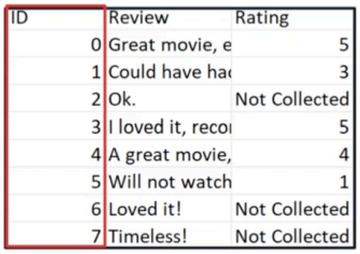
	Product Classification	Product	Year	Revenue
0	Early Prototype	C	2021	0
1	Early Prototype	A	2021	0
2	Pilot	B	2021	3885
3	Pilot	A	2022	2193
4	Pilot	B	2022	4224
5	Product	A	2023	3918
6	Product	B	2023	5093

	Year	2021	2022	2023
Product Classification	Product			
Early Prototype	A	0.0	NaN	NaN
	C	0.0	NaN	NaN
Pilot	A	NaN	2193.0	NaN
	B	3885.0	4224.0	NaN
Product	A	NaN	NaN	3918.0
	B	NaN	NaN	5093.0

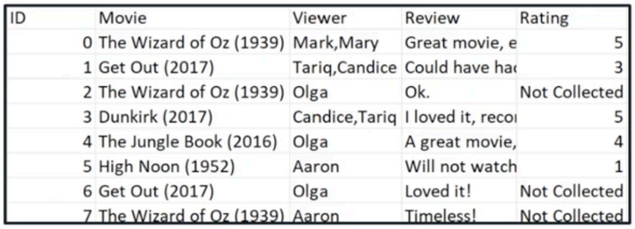
Merging



ID	Movie	Viewer
0	The Wizard of Oz (1939)	Mark,Mary
1	Get Out (2017)	Tariq,Candice
2	The Wizard of Oz (1939)	Olga
3	Dunkirk (2017)	Candice,Tariq
4	The Jungle Book (2016)	Olga
5	High Noon (1952)	Aaron
6	Get Out (2017)	Olga
7	The Wizard of Oz (1939)	Aaron

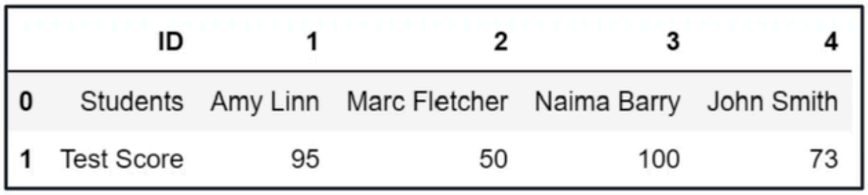


ID	Review	Rating
0	Great movie, e	5
1	Could have ha	3
2	Ok.	Not Collected
3	I loved it, reco	5
4	A great movie,	4
5	Will not watch	1
6	Loved it!	Not Collected
7	Timeless!	Not Collected

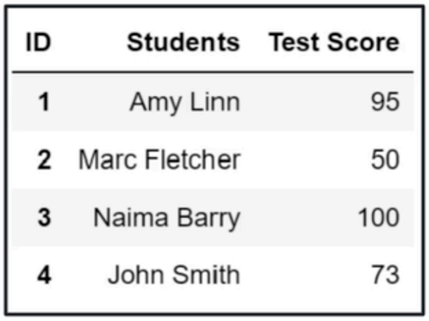


ID	Movie	Viewer	Review	Rating
0	The Wizard of Oz (1939)	Mark,Mary	Great movie, e	5
1	Get Out (2017)	Tariq,Candice	Could have ha	3
2	The Wizard of Oz (1939)	Olga	Ok.	Not Collected
3	Dunkirk (2017)	Candice,Tariq	I loved it, reco	5
4	The Jungle Book (2016)	Olga	A great movie,	4
5	High Noon (1952)	Aaron	Will not watch	1
6	Get Out (2017)	Olga	Loved it!	Not Collected
7	The Wizard of Oz (1939)	Aaron	Timeless!	Not Collected

Transposing

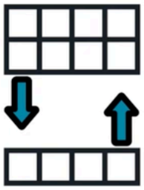


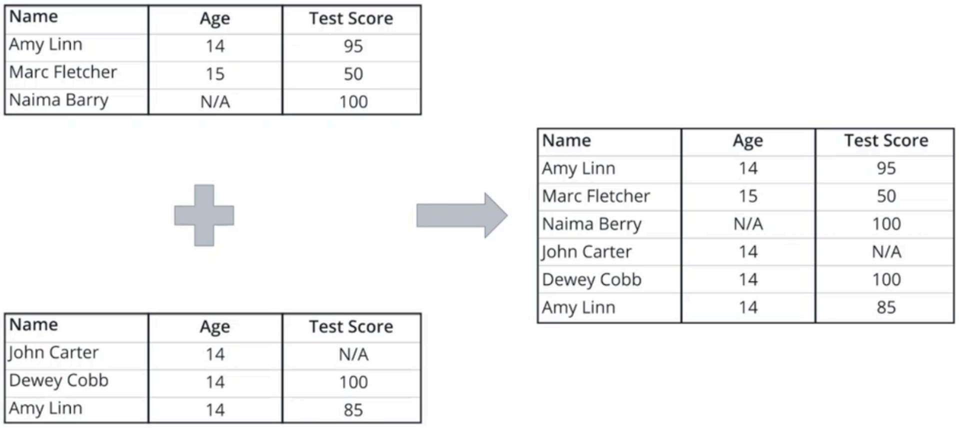
	ID	1	2	3	4
0	Students	Amy Linn	Marc Fletcher	Naima Barry	John Smith
1	Test Score	95	50	100	73



ID	Students	Test Score
1	Amy Linn	95
2	Marc Fletcher	50
3	Naima Barry	100
4	John Smith	73

Appending or Concatenating



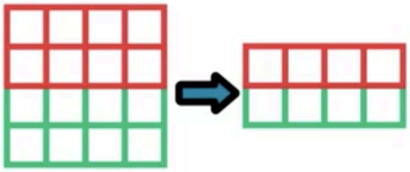


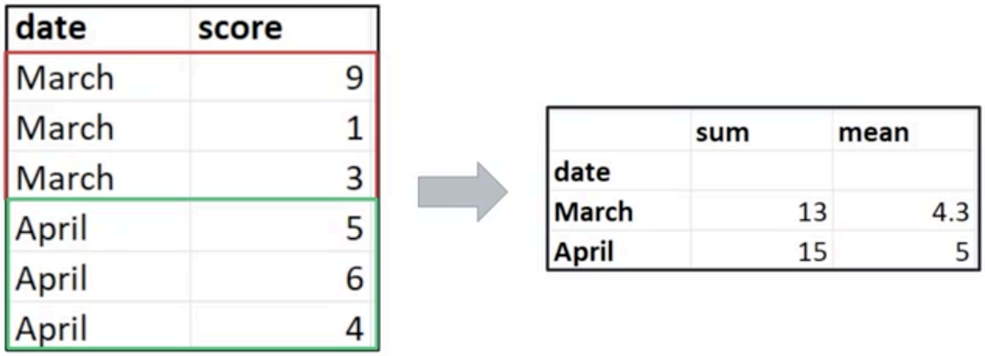
Name	Age	Test Score
Amy Linn	14	95
Marc Fletcher	15	50
Naima Barry	N/A	100

Name	Age	Test Score
John Carter	14	N/A
Dewey Cobb	14	100
Amy Linn	14	85

Name	Age	Test Score
Amy Linn	14	95
Marc Fletcher	15	50
Naima Berry	N/A	100
John Carter	14	N/A
Dewey Cobb	14	100
Amy Linn	14	85

Group-by and Aggregation





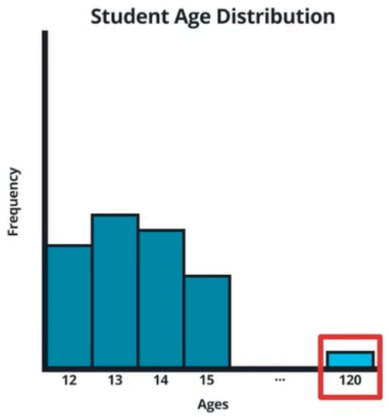
date	score
March	9
March	1
March	3
April	5
April	6
April	4

	sum	mean
date		
March	13	4.3
April	15	5

Dealing With Outliers

What Are Outliers?

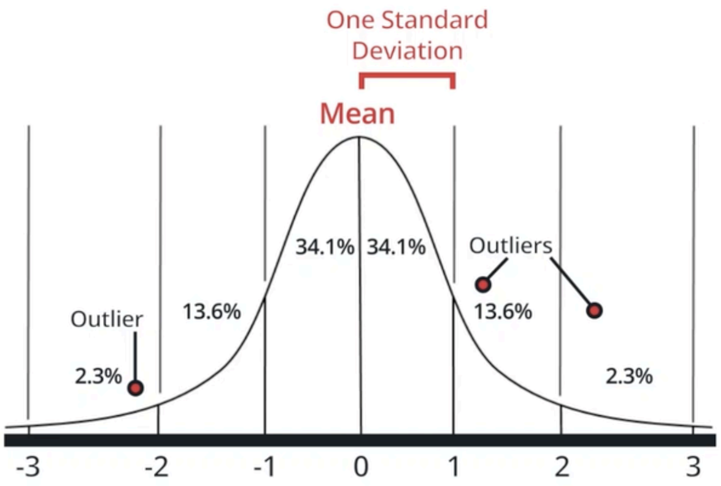
An outlier is a data point that is considerable distinct and from other data samples

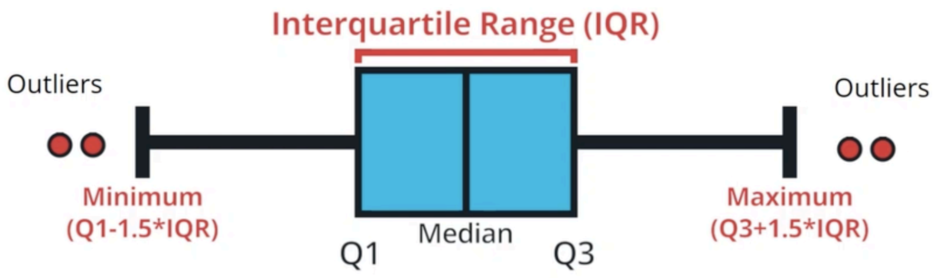


Age of 120 is an outlier ✖

Age of 120 is impossible

Identify Outliers

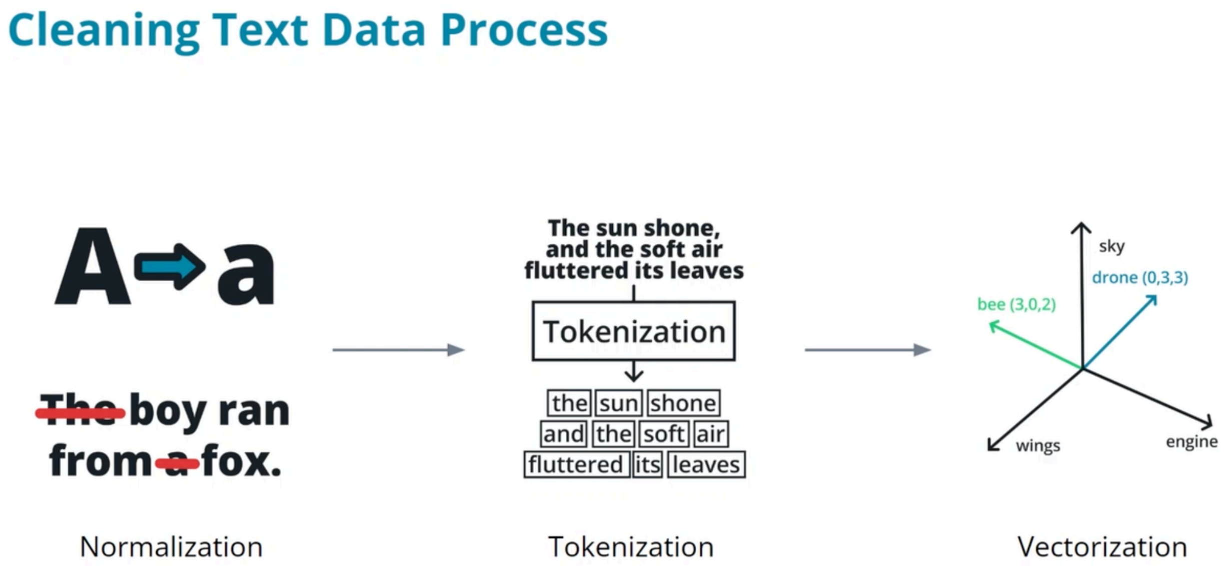




Dealing With Missing Data

It's important to identify if the missing values in the dataset are correctly represented. Sometimes the missing values are represented as characters like "-" and "#" or texts like "no data", which can be easily missed using the .isna() method. So we should always check if missing values are correctly represented and replace them with proper values like np.nan

Cleaning Text Data


Cleaning Text Data Process
A ⇨ a
~~The~~ boy ran from ~~a~~ fox.
The sun shone, and the soft air fluttered its leaves
Tokenization
the sun shone and the soft air fluttered its leaves
sky
drone (0,3,3)
bee (3,0,2)
wings
engine
Normalization
Tokenization
Vectorization

Normalization + additional cleaning.
Normalization involves getting text into a standard format (e.g., converting sentences into lowercase).
Additional cleaning operations can include:

- Removing words that aren't relevant to us, like “a”, “the”, and “is” that are called stopwords.
- Reducing words to their root base, which means turning all mentions of going and gone to go). It is also known as stemming.

Tokenization: In this stage, you divide your text into individual tokens from your dataset.

Vectorization: In this stage, you convert these tokens into a machine-readable format so they can then for example be used to train machine-learning models!

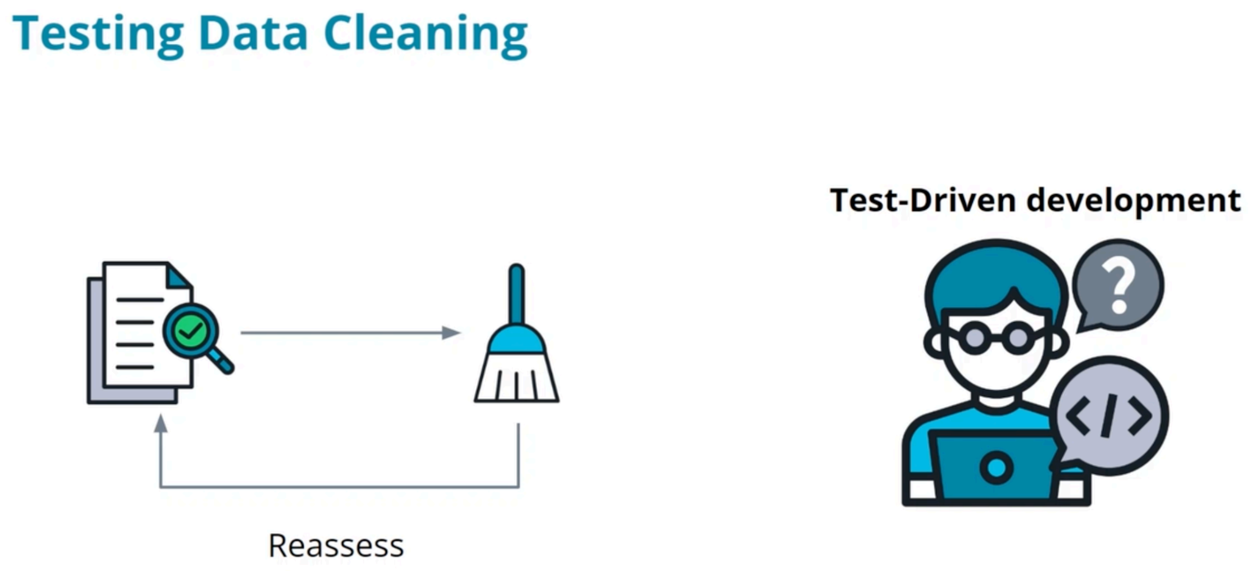
Cleaning Time Series Data

Some key considerations to remember are:

- Convert dates as pandas datetime objects using pd.to_datetime(COLUMN_NAME)
- Set your dataframe's index as the date/time column to turn the index into a Datetime index for more advanced functionality using df = df.set_index(COLUMN_NAME)

- Use resampling with time series to provide a time-based grouping with a target frequency, and aggregate based on that. For example, to get the yearly (Y) mean of values in the "COLUMN_NAME" column, use: df.resample("Y")[COLUMN_NAME].mean()

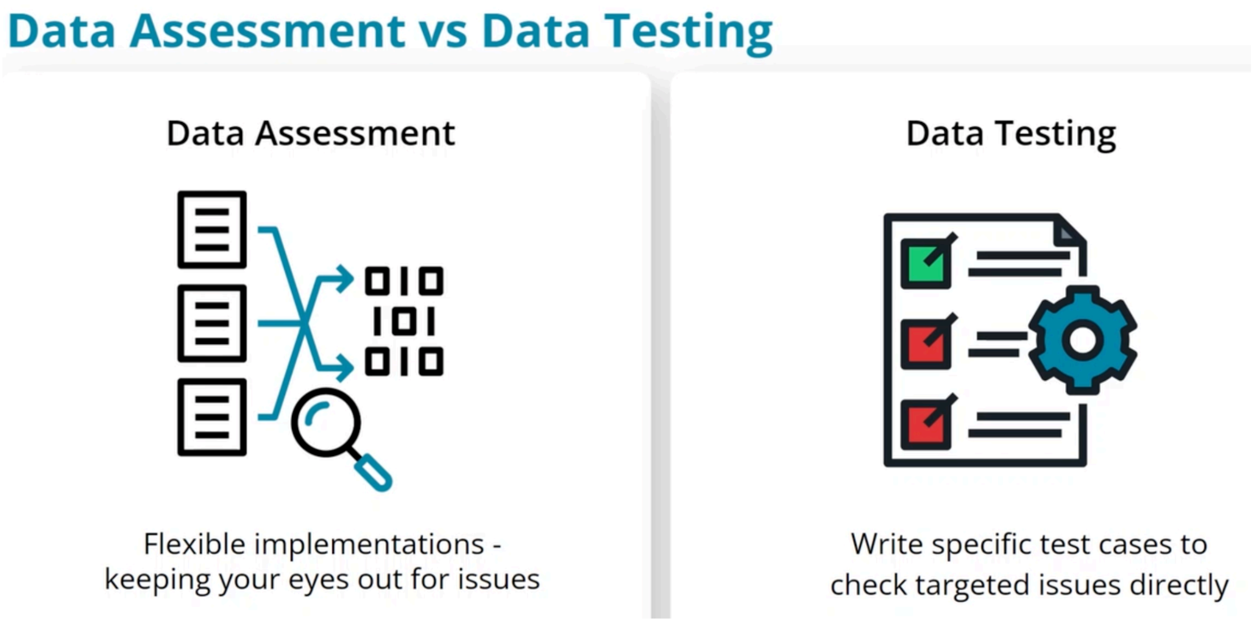
Testing Data Cleaning Visually



Testing Data Cleaning

Test-Driven development

Reassess



Data Assessment vs Data Testing

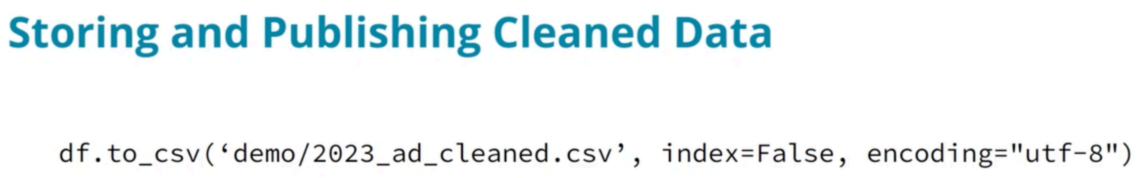
Data Assessment

Flexible implementations - keeping your eyes out for issues

Data Testing

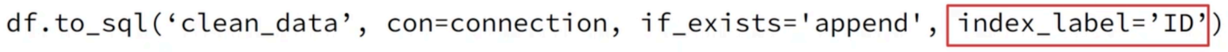
Write specific test cases to check targeted issues directly

Storing and Publishing Cleaned Data

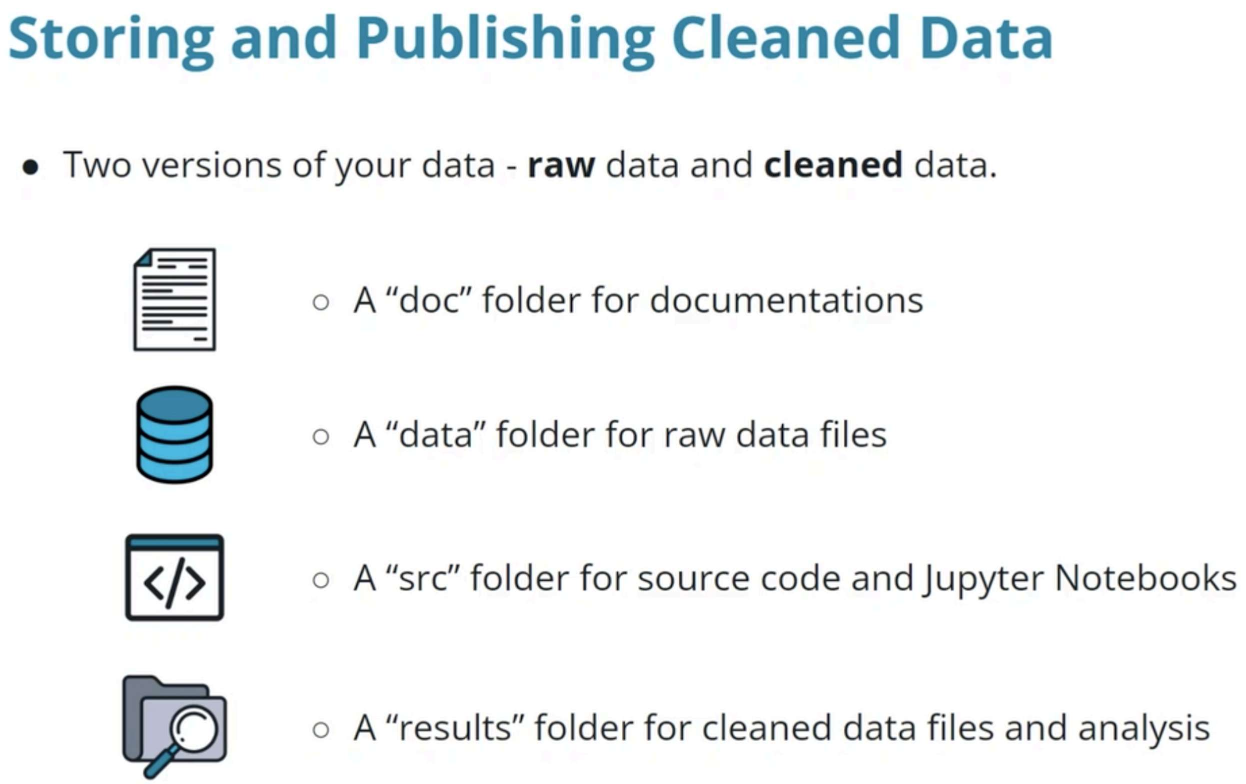


Storing and Publishing Cleaned Data

```
df.to_csv(‘demo/2023_ad_cleaned.csv’, index=False, encoding="utf-8")
```



```
df.to_sql(‘clean_data’, con=connection, if_exists='append', index_label=’ID’)
```



Storing and Publishing Cleaned Data

- Two versions of your data - **raw** data and **cleaned** data.
  - A “doc” folder for documentations
  - A “data” folder for raw data files
  - A “src” folder for source code and Jupyter Notebooks
  - A “results” folder for cleaned data files and analysis

Course 3: Data Visualization with Matplotlib and Seaborn

Data Visualization with Matplotlib and Seaborn

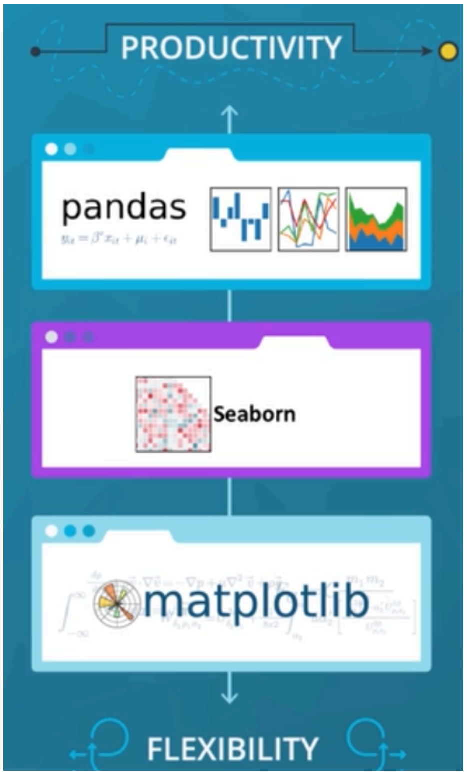
Exploratory vs. Explanatory Analyses

There are two main reasons for creating visuals using data:

- Exploratory analysis is done when you are searching for insights. These visualizations don't need to be perfect. You are using plots to find insights, but they don't need to be aesthetically appealing. You are the consumer of these plots, and you need to be able to find the answer to your questions from these plots.
- Explanatory analysis is done when you are providing your results for others. These visualizations need to provide you the emphasis necessary to convey your message. They should be accurate, insightful, and visually appealing.

The five steps of the data analysis process:

- Extract - Obtain the data from a spreadsheet, SQL, the web, etc.
- Clean - Here, we could use exploratory visuals.
- Explore - Here, we use exploratory visuals.
- Analyze - Here, we might use either exploratory or explanatory visuals.
- Share - Here is where explanatory visuals live.



Design of Visualizations

Levels of Measurement & Types of Data

The Four Levels of Measurement

In order to choose an appropriate plot type or method of analysis for your data, you need to understand the types of data you have. One common method divides the data into four levels of measurement:

Qualitative or categorical types (non-numeric types)

- 1. Nominal data: pure labels without inherent order (no label is intrinsically greater or less than any other)
- 2. Ordinal data: labels with an intrinsic order or ranking (comparison operations can be made between values, but the magnitude of differences are not be well-defined)

Quantitative or numeric types

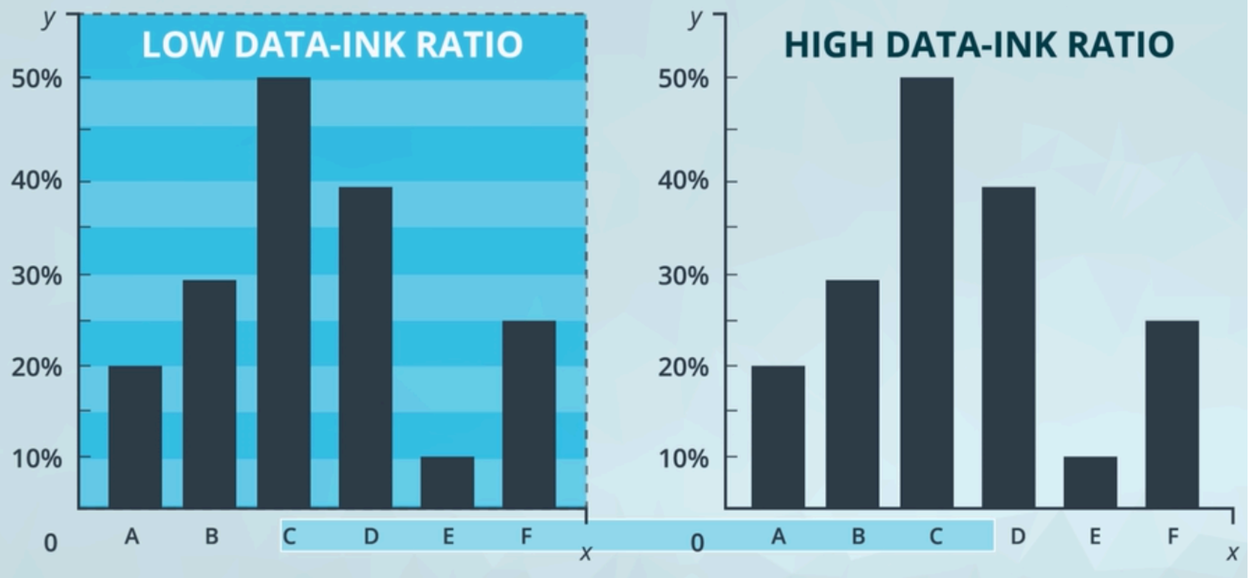
- 3. Interval data: numeric values where absolute differences are meaningful (addition and subtraction operations can be made)
- 4. Ratio data: numeric values where relative differences are meaningful (multiplication and division operations can be made)

All quantitative-type variables also come in one of two varieties: discrete and continuous.

- Discrete quantitative variables can only take on a specific set values at some maximum level of precision.
- Continuous quantitative variables can (hypothetically) take on values to any level of precision.

Distinguishing between continuous and discrete can be a little tricky – a rule of thumb is if there are few levels, and values can't be subdivided into further units, then it's discrete. Otherwise, it's continuous. If you have a scale that can only take natural number values between 1 and 5, that's discrete. A quantity that can be measured to two digits, e.g. 2.72, is best characterized as continuous, since we might hypothetically be able to measure to even more digits, e.g. 2.718. A tricky case like test scores measured between 0 and 100 can only be divided down to single integers, making it initially seem discrete. But since there are so many values, such a feature is usually considered as continuous.

Data Ink Ratio



The data-ink ratio, credited to Edward Tufte, is directly related to the idea of chart junk. The more of the ink in your visual that is related to conveying the message in the data, the better.

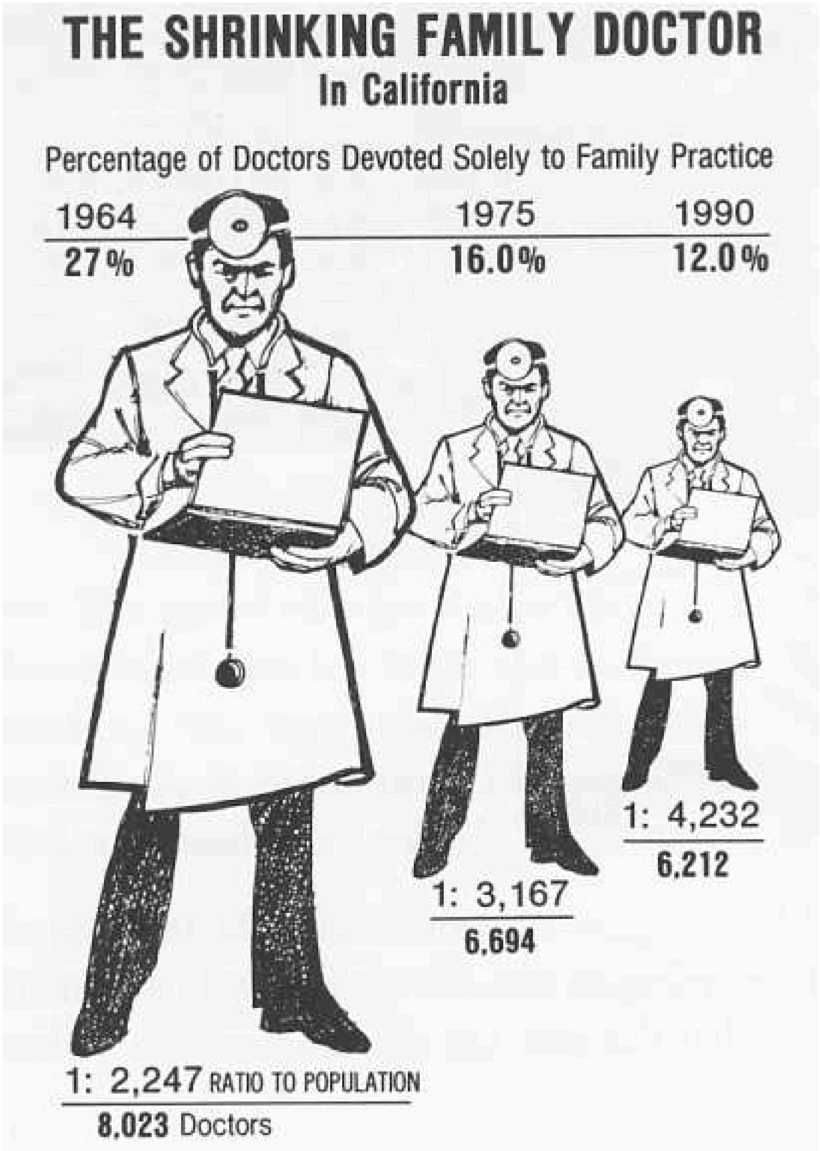
Design Integrity

It is key that when you build plots you maintain integrity for the underlying data.

One of the main ways discussed here for looking at data integrity was with the lie factor. Lie factor depicts the degree to which a visualization distorts or misrepresents the data values being plotted. It is calculated in the following way:

$$\text{lie factor} = \frac{\Delta\text{visual}/\text{visual}_{\text{start}}}{\Delta\text{data}/\text{data}_{\text{start}}}$$

The delta symbol (Δ) stands for difference or change. In words, the lie factor is the relative change shown in the graphic divided by the actual relative change in the data. Ideally, the lie factor should be 1: any other value means that there is some mismatch in the ratio of depicted change to actual change.



The number of pixels related to the largest image is 79,000 and 16,500 for the smallest. The percentage change is 27% to 12%. So, the lie factor is calculated as

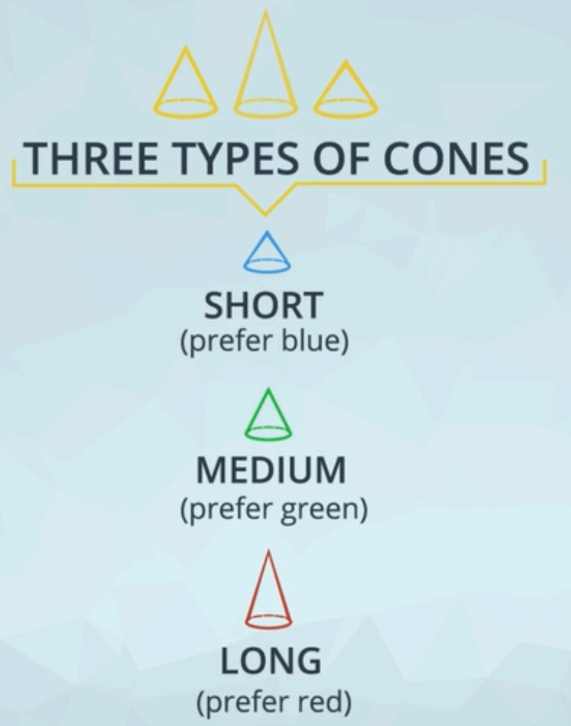
$$\text{lie factor} = \frac{(79000 - 16500)/16500}{(27 - 12)/12} = 3.03$$

Using Color

Color can both help and hurt a data visualization. Three tips for using color effectively.

- Before adding color to a visualization, start with black and white.
- When using color, use less intense colors - not all the colors of the rainbow, which is the default in many software applications.
- Color for communication. Use color to highlight your message and separate groups of interest. Don't add color just to have color in your visualization.

Designing for Color Blindness



THREE TYPES OF CONES

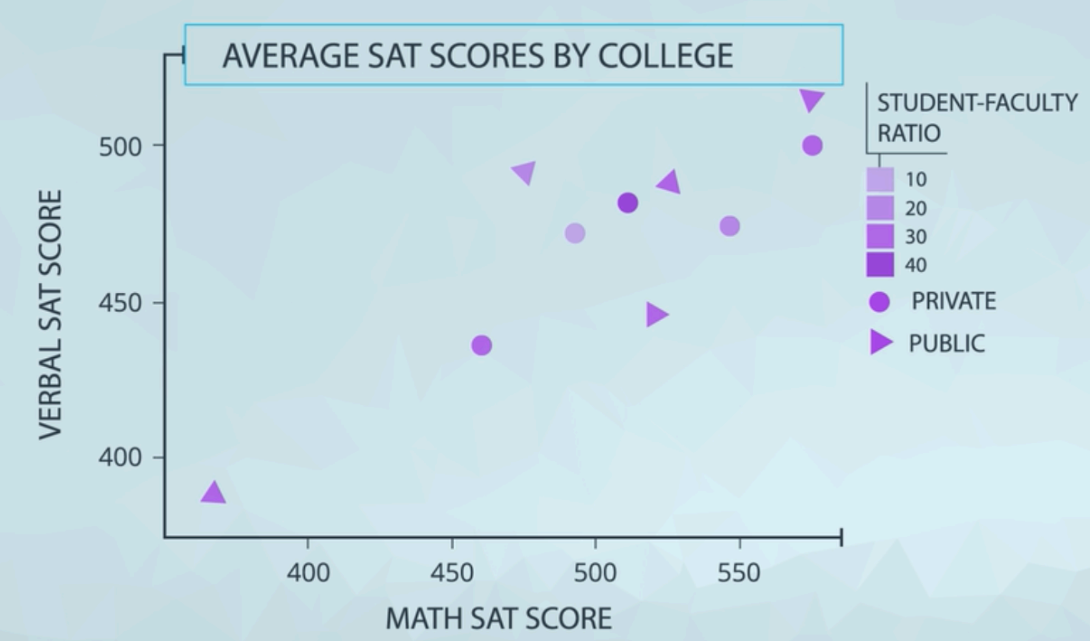
SHORT
(prefer blue)

MEDIUM
(prefer green)

LONG
(prefer red)

To be sensitive to those with colorblindness, you should use color palettes that do not move from red to green without using another element to distinguish this change like shape, position, or lightness. Both of these colors appear in a yellow tint to individuals with the most common types of colorblindness. Instead, use colors on a blue to orange palette.

Shape, Size, & Other Tools



In general, color and shape are best for categorical variables, while the size of marker can assist in adding additional quantitative data, as we demonstrated here.
Only use these additional encodings when absolutely necessary. Often, overuse of these additional encodings suggest you are providing too much information in a single plot. Instead, it might be better to break the information into multiple individual messages, so the audience can understand every aspect of your message.

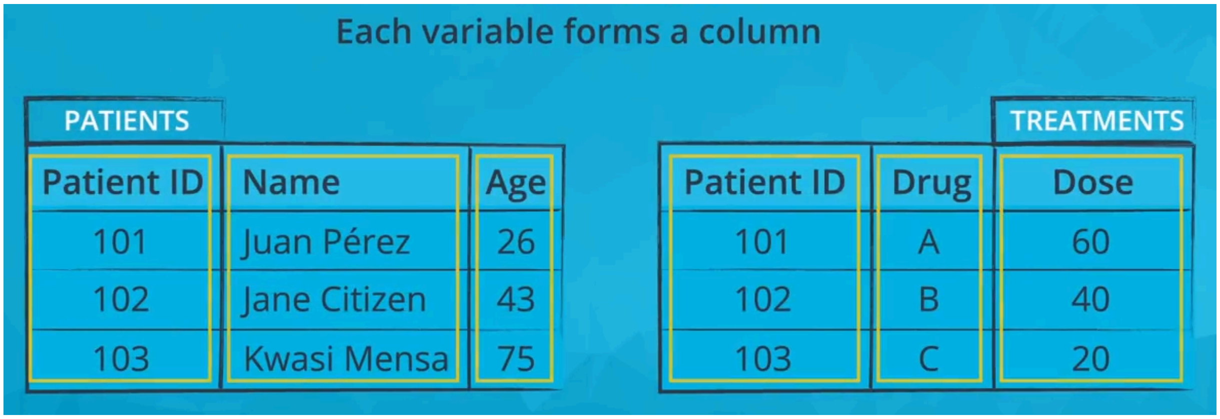
Univariate Exploration of Data

Tidy Data

In this course, it is expected that your data is organized in some kind of tidy format. In short, a tidy dataset is a tabular dataset where:

- each variable is a column
- each observation is a row
- each type of observational unit is a table

The first three images below depict a tidy dataset. This tidy dataset is in the field of healthcare and has two tables: one for patients (with their patient ID, name, and age) and one for treatments (with patient ID, what drug that patient is taking, and the dose of that drug).



Each variable forms a column

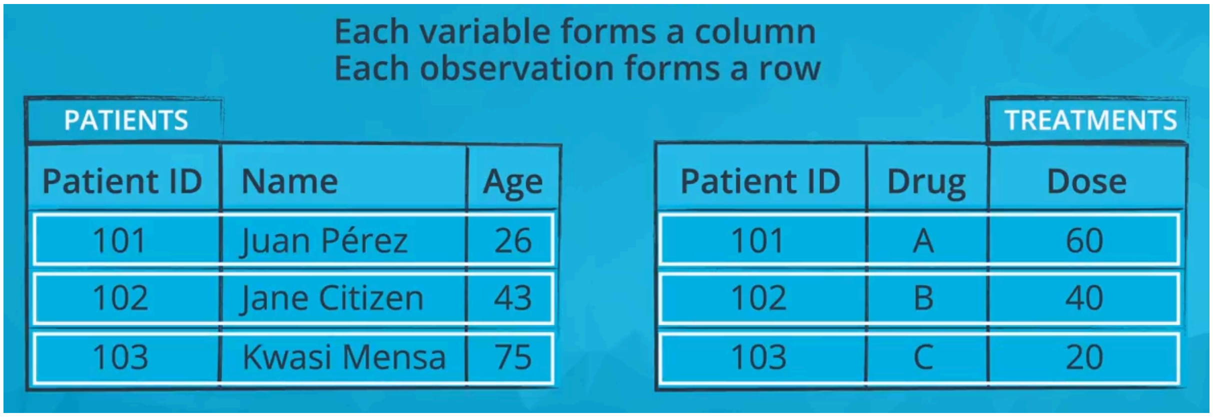
PATIENTS

Patient ID	Name	Age
101	Juan Pérez	26
102	Jane Citizen	43
103	Kwasi Mensa	75

TREATMENTS

Patient ID	Drug	Dose
101	A	60
102	B	40
103	C	20

Each variable in a tidy dataset must have its own column



Each variable forms a column
Each observation forms a row

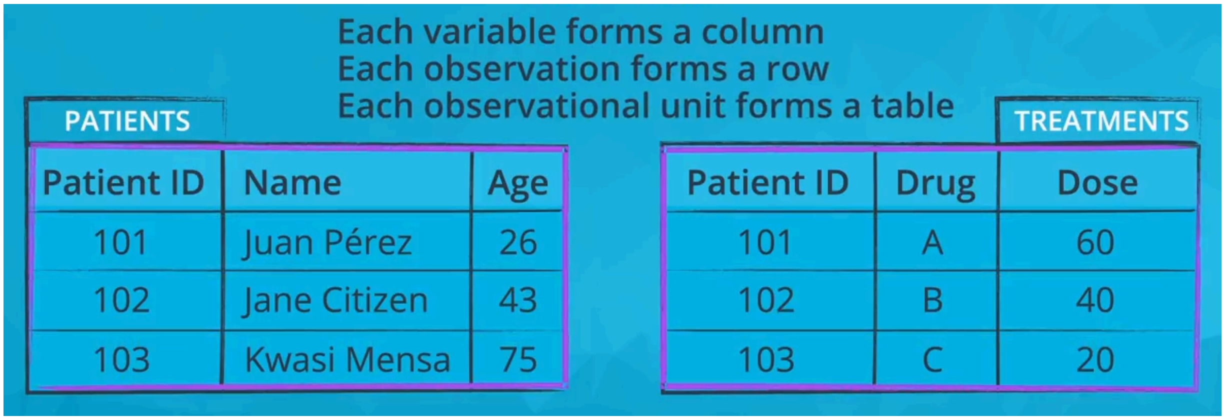
PATIENTS

Patient ID	Name	Age
101	Juan Pérez	26
102	Jane Citizen	43
103	Kwasi Mensa	75

TREATMENTS

Patient ID	Drug	Dose
101	A	60
102	B	40
103	C	20

Each observation in a tidy dataset must have its own row



Each variable forms a column
Each observation forms a row
Each observational unit forms a table

PATIENTS

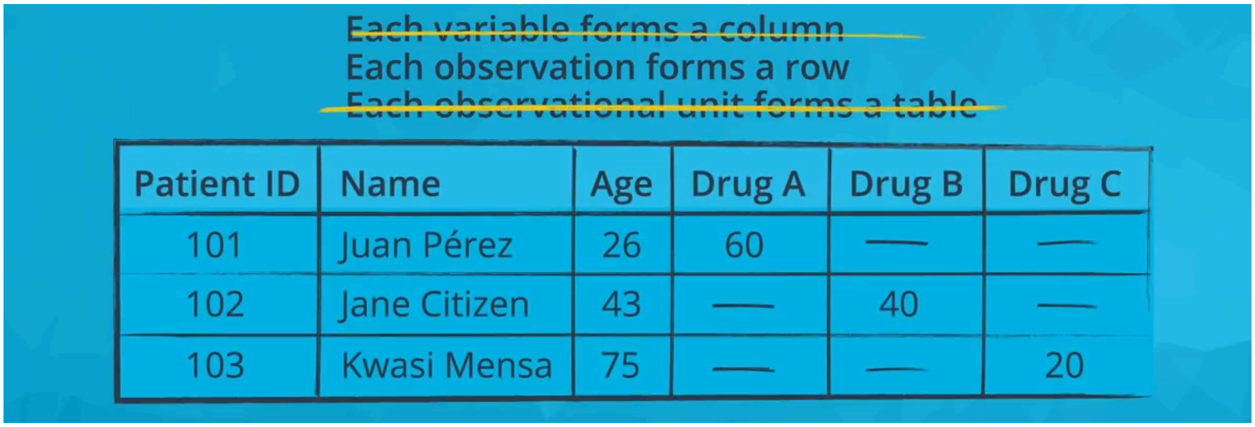
Patient ID	Name	Age
101	Juan Pérez	26
102	Jane Citizen	43
103	Kwasi Mensa	75

TREATMENTS

Patient ID	Drug	Dose
101	A	60
102	B	40
103	C	20

Each observational unit in a tidy dataset must have its own table

The next image depicts the same data but in one representation of a non-tidy format (there are other possible non-tidy representations). The Drug A, Drug B, and Drug C columns should form one 'Drug' column, since this is one variable. The entire table should be separated into two tables: a patients table and a treatments table.



~~Each variable forms a column~~
Each observation forms a row
~~Each observational unit forms a table~~

Patient ID	Name	Age	Drug A	Drug B	Drug C
101	Juan Pérez	26	60	—	—
102	Jane Citizen	43	—	40	—
103	Kwasi Mensa	75	—	—	20

Only the second rule of tidy data is satisfied in this non-tidy representation of the above data: each observation forms a row

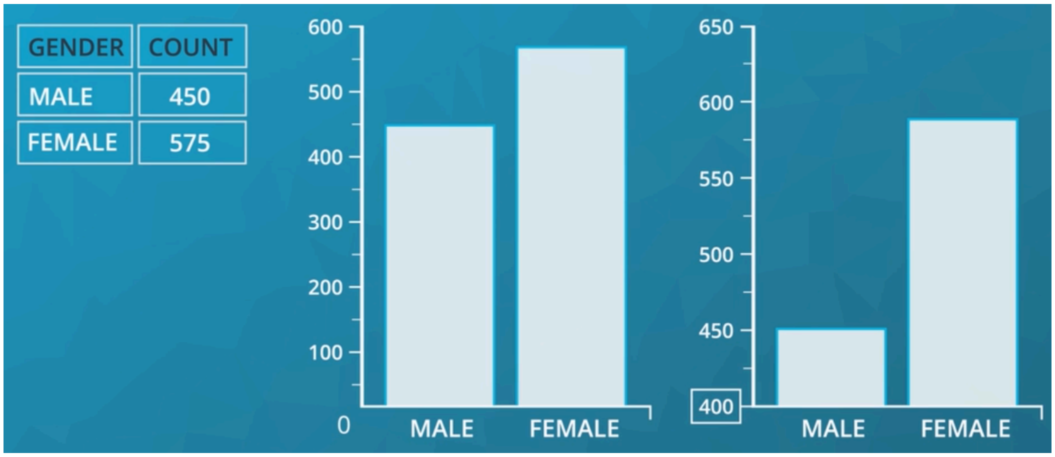
While the data provided to you in the course will all be tidy, in practice, you may need to perform tidying work before exploration. You should be comfortable with reshaping your data or perform transformations to split or combine features in your data, resulting in new data columns. These operations collectively are called data-wrangling.

This is also not to say that tidy data is the only useful form that data can take. In fact, as you work with a dataset, you might need to summarize it in a non-tidy form in order to generate appropriate visualizations. You'll see one example of this (bivariate plotting) in the next lesson, where categorical counts need to put into a matrix form in order to create a heat map.

Bar Charts

A bar chart depicts the distribution of a categorical variable. In a bar chart, each level of the categorical variable is depicted with a bar, whose height indicates the frequency of data points that take on that level.

- For nominal data, the bars can be ordered by frequency to easily see which category is the most common.
- Ordinal data should not be re-ordered because the inherent ordering of the levels is typically more important to display.

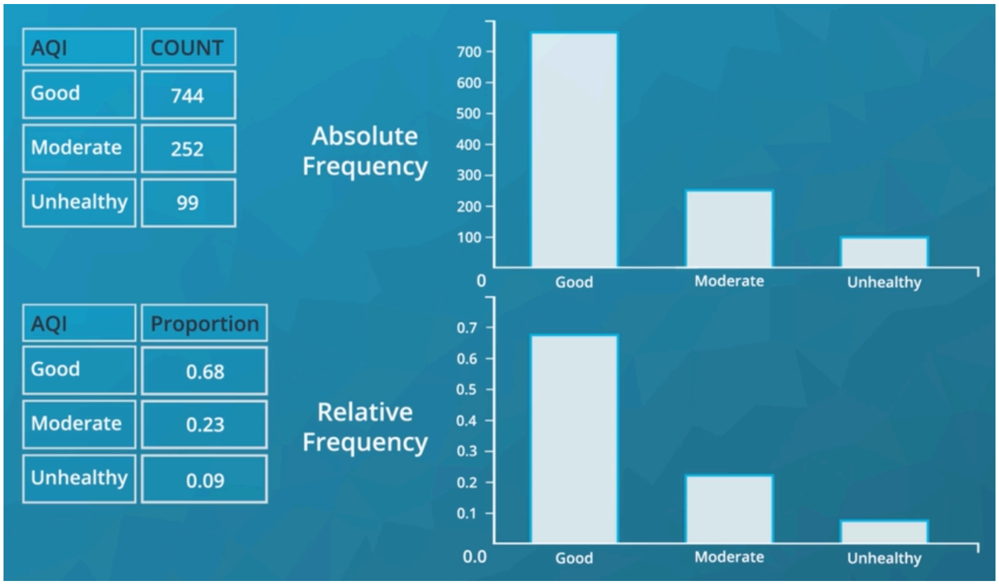


GENDER	COUNT
MALE	450
FEMALE	575

Absolute vs. Relative Frequency

In the previous concept, all bar charts were encoded with absolute frequency, which is the total number of data points for each category. While this can be helpful, there are times when you want to look at the frequency of a category as it relates to the total number of data points, this is relative frequency.

Changing the count axis to reflect relative proportions makes it easier to see how much each category contributes to the whole.

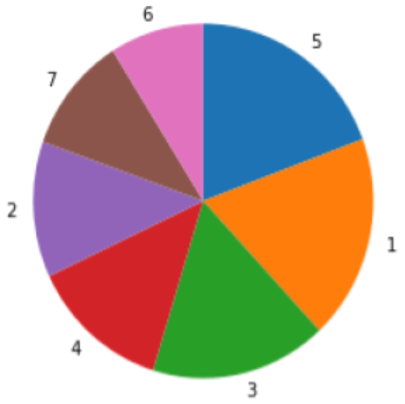


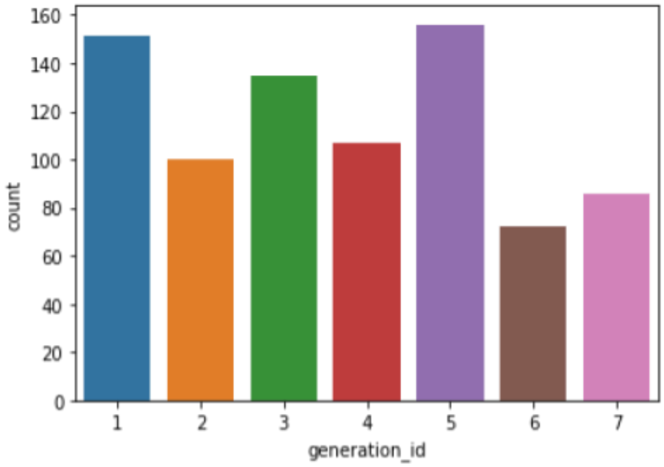
AQI	COUNT
Good	744
Moderate	252
Unhealthy	99

AQI	Proportion
Good	0.68
Moderate	0.23
Unhealthy	0.09

Pie Charts

A pie chart is a common univariate plot type that is used to depict relative frequencies for levels of a categorical variable. Frequencies in a pie chart are depicted as wedges drawn on a circle: the larger the angle or area, the more common the categorical value taken. Use a Pie chart only when the number of categories is less, and you'd like to see the proportion of each category on a chart.





Guidelines to Use a Pie Chart

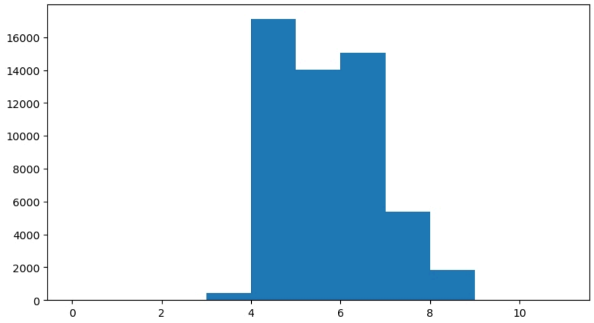
If you want to use a pie chart, try to follow certain guidelines:

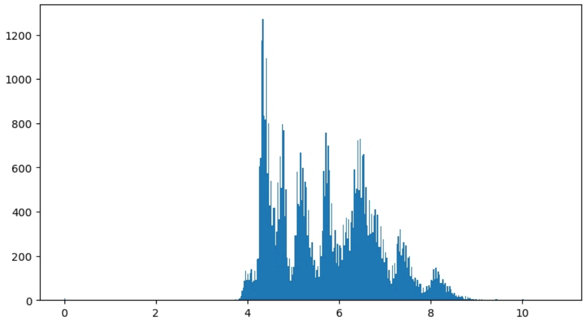
- Make sure that your interest is in relative frequencies. Areas should represent parts of a whole, rather than measurements on a second variable (unless that second variable can logically be summed up into some whole).
- Limit the number of slices plotted. A pie chart works best with two or three slices, though it's also possible to plot with four or five slices as long as the wedge sizes can be distinguished. If you have a lot of categories, or categories that have small proportional representation, consider grouping them together so that fewer wedges are plotted, or use an 'Other' category to handle them.
- Plot the data systematically. One typical method of plotting a pie chart is to start from the top of the circle, then plot each categorical level clockwise from most frequent to least frequent. If you have three categories and are interested in the comparison of two of them, a common plotting method is to place the two categories of interest on either side of the 12 o'clock direction, with the third category filling in the remaining space at the bottom.

If these guidelines cannot be met, then you should probably make use of a bar chart instead. A bar chart is a safer choice in general. The bar heights are more precisely interpreted than areas or angles, and a bar chart can be displayed more compactly than a pie chart. There's also more flexibility with a bar chart for plotting variables with a lot of levels, like plotting the bars horizontally.

Histograms

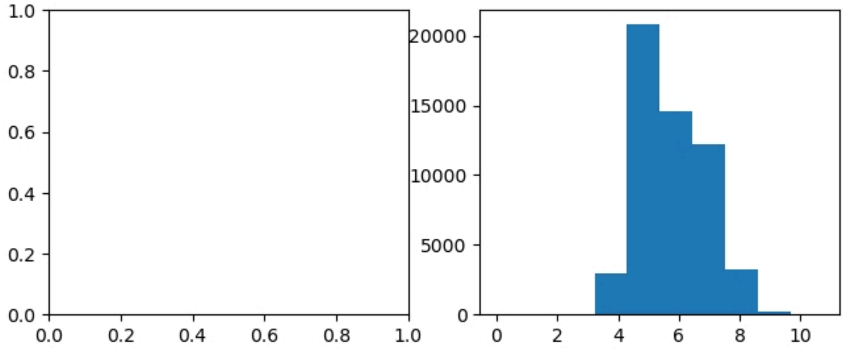
A histogram is used to plot the distribution of a numeric variable. It's the quantitative version of the bar chart. However, rather than plot one bar for each unique numeric value, values are grouped into continuous bins, and one bar for each bin is plotted to depict the number. You can use either Matplotlib or Seaborn to plot histograms. There is a mild variation in the calling syntax and what each library offers. For example, seaborn supports overlaying Gaussian Density Estimates.





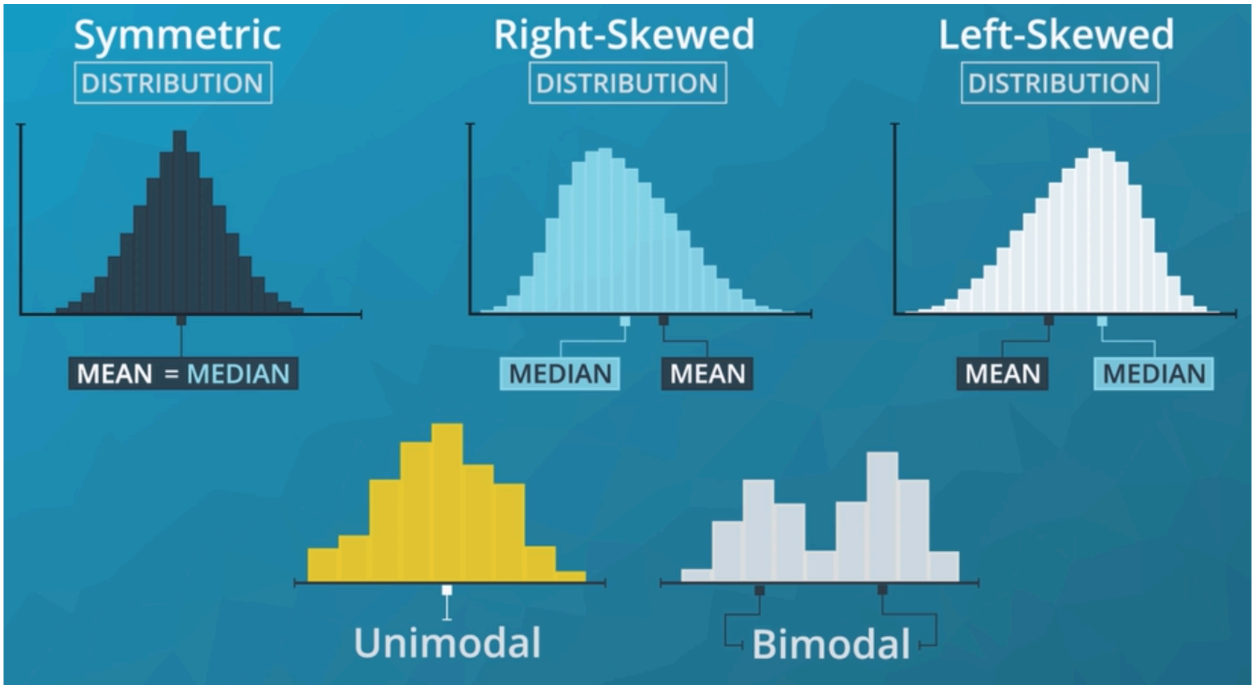
Figures, Axes, and Subplots

At this point, you've seen and had some practice with some basic plotting functions using matplotlib and seaborn. The previous page introduced something a little bit new: creating two side-by-side plots through the use of matplotlib's subplot() function. If you have any questions about how that or the figure() function worked, then read on. This page will discuss the basic structure of visualizations using matplotlib and how subplots work in that structure.

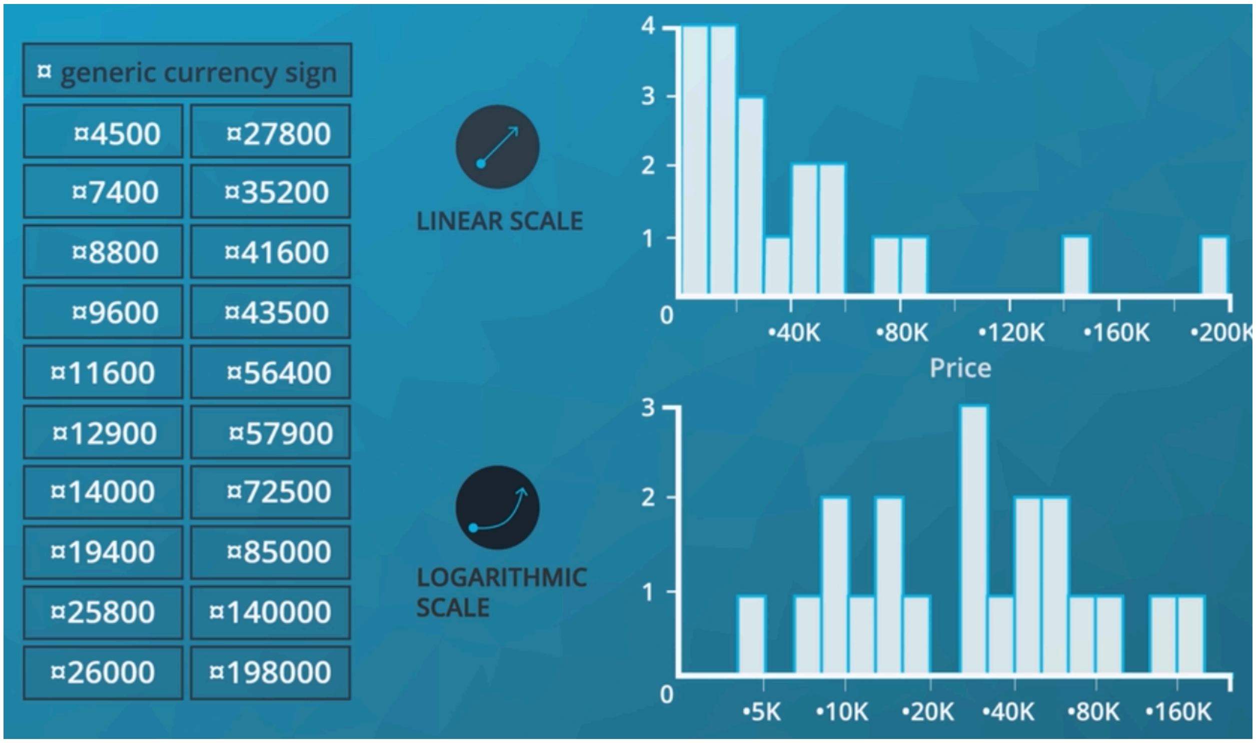


(Left) First matplotlib creates a figure object. (Center) Then an axes is placed on it. (Right) Then the chart is drawn.

Descriptive Statistics, Outliers and Axis Limits



Scales and Transformations



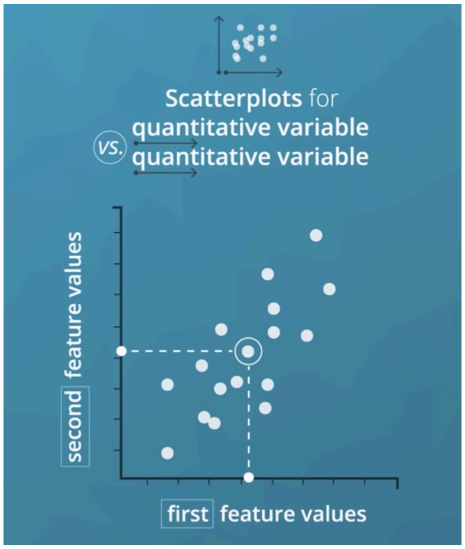
¤ generic currency sign	
¤4500	¤27800
¤7400	¤35200
¤8800	¤41600
¤9600	¤43500
¤11600	¤56400
¤12900	¤57900
¤14000	¤72500
¤19400	¤85000
¤25800	¤140000
¤26000	¤198000

Bivariate Exploration of Data

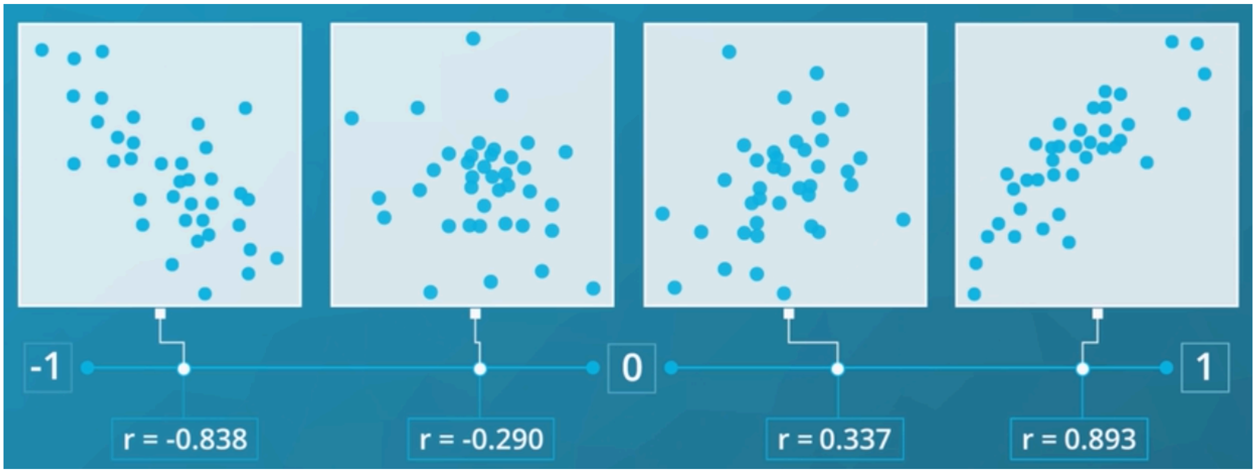
Scatterplots and Correlation

If we want to inspect the relationship between two numeric variables, the standard choice of plot is the scatterplot. In a scatterplot, each data point is plotted individually as a point, its x-position corresponding to one feature value and its y-position corresponding to the second.

A scatterplot is used to show the relationship between two quantitative variables. The two variables are indicated on X and Y-axis, respectively. Through the scatterplots, we can see clearly how these two variables correlate with each other.

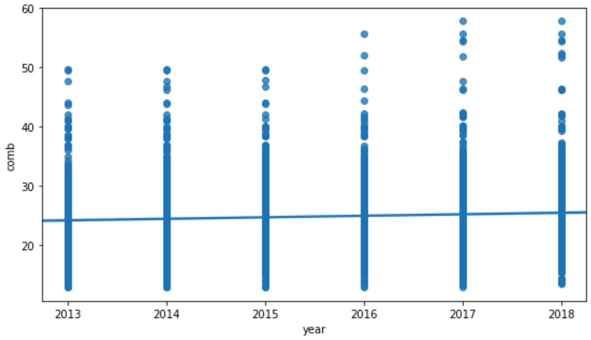


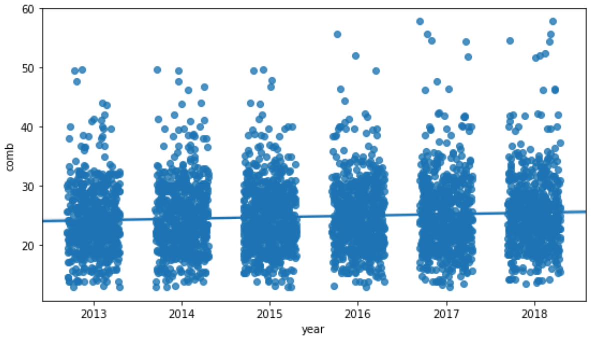
To quantify how strong the correlation is between the variables, we use a correlation coefficient. Pearson correlation coefficient (r) captures linear relationships. It is a value ranging from -1 to +1. A positive value of r indicates the increase in one variable tends to increase another variable. On the other hand, a negative r means the increase in one variable tends to cause a decrease in another variable. A value close to 0 indicates a weak correlation, and a value close to -1 and +1 indicates a strong correlation.



Overplotting, Transparency, and Jitter

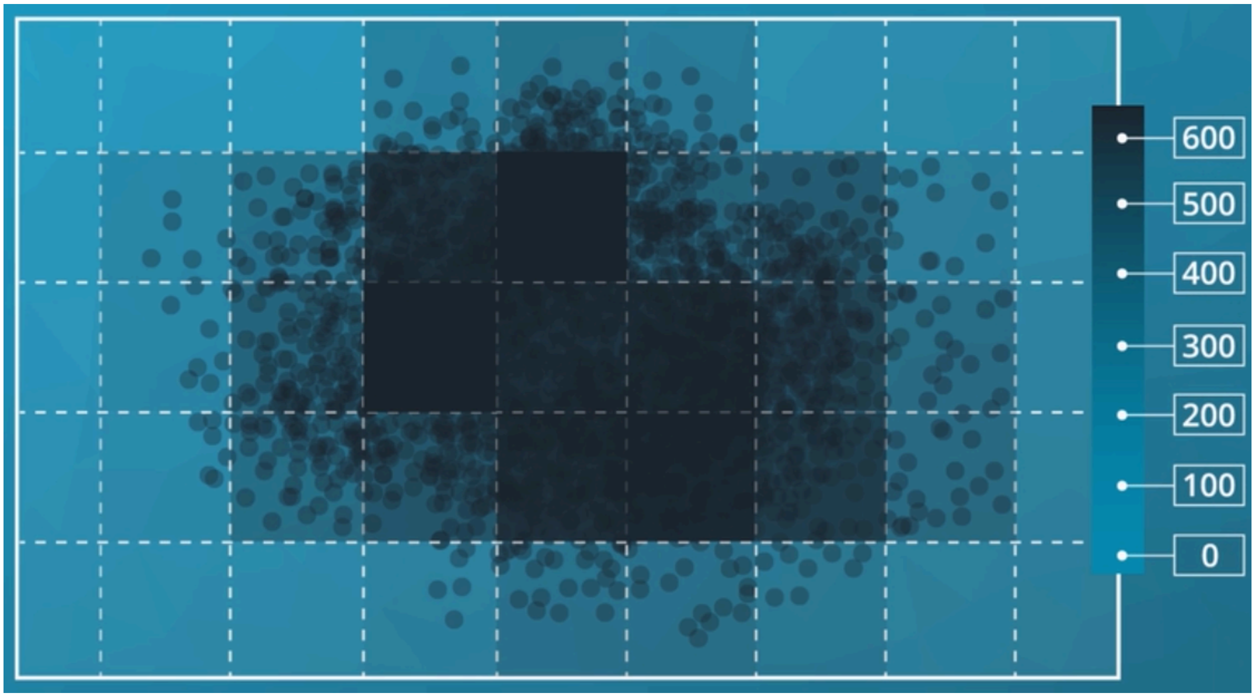
If we have a very large number of points to plot or our numeric variables are discrete-valued, then it is possible that using a scatterplot straightforwardly will not be informative. The visualization will suffer from overplotting, where the high amount of overlap in points makes it difficult to see the actual relationship between the plotted variables.





Heat Maps

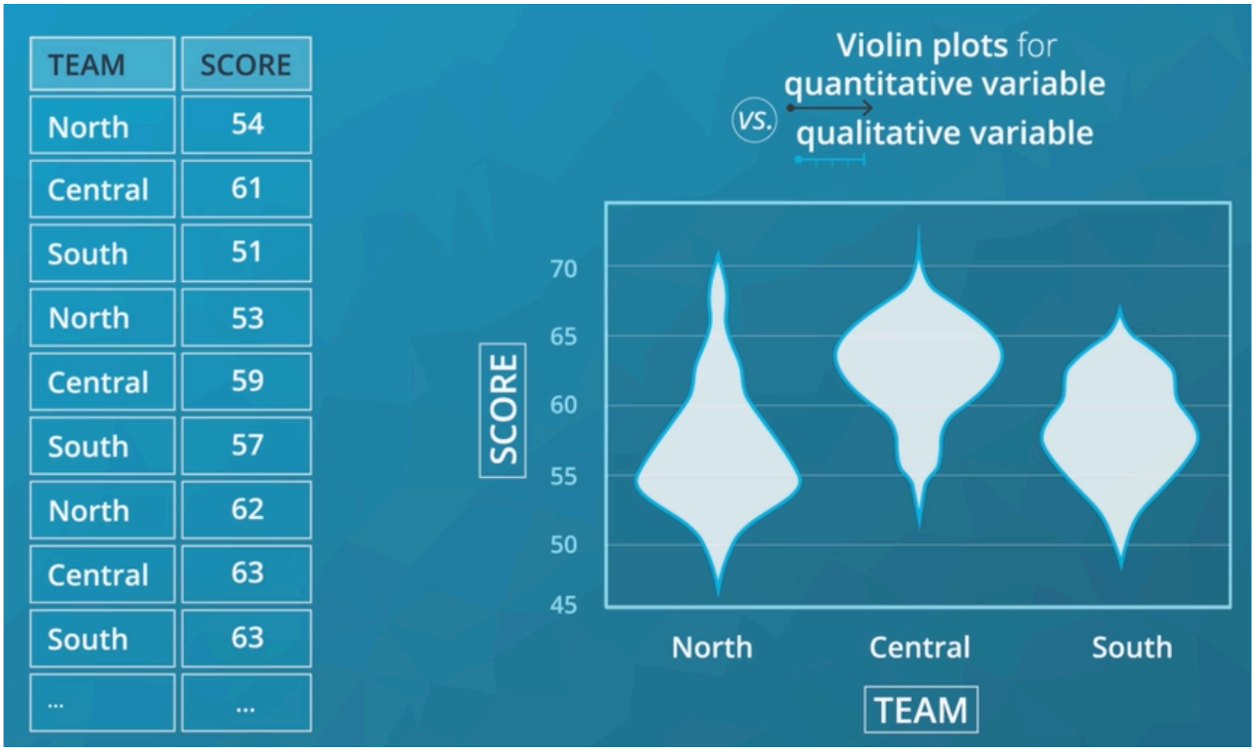
A heat map is a 2-d version of the histogram that can be used as an alternative to a scatterplot. Like a scatterplot, the values of the two numeric variables to be plotted are placed on the plot axes. Similar to a histogram, the plotting area is divided into a grid and the number of points in each grid rectangle is added up. Since there won't be room for bar heights, counts are indicated instead by grid cell color.



Violin Plots

Violin plots are a common way of showing the relationship between quantitative and qualitative variables. Instead of simply plotting summary statistics, violin plots use a kernel density estimate (KDE).

A KDE is like a smoothed histogram, an estimate of the data's probability distribution function. For each level of a qualitative variable, a distribution of the values on the quantitative variable is plotted. But beware, since these are estimates of densities, these can sometimes show unreal, particularly at the bounds of the distribution. For instance, given a distribution of cost could produce a violin with parts below zero.

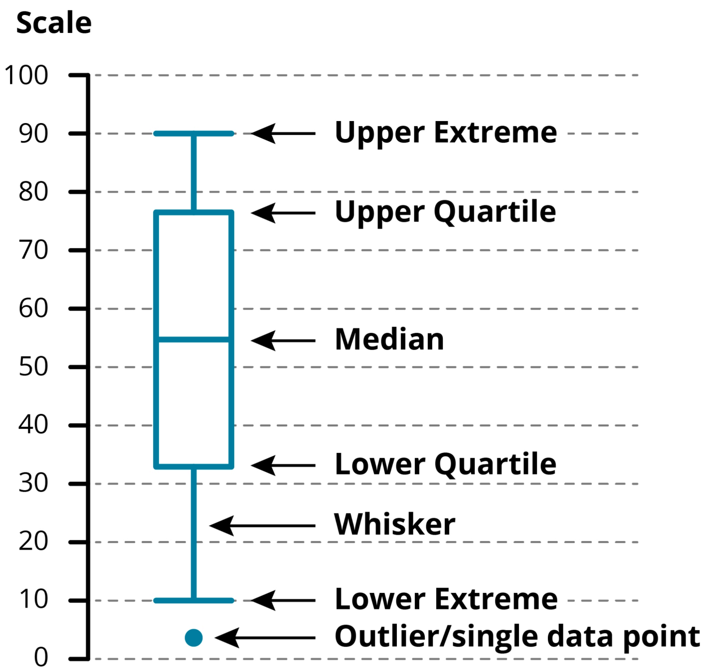


TEAM	SCORE
North	54
Central	61
South	51
North	53
Central	59
South	57
North	62
Central	63
South	63
...	...

Violin plots for quantitative variable VS. qualitative variable

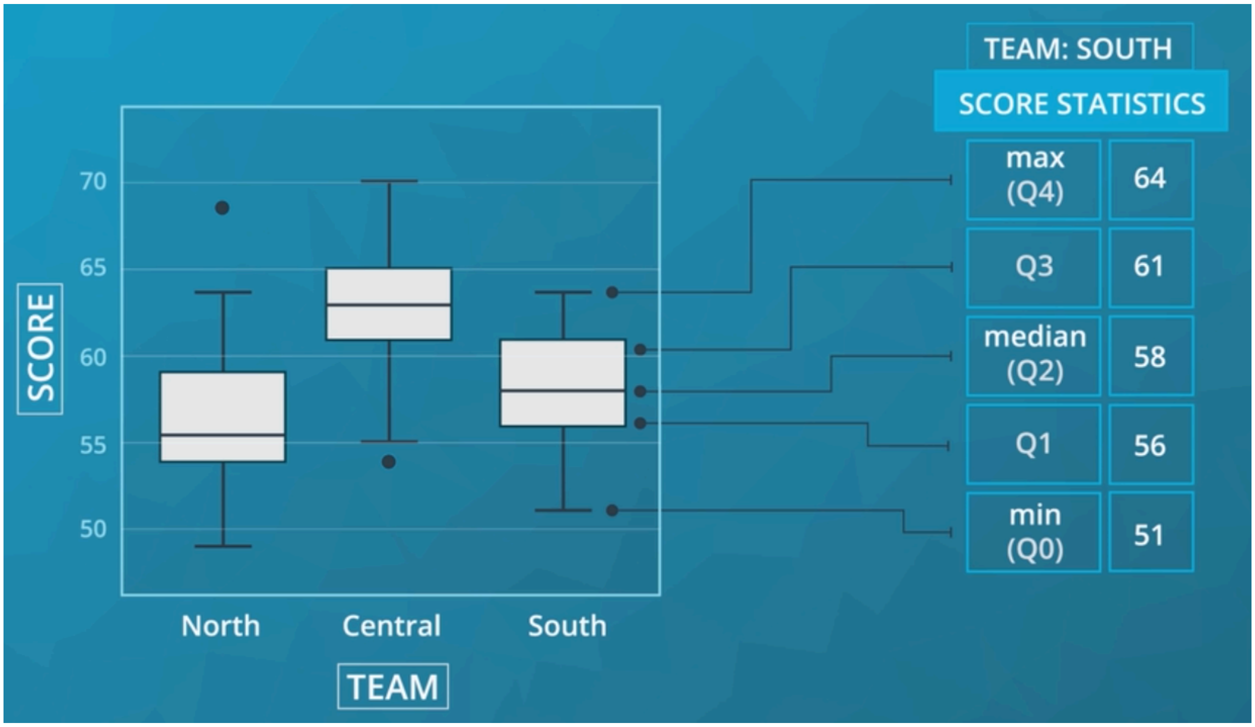
Box Plots

Relationships between numerical and categorical data are commonly shown uses box or violin plots.



Box plots simply display salient summary statistics on a plot, such as means, medians, and quartile boundaries.

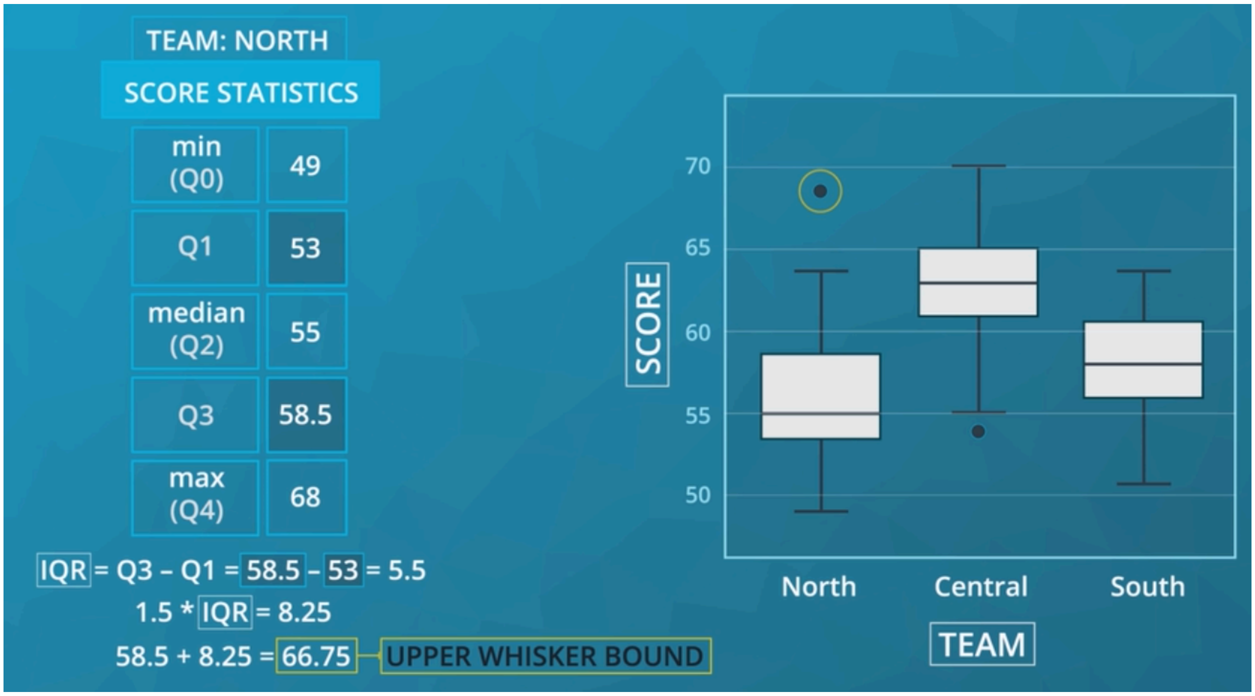
- Central line indicates the median
- Upper and lower edges show the 1st and 3rd quartiles
- Whiskers outside of the box indicate the largest and smallest values
- Outliers are plotted as points and are often in the 98th or 99th percentile of data
-



TEAM: SOUTH

SCORE STATISTICS

max (Q4)	64
Q3	61
median (Q2)	58
Q1	56
min (Q0)	51



TEAM: NORTH

SCORE STATISTICS

min (Q0)	49
Q1	53
median (Q2)	55
Q3	58.5
max (Q4)	68

IQR = Q3 – Q1 = 58.5 – 53 = 5.5

1.5 * IQR = 8.25

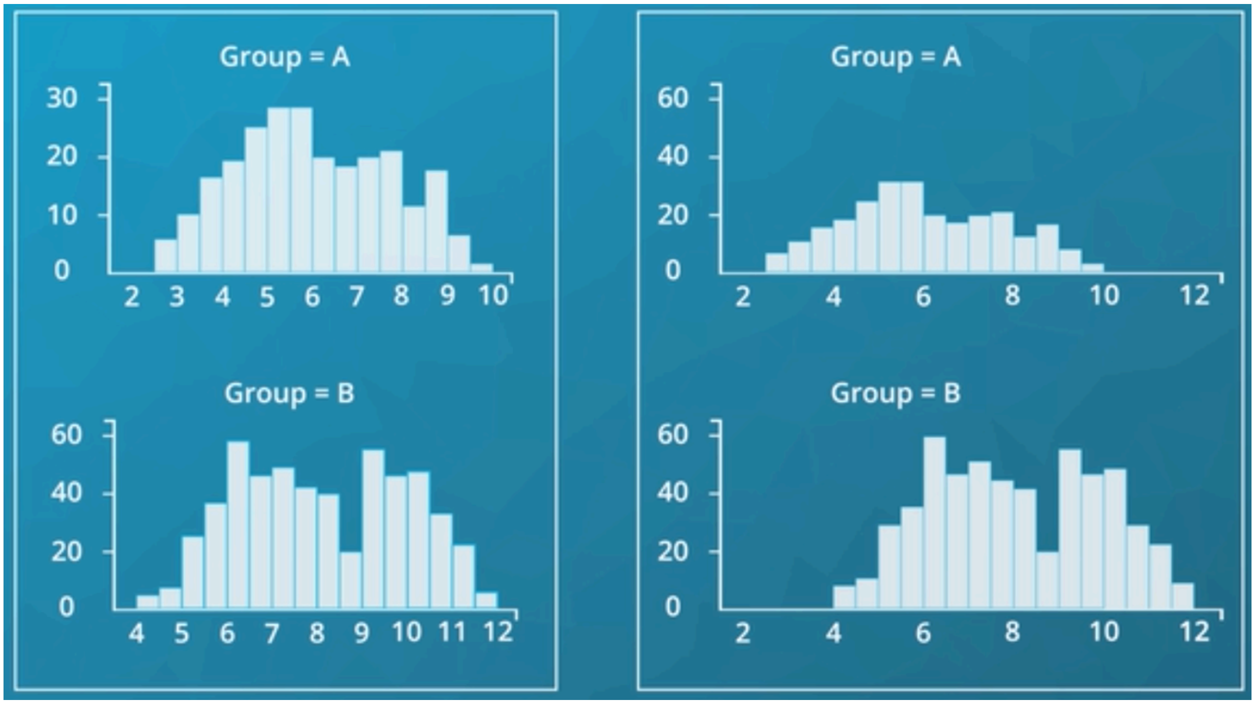
58.5 + 8.25 = 66.75 UPPER WHISKER BOUND

Clustered Bar Charts

To depict the relationship between two categorical variables, we can extend the univariate bar chart seen in the previous lesson into a clustered bar chart. Like a standard bar chart, we still want to depict the count of data points in each group, but each group is now a combination of labels on two variables. So we want to organize the bars into an order that makes the plot easy to interpret. In a clustered bar chart, bars are organized into clusters based on levels of the first variable, and then bars are ordered consistently across the second variable within each cluster.

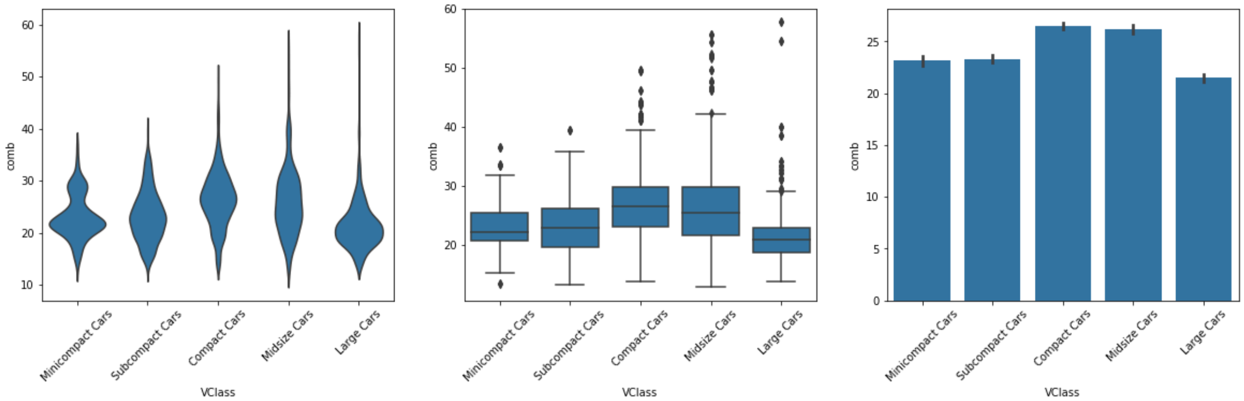
Faceting

One general visualization technique that will be useful for you to know about to handle plots of two or more variables is faceting. In faceting, the data is divided into disjoint subsets, most often by different levels of a categorical variable. For each of these subsets of the data, the same plot type is rendered on other variables. Faceting is a way of comparing distributions or relationships across levels of additional variables, especially when there are three or more variables of interest overall. While faceting is most useful in multivariate visualization, it is still valuable to introduce the technique here in our discussion of bivariate plots.



Adaptation of Univariate Plots

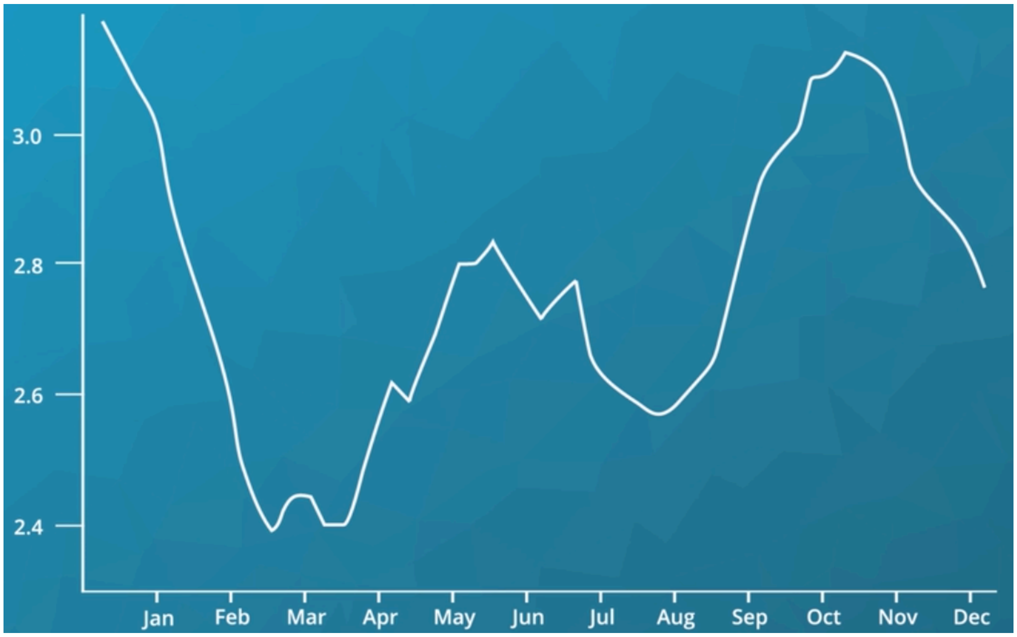
Histograms and bar charts were introduced in the previous lesson as depicting the distribution of numeric and categorical variables, respectively, with the height (or length) of bars indicating the number of data points that fell within each bar's range of values. These plots can be adapted for use as bivariate plots by, instead of indicating count by height, indicating a mean or other statistic on a second variable.



Line Plots

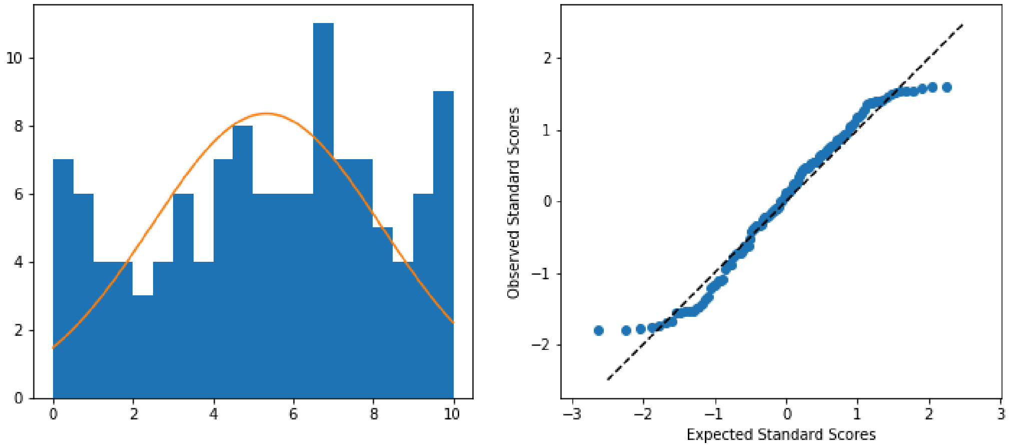
The line plot is a fairly common plot type that is used to plot the trend of one numeric variable against the values of a second variable. In contrast to a scatterplot, where all data points are plotted, in a line plot, only one point is plotted for every unique x-value or bin of x-values (like a histogram). If there are multiple observations in an x-bin, then the y-value of the point plotted in the line plot will be a summary statistic (like mean or median) of the data in the bin. The plotted points are connected with a line that emphasizes the sequential or connected nature of the x-values.

If the x-variable represents time, then a line plot of the data is frequently known as a time series plot. For example, we have only one observation per time period, like in stock or currency charts.



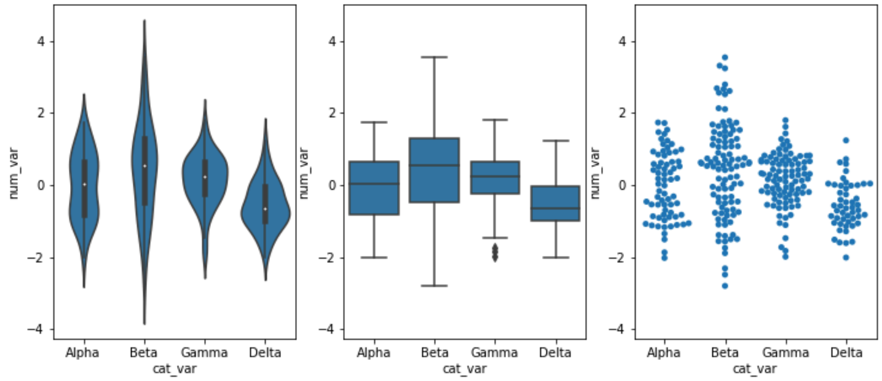
Q-Q Plots

There might be cases where you are interested to see how closely your numeric data follows some hypothetical distribution. This might be important for certain parametric statistical tests, like checking for assumptions of normality. In cases like this, you can use a quantile-quantile plot, or Q-Q plot, to make a visual comparison between your data and your reference distribution.



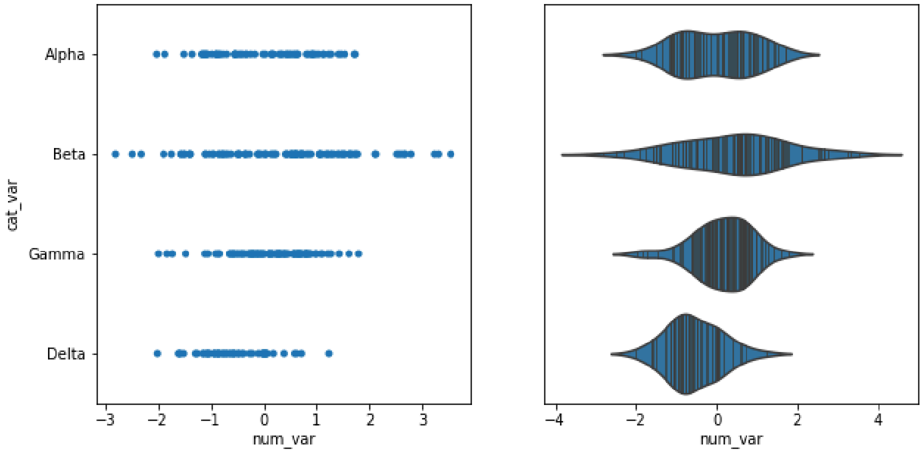
Swarm Plots

In this lesson, you saw many ways of depicting the relationship between a numeric variable and a categorical variable. Violin plots depicted distributions as density curves, while box plots took a more summary approach, plotting the quantiles as boxes with whiskers. Another alternative to these plots is the swarm plot. Similar to a scatterplot, each data point is plotted with position according to its value on the two variables being plotted. Instead of randomly jittering points as in a normal scatterplot, points are placed as close to their actual value as possible without allowing any overlap.



Rug and Strip Plots

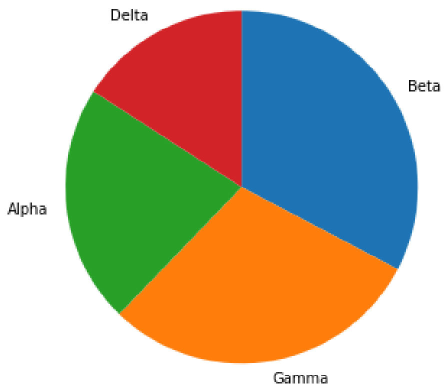
You might encounter, or be interested in, marginal distributions that are plotted alongside bivariate plots such as scatterplots. A marginal distribution is simply the univariate distribution of a variable, ignoring the values of any other variable. For quantitative data, histograms or density curves are fine choices for marginal plot, but you might also see the rug plot employed. In a rug plot, all of the data points are plotted on a single axis, one tick mark or line for each one. Compared to a marginal histogram, the rug plot suffers somewhat in terms of readability of the distribution, but it is more compact in its representation of the data.

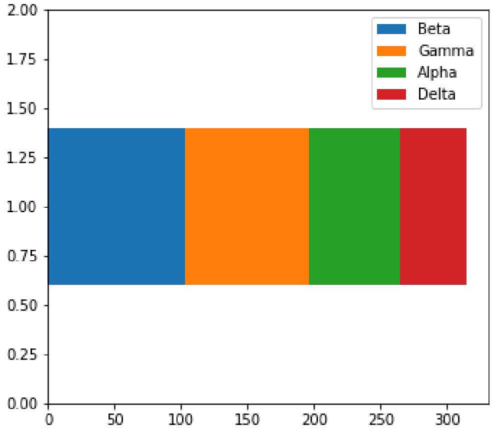


Stacked Plots

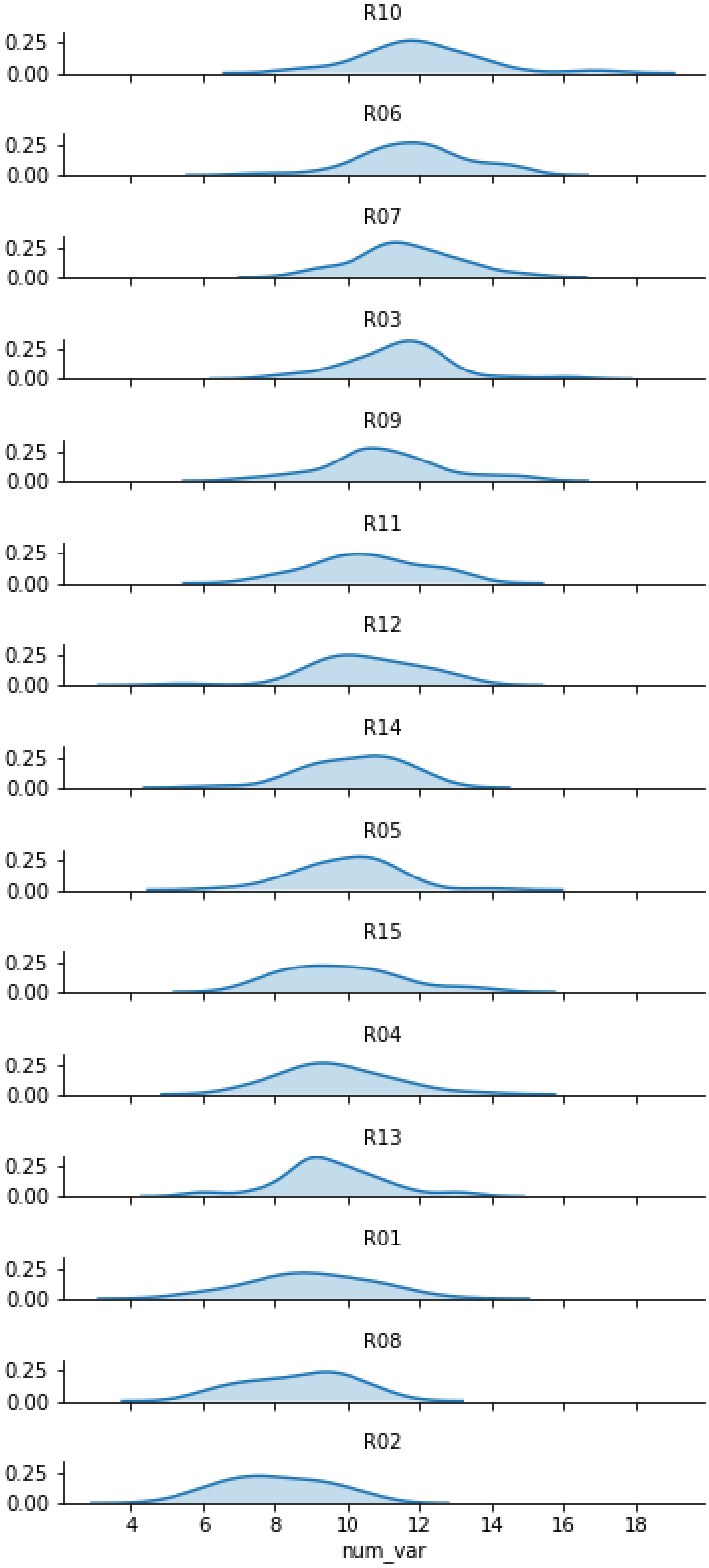
One common plotting technique has not been discussed thus far in the course, and that’s stacking. Stacked bar charts and histograms are not uncommon, but there are often better plot choices available.

The most basic stacked chart takes a single bar representing the full count, and divides it into colored segments based on frequencies on a categorical variable. If this sounds familiar, that's because it almost perfectly coincides with the description of a pie chart, except that the shape being divided is different.



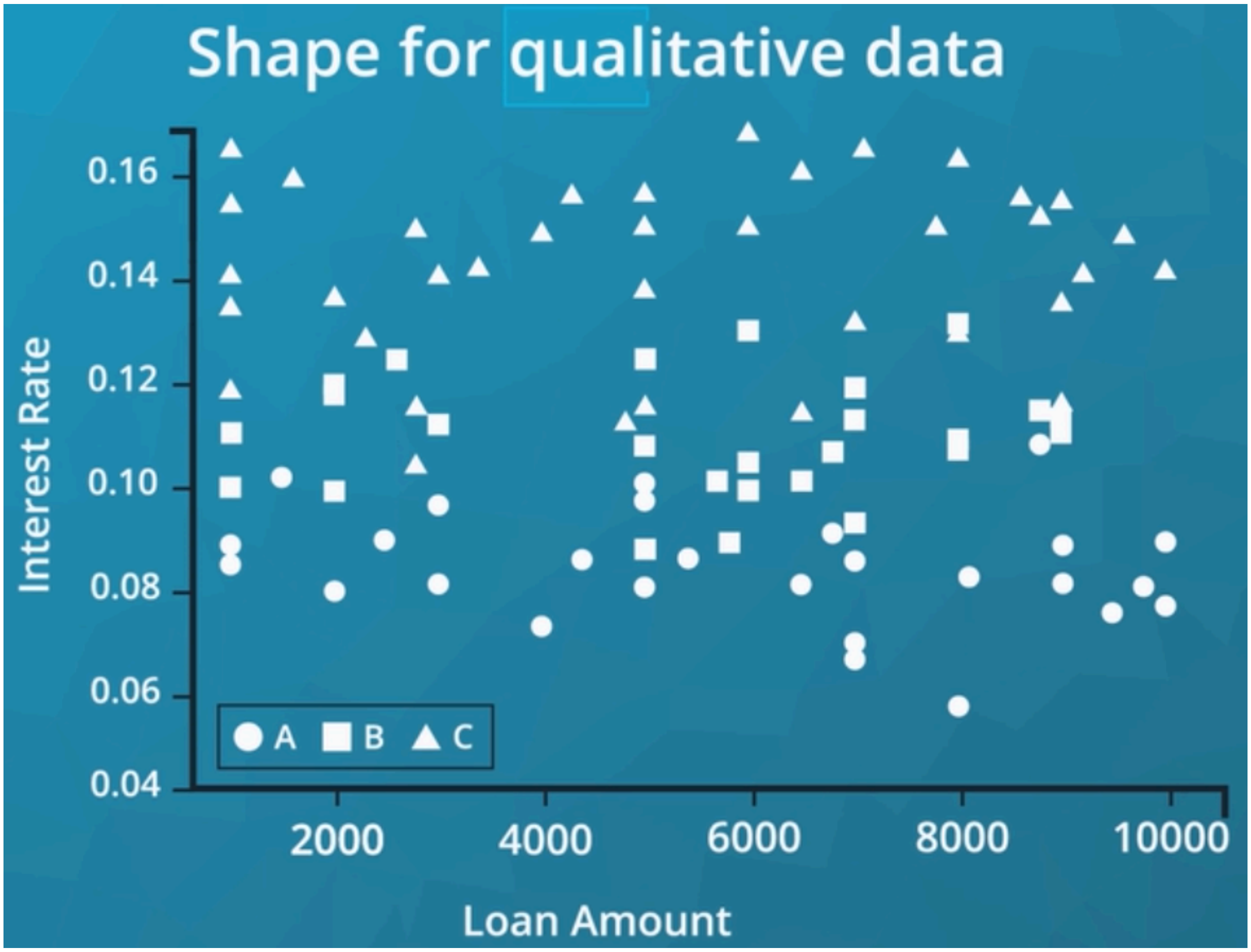


Ridgeline Plots



Multivariate Exploration of Data

Non-Positional Encodings for Third Variables



Non-Positional Encodings for Third Variables

There are four major cases to consider when we want to plot three variables together:

- Three numeric variables
- two numeric variables and one categorical variable
- one numeric variable and two categorical variables
- three categorical variables

A numerical variable is a variable where the value has meaning (e.g., weight or age), but a value such as a phone number doesn't have meaning in the numbers alone. A categorical variable is a variable that holds a type (e.g., species or hair color).

If we have at least two numeric variables, as in the first two cases, one common method for depicting the data is by using a scatterplot to encode two of the numeric variables, then using a non-positional encoding on the points to convey the value on the third variable, whether numeric or categorical. (You will see additional techniques later in the lesson that can also be applied to the other two cases, i.e., where we have at least two categorical variables.)

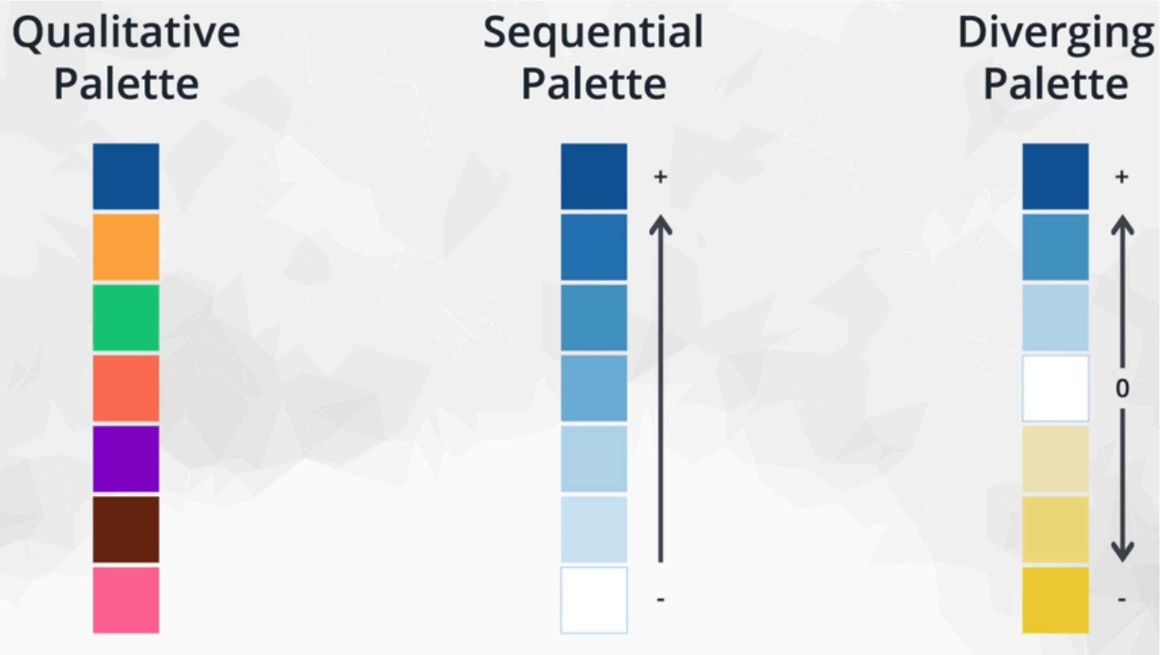
Three main non-positional encodings stand out: marker color, marker shape, marker size

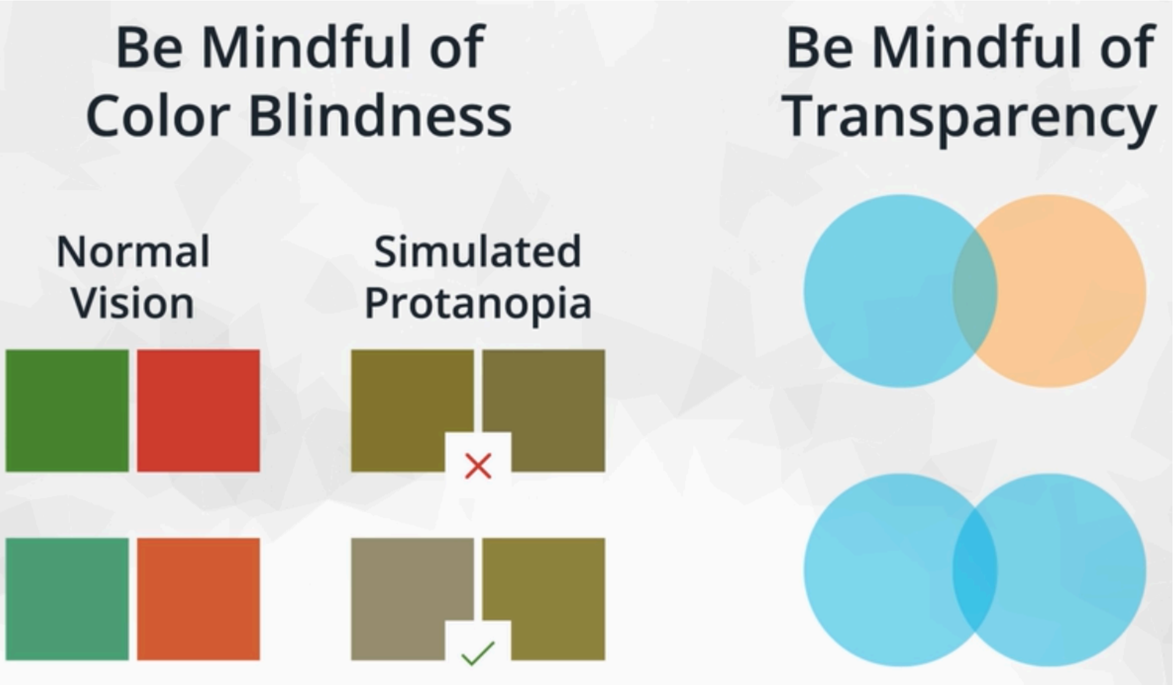
Color Palettes

Color is a very common encoding for variables, for both qualitative and quantitative variables. You've already seen this employed in previous lessons where position could not be used to encode a value:

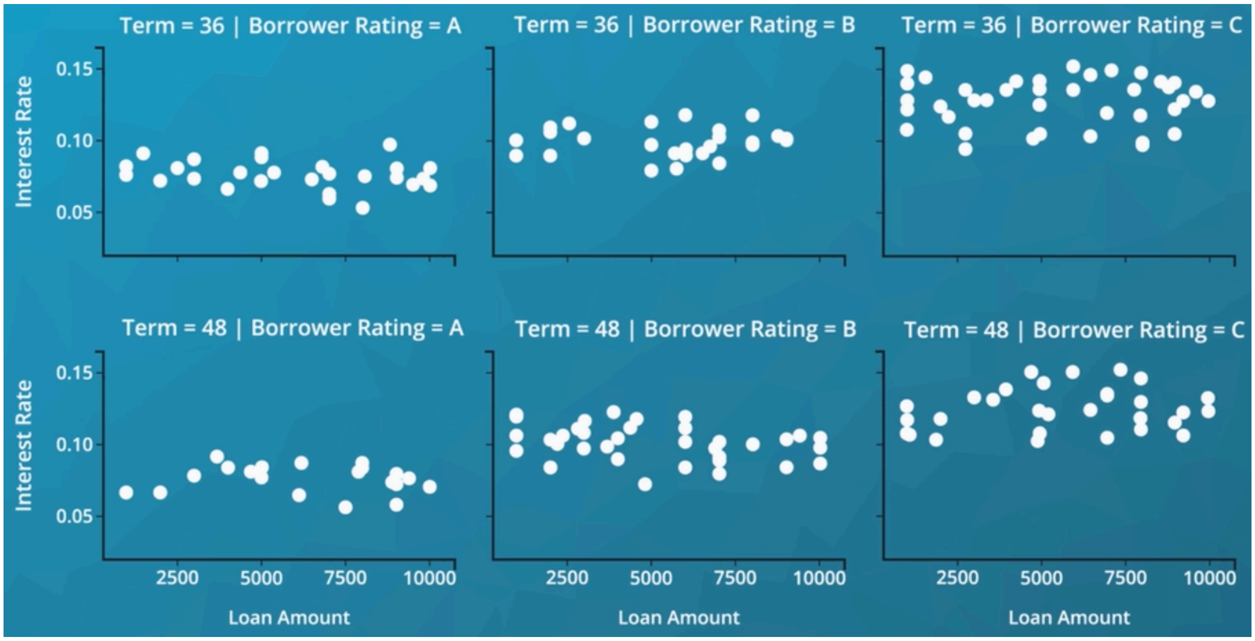
- color for category in a clustered bar chart
- color for count in a heat map (both as a 2-d histogram and as a 2-d bar chart)

Here, we'll look at how to employ color in scatterplots, as well as discuss more about color palette choices depending on the type of data you have.



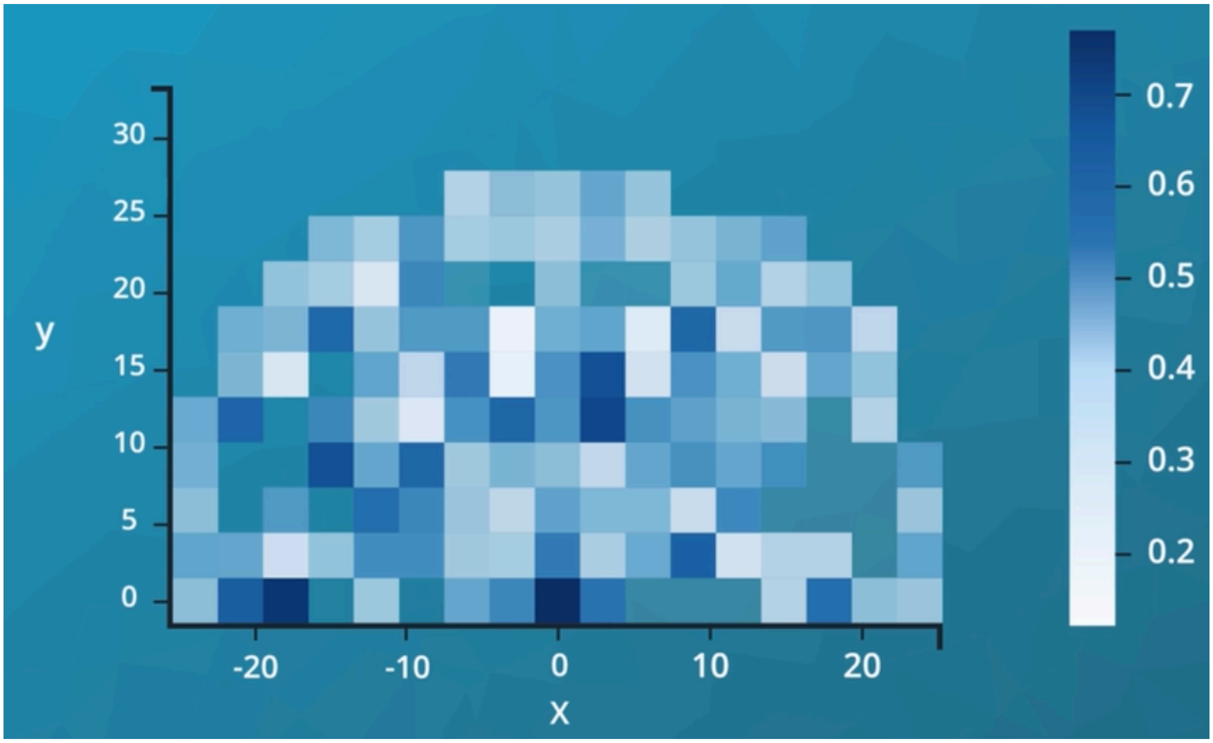


Faceting in Two Directions



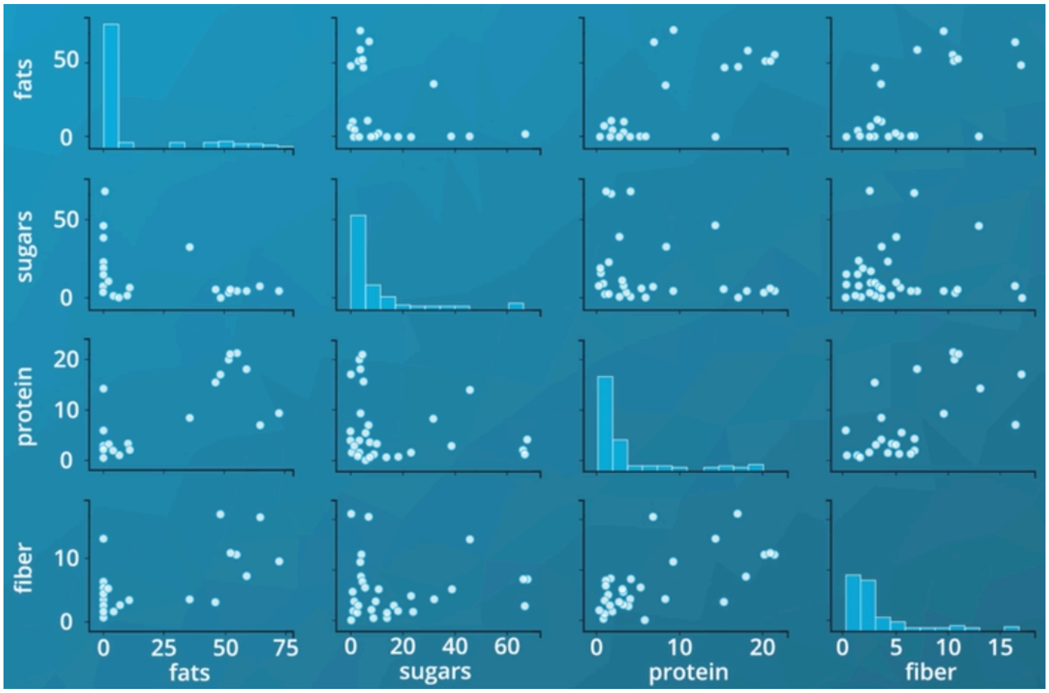
Other Adaptations of Bivariate Plots

You also saw one other way of expanding univariate plots into bivariate plots in the previous lesson: substituting count on a bar chart or histogram for the mean, median, or some other statistic of a second variable. This adaptation can also be done for bivariate plots like the heat map, clustered bar chart, and line plot, to allow them to depict multivariate relationships.



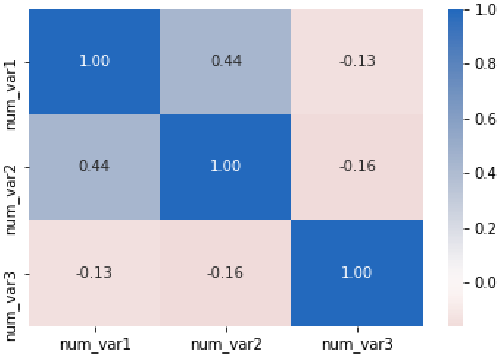
Plot Matrices

To move back to bivariate exploration for a bit, you might come out of your initial univariate investigation of the data wanting to look at the relationship between many pairs of variables. Rather than generate these bivariate plots one by one, a preliminary option you might consider for exploration is the creation of a plot matrix. In a plot matrix, a matrix of plots is generated. Each row and column represents a different variable, and a subplot against those variables is generated in each plot matrix cell. This contrasts with faceting, where rows and columns will subset the data, and the same variables are depicted in each subplot.



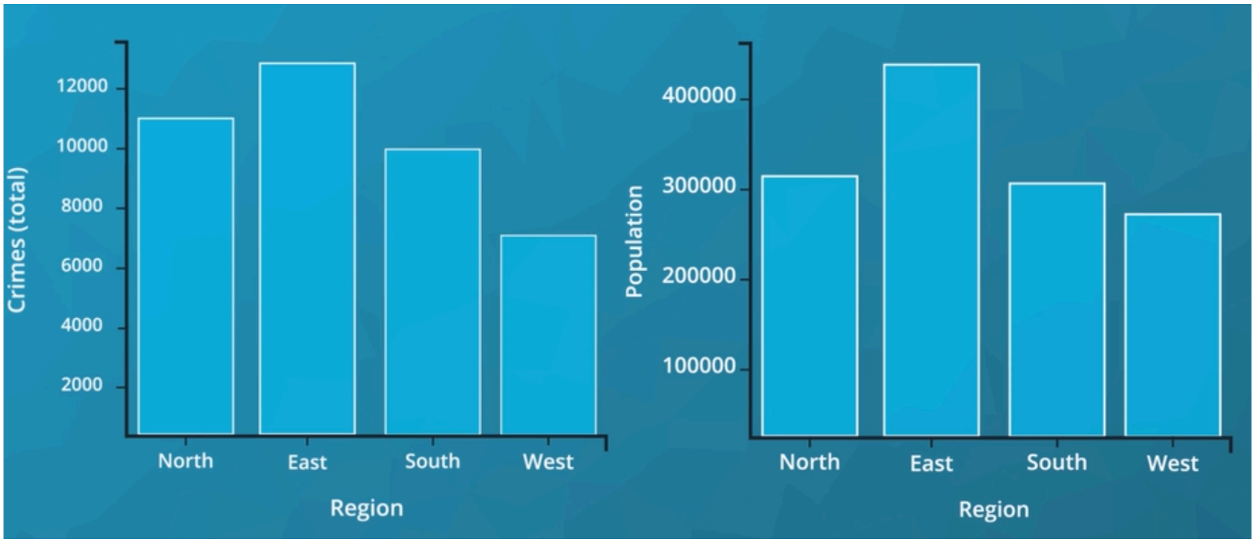
Correlation Matrices

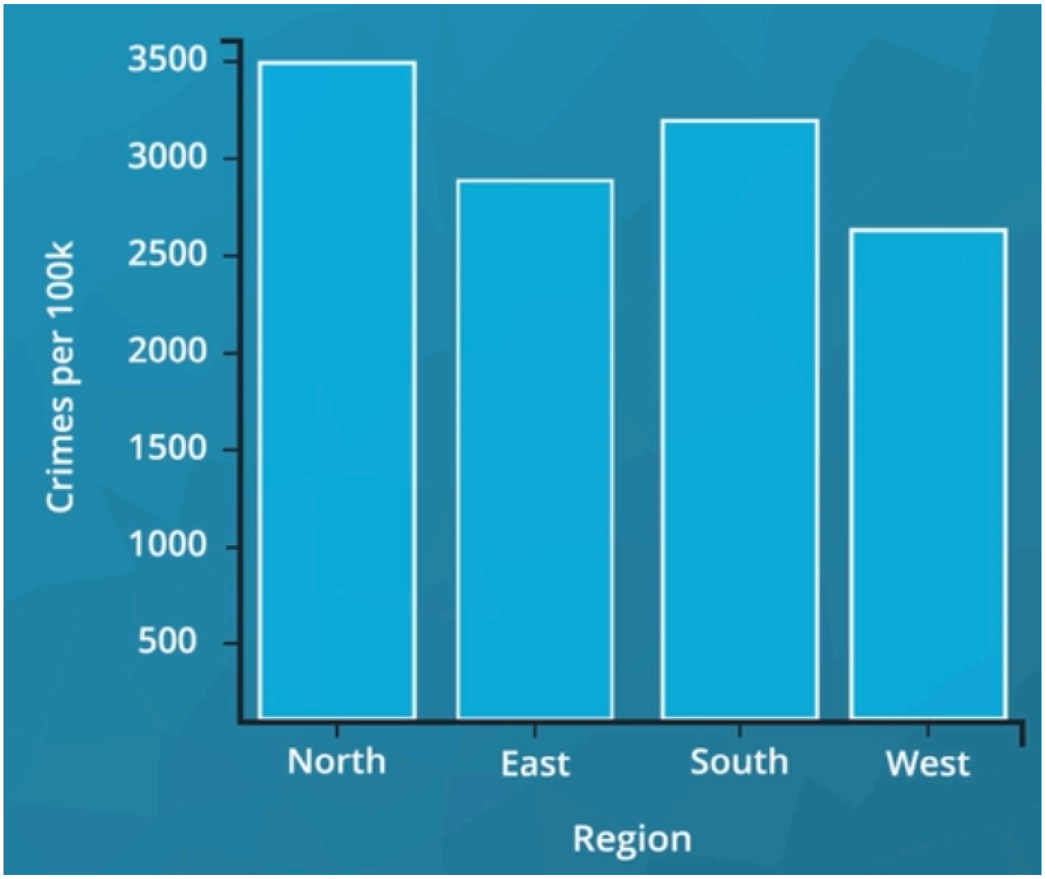
For numeric variables, it can be useful to create a correlation matrix as part of your exploration. While it's true that the Panda's .corr function is perfectly fine for computing and returning a matrix of correlation coefficients, it's not too much trouble to plot the matrix as a heat map to make it easier to see the strength of the relationships.



Feature Engineering

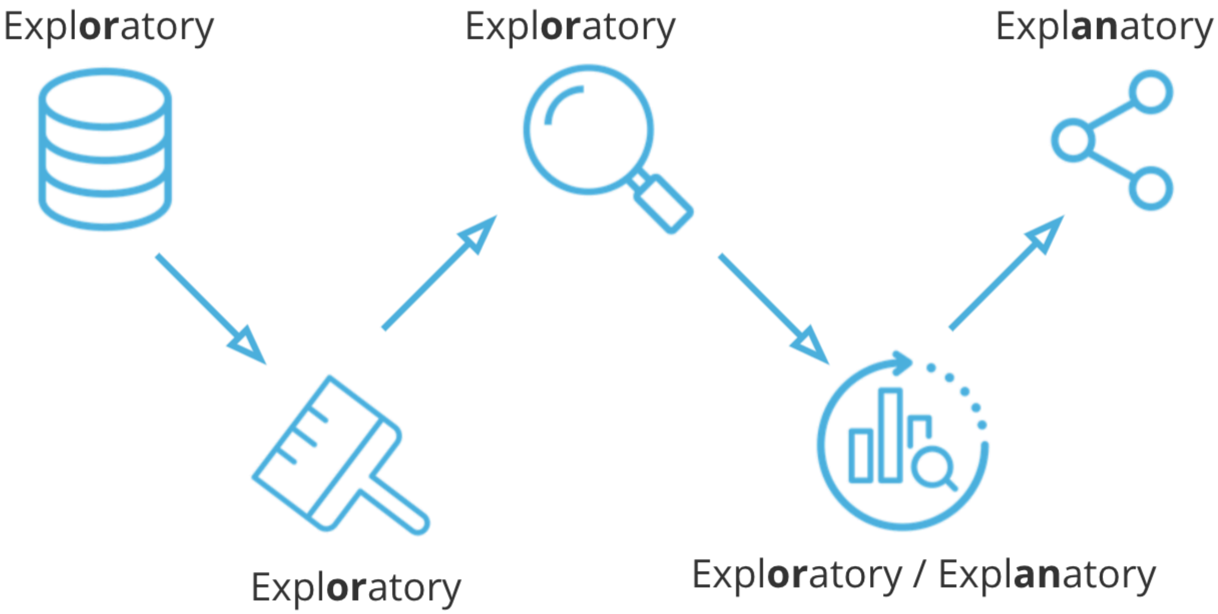
This is not so much an additional technique for adding variables to your plot, but a reminder that feature engineering is a tool that you can leverage as you explore and learn about your data. As you explore a dataset, you might find that two variables are related in some way. Feature engineering is all about creating a new variable with a sum, difference, product, or ratio between those original variables that may lend a better insight into the research questions you seek to answer.





Explanatory Visualizations

Revisiting the Data Analysis Process



The previous three lessons in the course have been focused on exploratory analyses. In phases with exploratory visualizations, the primary audience for the visuals will be you, the analyst. The plots that have been created and demonstrated haven't been particularly polished, just descriptive enough for you to gain insights into the data.

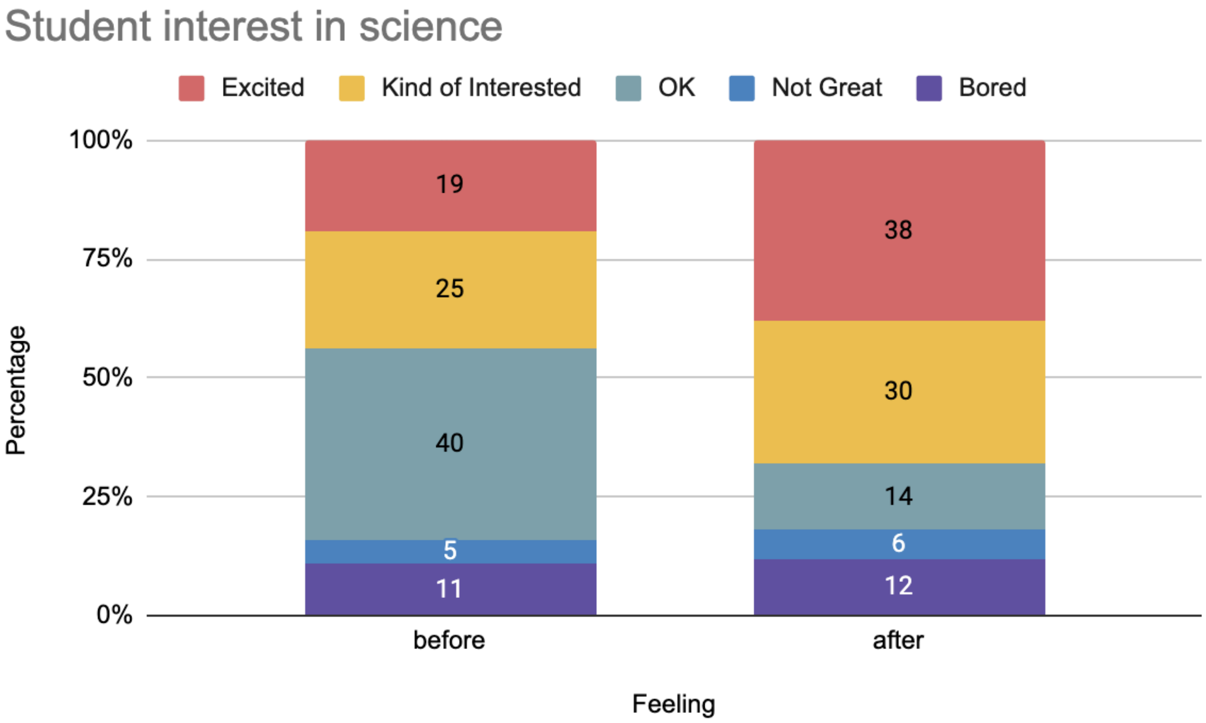
This lesson is focused on taking those insights and creating explanatory analyses. Here, your audience will be broader: your goal will be to convey your findings to other people who don't have the level of hands-on experience with the data as you. Visualizations under this banner should be focused on telling a specific story that you want to convey to that particular audience. Many times, these visualizations evolve from visuals created during the exploratory process, just polished up to highlight the specific intended insights. These highlights might change depending on the audience you're presenting to. You'll revisit those design concepts from earlier in the course to make your plots informative not just for yourself, but also compelling and understandable for others.

Tell A Story

Telling stories with data follows these steps:

- Start with a Question
- Repetition is a Good Thing
- Highlight the Answer
- Call Your Audience To Action

The question I want to answer in this practice is that "Do more students feel science is interesting after attending the science camp?" To answer this question, I built a stacked bar chart that highlights the comparison of the percentage in student's feelings before and after the science camp.



As you can see, although a similar percentage of students who feel science is not great or boring, it's very clear that there is a large increase in the percentage of students who feel science is interesting or exciting.
So my conclusion is that we should have more science camps because they will increase student's interest in science.

Polishing Plots

Choose an appropriate plot
Your choice of plot will depend on the number of variables that you have and their types: nominal, ordinal, discrete numeric, or continuous. Choice of plot also depends on the specific relationship that you want to convey. For example, whether you choose a violin plot, box plot, or adapted bar chart depends on how much data you have and whether distributions are significant or important. You'll be more likely to use a violin plot if you have a lot of data and the distributions are meaningful, and more inclined to use a box plot or bar chart if you have less data, or the distributions are less reliable.

Choose appropriate encodings
Your variables should impact not just the type of plot that is chosen, but also the variable encodings. For example, if you have three numeric variables, you shouldn't just assign x-position, y-position, and color encodings randomly. In many cases, the two variables that are

most important should take the positional encodings; if one represents an outcome or dependent variable, then it should be plotted on the y-axis. In other cases, it makes sense to plot the dependent measure with color, as though you are taking a top-down view of the plane defined by the two independent measures plotted on the axes.

Pay attention to design integrity

When setting up your plotting parameters, remember the design principles from earlier in the course.

Make sure that you minimize chart junk and maximize the data-ink ratio, as far as it maintains good interpretability of the data. When deciding whether or not to add non-positional encodings, make sure that they are meaningful. For example, using color in a frequency bar chart may not be necessary on its own, but will be useful if those colors are used again later in the same presentation, matched with their original groups. By the same token, avoid using the same color scheme for different variables to minimize the chance of reader confusion.

You should also ensure that your plot avoids lie factors as much as possible. If you have a bar chart or histogram, it is best to anchor them to a 0 baseline. If you're employing a scale transformation, signal this clearly in the title, axis labels, and tick marks.

Label axes and choose appropriate tick marks

- For your positional axes, make sure you include axis labels. This is less important in exploration when you have the code available and have an extended flow to your work, but when you're conveying only the key pieces to others, it's crucial. When you add an axis label, make sure you also provide the units of measurement, if applicable (e.g., stating "Height (cm)" rather than just "Height").
- As for tick marks, you should include at least three tick marks on each axis. This is especially important for data that has been transformed: you want enough tick marks so that the scale of the data can be communicated there. If your values are very large or very small numbers, consider using abbreviations to relabel the ticks (e.g., use "250K" instead of "250000").

Provide legends for non-positional variables

Make sure that you add a legend for variables not depicted on the axes of your plot. For color encoding, you can add a color bar to the side of the plot. The most important new thing here is that you provide a descriptive label to your legend or color bar, just as you would the axes of your plot.

Title your plot and include descriptive comments

Finally, make sure that you provide a descriptive title to your plot. If this is a key plot that presents important findings to others, aim to create a title that draws attention to those main points, rather than just state what variables are plotted.

Also, realize that while a visualization might be the core mechanism by which you convey findings, it need not stand alone. Comments in the text below or surrounding the plot can provide valuable context to help the reader understand your message, or reinforce the main points that they should have gotten.