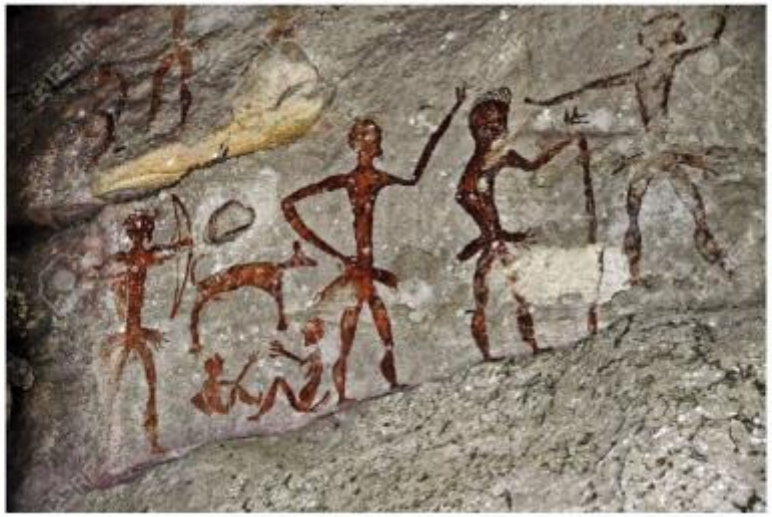
Introduction to Data Manipulation & Visualization

Visualization Objectives

- Record information
- Analyze data to support reasoning
- Confirm hypotheses
- Communicate ideas to others

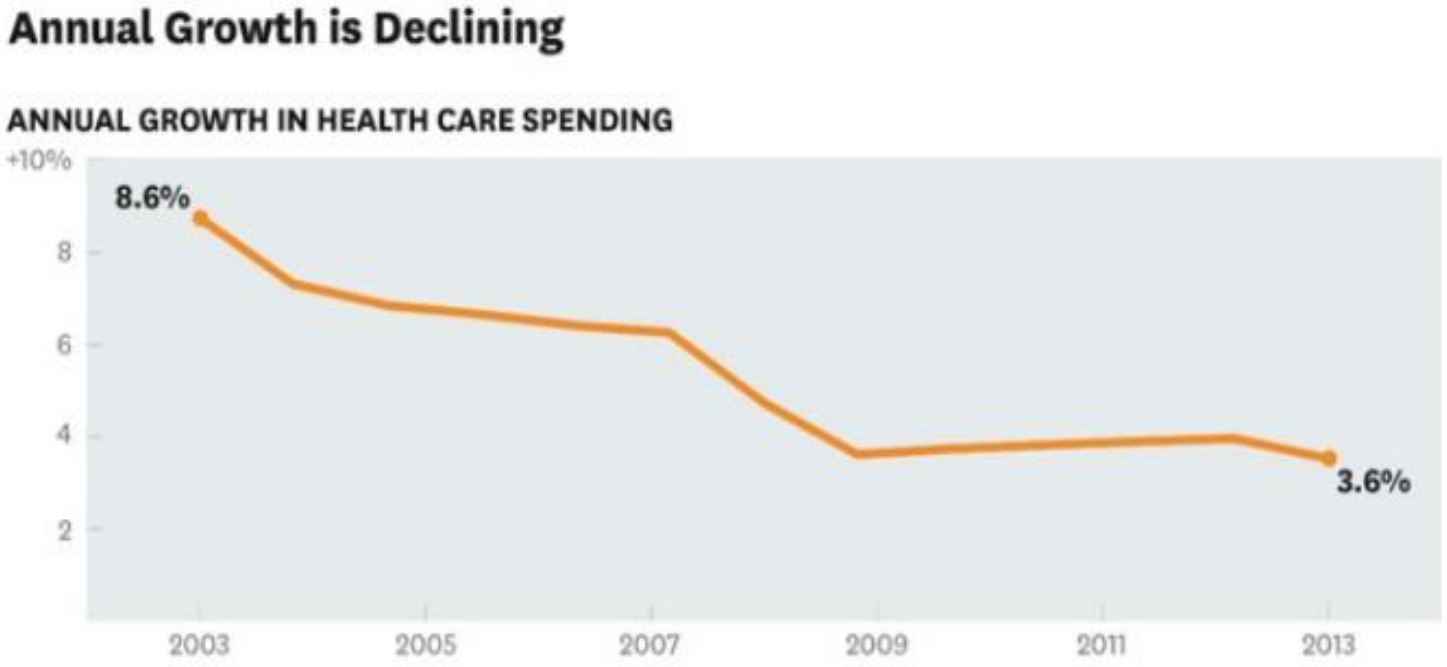
Why Visualize?

To record information



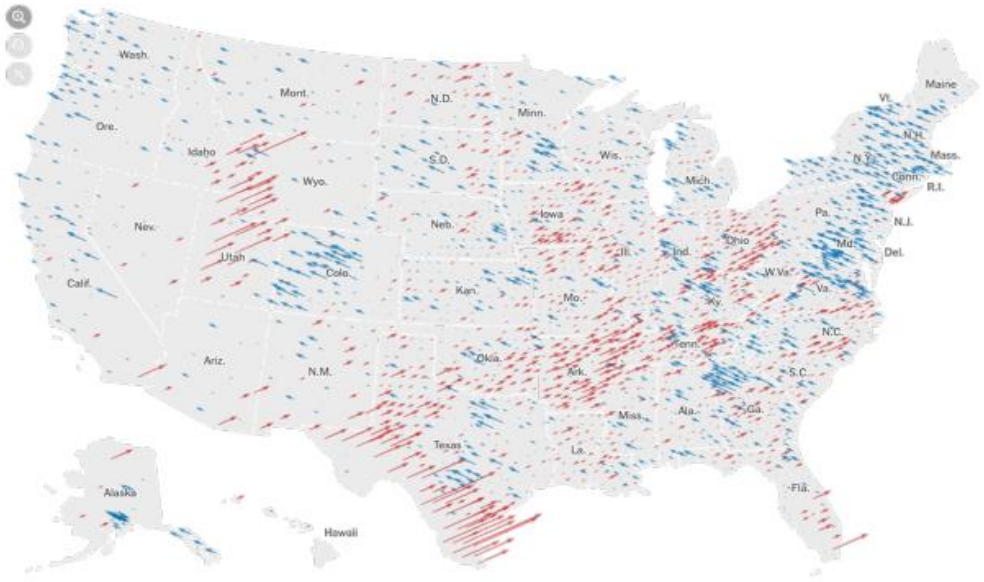
Why Visualize?

To communicate information



Why Visualize?

To analyze data



2020 US Elections (NYTimes)

Types of Plots

- Line plots
- Bar plots
- Scatter plots
- Box plots
- Histograms

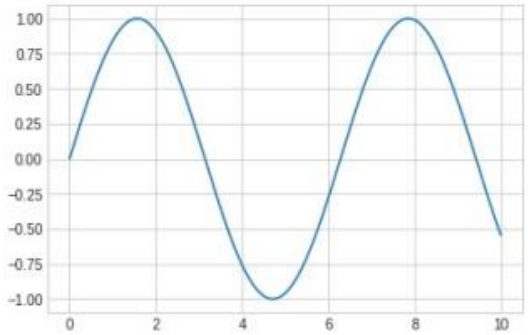
What are line plots?

Two types of relational plots:

1) **Scatter plots**
   - Each plot point is an independent observation

2) **Line plots**
   - Each plot point represents the same "thing", typically tracked over time

Line plot



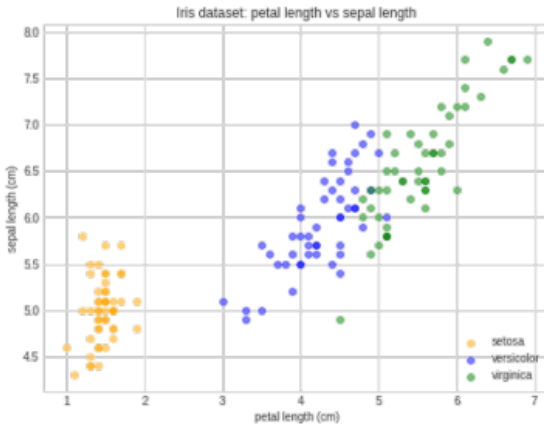
- Used for numeric data
- **Used to show trends**
- Compare two or more different variables over time
- Could be used to make predictions

View trends in data over time.

Examples: Stock price change over a five-year period or website page views during a month.

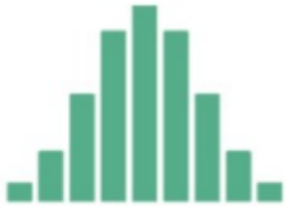
Scatter plots

- Investigate relationships between quantitative values.
  - Used to visualize relation between two numeric variables
  - Used to visualize correlation in a large data set



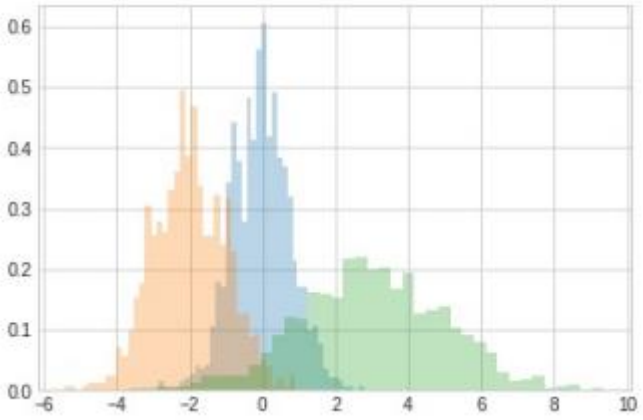
Histograms

- Understand the distribution of your data.
  - Displays the frequency distribution (shape)
- Summarize large data sets graphically
- Compare multiple distributions



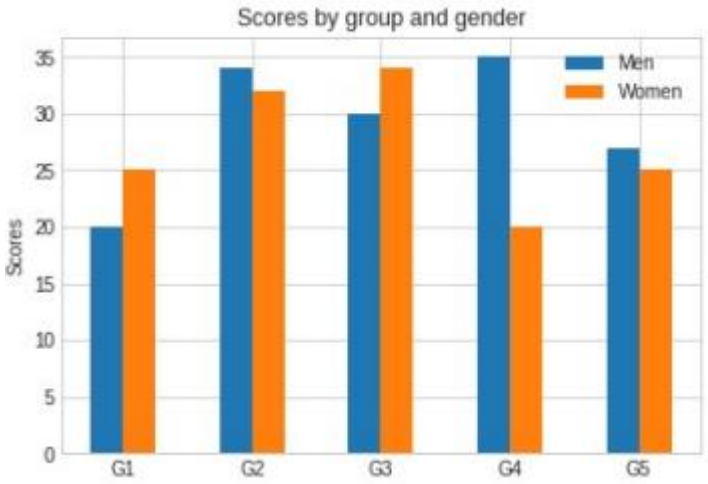
Examples:

- Number of customers by company size,
- Student performance on an exam,
- Frequency of a product defect.



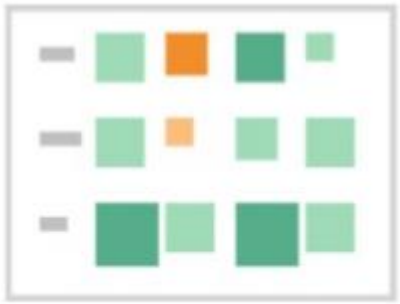
Bar plots

Compare data amongst different categories



Examples: percent of spending by department.

Heat Map



— Show the relationship between at least two factors.

Data Visualization

Part 1

Basic Visuals | Matplotlib, Seaborn

Basic Visualization Concepts, Introduction and Comparison b/t Matplotlib and Seaborn Python Libraries in Jupyter Notebook.

Part 2

Interactive Visuals | Plotly, Bokeh, Tableau, etc.

Deeper insights into more interactive and fun data visualization functions. Introduction to Plotly, Bokeh and Tableau.

Data Visualization-cont'd

What is data visualization?
Data visualization is the graphical representation of information and data.

What makes for effective data visualization?
Visualization **transforms data into images effectively and accurately** represent information about the data.

What are the advantages of data visualization?
Makes for easier **interpretation of patterns and trends** as opposed to looking at data in a tabular/spreadsheet format.

About Data Visualization

What Would You Like to Show?

- Relationships between variables
- Composition of the data over time
- Distribution of variable(s) in data
- Comparison of data with relation to time, variables, categories, etc.

Now, what you'll learn

- How do you choose an appropriate plot?
- How do you interpret common types of plots?
- What are best practices for drawing plots?

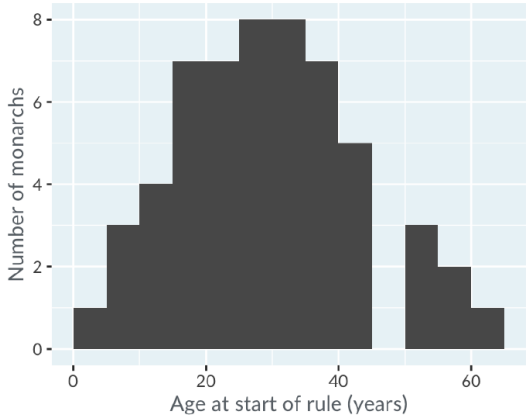
Continuous and categorical variables

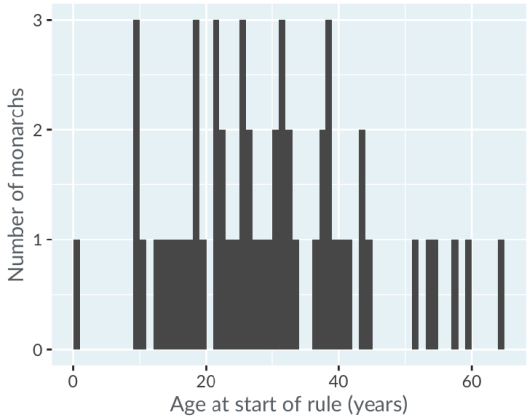
- Continuous: usually numbers
  - heights, temperatures, revenues
- Categorical: usually text
  - eye colors, countries, industry
- Can be either
  - age is continuous, but age group is categorical
  - time is continuous, month of year is categorical

Histograms

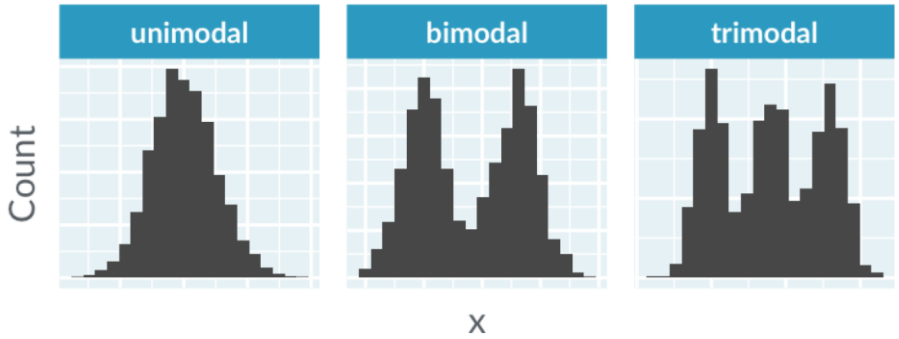
When should you use a histogram?

1) If you have continuous variable(s).
2) You want to ask questions about the shape of its distribution.

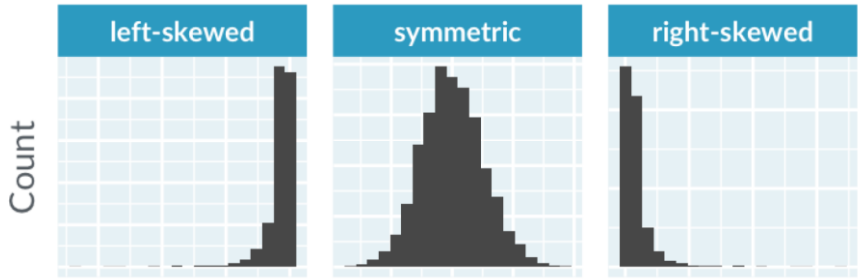




Modality: how many peaks?



Skewness: is it symmetric?

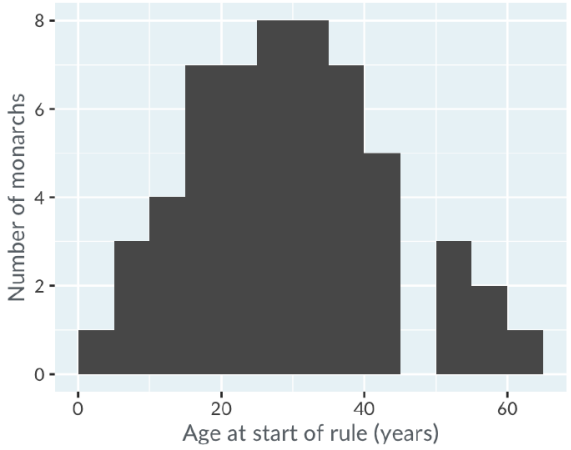


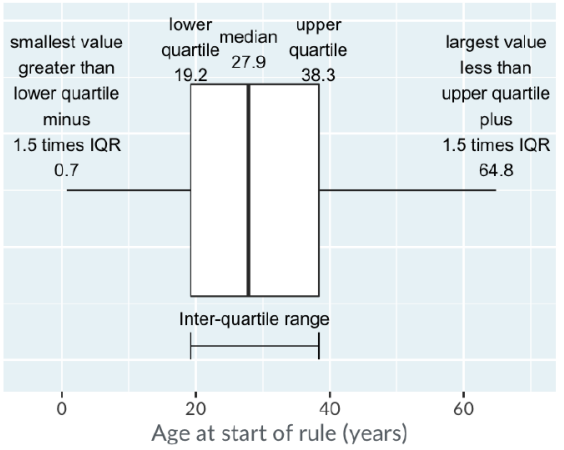
Box plots

When should you use a box plot?

1) When you have a continuous variable, split by a categorical variable.
2) When you want to compare the distributions of the continuous variable for each category.

Histogram vs. box plot

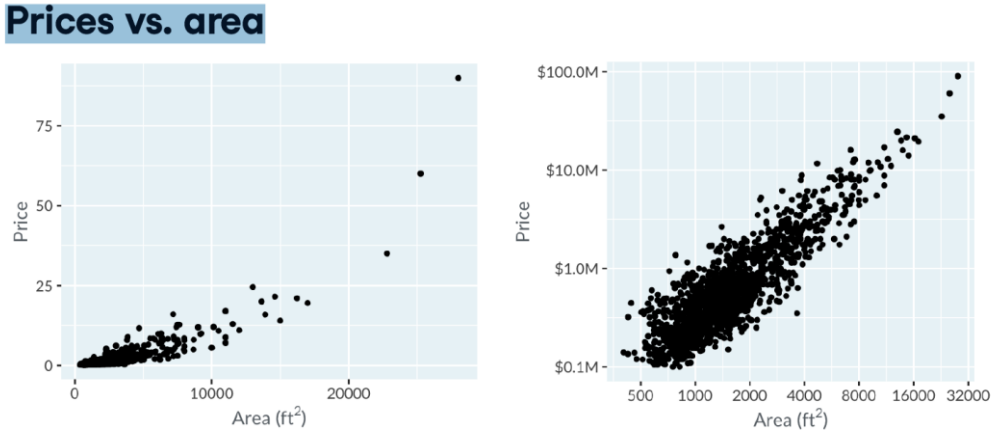




Scatter plots

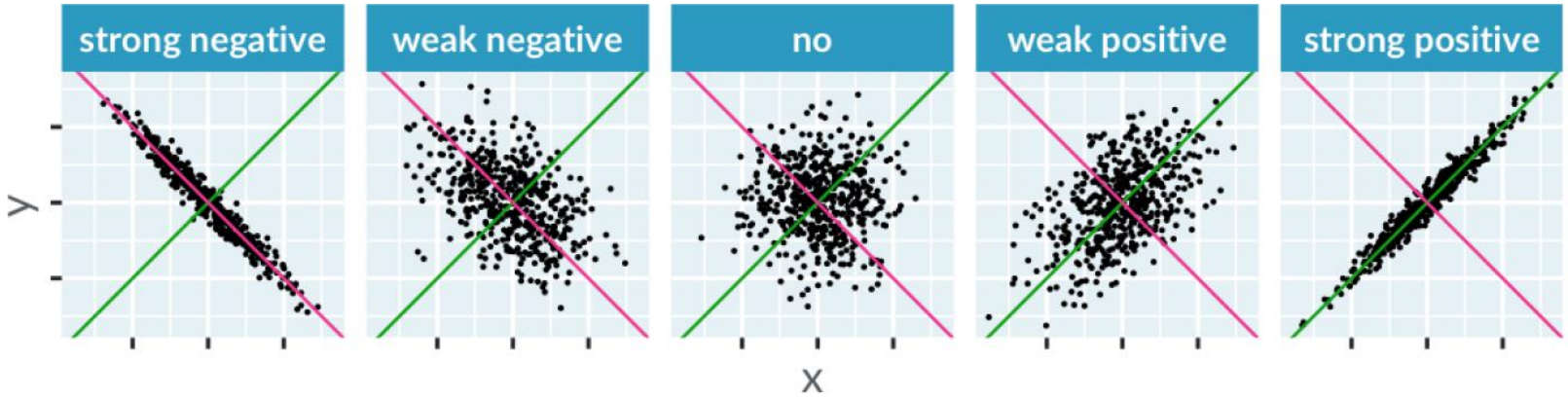
When should you use a scatter plot?

1) You have two continuous variables.
2) You want to answer questions about the relationship between the two variables.

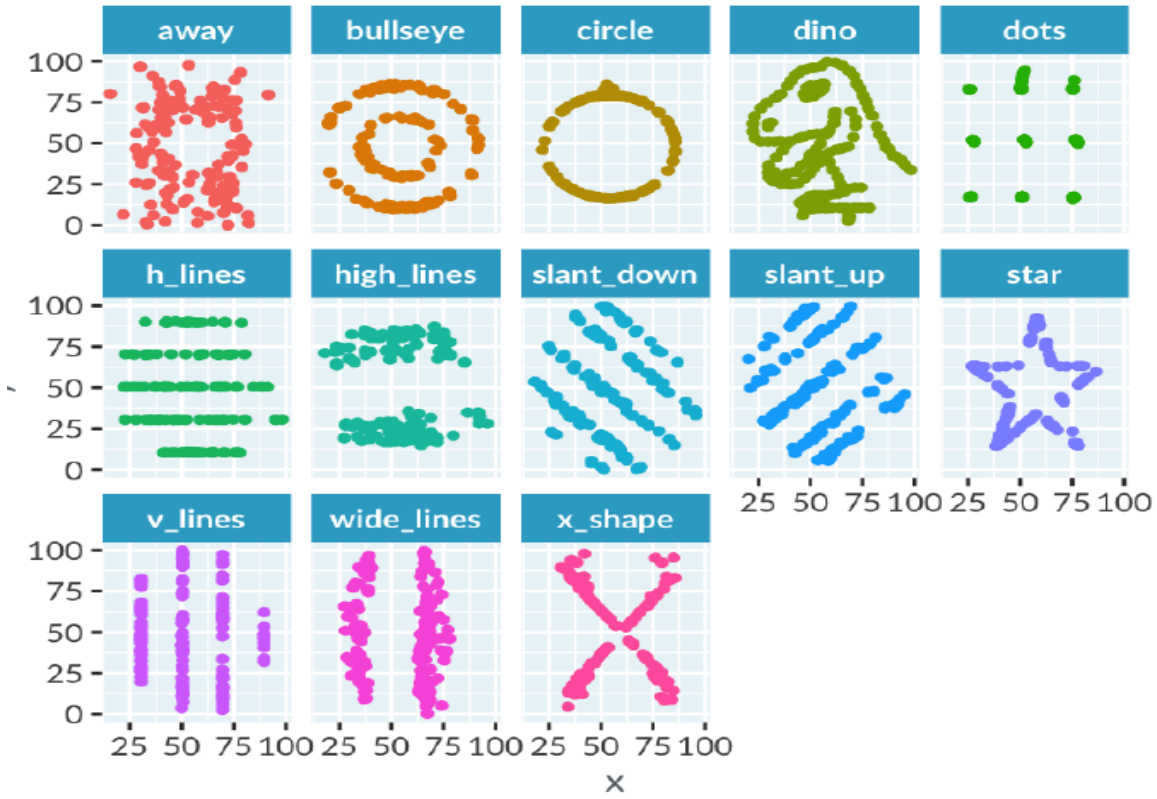


Correlation

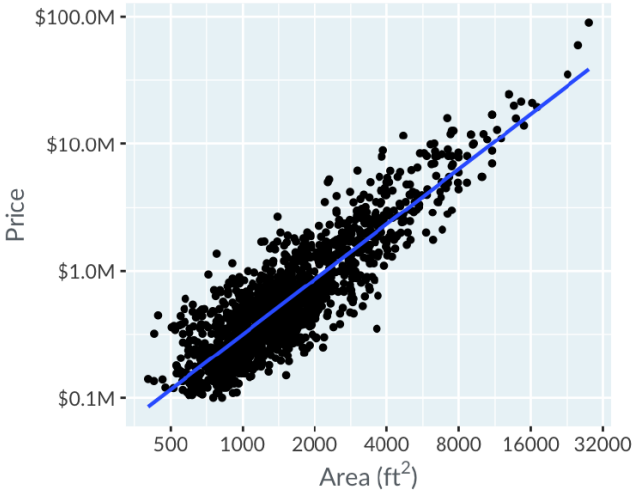
Straight line through the points?



Sometimes correlation isn't helpful



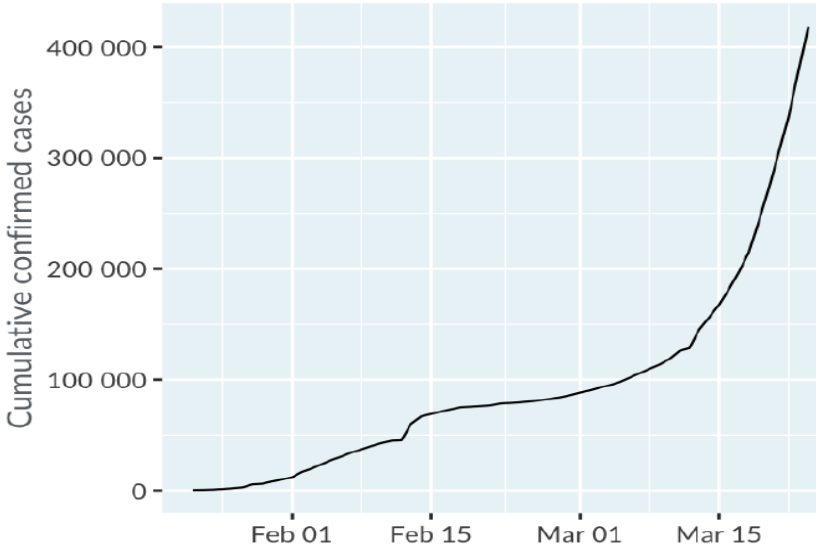
Adding trend lines



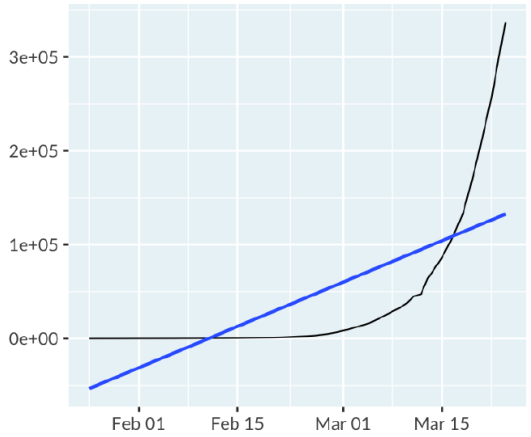
Line plots

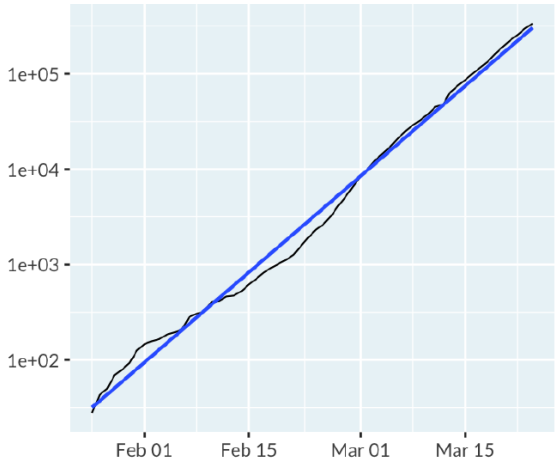
When should you use a line plot?

1. You have two continuous variables.
2. Consecutive observations are connected somehow. Usually, but not always, the x-axis is dates or times.



Trend lines + log scale



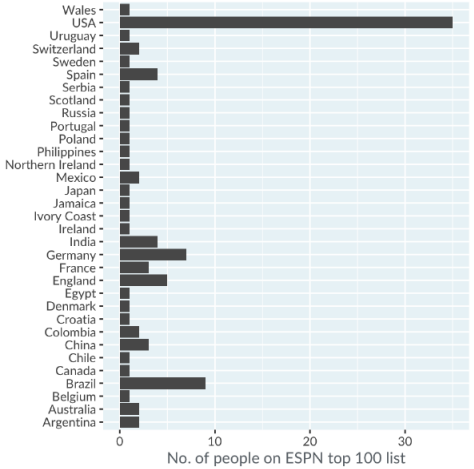


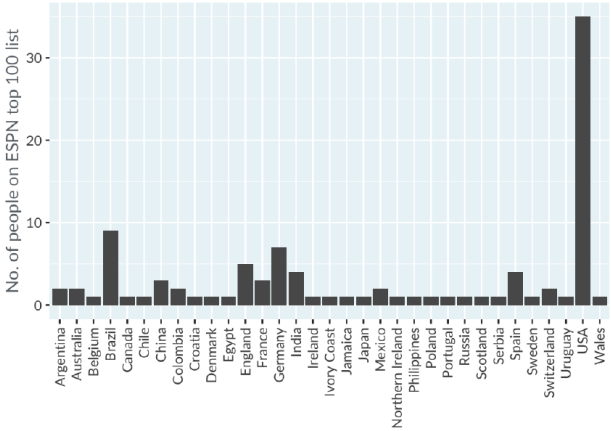
Bar plots:

When should you use a bar plot?

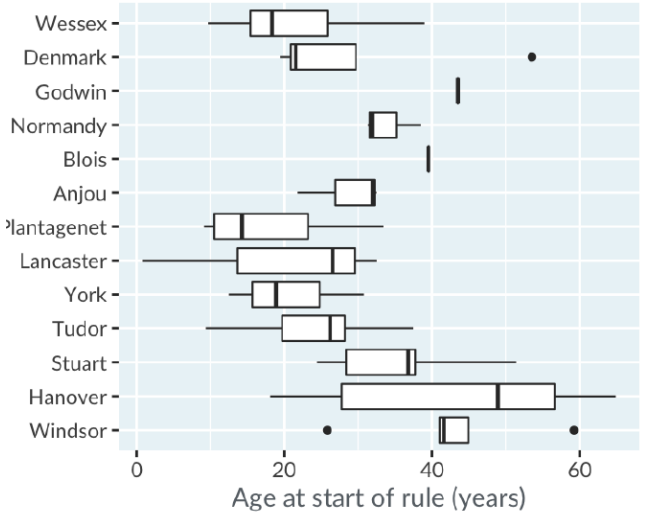
Most common cases:

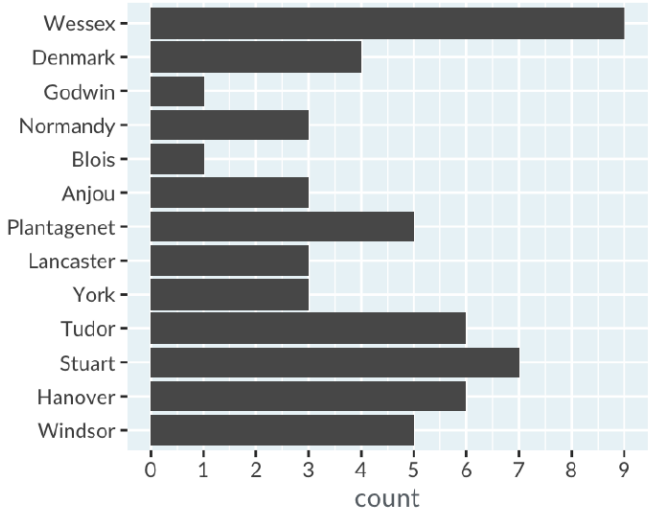
1) You have a categorical variable.
2) You want counts or percentages for each category.





Bar plots vs. box plots

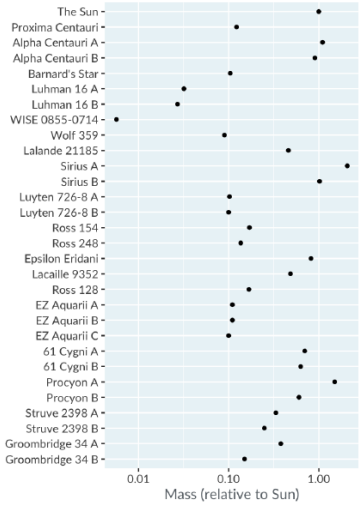




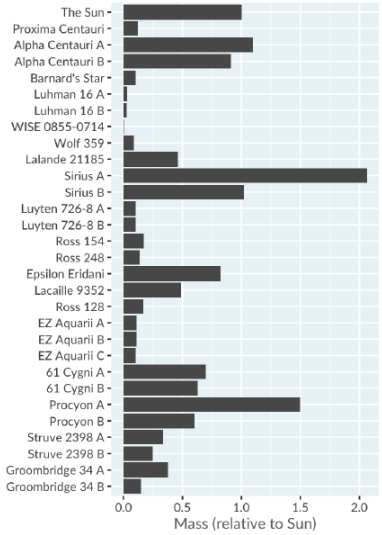
Dot plots

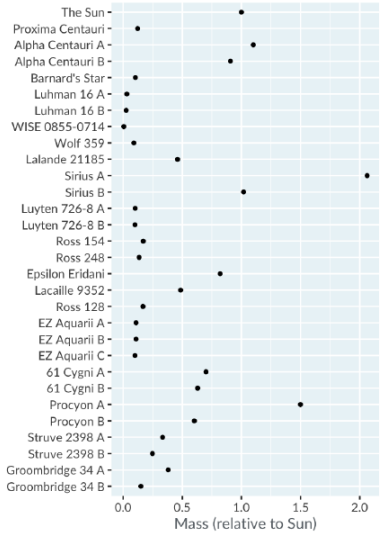
When should you use a dot plot?

1) You have a categorical variable.
2) You want to display numeric scores for each category on a log scale, or
3) You want to display multiple numeric scores for each category

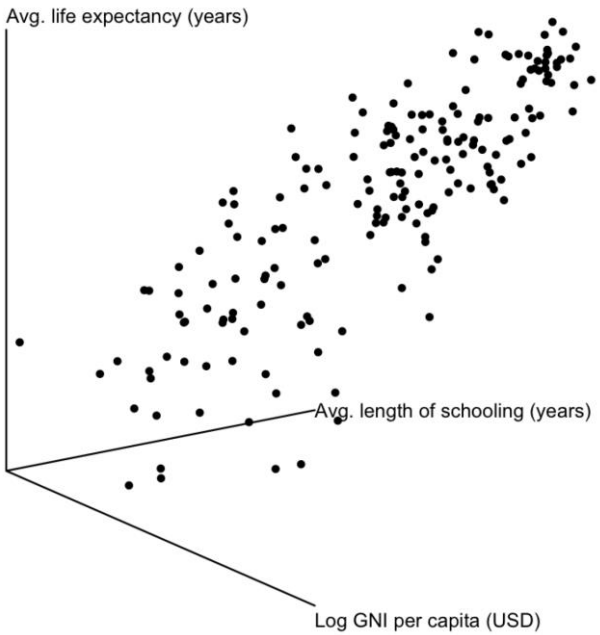


Bar plot vs. dot plot





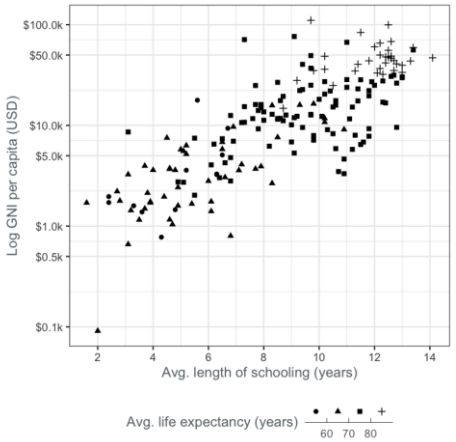
Higher dimensions: 3D



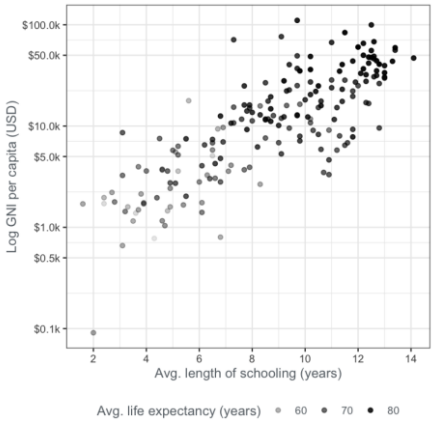
3D scatter plots

x and y are not the only dimensions

- Points also have these dimensions
  - color
  - size
  - transparency
  - shape

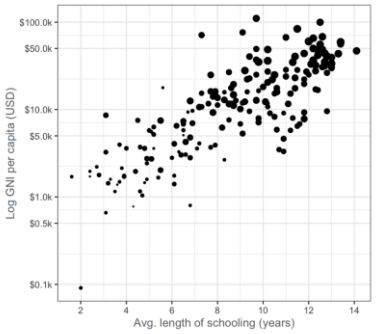


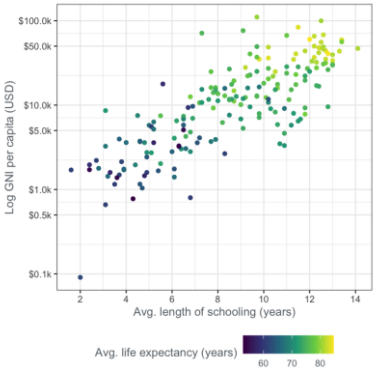
Shape



Transparency

Size

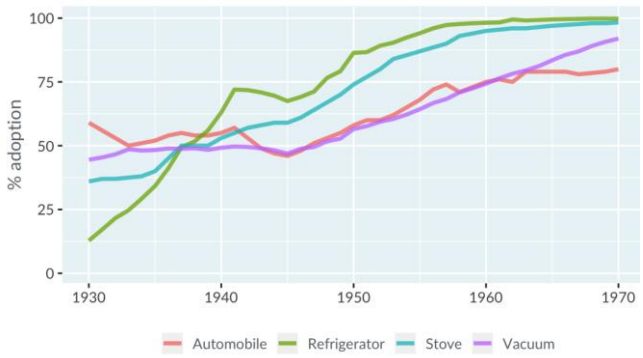




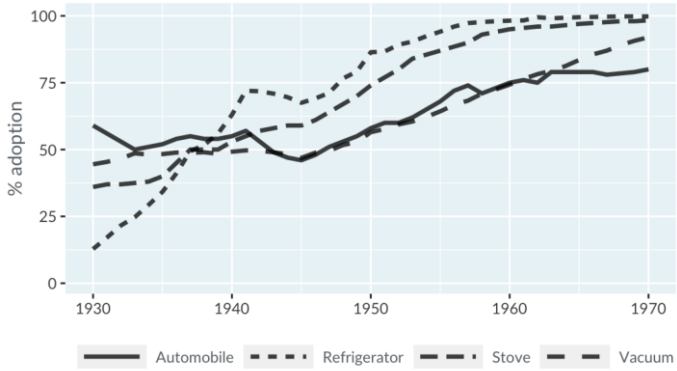
Color

Other dimensions for line plots

- color
- thickness
- Transparency
- Line type (solid, dashes, dots)



Color

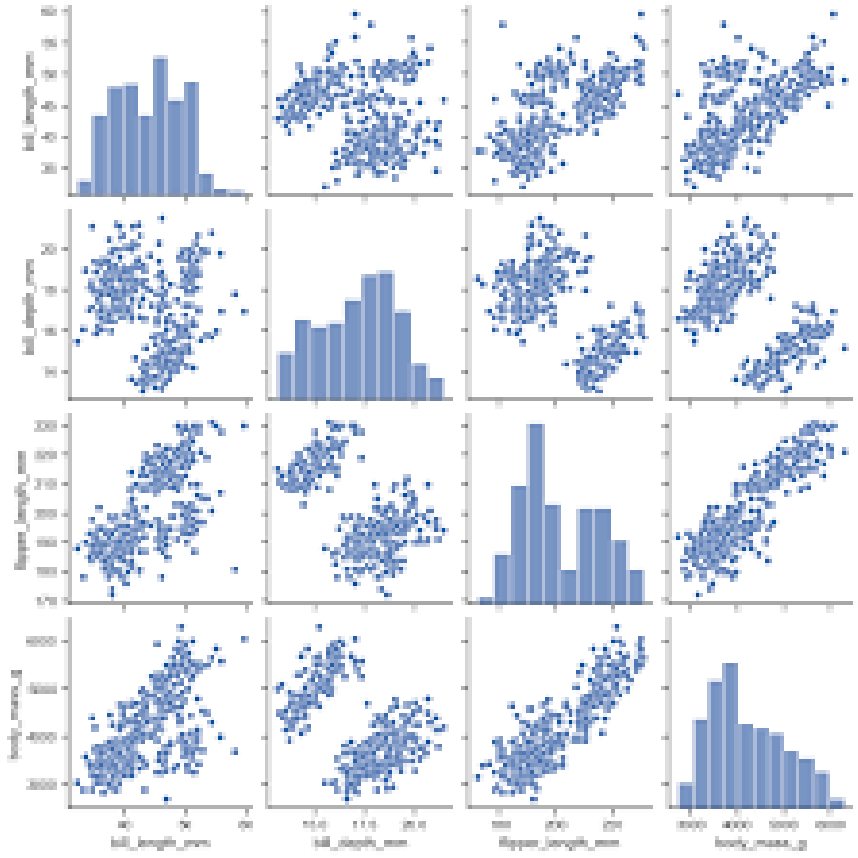


Linetype

Plotting many variables at once

When should you use a pair plot?

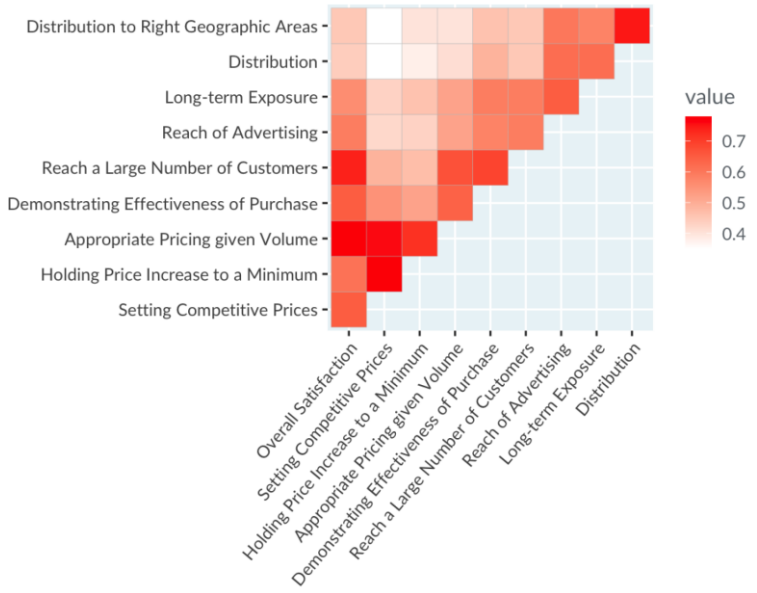
- You have a set of variables (either continuous, categorical, or a mix).
- You want to see the distribution for each variable.
- You want to see the relationship between each pair of variables.



Correlation heatmap

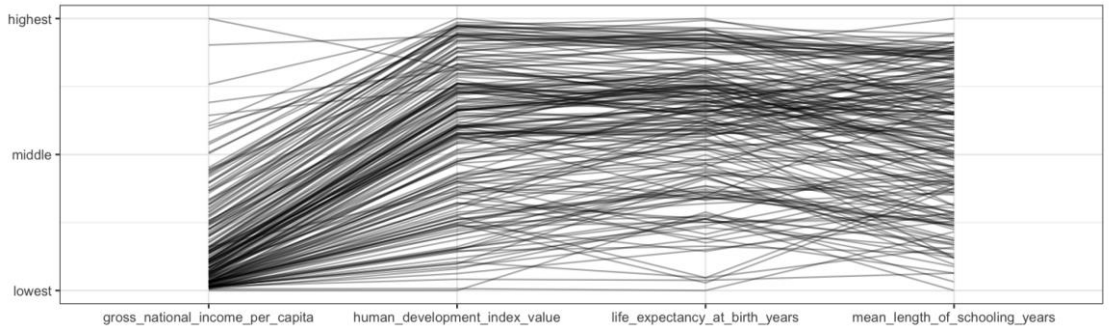
When should you use a correlation heatmap?

- You have lots of **continuous variables**.
- You want to a simple overview of how each pair of variables is related

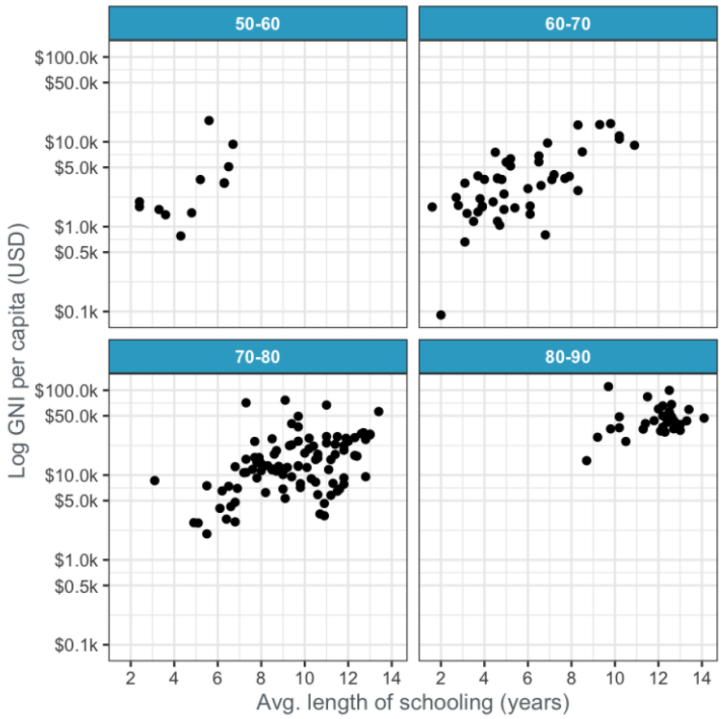


When should you use a parallel coordinates plot?

- You have lots of continuous variables.
- You want to find patterns across these variables, or
- You want to visualize clusters of observations.



Lots of panels

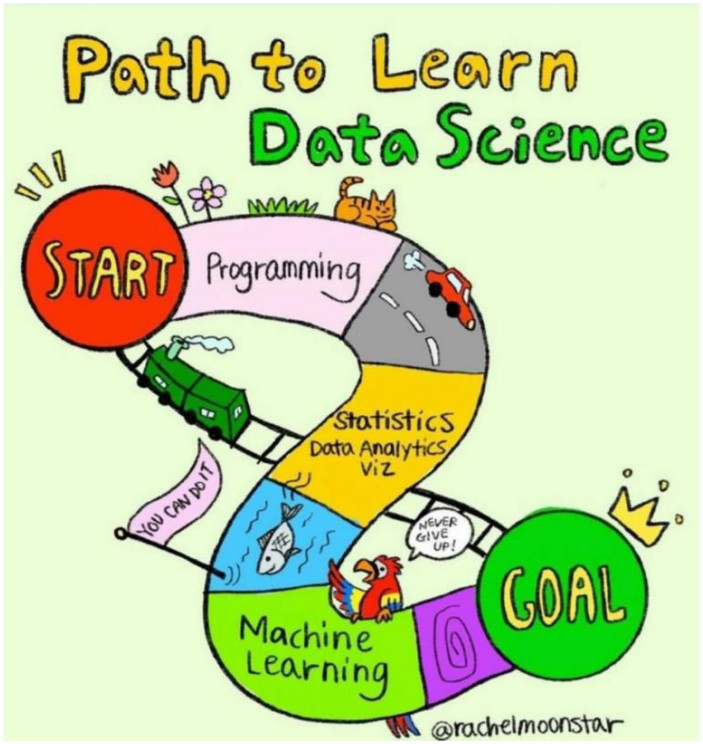


Summary

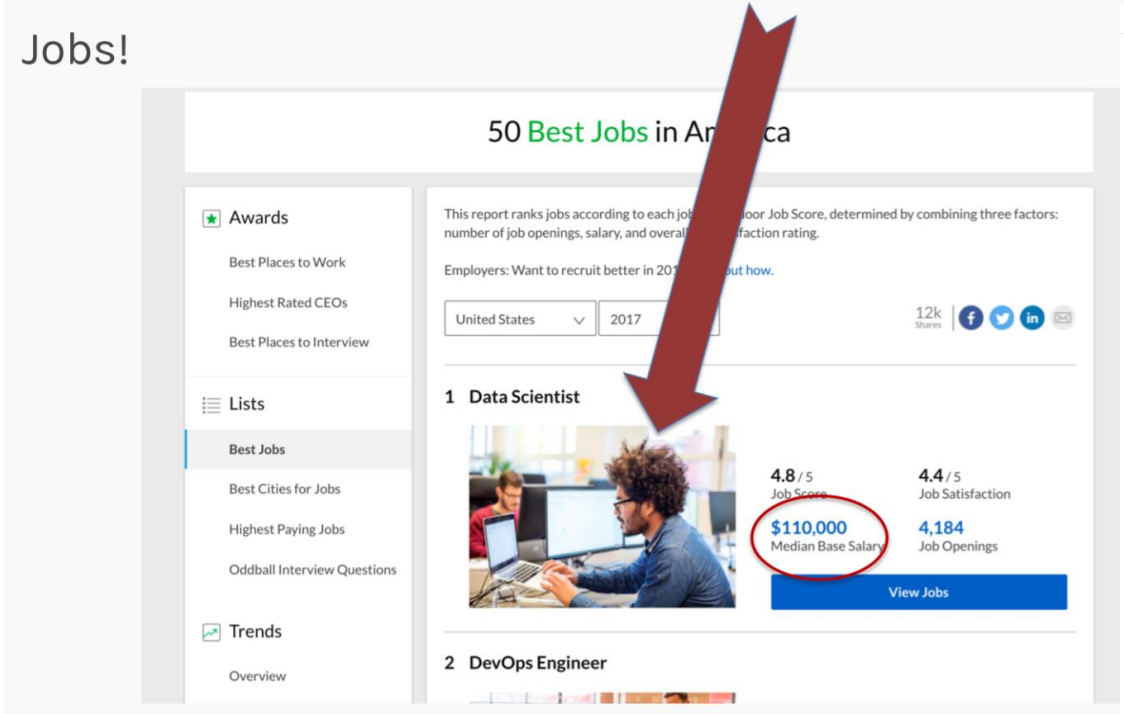
- Histograms: show a distribution
- Scatter plots: compare two numeric variables
- Line plots: show trends over time
- Bar plots: show counts by category
- Pair plot: compare many variables
- Correlation heatmap: show related variables
- Parallel coordinates plot: find patterns across variables

DATA MANIPULATION & VISUALIZATION

Data Science



Jobs!



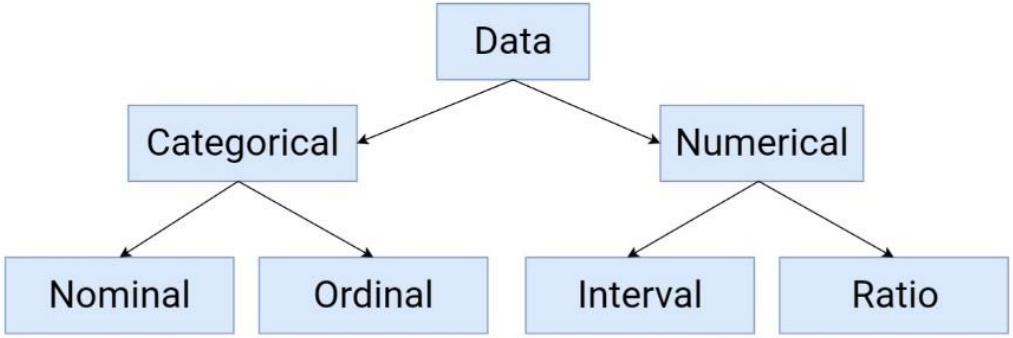
Data versus Information

- Data: Any observation collected in respect of any characteristic or event is called data.
- Information
  - Raw data carry/convey little meaning, when it is considered alone.
  - The data is minimized: processed/analyzed and then presented systematically.
    - It is converted into Information.
- Data, that is not converted into information is of little value for evaluation and planning and cannot be used by those who are involved in decision making.

Data Classification

Classification data can be divided into two types

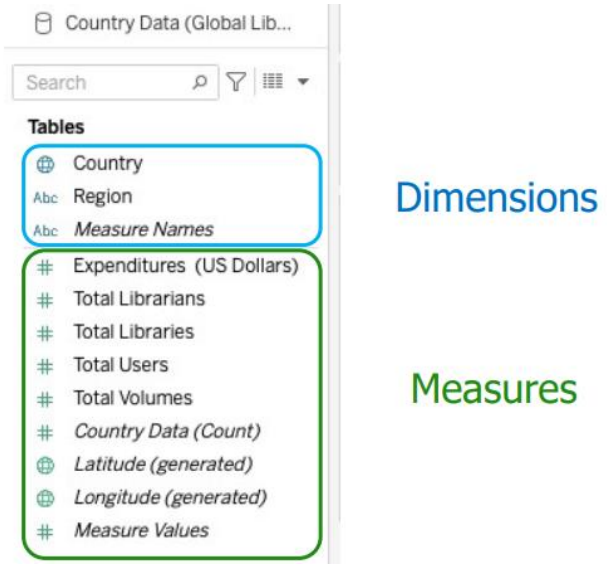
- Quantitative data (numerical);
- Qualitative data (descriptive, categorical/frequency count).



Data Field

A field, also known as a column, is a single piece of information from a record in a data set.

- Qualitative Field (Dimensions)
  - Describes or Categorizes Data
    - What, when or who
- Quantitative Field (Measures)
  - Numerical Data
  - Provides measurement for qualitative category
  - Can be used in calculations



Quantitative Data vs Qualitative Data Field

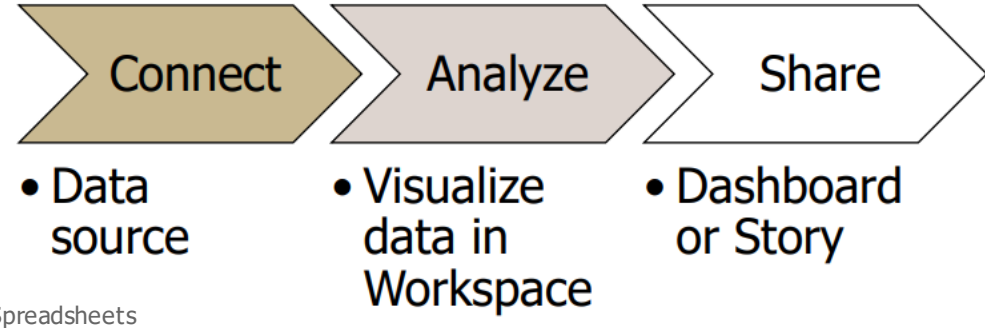
Quantitative Data has two types:

- (a) **Discrete**: Discrete variables can take only certain values.
- (b) **Continuous**: Continuous variables may take any value (typically between certain limits).

Qualitative Data is also called descriptive/ categorical data/ frequency count:

- When the data are arranged in categories on the basis of their quality and there is gap between two values,
- Qualitative data is initially expressed in non-numerical forms.

Data management



- Spreadsheets
  - Excel or csv file
- Relational Databases
  - MySQL or Oracle
- Cloud Data
  - • AWS or Microsoft Azure
- Other Sources

Data quality

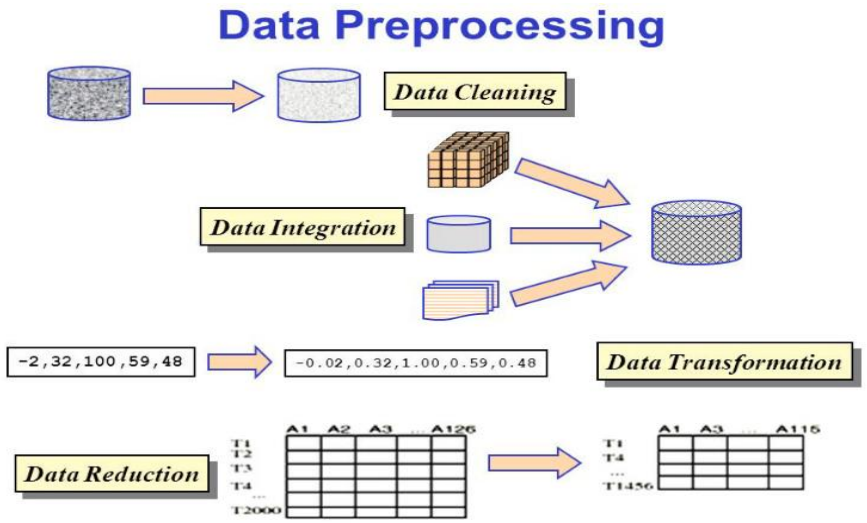
Data in the real world is dirty

- **Incomplete**: lacking attribute values, lacking certain attributes of interest, or containing only aggregate data,
- **Noisy**: containing errors or outliers, e.g., Salary="-10"
- **Inconsistent**: containing discrepancies in codes or names
  - e.g., Age="42" Birthday="03/07/1997"
  - e.g., Was rating "1,2,3", now rating "A, B, C"
  - e.g., discrepancy between duplicate records

Why Data Pre-processing?

Data Preprocessing is a technique uses to convert the raw data into a clean data set.

- Steps are executed to convert the data into a clean data set.
- Data is gathered from different sources (collected in raw format and it is not feasible for the analysis).
- This technique is performed before the execution of Iterative Analysis.
- Steps of data-preprocessing:
  - Data Cleaning
  - Data Integration
  - Data Transformation
  - Data Reduction



Data Cleaning

Data quality is a main issue and occurs anywhere in information systems.

These problems can be solved by Data Cleaning:

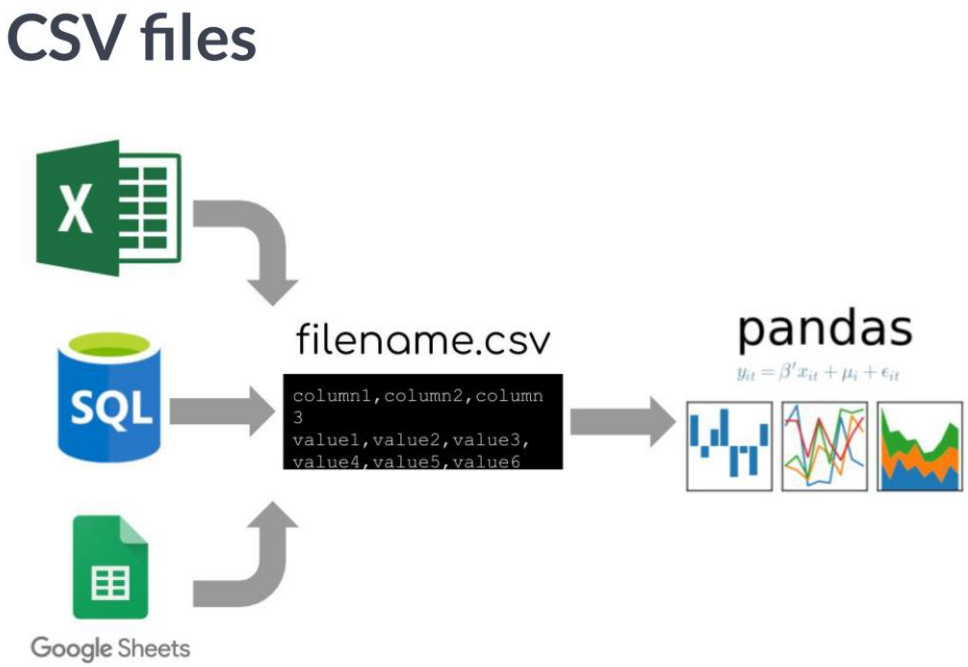
- is a process used to determine inaccurate, incomplete or unreasonable data
- and then improve the quality through correcting of detected errors
- => reduces errors and improves the data quality.

Data Cleaning can be a time consuming and tedious process, but it cannot be ignored.

Data quality criteria :

- accuracy, integrity, completeness, validity, consistency, uniqueness.

Data Integration: Loading tabular data from different sources



Introduction to Databases

A database consists of tables

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Table consist of columns and rows

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

Tables can be related

Census

state	sex	age	pop2000	pop2008
New York	F	0	120355	122194
New York	F	1	118219	119661
New York	F	2	119577	116413

State_Fact

name	abbreviation	type
New York	NY	state
Washington DC	DC	capitol
Washington	WA	state

Useful Python Libraries for Data visualization

NumPy

pandas

matplotlib

seaborn

bokeh

Matplotlib:

- Provides the **building blocks** for seaborn's and pandas visualizations
- It can also be used on its own to plot data

Pandas

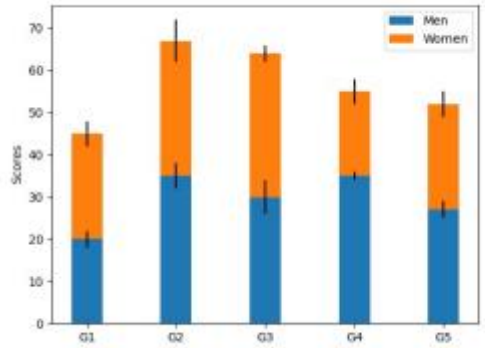
- It is a foundational library for **analyzing data**
- It also supports **basic plotting** capability

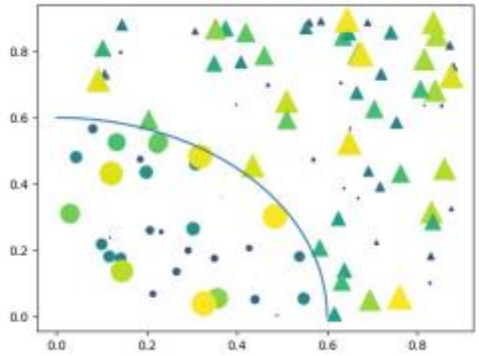
Seaborn

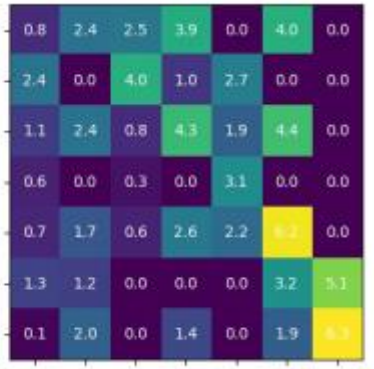
- Seaborn supports **complex visualizations** of data
- It is built on matplotlib and works best with pandas' dataframes

Matplotlib

- Used for basic plotting
- Highly customizable
- Works with NumPy and pandas







About Matplotlib:

- Matplotlib is a comprehensive library for creating static visualizations in Python.
- Usage: Matplotlib/Pandas is mostly used for quick plotting of Pandas DataFrames and time series analysis.

Advantages of Matplotlib:

- Easy to setup and use.
- Very customizable.

Llimitations of Matplotlib:

- Visual presentation tends to be simple compared to other tools.

About Seaborn:

- Seaborn is a Python data visualization library based on Matplotlib.
- It provides a **high-level interface for drawing attractive and informative statistical graphics**.
- Usage: Those who want to create **amplified data visuals, especially in color**.

Seaborn's Pros and Cons:

- Pro: Includes higher level interfaces and settings than does Matplotlib
- Pro: Relatively simple to use, just like Matplotlib.
- Pro: Easier to use when working with Dataframes.

- Con: Like Matplotlib, data visualization seems to be simpler than other tools.

Bokeh

- Bokeh is an interactive visualization Python library.
- Provides elegant and concise construction of versatile graphics.

- Usage: Can be used in Jupyter Notebooks and can provide high-performance interactive charts and plots.

Rules for variable names in Python

Rules for variable names

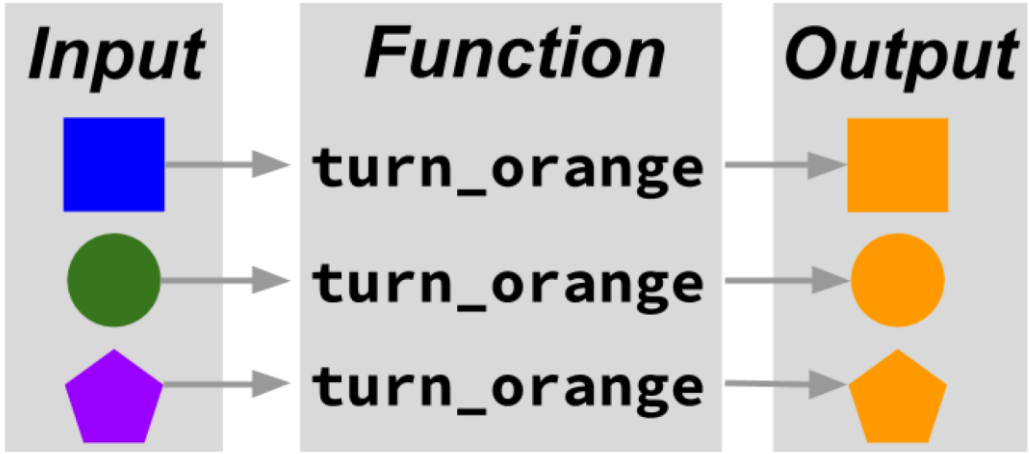
- Must start with a letter(usually lowercase)
- After first letter, can use letters/numbers/underscores
- No spaces or special characters
- Case sensitive (my_var is different from MY_VAR)

```
# Valid Variables
bayes_weight
b
bayes42
```

```
# Invalid Variables
bayes-height
bayes!
42bayes
```

Function

A function is an action



- Function name is always followed by parentheses ()

NumPy

- Fundamental package for scientific computing
- Exceptionally fast – written in C
- Main data structure:
  - ndarray : n-dimensional arrays of homogeneous data types
- Data manipulation ≈ NumPy array manipulation
- Used in other libraries - Matplotlib, pandas, scikit- learn

Pandas

- **Fundamental tool for handling and analyzing input data**
- Particularly **suited for tabular data**
- Implements powerful data operations
  - Easily read datasets from csv, txt, and other types of files
  - Datasets take the form of DataFrame objects
- Main data structures:
  - DataFrame: A table with rows and columns
  - Series: A single column

pandas Philosophy

There should be one -- and preferably only one -- obvious way to do it.

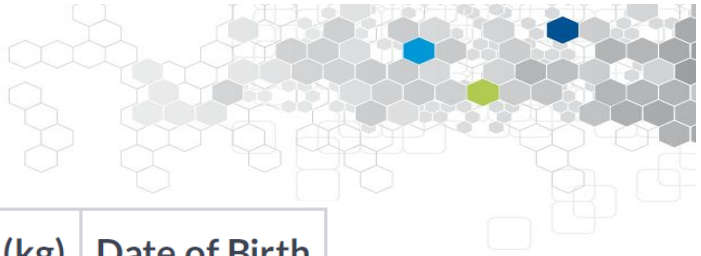
What's the point of pandas?

pandas is built on NumPy and Matplotlib

Outline: data manipulation with pandas

- **Chapter 1: DataFrames**
  - Sorting and subsetting
  - Creating new columns
- **Chapter 2: Aggregating Data**
  - Summary statistics
  - Counting
  - Grouped summary statistics
- **Chapter 3: Slicing and Indexing Data**
  - Subsetting using slicing
  - Indexes and subsetting using indexes
- **Chapter 4: Creating and Visualizing Data**
  - Plotting
  - Handling missing data
  - Reading data into a DataFrame

Rectangular data



Name	Breed	Color	Height (cm)	Weight (kg)	Date of Birth
Bella	Labrador	Brown	56	25	2013-07-01
Charlie	Poodle	Black	43	23	2016-09-16
Lucy	Chow Chow	Brown	46	22	2014-08-25
Cooper	Schnauzer	Gray	49	17	2011-12-11
Max	Labrador	Black	59	29	2017-01-20
Stella	Chihuahua	Tan	18	2	2015-04-20
Bernie	St. Bernard	White	77	74	2018-02-27

pandas DataFrames

To read a csv file ➔**pd.read_csv(path+filename)**

```
1   dogs = pd.read_csv("/content/dogs.csv")
2   print(dogs)
```

```
      name        breed  color  height_cm  weight_kg0
0    Bella     Labrador  Brown         56         251
1  Charlie       Poodle  Black         43         232
2     Lucy        Chow   Brown         46         223
3   Cooper    Schnauzer   Gray         49         174
4      Max     Labrador  Black         59         295
5   Stella    Chihuahua    Tan         18          26
6   Bernie  St. Bernard  White         77          74
```

Another data format

To read a pkl file ➔**pd.read_pkl(path+filename)**

```
unpickled_df = pd.read_pickle("/content/walmart_sales.pkl")
unpickled_df
```

	store	type	department	date	weekly_sales	is_holiday	temperature_c	fuel_price_usd_per_l	unemployment
0	1	A	1	2010-02-05	24924.50	False	5.727778	0.679451	8.106
1	1	A	2	2010-02-05	50605.27	False	5.727778	0.679451	8.106
2	1	A	3	2010-02-05	13740.12	False	5.727778	0.679451	8.106
3	1	A	4	2010-02-05	39954.04	False	5.727778	0.679451	8.106
4	1	A	5	2010-02-05	32229.38	False	5.727778	0.679451	8.106
...	...	...	...	...	...	...	...	...	...
413114	45	B	4	2012-10-26	24627.94	False	14.916667	1.025516	8.667
413115	45	B	5	2012-10-26	13256.59	False	14.916667	1.025516	8.667
413116	45	B	6	2012-10-26	1086.31	False	14.916667	1.025516	8.667
413117	45	B	7	2012-10-26	20356.73	False	14.916667	1.025516	8.667
413118	45	B	8	2012-10-26	37857.64	False	14.916667	1.025516	8.667

413119 rows × 9 columns

Exploring a DataFrame: .head(), .info(), .describe(), .shape

```
1  dogs.head()
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251
1	Charlie	Poodle	Black	43	232
2	Lucy	Chow	Brown	46	223
3	Cooper	Schnauzer	Gray	49	174
4	Max	Labrador	Black	59	295

```
1  dogs.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7 entries, 0 to 6
Data columns (total 5 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   name        7 non-null      object
 1   breed       7 non-null      object
 2   color       7 non-null      object
 3   height_cm   7 non-null      int64
 4   weight_kg0  7 non-null      int64
dtypes: int64(2), object(3)
memory usage: 408.0+ bytes
```

```
1  dogs.describe()
```

	height_cm	weight_kg0
count	7.000000	7.000000
mean	49.714286	182.142857
std	17.960274	98.128196
min	18.000000	26.000000
25%	44.500000	124.000000
50%	49.000000	223.000000
75%	57.500000	241.500000
max	77.000000	295.000000

```
1  dogs.shape
```

```
(7, 5)
```

Components of a DataFrame: .values, .columns and .index

```
1  dogs.values
```

```
array([['Bella', 'Labrador', 'Brown', 56, 251],
       ['Charlie', 'Poodle', 'Black', 43, 232],
       ['Lucy', 'Chow', 'Brown', 46, 223],
       ['Cooper', 'Schnauzer', 'Gray', 49, 174],
       ['Max', 'Labrador', 'Black', 59, 295],
       ['Stella', 'Chihuahua', 'Tan', 18, 26],
       ['Bernie', 'St. Bernard', 'White', 77, 74]], dtype=object)
```

```
<class
'numpy.ndarray'>
```

```
[20]  1  dogs.columns
```

```
Index(['name', 'breed', 'color', 'height_cm', 'weight_kg0'], dtype='object')
```

```
[21]  1  dogs.index
```

```
RangeIndex(start=0, stop=7, step=1)
```

Sorting and Subsetting

Sorting

Sorting in ascending order (by default)

```
1  dogs.sort_values("weight_kg0")
```

	name	breed	color	height_cm	weight_kg0
5	Stella	Chihuahua	Tan	18	26
6	Bernie	St. Bernard	White	77	74
3	Cooper	Schnauzer	Gray	49	174
2	Lucy	Chow	Brown	46	223
1	Charlie	Poodle	Black	43	232
0	Bella	Labrador	Brown	56	251
4	Max	Labrador	Black	59	295

Sorting in descending order

```
1  dogs.sort_values("weight_kg0", ascending=False)
```

	name	breed	color	height_cm	weight_kg0
4	Max	Labrador	Black	59	295
0	Bella	Labrador	Brown	56	251
1	Charlie	Poodle	Black	43	232
2	Lucy	Chow	Brown	46	223
3	Cooper	Schnauzer	Gray	49	174
6	Bernie	St. Bernard	White	77	74
5	Stella	Chihuahua	Tan	18	26

Sorting by multiple variables

```
1  dogs.sort_values(["weight_kg0", "height_cm"])
```

	name	breed	color	height_cm	weight_kg0
5	Stella	Chihuahua	Tan	18	26
6	Bernie	St. Bernard	White	77	74
3	Cooper	Schnauzer	Gray	49	174
2	Lucy	Chow	Brown	46	223
1	Charlie	Poodle	Black	43	232
0	Bella	Labrador	Brown	56	251
4	Max	Labrador	Black	59	295

Subsetting column/multiple columns

```
# Subsetting columns
dogs["name"]
```

```
0      Bella
1    Charlie
2       Lucy
3     Cooper
4        Max
5     Stella
6     Bernie
Name: name, dtype: object
```

```
dogs[["breed", "height_cm"]]
```

	breed	height_cm
0	Labrador	56
1	Poodle	43
2	Chow	46
3	Schnauzer	49
4	Labrador	59
5	Chihuahua	18
6	St. Bernard	77

```
cols_to_subset = ["breed", "height_cm"]
dogs[cols_to_subset]
```

	breed	height_cm
0	Labrador	56
1	Poodle	43
2	Chow	46
3	Schnauzer	49
4	Labrador	59
5	Chihuahua	18
6	St. Bernard	77

Subsetting rows (one column)

Subsetting based on **Condition**

```
1  dogs["height_cm"] > 50
```

```
0     True
1    False
2    False
3    False
4     True
5    False
6     True
Name: height_cm, dtype: bool
```

```
1  dogs[dogs["height_cm"] > 50]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251
4	Max	Labrador	Black	59	295
6	Bernie	St. Bernard	White	77	74

Subsetting based on text data

```
1  dogs[dogs["breed"] == "Labrador"]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251
4	Max	Labrador	Black	59	295

Subsetting based on dates

```
1  unpickled_df[ unpickled_df["date"] > "2012-10-15"]
```

Subsetting rows using .isin() : One column

```
1  is_black_or_brown = dogs["color"].isin(["Black", "Brown"])
2  dogs[is_black_or_brown]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251
1	Charlie	Poodle	Black	43	232
2	Lucy	Chow	Brown	46	223
4	Max	Labrador	Black	59	295

Subsetting based on multiple conditions (multiple columns)

&: and

```
1  is_lab = dogs["breed"] == "Labrador"
2  is_brown = dogs["color"] == "Brown"
3  dogs[is_lab & is_brown]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251

```
1  dogs[ (dogs["breed"] == "Labrador") & (dogs["color"] == "Brown") ]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251

Or :

```
1  is_lab = dogs["breed"] == "Labrador"
2  is_brown = dogs["color"] == "Brown"
3  dogs[is_lab | is_brown]
```

	name	breed	color	height_cm	weight_kg0
0	Bella	Labrador	Brown	56	251
2	Lucy	Chow	Brown	46	223
4	Max	Labrador	Black	59	295

New columns

Adding a new column

Conversion

```
1  dogs["height_m"] = dogs["height_cm"] / 100
2  print(dogs)
```

	name	breed	color	height_cm	weight_kg0	height_m
0	Bella	Labrador	Brown	56	251	0.56
1	Charlie	Poodle	Black	43	232	0.43
2	Lucy	Chow	Brown	46	223	0.46
3	Cooper	Schnauzer	Gray	49	174	0.49
4	Max	Labrador	Black	59	295	0.59
5	Stella	Chihuahua	Tan	18	26	0.18
6	Bernie	St. Bernard	White	77	74	0.77

$$\text{BMI} = \text{weight in kg}/(\text{height in m})^2$$

Operation:

```
1  dogs['bmi'] = dogs["weight_kg0"] / dogs["height_m"] **2
2  print(dogs.head())
```

	name	breed	color	height_cm	weight_kg0	height_m	bmi
0	Bella	Labrador	Brown	56	251	0.56	800.382653
1	Charlie	Poodle	Black	43	232	0.43	1254.732288
2	Lucy	Chow	Brown	46	223	0.46	1053.875236
3	Cooper	Schnauzer	Gray	49	174	0.49	724.698042
4	Max	Labrador	Black	59	295	0.59	847.457627

Summary statistics

Summarizing numerical data (statistics)

```
1  dogs["height_cm"].mean()
```

```
49.714285714285715
```

- `.median()`, `.mode()`
- `.min()`, `.max()`
- `.var()`, `.std()`
- `.sum()`
- `.quantile()`

Oldest:

```
1  unpickled_df["date"].min()
```

```
Timestamp('2010-02-05 00:00:00')
```

Youngest:

```
1  unpickled_df["date"].max()
```

```
Timestamp('2012-10-26 00:00:00')
```

From PMF to CDF

If you draw a random element from a distribution:

- PMF (Probability Mass Function) is the probability that you get exactly x
- CDF (Cumulative Distribution Function) is the probability that you get a value <= x

Example

PMF of {1, 2, 2, 3, 5}

PMF(1) = 1/5

PMF(2) = 2/5

PMF(3) = 1/5

PMF(5) = 1/5

CDF is the cumulative sum of the PMF.

CDF(1) = 1/5

CDF(2) = 3/5

CDF(3) = 4/5

CDF(5) = 1

Cumulative sum

```
1  dogs["weight_kg0"]
```

```
0    251
1    232
2    223
3    174
4    295
5     26
6     74
Name: weight_kg0, dtype: int64
```

```
1  dogs["weight_kg0"].cumsum()
```

```
0     251
1     483
2     706
3     880
4    1175
5    1201
6    1275
Name: weight_kg0, dtype: int64
```

Cumulative statistics

- .cummax()
- .cummin()
- .cumprod()

Dropping duplicate rows (according to columns) +Counting

```
print(dogs)
dogs.drop_duplicates(["color"])
```

```
      name        breed  color  height_cm  weight_kg0  height_m         bmi
0    Bella     Labrador  Brown         56         251      0.56  800.382653
1  Charlie       Poodle  Black         43         232      0.43 1254.732288
2     Lucy        Chow  Brown         46         223      0.46 1053.875236
3   Cooper    Schnauzer   Gray         49         174      0.49  724.698042
4      Max     Labrador  Black         59         295      0.59  847.457627
5   Stella    Chihuahua    Tan         18          26      0.18  802.469136
6   Bernie  St. Bernard  White         77          74      0.77  124.810255
```

	name	breed	color	height_cm	weight_kg0	height_m	bmi
0	Bella	Labrador	Brown	56	251	0.56	800.382653
1	Charlie	Poodle	Black	43	232	0.43	1254.732288
3	Cooper	Schnauzer	Gray	49	174	0.49	724.698042
5	Stella	Chihuahua	Tan	18	26	0.18	802.469136
6	Bernie	St. Bernard	White	77	74	0.77	124.810255

```
unique_dogs = dogs.drop_duplicates(subset=["name", "breed"])
print(unique_dogs)
```

```
      name        breed  color  height_cm  weight_kg0
0    Bella     Labrador  Brown         56         251
1  Charlie       Poodle  Black         43         232
2     Lucy        Chow  Brown         46         223
3   Cooper    Schnauzer   Gray         49         174
4      Max     Labrador  Black         59         295
5   Stella    Chihuahua    Tan         18          26
6   Bernie  St. Bernard  White         77          74
```

```
unique_dogs["breed"].value_counts()
```

```
Labrador      2
Chow          1
Chihuahua     1
Schnauzer     1
St. Bernard   1
Poodle        1
Name: breed, dtype: int64
```

unique_dogs["breed"].value_counts(sort=False)

```
# Proportions
unique_dogs["breed"].value_counts(normalize=True)
```

```
Labrador      0.285714
Chow          0.142857
Chihuahua     0.142857
Schnauzer     0.142857
St. Bernard   0.142857
Poodle        0.142857
Name: breed, dtype: float64
```

Summaries by group

```
print(dogs[dogs["color"] == "Black"]["weight_kg0"].mean())
print(dogs[dogs["color"] == "Brown"]["weight_kg0"].mean())
print(dogs[dogs["color"] == "White"]["weight_kg0"].mean())
print(dogs[dogs["color"] == "Gray"]["weight_kg0"].mean())
print(dogs[dogs["color"] == "Tan"]["weight_kg0"].mean())
```

```
263.5
237.0
74.0
174.0
26.0
```

```
dogs.groupby("color")["weight_kg0"].mean()
```

```
color
Black    263.5
Brown    237.0
Gray     174.0
Tan       26.0
White     74.0
Name: weight_kg0, dtype: float64
```

Multiple grouped summaries

```
dogs.groupby("color")["weight_kg0"].agg([min, max, sum])
```

color	min	max	sum
Black	232	295	527
Brown	223	251	474
Gray	174	174	174
Tan	26	26	26
White	74	74	74

Loading/storing a CSV

```
import pandas as pd
#loading a csv file
df = pd.read_csv('ransom.csv')


#Storing dataframe on a csv file

df.to_csv('ransom.csv', index=False)
```

Quiz

- Read and display the DataFrame of “ransom.csv”
- Analyze this Dataframe:
  - list of columns and their data type
  - Statistics results

Tabular data with pandas

Tabular Data

```
+-------------------------------------------------+
|        suspect        |     location    | price |
+-----------------------+-----------------+-------+
| Fred Frequentist      | Petroleum Plaza | 24.95 |
| Ronald Aylmer Fisher  | Clothing Club   | 20.15 |
+-------------------------------------------------+
```

DataFrame

```
                suspect        location  price
0      Fred Frequentist  Perolium Plaza  24.95
1  Ronald Aylmer Fisher   Clothing Club  20.15
```

Displaying a DataFrame

```
1  print(df)
```

```
   Letter  Frequency  Percentage
0       A   24373121         8.1
1       B    4762938         1.6
2       C    8982417         3.0
3       D   10805580         3.6
4       E   37907119        12.6
5       F    7486889         2.5
6       G    5143059         1.7
7       H   18058207         6.0
8       I   21820970         7.3
9       J     474021         0.2
10      K    1720909         0.6
11      L   11730498         3.9
12      M    7391366         2.5
13      N   21402466         7.1
14      O   23215532         7.7
15      P    5719422         1.9
16      Q     297237         0.1
17      R   17897352         5.9
18      S   19059775         6.3
19      T   28691274         9.5
20      U    8022379         2.7
21      V    2835696         0.9
22      W    6505294         2.2
23      X     562732         0.2
24      Y    5910495         2.0
25      Z      93172         0.0
```

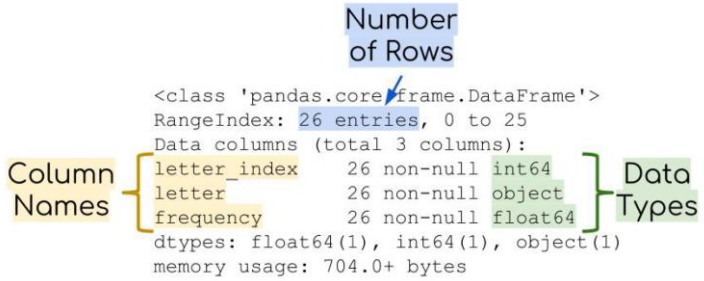
Inspecting a DataFrame (.head(), .info())

```
1  df.head()
2  # print(df.head())
```

	Letter	Frequency	Percentage
0	A	24373121	8.1
1	B	4762938	1.6
2	C	8982417	3.0
3	D	10805580	3.6
4	E	37907119	12.6

```
1
2  df.info()
3  # print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 26 entries, 0 to 25
Data columns (total 3 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   Letter      26 non-null     object
 1   Frequency   26 non-null     int64
 2   Percentage  26 non-null     float64
dtypes: float64(1), int64(1), object(1)
memory usage: 752.0+ bytes
```



Columns names

Columns names are strings

```
1   df.columns
```

```
Index(['Letter', 'Frequency', 'Percentage'], dtype='object')
```

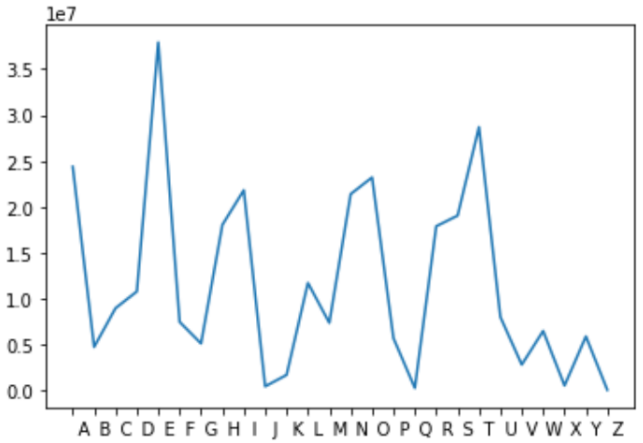
From DataFrame to Visualization

- Importing a Module
  - import pandas
- Importing a module with an alias
  - import pandas as pd

```
1  import pandas as pd
2  from matplotlib import pyplot as plt
3  # Pandas loads our data
4  df = pd.read_csv("/content/ransom.csv")
5  print(df.columns)
6  # print(df.head())
```

```
Index(['Letter', 'Frequency', 'Percentage'], dtype='object')
```

```
1  # Matplotlib plots and displays
2  plt.plot(df["Letter"], df["Frequency"])
3  plt.show()
```

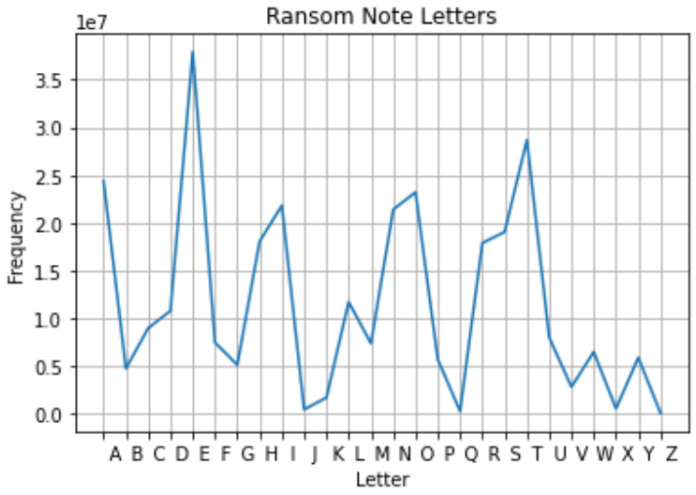


Functions perform actions:

- pd.read_csv() turns a csv file into a table in Python
- plt.plot() turns data into a line plot
- plt.show() displays plot in a new window

Axes and title labels

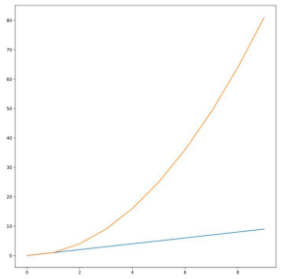
```
1  # Axes and title labels
2  plt.plot(df.Letter,df.Frequency,label='Frequency')
3  plt.xlabel("Letter")
4  plt.ylabel("Frequency")
5  plt.title("Ransom Note Letters")
6  plt.grid()
7  # Labels anywhere before
8  plt.show()
```



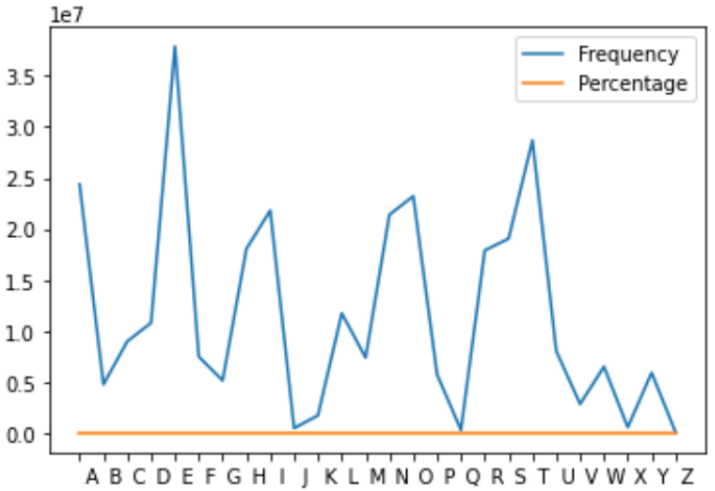
Multiple Lines

Adding labels and legends

```
plt.plot(data1.x_values,
         data1.y_values)

plt.plot(data2.x_values,
         data2.y_values)

plt.show()
```



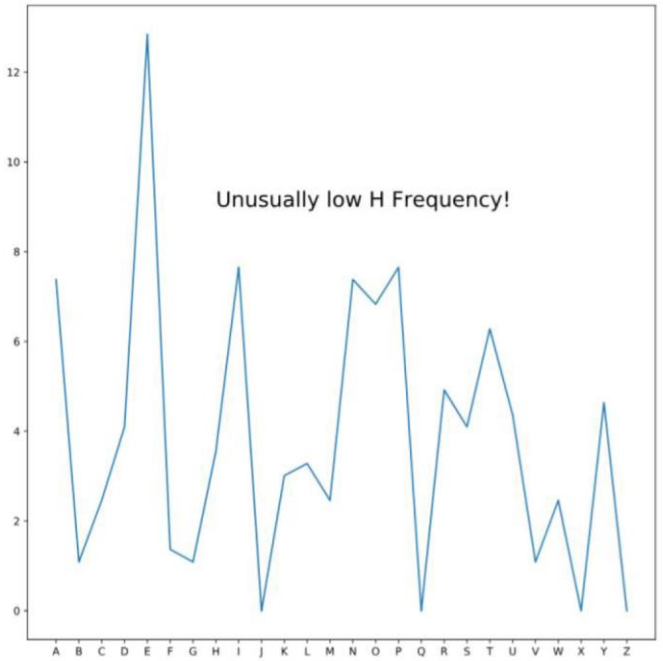
```
1  plt.plot(df.Letter,df.Frequency,label='Frequency')
2  plt.plot(df.Letter,df.Percentage,label='Percentage')
3  plt.legend()
4  plt.show()
```



Arbitrary text

```
plt.text(xcoord,
    ycoord,
    "Text Message")
```

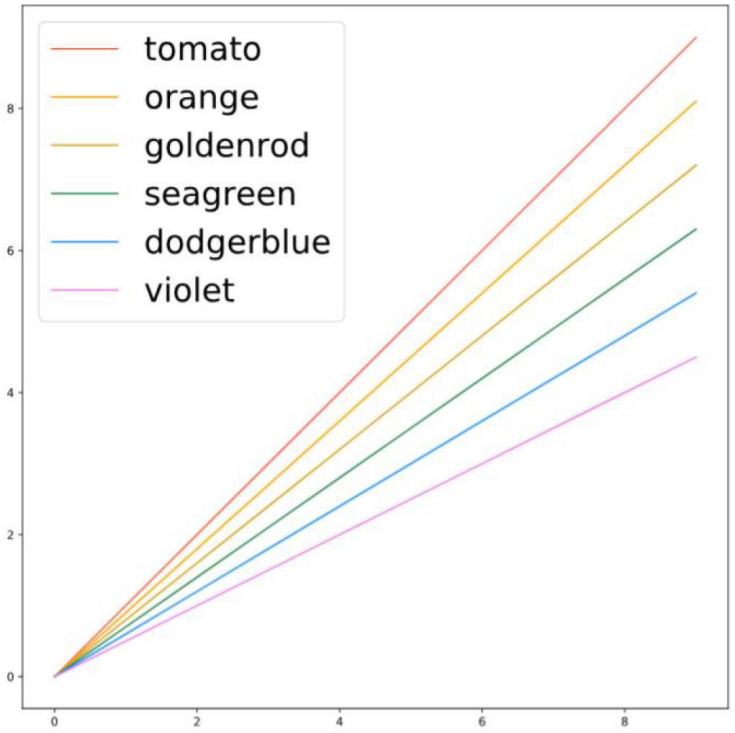
```
plt.text(5,
    9,
    "Unusually low H frequency!"
```




```
plt.title("Variation of", fontsize=17)
```

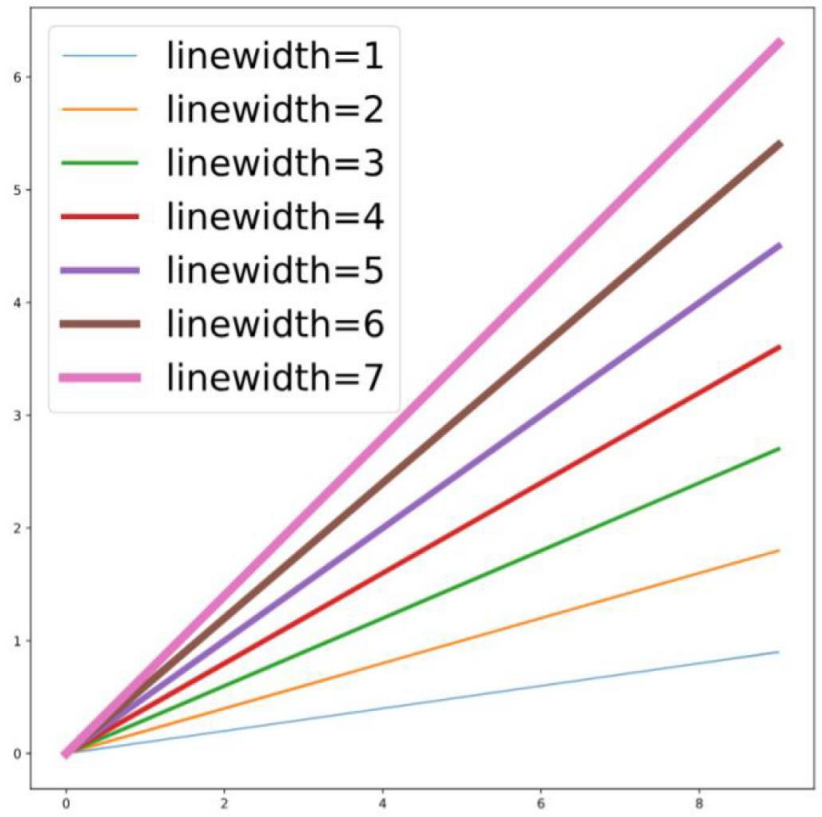
Changing line color

```
plt.plot(x, y1, color="tomato")
plt.plot(x, y2, color="organge")
plt.plot(x, y3, color="goldenrod")
plt.plot(x, y4, color="seagreen")
plt.plot(x, y5, color="dodgerblue")
plt.plot(x, y6, color="violet")
```



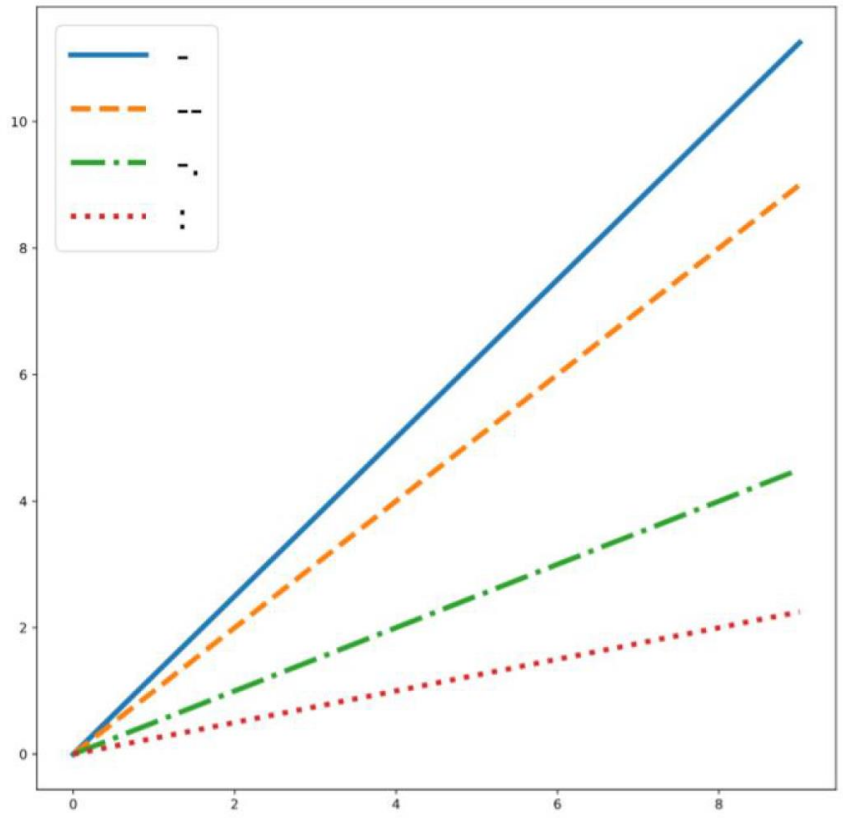
Changing line width

```
plt.plot(x, y1, linewidth=1)
plt.plot(x, y2, linewidth=2)
plt.plot(x, y3, linewidth=3)
plt.plot(x, y4, linewidth=4)
plt.plot(x, y5, linewidth=5)
plt.plot(x, y6, linewidth=6)
plt.plot(x, y7, linewidth=7)
```



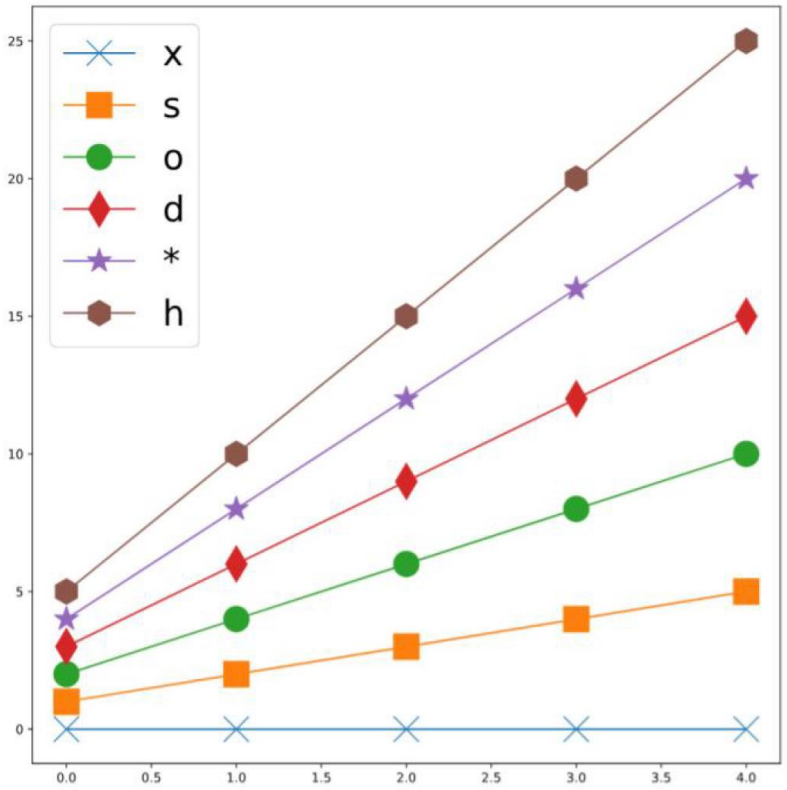
Changing line style

```
plt.plot(x, y1, linestyle='-')
plt.plot(x, y2, linestyle='--')
plt.plot(x, y3, linestyle='-.')
plt.plot(x, y4, linestyle=':')
```



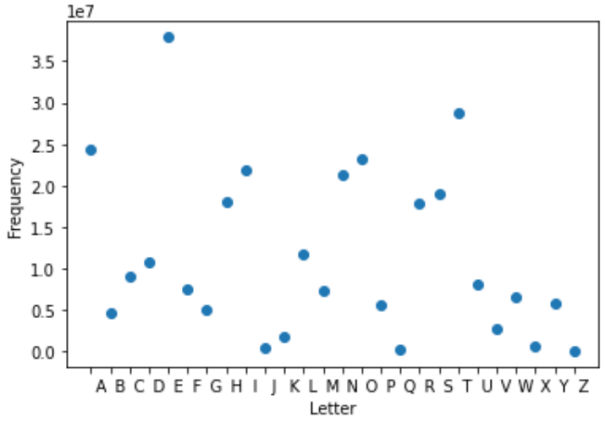
Adding markers

```
plt.plot(x, y1, marker='x')
plt.plot(x, y2, marker='s')
plt.plot(x, y3, marker='o')
plt.plot(x, y4, marker='d')
plt.plot(x, y5, marker='*')
plt.plot(x, y6, marker='h')
```



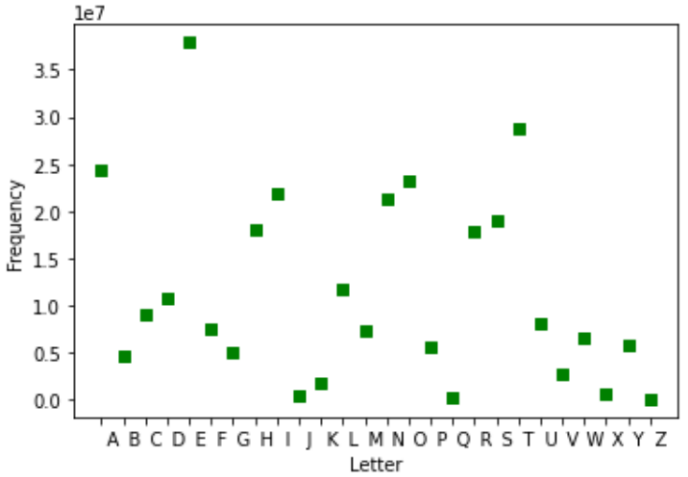
Creating a scatter plot

```
1  plt.scatter(df.Letter,df.Frequency)
2  plt.xlabel("Letter")
3  plt.ylabel("Frequency")
4  plt.show()
```

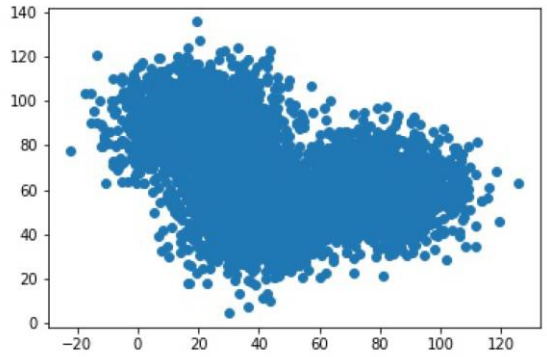


Keyword arguments

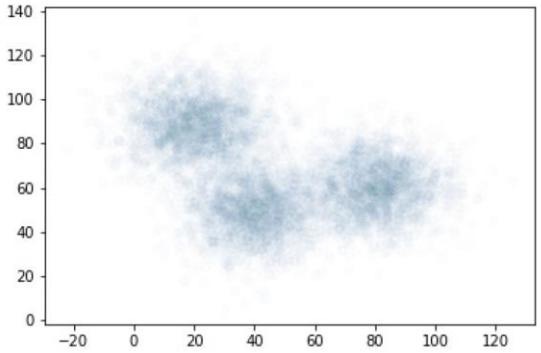
```
1  plt.scatter(df.Letter,df.Frequency,color='green', marker='s')
2  plt.xlabel("Letter")
3  plt.ylabel("Frequency")
4  plt.show()
```



Changing marker transparency




```
plt.scatter(df.x_data,
            df.y_data,
            alpha=0.1)
```



Data Visualization with Matplotlib

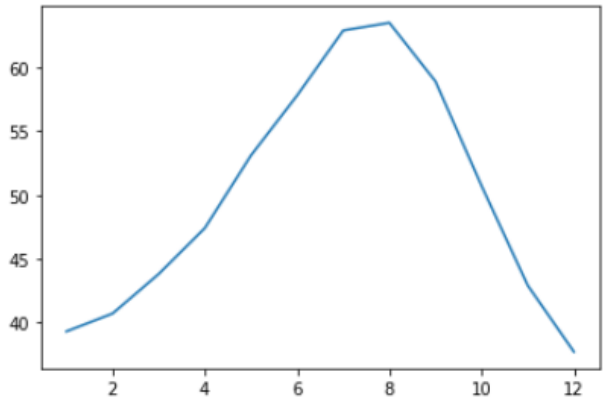
Basic plot (1)

```
1  seattle_weather=pd.read_csv("/content/seattle_weather.csv")
```

```
1  np.unique(seattle_weather["DATE"])
```

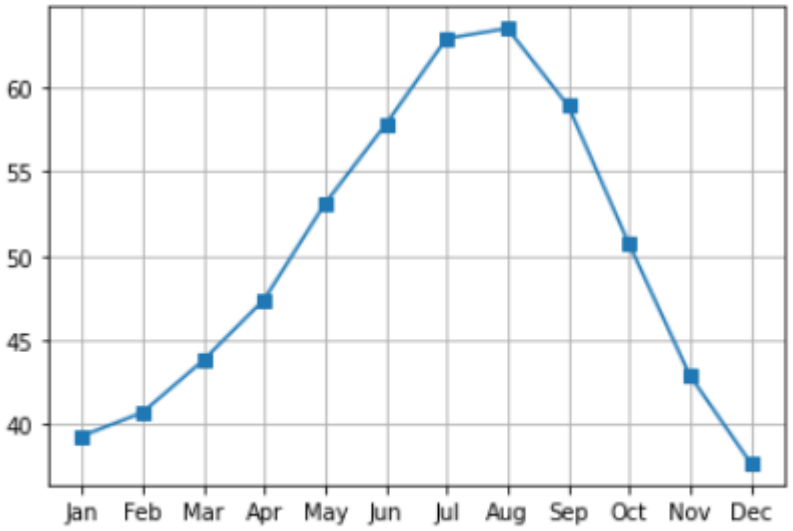
```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12])
```

```
1  plt.plot(seattle_weather["DATE"].loc[:11], seattle_weather["MLY-TAVG-NORMAL"].loc[:11])
2  plt.show()
```



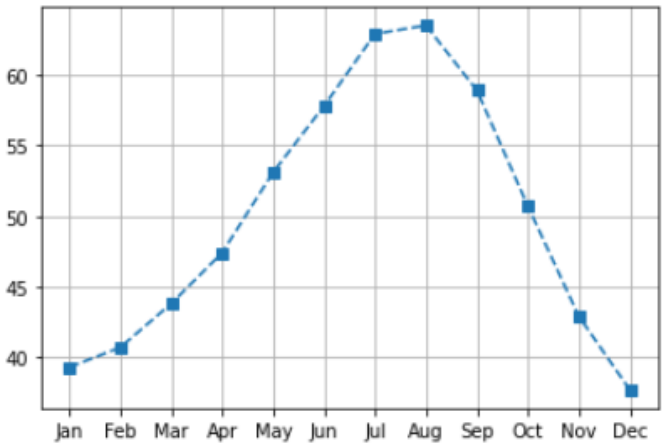
Basic Plot + grid+ marker

```
1  import calendar
2  seattle_weather['Month'] = seattle_weather['DATE'].apply(lambda x: calendar.month_abbr[x])
3  plt.plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="s")
4  plt.grid()
5  plt.show()
```



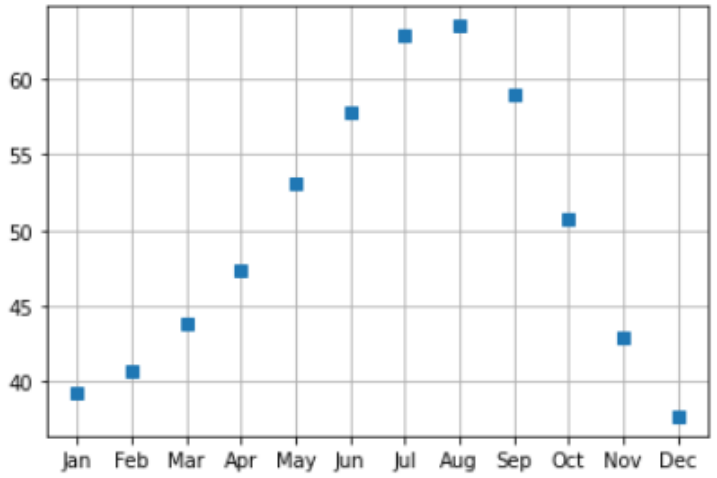
Setting the linestyle or

```
import calendar
seattle_weather['Month'] = seattle_weather['DATE'].apply(lambda x: calendar.month_abbr[x])
plt.plot(seattle_weather["Month"].loc[:11],
         seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="s",
        linestyle="--")
plt.grid()
plt.show()
```



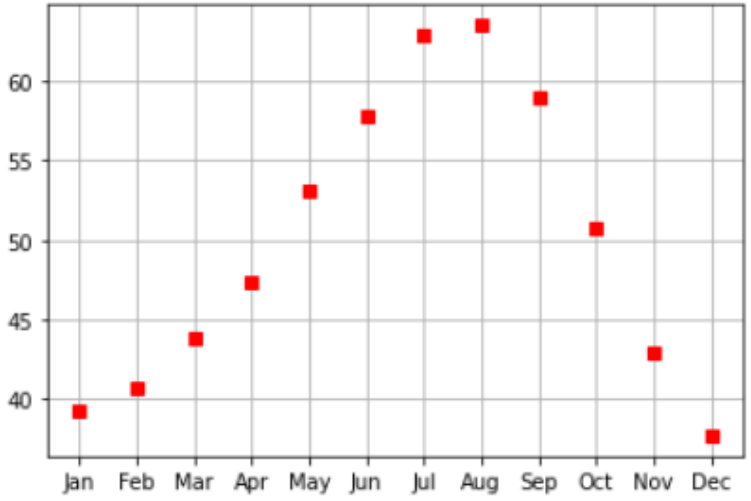
Eliminating lines with linestyle

```
1  import calendar
2  seattle_weather['Month'] = seattle_weather['DATE'].apply(lambda x: calendar.month_abbr[x])
3  plt.plot(seattle_weather["Month"].loc[:11],
4           seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="s",
5           linestyle="None")
6  plt.grid()
7  plt.show()
```



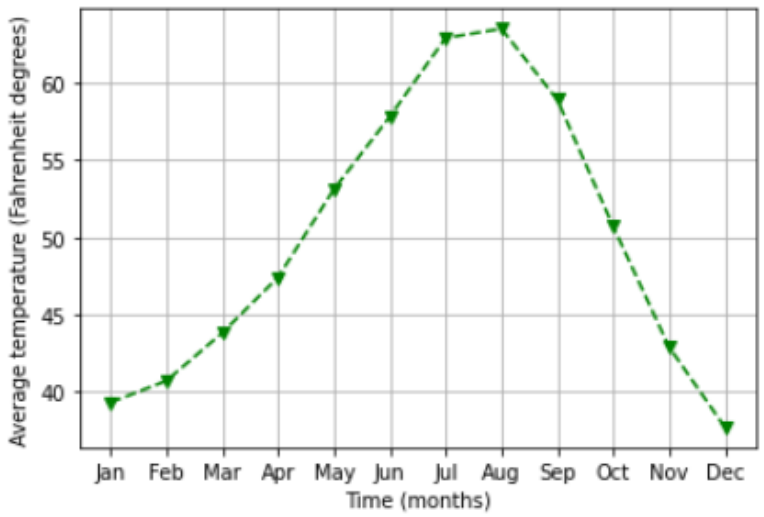
Choosing color

```
1  import calendar
2  seattle_weather['Month'] = seattle_weather['DATE'].apply(lambda x: calendar.month_abbr[x])
3  plt.plot(seattle_weather["Month"].loc[:11],
4           seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="s",
5           linestyle="None", color="r")
6  plt.grid()
7  plt.show()
```



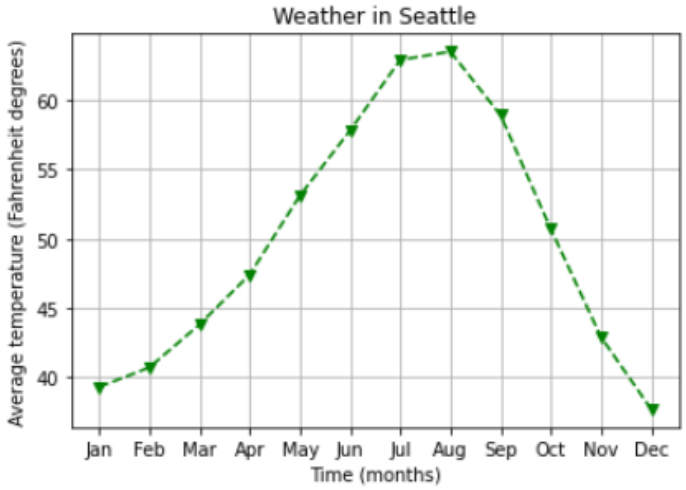
Setting the x &y axis label

```
plt.plot(seattle_weather["Month"].loc[:11],
         seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="v",
        linestyle="--", color="g")
plt.grid()
plt.xlabel("Time (months)")
plt.ylabel("Average temperature (Fahrenheit degrees)")
plt.show()
```



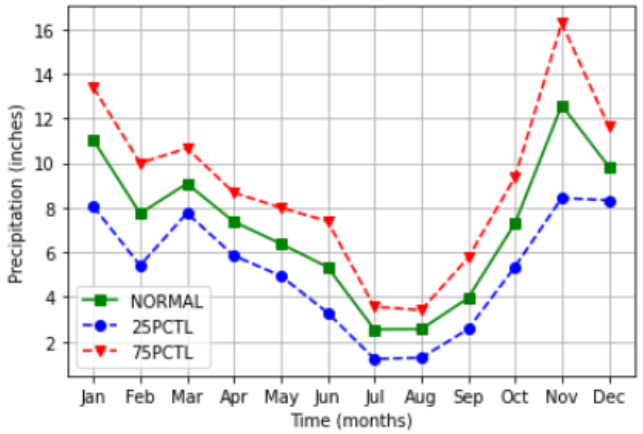
Adding a title

```
plt.plot(seattle_weather["Month"].loc[:11],
         seattle_weather["MLY-TAVG-NORMAL"].loc[:11],marker="v",
        linestyle="--", color="g")
plt.grid()
plt.xlabel("Time (months)")
plt.ylabel("Average temperature (Fahrenheit degrees)")
plt.title("Weather in Seattle")
plt.show()
```



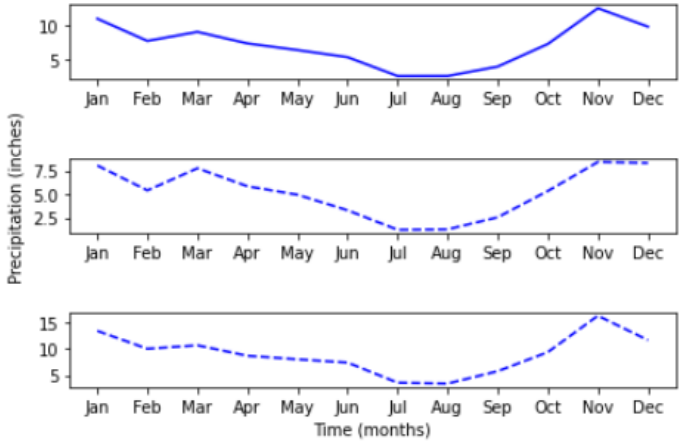
Adding more data: multi-plot

```
1  plt.plot(seattle_weather["Month"].loc[:11],seattle_weather["MLY-PRCP-NORMAL"].loc[:11],color='g',marker='s',label='NORMAL')
2  plt.plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-25PCTL"].loc[:11],linestyle='--', color='b',marker='o',label='25PCTL')
3  plt.plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-75PCTL"].loc[:11],linestyle='--', color='r',marker='v',label='75PCTL')
4  plt.xlabel("Time (months)")
5  plt.ylabel("Precipitation (inches)")
6  plt.grid()
7  plt.legend()
8  plt.show()
```



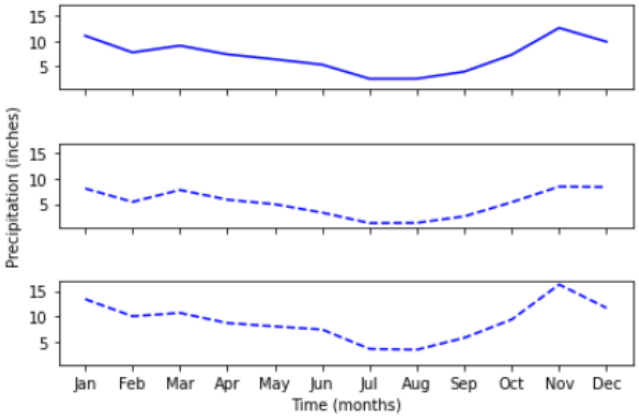
Small multiples: sub-plot

```
1   # Subplots
2   fig, ax = plt.subplots(3, 1)
3   ax[0].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-NORMAL"].loc[:11],
4   color='b')
5   ax[1].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-25PCTL"].loc[:11],
6   linestyle='--', color='b')
7   ax[2].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-75PCTL"].loc[:11],
8   linestyle='--', color='b')
9   ax[1].set_ylabel("Precipitation (inches)")
10  ax[2].set_xlabel("Time (months)")
11  fig.tight_layout()
12  plt.show()
```



Sharing the y & x axis range

```
1  # Subplots
2  fig, ax = plt.subplots(3, 1, sharey=True, sharex=True)
3  ax[0].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-NORMAL"].loc[:11],
4  color='b')
5  ax[1].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-25PCTL"].loc[:11],
6  linestyle='--', color='b')
7  ax[2].plot(seattle_weather["Month"].loc[:11], seattle_weather["MLY-PRCP-75PCTL"].loc[:11],
8  linestyle='--', color='b')
9  ax[1].set_ylabel("Precipitation (inches)")
10 ax[2].set_xlabel("Time (months)")
11 fig.tight_layout()
12 plt.show()
```



Pre-processing operations

```
climate_change=pd.read_csv("/content/climate_change.csv")
climate_change['day'] = pd.to_datetime(climate_change['date']).dt.day
climate_change['month'] = pd.to_datetime(climate_change['date']).dt.month
climate_change['year'] = pd.to_datetime(climate_change['date']).dt.year
climate_change.drop(['date'], axis=1)
climate_change = climate_change.drop('date', 1)
climate_change = climate_change[['day', 'month', 'year', 'co2', 'relative_temp']]
print(climate_change)
```

```
     day  month  year     co2  relative_temp
0      6      3  1958  315.71           0.10
1      6      4  1958  317.45           0.01
2      6      5  1958  317.50           0.08
3      6      6  1958     NaN          -0.05
4      6      7  1958  315.86           0.06
..   ...    ...   ...     ...            ...
701    6      8  2016  402.27           0.98
702    6      9  2016  401.05           0.87
703    6     10  2016  401.59           0.89
704    6     11  2016  403.55           0.93
705    6     12  2016  404.45           0.81

[706 rows x 5 columns]
```

Handling missing data

- Eliminating samples/features with missing cells.
- Estimating -missing values via interpolation (or statistics values)

Drop NAN values

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	0.0	11.0	12.0	NaN

We can use **isnull()** method to check whether a cell contains a numeric value (False) or if data is missing (True):

```
df.isnull()
```

	A	B	C	D
0	False	False	False	False
1	False	False	True	False
2	False	False	False	True

For a larger DataFrame, we may want to use the **sum()** method which returns the number of missing values per column:

```
df.isnull().sum()
```

```
A    0
B    0
C    1
D    1
dtype: int64
```

```
df.dropna()
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

We can drop columns that have at least one NaN in any row by setting the axis argument to 1:

```
df.dropna(axis=1)
```

	A	B
0	1.0	2.0
1	5.0	6.0
2	0.0	11.0

The dropna() method has several additional parameters:

```
# only drop rows where all columns are NaN
df.dropna(how='all')
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
1	5.0	6.0	NaN	8.0
2	0.0	11.0	12.0	NaN

```
# drop rows that do not have at least 4 non-NaN values
df.dropna(thresh=4)
```

	A	B	C	D
0	1.0	2.0	3.0	4.0

```
# only drop rows where NaN appear in specific columns (here: 'C')
df.dropna(subset=['C'])
```

	A	B	C	D
0	1.0	2.0	3.0	4.0
2	0.0	11.0	12.0	NaN

Replace NAN Values

Methods to replace NaN values with zeros in Pandas DataFrame:

- **fillna()**
  The `fillna()` function is used to fill NA/NaN values using the specified method.
- **replace()**
  The `dataframe.replace()` function in Pandas can be defined as a simple method used to replace a string, regex, list, dictionary etc. in a DataFrame.

Steps to replace NaN values:

- For one column using pandas:

```
df['DataFrame Column'] = df['DataFrame Column'].fillna(0)
```

- For one column using numpy:

```
df['DataFrame Column'] = df['DataFrame Column'].replace(np.nan, 0)
```

- For the whole DataFrame using pandas:

```
df.fillna(0)
```

- For the whole DataFrame using numpy:

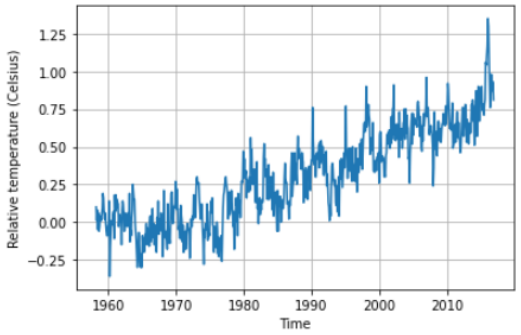
```
df.replace(np.nan, 0)
```

Replace NAN values by Interpolation

Plotting time-series data

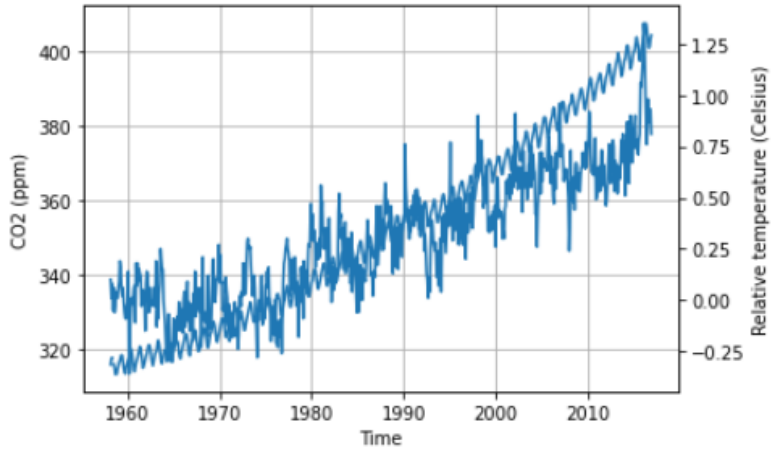
Climate change time-series

```
import pandas as pd
import matplotlib.pyplot as plt
# Read the data from file using read_csv
climate_change = pd.read_csv('climate_change.csv', parse_dates=True, index_col="date")
# Use the parse_dates key-word argument to parse the "date" column as dates.
# Use the index_col key-word argument to set the "date" column as the index.
# Plot time-series data

# Add the time-series for "relative_temp" to the plot. Use the DataFrame index for the x value and the "relative_temp" column for the y values.
plt.plot(climate_change.index, climate_change["relative_temp"])

# Set the x-axis label to 'Time'.
plt.xlabel('Time')

# Set the y-axis label to 'Relative temperature (Celsius)'.
plt.ylabel('Relative temperature (Celsius)')
plt.grid()
# Show the figure
plt.show()
```



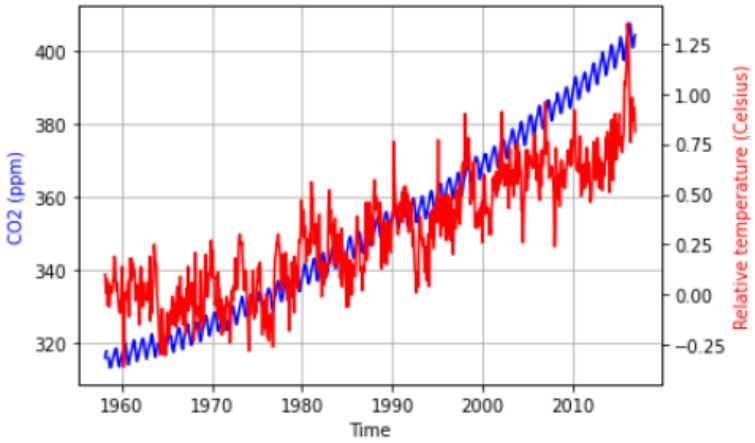
Using twin axes

```
1  plt.plot(climate_change.index, climate_change["co2"])
2  plt.xlabel('Time')
3  plt.ylabel('CO2 (ppm)')
4  plt.grid()
5  plt2 = plt.twinx()
6  plt2.plot(climate_change.index, climate_change["relative_temp"])
7  plt2.set_ylabel('Relative temperature (Celsius)')
8  # plt2.grid()
9  plt.show()
```



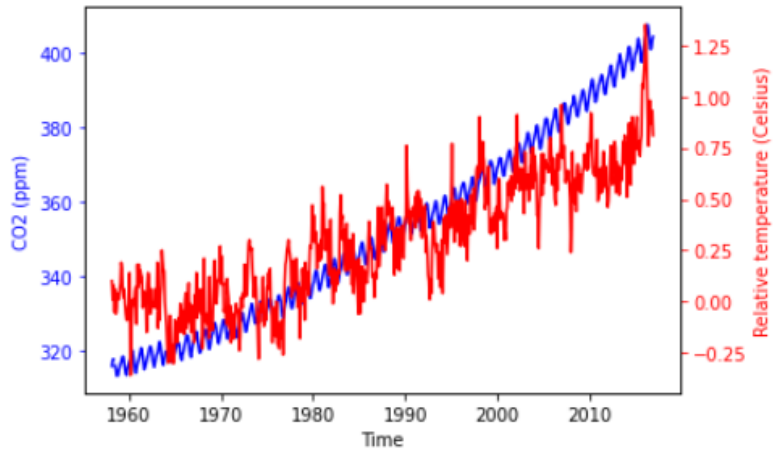
Separating variables by color

```
1  plt.plot(climate_change.index, climate_change["co2"], color='blue')
2  plt.xlabel('Time')
3  plt.ylabel('CO2 (ppm)', color='blue')
4  plt.grid()
5  plt2 = plt.twinx()
6  plt2.plot(climate_change.index, climate_change["relative_temp"], color='red')
7  plt2.set_ylabel('Relative temperature (Celsius)', color='red')
8  # plt2.grid()
9  plt.show()
```



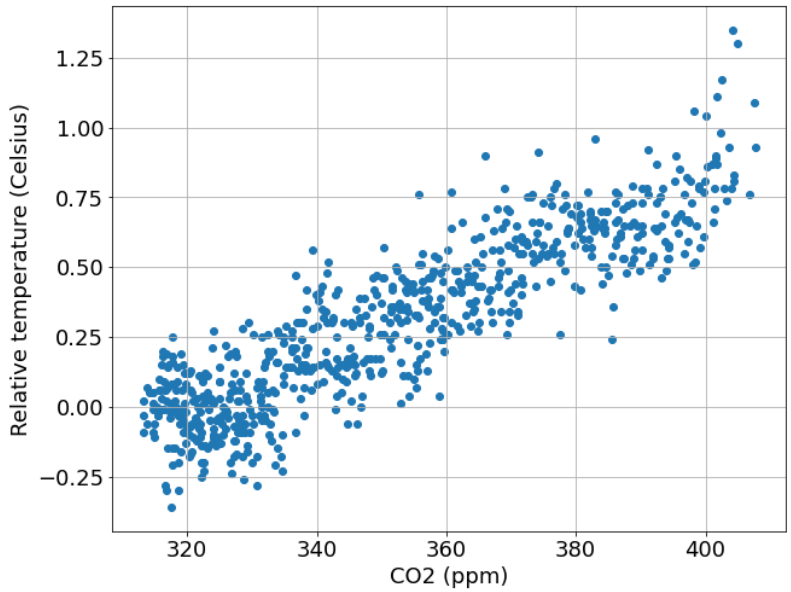
Coloring the ticks

```
1  plt.plot(climate_change.index, climate_change["co2"],color='blue')
2  plt.xlabel('Time')
3  plt.ylabel('CO2 (ppm)', color='blue')
4  plt.tick_params('y', colors='blue')
5  plt2 = plt.twinx()
6  plt2.plot(climate_change.index,climate_change["relative_temp"],color='red')
7  plt2.set_ylabel('Relative temperature (Celsius)',color='red')
8  plt2.tick_params('y', colors='red')
9  plt.show()
```



Scatter plots

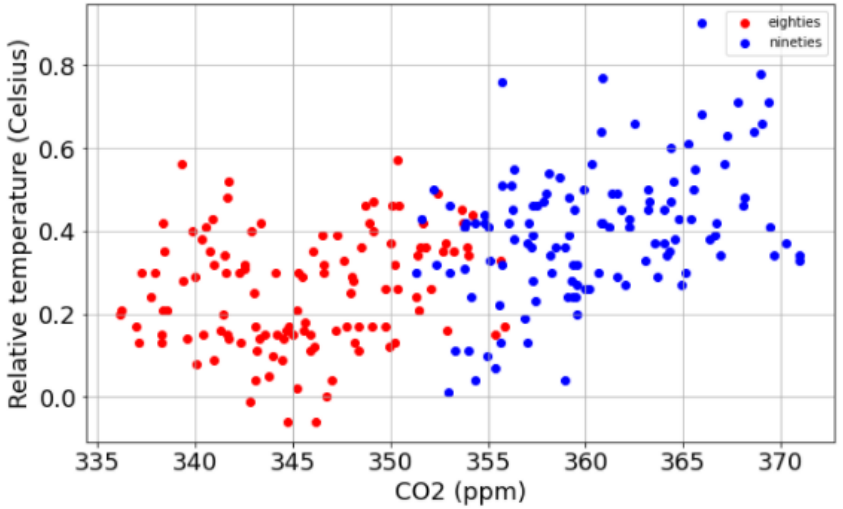
```
plt.figure(figsize=(10,8))
plt.scatter(climate_change["co2"], climate_change["relative_temp"])
plt.xlabel("CO2 (ppm)", fontsize=18)
plt.ylabel("Relative temperature (Celsius)", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.grid()
plt.show()
```



A scatter plot is a diagram where each value in the data set is represented by a dot.

Customizing scatter plots: Encoding a comparison by color

```
1  plt.figure(figsize=(10,6))
2  eighties = climate_change["1980-01-01":"1989-12-31"]
3  nineties = climate_change["1990-01-01":"1999-12-31"]
4  plt.scatter(eighties["co2"], eighties["relative_temp"],color="red", label="eighties")
5  plt.scatter(nineties["co2"], nineties["relative_temp"],color="blue", label="nineties")
6  plt.legend()
7  plt.xlabel("CO2 (ppm)", fontsize=18)
8  plt.ylabel("Relative temperature (Celsius)", fontsize=18)
9  plt.xticks(fontsize=18)
10 plt.yticks(fontsize=18)
11 plt.grid()
12 plt.show()
```



Preparing your figures to share with others

Changing plot style

Choosing a style:➔ plt.style.use("ggplot")

Back to the default:➔plt.style.use("default")

The available styles

https://matplotlib.org/stable/gallery/style_sheets/style_sheets_reference.html

Setting a style

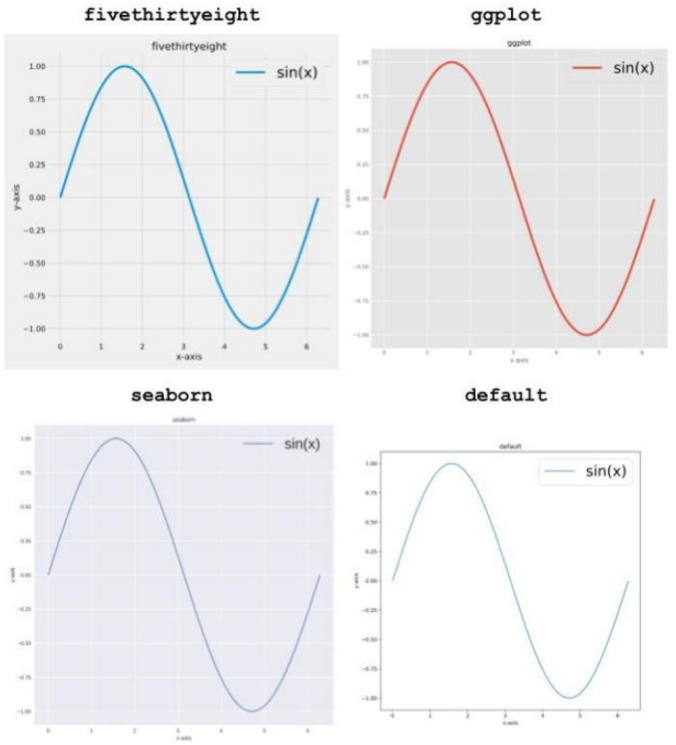
Before any other plotting code:

```
plt.style.use('fivethirtyeight')
```

```
plt.style.use('ggplot')
```

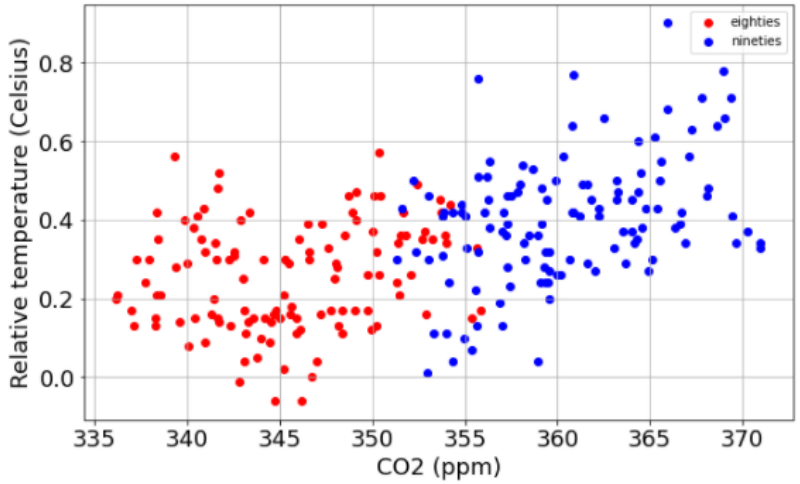
```
plt.style.use('seaborn')
```

```
plt.style.use('default')
```



Saving the figure to file

```
fig=plt.figure(figsize=(10,6))
plt.style.use("default")
eighties = climate_change["1980-01-01":"1989-12-31"]
nineties = climate_change["1990-01-01":"1999-12-31"]
plt.scatter(eighties["co2"], eighties["relative_temp"],color="red", label="eighties")
plt.scatter(nineties["co2"], nineties["relative_temp"],color="blue", label="nineties")
plt.legend()
plt.xlabel("CO2 (ppm)", fontsize=18)
plt.ylabel("Relative temperature (Celsius)", fontsize=18)
plt.xticks(fontsize=18)
plt.yticks(fontsize=18)
plt.grid()
plt.show()
fig.savefig("scatterClimateChange.png")
```



Different file formats

```
fig.savefig("gold_medals.jpg")
```

```
fig.savefig("gold_medals.jpg", quality=50)
```

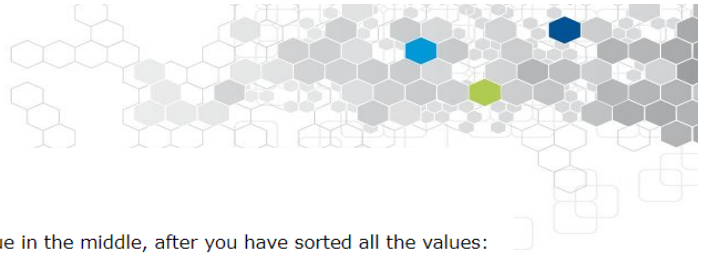
```
fig.savefig("gold_medals.svg")
```

Resolution

```
fig.savefig("gold_medals.png", dpi=300)
```

Comparisons:
bar charts
&
Statistical plotting

Mean, Median, Mode



Mean, Median, and Mode

What can we learn from looking at a group of numbers?

In Machine Learning (and in mathematics) there are often three values that interests us:

- **Mean** - The average value
- **Median** - The mid point value
- **Mode** - The most common value

Example: We have registered the speed of 13 cars:

```
speed = [99,86,87,88,111,86,103,87,94,78,77,85,86]
```

What is the average, the middle, or the most common speed value?

Mean

The mean value is the average value.

To calculate the mean, find the sum of all values, and divide the sum by the number of values:

```
(99+86+87+88+111+86+103+87+94+78+77+85+86) / 13 = 89.77
```

Median

The median value is the value in the middle, after you have sorted all the values:

```
77, 78, 85, 86, 86, 86, 87, 87, 88, 94, 99, 103, 111
```

Mode

The Mode value is the value that appears the most number of times:

```
99, 86, 87, 88, 111, 86, 103, 87, 94, 78, 77, 85, 86 = 86
```

What is Standard Deviation?

Standard deviation is a number that describes how spread out the values are.

A low standard deviation means that most of the numbers are close to the mean (average) value.

A high standard deviation means that the values are spread out over a wider range.

Example: This time we have registered the speed of 7 cars:

```
speed = [86,87,88,86,87,85,86]
```

The standard deviation is:

```
0.9
```

Meaning that most of the values are within the range of 0.9 from the mean value, which is 86.4.

Let us do the same with a selection of numbers with a wider range:

```
speed = [32,111,138,28,59,77,97]
```

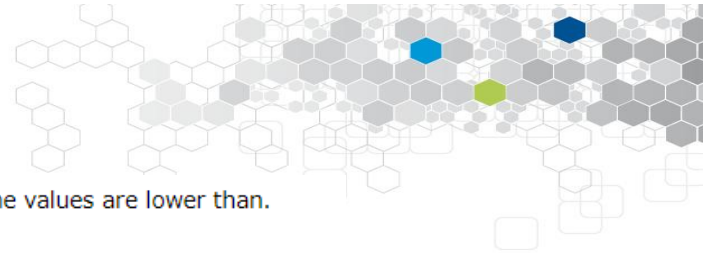
The standard deviation is:

```
37.85
```

Meaning that most of the values are within the range of 37.85 from the mean value, which is 77.4.

As you can see, a higher standard deviation indicates that the values are spread out over a wider range.

What are Percentiles



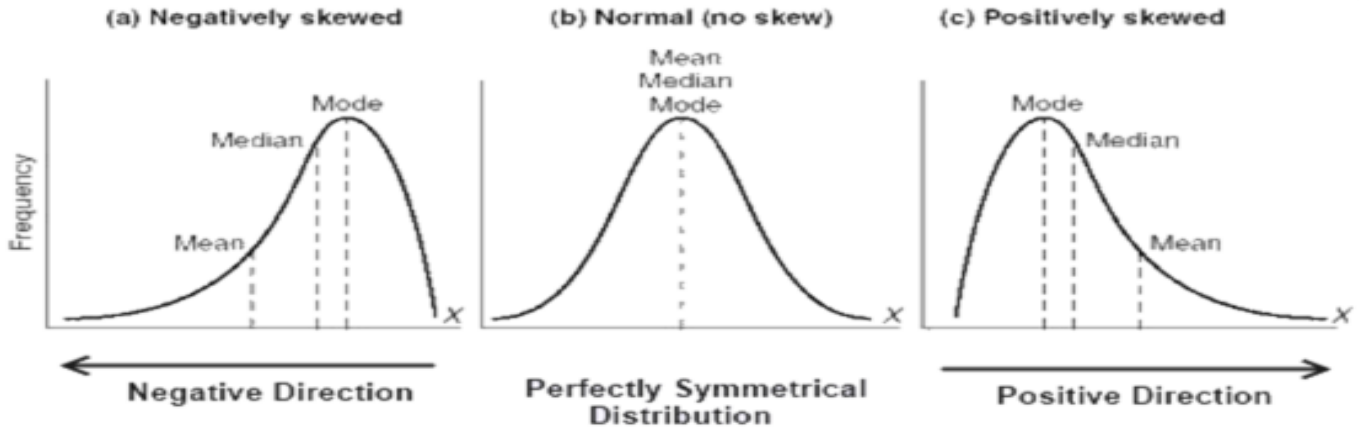
Percentiles are used in statistics to give you a number that describes the value that a given percent of the values are lower than.

Example: Let's say we have an array of the ages of all the people that lives in a street.

```
ages = [5,31,43,48,50,41,7,11,15,39,80,82,32,2,8,6,25,36,27,61,31]
```

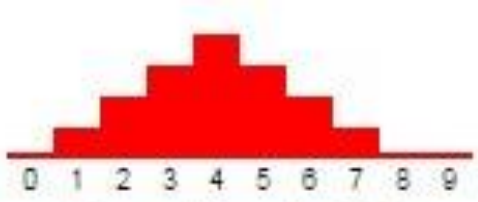
What is the 75. percentile? The answer is 43, meaning that 75% of the people are 43 or younger.

COMPARAISON ENTRE MOYENNE, MÉDIANE, ET MODE

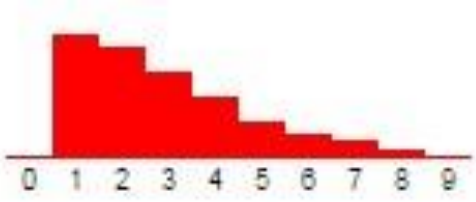


- La moyenne est sensible aux valeurs aberrantes, contrairement à la médiane.
- Parfois, la médiane peut fournir plus d'informations sur les données que la moyenne.
    - Dans le cas d'une distribution symétrique, la moyenne, la médiane et le mode sont approximativement les mêmes et la distribution est approximativement gaussienne ou normale.
    - Si la moyenne est inférieure à la médiane, et la médiane inférieure au mode, la distribution est asymétrique à gauche
    - Si la distribution est asymétrique à droite, cela signifie que nous avons un petit nombre de valeurs élevées. Ces valeurs augmenteront la moyenne, mais elles n'affecteront pas vraiment la médiane.

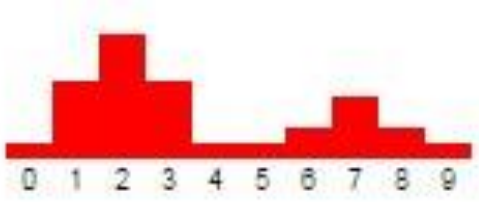
Data Distribution



Symmetric, unimodal, bell-shaped

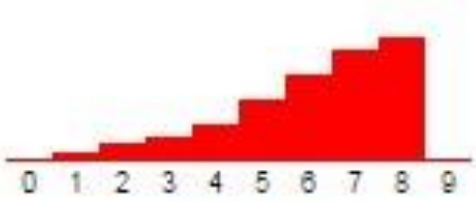


Skewed right

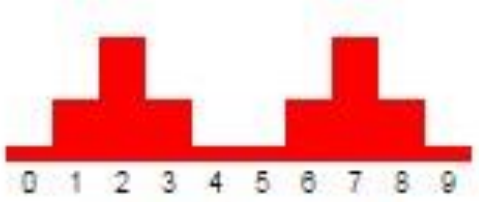


Non-symmetric, bimodal

Uniform



Skewed left



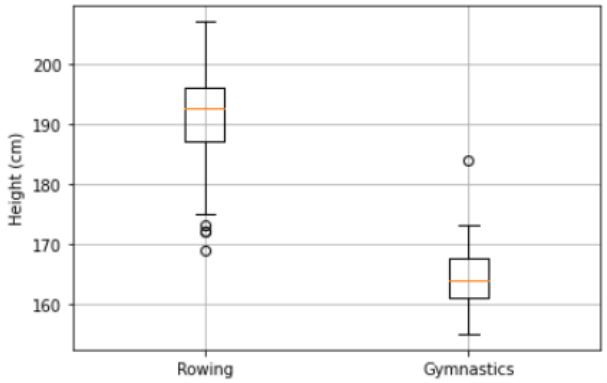
Symmetric, bimodal

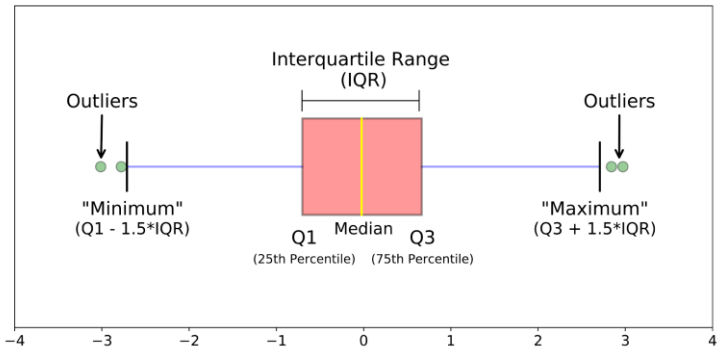
Adding boxplots

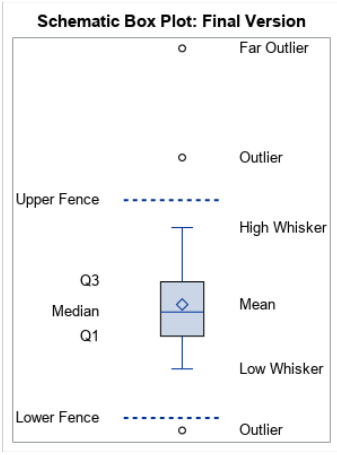
```
mens_rowing = pd.read_csv('/content/mens-rowing.csv')
mens_gymnastics = pd.read_csv('/content/mens_gymnastics.csv')
```

```
1  ticks = [ 1, 2]
2  labels = ["Rowing", "Gymnastics"]
3  plt.boxplot([mens_rowing["Height"],mens_gymnastics["Height"]])
4  plt.xticks(ticks, labels)
5  plt.ylabel("Height (cm)")
6  plt.grid()
7  plt.show()
```

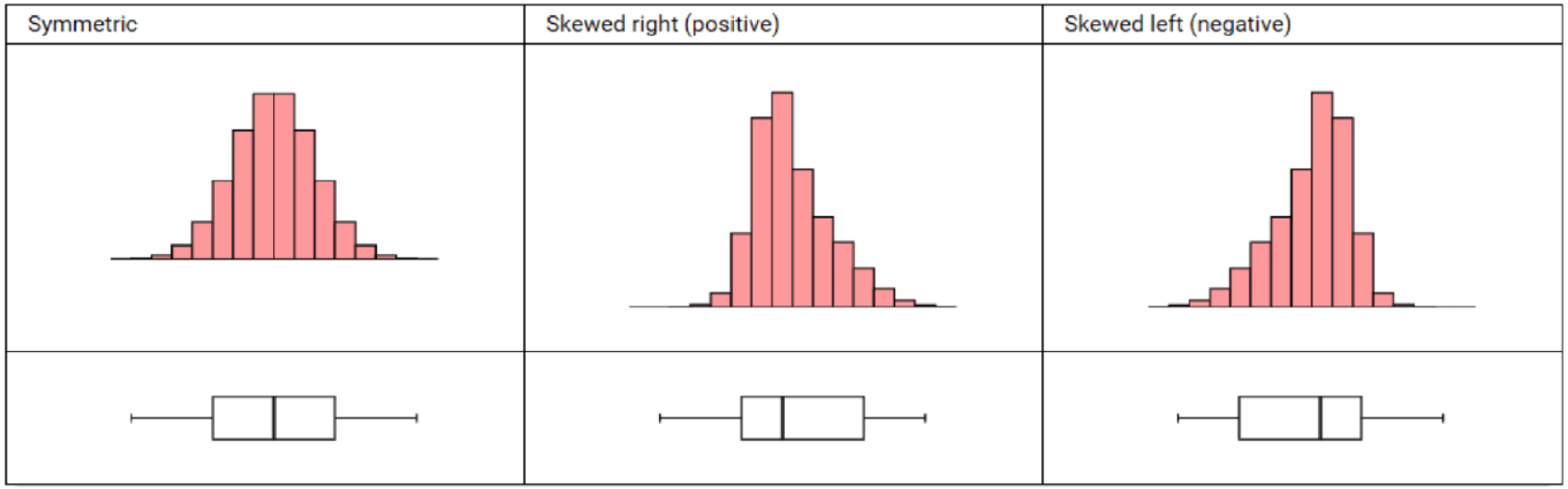
```
/usr/local/lib/python3.7/dist-packages/numpy/core/_asarray.py:83: Visibl
  return array(a, dtype, copy=False, order=order)
```







Boxplots versus mean & median



Olympic medals

```
medals = pd.read_csv('/content/medals_by_country_2016.csv', index_col=0)
medals.head()
```

	Bronze	Gold	Silver
United States	67	137	52
Germany	67	47	43
Great Britain	26	64	55
Russia	35	50	28
China	35	44	30

```
medals.index
```

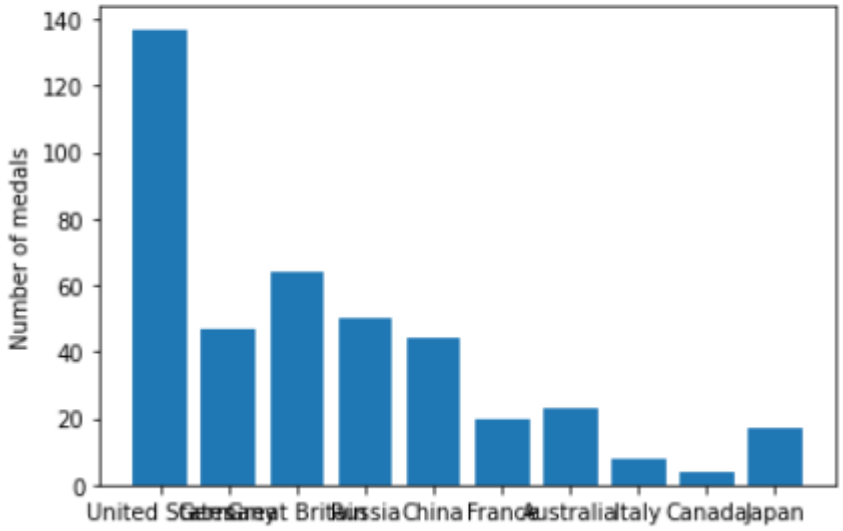
```
Index(['United States', 'Germany', 'Great Britain', 'Russia', 'China',
       'France', 'Australia', 'Italy', 'Canada', 'Japan'],
      dtype='object')
```

```
medals.shape
```

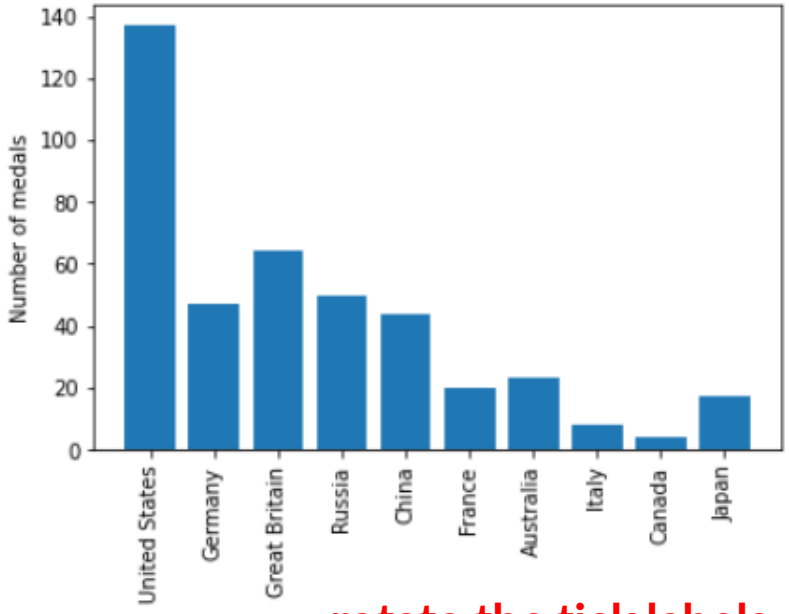
```
(10, 3)
```

Olympic medals: visualizing the data

```
1  plt.bar(medals.index, medals["Gold"])
2  plt.ylabel("Number of medals")
3  plt.show()
```



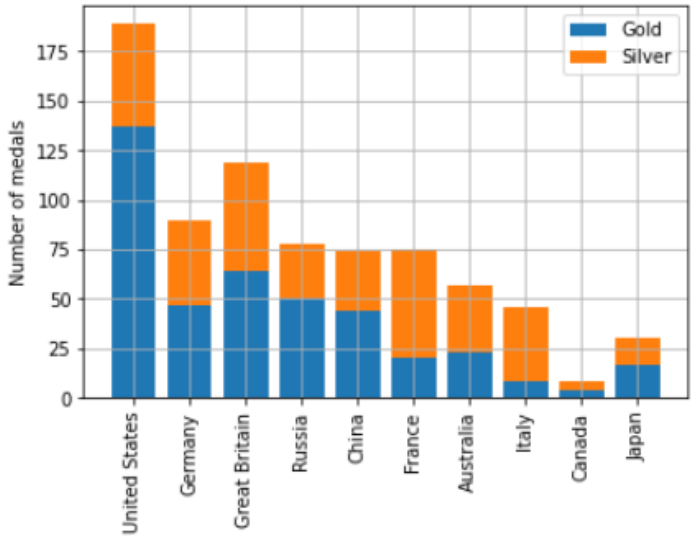

```
1  plt.bar(medals.index, medals["Gold"])
2  plt.xticks(medals.index, rotation=90)
3  plt.ylabel("Number of medals")
4  plt.show()
```



rotate the tick labels

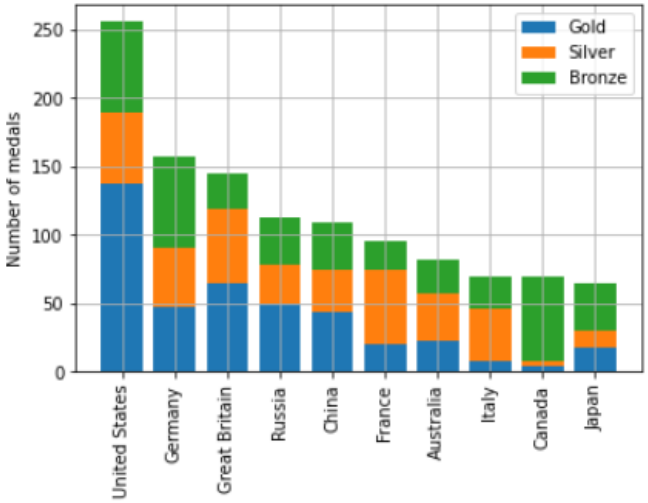
Visualizing the other medals

```
1  plt.bar(medals.index, medals["Gold"],label='Gold')
2  plt.bar(medals.index, medals["Silver"], bottom=medals["Gold"],label='Silver')
3  plt.xticks(medals.index, rotation=90)
4  plt.ylabel("Number of medals")
5  plt.legend()
6  plt.grid()
7  plt.show()
```



Olympic medals: visualizing all three

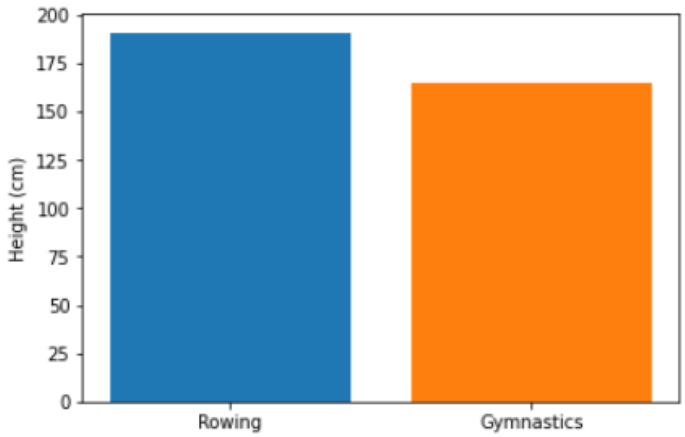
```
1  plt.bar(medals.index, medals["Gold"],label='Gold')
2  plt.bar(medals.index, medals["Silver"], bottom=medals["Gold"],label='Silver')
3  plt.bar(medals.index, medals["Bronze"],bottom=medals["Gold"] + medals["Silver"],label='Bronze')
4  plt.xticks(medals.index, rotation=90)
5  plt.ylabel("Number of medals")
6  plt.legend()
7  plt.grid()
8  plt.show()
```



Comparisons: histograms

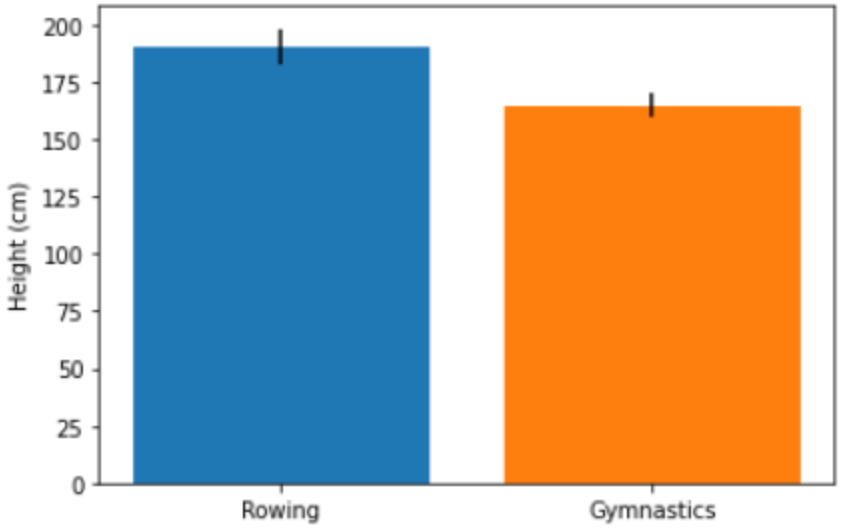
```
mens_rowing = pd.read_csv('/content/mens-rowing.csv')
mens_gymnastics = pd.read_csv('/content/mens_gymnastics.csv')
```

```
plt.bar("Rowing", mens_rowing["Height"].mean())
plt.bar("Gymnastics", mens_gymnastics["Height"].mean())
plt.ylabel("Height (cm)")
plt.show()
```



Adding error bars to bar charts

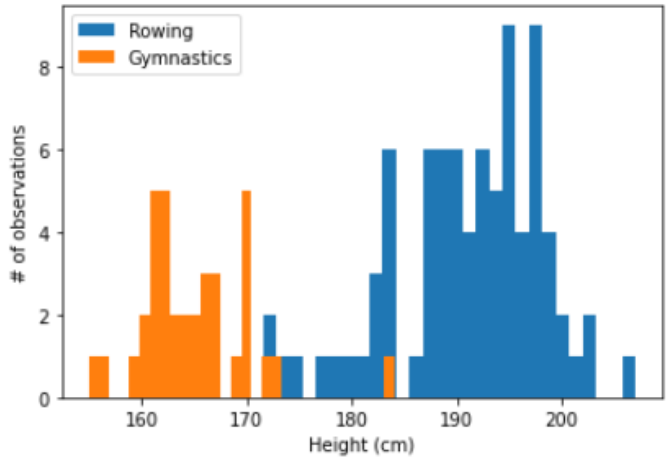
```
1  # Adding error bars to bar charts
2  plt.bar("Rowing",mens_rowing["Height"].mean(),yerr=mens_rowing["Height"].std())
3  plt.bar("Gymnastics",mens_gymnastics["Height"].mean(),yerr=mens_gymnastics["Height"].std())
4  plt.ylabel("Height (cm)")
5  plt.show()
```



Introducing histograms

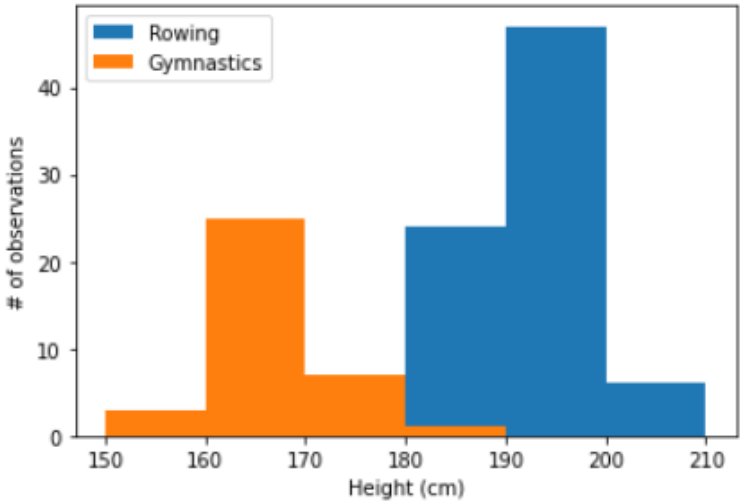
```
mens_rowing = pd.read_csv('/content/mens-rowing.csv')
mens_gymnastics = pd.read_csv('/content/mens_gymnastics.csv')
```

```
# Introducing histograms
plt.hist(mens_rowing["Height"], label="Rowing", bins=30)
plt.hist(mens_gymnastics["Height"], label="Gymnastics", bins=30)
plt.xlabel("Height (cm)")
plt.ylabel("# of observations")
plt.legend()
plt.show()
```



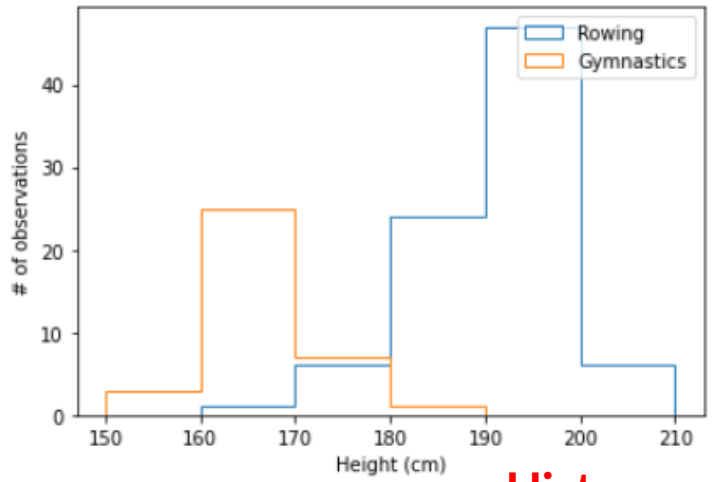
Customizing histograms: setting bin boundaries

```
1  # Introducing histograms (instead of bins)
2  plt.hist(mens_rowing["Height"], label="Rowing", bins=[150, 160, 170, 180, 190, 200, 210])
3  plt.hist(mens_gymnastics["Height"], label="Gymnastics", bins=[150, 160, 170, 180, 190, 200, 210])
4  plt.xlabel("Height (cm)")
5  plt.ylabel("# of observations")
6  plt.legend()
7  plt.show()
```



Customizing histograms: transparency

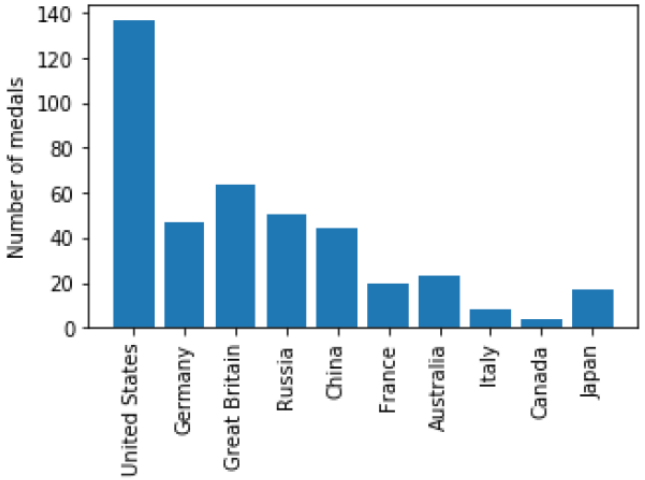
```
1  # Introducing histograms (instead of bins)
2  plt.hist(mens_rowing["Height"], label="Rowing", bins=[150, 160, 170, 180, 190, 200, 210],histtype="step")
3  plt.hist(mens_gymnastics["Height"], label="Gymnastics", bins=[150, 160, 170, 180, 190, 200, 210],histtype="step")
4  plt.xlabel("Height (cm)")
5  plt.ylabel("# of observations")
6  plt.legend()
7  plt.show()
```



Histogram with a histtype of step

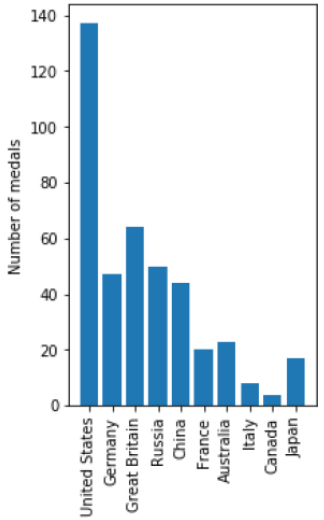
Figure size

```
fig.set_size_inches([5, 3])
```



Another aspect ratio

```
fig.set_size_inches([3, 5])
```



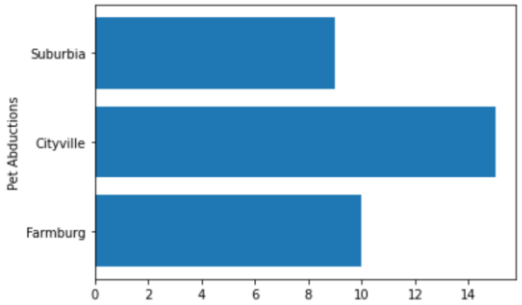
Making a bar chart

```
1  columns = ['precinct', 'pets_abducted']
2  df = pd.DataFrame([['Farmburg', 10],
3                     ['Cityville', 15],['Suburbia', 9]],columns=columns)
4  print(df)
5  df.columns
```

```
    precinct  pets_abducted
0   Farmburg             10
1  Cityville             15
2   Suburbia              9
Index(['precinct', 'pets_abducted'], dtype='object')
```

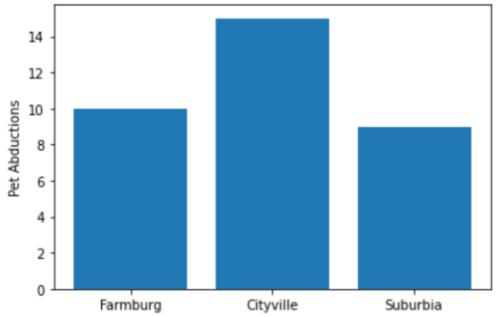
Horizontal bar charts

```
1  plt.barh(df.precinct,df.pets_abducted)
2  plt.ylabel('Pet Abductions')
3  plt.show()
```

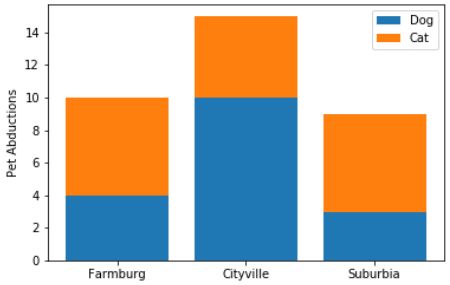


Vertical bar charts

```
1  plt.bar(df.precinct,df.pets_abducted)
2  plt.ylabel('Pet Abductions')
3  plt.show()
```



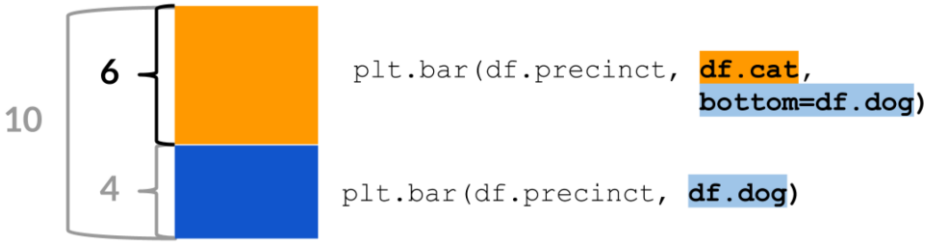
Stacked bar charts

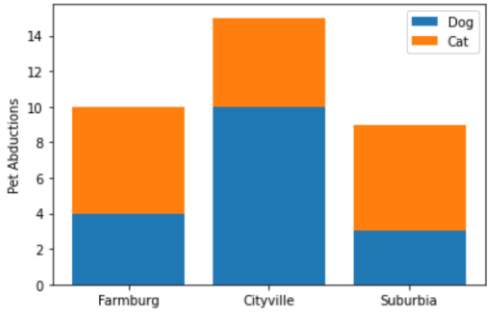



```
1  columns = ['precinct', 'dog','cat']
2  df = pd.DataFrame([['Farmburg', 4,6],
3                     ['Cityville', 10,5],['Suburbia', 3,6]],columns=columns)
4  print(df)
5  df.columns
```

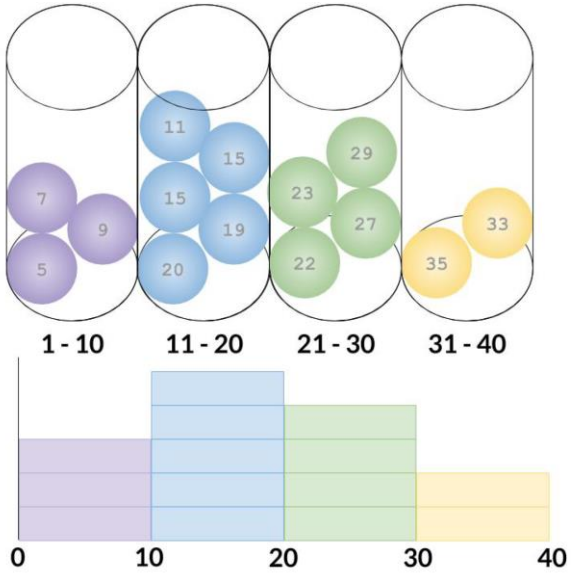
```
   precinct  dog  cat
0   Farmburg    4    6
1  Cityville   10    5
2   Suburbia    3    6
Index(['precinct', 'dog', 'cat'], dtype='object')
```

```
1  plt.bar(df.precinct,df.dog,label='Dog')
2  plt.bar(df.precinct,df.cat,bottom=df.dog,label='Cat')
3  plt.ylabel('Pet Abductions')
4  plt.legend()
5  plt.show()
```

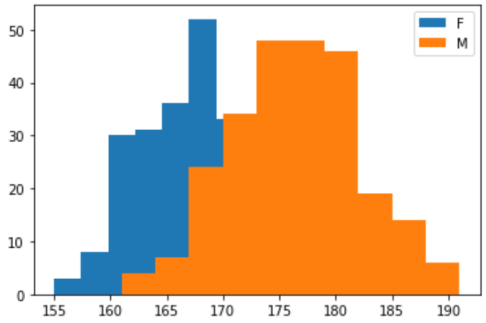




Making a histogram

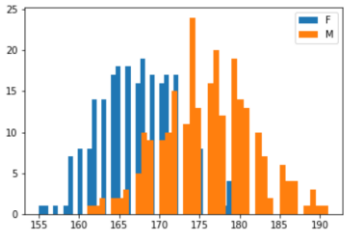



```python
import numpy as np
mu = 168 #mean
sigma = 5 #stddev
sample = 250
np.random.seed(0)
height_f = np.random.normal(mu, sigma, sample).astype(int)
mu = 176 #mean
sigma = 6 #stddev
sample = 250
np.random.seed(1)
height_m = np.random.normal(mu, sigma, sample).astype(int)

gym = pd.DataFrame({'height_f': height_f, 'height_m': height_m})
# gym.plot()
plt.hist(height_f,label='F')
plt.hist(height_m,label='M')
plt.legend()
plt.show()
```



Changing bins

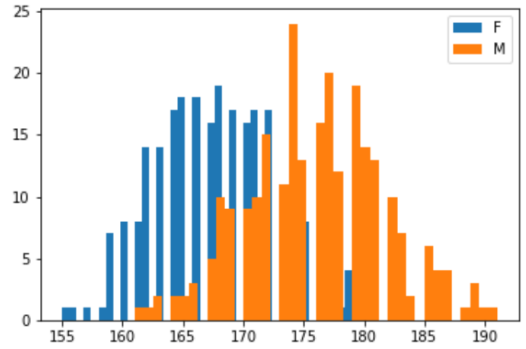
```python
import numpy as np
mu = 168 #mean
sigma = 5 #stddev
sample = 250
np.random.seed(0)
height_f = np.random.normal(mu, sigma, sample).astype(int)
mu = 176 #mean
sigma = 6 #stddev
sample = 250
np.random.seed(1)
height_m = np.random.normal(mu, sigma, sample).astype(int)

gym = pd.DataFrame({'height_f': height_f, 'height_m': height_m})
# plt.hist(data, bins=nbins)
plt.hist(height_f,label='F', bins=40)
plt.hist(height_m,label='M', bins=40)
plt.legend()
plt.show()
```



Normalizing

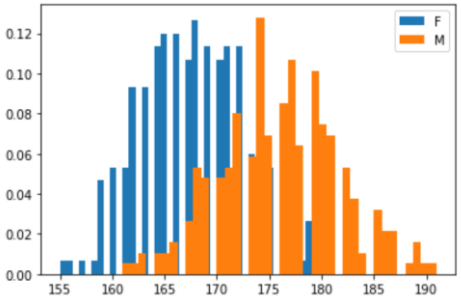
Unnormalized bar plot

```
15  plt.hist(height_f,label='F', bins=40)
16  plt.hist(height_m,label='M', bins=40)
17  plt.legend()
18  plt.show()
```



Normalized bar plot

```
1  plt.hist(height_f,label='F', density=True, bins=40)
2  plt.hist(height_m,label='M', density=True, bins=40)
3  plt.legend()
4  plt.show()
5
6
```

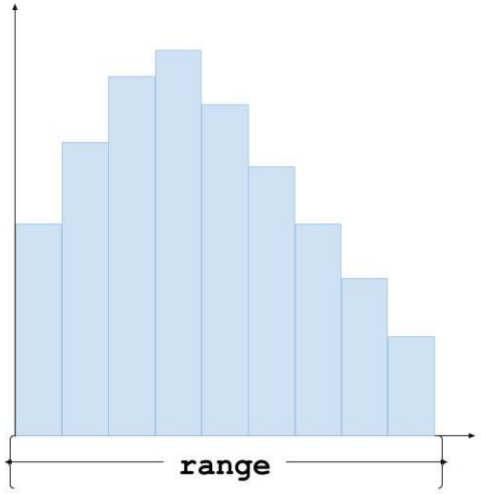


Sum of bar area = 1

Changing range

```
plt.hist(data,
         range=(xmin, xmax))
```

```
plt.hist(gravel.mass,
         range=(50, 100))
```



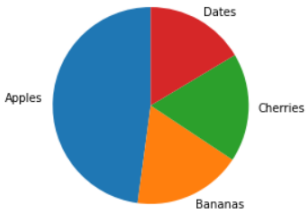
Syntax of pie

```
# plot a Pie Chart for quantitative column with label category column
plt.pie(df.quantitative_column, labels = df.category_column)
```

Creating Pie Charts

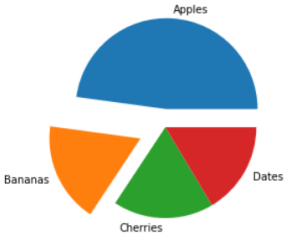
```
import matplotlib.pyplot as plt
import numpy as np
```

```
y = np.array([335, 125, 125, 115])
mylabels = ["Apples", "Bananas", "Cherries", "Dates"]

plt.pie(y, labels = mylabels, startangle = 90)
plt.show()
```



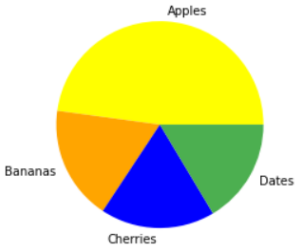
Explode

```
myexplode = [0.2, 0.3, 0, 0]
plt.pie(y, labels = mylabels, explode = myexplode)
plt.show()
```



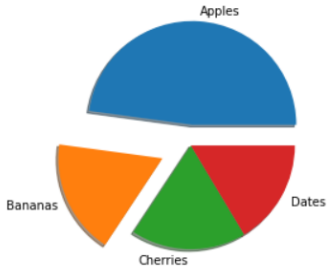
Colors

```
mycolors = ["yellow", "orange", "b", "#4CAF50"]

plt.pie(y, labels = mylabels, colors = mycolors)
plt.show()
```



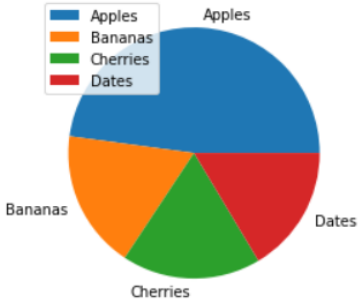
Shadow

```
plt.pie(y, labels = mylabels, explode = myexplode, shadow = True)
plt.show()
```



Legend

```
plt.pie(y, labels = mylabels)
plt.legend(loc='upper left')
plt.show()
```

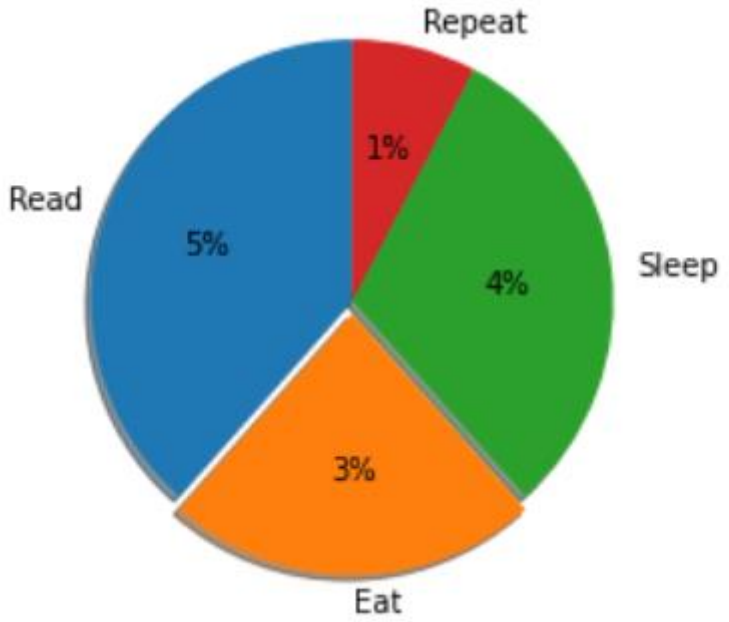


Legend With Header

```
plt.pie(y, labels = mylabels)
plt.legend(loc='upper left',title = "Four Fruits:")
plt.show()
```

How to have actual values in Matplotlib Pie Chart displayed?

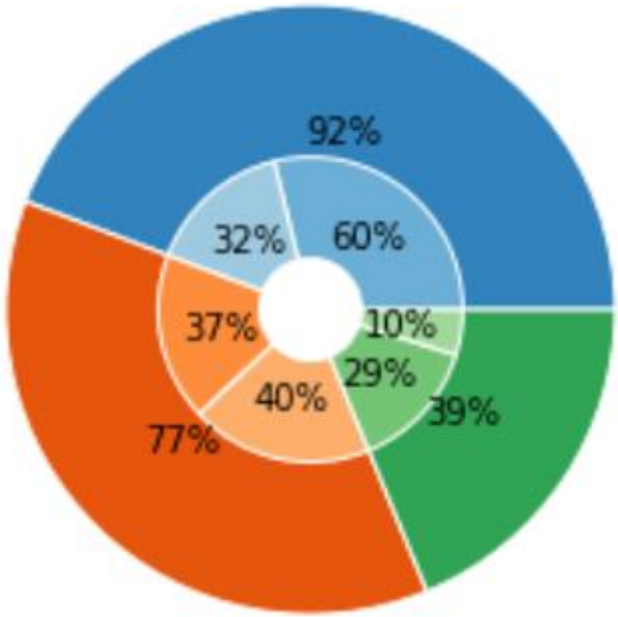
```
1  import matplotlib.pyplot as plt
2  labels = ('Read', 'Eat', 'Sleep', 'Repeat')
3  fracs = [5, 3, 4, 1]
4  total = sum(fracs)
5  explode = (0, 0.05, 0, 0)
6  plt.pie(fracs, explode=explode, labels=labels,
7  autopct=lambda p: '{:.0f}%'.format(p * total / 100),
8  shadow=True, startangle=90)
```



Make a pie chart using **labels, fracs** and **explode** with **autopct=lambda p: <calculation for percentage>.**

How to plot a nested pie chart in Matplotlib?

```
fig, ax = plt.subplots()
size = 0.5
vals = np.array([[60., 32.], [37., 40.], [29., 10.]])
cmap = plt.get_cmap("tab20c")
outer_colors = cmap(np.arange(3)*4)
inner_colors = cmap([1, 2, 5, 6, 9, 10])
freq1=vals.sum(axis=1)
freq2=vals.flatten()
total1=sum(freq1)
total2=sum(freq2)
ax.pie(freq1, radius=1, colors=outer_colors,
       wedgeprops=dict(width=size/0.9, edgecolor='w'),
       autopct=lambda p: '{:.0f}%'.format(p * total1 / 100))
ax.pie(freq2, radius=1-size, colors=inner_colors,
       wedgeprops=dict(width=size/1.5, edgecolor='w'),
       autopct=lambda p: '{:.0f}%'.format(p * total2 / 100))
```



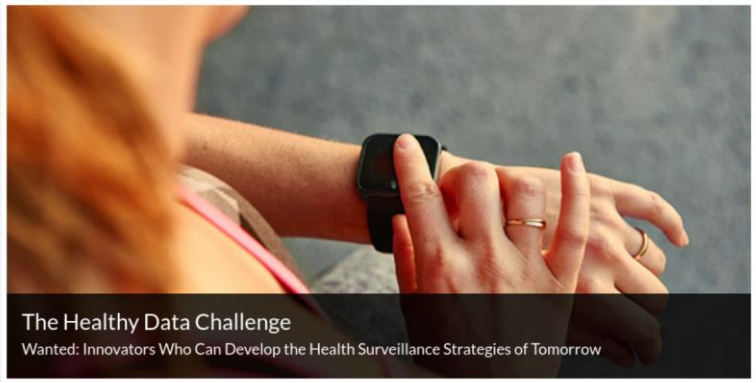
Height and weight

Centers for Disease Control and Prevention
CDC 24/7: Saving Lives, Protecting People™

SEARCH

CDC A-Z INDEX

Behavioral Risk Factor Surveillance System

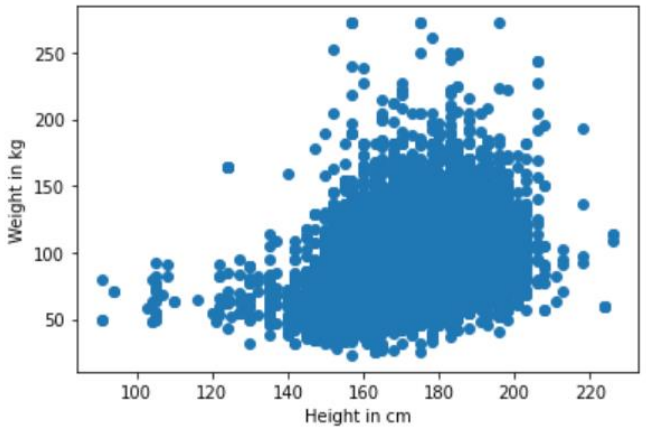


BRFSS™

The Behavioral Risk Factor Surveillance System (BRFSS) is the nation's premier system of health-related telephone surveys that collect state data about U.S. residents regarding their health-related risk behaviors, chronic health conditions, and use of preventive services. Established in 1984 with 15 states, BRFSS now collects data in all 50 states as well as the District of Columbia and three U.S. territories. BRFSS completes more than 400,000 adult interviews each year, making it the largest continuously conducted health survey system in the world. See More.

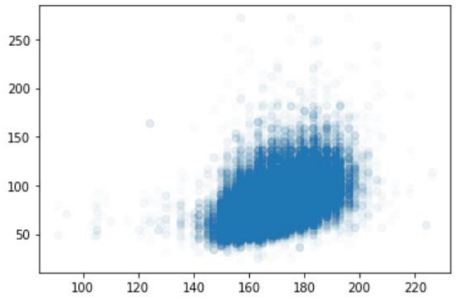
Weight vs height

```
# Scatter plot
brfss = pd.read_hdf('brfss.hdf5', 'brfss')
height = brfss['HTM4']
weight = brfss['WTKG3']
plt.plot(height, weight, 'o')
plt.xlabel('Height in cm')
plt.ylabel('Weight in kg')
plt.show()
```

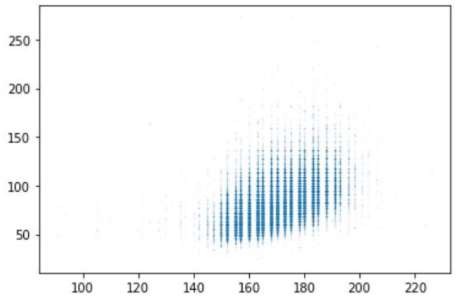


Transparency + marker size

```
1  plt.plot(height, weight, 'o', alpha=0.02)
2  plt.show()
```

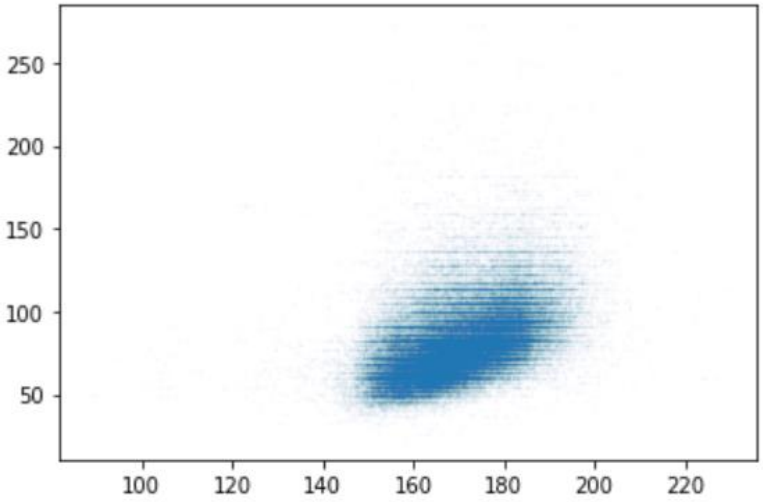



```
1  # Marker size
2  plt.plot(height, weight, 'o', markersize=1,alpha=0.02)
3  plt.show()
```



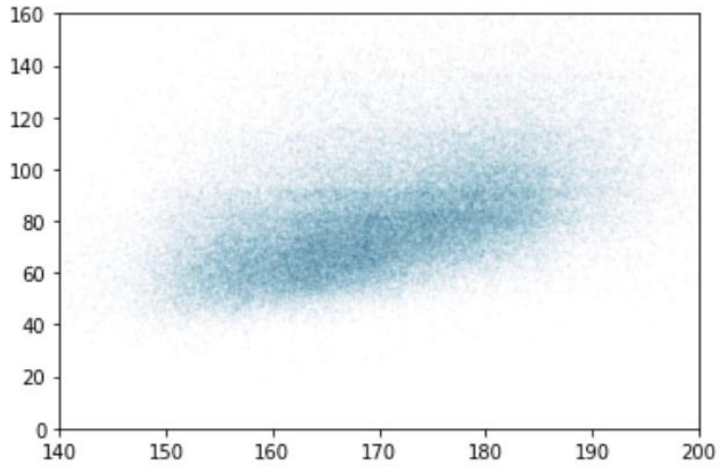
Jittering

```
1   height_jitter = height + np.random.normal(0, 2, size=len(brfss))
2   plt.plot(height_jitter, weight, 'o', markersize=1, alpha=0.02)
3   plt.show()
```



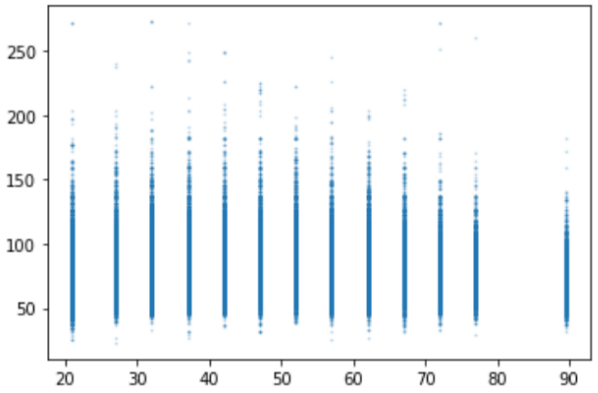
Zoom

```
1  # Zoom
2  plt.plot(height_jitter, weight_jitter, 'o', markersize=1, alpha=0.02)
3  plt.axis([140, 200, 0, 160])
4  plt.show()
```

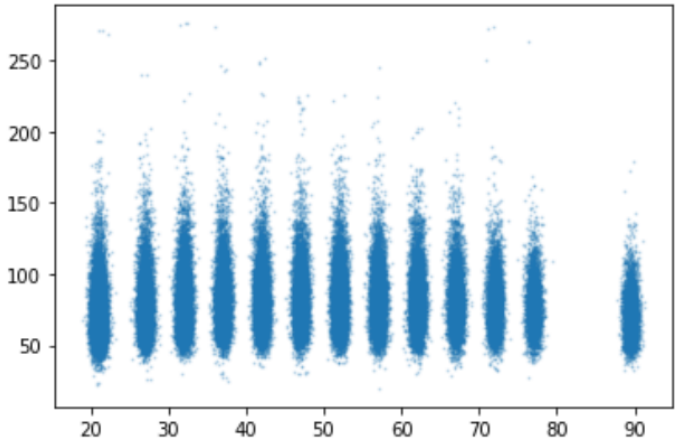


Weight and age

```
1  age = brfss['AGE']
2  weight = brfss['WTKG3']
3  plt.plot(age, weight, 'o', markersize=1, alpha=0.2)
4  plt.show()
```

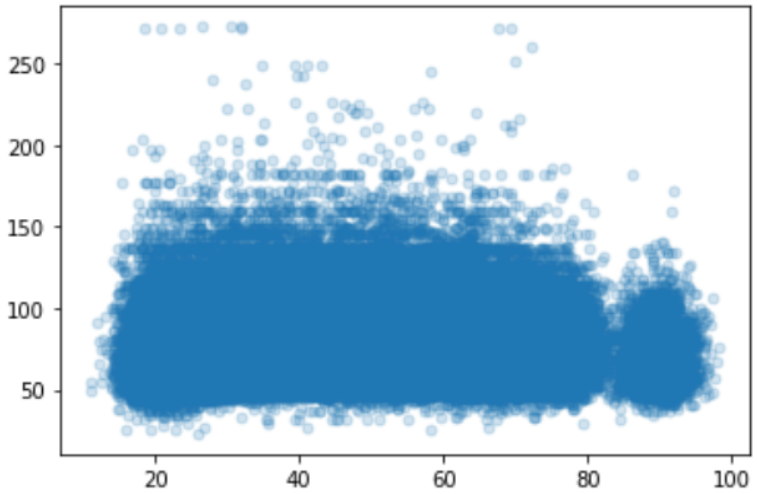



```
1  age = brfss['AGE'] + np.random.normal(0, 0.5, size=len(brfss))
2  weight = brfss['WTKG3'] + np.random.normal(0, 2, size=len(brfss))
3  plt.plot(age, weight, 'o', markersize=1, alpha=0.2)
4  plt.show()
```



Weight vs noisy age

```
1  age = brfss['AGE'] + np.random.normal(0, 2.5, size=len(brfss))
2  weight = brfss['WTKG3']
3  plt.plot(age, weight, 'o', markersize=5, alpha=0.2)
4  plt.show()
```



Log scale

```
plt.yscale('log')
```

INTERMEDIATE DATA MANIPULATION & VISUALIZATION

INTRODUCTION TO SEABORN

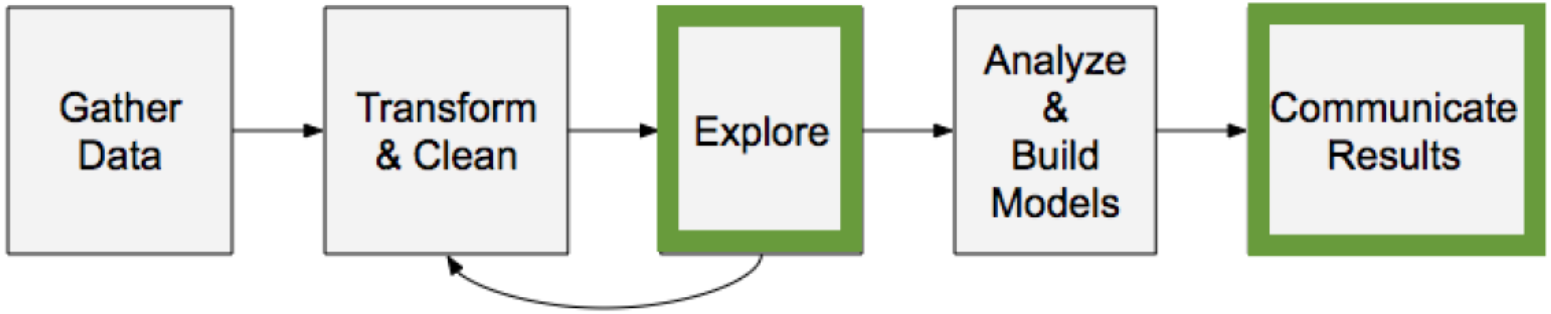
What is Seaborn?

- Python data visualization library
- Easily create the most common types of plots

Advantages of Seaborn

- Easy to use
- Works well with pandas data structures
- Built on top of matplotlib

Why is Seaborn useful?



Relational plots

- Show the relationship between two quantitative variables
  - Examples: scatter plots, line plots

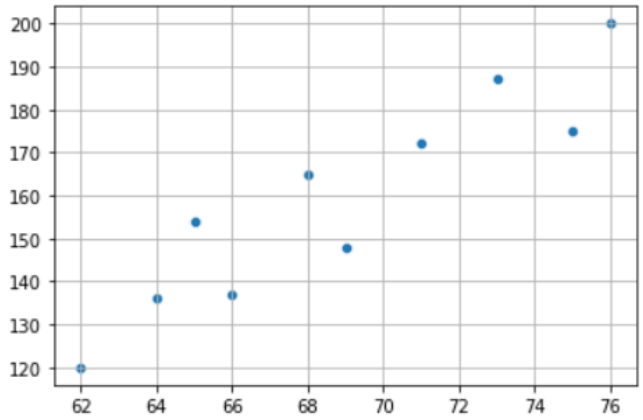
Categorical plots

- Show the distribution of a quantitative variable within categories defined by a categorical variable
  - Examples: bar plots, count plots, box plots, point plots

Example 1: Scatter plot

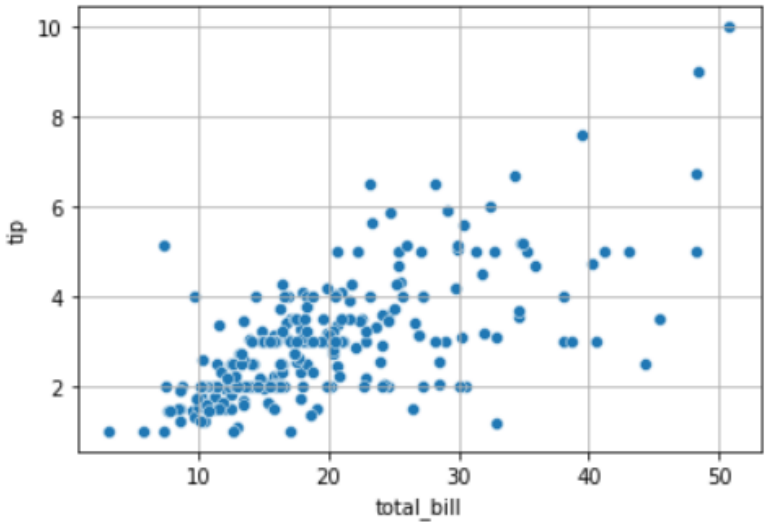
```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
```

```
1  height = [62, 64, 69, 75, 66, 68, 65, 71, 76, 73]
2  weight = [120, 136, 148, 175, 137,165, 154, 172, 200, 187]
3  sns.scatterplot(x=height, y=weight)
4  plt.grid()
5  plt.show()
```



A basic scatter plot

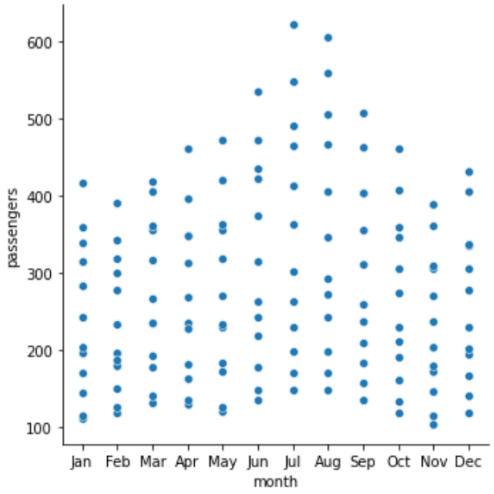
```
1  import matplotlib.pyplot as plt
2  import seaborn as sns
3  tips = sns.load_dataset("tips")
4  sns.scatterplot(x="total_bill",y="tip",data=tips)
5  plt.grid()
6  plt.show()
```



Scatter Plot (seaborn)

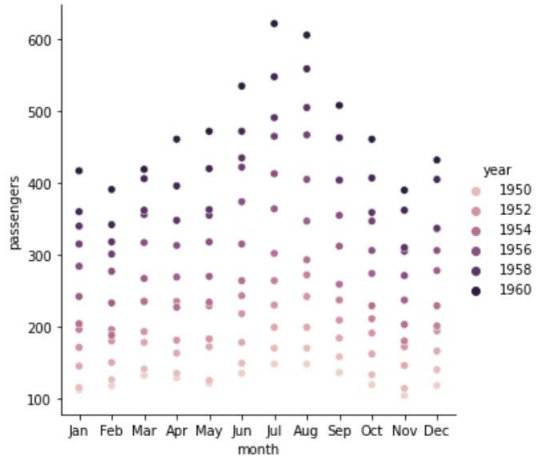
```
1  # Scatter Plot
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  import numpy as np
6  a = sns.load_dataset("flights")
7  sns.relplot(x = "month",y = "passengers",data =a ) # scatter plot
```

```
<seaborn.axisgrid.FacetGrid at 0x7f06c7f1f550>
```




```
1  # Scatter Plot
2  import seaborn as sns
3  import matplotlib.pyplot as plt
4  import pandas as pd
5  import numpy as np
6  a = sns.load_dataset("flights")
7  sns.relplot(x = "month",y = "passengers",hue="year",data =a ) # scatter plot
```

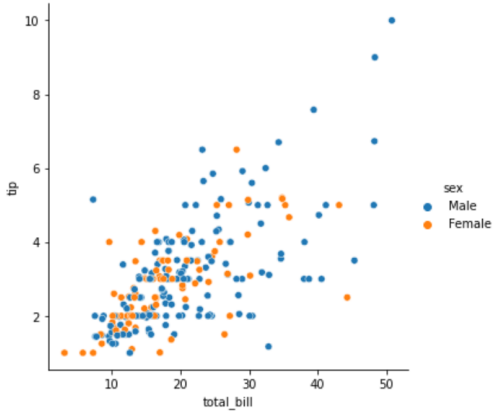
```
<seaborn.axisgrid.FacetGrid at 0x7f06c7ea8550>
```



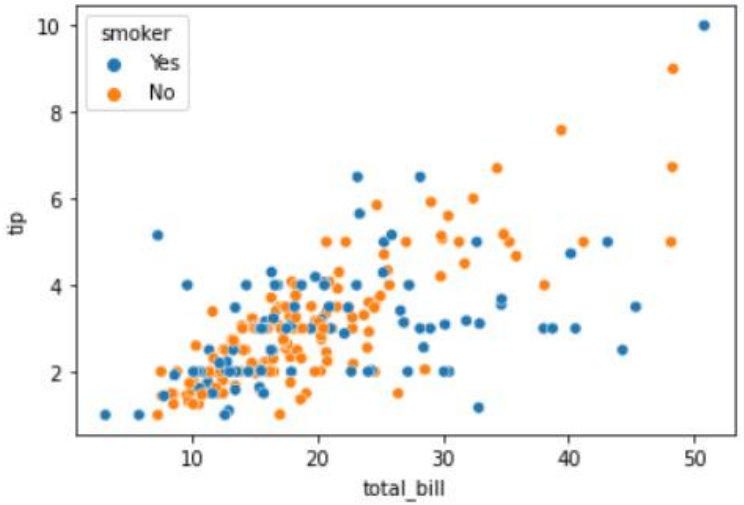
A scatter plot with hue

```
1  b = sns.load_dataset("tips")
2  print(b)
3  sns.relplot(x="total_bill", y="tip",data = b,hue='sex')
```

```
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
<seaborn.axisgrid.FacetGrid at 0x7f06c054b510>
```

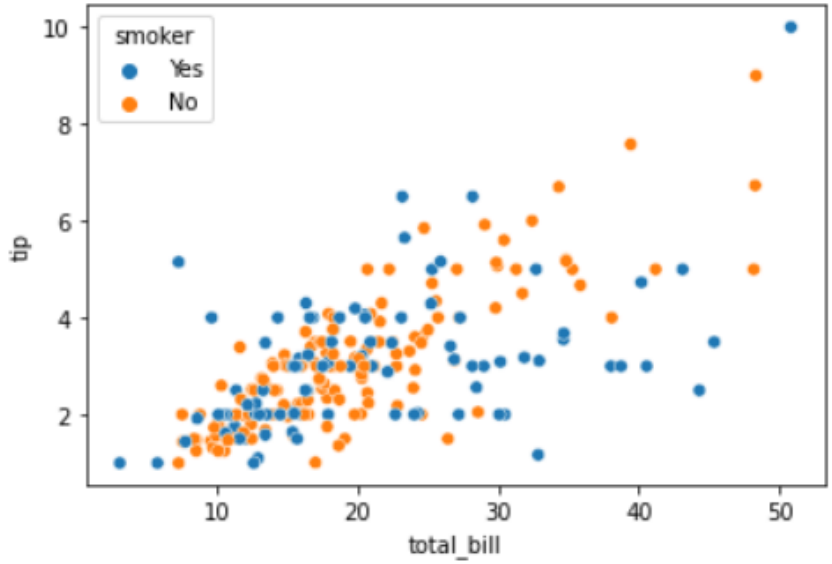



```
1  sns.scatterplot(x="total_bill",y="tip",data=tips,hue="smoker")
2  plt.show()
```



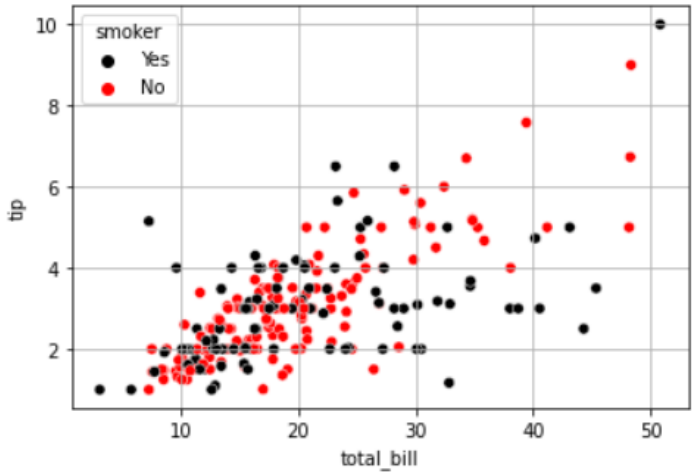
Setting hue order

```
1   sns.scatterplot(x="total_bill",y="tip",data=tips, hue="smoker",hue_order=["Yes","No"])
2   plt.show()
```



Specifying hue colors

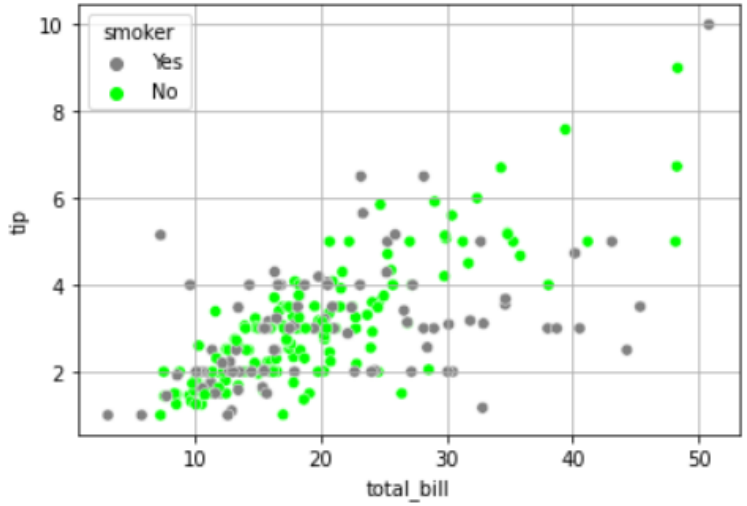
```
1  hue_colors = {"Yes": "black", "No": "red"}
2  sns.scatterplot(x="total_bill",y="tip",data=tips,hue="smoker", palette=hue_colors)
3  plt.grid()
4  plt.show()
```



	Color	Matplotlib name	Matplotlib abbreviation	HTML color code (hex)
	blue	"blue"	"b"	#0000ff
	green	"green"	"g"	#008000
	red	"red"	"r"	#ff0000
	green/blue	"cyan"	"c"	#00bfbf
	purple	"magenta"	"m"	#bf00bf
	yellow	"yellow"	"y"	#bfbf00
	black	"black"	"k"	#000000
	white	"white"	"w"	#ffffff

Using HTML hex color codes with hue

```
1  hue_colors = {"Yes": "#808080","No": "#00FF00"}
2  sns.scatterplot(x="total_bill",y="tip",data=tips,hue="smoker",palette=hue_colors)
3  plt.grid()
4  plt.show()
```



Categorical plots

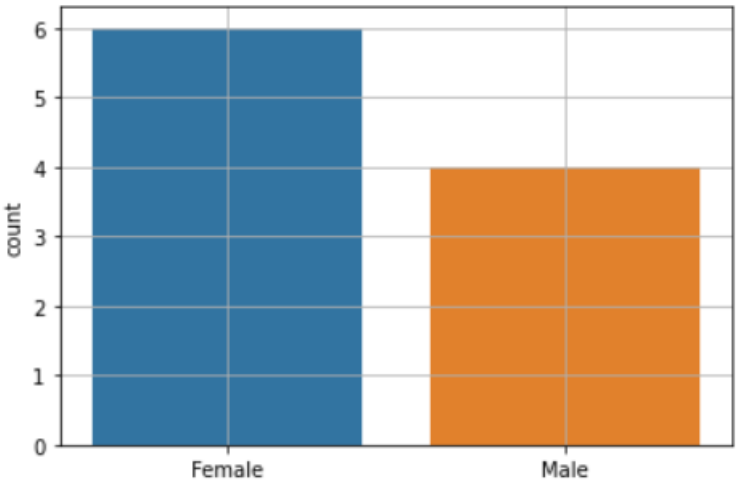
Examples: count plots, bar plots

Involve a categorical variable

Comparisons between groups

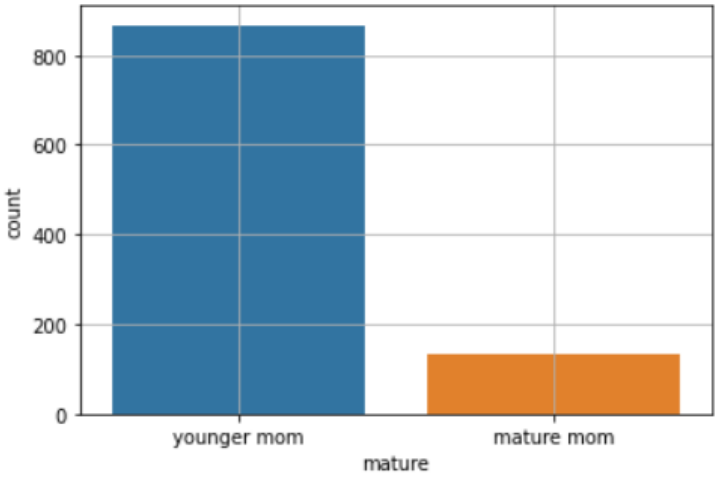
Example 2: Create a count plot

```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3  gender = ["Female", "Female","Female", "Female","Female", "Female", "Male","Male", "Male", "Male"]
4  sns.countplot(x=gender)
5  plt.grid()
6  plt.show()
```



Using DataFrames with countplot()

```
1  import pandas as pd
2  import matplotlib.pyplot as plt
3  import seaborn as sns
4  df = pd.read_csv("/content/ncbirths.csv")
5  sns.countplot(x="mature",data=df)
6  plt.grid()
7  plt.show()
```

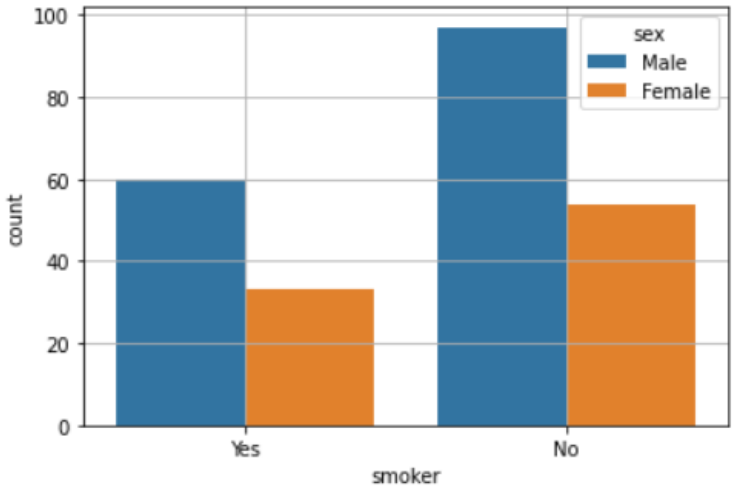



```
1  df.head()
```

	fage	mage	mature	weeks	premie	visits	marital	gained	weight	lowbirthweight	gender	habit	whitemom
0	NaN	13	younger mom	39.0	full term	10.0	not married	38.0	7.63	not low	male	nonsmoker	not white
1	NaN	14	younger mom	42.0	full term	15.0	not married	20.0	7.88	not low	male	nonsmoker	not white
2	19.0	15	younger mom	37.0	full term	11.0	not married	38.0	6.63	not low	female	nonsmoker	white
3	21.0	15	younger mom	41.0	full term	6.0	not married	34.0	8.00	not low	male	nonsmoker	white
4	NaN	15	younger mom	39.0	full term	9.0	not married	27.0	6.38	not low	female	nonsmoker	not white

Using hue with count plots

```
1  sns.countplot(x="smoker",data=tips,hue="sex")
2  plt.grid()
3  plt.show()
```



Introducing relplot()

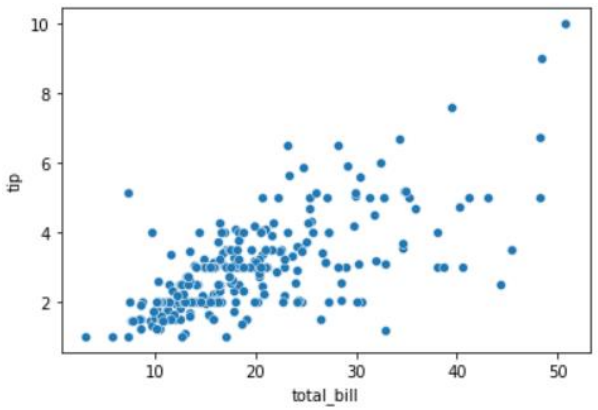
Create "relational plots": scatter plots or line plots

Show relationship between two quantitative variables

- Why use relplot() instead of scatterplot() ?
  - relplot() lets you create subplots in a single figure

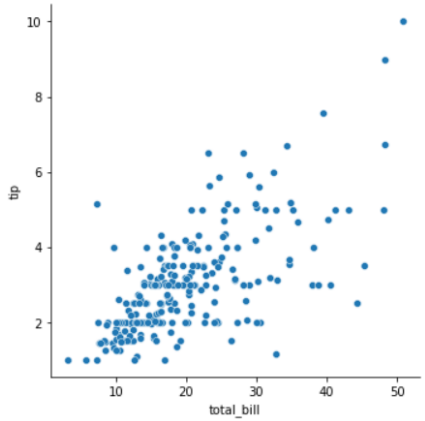
Using scatterplot()

```
1  sns.scatterplot(x="total_bill",y="tip",data=tips)
2  plt.show()
```



Using relplot()

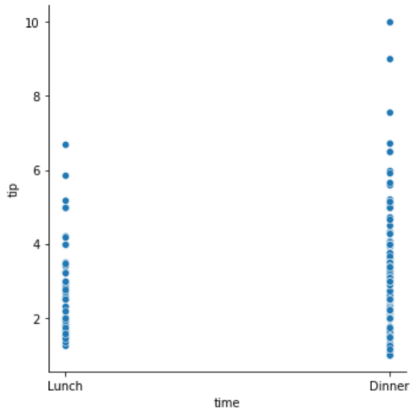
```
1  sns.relplot(x="total_bill",y="tip",data=tips,kind="scatter")
2  plt.show()
```



Relational plots

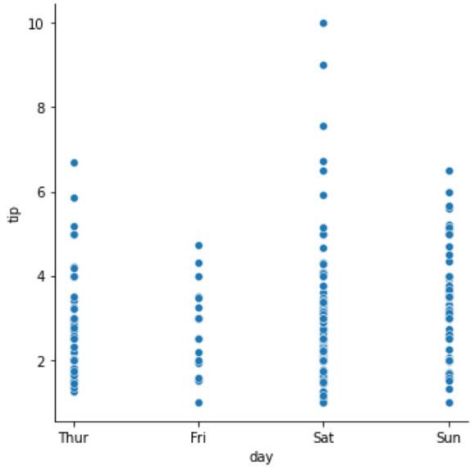
```
1  b = sns.load_dataset("tips")
2  print(b)
3  sns.relplot(x ="time",y = "tip",data = b)
```

```
     total_bill   tip     sex smoker   day    time  size
0         16.99  1.01  Female     No   Sun  Dinner     2
1         10.34  1.66    Male     No   Sun  Dinner     3
2         21.01  3.50    Male     No   Sun  Dinner     3
3         23.68  3.31    Male     No   Sun  Dinner     2
4         24.59  3.61  Female     No   Sun  Dinner     4
..          ...   ...     ...    ...   ...     ...   ...
239       29.03  5.92    Male     No   Sat  Dinner     3
240       27.18  2.00  Female    Yes   Sat  Dinner     2
241       22.67  2.00    Male    Yes   Sat  Dinner     2
242       17.82  1.75    Male     No   Sat  Dinner     2
243       18.78  3.00  Female     No  Thur  Dinner     2

[244 rows x 7 columns]
<seaborn.axisgrid.FacetGrid at 0x7f06c36dc790>
```



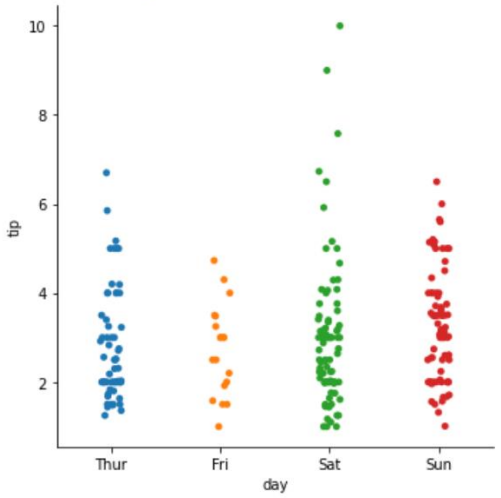

```
1  sns.relplot(x ="day",y = "tip",data = b)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f06c35c85d0>
```



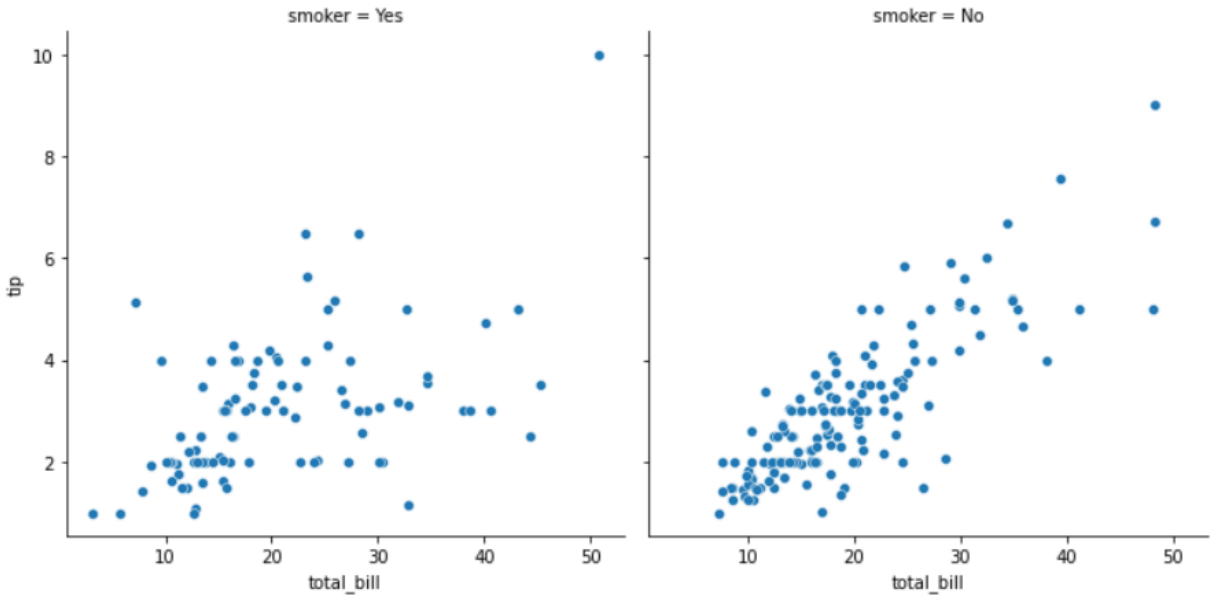

```
1  sns.catplot(x ="day",y = "tip",data = b)
```

```
<seaborn.axisgrid.FacetGrid at 0x7f06c3533450>
```



Subplots in columns

```
1  import seaborn as sns
2  import matplotlib.pyplot as plt
3  sns.relplot(x="total_bill",y="tip",data=tips,kind="scatter",col="smoker")
4  plt.show()
```



Subplots in rows

```
1  sns.relplot(x="total_bill",y="tip",data=tips,kind="scatter",row="smoker")
2  plt.show()
```