Computer Science

CSC380: Principles of Data Science

Data Analysis, Collection, and Visualization 1

Xinchen Yu

Announcements

- HW01 was out (due next Wednesday, Jan 28 by 11:59pm)
- Lecture participation self-report form in course website
- Office hours start tomorrows

Today's plan

- Basic data processing using Pandas (after-class)
- Descriptive statistics using Pandas
- Basic data visualization

Pandas

Pandas

Open source library for data handling and manipulation in high-performance environments.

Installation If you are using Anaconda package manager,

```
conda install pandas
```

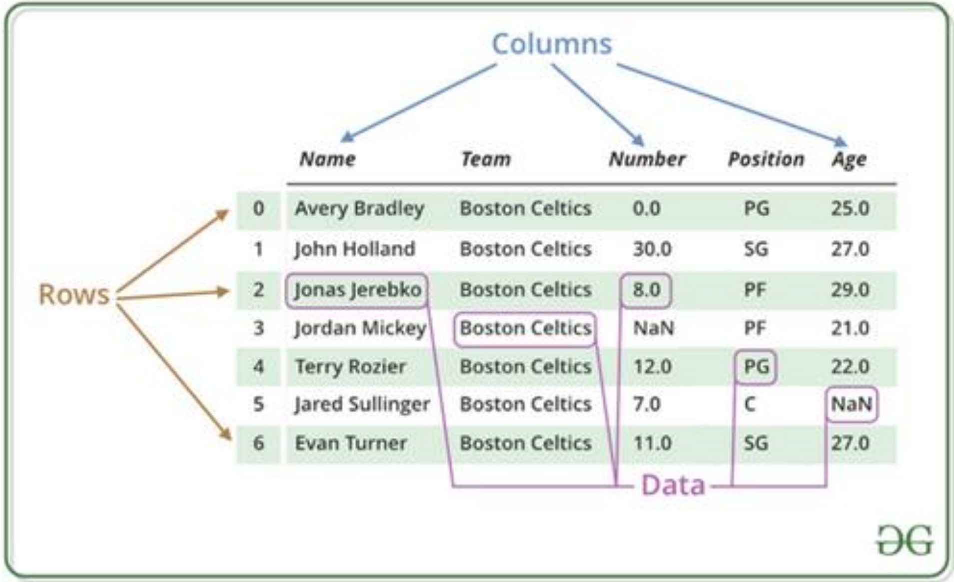
Or if you are using PyPi (pip) package manager,

```
pip install pandas
```

See Pandas documentation for more detailed instructions
https://pandas.pydata.org/docs/getting_started/install.html

DataFrame

Primary data structure : Essentially a table



Q: how is it different from an array?

```
array([[30, 32, 35],
       [40, 42, 45],
       [50, 52, 55]])
```

- Dataframes' elements' data types can be mixed; an array usually store elements of same type

DataFrame Example

Create and print an entire DataFrame

```python
# import pandas as pd
import pandas as pd

# list of strings
lst = ['Geeks', 'For', 'Geeks', 'is',
            'portal', 'for', 'Geeks']

# Calling DataFrame constructor on list
df = pd.DataFrame(lst)
print(df)
```

	0
0	Geeks
1	For
2	Geeks
3	is
4	portal
5	for
6	Geeks

DataFrame Example

Can create *named columns* using dictionary

```
import pandas as pd

# intialise data of lists.
data = {'Name':['Tom', 'nick', 'krish', 'jack'],
        'Age':[20, 21, 19, 18]}

# Create DataFrame
df = pd.DataFrame(data)

# Print the output.
print(df)
```

	Name	Age
0	Tom	20
1	nick	21
2	krish	19
3	jack	18

all data must have the same length

DataFrame : Selecting Columns

Select columns to print by name

```
# Import pandas package
import pandas as pd

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# select two columns
print(df[['Name', 'Qualification']])
```

	Name	Qualification
0	Jai	Msc
1	Princi	MA
2	Gaurav	MCA
3	Anuj	Phd

access columns by name, not the column index!

DataFrame : Selecting Columns

[35]:
```
import pandas as pd
data = {'Name': ['tom', 'nick'], 'Age': [10,20]}
df = pd.DataFrame(data)
```

[36]:
```
df[['Name']]
```

[36]:

	Name
0	tom
1	nick

[37]:
```
df['Name']
```

[37]:
```
0     tom
1    nick
Name: Name, dtype: object
```

[38]:
```
type(df[['Name']]), type(df['Name'])
```

[38]:
```
(pandas.core.frame.DataFrame, pandas.core.series.Series)
```

still a DataFrame

essentially, a DataFrame's single row or column

pandas.Series

class pandas.Series(*data=None*, *index=None*, *dtype=None*, *name=None*, *copy=False*, *fastpath=False*) [source]

One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object supports both integer- and label-based indexing and provides a host of methods for performing operations involving the index. Statistical methods from ndarray have been overridden to automatically exclude missing data (currently represented as NaN).

DataFrame : Selecting Rows

Use df.loc to access certain rows

```
import pandas as pd
import numpy as np

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Print rows 1 & 2
row = df.loc[1:2]
print(row)
```

Output

```
     Name  Age    Address Qualification
1  Princi   24     Kanpur            MA
2  Gaurav   22  Allahabad           MCA
```

(still a DataFrame)

1:2 includes 2! This is different from Python array indexing

DataFrame : Selecting Rows


```
[6]: import pandas as pd
     data = {'Name': ['tom', 'nick'], 'Age': [10,20]}
     df = pd.DataFrame(data)

[19]: df.loc[1:1]
```

[19]:

	Name	Age
1	nick	20

```
[20]: df.loc[1]

[20]: Name    nick
      Age       20
      Name: 1, dtype: object

[21]: type(df.loc[1:1]), type(df.loc[1])

[21]: (pandas.core.frame.DataFrame, pandas.core.series.Series)
```

- df.loc[1:1] is DataFrame object
- df.loc[1] is a series

DataFrame : Selecting Rows

`head()` and `tail()` select rows from beginning / end

handy when we would like to get a sense of what a big table looks like

```
import pandas as pd
import numpy as np

# Define a dictionary containing employee data
data = {'Name':['Jai', 'Princi', 'Gaurav', 'Anuj'],
        'Age':[27, 24, 22, 32],
        'Address':['Delhi', 'Kanpur', 'Allahabad', 'Kannauj'],
        'Qualification':['Msc', 'MA', 'MCA', 'Phd']}

# Convert the dictionary into DataFrame
df = pd.DataFrame(data)

# Print first / last rows
first2 = df.head(2)
last2 = df.tail(2)
print(first2)
print('\n', last2)
```

Output

```
     Name  Age Address Qualification
0     Jai   27   Delhi           Msc
1  Princi   24  Kanpur            MA

     Name  Age    Address Qualification
2  Gaurav   22  Allahabad           MCA
3    Anuj   32    Kannauj           Phd
```

Reading Data from Files

Easy reading / writing of standard formats

```
df = pd.read_json("data.json")
print(df)
df.to_csv("data.csv", index=False)
df_csv = pd.read_csv("data.csv")
print(df_csv.head(2))
```

index ↓

Output

```
     Duration  Pulse  Maxpulse  Calories
0          60    110       130     409.1
1          60    117       145     479.0
2          60    103       135     340.0
3          45    109       175     282.4
4          45    117       148     406.0
..        ...    ...       ...       ...
164        60    105       140     290.8
165        60    110       145     300.4
166        60    115       145     310.2
167        75    120       150     320.4
168        75    125       150     330.4

[169 rows x 4 columns]
   Duration  Pulse  Maxpulse  Calories
0        60    110       130     409.1
1        60    117       145     479.0
```

Json format: e.g. X(twitter) API

```
{
    "fruits": ["apple", "banana", "cherry"],
    "numbers": [1, 2, 3, 4],
    "mixed": [true, "hello", null]
}
```

CSV format (comma separated values)

```
Name,Age,City
Alice,25,New York
Bob,30,San Francisco
Charlie,22,Chicago
```

Data Type Conversions

Working with DataFrames outside of Pandas can be tricky

```
df['Duration']
```

```
0      60
1      60
2      60
3      45
4      45
       ..
164    60
165    60
166    60
167    75
168    75
Name: Duration, Length: 169, dtype: int64
```

Q: is this a DataFrame object or Series object?

A: a Series object

We can easily convert a Series to built-in types, e.g., a list.

```
L = df['Duration'].to_list()
print(L)
```

```
[60, 60, 60, 45, 45, 60, 60, 45, 30, 60, 60, 60, 60, 60, 60, 60, 60, 45, 60, 45, 60, 45, 60, 45, 60, 60, 60, 60, 60,
60, 60, 45, 60, 60, 60, 60, 60, 60, 60, 45, 45, 60, 60, 60, 60, 60, 60, 45, 45, 60, 60, 80, 60, 60, 30, 60, 60, 45, 2
0, 45, 210, 160, 160, 45, 20, 180, 150, 150, 20, 300, 150, 60, 90, 150, 45, 90, 45, 45, 120, 270, 30, 45, 30, 120, 4
5, 30, 45, 120, 45, 20, 180, 45, 30, 15, 20, 20, 30, 25, 30, 90, 20, 90, 90, 90, 30, 30, 180, 30, 90, 210, 60, 45, 1
5, 45, 60, 60, 60, 60, 60, 60, 30, 45, 60, 60, 60, 60, 60, 60, 90, 60, 60, 60, 60, 60, 60, 20, 45, 45, 45, 20, 60, 6
0, 45, 45, 60, 45, 60, 60, 30, 60, 60, 60, 60, 30, 60, 60, 60, 60, 60, 30, 30, 45, 45, 45, 60, 60, 60, 75, 75]
```

Data Type Conversions

Or, to a numpy array

```
[6]: import pandas as pd
     data = {'Name': ['tom', 'nick'], 'Age': [10,20]}
     df = pd.DataFrame(data)
```

```
[29]: df
```

[29]:

	Name	Age
0	tom	10
1	nick	20

```
[31]: df.to_numpy()

[31]: array([['tom', 10],
             ['nick', 20]], dtype=object)
```

```
[40]: df['Name'].to_numpy()

[40]: array(['tom', 'nick'], dtype=object)
```

`to_numpy()`: can take `Series` and `DataFrame` objects as input

Numpy: Python library for scientific computing

Summary Statistics in Pandas

Compute summary statistics on Pandas Series

```
print('Min: ', df['Duration'].min())
print('Max: ', df['Duration'].max())
print('Median: ', df['Duration'].median())
```

```
Min:  15
Max:  300
Median:  60.0
```

Can also count occurrences of unique values,

```
df['Duration'].value_counts()
```

```
60      79
45      35
30      16
20       9
90       8
150      4
120      3
180      3
15       2
75       2
160      2
210      2
270      1
25       1
300      1
80       1
Name: Duration, dtype: int64
```

s = df['Duration'].value_counts()
s[60]=79.

Summary Statistics

Compute summary statistics on each column of Dataframe

[42]:
```
import pandas as pd
data = {'Name': ['tom', 'nick'], 'Age': [10,20], 'Height': [6.2, 5.5]}
df = pd.DataFrame(data)
df
```

[42]:

	Name	Age	Height
0	tom	10	6.2
1	nick	20	5.5

[43]:
```
df.describe()
```

use describe() to get a summary of the data

[43]:

	Age	Height
count	2.000000	2.000000
mean	15.000000	5.850000
std	7.071068	0.494975
min	10.000000	5.500000
25%	12.500000	5.675000
50%	15.000000	5.850000
75%	17.500000	6.025000
max	20.000000	6.200000

More on Pandas

Many database operations are available

- You can specify index, which can speed up some operations
- You can do ‘join’
- You can do ‘where’ clause to filter the data
- You can do ‘group by’

More on Pandas

pandas

Search the docs ...

- Installation
- Package overview
- **Getting started tutorials**
  - What kind of data does pandas handle?
  - How do I read and write tabular data?
  - How do I select a subset of a `DataFrame` ?
  - How to create plots in pandas?
  - How to create new columns derived from existing columns?
  - How to calculate summary statistics?
  - How to reshape the layout of tables?
  - **How to combine data from multiple tables?**
  - How to handle time series data with ease?
  - How to manipulate textual data?

Doing it by yourself helps a lot!

Descriptive Statistics (using Pandas)

Descriptive Statistics Overview

- Given a data array, oftentimes useful to summarize it using some of its key features
  - **Range**
  - **Histogram**
  - **Mean**
  - **Median**
  - **Mode**

Range

- Difference between highest (maximum) and lowest (minimum) values
- [min, max] is called the *range interval*

Example what is the range of the following dataset?

4, 7, 2, 9, 12

Max: 12

Min: 2

=> Range interval = [2, 12], Range = 12 – 2 = 10

Histogram

Split the *range interval* into equally-sized bins and report counts in each bin.

Example Taking the ages of the presidents of the United States at the time of their inauguration (in total 44 points)

57,61,57,57,58,57,61,54,68,51, .. 47,70

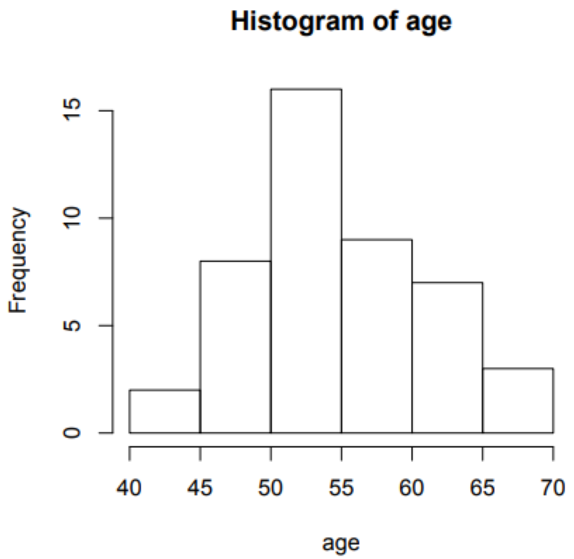
Bins: (40, 45], (45, 50], (50, 55], (55, 60], (60, 65], (65, 70]

Histogram

Counts in different bins

(40, 45]	(45, 50]	(50, 55]	(55, 60]	(60, 65]	(65, 70]
2	8	16	9	7	3

We can also visualize the histogram using a bar plot:



Histogram can show the "spread" of data

Mean

- Average of the data $x_1, \ldots, x_n$
- In formula:

$$\bar{x} = \frac{1}{n}(x_1 + \cdots + x_n) =: \frac{1}{n}\sum_{i=1}^{n} x_i$$

Example heights of 3 students are 1.71, 1.84, 1.64 (m)

their average height $\bar{x} = \frac{1}{3}$(1.71+1.84+1.63) = 1.73 (m)

Median

For data $x_1, x_2, \ldots, x_N$ sort the data,

$$x_{(1)}, x_{(2)}, \ldots, x_{(n)}$$

- Notation $x_{(i)}$ means the i-th *lowest* value, e.g. $x_{(i-1)} \leq x_{(i)} \leq x_{(i+1)}$
- $x_{(1)}, x_{(2)}, \ldots, x_{(n)}$ are called *order statistics* ← not summary info, but rather a transformation

If n is **odd** then find the middle datapoint,

$$\text{median}(x_1, \ldots, x_n) = x_{((n+1)/2)}$$

If n is **even** then average between both middle datapoints,

$$\text{median}(x_1, \ldots, x_n) = \frac{1}{2}\left(x_{(n/2)} + x_{(n/2+1)}\right)$$

Median

What is the median of the following data?

1, 2, 3, 4, 5, 6, 8, 9 **4.5**

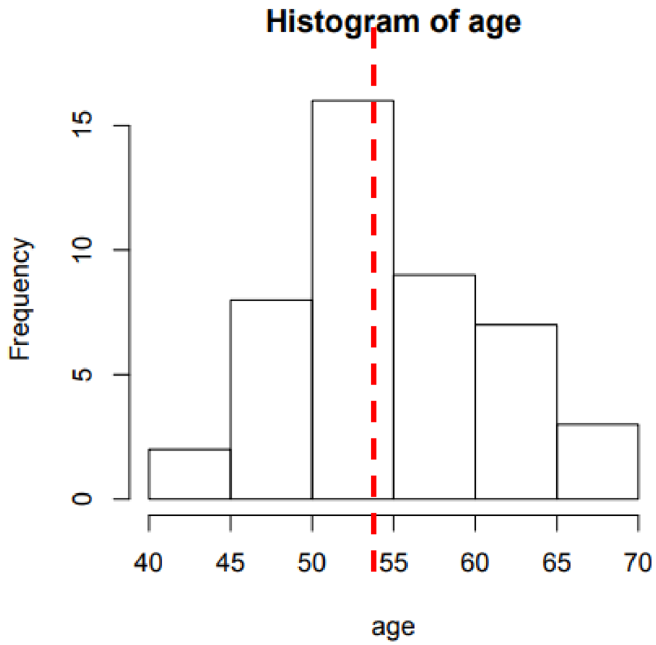
What is the median of the following data?

1, 2, 3, 4, 5, 6, 8, 100 **4.5**

Median is *robust* to outliers

Median

- *Roughly speaking*, median is the point where half of the population is below it and half of the population is above it



Mode

- Value of highest number of appearances

Example what is the mode of the following dataset?

1,1,2,3,7,8,8,8,9

Count of 8: 3

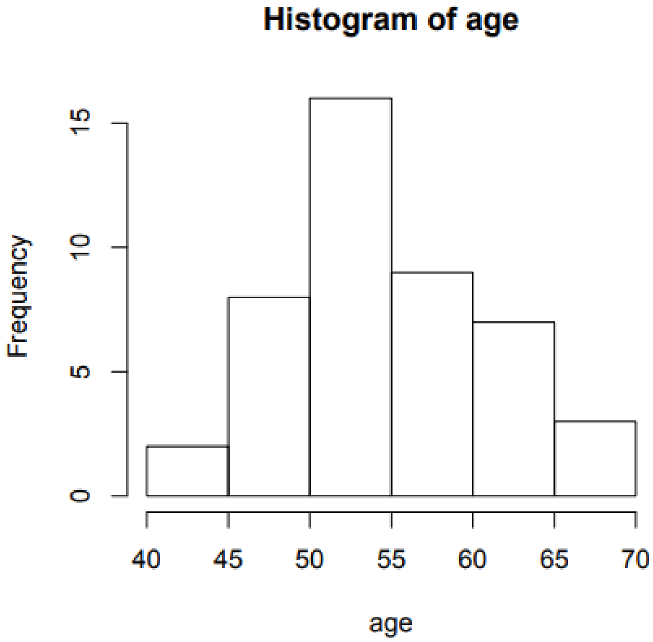
Count of 1: 2

Counts of other numbers: 1

=> Mode = 8

Mode

- *Roughly speaking,* mode is the location of the histogram with the tallest bar



Measuring Spread: sample variance

Another way to measure the spread is the sample variance,

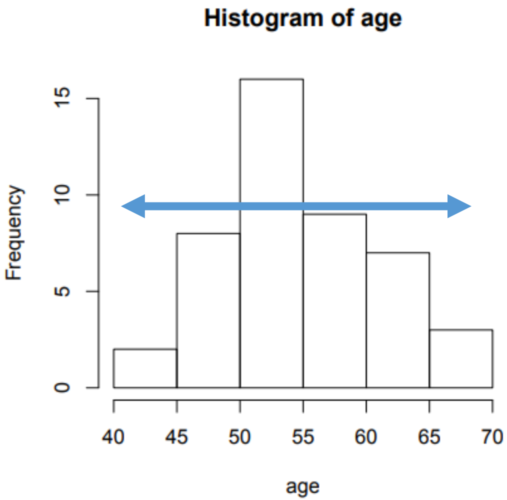
$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

$$s^2 = \frac{1}{N-1}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Biased

Sample mean

Unbiased



Sample Variance

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

Example calculate the sample variance of sample

4, 9, 10, 6, 6

Sample mean: $\bar{x} = \frac{4+9+10+6+6}{5} = 7$

5 terms in the summation:

$$(4-7)^2, (9-7)^2, (10-7)^2, (6-7)^2, (6-7)^2$$

$$9, \quad 4, \quad 9, \quad 1, \quad 1$$

$$\sigma^2 = \frac{1}{5}(9+4+9+1+1) = 4.8$$

Sample variance

- When is the variance of a sample zero?

$$\sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \bar{x})^2$$

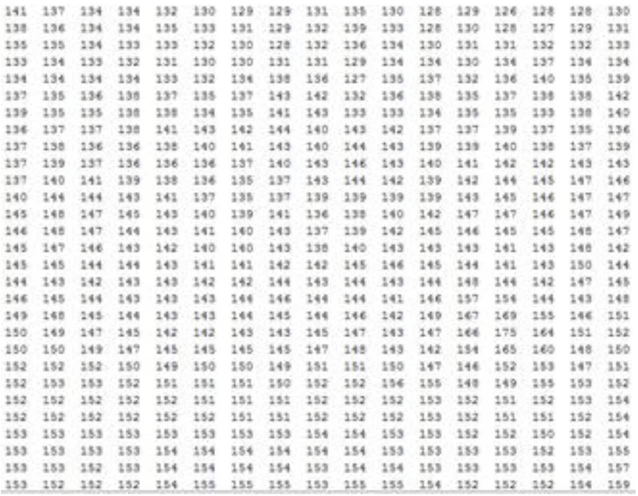
- Variance of a sample is zero if all x_i's are identical, e.g.

5, 5, .., 5

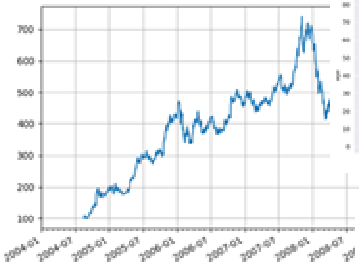
- Variance measures the degree of “fluctuations” in the data
- Standard deviation*:* square root of variance, σ

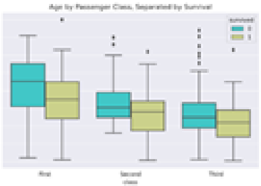
Data Visualization

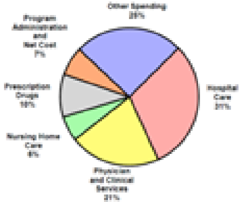
Data Analysis, Exploration, and Visualization



Encoding





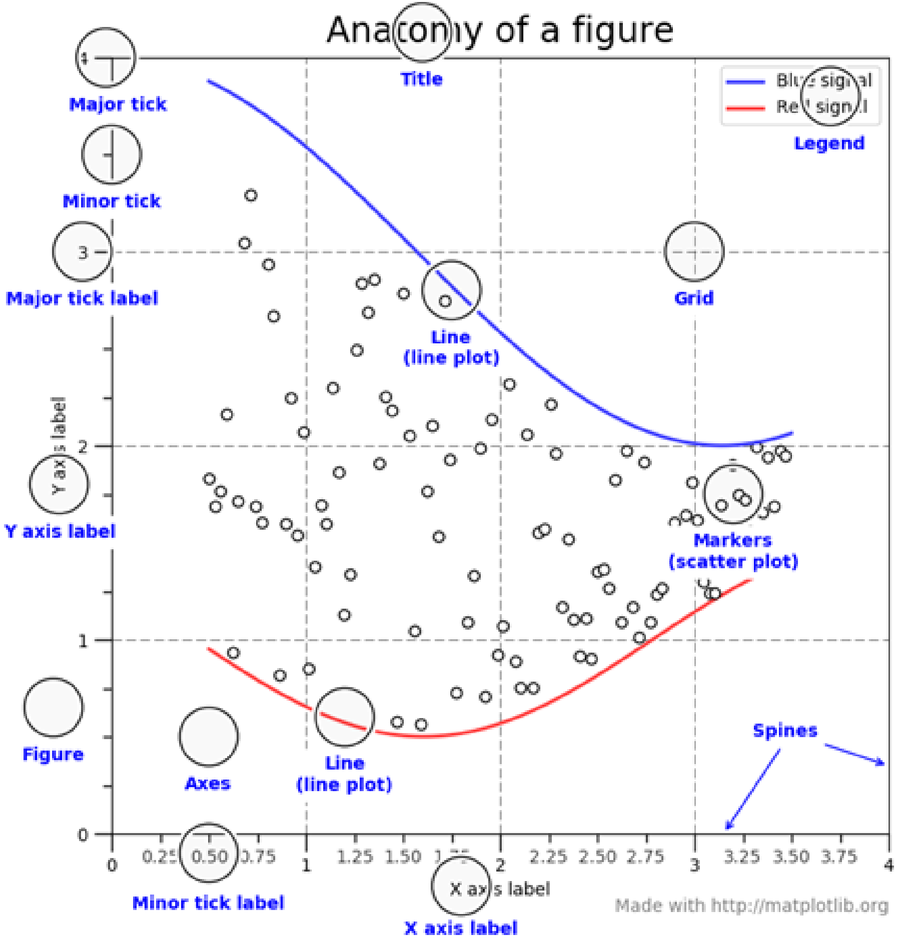


Iterate

Visual Perception

Understanding

matplotlib



components of a Matplotlib figure

Documentation + tutorials:

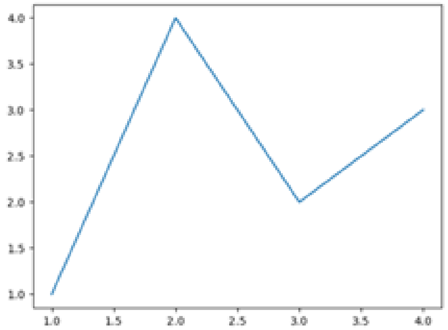
https://matplotlib.org/

Data visualization in Python…

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

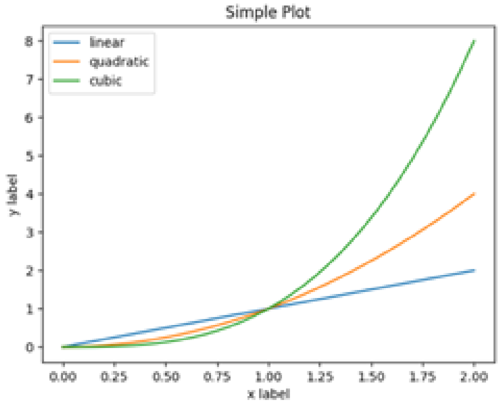
Create a simple figure

```
fig, ax = plt.subplots()  # Create a figure containing a single axes.
ax.plot([1, 2, 3, 4], [1, 4, 2, 3])  # Plot some data on the axes.
```

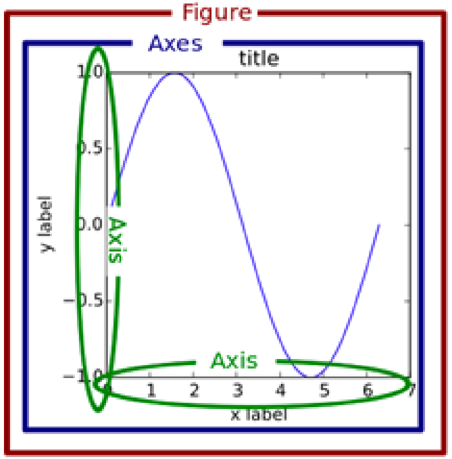


A more complicated plot…

```
x = np.linspace(0, 2, 100)

# Note that even in the OO-style, we use `.pyplot.figure` to create the figure.
fig, ax = plt.subplots()  # Create a figure and an axes.
ax.plot(x, x, label='linear')  # Plot some data on the axes.
ax.plot(x, x**2, label='quadratic')  # Plot more data on the axes...
ax.plot(x, x**3, label='cubic')  # ... and some more.
ax.set_xlabel('x label')  # Add an x-label to the axes.
ax.set_ylabel('y label')  # Add a y-label to the axes.
ax.set_title("Simple Plot")  # Add a title to the axes.
ax.legend()  # Add a legend.
```

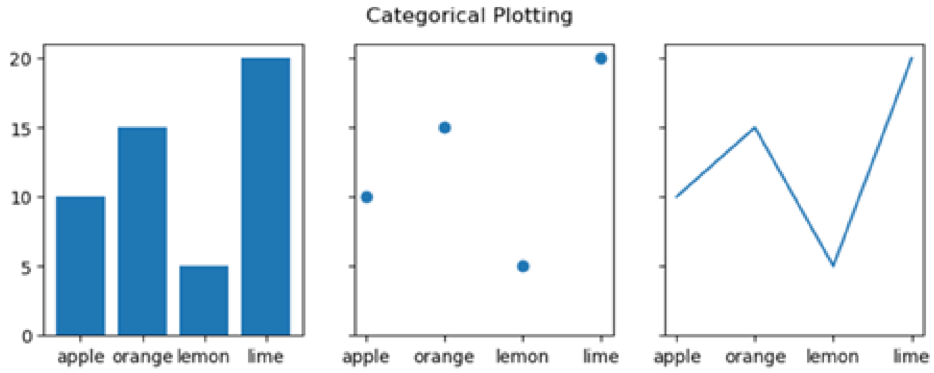


Axes: entire area of plot
Axis: horizontal or vertical (2d)



`subplot()` function: draw multiple plots in one figure

```
data = {'apple': 10, 'orange': 15, 'lemon': 5, 'lime': 20}
names = list(data.keys())
values = list(data.values())

fig, axs = plt.subplots(1, 3, figsize=(9, 3), sharey=True)
axs[0].bar(names, values)
axs[1].scatter(names, values)
axs[2].plot(names, values)
fig.suptitle('Categorical Plotting')
```



Aside: generating random data

- Numpy: Python lib for scientific computing

- It has general-purpose random number generator *rand*

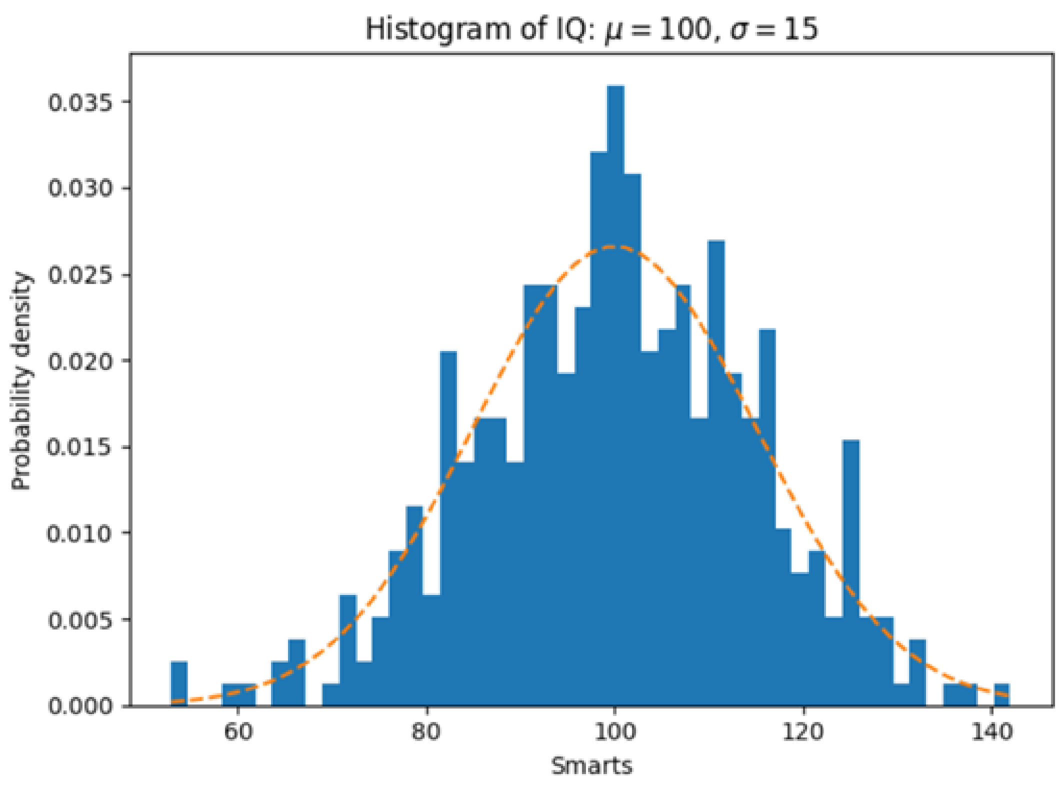
```
import numpy as np

# Generate an array with 5 random numbers between 0 and 1
random_array_1d = np.random.rand(5)

# Print the generated random array
print(random_array_1d)
```

```
[0.70620389 0.38344751 0.12382312 0.85396815 0.3684137 ]  # This will vary each time
```

Histogram

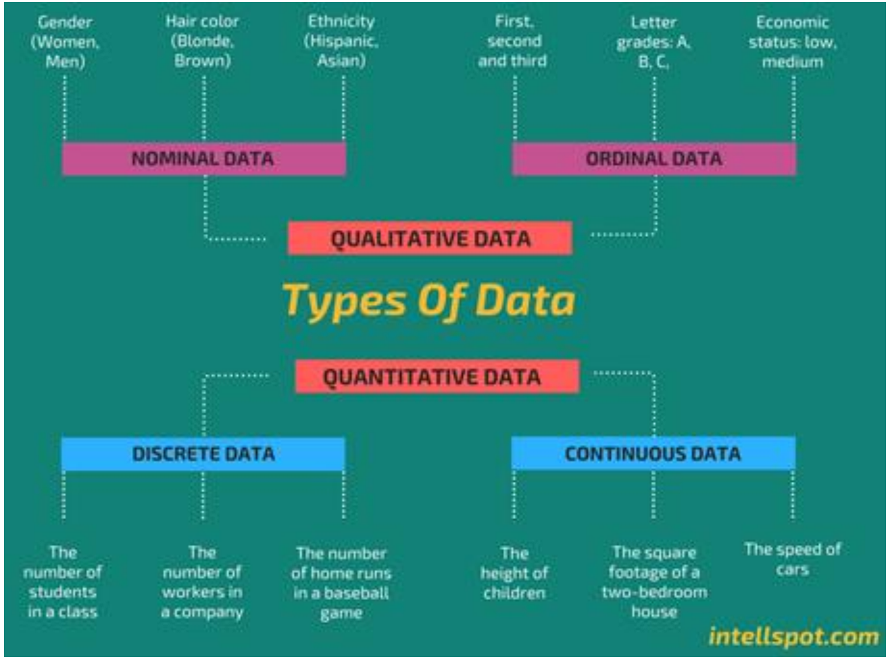

```
import numpy as np
import matplotlib.pyplot as plt

np.random.seed(19680801)

# example data
mu = 100  # mean of distribution
sigma = 15  # standard deviation of distribution
x = mu + sigma * np.random.randn(437)

num_bins = 50

fig, ax = plt.subplots()

# the histogram of the data
n, bins, patches = ax.hist(x, num_bins, density=True)

# add a 'best fit' line
y = ((1 / (np.sqrt(2 * np.pi) * sigma)) *
     np.exp(-0.5 * (1 / sigma * (bins - mu))**2))
ax.plot(bins, y, '--')
ax.set_xlabel('Smarts')
ax.set_ylabel('Probability density')
ax.set_title(r'Histogram of IQ: $\mu=100$, $\sigma=15$')

# Tweak spacing to prevent clipping of ylabel
fig.tight_layout()
plt.show()
```

Generate 437 random data; randn similar to rand



Types of Data

Data come in many forms, each requiring different approaches & models



Qualitative or **categorical**: can partition values into classes

Quantitative: can perform arithmetic operations (e.g., addition, subtraction, ordering)

We often refer to different types of data as ***variables***

Categorical Variables

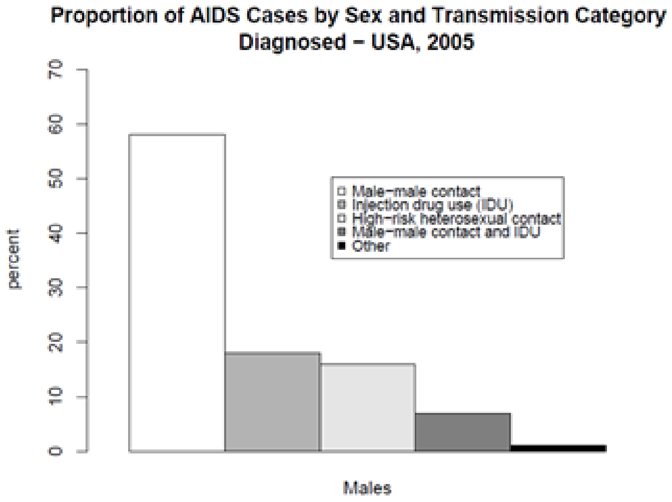
Examples

- Blood Type: A, B, AB, or O
- Political Party: Democrat, Republican, etc.
- Word Identity: NP, VP, N, V, Adj, Adv, etc.
- Roll of a die: 1,2,3,4,5 or 6 ← **Numerical data can be categorical depending on context**

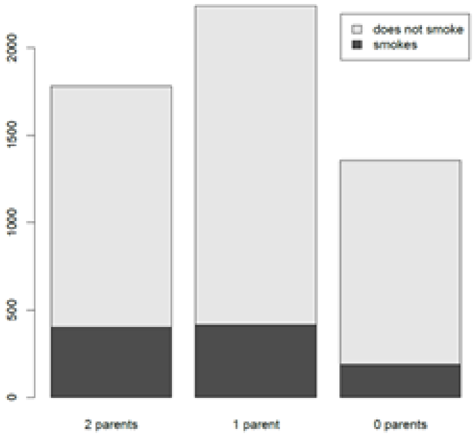
Conversion: Quantitative data can be converted to categorical by defining ranges:

- Small [0, 10cm), Medium [10, 100cm), Large [100cm, 1m), XL [1m, -)
- Low [less than -100dB), Moderate [-100dB, -50dB), Loud [over -50dB)

Visualizing Categorical Variables



	student smokes	student does not smoke	total
2 parents smoke	400	1380	1780
1 parent smokes	416	1823	2239
0 parents smoke	188	1168	1356
total	1004	4371	5375

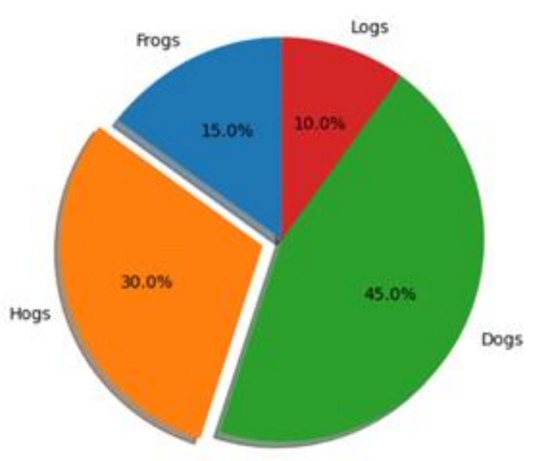


Pie Chart

Circular chart divided into sectors, illustrating relative magnitudes in frequencies or percentage.

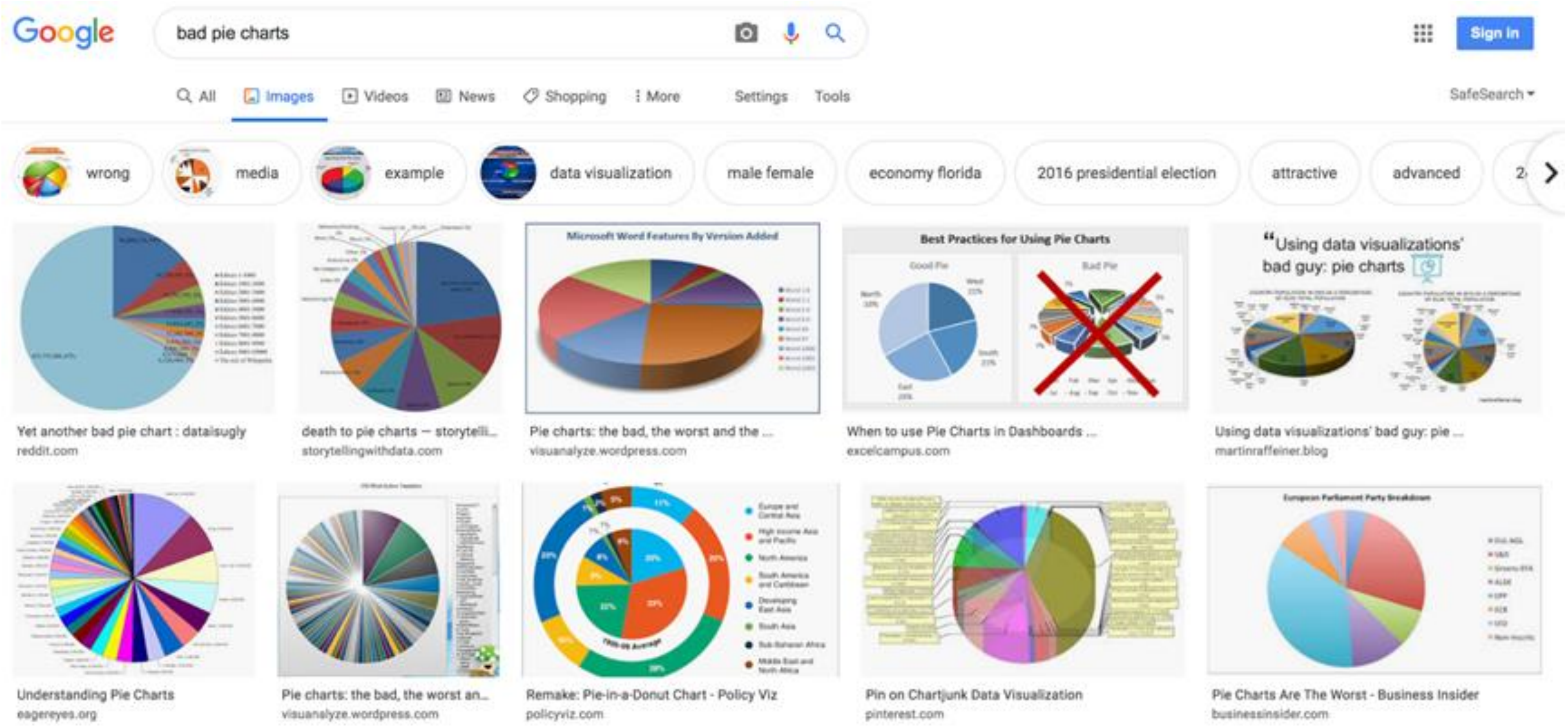
In a pie chart, *the area is proportional to the quantity it represents*

```
import matplotlib.pyplot as plt

# Pie chart, where the slices will be ordered and plotted counter-clockwise:
labels = 'Frogs', 'Hogs', 'Dogs', 'Logs'
sizes = [15, 30, 45, 10]
explode = (0, 0.1, 0, 0)  # only "explode" the 2nd slice (i.e. 'Hogs')

fig1, ax1 = plt.subplots()
ax1.pie(sizes, explode=explode, labels=labels, autopct='%1.1f%%',
        shadow=True, startangle=90)
ax1.axis('equal')  # Equal aspect ratio ensures that pie is drawn as a circle.

plt.show()
```



Be careful with using pie charts:

- Maybe unsuitable if too many sectors are present
- 3d charts can distort the sizes of the sectors; using 2d is recommended

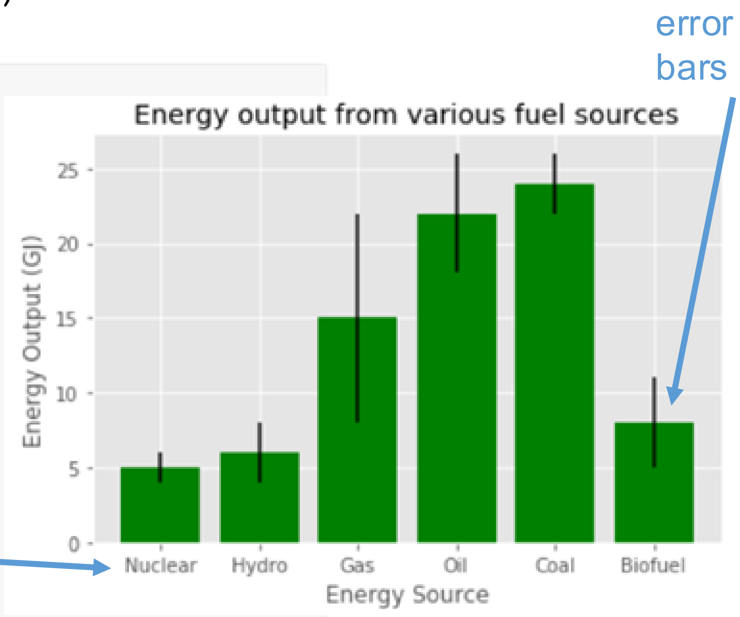


Bar Chart

We perceive differences in height / length better than area…

`plt.bar()`

```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
variance = [1, 2, 7, 4, 2, 3]

x_pos = [i for i, _ in enumerate(x)]

plt.bar(x_pos, energy, color='green', yerr=variance)
plt.xlabel("Energy Source")
plt.ylabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")

plt.xticks(x_pos, x)

plt.show()
```



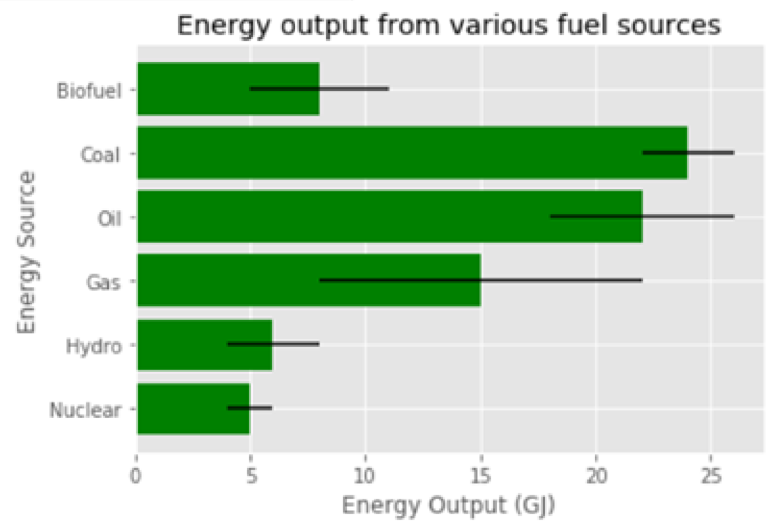
[Source: https://benalexkeen.com/bar-charts-in-matplotlib/]

Bar Chart

Horizontal version

`plt.barh()`

```
x = ['Nuclear', 'Hydro', 'Gas', 'Oil', 'Coal', 'Biofuel']
energy = [5, 6, 15, 22, 24, 8]
variance = [1, 2, 7, 4, 2, 3]

x_pos = [i for i, _ in enumerate(x)]

plt.barh(x_pos, energy, color='green', xerr=variance)
plt.ylabel("Energy Source")
plt.xlabel("Energy Output (GJ)")
plt.title("Energy output from various fuel sources")

plt.yticks(x_pos, x)

plt.show()
```



[Source: https://benalexkeen.com/bar-charts-in-matplotlib/]

Bar Chart

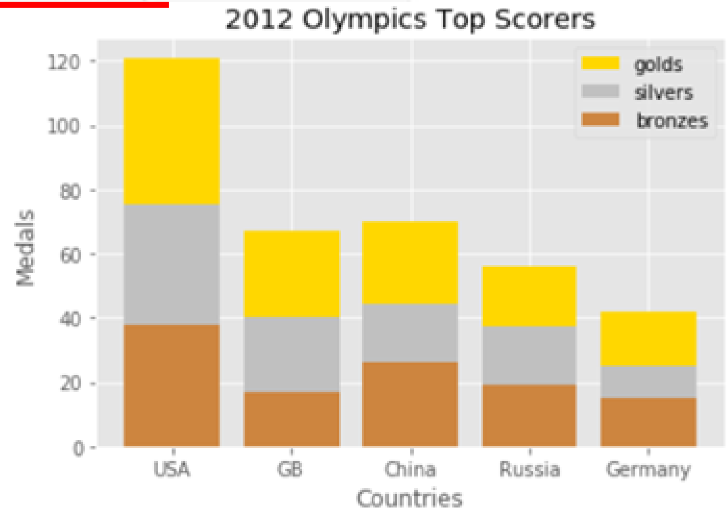
Multiple groups of bars…

```
import numpy as np

N = 5
men_means = (20, 35, 30, 35, 27)
women_means = (25, 32, 34, 20, 25)

ind = np.arange(N)    // [1,2,3,4,5]
width = 0.35           width of bar
plt.bar(ind, men_means, width, label='Men')
plt.bar(ind + width, women_means, width,
    label='Women')     add the offset here

plt.ylabel('Scores')
plt.title('Scores by group and gender')

plt.xticks(ind + width / 2, ('G1', 'G2', 'G3', 'G4', 'G5'))
plt.legend(loc='best')
plt.show()
```

[Source: https://benalexkeen.com/bar-charts-in-matplotlib/]

Stacked Bar Chart


```
countries = ['USA', 'GB', 'China', 'Russia', 'Germany']
bronzes = np.array([38, 17, 26, 19, 15])
silvers = np.array([37, 23, 18, 18, 10])
golds = np.array([46, 27, 26, 19, 17])
ind = [x for x, _ in enumerate(countries)]

plt.bar(ind, golds, width=0.8, label='golds', color='gold', bottom=silvers+bronzes)
plt.bar(ind, silvers, width=0.8, label='silvers', color='silver', bottom=bronzes)
plt.bar(ind, bronzes, width=0.8, label='bronzes', color='#CD853F')

plt.xticks(ind, countries)
plt.ylabel("Medals")
plt.xlabel("Countries")
plt.legend(loc="upper right")
plt.title("2012 Olympics Top Scorers")

plt.show()
```



[Source: https://benalexkeen.com/bar-charts-in-matplotlib/]

Two-Way Table

When there are two categorical variables:
Also called contingency table or cross tabulation table…

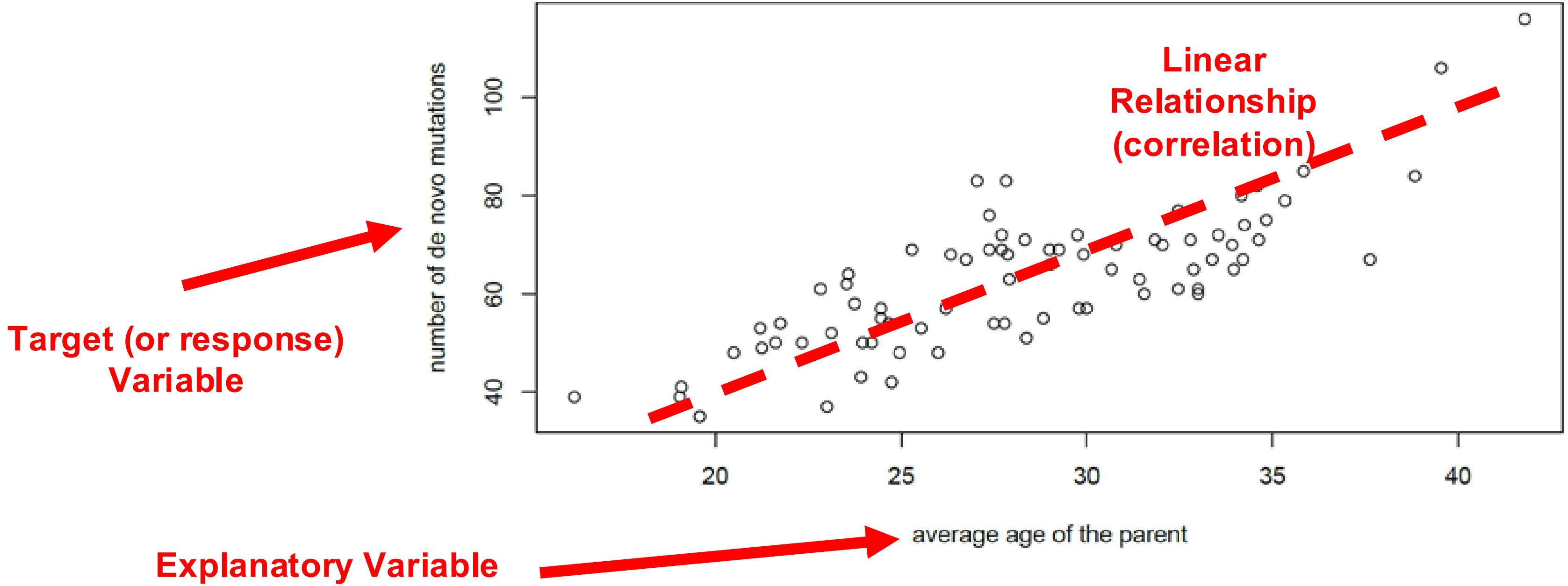
Example We asked 5375 students and collected their smoking status and their parents' smoking status, and summarize it as:

	student smokes	student does not smoke	total
2 parents smoke	400	1380	1780
1 parent smokes	416	1823	2239
0 parents smoke	188	1168	1356
total	1004	4371	5375

Q: Is there any correlation between parents' and child's smoking statuses? Are students with 2 parents smoking more likely to smoke (compared with general students)?

Scatterplot

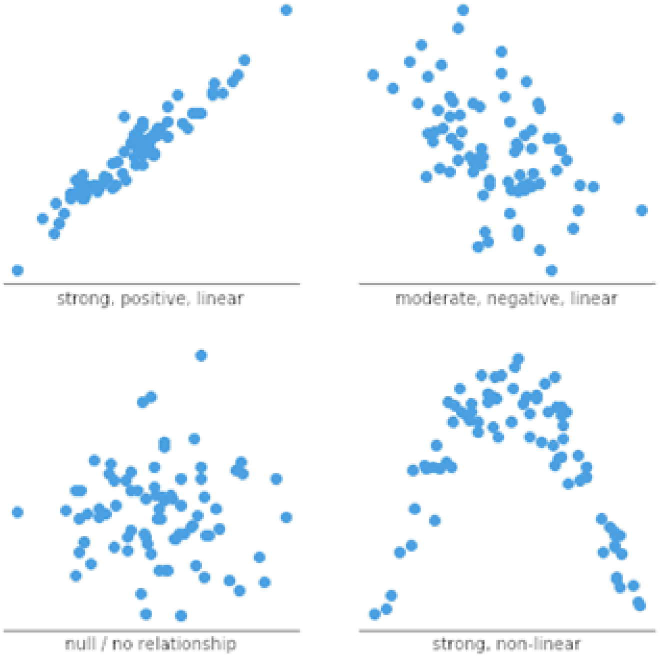
Compares relationship between two quantitative variables…



Useful for many prediction tasks:
e.g. house price prediction, salary prediction, stock price prediction, etc.

Scatterplot

Compares relationship between two quantitative variables…

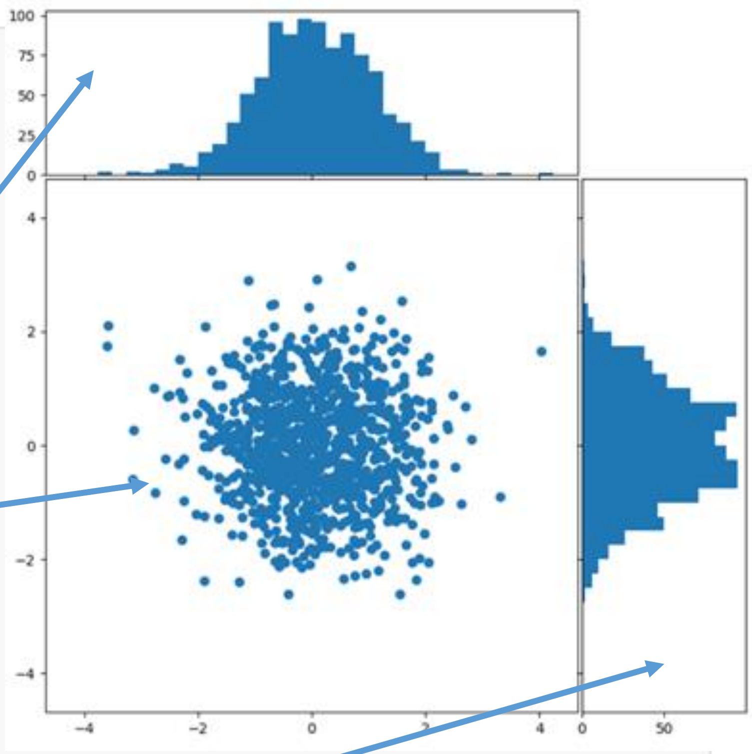


Relationship can also be:

- Nonlinear (e.g. "curvy")
- Clustered or grouped

Scatterplot + Histogram

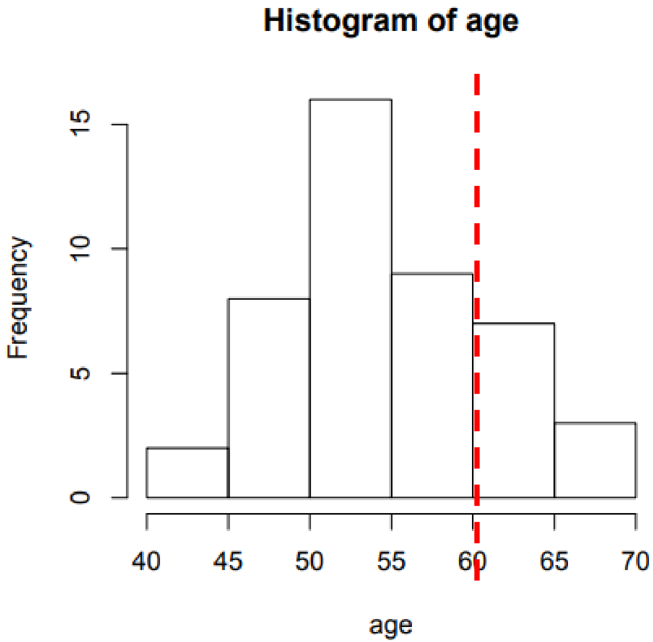

```
import numpy as np
import matplotlib.pyplot as plt

# Fixing random state for reproducibility
np.random.seed(19680801)

# some random data
x = np.random.randn(1000)
y = np.random.randn(1000)


def scatter_hist(x, y, ax, ax_histx, ax_histy):
    # no labels
    ax_histx.tick_params(axis="x", labelbottom=False)
    ax_histy.tick_params(axis="y", labelleft=False)

    # the scatter plot:
    ax.scatter(x, y)

    # now determine nice limits by hand:
    binwidth = 0.25
    xymax = max(np.max(np.abs(x)), np.max(np.abs(y)))
    lim = (int(xymax/binwidth) + 1) * binwidth

    bins = np.arange(-lim, lim + binwidth, binwidth)
    ax_histx.hist(x, bins=bins)
    ax_histy.hist(y, bins=bins, orientation='horizontal')
```



Full Code:
https://matplotlib.org/stable/gallery/lines_bars_and_markers/scatter_hist.html

Percentile / Quartile

Question Is 60yrs old for a US president? Why or why not?



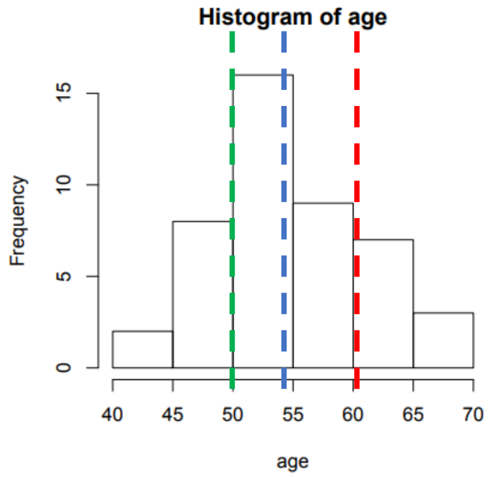
The number of presidents <60: 33
Total number of presidents: 44

About 75% of presidents younger than 60yrs old
=> 60yrs old = 0.75 Quantile or 75^{th} Percentile

Measuring Spread

Quartile divide data into 4 equally-sized bins,

- **1st Quartile** : Lowest 25% of data
- **2nd Quartile** : Median (lowest 50% of data)
- **3rd Quartile** : 75% of data is below 3rd quartile
- **4th Quartile** : The maximum value



Compute using `np.quantile()` :

```
x = np.random.rand(10) * 100
q = np.quantile(x, (0.25, 0.5, 0.75))
np.set_printoptions(precision=1)
print( "X: " , x )
print( "Q: " , q )
```

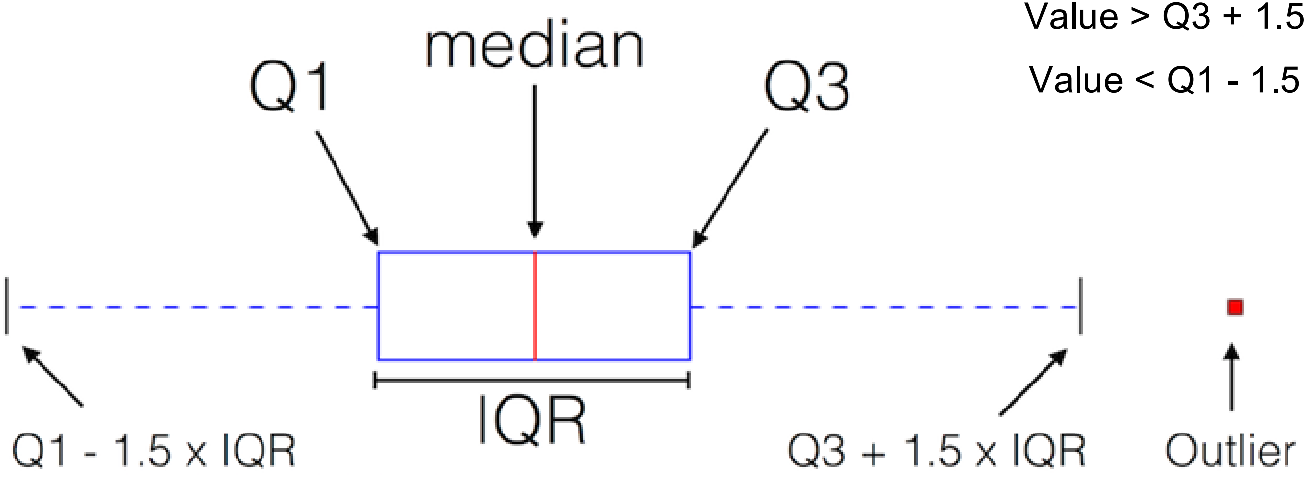
```
X:  [90.7 73.9 31.7  2.8 56.3 95.7 15.6 75.8  4.1 19.5]
Q:  [16.6 44.  75.3]
```

Box Plot

Outliers are atypical data whose

Value > Q3 + 1.5 x IQR

Value < Q1 - 1.5 x IQR



Interquartile-Range (IQR) Measures interval containing 50% of data

IQR = Q3 – Q1

Region of *typical* data

Box Plot

48 52 57 61 64 72 76 77 81 85 88

Median

48 52 57 61 64 (72) 76 77 81 85 88

48 52 57 61 64 72 | 72 76 77 81 85 88

First half | Second half

Q1 | Q3

48 52 57 ↓ 61 64 72 | 72 76 77 ↓ 81 85 88

First half | Second half

$$Q1 = \frac{57 + 61}{2} = 59$$

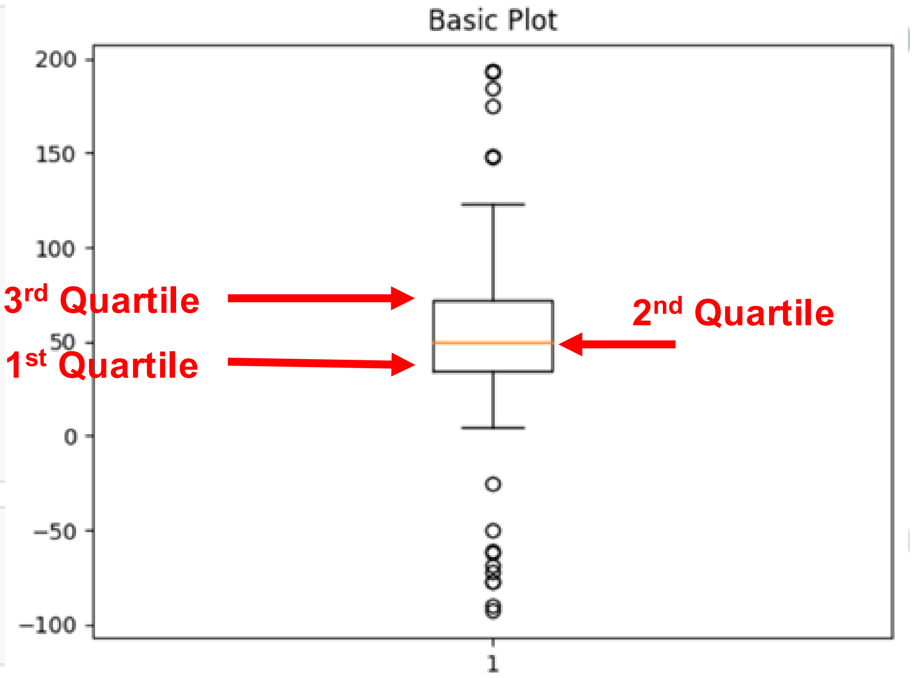
$$Q3 = \frac{77 + 81}{2} = 79$$

$$IQR = Q3 - Q1$$
$$IQR = 79 - 59 = 20$$

Box Plot


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

# Fixing random state for reproducibility
np.random.seed(19680801)

# fake up some data
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))
```

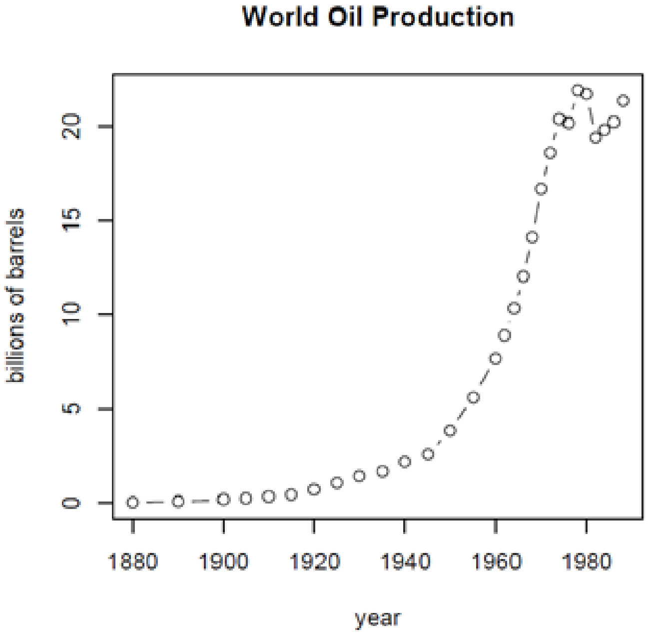
```
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
```

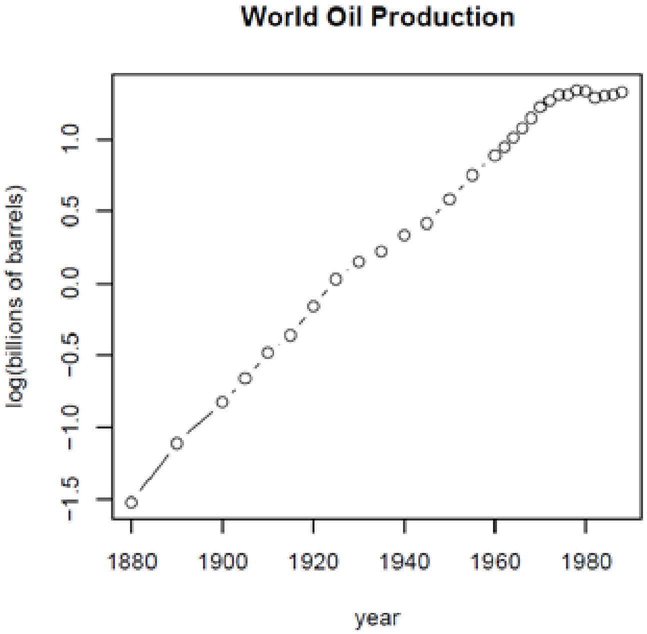


Logarithm Scale

Changing limits and base of y-scale highlights different aspects…

if y = e^x, then log(y) = x => becomes linear in x
if y = b^x, then log(y) = log(b)*x





…log-scale emphasizes relative changes in smaller quantities

More Visualization Resources

datavizcatalogue.com

matplotlib

matplotlib.org

scikit learn

scikit-learn.org