Introduction to Data Science

Yonatan Friedman & Erez Feuer

Table of contents

71253

Part I

Tirgul

1 Tutorial 1

1.1 Getting to Know Pandas and Jupyter Notebooks

Jupyter Notebooks are an open-source web application that can be used to create and share documents that contain live code, equations, visualizations and narrative text.

Pandas is a software library written for the Python programming language for data manipulation and analysis.

1.1.1 In today's tirgul we will learn:

- A bit about Jupyter Notebook
- How to import data into Pandas and
- Some basic tools for manipulation.

1.2 Opening a Jupyter Notebook

If you've never used a Jupyter Notebook before…

1. Install Anaconda (https://docs.anaconda.com/anaconda/install/)
2. Option 1: Open Anaconda Navigator and click on "Jupyter Notebook." Option 2: Open terminal and type `jupyter notebook` followed by Enter.**
3. Navigate to the folder (i.e., directory) that you want to be in.
4. Click "New–>Python 3 Notebook"

**There are lots of other ways to do this. You can also use Visual Studio Code, or Binder… the choice is yours.

1.3 About Jupyter Notebook

Jupyter Notebook is an open-source web application that allows you to create and share documents that contain live code, equations, visualizations, and narrative text. It's a great tool for interactive data analysis, scientific computing, and educational purposes.

In Jupyter Notebook, you'll see two main types of cells: **code cells** and **markdown cells**.

1.3.1 Code cell

Code cells are where you can write and execute Python (or other programming language) code. To create a new code cell, click on the `+` icon in the toolbar or use the `Insert` menu. To run a code cell, you can click the `Run` button in the toolbar, or press `Shift + Enter` on your keyboard. The code in the cell will be executed, and any output will be displayed below the cell.

1.3.2 Markdown cell

Markdown cells are where you can write formatted text using Markdown syntax. Markdown is a lightweight markup language that allows you to add headings, lists, links, images, and more to your text. In fact, **this is a markdown cell** To create a new markdown cell, click on the `+` icon and select `Markdown` from the dropdown menu. To render the markdown as formatted text, you can also run the cell using `Shift + Enter`.

1.3.3 Execution order

The order of appearance of cells in your notebook **doesn't necessarily determine the order in which they are executed**. Instead, Jupyter Notebook keeps track of the order in which cells are run, and uses that order to ensure that all necessary code is executed before it's used in later cells. You can see the execution order of cells by looking at the number to the left of the cell (e.g., `[1]`) - this indicates the order in which the cell was executed.

In summary, Jupyter Notebook is a powerful tool for interactive data analysis and scientific computing. Code cells allow you to write and execute Python code, while markdown cells allow you to write formatted text using Markdown syntax. You can run cells using the `Run` button or `Shift + Enter`, and Jupyter Notebook keeps track of the order in which cells are executed to ensure that your code runs correctly.

1.4 Short Markdown example

To create headers, you can use the "#" symbol followed by a space, and then type your header text. The number of "#" symbols determines the level of the header, with one "#" indicating the largest header and six "#" indicating the smallest header. For example:

```
# This is a level 1 header

## This is a level 2 header

### This is a level 3 header
```

To go down a row and write bullet points, you can use the "*" symbol followed by a space, and then type your bullet text. For example:

- This is a bullet point
- This is another bullet point

1.4.1 There are many more things you can do, this cheat sheet can be very helpful.

1.5 Download the required dataset

We are going to be using a dataset based on information from 60k OkCupid users. The dataset was collected by web scraping the OKCupid.com website on 2012/06/30, and includes profiles of people within a 25 mile radius of San Francisco, who were online in the previous year (after 06/30/2011), with at least one profile picture. The data includes age, sex, orientation as well as text data from open ended descriptions. You can download the dataset here: https://github.com/ErezFeuer/Introduction-to-Data-Science-Tutorials/blob/master/tirgul_1/okcupid_profiles.csv.

How to download from files from Github https://www.gitkraken.com/learn/git/github-download#how-to-downlaod-a-file-from-github

1.6 Getting started with Pandas

```
# import the pandas library
import pandas as pd
```

```
# import the dataset
okcupid_df = pd.read_csv('okcupid_profiles.csv')
```

You just created a `pandas DataFrame`.

We can look at our data by using the `.head()` method.

By default, this shows the header (column names) and the first five rows.

Passing an integer, n, to `.head()` returns that number of rows.

```
# peak at the data
okcupid_df.head(10)
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
0	22	single	m	straight	a little extra	strictly anything	socially	never	working on c
1	35	single	m	straight	average	mostly other	often	sometimes	working on s
2	38	available	m	straight	thin	anything	socially	NaN	graduated fr
3	23	single	m	straight	thin	vegetarian	socially	NaN	working on c
4	29	single	m	straight	athletic	NaN	socially	never	graduated fr
5	29	single	m	straight	average	mostly anything	socially	NaN	graduated fr
6	32	single	f	straight	fit	strictly anything	socially	never	graduated fr
7	31	single	f	straight	average	mostly anything	socially	never	graduated fr
8	24	single	f	straight	NaN	strictly anything	socially	NaN	graduated fr
9	37	single	m	straight	athletic	mostly anything	not at all	never	working on t

Alternatively, to see the last n rows, use `.tail()`.

```
okcupid_df.tail(10)
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	educa
9990	33	single	m	straight	average	mostly anything	socially	never	gradu
9991	27	single	m	gay	athletic	NaN	socially	never	gradu
9992	39	single	f	straight	fit	mostly anything	socially	never	gradu
9993	38	single	f	straight	athletic	NaN	socially	sometimes	gradu
9994	29	single	m	straight	average	NaN	socially	never	colleg
9995	24	single	f	straight	curvy	anything	socially	never	gradu
9996	24	single	f	straight	skinny	NaN	socially	never	gradu
9997	19	seeing someone	f	bisexual	athletic	NaN	rarely	never	NaN
9998	47	single	f	straight	a little extra	NaN	rarely	never	NaN
9999	29	single	f	straight	NaN	mostly vegan	socially	NaN	gradu

1.6.1 To find the number of rows, you can use the `len()` function. Alternatively, you can use the `shape` attribute.

```
# get the length of the dataframe
len(okcupid_df)
```

```
10000
```

```
# get the size of the dataframe (rows, columns)
okcupid_df.shape
```

```
(10000, 21)
```

1.6.2 What are all the columns in our dataset?

```
# get columns
okcupid_df.columns
```

```
Index(['age', 'status', 'sex', 'orientation', 'body_type', 'diet', 'drinks',
       'drugs', 'education', 'ethnicity', 'height', 'income', 'job',
       'last_online', 'location', 'offspring', 'pets', 'religion', 'sign',
       'smokes', 'speaks'],
      dtype='object')
```

```
# get column data types
okcupid_df.dtypes
```

```
age             int64
status         object
sex            object
orientation    object
body_type      object
diet           object
drinks         object
drugs          object
education      object
ethnicity      object
```

```
height          int64
income          int64
job            object
last_online    object
location       object
offspring      object
pets           object
religion       object
sign           object
smokes         object
speaks         object
dtype: object
```

1.6.3 Accessing a single column

```
# access a single column
okcupid_df[['height']]
```

	height
0	75
1	70
2	68
3	71
4	66
...	...
9995	66
9996	59
9997	61
9998	64
9999	68

```
# let's try that a different way...
okcupid_df['height']
```

```
0       75
1       70
2       68
3       71
4       66
```

```
        ..
9995    66
9996    59
9997    61
9998    64
9999    68
Name: height, Length: 10000, dtype: int64
```

```
# and another way...
okcupid_df.height
```

```
0       75
1       70
2       68
3       71
4       66
        ..
9995    66
9996    59
9997    61
9998    64
9999    68
Name: height, Length: 10000, dtype: int64
```

It is preferrable to use the bracket notation as a column name might inadvertently have the same name as a `DataFrame` (or `Series`) method. In addition, only bracket notation can be used to create a new column. If you try and use attribute access to create a new column, you'll create a new attribute, *not* a new column.

You can either use a single bracket or a double bracket. The single bracket will output a `pandas Series`, while a double bracket will output a `pandas DataFrame`. A `pandas Series` is a single vector of data (e.g., a NumPy array) with "an associated array of data labels, called its *index*." A `DataFrame` also has an index.

In our example, the indices are an array of sequential integers, which is the default. You can find them in the left-most position, without a column label. Indices need not be a sequence of integers. They can, for example, be dates or strings. Note that indices do *not* need to be unique.

1.7 Rename, Index, and Slice

You may have noticed that the `height` column data is in inches. Let's rename the column so that the units are included in the name.

```
# rename the height column
okcupid_df.rename(columns={'height' : 'height_inches'}, inplace=True)
```

```
# check to see that the change was implemented
okcupid_df.columns
```

```
Index(['age', 'status', 'sex', 'orientation', 'body_type', 'diet', 'drinks',
       'drugs', 'education', 'ethnicity', 'height_inches', 'income', 'job',
       'last_online', 'location', 'offspring', 'pets', 'religion', 'sign',
       'smokes', 'speaks'],
      dtype='object')
```

Indices, like column names, can be used to select data. Indices can be used to select particular rows. In fact, you can do something like `.head()` with slicing using the `[]` operator.

```
okcupid_df[:5]
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
0	22	single	m	straight	a little extra	strictly anything	socially	never	working on col
1	35	single	m	straight	average	mostly other	often	sometimes	working on spa
2	38	available	m	straight	thin	anything	socially	NaN	graduated fron
3	23	single	m	straight	thin	vegetarian	socially	NaN	working on col
4	29	single	m	straight	athletic	NaN	socially	never	graduated fron

Before we continue, let's spend some more time looking at a few useful ways to index data—that is, select rows. `.loc` primarily works with string labels. It accepts a single label, a list (or array) of labels, or a slice of labels (e.g., 'a' : 'f').

```
# another way to get a single column...
okcupid_df.loc[:, 'sex']
```

```
0       m
1       m
```

```
2       m
3       m
4       m
       ..
9995    f
9996    f
9997    f
9998    f
9999    f
Name: sex, Length: 10000, dtype: object
```

```
# we can also index based on rows
okcupid_df[2:4]
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
2	38	available	m	straight	thin	anything	socially	NaN	graduated from masters pr
3	23	single	m	straight	thin	vegetarian	socially	NaN	working on college/univers

The difference is that the former returns a Series because we selected a single lable, while the latter returns a DataFrame because we selected a range of positions.

Another indexing option, .iloc, primarily works with integer positions. To select specific rows, we can do the following.

```
okcupid_df.iloc[[1, 5, 6, 9]]
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
1	35	single	m	straight	average	mostly other	often	sometimes	working on space
5	29	single	m	straight	average	mostly anything	socially	NaN	graduated from
6	32	single	f	straight	fit	strictly anything	socially	never	graduated from
9	37	single	m	straight	athletic	mostly anything	not at all	never	working on two-

We can select a range of rows and specify the step value.

```
okcupid_df.iloc[25:50:5]
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
25	28	single	m	straight	fit	anything	rarely	never	graduated from college
30	27	single	f	straight	average	anything	socially	NaN	working on college/uni
35	26	single	m	straight	athletic	NaN	NaN	never	NaN
40	30	single	m	straight	average	NaN	often	never	graduated from master
45	27	single	m	straight	average	mostly anything	socially	never	college/university

1.7.1 We can also create a new column

Here we create a new column with height in cm

```
okcupid_df['height_cm'] = okcupid_df['height_inches']*2.54
okcupid_df['height_cm']
```

```
0       190.50
1       177.80
2       172.72
3       180.34
4       167.64
         ...
9995    167.64
9996    149.86
9997    154.94
9998    162.56
9999    172.72
Name: height_cm, Length: 10000, dtype: float64
```

1.7.1.1 Another example

```
okcupid_df['speaks']
```

```
0                                                    english
1       english (fluently), spanish (poorly), french (...
2                                     english, french, c++
3                                english, german (poorly)
4                                                    english
                              ...
9995                                                 english
```

```
9996                                    english, chinese
9997                                            english
9998                                            english
9999                                    english, spanish
Name: speaks, Length: 10000, dtype: object
```

Now we'll create a column that has only the first language the person speaks.

```
okcupid_df['speaks first'] = okcupid_df['speaks'].dropna().apply(lambda x: x.split(',')[0]
okcupid_df['speaks first']
```

```
0                   english
1         english (fluently)
2                   english
3                   english
4                   english
                 ...
9995                english
9996                english
9997                english
9998                english
9999                english
Name: speaks first, Length: 10000, dtype: object
```

1.7.2 Slicing by condition

Lets say we want a df with only the females.

```
f_okcupid_df = okcupid_df.loc[okcupid_df['sex'] == 'f']
f_okcupid_df
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
6	32	single	f	straight	fit	strictly anything	socially	never	graduated
7	31	single	f	straight	average	mostly anything	socially	never	graduated
8	24	single	f	straight	NaN	strictly anything	socially	NaN	graduated
13	30	single	f	straight	skinny	mostly anything	socially	never	graduated
14	29	single	f	straight	thin	mostly anything	socially	never	working o
...	...	...	...	...	...	...	...	...	...
9995	24	single	f	straight	curvy	anything	socially	never	graduated

	age	status	sex	orientation	body_type	diet	drinks	drugs	education
9996	24	single	f	straight	skinny	NaN	socially	never	graduated
9997	19	seeing someone	f	bisexual	athletic	NaN	rarely	never	NaN
9998	47	single	f	straight	a little extra	NaN	rarely	never	NaN
9999	29	single	f	straight	NaN	mostly vegan	socially	NaN	graduated

1.8 Summary statistics

1.8.1 A useful method that generates various summary statistics is `.describe()`.

```
# get summary statistics
okcupid_df.describe()
```

	age	height_inches	income	height_cm
count	10000.000000	10000.000000	10000.000000	10000.000000
mean	32.073500	68.331200	19307.189900	173.561248
std	9.444025	3.908482	93842.703841	9.927545
min	18.000000	36.000000	-1.000000	91.440000
25%	25.000000	66.000000	-1.000000	167.640000
50%	30.000000	68.000000	-1.000000	172.720000
75%	36.000000	71.000000	-1.000000	180.340000
max	110.000000	95.000000	1000000.000000	241.300000

The summary statistics are based on non-missing values and count reflects that. The values depend on what it's called on. If the DataFrame includes both numeric and object (e.g., strings) dtypes, it will default to summarizing the numeric data. If .describe() is called on strings, for example, it will return the count, number of unique values, and the most frequent value along with its count.

```
# describe can be used on a single column
okcupid_df['income'].describe()
```

```
count      10000.000000
mean       19307.189900
std        93842.703841
min           -1.000000
25%           -1.000000
```

```
50%           -1.000000
75%           -1.000000
max      1000000.000000
Name: income, dtype: float64
```

```
# and on non-numerical data
okcupid_df['sex'].describe()
```

```
count     10000
unique        2
top           m
freq       5944
Name: sex, dtype: object
```

1.8.2 Unique values

We can find out how the unique values in a column

```
okcupid_df['drinks'].unique()
```

```
array(['socially', 'often', 'not at all', 'rarely', nan, 'very often',
       'desperately'], dtype=object)
```

1.9 Manipulating Data

1.9.1 Let's seperate the data based on sex

```
# male users
males=okcupid_df.loc[okcupid_df["sex"]=="m"]
n_males = len(males)

# female users
females=okcupid_df.loc[okcupid_df["sex"]=="f"]
n_females = len(females)

n_population = len(okcupid_df)

# how many users are male and how many are female?
```

```
print(f"{n_males} males ({n_males/n_population:.1%}), {n_females} females ({n_females/n_po
```

```
5944 males (59.4%), 4056 females (0.4056)
```

Note that above I set the male percentage for 1 decimal point with `:.1` and for the female I didn't

1.9.2 Let's examine the age distribution

```
# summary statistics on the 'age' column
okcupid_df['age'].describe()
```

```
count    10000.000000
mean        32.073500
std          9.444025
min         18.000000
25%         25.000000
50%         30.000000
75%         36.000000
max        110.000000
Name: age, dtype: float64
```

```
# how many users are older than 80?
(okcupid_df["age"]>80).sum()
```

```
1
```

```
# can also be written this way
sum(okcupid_df['age']>80)
```

```
1
```

1.9.3 Who are the age outliers?

```
okcupid_df[okcupid_df["age"]>80]
```

	age	status	sex	orientation	body_type	diet	drinks	drugs	education	ethnicity	...	last_
2512	110	single	f	straight	NaN	NaN	NaN	NaN	NaN	NaN	...	2012-

Let's assume the 110-year-old lady and the athletic 109-year-old gentleman (who's working on a masters program) are outliers: we get rid of them so the following plots look better. They didn't say much else about themselves, anyway.

```
# eliminate users with age >= 80
okcupid_df=okcupid_df[okcupid_df["age"]<=80]
```

Is the age distribution for men different than that for women?

```
print("Mean and median age for males:   {:.2f}, {:.2f}".format(males["age"].mean(),males["
print("Mean and median age for females: {:.2f}, {:.2f}".format(females["age"].mean(),femal
```

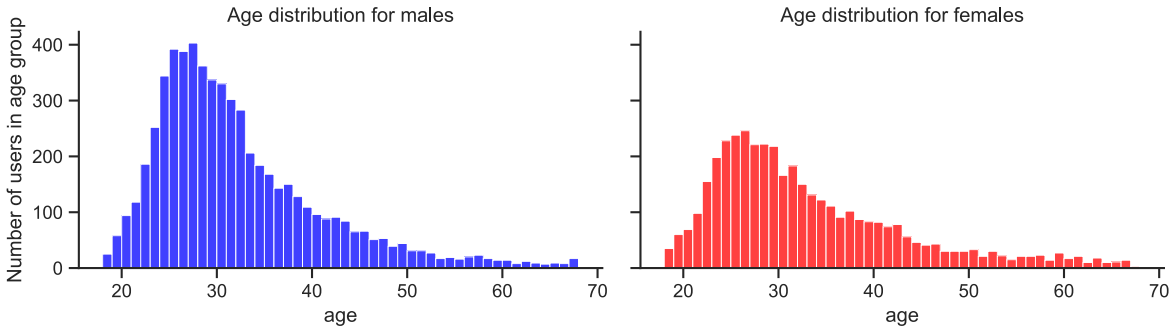
```
Mean and median age for males:   31.75, 30.00
Mean and median age for females: 32.55, 30.00
```

1.9.4 We can also plot the data (don't worry if you don't understand the code)

```
# add plotting libraries to notebook
%matplotlib inline
%config InlineBackend.figure_format='svg'
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
sns.set_style("ticks")
sns.set_context(context="notebook",font_scale=1)
```

```
# draw age histograms for male and female users
fig,(ax1,ax2) = plt.subplots(ncols=2,figsize=(10,3),sharey=True,sharex=True)
sns.histplot(males["age"], ax=ax1,
             bins=range(okcupid_df["age"].min(),okcupid_df["age"].max()),
             kde=False,
```

```
            color="b")
ax1.set_title("Age distribution for males")
sns.histplot(females["age"], ax=ax2,
             bins=range(okcupid_df["age"].min(),okcupid_df["age"].max()),
             kde=False,
             color="r")
ax2.set_title("Age distribution for females")
ax1.set_ylabel("Number of users in age group")
for ax in (ax1,ax2):
    sns.despine(ax=ax)
fig.tight_layout()
```

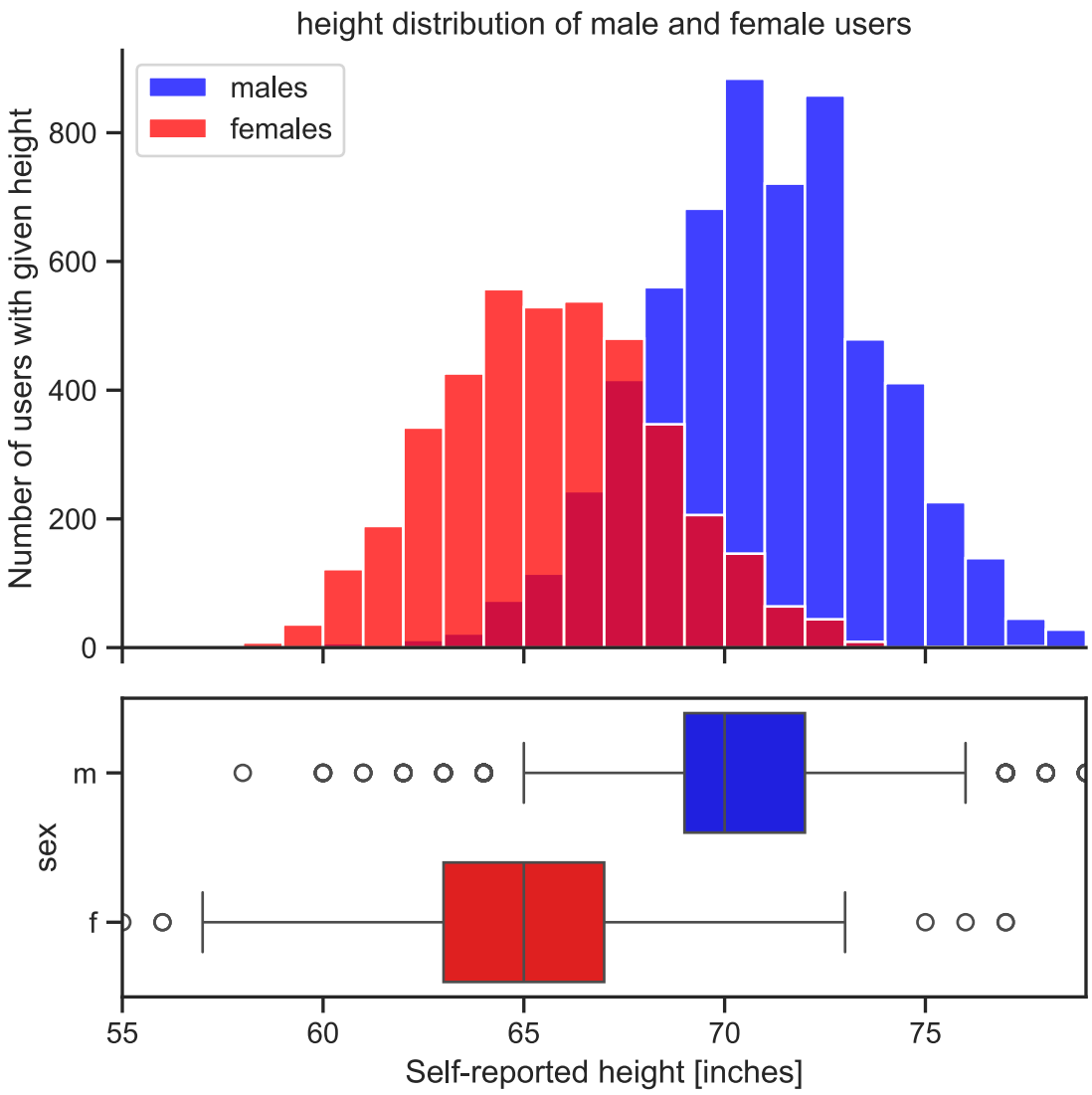


1.9.5 Now let's study height distribution and compare with official data from the US Centers of Disease Control and Prevention

```
fig,(ax,ax2) = plt.subplots(nrows=2,sharex=True,figsize=(6,6),gridspec_kw={'height_ratios'

# Plot histograms of height
bins=range(55,80)
sns.histplot(males["height_inches"].dropna(), ax=ax,
             bins=bins,
             kde=False,
             color="b",
             label="males")
sns.histplot(females["height_inches"].dropna(), ax=ax,
             bins=bins,
             kde=False,
             color="r",
             label="females")
```

```
ax.legend(loc="upper left")
ax.set_xlabel("")
ax.set_ylabel("Number of users with given height")
ax.set_title("height distribution of male and female users");

# Make aligned boxplots
sns.boxplot(data=okcupid_df,y="sex",x="height_inches",orient="h",ax=ax2,palette={"m":"b","
plt.setp(ax2.artists, alpha=.5)
ax2.set_xlim([min(bins),max(bins)])
ax2.set_xlabel("Self-reported height [inches]")

sns.despine(ax=ax)
fig.tight_layout()
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_35920/2199488932.py:21: FutureWarn

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assig

  sns.boxplot(data=okcupid_df,y="sex",x="height_inches",orient="h",ax=ax2,palette={"m":"b","
```



Males are obviously taller than females, and the two distributions make sense.

1.9.6 How do OkCupid user's heights compare to the general population?

How does this compare with general population data? Are OkCupid users maybe cheating and overreporting their height?

The CDC publishes growth charts, which contain height data for the general US population. The dataset reports statistics (3rd, 5th, 10th, 25th, 50th, 75th, 90th, 95th, 97th percentiles)

for stature for different ages from 2 to 20 years. This (and more) data is plotted by the CDC in these beautiful charts (https://www.cdc.gov/growthcharts/data/set2clinical/set2color.pdf).

```
# Import a CSV file for growth chart data
cdc=pd.read_csv("https://www.cdc.gov/growthcharts/data/zscore/statage.csv")

# print first ten rows of data
cdc.head(10)
```

	Sex	Agemos	L	M	S	P3	P5	P10	P25	P50
0	1	24.0	0.941524	86.452201	0.040322	79.910844	80.729773	81.991714	84.102892	86.452
1	1	24.5	1.007208	86.861609	0.040396	80.260371	81.088685	82.364010	84.494706	86.861
2	1	25.5	0.837251	87.652473	0.040578	81.005294	81.834452	83.113871	85.258877	87.652
3	1	26.5	0.681493	88.423264	0.040723	81.734157	82.564061	83.847162	86.005173	88.423
4	1	27.5	0.538780	89.175492	0.040833	82.448456	83.278986	84.565344	86.735069	89.175
5	1	28.5	0.407697	89.910409	0.040909	83.149450	83.980453	85.269620	87.449772	89.910
6	1	29.5	0.286762	90.629078	0.040952	83.838194	84.669484	85.960983	88.150284	90.629
7	1	30.5	0.174489	91.332424	0.040965	84.515583	85.346943	86.640272	88.837454	91.332
8	1	31.5	0.069445	92.021272	0.040950	85.182380	86.013566	87.308201	89.512019	92.021
9	1	32.5	-0.029721	92.696379	0.040909	85.839250	86.669993	87.965401	90.174637	92.696

```
# Adjust the data to fit our format
cdc["Age"]=cdc["Agemos"]/12 # convert age in months to age in fractional years
cdc["Sex"]=cdc["Sex"].replace({1:"male",2:"female"}) # align to our convention

# group the data into percentiles
percentiles=[3,5,10,25,50,75,90,95,97]
percentile_columns=["P"+str(p) for p in percentiles] # names of percentile columns
cdc[percentile_columns]=cdc[percentile_columns]*0.393701 # convert percentile columns from
cdc20=cdc[cdc["Age"]==20].set_index("Sex") # Select the two rows corresponding to 20-year-

print("Height Percentiles for 20-year-old US population [inches]")
cdc20[percentile_columns]
```

```
Height Percentiles for 20-year-old US population [inches]
```

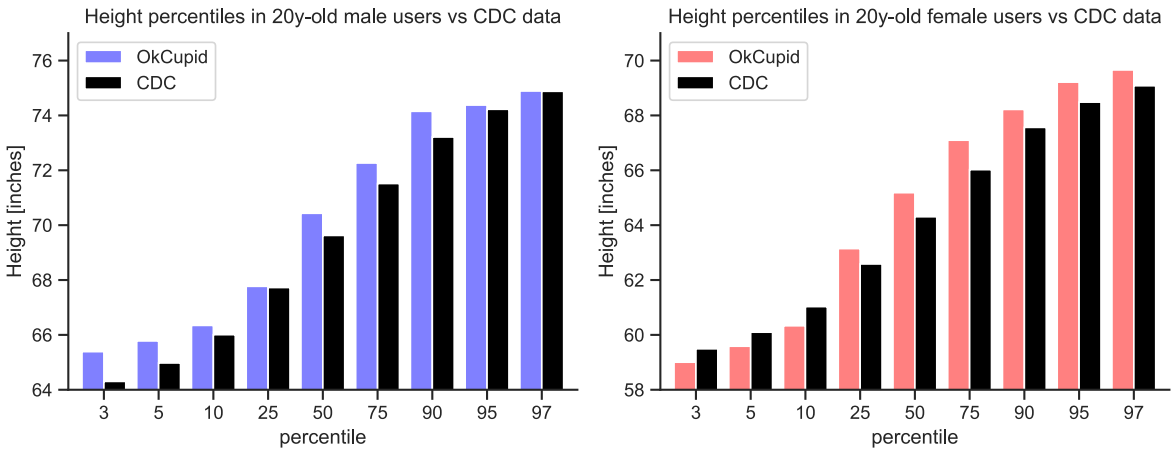
Sex	P3	P5	P10	P25	P50	P75	P90	P95	P97
male	64.304496	64.975961	66.007627	67.725520	69.625720	71.517283	73.212615	74.224061	74.8
female	59.492618	60.098742	61.030770	62.584736	64.306433	66.023163	67.564157	68.484561	69.0

1.9.7 Let's compare the stats for reported heights of the OkCupid 20-year-olds to the CDC stats for 20-year-olds.

```
mheights=males.loc[males["age"]==20,"height_inches"] # heights of 20-year-old males
fheights=females.loc[females["age"]==20,"height_inches"] # heights of 20-year-old females

# To smooth the computation of percentiles, jitter height data by adding
# uniformly distributed noise in the range [-0.5,+0.5]
mheightsj=mheights+np.random.uniform(low=-0.5,high=+0.5,size=(len(mheights),))
fheightsj=fheights+np.random.uniform(low=-0.5,high=+0.5,size=(len(fheights),))

# For each of the available percentiles in CDC data, compute the corresponding percentile
stats=[]
for percentile,percentile_column in zip(percentiles,percentile_columns):
    stats.append({"sex":"male",
                  "percentile":percentile,
                  "CDC":cdc20.loc["male",percentile_column],
                  "OkCupid":mheightsj.quantile(percentile/100)})
    stats.append({"sex":"female",
                  "percentile":percentile,
                  "CDC":cdc20.loc["female",percentile_column],
                  "OkCupid":fheightsj.quantile(percentile/100)})
stats=pd.DataFrame(stats).set_index(["sex","percentile"]).sort_index()

# For each percentile, compute the gap between users and CDC
stats["gap"]=stats["OkCupid"]-stats["CDC"]
```

```
# plot data
fig,(ax1,ax2)=plt.subplots(ncols=2,sharex=True,figsize=(10,4))
stats.loc["male"][["OkCupid","CDC"]].plot.bar(ax=ax1,color=[[0.5,0.5,1],"k"],alpha=1,width
stats.loc["female"][["OkCupid","CDC"]].plot.bar(ax=ax2,color=[[1,0.5,0.5],"k"],alpha=1,wid
ax1.set_ylim([64,77])
ax2.set_ylim([58,71])
ax1.set_ylabel("Height [inches]")
```

```
ax2.set_ylabel("Height [inches]")
ax1.set_title("Height percentiles in 20y-old male users vs CDC data")
ax2.set_title("Height percentiles in 20y-old female users vs CDC data")
for ax in (ax1,ax2):
    sns.despine(ax=ax)
fig.tight_layout()
```



The height statistics of our user population matches remarkably well with CDC data. It looks like the OkCupid users, males and females alike, may be slightly over-reporting their height (still, much less than one could expect), but there could be many other reasonable explanations for this gap. For example, the San-Francisco population may be taller than the general US population. This might be a starting point to further investigate the issue.

1.10 More Pandas Resources

- Lectures on Pandas from UC Berekely https://ds100.org/fa20/lecture/lec05/
- Getting started with Pandas https://pandas.pydata.org/pandas-docs/stable/getting_started/index.html#started
- 10 minute guide to Pandas https://pandas.pydata.org/pandas-docs/stable/user_guide/10min.html
- The Pandas Cookbook. This provides a nice overview of some of the basic Pandas functions. However, it is slightly out of date. https://nbviewer.jupyter.org/github/jvns/pandas-cookbook/tree/master/cookbook/

- Learn Pandas. A set of lessons providing an overview of the Pandas library. https://bitbucket.org/hrojas/learn-pandas/src/master/ Python for Data Science Another set of notebook demonstrating Pandas functionality.

2 Tutorial 2

2.1 Topics

1. Summary statistics
2. Quantiles
3. Histograms
4. Encoding categorical varaibles

2.2 Important Python Packages

- Pandas
- Seaborn
- Matplotlib

```
# import necessary packages
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
```

2.3 Today's datasets

- City Temperatures – Daily temperature for different international cities (download .csv here)

```
# read CSV
temp_df = pd.read_csv('city_temp.csv')
```

```
# examine the data -- in visual studio code we can also do this another way
temp_df
```

	Country	City	Month	Day	Year	AvgTemperature
0	Malawi	Lilongwe	1	1	1995	69.5
1	Malawi	Lilongwe	1	2	1995	69.5
2	Malawi	Lilongwe	1	3	1995	67.5
3	Malawi	Lilongwe	1	4	1995	68.5
4	Malawi	Lilongwe	1	5	1995	66.7
...	...	...	...	...	...	...
47392	US	Rochester	5	9	2020	33.9
47393	US	Rochester	5	10	2020	41.4
47394	US	Rochester	5	11	2020	40.7
47395	US	Rochester	5	12	2020	38.9
47396	US	Rochester	5	13	2020	34.0

```
# which cities do we have data for
temp_df['City'].unique()
```

```
array(['Lilongwe', 'Capetown', 'Tel Aviv', 'Amman', 'Beirut', 'Rochester'],
      dtype=object)
```

```
# isolate data from a single city (e.g., Tel Aviv)
city = 'Tel Aviv'
TA_temp = temp_df.loc[temp_df['City'] == city]
TA_temp
```

	Country	City	Month	Day	Year	AvgTemperature
14959	Israel	Tel Aviv	1	1	1995	57.3
14960	Israel	Tel Aviv	1	2	1995	56.1
14961	Israel	Tel Aviv	1	3	1995	55.9
14962	Israel	Tel Aviv	1	4	1995	56.9
14963	Israel	Tel Aviv	1	5	1995	56.6
...	...	...	...	...	...	...
19595	Israel	Tel Aviv	9	11	2007	79.5
19596	Israel	Tel Aviv	9	12	2007	79.7
19597	Israel	Tel Aviv	9	13	2007	79.7
19598	Israel	Tel Aviv	9	14	2007	79.6
19599	Israel	Tel Aviv	9	15	2007	80.0

```
# get summary statistics for a single city
TA_temp['AvgTemperature'].describe()
```

```
count    4641.000000
mean       54.020448
std        50.624184
min       -99.000000
25%        59.400000
50%        68.700000
75%        78.600000
max        88.500000
Name: AvgTemperature, dtype: float64
```

```
TA_temp = TA_temp.loc[TA_temp['AvgTemperature']>(-50)]
```

```
# convert to Celsius
TA_temp['AvgTemp_C'] = (TA_temp['AvgTemperature'] - 32)*(5/9)
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_82655/3623248864.py:2: SettingWitl
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guid
  TA_temp['AvgTemp_C'] = (TA_temp['AvgTemperature'] - 32)*(5/9)
```

```
# get summary stasitics in Celsius
TA_temp['AvgTemp_C'].describe()
```

```
count    4196.000000
mean       21.249325
std         5.193370
min         7.277778
25%        16.555556
50%        21.611111
75%        26.277778
max        31.388889
Name: AvgTemp_C, dtype: float64
```

```
# get the mean for the city you chose
mean_temp = TA_temp['AvgTemp_C'].mean()
print(f"The mean temperature in {city} is: {mean_temp:.2f} degrees Celcius")
```

```
The mean temperature in Tel Aviv is: 21.25 degrees Celcius
```

```
# get the median temperature for the city you chose
median_temp = TA_temp['AvgTemp_C'].median()
print(f"The median temperature in {city} is: {median_temp:.2f} degrees Celcius")
```

```
The median temperature in Tel Aviv is: 21.61 degrees Celcius
```

```
# get the 10th percentile for the city you chose
percentile_10 = TA_temp['AvgTemp_C'].quantile(.1)
print(f"The tenth percentile in {city} is: {percentile_10:.2f} degrees Celcius")
```

```
The tenth percentile in Tel Aviv is: 14.17 degrees Celcius
```

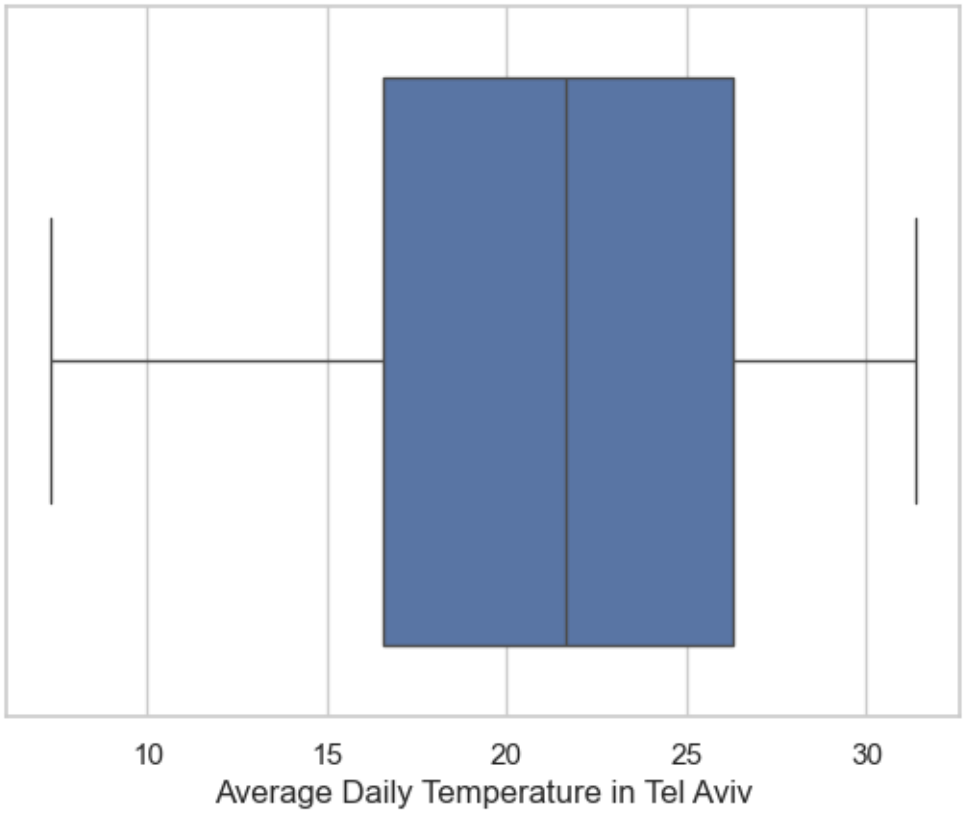
```
# get the 90th percentile for the city you chose
percentile_90 = TA_temp['AvgTemp_C'].quantile(.9)
print(f"The ninetieth percentile in {city} is: {percentile_90:.2f} degrees Celcius")
```

```
The ninetieth percentile in Tel Aviv is: 27.67 degrees Celcius
```

```
# begin plotting
sns.set_theme(style="whitegrid")
```

```
# make a box plot of temperature for the city you chose
fig, ax = plt.subplots()
# ax = sns.boxplot(x=TA_temp.AvgTemp_C)
sns.boxplot(x=TA_temp['AvgTemp_C'], ax=ax)
ax.set(xlabel=f'Average Daily Temperature in {city}')
```

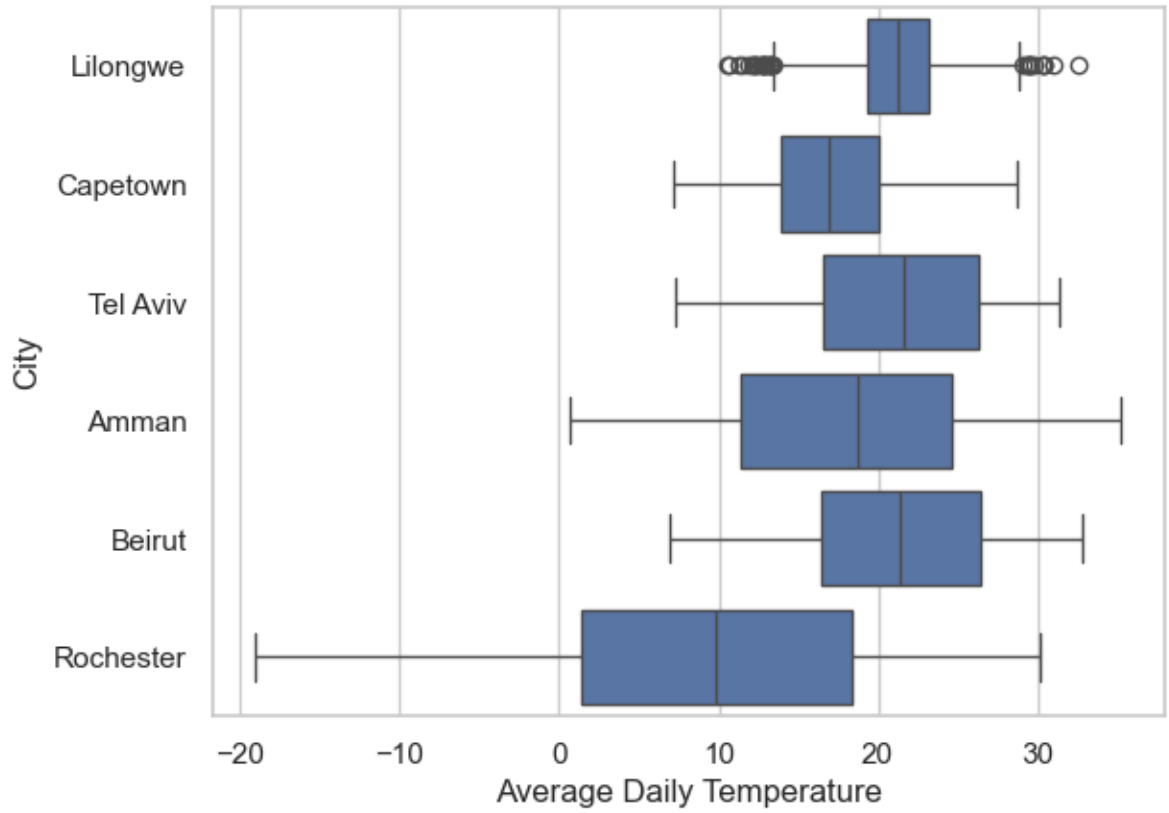
```
[Text(0.5, 0, 'Average Daily Temperature in Tel Aviv')]
```




```
# compare all the cities

# clean data
temp_df = temp_df.loc[temp_df['AvgTemperature']>(-50)]
temp_df['AvgTemp_C'] = (temp_df['AvgTemperature'] - 32)*(5/9)

# plot
fig, ax = plt.subplots()
sns.boxplot(x=temp_df['AvgTemp_C'], y=temp_df['City'], ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

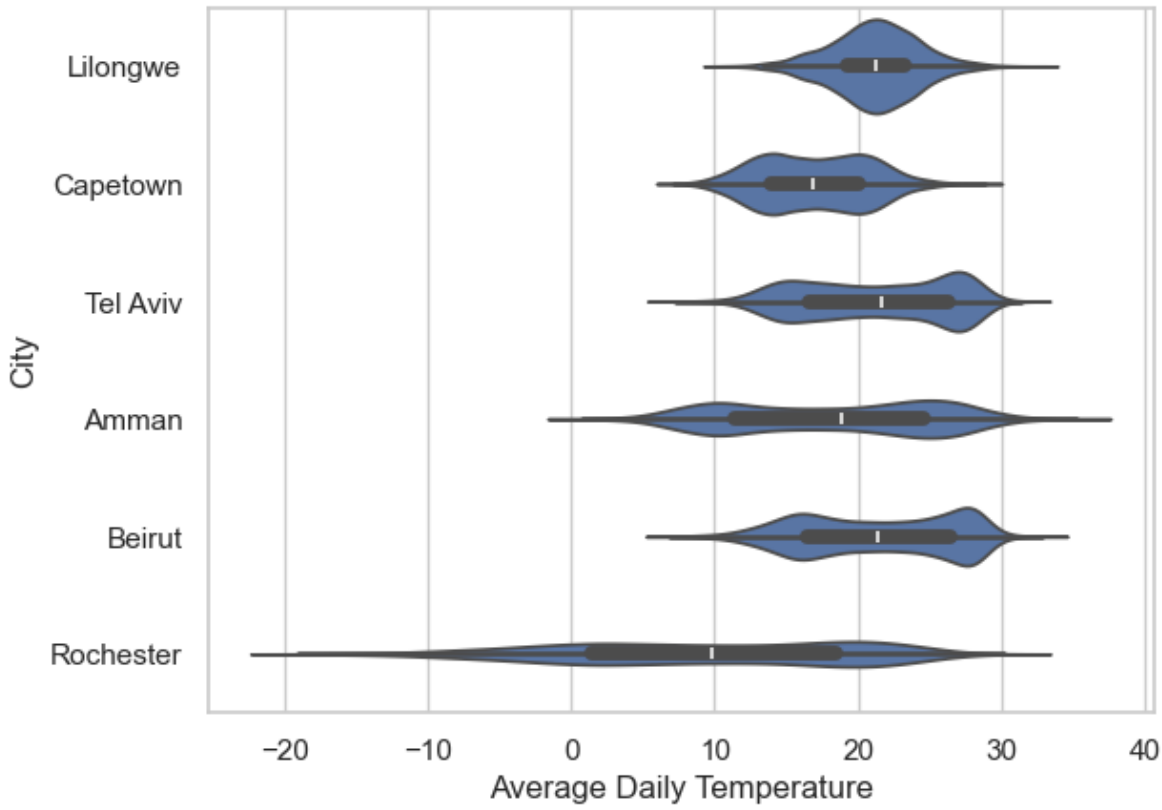
```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_82655/3601528637.py:5: SettingWitl
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  temp_df['AvgTemp_C'] = (temp_df['AvgTemperature'] - 32)*(5/9)
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```




```
# make a violin plot of the temperature
fig, ax = plt.subplots()
sns.violinplot(x=temp_df['AvgTemp_C'], y=temp_df['City'], ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```

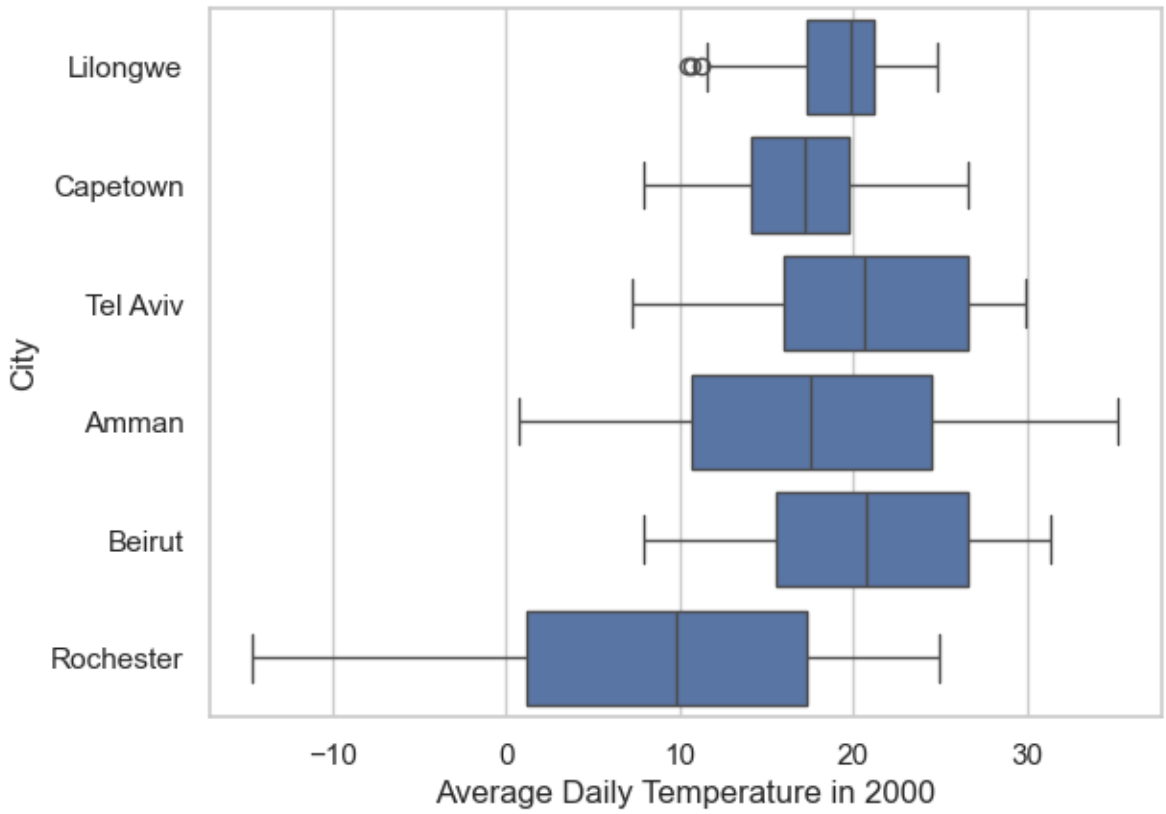


2.4 Try out

1. plot the Average Daily Temperature of the year 2000 for all cities
2. plot the Average Daily Temperature of January 1st for all cities in all years

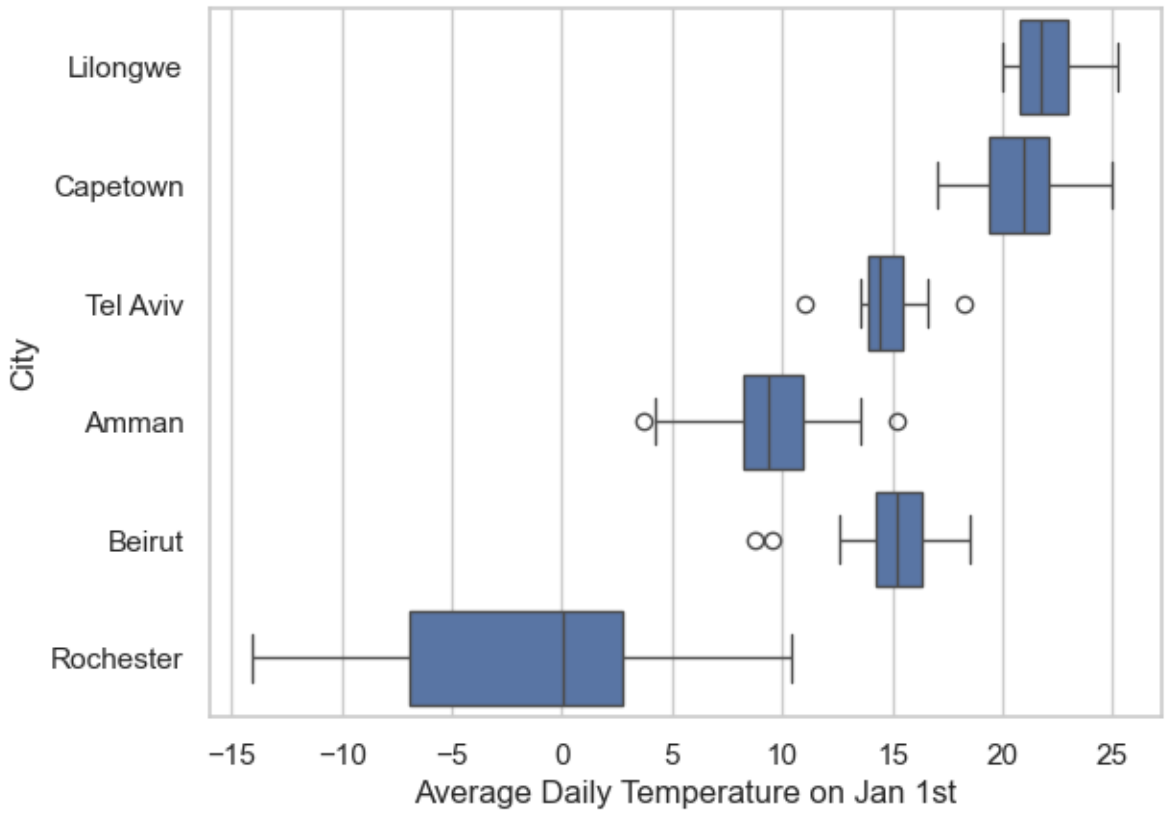
```
milenium = temp_df.loc[temp_df['Year'] == 2000]
# plot
fig, ax = plt.subplots()
sns.boxplot(data=milenium, x='AvgTemp_C', y='City', ax=ax)
ax.set(xlabel='Average Daily Temperature in 2000')
```

```
[Text(0.5, 0, 'Average Daily Temperature in 2000')]
```



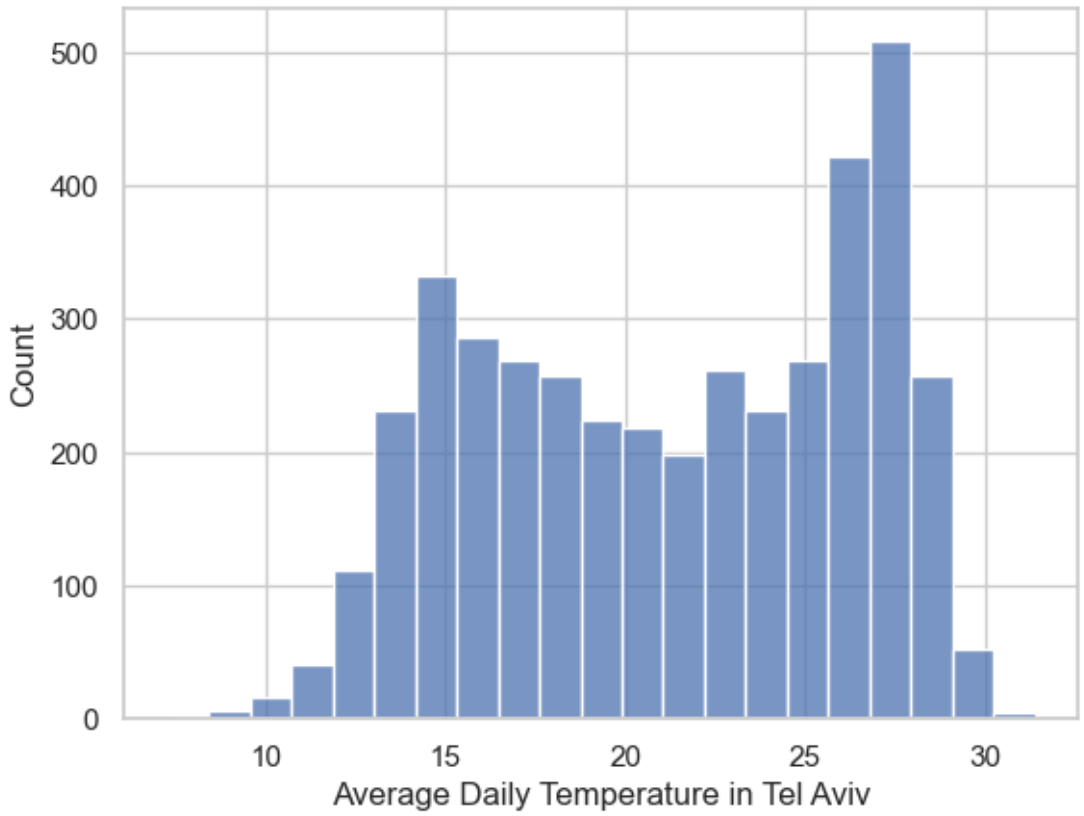

```
jan_df = temp_df.loc[(temp_df['Day'] == 1)&((temp_df['Month'] == 1))]
# plot
fig, ax = plt.subplots()
sns.boxplot(data=jan_df, x='AvgTemp_C', y='City', ax=ax)
ax.set(xlabel='Average Daily Temperature on Jan 1st')
```

```
[Text(0.5, 0, 'Average Daily Temperature on Jan 1st')]
```



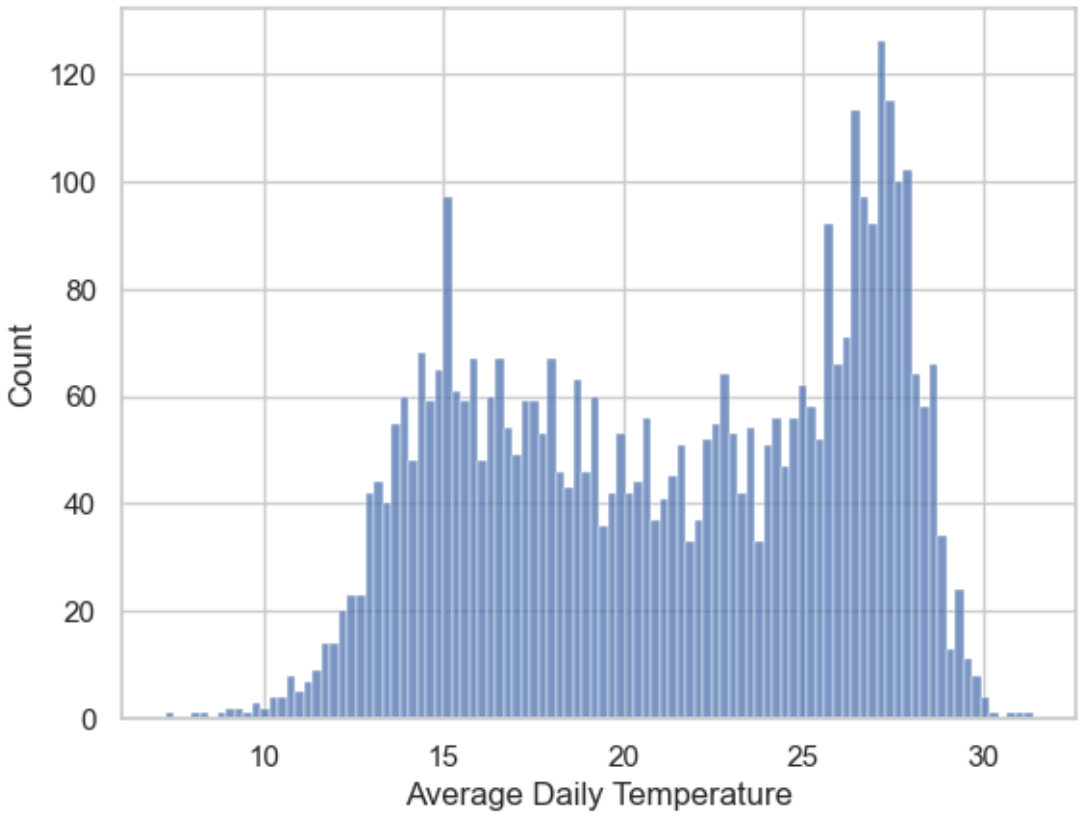

```
# make a histogram of the data for the city you chose
fig, ax = plt.subplots()
sns.histplot(x=TA_temp['AvgTemp_C'], ax=ax)
ax.set(xlabel=f'Average Daily Temperature in {city}')
```

```
[Text(0.5, 0, 'Average Daily Temperature in Tel Aviv')]
```



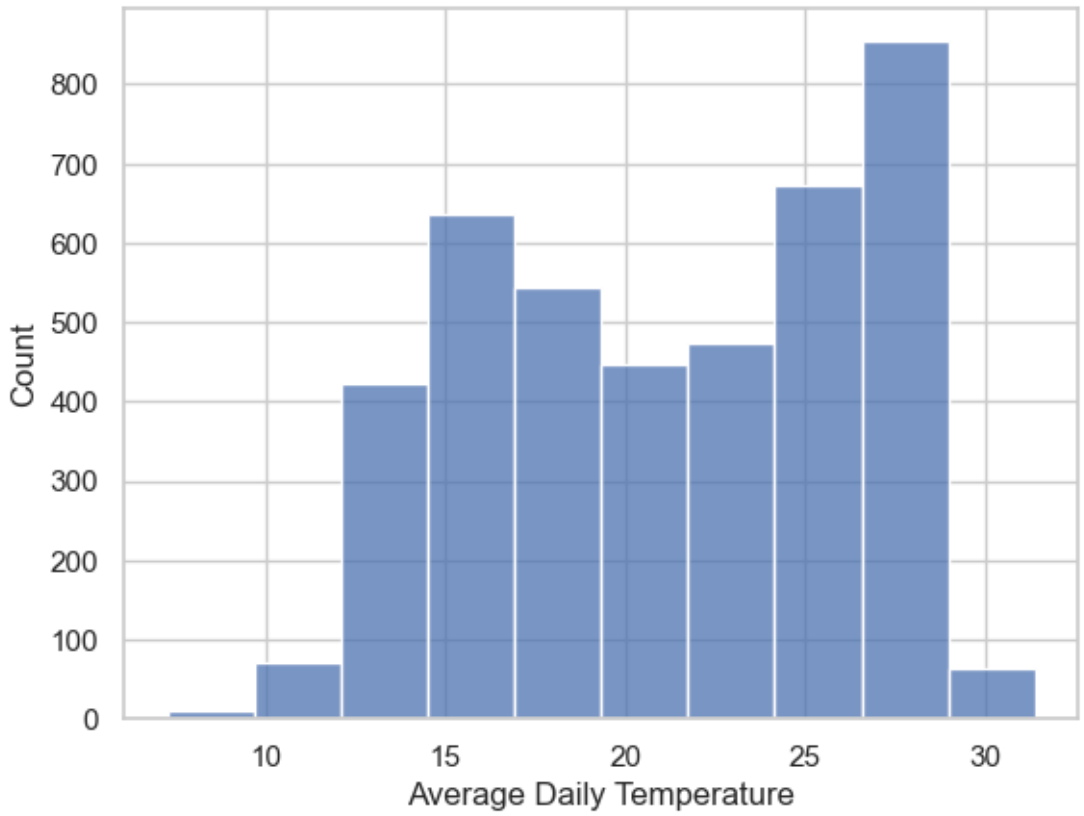

```
# play around with the bin size for the histogram -- try more bins
fig, ax = plt.subplots()
sns.histplot(x=TA_temp['AvgTemp_C'], bins=100, ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```



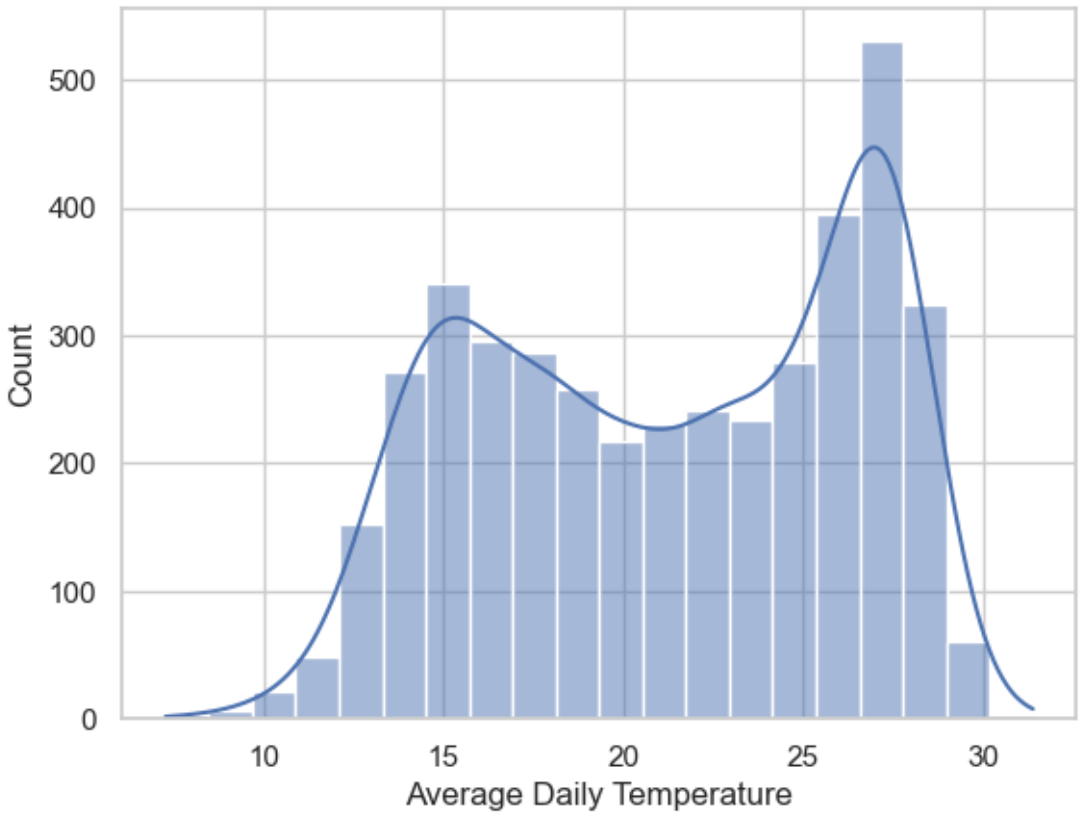

```
# now fewer bins
fig, ax = plt.subplots()
sns.histplot(x=TA_temp.AvgTemp_C, bins=10, ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```



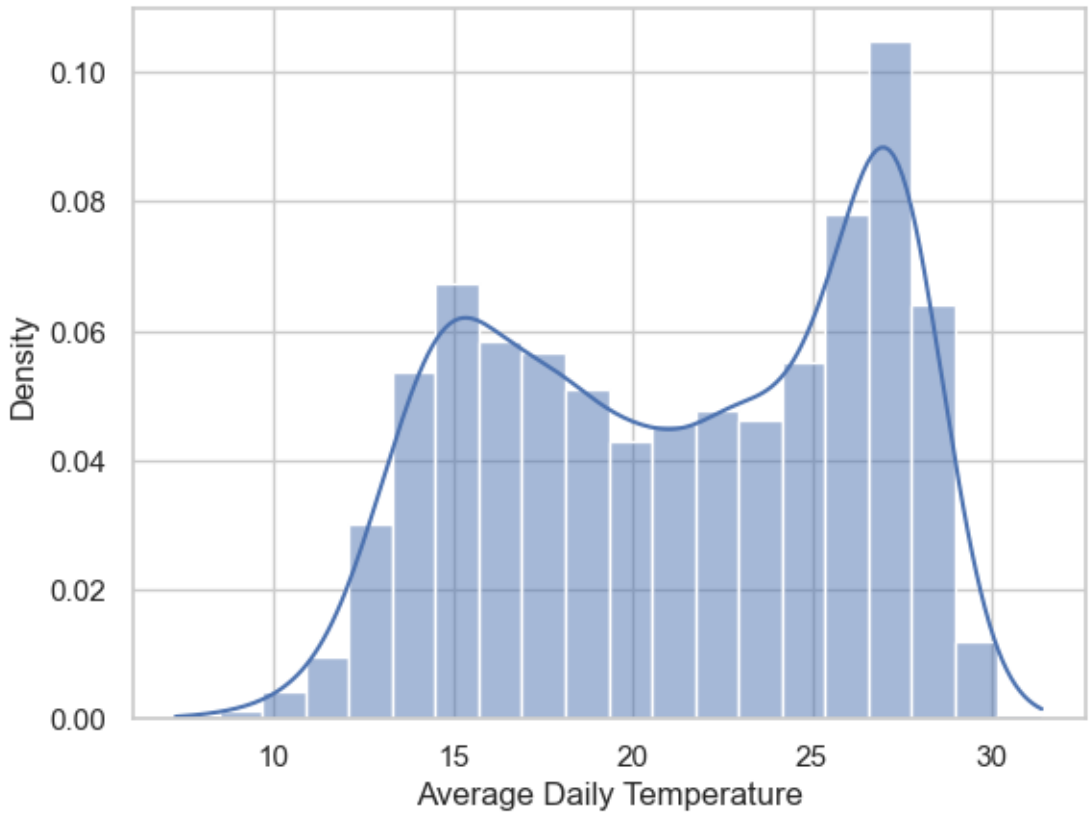

```
# add kernel density estimator
fig, ax = plt.subplots()
sns.histplot(x=TA_temp.AvgTemp_C, bins=20, kde = True, ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```




```
# how can we normalize the histogram data?
fig, ax = plt.subplots()
sns.histplot(x=TA_temp['AvgTemp_C'], bins=20, kde=True, stat="density", ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

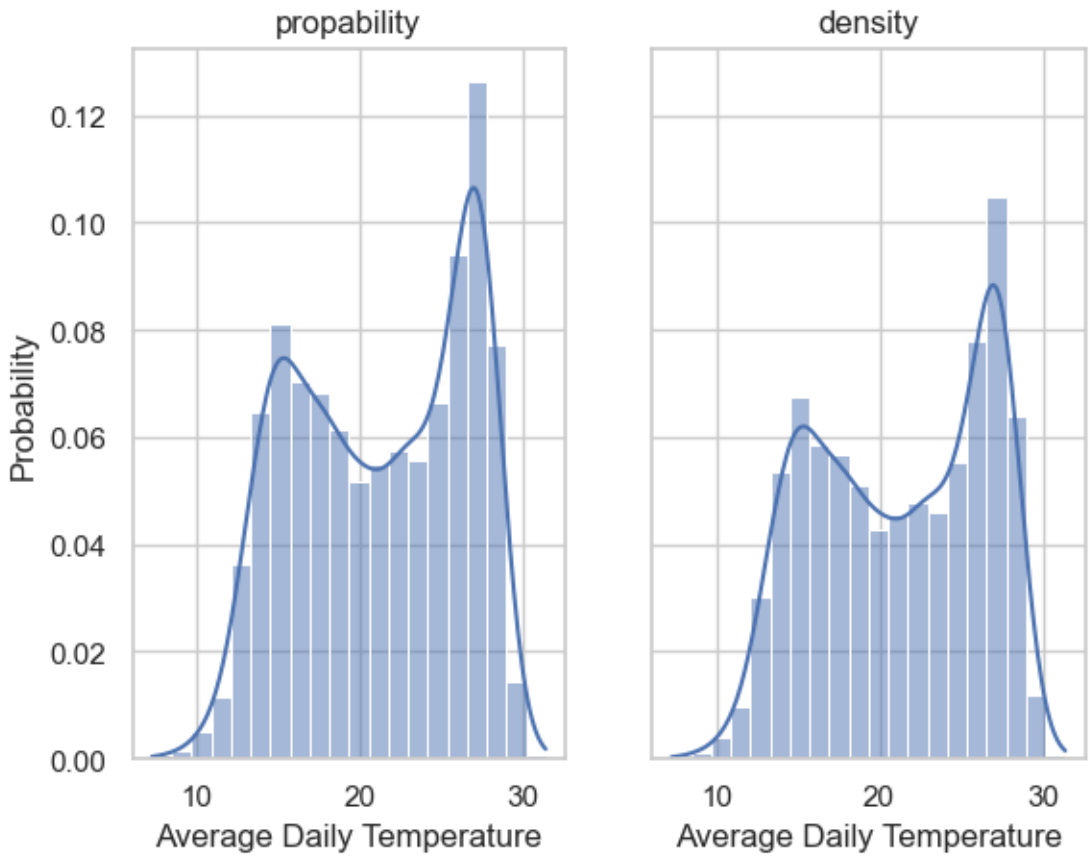
```
[Text(0.5, 0, 'Average Daily Temperature')]
```



what's the difference between the "density" stat and "probablity" stat? read the documentation.

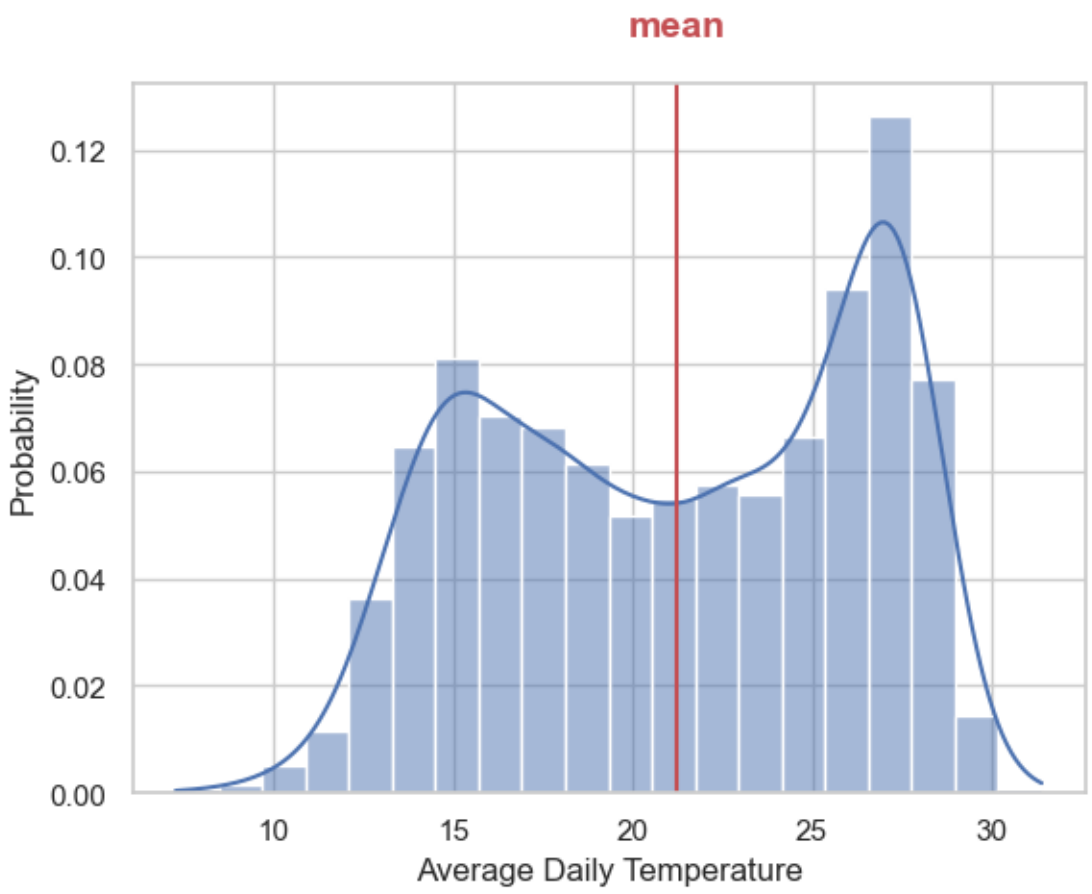
```
# plotting 2 side by side
fig, ax = plt.subplots(1,2, sharey=True)

sns.histplot(x=TA_temp.AvgTemp_C, bins=20, kde=True, stat="probability", ax=ax[0])
ax[0].set(title='propability', xlabel='Average Daily Temperature')

sns.histplot(x=TA_temp.AvgTemp_C, bins=20, kde=True, stat="density", ax=ax[1])
ax[1].set(title='density', xlabel='Average Daily Temperature')
```

```
[Text(0.5, 1.0, 'density'), Text(0.5, 0, 'Average Daily Temperature')]
```




```
# add the mean to the plot
mean_temp = TA_temp['AvgTemp_C'].mean()

fig, ax = plt.subplots()
sns.histplot(x=TA_temp['AvgTemp_C'],
             bins=20,
             kde=True,
             stat="probability",
             ax=ax,
             )
ax.set(xlabel='Average Daily Temperature')
ax.axvline(mean_temp, label='mean', color='r')
ax.text(mean_temp,0.14, 'mean', va='bottom',
        ha='center', fontsize=14, weight='bold', color='r')
```

```
Text(21.249324753733717, 0.14, 'mean')
```

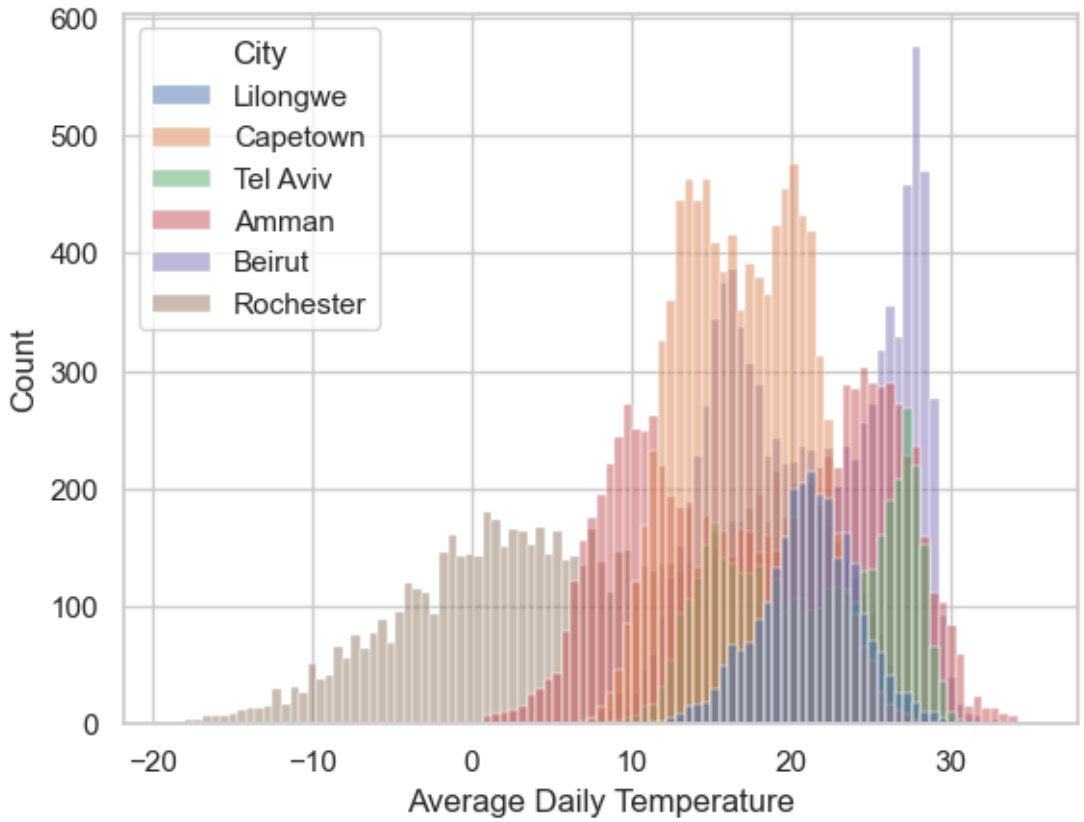


2.5 Try out

- add the mode
- add the median

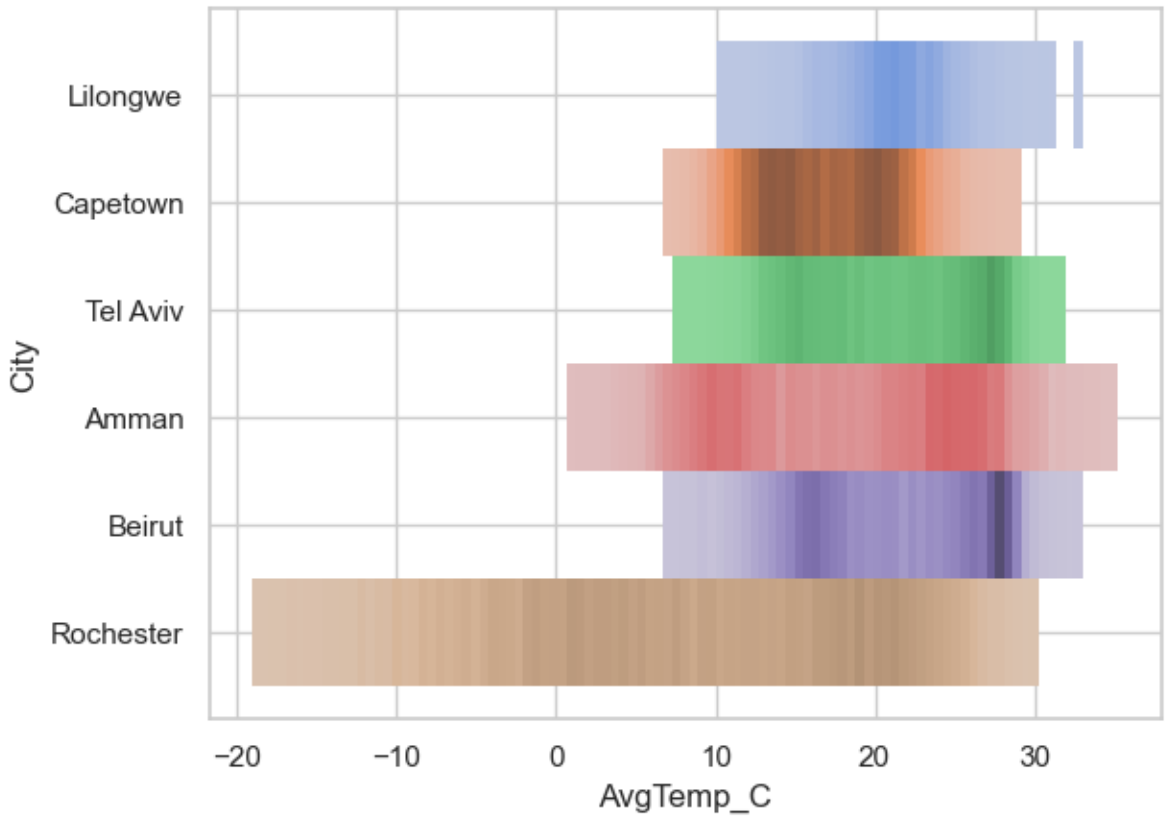
```
# make a histogram of the data
fig, ax = plt.subplots()
sns.histplot(x=temp_df['AvgTemp_C'], hue=temp_df['City'], ax=ax)
ax.set(xlabel='Average Daily Temperature')
```

```
[Text(0.5, 0, 'Average Daily Temperature')]
```




```
# another type of histogram
fig, ax = plt.subplots()
sns.histplot(x=temp_df['AvgTemp_C'], y=temp_df['City'],
             hue=temp_df['City'], legend=False, ax=ax)
```

```
<Axes: xlabel='AvgTemp_C', ylabel='City'>
```



2.6 Encoding categorical variables

Sometimes, for reasons that will be clear on the HW, we'll want to encode our categorical variables so that they are numbers instead.

There are many ways that we can achive this.

Here will learn one, for more examples see: https://pbpython.com/categorical-encoding.html

```
# what are our cities again?
temp_df['City'].unique()
```

```
array(['Lilongwe', 'Capetown', 'Tel Aviv', 'Amman', 'Beirut', 'Rochester'],
      dtype=object)
```

```
# dictionary for encoding cities (note: we can encode more than one variable at a time)
cleanup_cities = {"City": {"Lilongwe": 1,
                           "Capetown": 2,
                           "Tel Aviv": 3,
                           "Amman": 4,
                           "Beirut": 5,
                           "Rochester": 6}}
```

```
# new dataframe with encoded values
temp_df_encoded = temp_df.replace(cleanup_cities)
temp_df_encoded.dtypes
```

```
Country            object
City                int64
Month               int64
Day                 int64
Year                int64
AvgTemperature    float64
AvgTemp_C         float64
dtype: object
```

```
# option 2 -- use Pandas

# what are our data types?
temp_df.dtypes
```

```
Country            object
City               object
Month               int64
Day                 int64
Year                int64
AvgTemperature    float64
AvgTemp_C         float64
dtype: object
```

```
# assign city to be a categorical variable
temp_df["City"] = temp_df["City"].astype('category')
temp_df.dtypes
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_82655/1724604918.py:2: SettingWitl
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  temp_df["City"] = temp_df["City"].astype('category')
```

```
Country             object
City              category
Month                int64
Day                  int64
Year                 int64
AvgTemperature     float64
AvgTemp_C          float64
dtype: object
```

```
# use codes to encode variable
temp_df["City_encoded"] = temp_df["City"].cat.codes
temp_df.dtypes
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_82655/3917241889.py:2: SettingWitl
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide
  temp_df["City_encoded"] = temp_df["City"].cat.codes
```

```
Country             object
City              category
Month                int64
Day                  int64
Year                 int64
AvgTemperature     float64
AvgTemp_C          float64
City_encoded          int8
dtype: object
```

3 Tutorial 3

3.1 Principal Component Analysis

Principal Component Analysis, or PCA for short, is a method for reducing the dimensionality of data.

It can be thought of as a projection method where data with m-columns (features) is projected into a subspace with m or fewer columns, whilst retaining the essence of the original data.

The PCA method can be described and implemented using the tools of linear algebra.

PCA is an operation applied to a dataset, represented by an n x m matrix A that results in a projection of A which we will call B.

By default, PCA creates as many axes as there are dimensions to a given dataset but it ranks these directions, called principal components (PCs), in the order of importance: the first PC always captures the most amount of data variance possible, the second one shows the second-largest amount of variance, and so forth.

After the data has been projected into this new subspace, we might drop out some of the axes and thus reduce dimensionality without losing much important information.

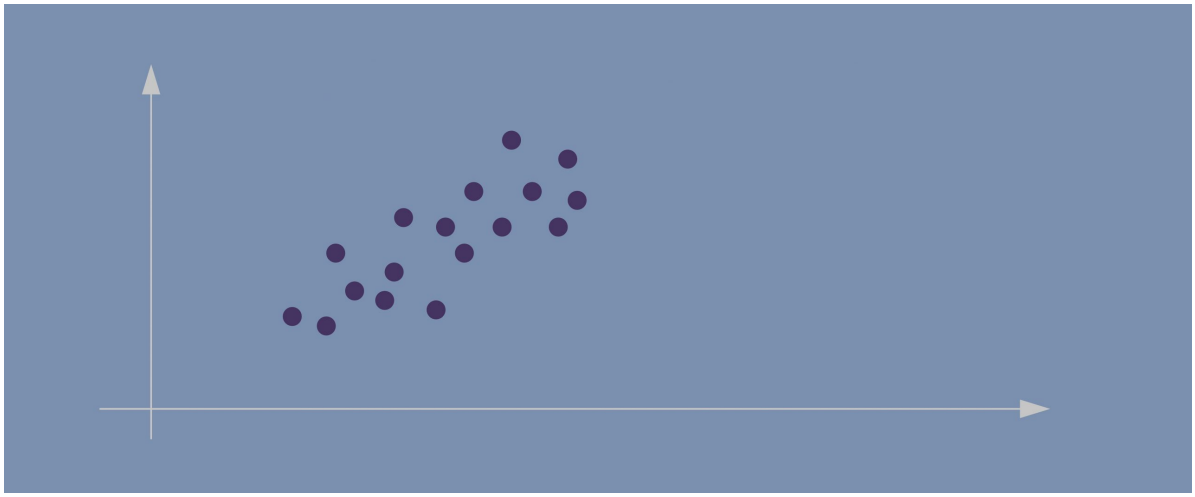
For more details on the steps:

https://machinelearningmastery.com/calculate-principal-component-analysis-scratch-python/

3.1.1 How does PCA work?

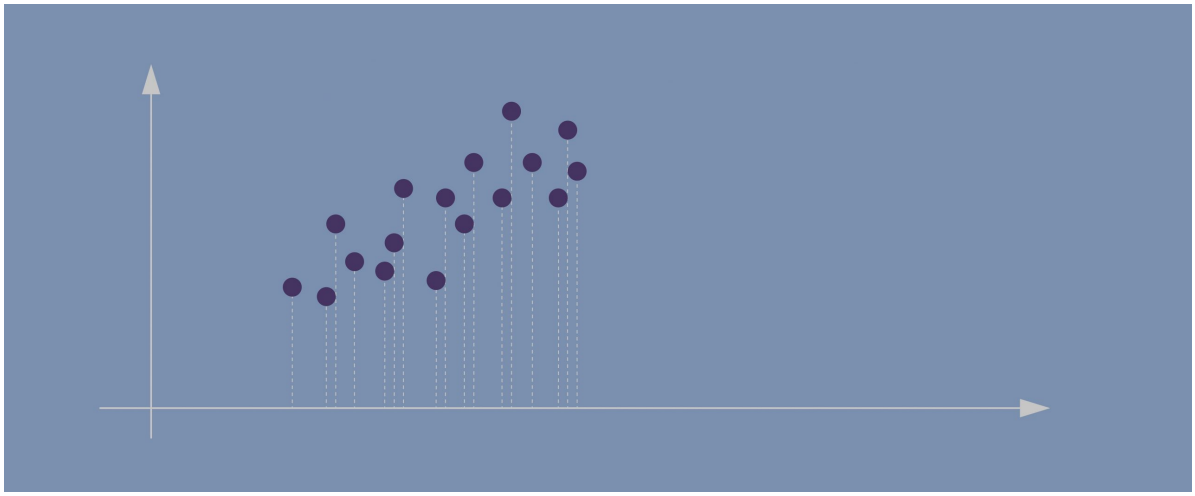
1. Imagine this is our dataset that we're trying to cluster; we only have two dimensions.

```
%matplotlib inline
from IPython.display import Image
Image('image_1.png')
```



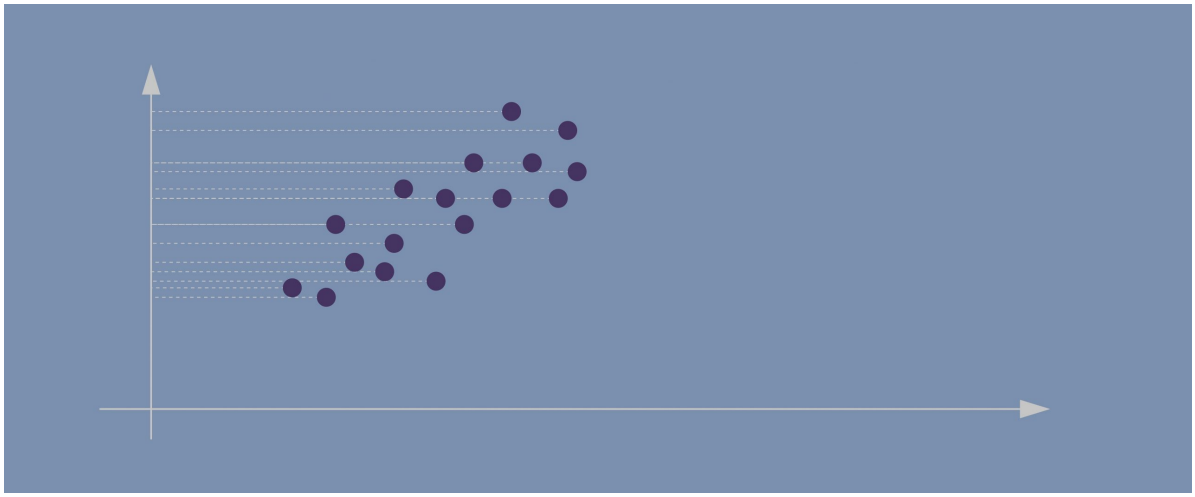
2. If we project the data onto the horizontal axis (our attribute 1) we won't see much spread; it will show a nearly equal distribution of the observations.

```
Image('image_2.png')
```



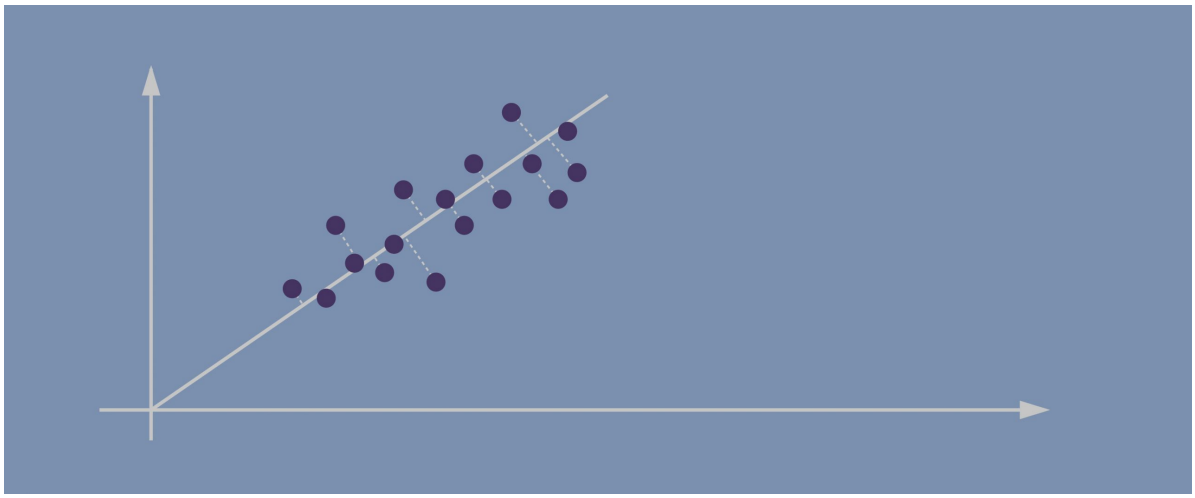
3. Attribute 2, obviously, isn't hugely helpful either.

```
Image('image_3.png')
```



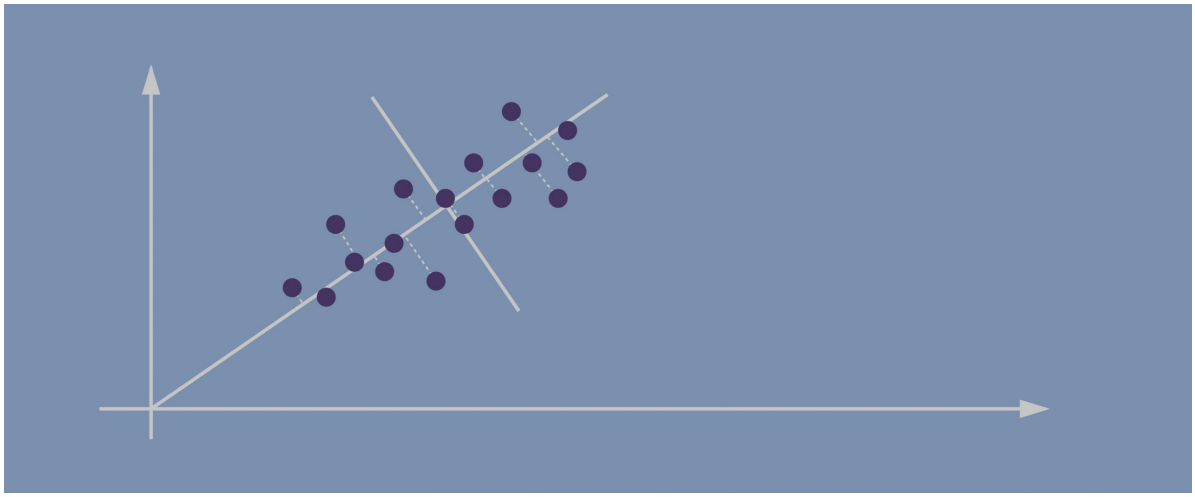
4. Evidently, the data points in our case are spreading diagonally so we need a new line that would better capture this.

```
Image('image_4.png')
```



5. The second PC must represent the second maximum amount of variance; it's going to be a line that's orthogonal to our first axis. *Due to PCA's math being based on eigenvectors and eigenvalues, new principal components will always come out orthogonal to the ones before them.

```
Image('image_5.png')
```



3.1.1.1 Important!

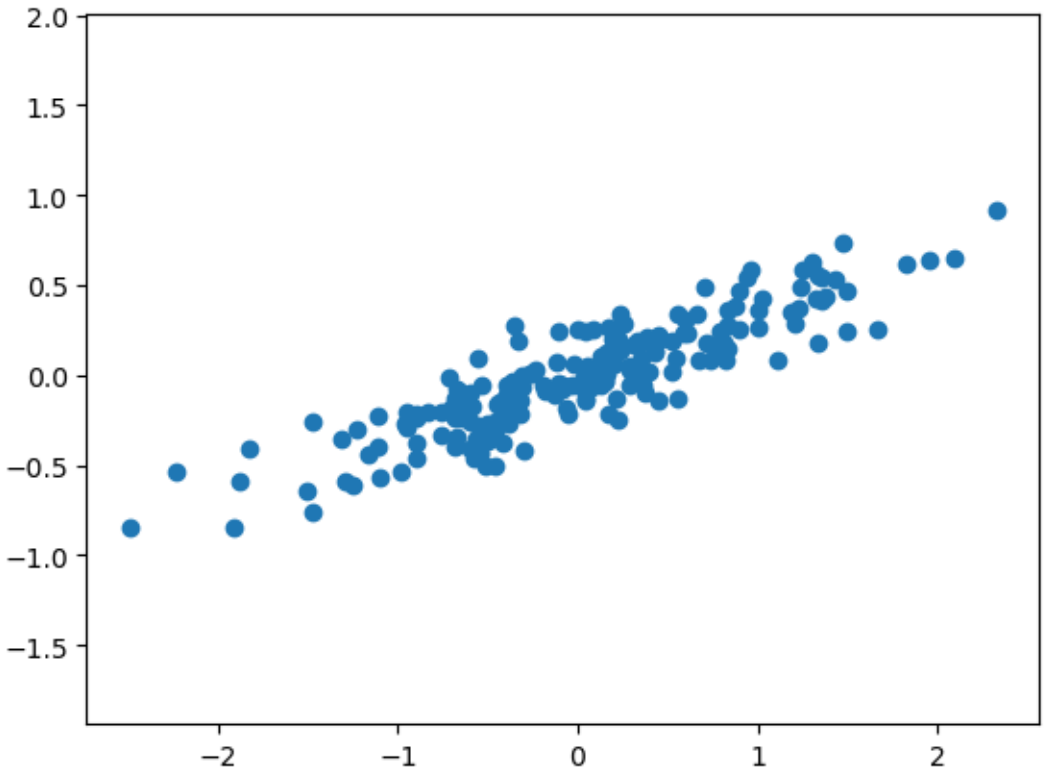
Before applying PCA, we must ensure that all our attributes (dimensions) are centered around zero and have a standard deviation of 1. The method won't work if we have entirely different scales for our data.

3.2 Example

https://jakevdp.github.io/PythonDataScienceHandbook/05.09-principal-component-analysis.html

```
# import packages
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import pandas as pd
```

```
# create and plot some random data
rng = np.random.RandomState(1)
X = np.dot(rng.rand(2, 2), rng.randn(2, 200)).T
plt.scatter(X[:, 0], X[:, 1])
plt.axis('equal');
```




```
# use sklearn to do PCA
pca = PCA(n_components=2)
pca.fit(X);
```

`pca.components_` returns the direction of each principal component, sorted by decreasing explained variance. These are the **eigenvectors** of the decomposition.

```
# get components
print(pca.components_)
```

```
[[-0.94446029 -0.32862557]
 [-0.32862557  0.94446029]]
```

`pca.explained_variance_` returns the **variance** explained by each component, sorted by decreasing magnitude. These are the **eigenvalues** of the decomposition. They do not sum up to 1.

```
# get explained variance
print(pca.explained_variance_)
```

```
[0.7625315 0.0184779]
```

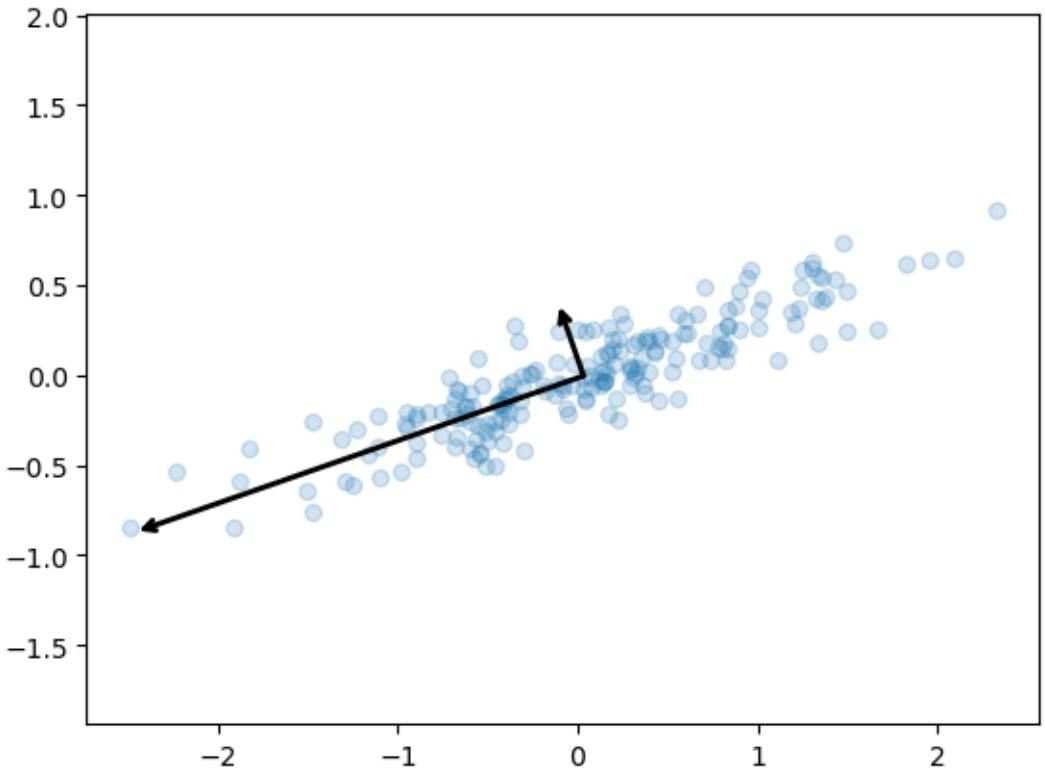
`pca.explained_variance_ratio_` returns the **proportion of total variance** explained by each component. They sum up to 1. These are computed as:

```
explained_variance_ratio_[i] = explained_variance_[i] / sum(explained_variance_)
```

```
# get explained variance ratio
print(pca.explained_variance_ratio_)
```

To see what these numbers mean, let's visualize them as vectors over the input data, using the "components" to define the direction of the vector, and the "explained variance" to define the squared-length of the vector.

```
def draw_vector(v0, v1, ax=None):
    ax = ax or plt.gca()
    arrowprops=dict(arrowstyle='->',
                    linewidth=2,
                    shrinkA=0, shrinkB=0)
    ax.annotate('', v1, v0, arrowprops=arrowprops)

# plot data
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
for length, vector in zip(pca.explained_variance_, pca.components_):
    v = vector * 3 * np.sqrt(length)
    draw_vector(pca.mean_, pca.mean_ + v)
plt.axis('equal');
```



These vectors represent the principal axes of the data, and the length of the vector is an indication of how "important" that axis is in describing the distribution of the data—more precisely, it is a measure of the variance of the data when projected onto that axis. The projection of each data point onto the principal axes are the "principal components" of the data.

3.2.1 Dimensional reduction

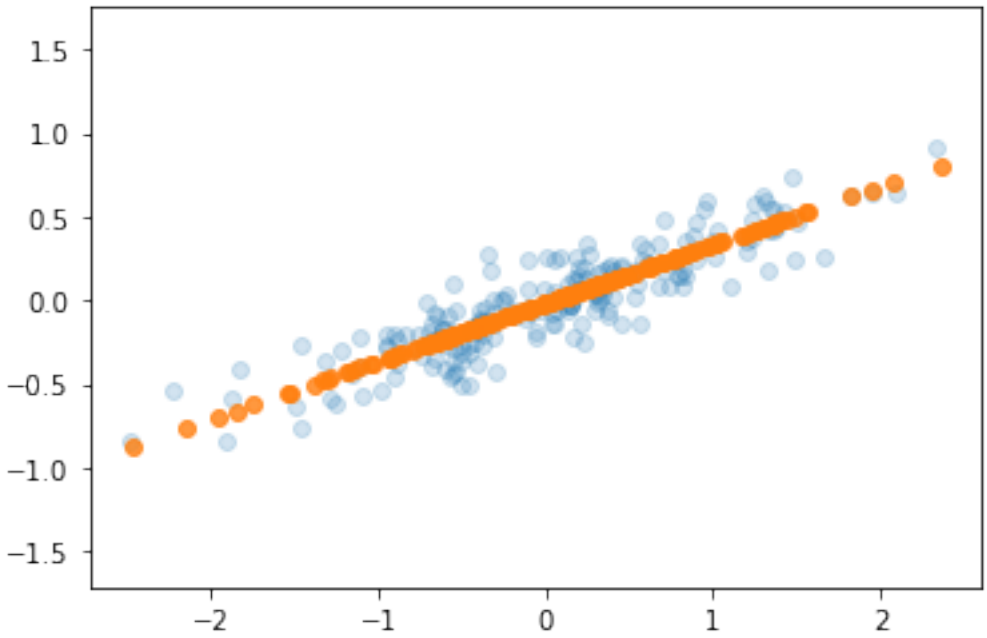
Using PCA for dimensionality reduction involves zeroing out one or more of the smallest principal components, resulting in a lower-dimensional projection of the data that preserves the maximal data variance.

```
pca = PCA(n_components=1)
pca.fit(X)
X_pca = pca.transform(X)
print("original shape:   ", X.shape)
print("transformed shape:", X_pca.shape)
```

```
original shape:    (200, 2)
transformed shape: (200, 1)
```

The transformed data has been reduced to a single dimension. To understand the effect of this dimensionality reduction, we can perform the inverse transform of this reduced data and plot it along with the original data.

```
X_new = pca.inverse_transform(X_pca)
plt.scatter(X[:, 0], X[:, 1], alpha=0.2)
plt.scatter(X_new[:, 0], X_new[:, 1], alpha=0.8)
plt.axis('equal');
```



The light points are the original data, while the dark points are the projected version. This makes clear what a PCA dimensionality reduction means: the information along the least important principal axis or axes is removed, leaving only the component(s) of the data with the highest variance. The fraction of variance that is cut out (proportional to the spread of points about the line formed in this figure) is roughly a measure of how much "information" is discarded in this reduction of dimensionality.

This reduced-dimension dataset is in some senses "good enough" to encode the most important relationships between the points: despite reducing the dimension of the data by 50%, the overall relationship between the data points are mostly preserved.

3.2.2 Example 2: PCA to Speed-up Machine Learning Algorithms

The MNIST database of handwritten digits has 784 feature columns (784 dimensions).

We can break it into a training set of 60,000 examples, and a test set of 10,000 examples.

https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60

```
# get MNIST dataset
from sklearn.datasets import fetch_openml
mnist = fetch_openml('mnist_784')
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/datasets/_openml.py:968: FutureWarning: 
  warn(
```

```
# the data consists of images -- there are 70,000 images with 784 dimensions (784 features
mnist.data.shape
```

```
(70000, 784)
```

```
# each image is a number
mnist.target[0:10]
```

```
0    5
1    0
2    4
3    1
4    9
5    2
6    1
7    3
8    1
9    4
Name: class, dtype: category
Categories (10, object): ['0', '1', '2', '3', ..., '6', '7', '8', '9']
```

```
# split into training and test sets
from sklearn.model_selection import train_test_split
```

```
# test_size: what proportion of original data is used for test set
test_size = 1/7.0
(train_img,
 test_img,
 train_lbl,
 test_lbl) = train_test_split(mnist.data,
                              mnist.target,
                              test_size=test_size,
                              random_state=0)
```

```
# standardize the data
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()

# fit on training set only.
scaler.fit(train_img)

# apply transform to both the training set and the test set.
train_img = scaler.transform(train_img)
test_img = scaler.transform(test_img)
```

3.2.3 Apply PCA

Notice the code below uses `PCA(0.95)` for the number of components parameter. This means that scikit-learn will automatically choose the minimum number of principal components such that the cumulative sum of the `explained_variance_ratio_` is at least 95%. In other words, the selected components together retain 95% of the total variance present in the original data.

```
# make an instance of the model
pca = PCA(.95)
```

```
# fit on training data set
pca.fit(train_img)
```

```
PCA(n_components=0.95)
```

```
# how many components are there?
pca.n_components_
```

327

3.3 What do we gain from this?

```
# apply the transform to both the training set and the test set.
train_img = pca.transform(train_img)
test_img = pca.transform(test_img)
```

```
# let's make a logistic model
from sklearn.linear_model import LogisticRegression

# default solver is incredibly slow so changed to 'lbfgs'
logisticRegr = LogisticRegression(solver = 'lbfgs', max_iter=1000)

# train model
logisticRegr.fit(train_img, train_lbl)
```

```
LogisticRegression(max_iter=1000)
```

```
# predict for multiple observations (images) at once
print(f'Predicted\t{logisticRegr.predict(test_img[0:10])}')
print(f'Observed\t{np.array(test_lbl[0:10])}')
```

```
Predicted   ['0' '4' '1' '2' '4' '7' '7' '1' '1' '7']
Observed    ['0' '4' '1' '2' '7' '9' '7' '1' '1' '7']
```

```
# how does the model do?
score = logisticRegr.score(test_img, test_lbl)
score
```

```
0.9184
```

how does changing the amount of variance affect the model?

```
pd.DataFrame(data = [[1.00, 784, 48.94, .9158],
                     [.99, 541, 34.69, .9169],
```

```
                [.95, 330, 13.89, .92],
                [.90, 236, 10.56, .9168],
                [.85, 184, 8.85, .9156]],
          columns = ['Variance Retained',
                     'Number of Components',
                     'Time (seconds)',
                     'Accuracy'])
```

	Variance Retained	Number of Components	Time (seconds)	Accuracy
0	1.00	784	48.94	0.9158
1	0.99	541	34.69	0.9169
2	0.95	330	13.89	0.9200
3	0.90	236	10.56	0.9168
4	0.85	184	8.85	0.9156

3.4 Looking at the data a more broadly - pair plot, clustering, ploting PCA

seaborn has some powerful functions for visualisation of the whole dataset

```
# import packages
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.decomposition import PCA
import pandas as pd
```

Load the breast cancer dataset. In this dataset features are computed from a digitized image of a fine needle aspirate (FNA) of a breast mass. They describe characteristics of the cell nuclei present in the image.

```
from sklearn.datasets import load_breast_cancer
import pandas as pd

cancer_data = load_breast_cancer()
data = pd.DataFrame(cancer_data.data, columns=cancer_data.feature_names)
all_data = data.copy()
all_data['target'] = cancer_data.target
```

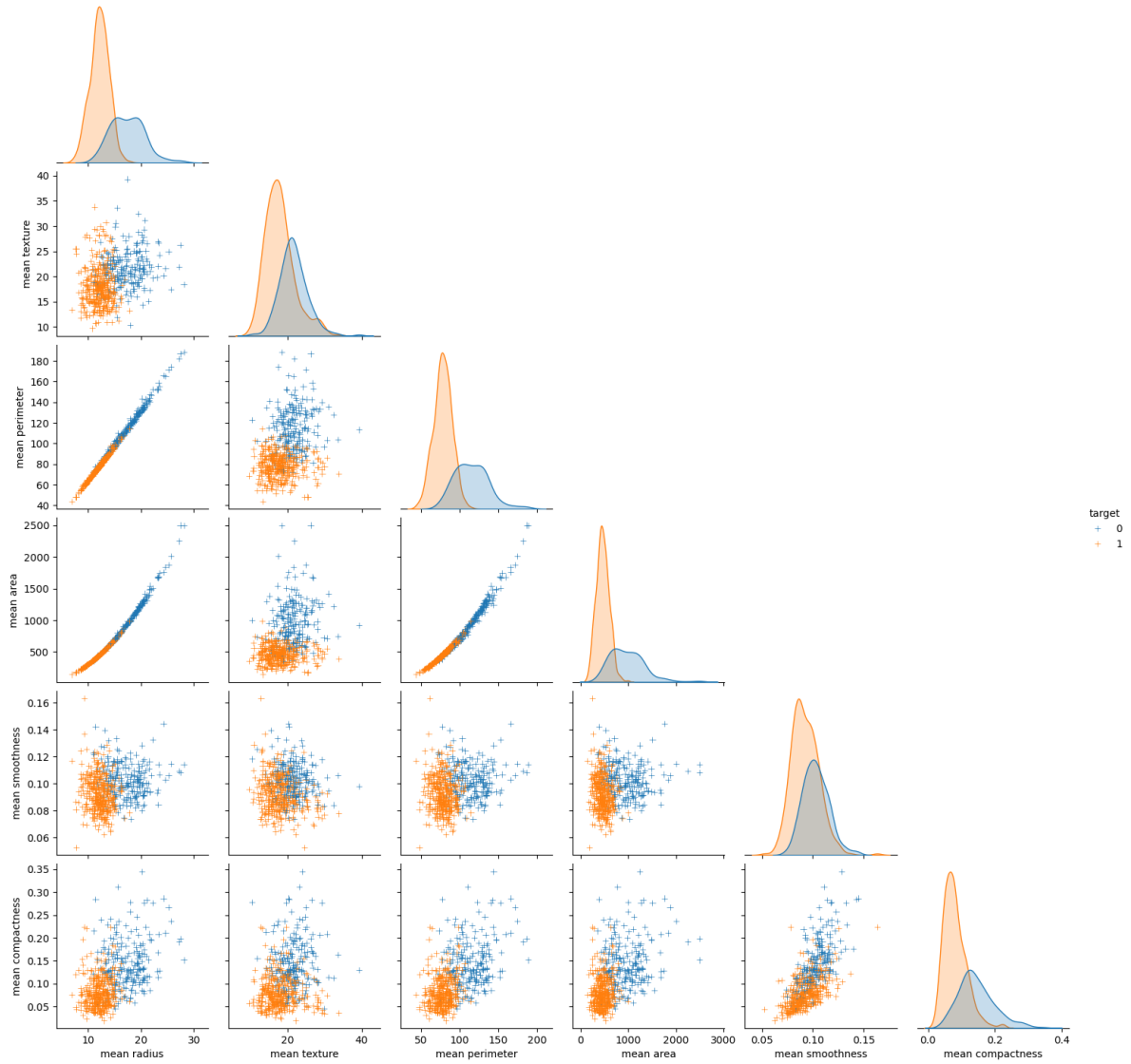
```
data.columns
```

```
Index(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error', 'fractal dimension error',
       'worst radius', 'worst texture', 'worst perimeter', 'worst area',
       'worst smoothness', 'worst compactness', 'worst concavity',
       'worst concave points', 'worst symmetry', 'worst fractal dimension'],
      dtype='object')
```

3.4.1 Pair plot

A pair plot in Seaborn is a grid of scatterplots and histograms showing the pairwise relationships and distributions of multiple variables in a dataset. It can be used to visualize patterns and correlations between variables, identify potential outliers or clusters, and guide further analysis.

```
sns.pairplot(data=all_data[['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness','target']], hue='target',corner=True, markers
```



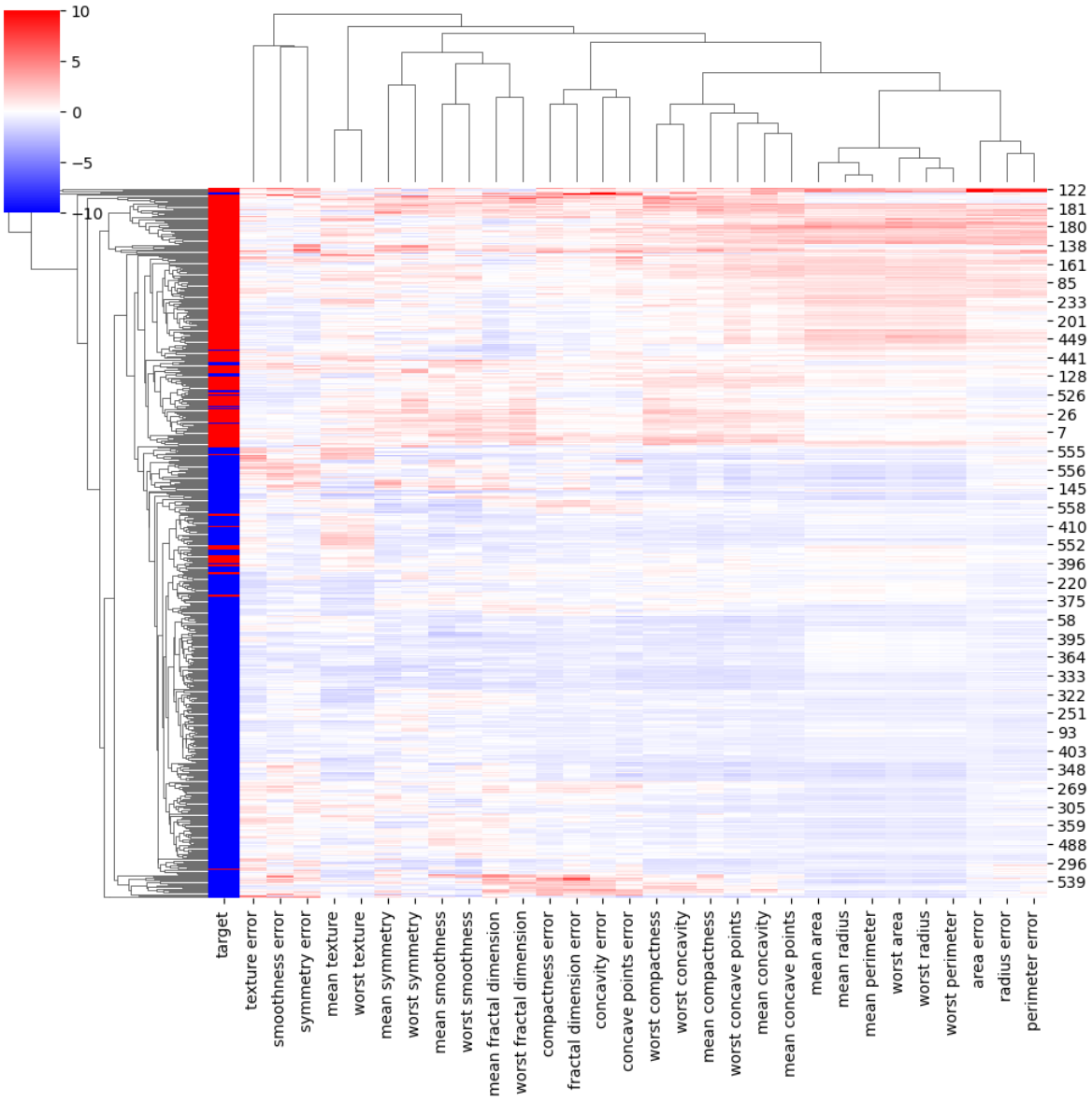
3.4.2 Clustermap

A clustermap in Seaborn is a heatmap-like plot that arranges rows and columns of a dataset based on their similarity and creates dendrograms to show hierarchical clustering. It can be used to explore patterns and relationships in high-dimensional data, identify groups or clusters of similar observations, and guide feature selection or dimensionality reduction.

```
# first we standardize the data by computing the z score
z = data.apply(lambda x:(x-x.mean())/x.std(), axis=0)
```

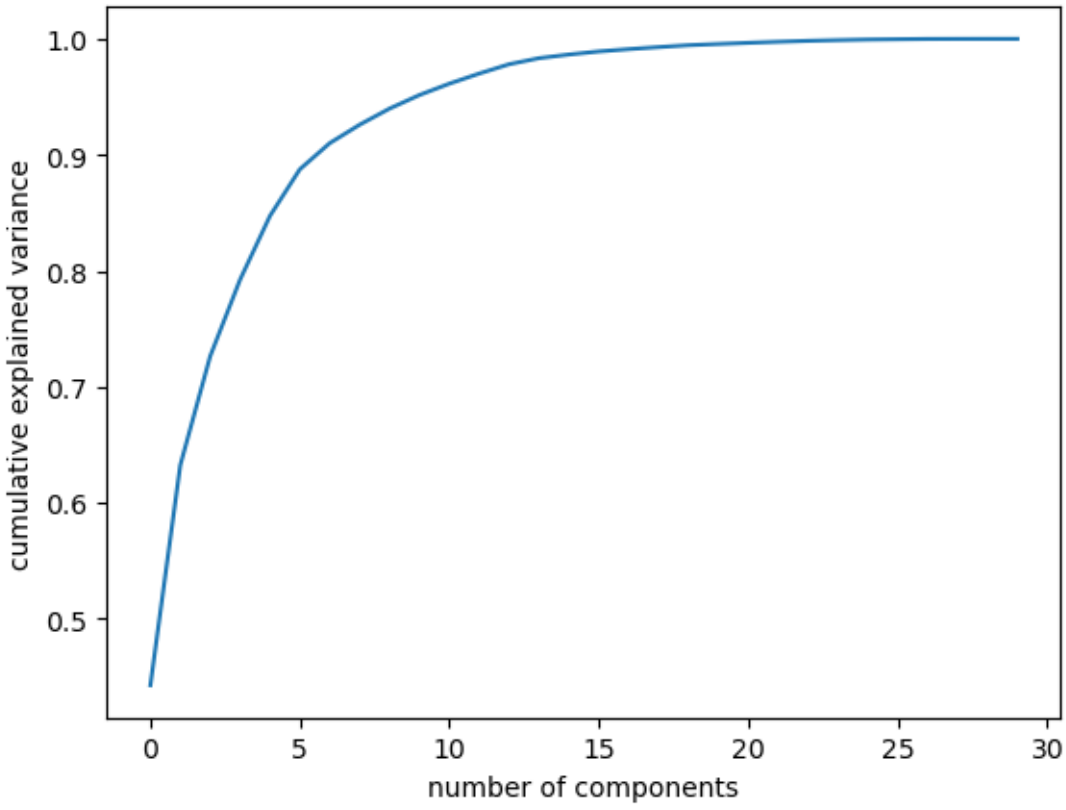
```
# Ploting the clustermap
lut = dict(zip(all_data['target'].unique(), "rbg"))
row_colors = all_data['target'].map(lut)
# sns.clustermap(z, row_colors=row_colors)

sns.clustermap(z, row_colors=row_colors, vmax=10, vmin=-10, cmap='bwr')

# another way - instead of manualy calculating the z score you can set the z_score arg to
# sns.clustermap(data, row_colors=row_colors, z_score=1)
```

```
/opt/anaconda3/lib/python3.11/site-packages/seaborn/matrix.py:560: UserWarning: Clustering la
  warnings.warn(msg)
/opt/anaconda3/lib/python3.11/site-packages/seaborn/matrix.py:560: UserWarning: Clustering la
  warnings.warn(msg)
```



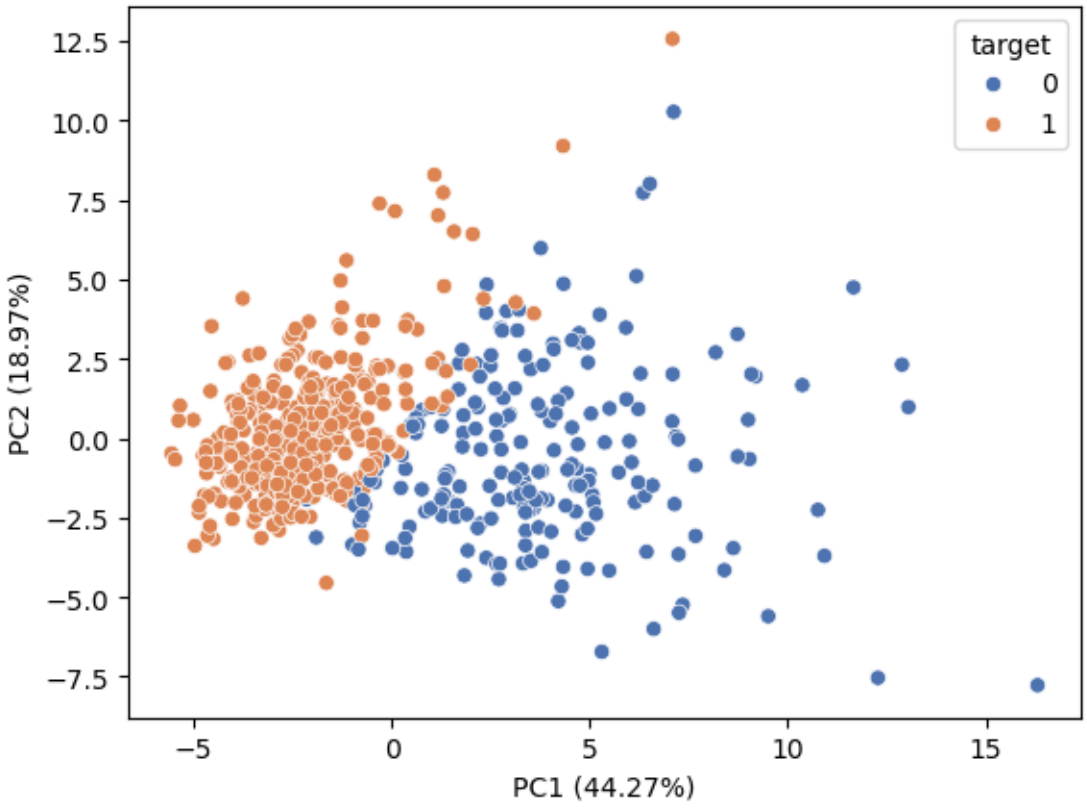
3.4.3 Ploting PCA

```
pca = PCA().fit(z)
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```




```
X = z
pca = PCA().fit(X)
X_trans = pca.transform(X)
colors = all_data['target']

sns.scatterplot(x=X_trans[:,0], y=X_trans[:,1], hue=colors, palette='deep')

plt.xlabel ('PC1 ({:.2f}%)'.format(100*pca.explained_variance_ratio_[0]))
plt.ylabel ('PC2 ({:.2f}%)'.format(100*pca.explained_variance_ratio_[1]))
```

```
Text(0, 0.5, 'PC2 (18.97%)')
```


target
0
1
12.5
10.0
7.5
5.0
2.5
0.0
−2.5
−5.0
−7.5
PC2 (18.97%)
−5
0
5
10
15
PC1 (44.27%)

4 Tutorial 4

4.1 Topics

1. K-means clustering
2. Hierarchical clustering

4.2 K-Means Clustering

The k-means clustering method is an unsupervised machine learning technique used to identify clusters of data objects in a dataset. There are many different types of clustering methods, but k-means is one of the oldest and most approachable. These traits make implementing k-means clustering in Python reasonably straightforward, even for novice programmers and data scientists.

The k-means algorithm searches for a pre-determined number of clusters within an unlabeled multidimensional dataset. It accomplishes this using a simple conception of what the optimal clustering looks like:

The "cluster center" is the arithmetic mean of all the points belonging to the cluster. Each point is closer to its own cluster center than to other cluster centers.

https://jakevdp.github.io/PythonDataScienceHandbook/05.11-k-means.html

https://realpython.com/k-means-clustering-python/#:~:text=The%20k%2Dmeans%20clustering%20method,da

4.3 Example

```
# import packages
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
```

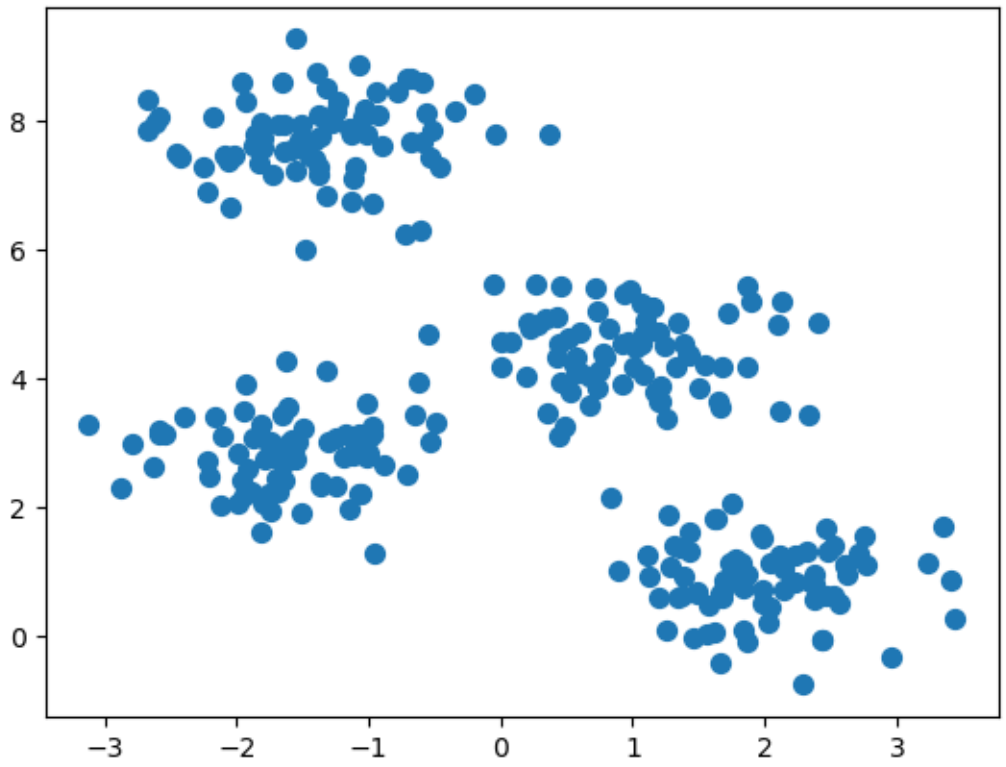
```
from sklearn.metrics import silhouette_score
from scipy.cluster.hierarchy import fcluster
```

4.3.1 Starting from the end

Before getting to the algorithm itself, let's see what K-Means actually does

4.3.2 Create random data grouped in four "blobs"

```
X, y_true = make_blobs(n_samples=300,
                       centers=4,
                       cluster_std=0.60,
                       random_state=0)
plt.scatter(X[:, 0],
            X[:, 1],
            s=50);
```



4.3.3 Calculate K-Means

```
kmeans = KMeans(n_clusters=4)
kmeans.fit(X)
clusters_kmeans = kmeans.predict(X)
clusters_kmeans
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_kmeans.py:870: FutureWarning: Th
  warnings.warn(
```

```
array([2, 1, 0, 1, 2, 2, 3, 0, 1, 1, 3, 1, 0, 1, 2, 0, 0, 2, 3, 3, 2, 2,
       0, 3, 3, 0, 2, 0, 3, 0, 1, 1, 0, 1, 1, 1, 1, 1, 3, 2, 0, 3, 0, 0,
       3, 3, 1, 3, 1, 2, 3, 2, 1, 2, 2, 3, 1, 3, 1, 2, 1, 0, 1, 3, 3, 3,
       1, 2, 1, 3, 0, 3, 1, 3, 3, 1, 3, 0, 2, 1, 2, 0, 2, 2, 1, 0, 2, 0,
       1, 1, 0, 2, 1, 3, 3, 0, 2, 2, 0, 3, 1, 2, 1, 2, 0, 2, 2, 0, 1, 0,
       3, 3, 2, 1, 2, 0, 1, 2, 2, 0, 3, 2, 3, 2, 2, 2, 2, 3, 2, 3, 1, 3,
       3, 2, 1, 3, 3, 1, 0, 1, 1, 3, 0, 3, 0, 3, 1, 0, 1, 1, 1, 0, 1, 0,
       2, 3, 1, 3, 2, 0, 1, 0, 0, 2, 0, 3, 3, 0, 2, 0, 0, 1, 2, 0, 3, 1,
       2, 2, 0, 3, 2, 0, 3, 3, 0, 0, 0, 0, 2, 1, 0, 3, 0, 0, 3, 3, 3, 0,
       3, 1, 0, 3, 2, 3, 0, 1, 3, 1, 0, 1, 0, 3, 0, 0, 1, 3, 3, 2, 2, 0,
       1, 2, 2, 3, 2, 3, 0, 1, 1, 0, 0, 1, 0, 2, 3, 0, 2, 3, 1, 3, 2, 0,
       2, 1, 1, 1, 1, 3, 3, 1, 0, 3, 2, 0, 3, 3, 3, 2, 2, 1, 0, 0, 3, 2,
       1, 3, 0, 1, 0, 2, 2, 3, 3, 0, 2, 2, 2, 0, 1, 1, 2, 2, 0, 2, 2, 2,
       1, 3, 1, 0, 2, 2, 1, 1, 1, 2, 2, 0, 1, 3], dtype=int32)
```

```
len(clusters_kmeans)
```

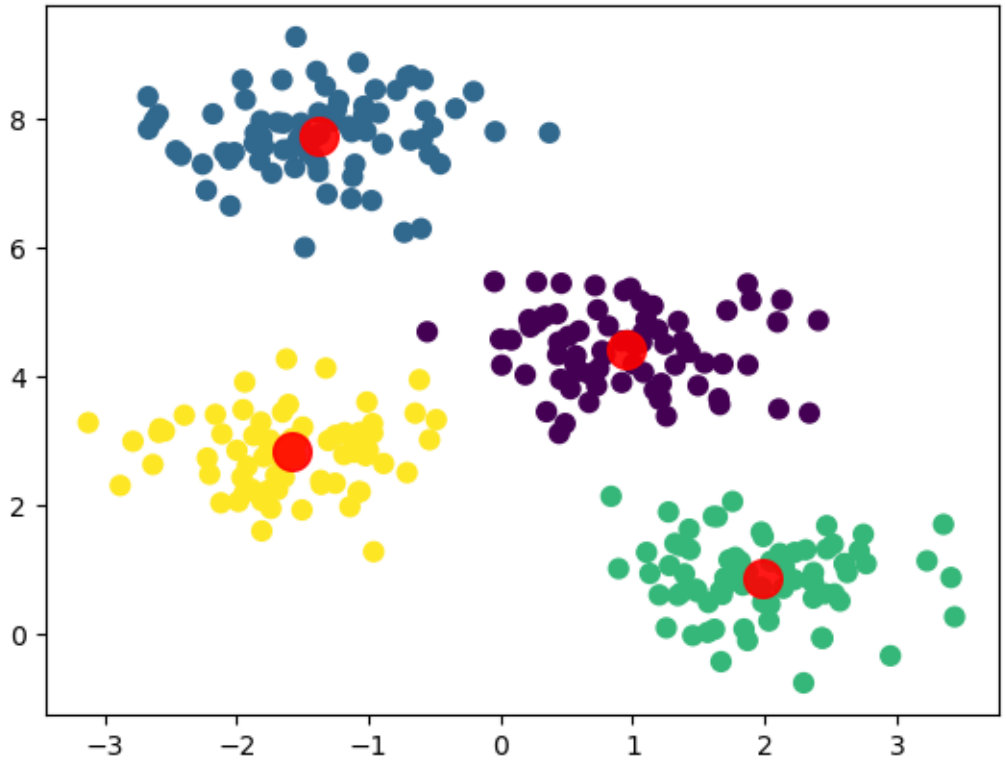
```
300
```

4.3.4 Plot results

```
# blobs
plt.scatter(X[:, 0],
            X[:, 1],
            c=clusters_kmeans,
            s=50,
            cmap='viridis')

# center of blob
```

```
centers = kmeans.cluster_centers_
plt.scatter(centers[:, 0],
            centers[:, 1],
            c='red',
            s=200,
            alpha=0.9)
```

```
<matplotlib.collections.PathCollection at 0x125c68b90>
```



4.3.5 How does the algorithm work?

K-Means uses the Expectation-Maximization Algorithm

How does this work?

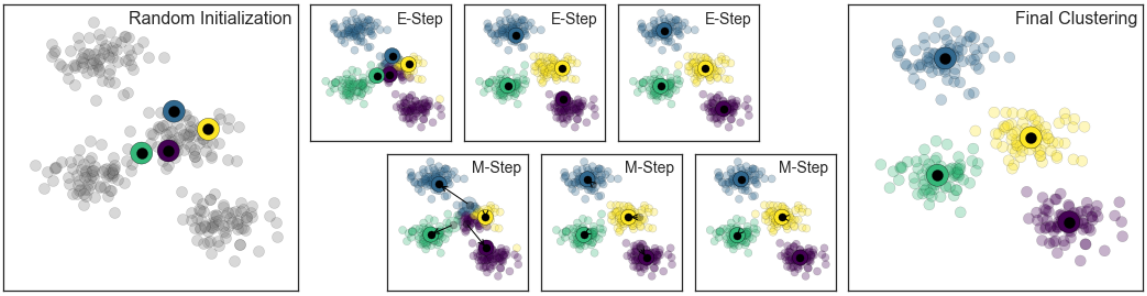
1. Guess some cluster centers
2. Repeat until converged

- E-Step: assign points to the nearest cluster center
- M-Step: set the cluster centers to the mean

The "E-step" or "Expectation step" involves updating our expectation of which cluster each point belongs to.

The "M-step" or "Maximization step" involves maximizing some fitness function that defines the location of the cluster centers. In this case, that maximization is accomplished by taking a simple mean of the data in each cluster.

```
%matplotlib inline
from IPython.display import Image
Image('image_1.png')
```



4.3.6 Visualisation

Here is a cool video for visualising the prosses of K-means clustering.

4.3.7 Choosing the right number of clusters

We'll look at two methods for choosing the correct number of clusters.

The elbow method and the silhouette method. We can use these methods together. No need to pick just one.

The quality of the cluster assignments is determined by computing the sum of the squared error (SSE) after the centroids converge.

SSE is defined as the sum of the squared distance between centroid and each member of the cluster.

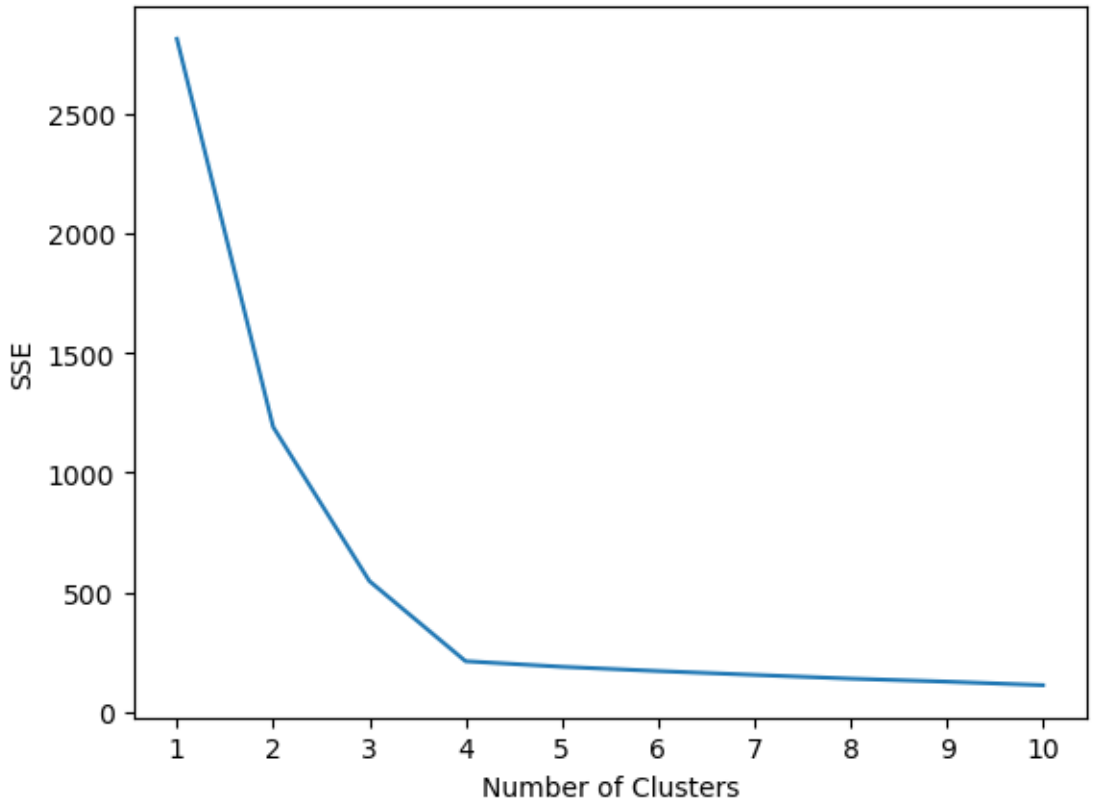
4.3.8 The elbow method

To perform the elbow method, run several k-means, increment k with each iteration, and record the SSE.

```
# loop over different numbers of clusters
kmeans_kwargs = {"init": "random",
                 "n_init": 10,
                 "max_iter": 300,
                 "random_state": 42,}

# A list holds the SSE values for each k
sse = []
for k in range(1, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    error = kmeans.inertia_
    sse.append(kmeans.inertia_)
    print(f"With {k} clusters the SSE was {error}")
```

```
With 1 clusters the SSE was 2812.137595303235
With 2 clusters the SSE was 1190.7823593643443
With 3 clusters the SSE was 546.8911504626299
With 4 clusters the SSE was 212.00599621083478
With 5 clusters the SSE was 188.77323556773723
With 6 clusters the SSE was 170.94840955438684
With 7 clusters the SSE was 154.8848618157605
With 8 clusters the SSE was 139.20927769246356
With 9 clusters the SSE was 126.56204002887225
With 10 clusters the SSE was 111.8591054874254
```

```
# plot the data
plt.plot(range(1, 11), sse)
plt.xticks(range(1, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("SSE")
plt.show()
```



Determining the elbow point in the SSE curve isn't always straightforward. If you're having trouble choosing the elbow point of the curve, then you could use a Python package, kneed, to identify the elbow point programmatically.

4.3.9 The silhouette coefficient

The silhouette coefficient is a measure of cluster cohesion and separation. It quantifies how well a data point fits into its assigned cluster based on two factors: 1. How close the data point is to other points in the cluster 2. How far away the data point is from points in other clusters

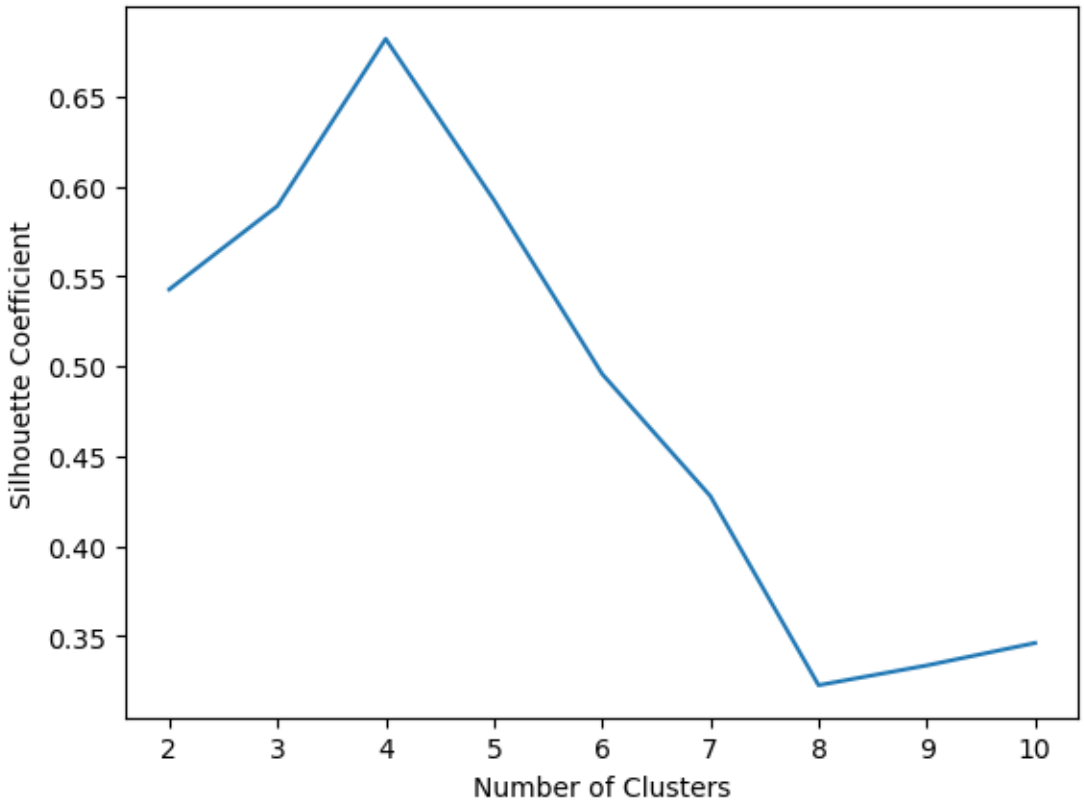
Silhouette coefficient values range between -1 and 1. Larger numbers indicate that samples are closer to their clusters than they are to other clusters.

```
# A list holds the silhouette coefficients for each k
silhouette_coefficients = []
```

```python
# Notice you start at 2 clusters for silhouette coefficient
for k in range(2, 11):
    kmeans = KMeans(n_clusters=k, **kmeans_kwargs)
    kmeans.fit(X)
    score = silhouette_score(X, kmeans.labels_)
    silhouette_coefficients.append(score)
    print(f"With {k} clusters the score was {score}")
```

```
With 2 clusters the score was 0.5426422297358303
With 3 clusters the score was 0.5890390393551768
With 4 clusters the score was 0.6819938690643478
With 5 clusters the score was 0.5923875148758644
With 6 clusters the score was 0.49563409602576686
With 7 clusters the score was 0.4277257665723784
With 8 clusters the score was 0.32252064195197455
With 9 clusters the score was 0.3335178002757919
With 10 clusters the score was 0.34598612018426733
```

```python
plt.plot(range(2, 11), silhouette_coefficients)
plt.xticks(range(2, 11))
plt.xlabel("Number of Clusters")
plt.ylabel("Silhouette Coefficient")
plt.show()
```


Silhouette Coefficient
0.65
0.60
0.55
0.50
0.45
0.40
0.35
2
3
4
5
6
7
8
9
10
Number of Clusters

5 Hierarchical Clustering

Hierarchical clustering is a type of unsupervised machine learning algorithm used to cluster unlabeled data points. Hierarchical clustering groups together data points with similar characteristics.

There are two types of hierarchical clustering: agglomerative and divisive. * Agglomerative: data points are clustered using a bottom-up approach starting with individual data points * Divisive: all the data points are treated as one big cluster and the clustering process involves dividing the one big cluster into several small clusters.

Today, we'll focus on agglomerative clustering.

https://scikit-learn.org/stable/auto_examples/cluster/plot_agglomerative_dendrogram.html
https://stackabuse.com/hierarchical-clustering-with-python-and-scikit-learn
https://joernhees.de/blog/2015/08/26/scipy-hierarchical-clustering-and-dendrogram-tutorial/

5.0.1 Steps to perform agglomerative clustering

1. Treat each data point as its own cluster. The number of clusters at the start will be K, where K is the number of data points.
2. Reduce the data to $K-1$ clusters by joining the two closest clusters (data points).
3. Reduce the data to $K-2$ clusters by again joining the two closest clusters.
4. Repeat this process until one big cluster is formed.
5. Develop dendograms to analyze the clusters.

5.0.2 Visualisation

Here is a video for visualising the prosses of hierarchical clustering.

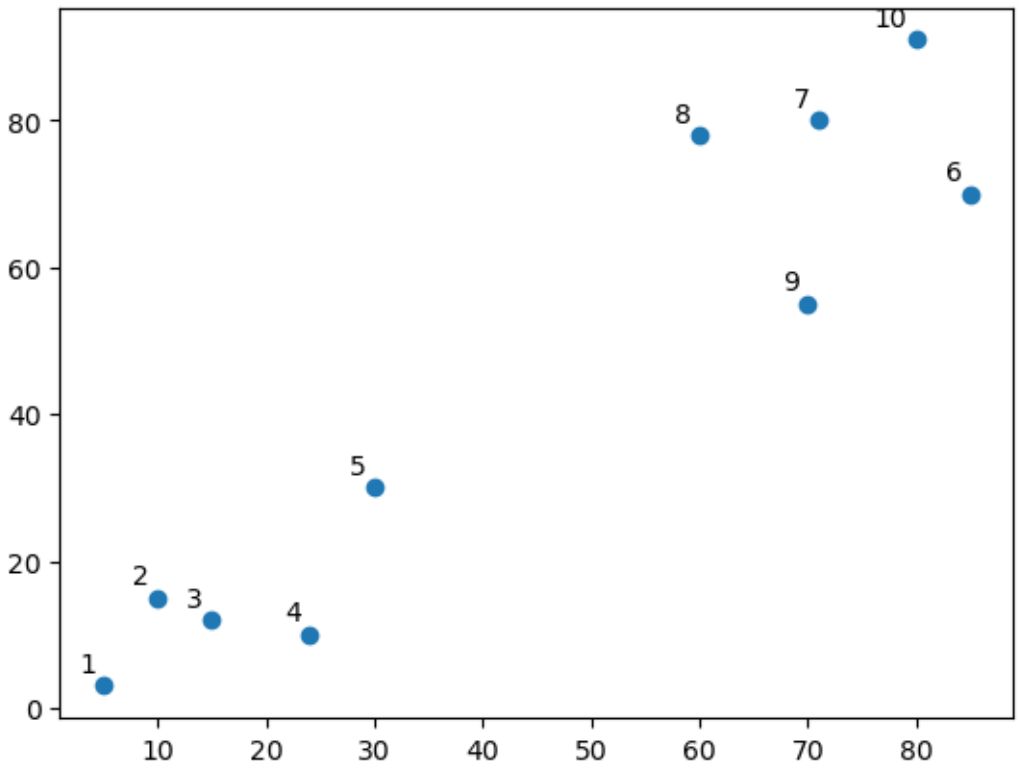
5.0.3 Worked Example

```
# import packages
import numpy as np
import pandas as pd
```

```
from matplotlib import pyplot as plt
import scipy.cluster.hierarchy as shc
from sklearn.cluster import AgglomerativeClustering
```

```
# make up data
points = np.array([[5,3],
                   [10,15],
                   [15,12],
                   [24,10],
                   [30,30],
                   [85,70],
                   [71,80],
                   [60,78],
                   [70,55],
                   [80,91],])
```

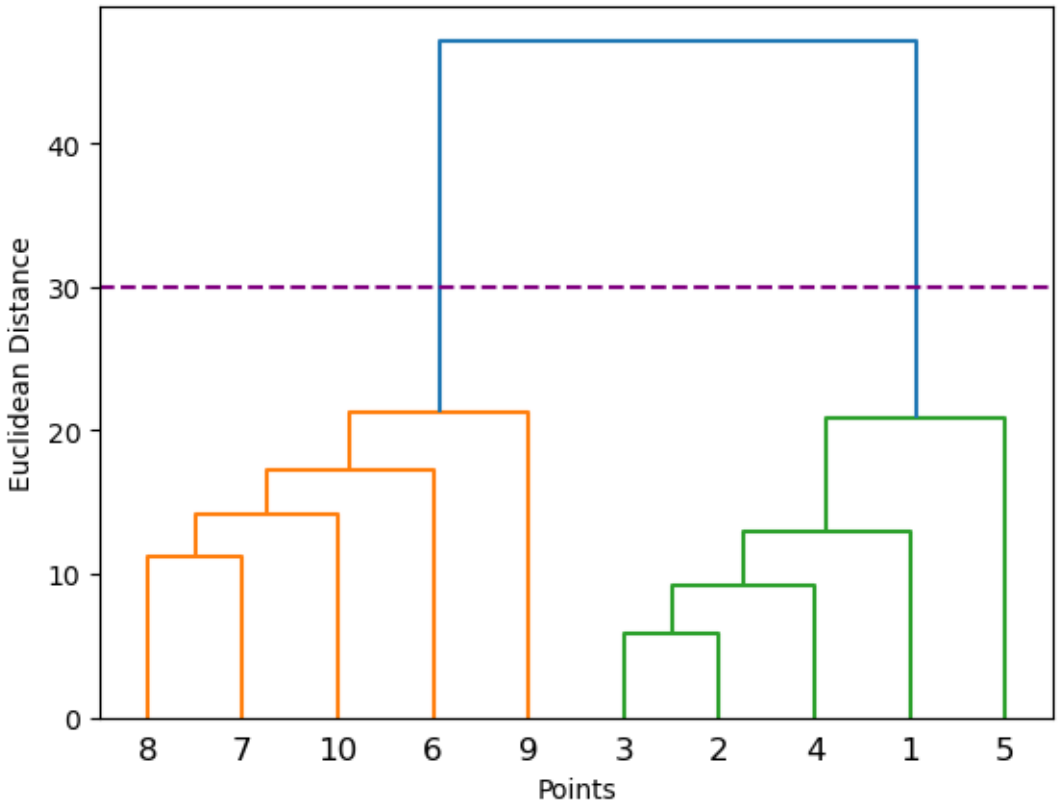
```
# plot the data
labels = range(1, 11)
plt.figure()
plt.scatter(points[:,0],
            points[:,1],
            label='True Position')

for label, x, y in zip(labels, points[:, 0], points[:, 1]):
    plt.annotate(label,
                 xy=(x, y),
                 xytext=(-3, 3),
                 textcoords='offset points',
                 ha='right',
                 va='bottom')
```



5.0.4 Do you see any "clusters"?

```
linked = shc.linkage(points, 'single')

labelList = range(1, 11)

fig = plt.figure()
shc.dendrogram(linked,
               orientation='top',
               labels=labelList,
               distance_sort='descending',
               show_leaf_counts=True)
plt.xlabel('Points')
plt.ylabel('Euclidean Distance')
plt.axhline(y= 30, c='purple', ls = '--')
```

```
<matplotlib.lines.Line2D at 0x127e6b710>
```



5.0.4.1 Where is the longest distance without a horizontal line?

```
# create actual clusters
cluster = AgglomerativeClustering(n_clusters=2,
                                  affinity='euclidean',
                                  linkage='ward')
cluster.fit_predict(points)
```

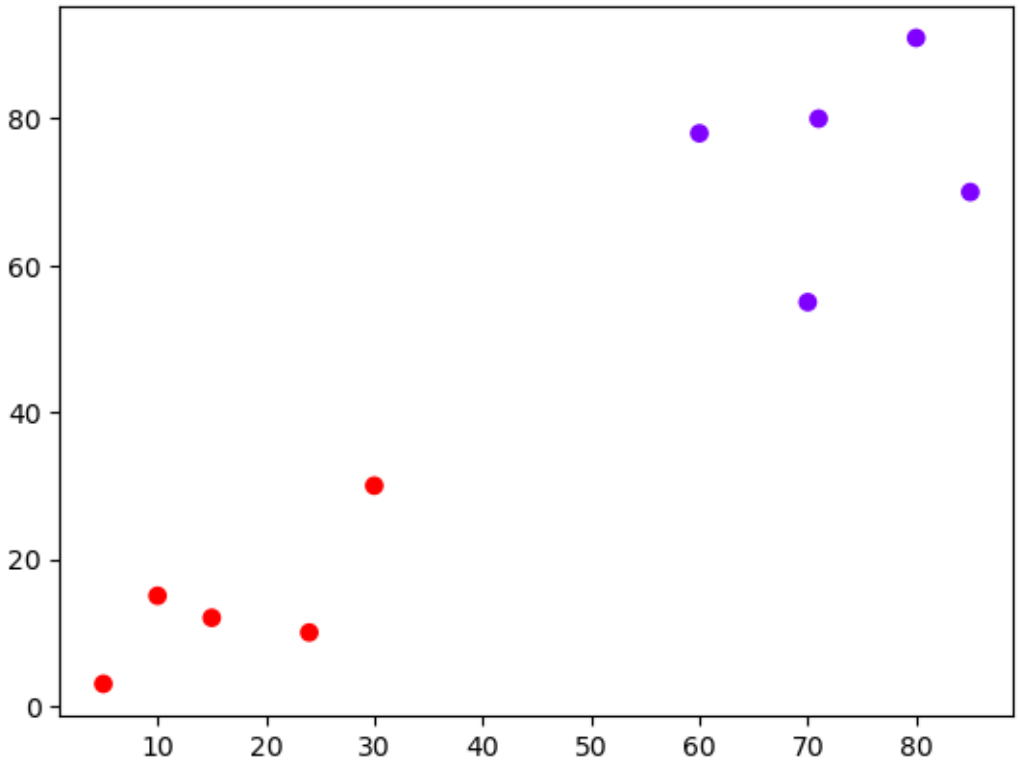
```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_agglomerative.py:983: FutureWarn
  warnings.warn(
```

```
array([1, 1, 1, 1, 1, 0, 0, 0, 0, 0])
```

5.0.4.2 The output array shows us which cluster each point is in

```
# plot the original data with the clusters
plt.scatter(points[:,0], points[:,1],
            c=cluster.labels_,
            cmap='rainbow')
```

```
<matplotlib.collections.PathCollection at 0x127dfef90>
```



5.0.5 Example 2 – Real Data

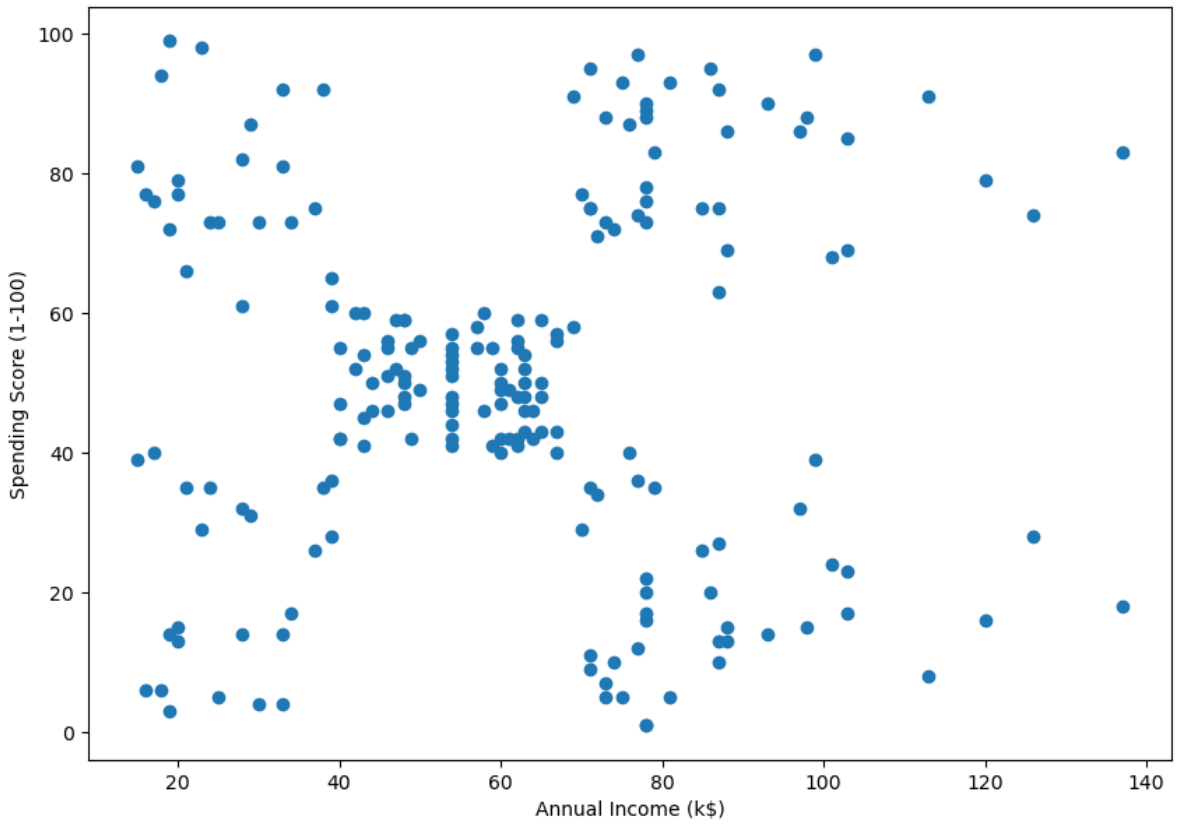
download .csv here

```
# import data
customer_data = pd.read_csv('shopping-data.csv')
customer_data.head()
```

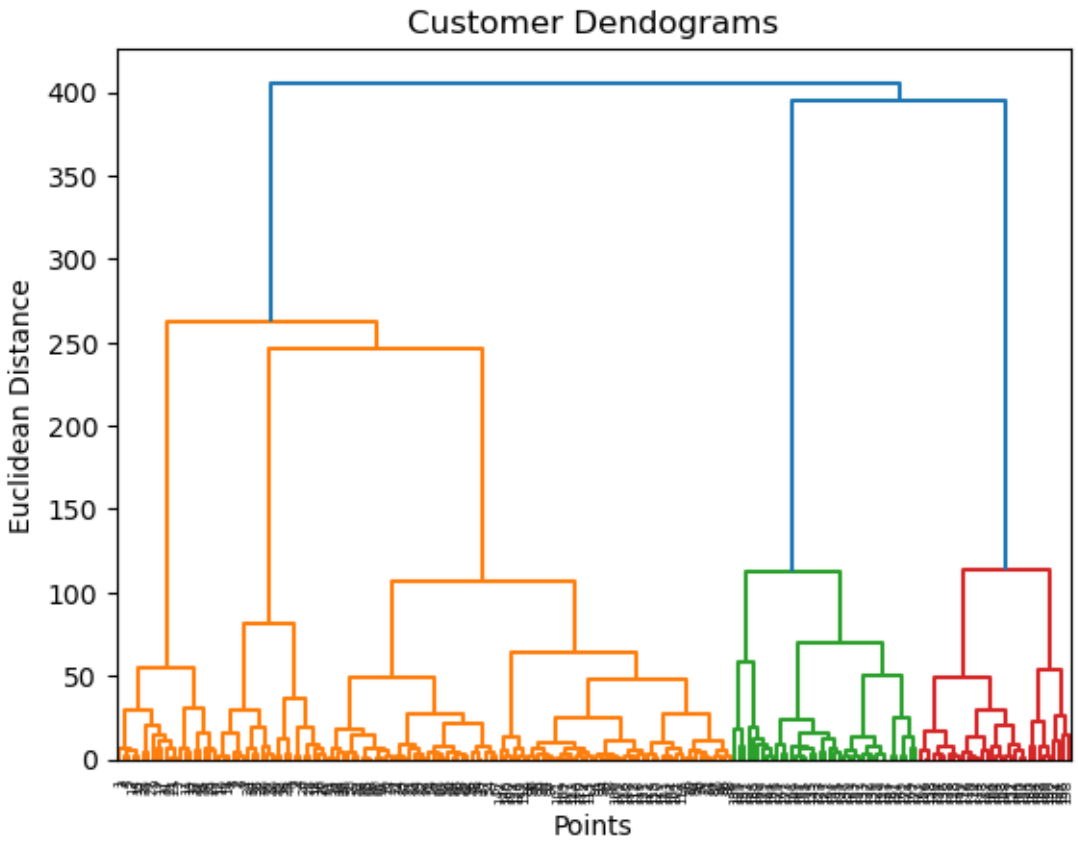
	CustomerID	Genre	Age	Annual Income (k$)	Spending Score (1-100)
0	1	Male	19	15	39
1	2	Male	21	15	81
2	3	Female	20	16	6
3	4	Female	23	16	77
4	5	Female	31	17	40

```
# isolate data from annual income and spending score columns
data = customer_data.iloc[:, 3:5].values
plt.figure(figsize=(10, 7))
plt.scatter(data[:,0], data[:,1])
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
```

```
Text(0, 0.5, 'Spending Score (1-100)')
```



```
# examine dendogram
plt.figure()
plt.title("Customer Dendograms")
plt.xlabel('Points')
plt.ylabel('Euclidean Distance')
dend = shc.dendrogram(shc.linkage(data, method='ward'))
```



5.0.6 Finding the right number of clusters

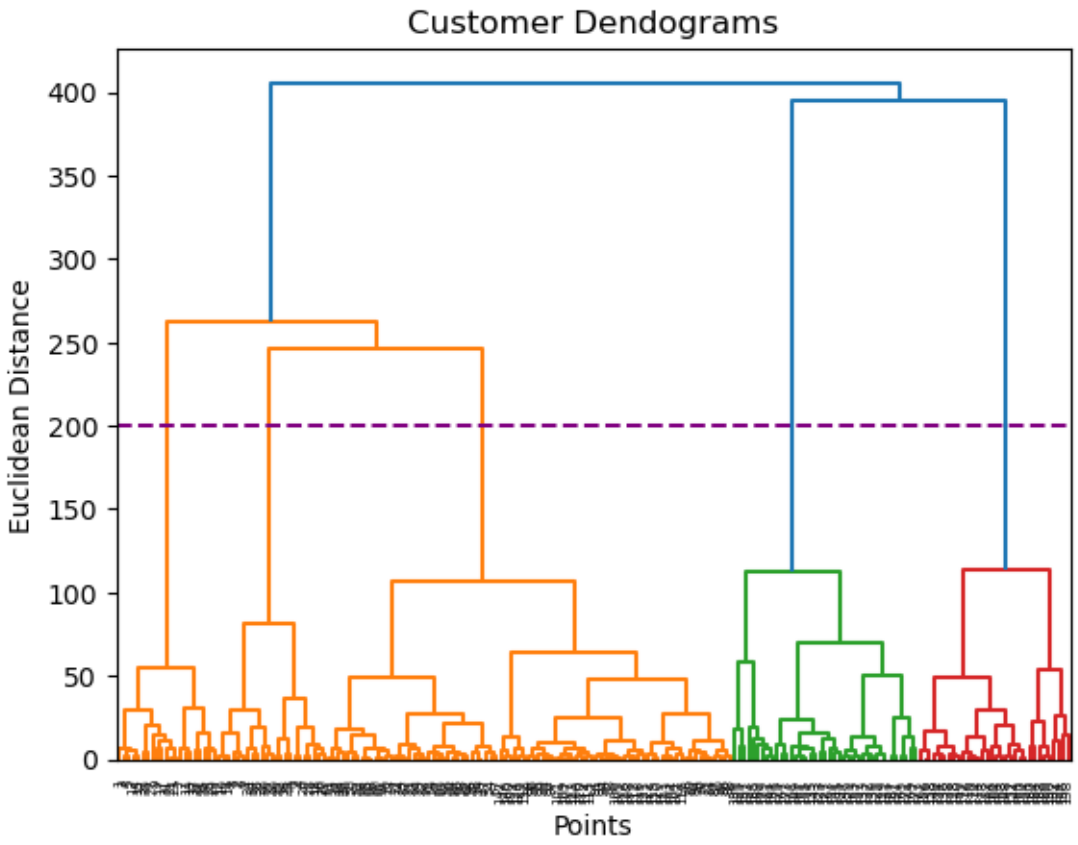
5.0.6.1 Method 1: visualization

Where is the longest distance without a horizontal line?

```
Z = shc.linkage(data, method='ward')
```

```
plt.figure()
plt.title("Customer Dendograms")
plt.xlabel('Points')
plt.ylabel('Euclidean Distance')
dend = shc.dendrogram(Z)
plt.axhline(y= 200, c='purple', ls = '--')
```

```
<matplotlib.lines.Line2D at 0x128518b50>
```



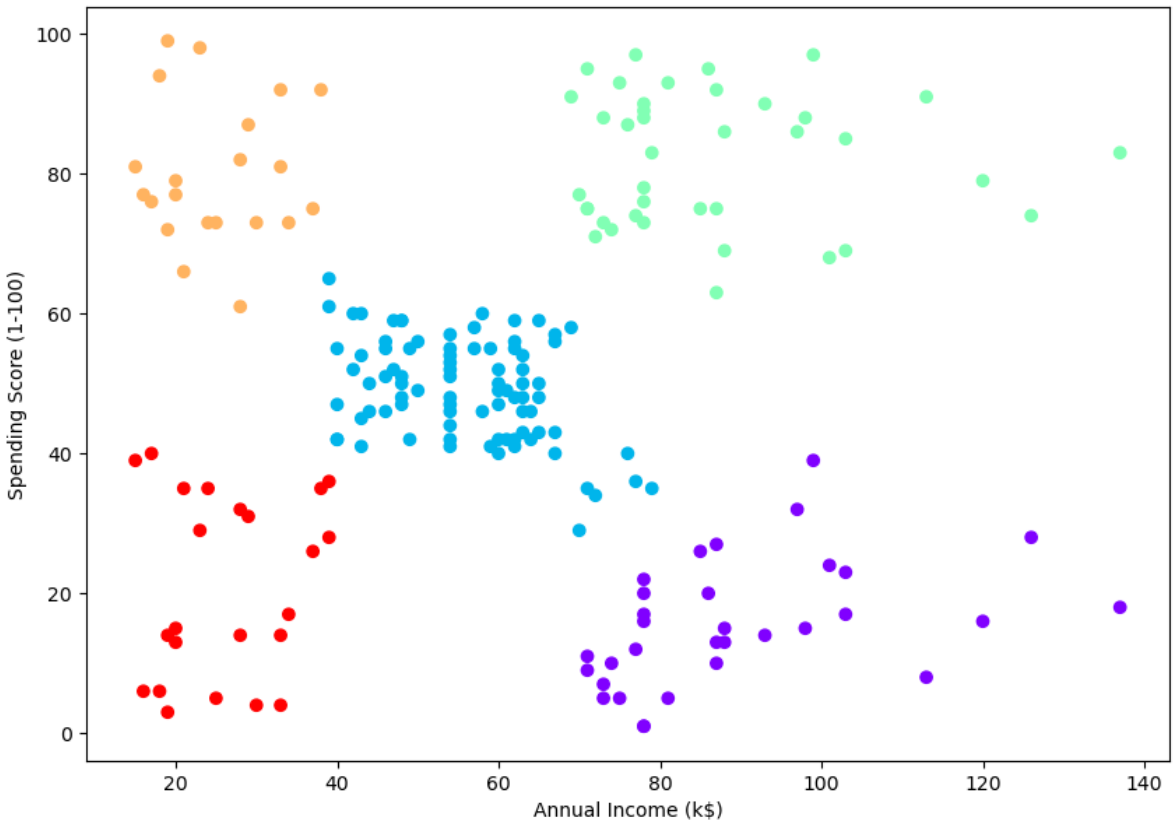

```
# create clusters
cluster = AgglomerativeClustering(n_clusters=5, affinity='euclidean', linkage='ward')
cluster.fit_predict(data)
```

```
/opt/anaconda3/lib/python3.11/site-packages/sklearn/cluster/_agglomerative.py:983: FutureWarn
  warnings.warn(
```

```
array([4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3,
       4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 3, 4, 1,
       4, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 2, 1, 2, 0, 2, 0, 2,
       1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 1, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2, 0, 2,
       0, 2])
```

```
# look at the clusters
plt.figure(figsize=(10, 7))
plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
plt.xlabel('Annual Income (k$)')
plt.ylabel('Spending Score (1-100)')
```

```
Text(0, 0.5, 'Spending Score (1-100)')
```

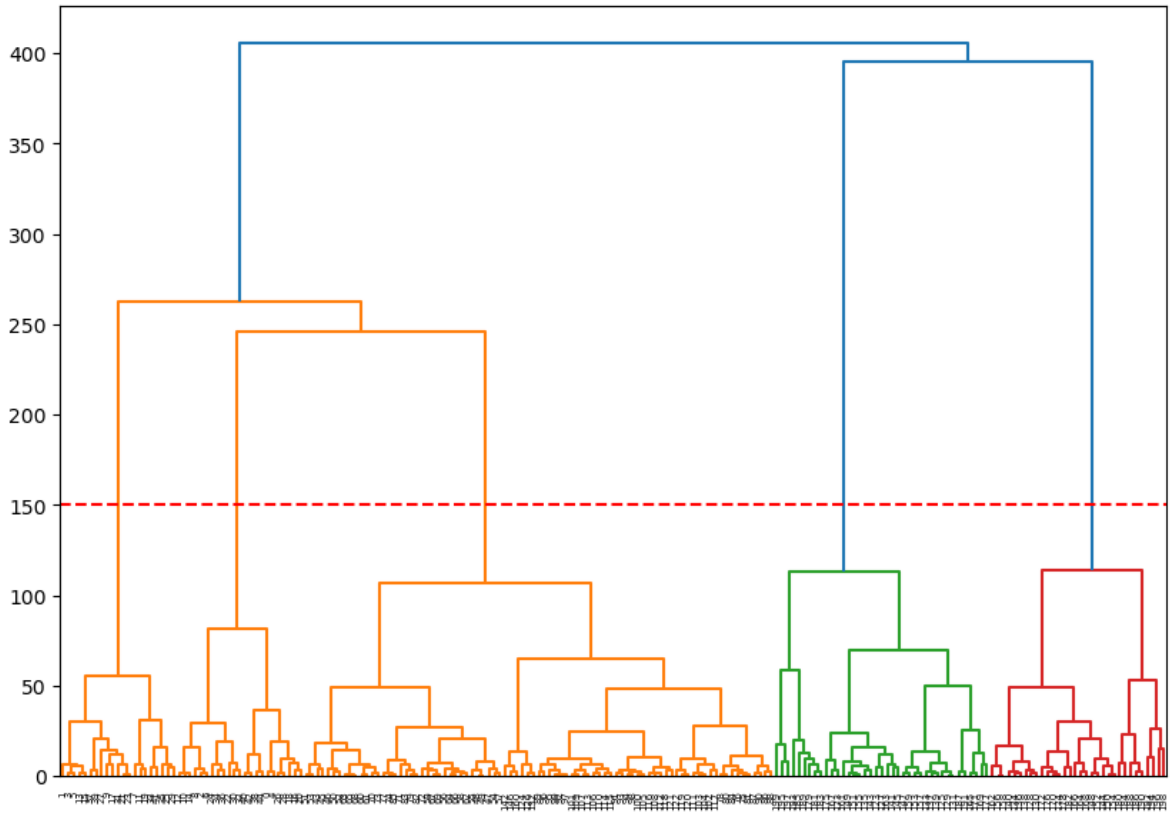


5.0.6.2 Method 2: threshold

we don't need to count the number of clusters, it's enough to give the threshold value.

```
# Set the threshold value where you want to cut the dendrogram
threshold_value = 150  # Adjust this value as needed

# Assuming 'data' is your dataset
Z = shc.linkage(data, method='ward')

# Plot the dendrogram
plt.figure(figsize=(10, 7))
dend = shc.dendrogram(Z)
plt.axhline(y=threshold_value, color='r', linestyle='--')
plt.show()

# Get the cluster labels
```

```
clusters = fcluster(Z, threshold_value, criterion='distance')

# Print the number of clusters
num_clusters = len(set(clusters))
print(f'The number of clusters is: {num_clusters}')
```

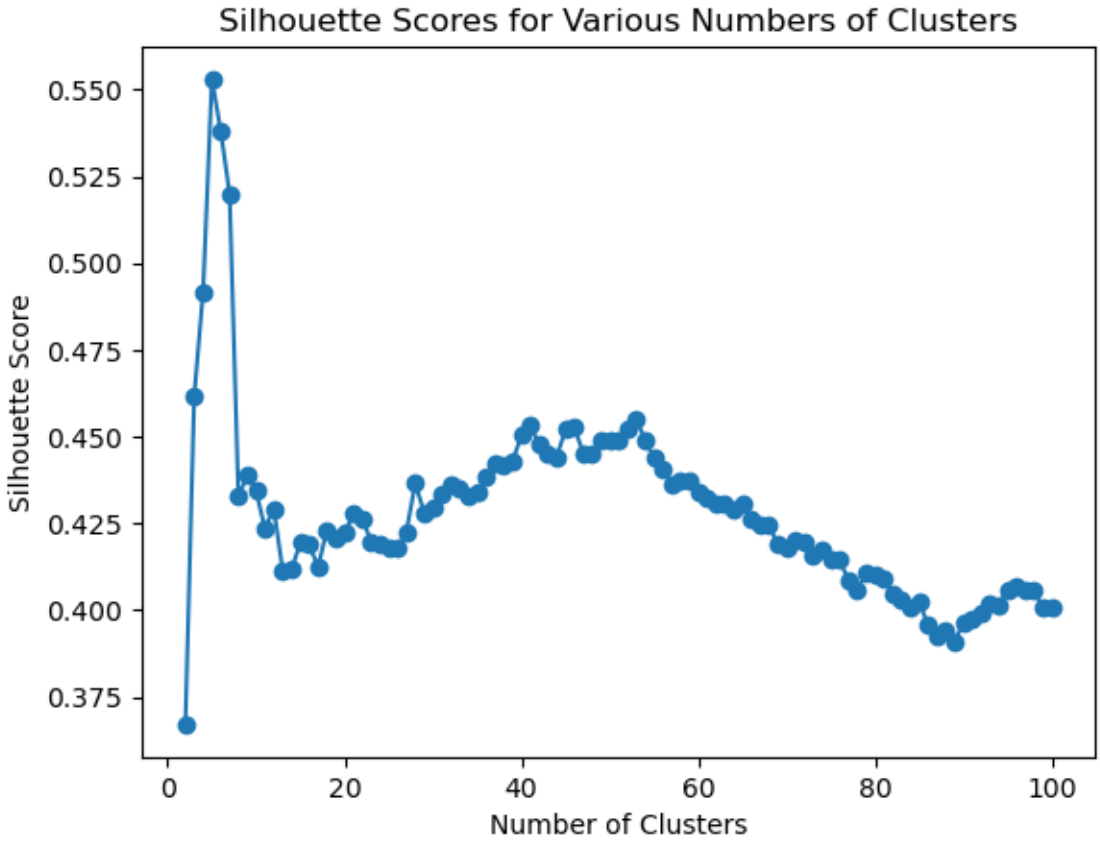



```
The number of clusters is: 5
```

5.0.6.3 Method 3: silhouette

```
# Assuming 'data' is your dataset and 'Z' is the linkage matrix
range_n_clusters = list(range(2, 101))
silhouette_scores = []

for n_clusters in range_n_clusters:
    cluster_labels = fcluster(Z, n_clusters, criterion='maxclust')
```

```
    silhouette_avg = silhouette_score(data, cluster_labels)
    silhouette_scores.append(silhouette_avg)
    # print(f"For n_clusters = {n_clusters}, the silhouette score is: {silhouette_avg}")

# Plot the silhouette scores
fig, ax = plt.subplots()
ax.plot(range_n_clusters, silhouette_scores, marker='o')
ax.set_title('Silhouette Scores for Various Numbers of Clusters')
ax.set_xlabel('Number of Clusters')
ax.set_ylabel('Silhouette Score')
plt.show()

print(f'Max silhouette score at {silhouette_scores.index(max(silhouette_scores))+2} cluste
```




```
Max silhouette score at 5 clusters
```

6 Play Time

Use the dataset `cluster_play_data.csv` to play with the clustering method we learned. download .csv here
First step hint - try to find out on what columns to preform the clustering.

```
# import data

# # uncomment below:
# play = pd.read_csv('cluster_play_data.csv')
# play.head()
```

```
# # uncomment below:
# sns.pairplot(play)
```

```
# # uncomment below:
# # examine dendogram
# # isolate data from annual income and spending score columns
# # data = customer_data.iloc[:, 3:5].values
# data = play[['A','C']].values
# plt.figure()
# plt.title("Customer Dendograms")
# plt.xlabel('Points')
# plt.ylabel('Euclidean Distance')
# dend = shc.dendrogram(shc.linkage(data, method='ward'))
```

```
# # uncomment below:
# # create clusters
# cluster = AgglomerativeClustering(n_clusters=6, affinity='euclidean', linkage='ward')
# cluster.fit_predict(data)

# # look at the clusters
# plt.figure(figsize=(10, 7))
# plt.scatter(data[:,0], data[:,1], c=cluster.labels_, cmap='rainbow')
# plt.xlabel('Annual Income (k$)')
```

```
# plt.ylabel('Spending Score (1-100)')
```

7 Tutorial 5

7.1 Topics

- Standard error vs. Standard deviation
- Boostrapping
- Confidence Intervals

7.2 Standard error vs. standard deviation

```
import numpy as np
import seaborn as sns

np.random.seed(42)
sns.set_theme(style="whitegrid")
```

Let's define a "population" of size 1,000,000. Each member of the population will be a number from the standard normal distribution. We will be using the numpy function `randn`

```
population = np.random.randn(1000000)
```

```
population
```

```
array([ 0.49671415, -0.1382643 ,  0.64768854, ..., -0.11297975,
        1.46914237,  0.47643025])
```

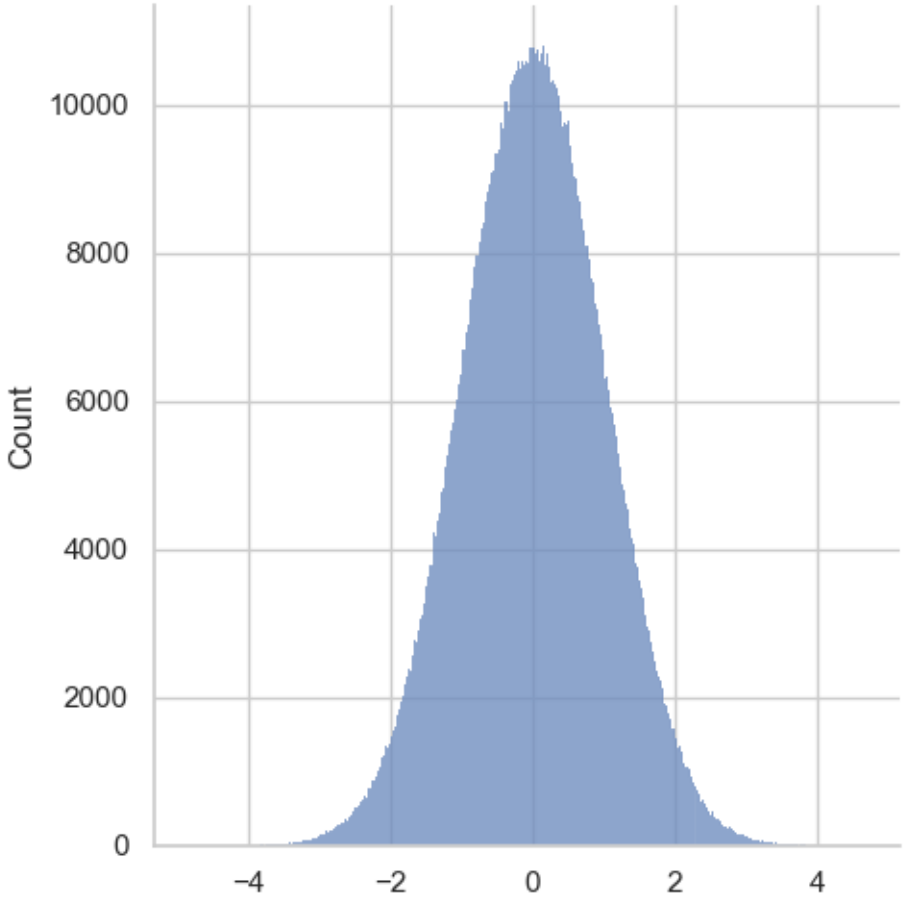
What's the population's mean and standard deviation?

```
mean_value = population.mean()
std = population.std()
print(f"The mean value is {mean_value} and the standard deviation is {std}")
```

```
The mean value is -0.0015997564542563718 and the standard deviation is 1.000188036804731
```

What does the distribution of the population look like?

```
sns.displot(population)
```



If this were the real world, sampling 1,000,000 people would be difficult. Let's say that our budget allowed us to sample only 100 people at a time. How can we simulate that?

```
sample_size = 100
sample_100 = np.random.choice(population, sample_size)
```

What's the sample's mean and standard deviation?

```
mean_100 = sample_100.mean()
std_100 = sample_100.std()
print(f"The mean value is {mean_100}\nThe standard deviation is {std_100}")
```

```
The mean value is -0.1753180501180135
The standard deviation is 1.1706030198688049
```

Notice that our sample's mean value is pretty different from that of the population. What if we take a bigger sample size?

```
sample_size = 1000
sample_1000 = np.random.choice(population, sample_size)
mean_1000 = sample_1000.mean()
std_1000 = sample_1000.std()
print(f"The mean value is {mean_1000}\nThe standard deviation is {std_1000}")
```

```
The mean value is -0.008352574043024205
The standard deviation is 1.003117759041072
```

Or smaller...

```
sample_size = 5
sample_5 = np.random.choice(population, sample_size)
mean_5 = sample_5.mean()
std_5 = sample_5.std()
print(f"The mean value is {mean_5}\nThe standard deviation is {std_5}")
```

```
The mean value is 0.03172109602313291
The standard deviation is 0.758566765992809
```

The standard error of the mean (SEM) measures the precision of the estimate of the sample mean. We can use the formula:

$\text{SEM} = \frac{\sigma}{\sqrt{n}}$

- σ = the standard deviation of the sample
- n = the sample size

Now we can compare our standard errors using the different sample sizes.

```python
se_5 = std_5/np.sqrt(5)
se_100 = std_100/np.sqrt(100)
se_1000 = std_1000/np.sqrt(1000)

print(f"The SEM when the sample size was 5 was {se_5}")
print(f"The SEM when the sample size was 100 was {se_100}")
print(f"The SEM when the sample size was 1000 was {se_1000}")
```

```
The SEM when the sample size was 5 was 0.3392413708464193
The SEM when the sample size was 100 was 0.11706030198688049
The SEM when the sample size was 1000 was 0.03172136879933749
```

8 Bootstrapping

Bootstrapping is a statistical method that involves repeatedly resampling a dataset with replacement to estimate the distribution of a statistic. This technique allows for the assessment of the accuracy and variability of sample estimates, such as the mean or standard deviation, without making strict assumptions about the underlying population distribution. By generating many resampled datasets (called bootstrap samples) and calculating the statistic of interest for each sample, bootstrapping provides an empirical approximation of the sampling distribution. This method is particularly useful when dealing with small samples or when traditional parametric assumptions cannot be applied.

Let's create a numpy array to represent heights of females at the Faculta

```
np.random.seed(42)
mean = 162.2
std = 5.5
sample_size = 1000
heights = np.random.normal(loc=mean,
                           scale=std,
                           size=sample_size)

# sample mean
mean_value = heights.mean()

# population standard deviation
std = heights.std()

print(f"The population's mean value is {mean_value}\nThe standard deviation is {std}")
```

```
The population's mean value is 162.30632630702277
The standard deviation is 5.382994142610448
```

Now we can construct a simulated sampling distribution

```
boot_straps = 500
sample_size
```

```
sample_means = np.zeros(boot_straps)
for ii in range(boot_straps):
    sample = np.random.choice(heights,
                              size=sample_size,
                              replace=True)
    sample_means[ii]= sample.mean()
```

```
# now we can find the standard deviation of the means
se_mean_height_bootstrap = sample_means.std()
print(f"The standard error of the mean calculated using bootstrapping is: {se_mean_height_
```

```
The standard error of the mean calculated using bootstrapping is: 0.175
```

Reminder, for the mean, it is also possible to estimate the Standard Error (SE) of the mean by dividing the standard deviation (STD) of the values by the square root of the sample size. The formula for the Standard Error of the Mean (SEM) is:

$\text{SEM} = \frac{\sigma}{\sqrt{n}}$

where σ is the standard deviation of the sample, and n is the sample size.

For other statistics, such as the median, there is no such analytical formula. The utility of the bootstrap method lies in its ability to estimate the SE of these other statistics. By repeatedly resampling the data with replacement and calculating the statistic of interest for each resample, bootstrapping provides an empirical distribution of the statistic. From this distribution, we can estimate the SE and other measures of variability.

```
se_mean_height_analytical = heights.std()/np.sqrt(sample_size)
print(f"The standard error of the mean calculated using the analytical formula is: {se_mea
```

```
The standard error of the mean calculated using the analytical formula is: 0.170
```

8.1 Confidence Intervals

We can also compute the confidence interval for our estimate of the mean. Let's say we want the 90 percent confidence interval.

```
conf_level = 0.9
low_bound = (1-conf_level)/2
up_bound = (1+conf_level)/2
```

```
CI = (np.quantile(sample_means, low_bound),
      np.quantile(sample_means, up_bound))
```

```
print(f"The lower bound of the confidence interval is: {CI[0]}")
print(f"The upper bound of the confidence interval is: {CI[1]}")
```

```
The lower bound of the confidence interval is: 162.0432374294097
The upper bound of the confidence interval is: 162.6211799101076
```

8.2 Can you bootsrap confidence interval for the median?

```
# make a bootstrapped population of the median
boot_straps = 500
sample_size
sample_medians = np.zeros(boot_straps)
for ii in range(boot_straps):
    sample = np.random.choice(heights,
                              size=sample_size,
                              replace=True)
    sample_medians[ii]= np.median(sample)

# compute CI for the bootstrapped population of the median
conf_level = 0.9
low_bound = (1-conf_level)/2
up_bound = (1+conf_level)/2
CI = (np.quantile(sample_medians, low_bound),
      np.quantile(sample_medians, up_bound))

# # uncomment below:
# print(f"The lower bound of the confidence interval is: {CI[0]}")
# print(f"The upper bound of the confidence interval is: {CI[1]}")
```

9 Tutorial 6

9.1 Topics

- Hypothesis testing using permutations
- t-test with the scipy.stats library

9.2 Hypothesis testing

A statistical hypothesis test is a method of statistical inference used to decide whether the data at hand sufficiently support a particular hypothesis.

https://en.wikipedia.org/wiki/Statistical_hypothesis_testing

9.3 Permutation tests

A permutation test (also called a randomization test, re-randomization test, or an exact test) is a type of statistical significance test in which the distribution of the test statistic under the null hypothesis is obtained by calculating all possible values of the test statistic under all possible rearrangements of the observed data points

https://en.wikipedia.org/wiki/Resampling_(statistics)#Permutation_tests

To illustrate the basic idea of a permutation test, suppose we collect random variables X_A and X_B for each individual from two groups A and B whose sample means are $\bar{x}_A$ and $\bar{x}_B$, and that we want to know whether X_A and X_B come from the same distribution. Let n_A and n_B be the sample size collected from each group. The permutation test is designed to determine whether the observed difference between the sample means is large enough to reject, at some significance level, the null hypothesis H_0 that the data drawn from A is from the same distribution as the data drawn from B.

The test proceeds as follows. First, the difference in means between the two samples is calculated: this is the observed value of the test statistic,T_{obs}.

Next, the observations of groups A and B are pooled, and the difference in sample means is calculated and recorded for every possible way of dividing the pooled values into two groups

of size n_A and n_B (i.e., for every permutation of the group labels A and B). The set of these calculated differences is the exact distribution of possible differences (for this sample) under the null hypothesis that group labels are exchangeable (i.e., are randomly assigned).

The one-sided p-value of the test is calculated as the proportion of sampled permutations where the difference in means was greater than or equal to T_{obs}. The two-sided p-value of the test is calculated as the proportion of sampled permutations where the [[absolute difference]] was greater than or equal to $|T_{\text{obs}}|$.

Alternatively, if the only purpose of the test is to reject or not reject the null hypothesis, one could sort the recorded differences, and then observe if T_{obs} is contained within the middle $(1-\alpha) \times 100\%$ of them, for some significance level α. If it is not, we reject the hypothesis of identical probability curves at the $\alpha \times 100\%$ significance level.

9.3.1 The Iris dataset

The data set consists of 50 samples from each of three species of Iris (Iris setosa, Iris virginica and Iris versicolor). Four features were measured from each sample: the length and the width of the sepals and petals, in centimeters.

9.4 Our Null Hypothesis

- There is no difference between mean sepal length for Iris virginca and Iris versicolor.

```
# import packages
import numpy as np
from sklearn import datasets
import pandas as pd
```

```
iris = datasets.load_iris()
df= pd.DataFrame(data= np.c_[iris['data'], iris['target']],
                 columns= iris['feature_names'] + ['target'])
df['species'] = pd.Categorical.from_codes(iris.target, iris.target_names)
```

```
df.head(10)
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa
5	5.4	3.9	1.7	0.4	0.0	setosa
6	4.6	3.4	1.4	0.3	0.0	setosa
7	5.0	3.4	1.5	0.2	0.0	setosa
8	4.4	2.9	1.4	0.2	0.0	setosa
9	4.9	3.1	1.5	0.1	0.0	setosa

We'll use a permutation test to compare the mean sepal length for Iris virginca and Iris versicolor. We start by comparing the actual mean sepal length for the species we're intrested in.

```
# group by species
by_species = df.groupby("species")
by_species.head(10)
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39706/2135608631.py:2: FutureWarni
  by_species = df.groupby("species")
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
0	5.1	3.5	1.4	0.2	0.0	setosa
1	4.9	3.0	1.4	0.2	0.0	setosa
2	4.7	3.2	1.3	0.2	0.0	setosa
3	4.6	3.1	1.5	0.2	0.0	setosa
4	5.0	3.6	1.4	0.2	0.0	setosa
5	5.4	3.9	1.7	0.4	0.0	setosa
6	4.6	3.4	1.4	0.3	0.0	setosa
7	5.0	3.4	1.5	0.2	0.0	setosa
8	4.4	2.9	1.4	0.2	0.0	setosa
9	4.9	3.1	1.5	0.1	0.0	setosa
50	7.0	3.2	4.7	1.4	1.0	versicolor
51	6.4	3.2	4.5	1.5	1.0	versicolor
52	6.9	3.1	4.9	1.5	1.0	versicolor
53	5.5	2.3	4.0	1.3	1.0	versicolor
54	6.5	2.8	4.6	1.5	1.0	versicolor
55	5.7	2.8	4.5	1.3	1.0	versicolor
56	6.3	3.3	4.7	1.6	1.0	versicolor
57	4.9	2.4	3.3	1.0	1.0	versicolor

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
58	6.6	2.9	4.6	1.3	1.0	versicolor
59	5.2	2.7	3.9	1.4	1.0	versicolor
100	6.3	3.3	6.0	2.5	2.0	virginica
101	5.8	2.7	5.1	1.9	2.0	virginica
102	7.1	3.0	5.9	2.1	2.0	virginica
103	6.3	2.9	5.6	1.8	2.0	virginica
104	6.5	3.0	5.8	2.2	2.0	virginica
105	7.6	3.0	6.6	2.1	2.0	virginica
106	4.9	2.5	4.5	1.7	2.0	virginica
107	7.3	2.9	6.3	1.8	2.0	virginica
108	6.7	2.5	5.8	1.8	2.0	virginica
109	7.2	3.6	6.1	2.5	2.0	virginica

Seperate out the species we're interested in

```
virginica = by_species.get_group("virginica")
versicolor = by_species.get_group("versicolor")
```

work with only 10 rows of each species (this is done to make things more interesting because the more rows we look at the more significant result we will get in the end)

```
n_rows = 10  # number of rows to keep should be less than 50
virginica = virginica[0:n_rows]
versicolor = versicolor[0:n_rows]
```

Concatenate the two species into one data frame

```
data = pd.concat([virginica, versicolor])
data
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
100	6.3	3.3	6.0	2.5	2.0	virginica
101	5.8	2.7	5.1	1.9	2.0	virginica
102	7.1	3.0	5.9	2.1	2.0	virginica
103	6.3	2.9	5.6	1.8	2.0	virginica
104	6.5	3.0	5.8	2.2	2.0	virginica
105	7.6	3.0	6.6	2.1	2.0	virginica
106	4.9	2.5	4.5	1.7	2.0	virginica
107	7.3	2.9	6.3	1.8	2.0	virginica

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species
108	6.7	2.5	5.8	1.8	2.0	virginica
109	7.2	3.6	6.1	2.5	2.0	virginica
50	7.0	3.2	4.7	1.4	1.0	versicolor
51	6.4	3.2	4.5	1.5	1.0	versicolor
52	6.9	3.1	4.9	1.5	1.0	versicolor
53	5.5	2.3	4.0	1.3	1.0	versicolor
54	6.5	2.8	4.6	1.5	1.0	versicolor
55	5.7	2.8	4.5	1.3	1.0	versicolor
56	6.3	3.3	4.7	1.6	1.0	versicolor
57	4.9	2.4	3.3	1.0	1.0	versicolor
58	6.6	2.9	4.6	1.3	1.0	versicolor
59	5.2	2.7	3.9	1.4	1.0	versicolor

Get the actual mean sepal length for each

```
grouped = data.groupby("species", observed=True)['sepal length (cm)'].mean()
print(f'THe mean of sepal length for virginica is {grouped["virginica"]}')
print(f'THe mean of sepal length for versicolor is {grouped["versicolor"]}')
```

```
THe mean of sepal length for virginica is 6.57
THe mean of sepal length for versicolor is 6.1
```

```
abs_dif = np.abs(grouped["virginica"] - grouped["versicolor"])
print(f'The difference between the means is {abs_dif}')
```

```
The difference between the means is 0.47000000000000064
```

```
data['species_shuffled'] = data['species'].sample(frac=1, replace=False).values
data
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species	specie
100	6.3	3.3	6.0	2.5	2.0	virginica	virgin
101	5.8	2.7	5.1	1.9	2.0	virginica	versic
102	7.1	3.0	5.9	2.1	2.0	virginica	versic
103	6.3	2.9	5.6	1.8	2.0	virginica	virgin
104	6.5	3.0	5.8	2.2	2.0	virginica	versic

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target	species	specie
105	7.6	3.0	6.6	2.1	2.0	virginica	versic
106	4.9	2.5	4.5	1.7	2.0	virginica	versic
107	7.3	2.9	6.3	1.8	2.0	virginica	versic
108	6.7	2.5	5.8	1.8	2.0	virginica	virgin
109	7.2	3.6	6.1	2.5	2.0	virginica	virgin
50	7.0	3.2	4.7	1.4	1.0	versicolor	versic
51	6.4	3.2	4.5	1.5	1.0	versicolor	virgin
52	6.9	3.1	4.9	1.5	1.0	versicolor	virgin
53	5.5	2.3	4.0	1.3	1.0	versicolor	versic
54	6.5	2.8	4.6	1.5	1.0	versicolor	virgin
55	5.7	2.8	4.5	1.3	1.0	versicolor	virgin
56	6.3	3.3	4.7	1.6	1.0	versicolor	versic
57	4.9	2.4	3.3	1.0	1.0	versicolor	virgin
58	6.6	2.9	4.6	1.3	1.0	versicolor	versic
59	5.2	2.7	3.9	1.4	1.0	versicolor	virgin

```
data['species_shuffled'] = data['species'].sample(frac=1, replace=False).values
grouped = data.groupby("species_shuffled", observed=True)['sepal length (cm)'].mean()
print(f'The mean of sepal length for virginica is {grouped["virginica"]}')
print(f'The mean of sepal length for versicolor is {grouped["versicolor"]}')

sampled_diff = np.abs(grouped["virginica"] - grouped["versicolor"])
print(f'The difference between the means is {sampled_diff}')
```

```
The mean of sepal length for virginica is 6.58
The mean of sepal length for versicolor is 6.09
The difference between the means is 0.4900000000000002
```

```
runs = 1000
diffs = np.zeros(runs)
for ii in range(runs):
    data['species_shuffled'] = data['species'].sample(frac=1, replace=False).values
    grouped = data.groupby("species_shuffled", observed=True)['sepal length (cm)'].mean()
    diffs[ii]= np.abs(grouped["virginica"] - grouped["versicolor"])
```

```
# compute our p-value
larger = np.where(diffs>=abs_dif, 1, 0)
p_val = np.sum(larger)/runs
```

```
print(f'The p-value is {p_val}')
np.sum(larger)
```

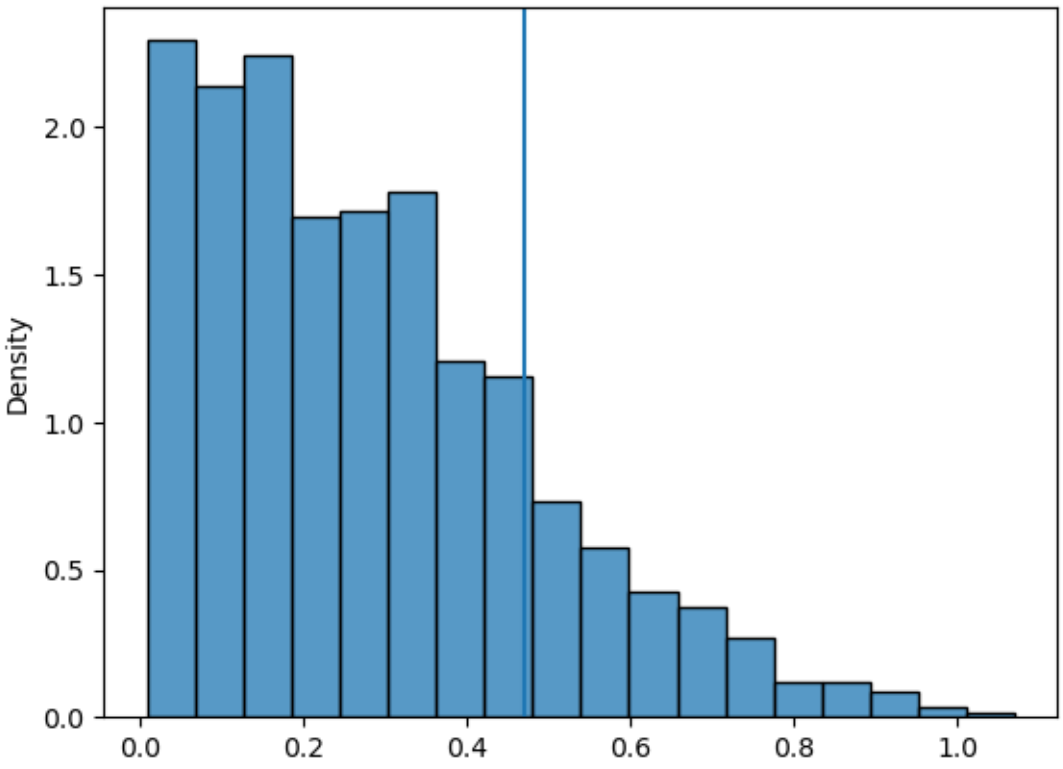
```
The p-value is 0.181
```

```
181
```

9.4.1 Let's look at it visually

```
import seaborn as sns
ax = sns.histplot(diffs, stat='density')
ax.axvline(abs_dif)
```

```
<matplotlib.lines.Line2D at 0x128ad8bd0>
```



10 3. Using `scipy.stats` tests

We can compare the results above to a t-test using the `scipy.stats` library

10.0.1 The scipy.stats library

This module contains a large number of probability distributions, summary and frequency statistics, correlation functions and statistical tests, masked statistics, kernel density estimation, quasi-Monte Carlo functionality, and more.

https://docs.scipy.org/doc/scipy/reference/stats.html

```
import scipy.stats as stats
```

```
stats.ttest_ind(virginica['sepal length (cm)'], versicolor['sepal length (cm)'])
```

```
TtestResult(statistic=1.3707421122118832, pvalue=0.18730715303330478, df=18.0)
```

11 Tutorial 7

11.1 Dependence between variables

1. Correlation and regression
2. Linear regression – OLS

11.1.1 1. Correlation and Regression

- Pearson's coefficient measures linear correlation (r).
- Spearman coefficient compares the 'ranks' of data.
- NumPy, SciPy, and Pandas all have functions that can be used to calculate these coefficients.

11.1.2 Correlation in Numpy (Pearson's r)

Required packages

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

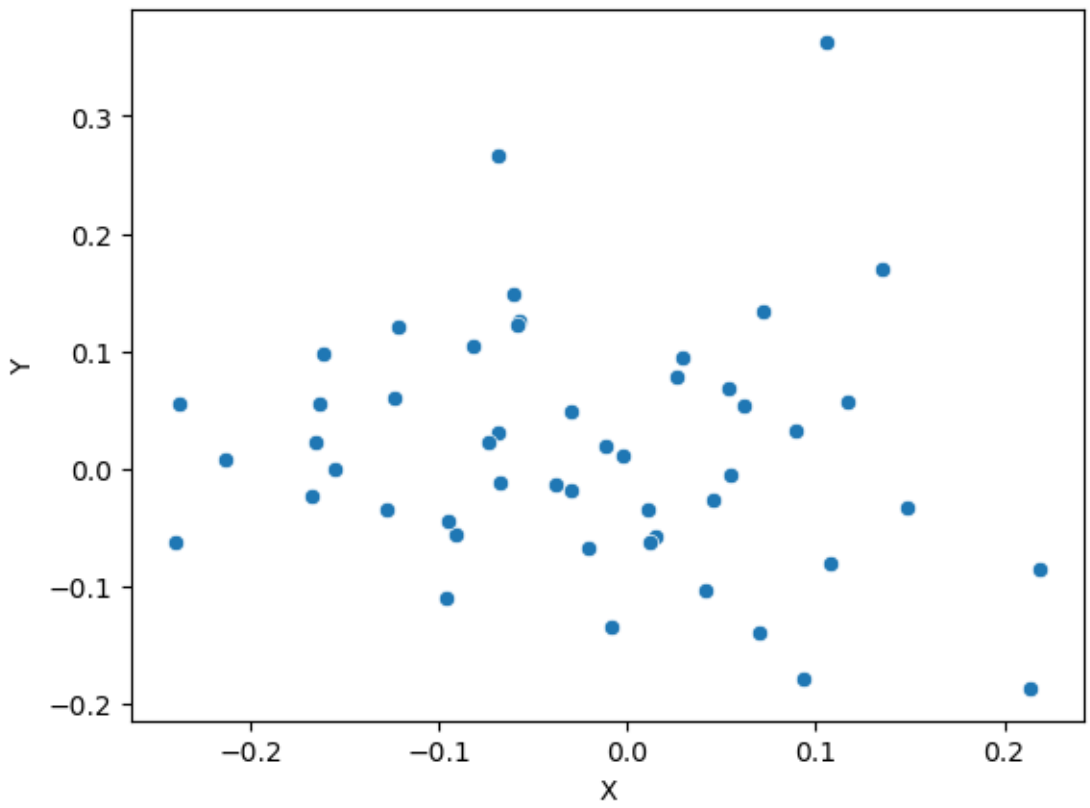
Let's make up some numbers

```
mu, sigma = 0, 0.1 # mean and standard deviation
x = np.random.normal(mu, sigma, 50)
y = np.random.normal(mu, sigma, 50)
z = np.random.normal(mu, sigma, 50)
```

What does our data look like?

```
fig, ax = plt.subplots()
sns.scatterplot(x=x, y=y, ax=ax)
ax.set(xlabel='X', ylabel='Y')
```

```
[Text(0.5, 0, 'X'), Text(0, 0.5, 'Y')]
```



Use Numpy to get pearson's correlation between `x` and `y`

```
np.corrcoef(x, y)
```

```
array([[ 1.        , -0.12455511],
       [-0.12455511,  1.        ]])
```

11.1.3 The result is a correlation matrix. Each cell in the table shows the correlation between two variables.

- The values on the main diagonal of the correlation matrix (upper left and lower right) are equal to 1. These corresponds to the correlation coefficient for `x` and `x` and `y` and `y`, so they will always be equal to 1.

- The values on the bottom left and top right show the pearson's correlation coefficient for `x` and `y`.

We can do the same thing with more than two variables

```
np.corrcoef([x, y, z])
```

```
array([[ 1.        , -0.12455511,  0.03365426],
       [-0.12455511,  1.        , -0.14554429],
       [ 0.03365426, -0.14554429,  1.        ]])
```

11.2 Correlation in SciPy (Pearson and Spearman)

Import required packages

```
import scipy.stats
```

```
# pearson
scipy.stats.pearsonr(x, y)
```

```
PearsonRResult(statistic=-0.12455510674830719, pvalue=0.3887841983709771)
```

```
# spearman
scipy.stats.spearmanr(x, y)
```

```
SignificanceResult(statistic=-0.13661464585834332, pvalue=0.34413653070929573)
```

11.2.0.1 Note: these functions return both the correlation coefficient and the p-value

11.3 Correlation in Pandas

Time for some real data :)

11.3.1 Today's dataset: Birthweight

This dataset contains information on new born babies and their parents.

https://www.sheffield.ac.uk/mash/statistics/datasets

Required packages

```
import pandas as pd
```

```
# import dataset
file = "./birthweight.csv"
df = pd.read_csv(file)
```

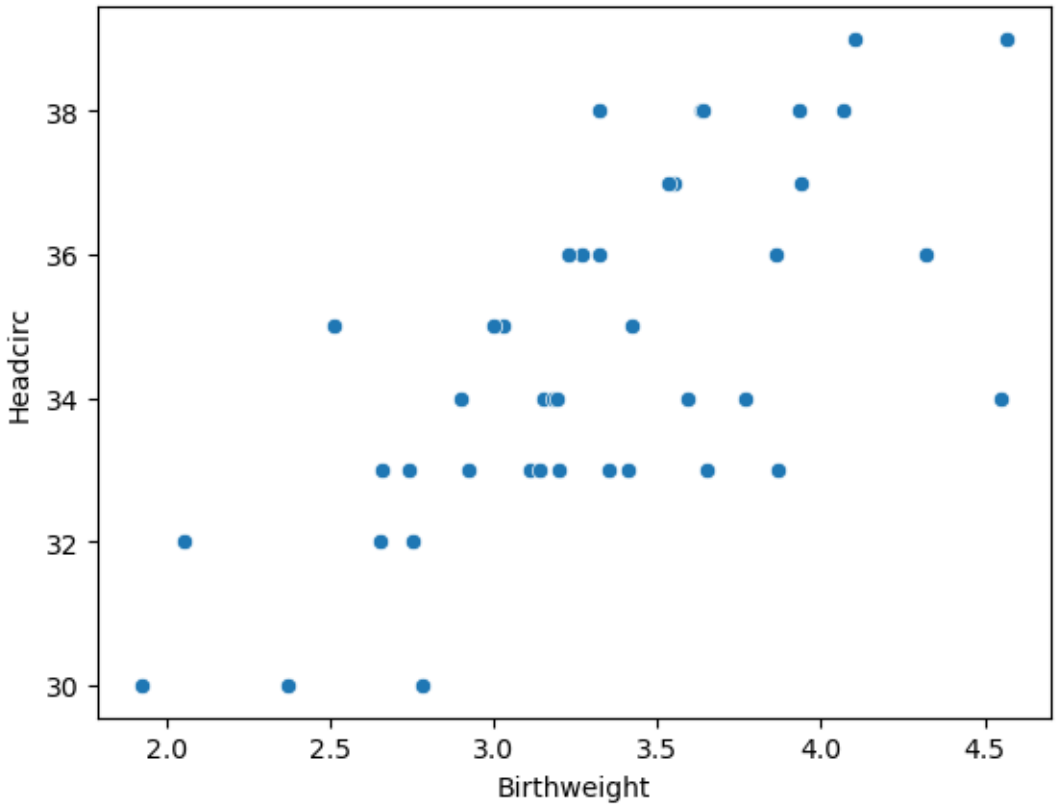
```
df.head()
```

	Length	Birthweight	Headcirc	Gestation	smoker	mage	mnocig	mheight	mppwt	fage	fedyrs
0	56	4.55	34	44	0	20	0	162	57	23	10
1	53	4.32	36	40	0	19	0	171	62	19	12
2	58	4.10	39	41	0	35	0	172	58	31	16
3	53	4.07	38	44	0	20	0	174	68	26	14
4	54	3.94	37	42	0	24	0	175	66	30	12

11.3.2 Is there any correlation between birthweight and head circumfrence?

```
# First, let's visualize the data
sns.scatterplot(x=df['Birthweight'], y=df['Headcirc'])
```

```
<Axes: xlabel='Birthweight', ylabel='Headcirc'>
```




```
df['Birthweight'].corr(df['Headcirc'])
```

```
0.6846156184774087
```

11.3.3 There are lots of ways to calculate correlation in Pandas...

```
df['Birthweight'].corr(df['Headcirc'],
                       method='pearson')
```

```
0.6846156184774087
```

```
df['Birthweight'].corr(df['Headcirc'],
                       method='spearman')
```

```
0.6772599775176383
```

11.3.4 We can also find the correlation between all the variables in our dataframe at once

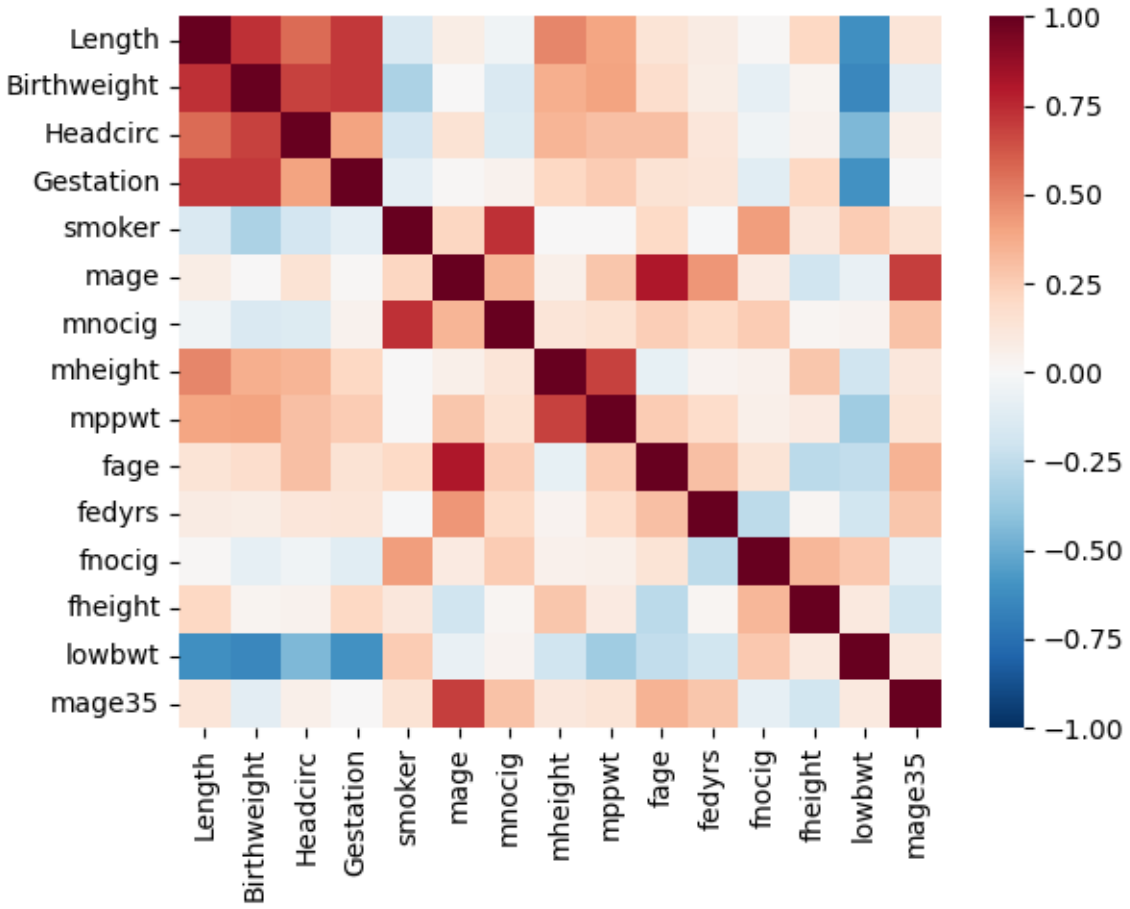
```
pearsoncorr = df.corr(method ='pearson')
pearsoncorr
```

	Length	Birthweight	Headcirc	Gestation	smoker	mage	mnocig	mheigl
Length	1.000000	0.726833	0.563172	0.705111	-1.534062e-01	0.075268	-0.039843	0.4849
Birthweight	0.726833	1.000000	0.684616	0.708303	-3.142339e-01	0.000173	-0.152335	0.3630
Headcirc	0.563172	0.684616	1.000000	0.404635	-1.828719e-01	0.145842	-0.132988	0.3370
Gestation	0.705111	0.708303	0.404635	1.000000	-9.474608e-02	0.010778	0.043195	0.2105
smoker	-0.153406	-0.314234	-0.182872	-0.094746	1.000000e+00	0.212479	0.727218	0.0003
mage	0.075268	0.000173	0.145842	0.010778	2.124788e-01	1.000000	0.340294	0.0599
mnocig	-0.039843	-0.152335	-0.132988	0.043195	7.272181e-01	0.340294	1.000000	0.1264
mheight	0.484992	0.363055	0.337047	0.210503	3.532676e-04	0.059956	0.126439	1.0000
mppwt	0.398197	0.400886	0.302854	0.255082	9.808342e-16	0.274168	0.148945	0.6806
fage	0.137184	0.175710	0.301151	0.142175	1.975014e-01	0.806584	0.248425	-0.079
fedyrs	0.079485	0.071045	0.123892	0.130987	-1.489058e-02	0.441683	0.198526	0.0352
fnocig	0.008800	-0.093136	-0.046837	-0.113831	4.176330e-01	0.090927	0.257307	0.0483
fheight	0.208358	0.031022	0.041509	0.207597	1.106327e-01	-0.199547	0.020672	0.2743
lowbwt	-0.609928	-0.651964	-0.446849	-0.602935	2.530122e-01	-0.076394	0.035384	-0.198
mage35	0.130502	-0.108947	0.055386	0.007395	1.469385e-01	0.692664	0.290574	0.1160

11.3.5 That isn't so helpful... we can also make a heatmap to display the data visually

```
fig, ax = plt.subplots()

sns.heatmap(pearsoncorr,
            cmap='RdBu_r', ax=ax, vmin=-1, vmax=1)
```

```
<Axes: >
```



11.4 Linear Regression

In statistics, linear regression is a linear approach to modeling the relationship between a scalar response and one or more explanatory variables (also known as dependent and independent variables).

https://en.wikipedia.org/wiki/Linear_regression

We have a lot of options for for studying linear regression

11.4.1 Scipy.stats

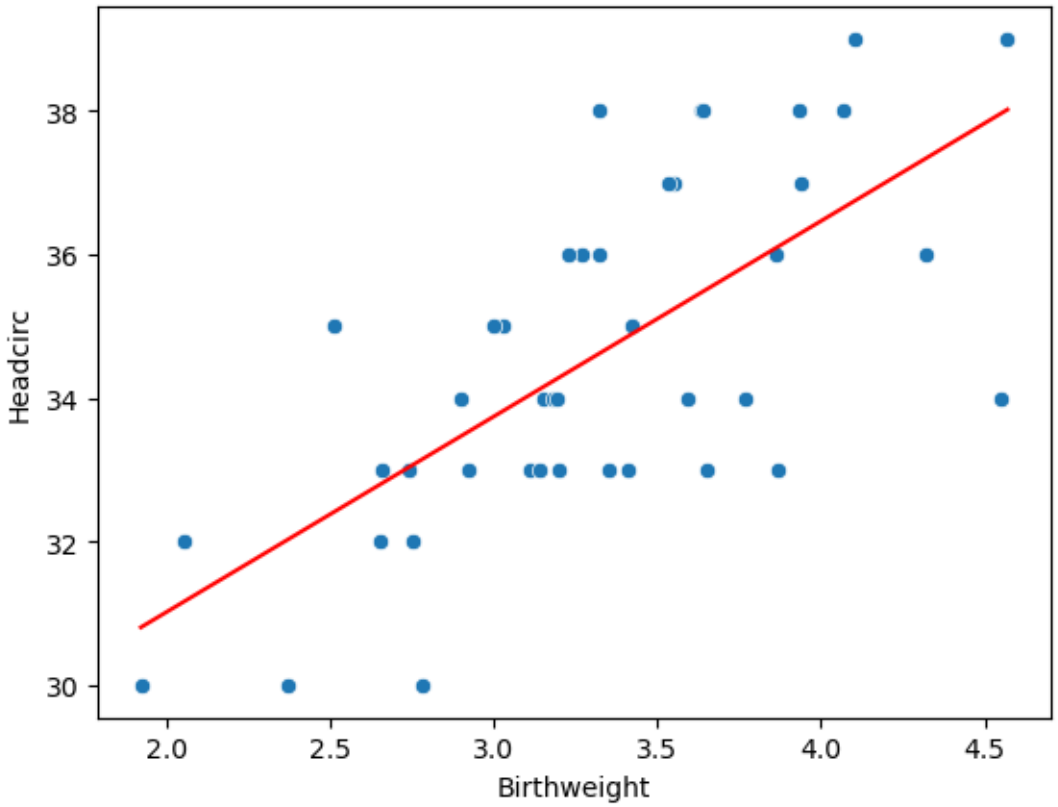
```
result = scipy.stats.linregress(df['Birthweight'],
                                df['Headcirc'])
```

```
print(result)
```

```
LinregressResult(slope=2.720563928236349, intercept=25.58239845298082, rvalue=0.6846156184774
```

A reminder of what the data look like...

```
fig, ax = plt.subplots()
sns.scatterplot(x=df['Birthweight'], y=df['Headcirc'], ax=ax)
x = np.linspace(df['Birthweight'].min(), df['Birthweight'].max(), 100)
y = result.intercept + result.slope * x
ax.plot(x, y, color='red')
```




```
result.slope
```

```
2.720563928236349
```

```
result.intercept
```

```
25.58239845298082
```

```
result.rvalue**2
```

```
0.46869854506320485
```

11.4.2 Using sklearn

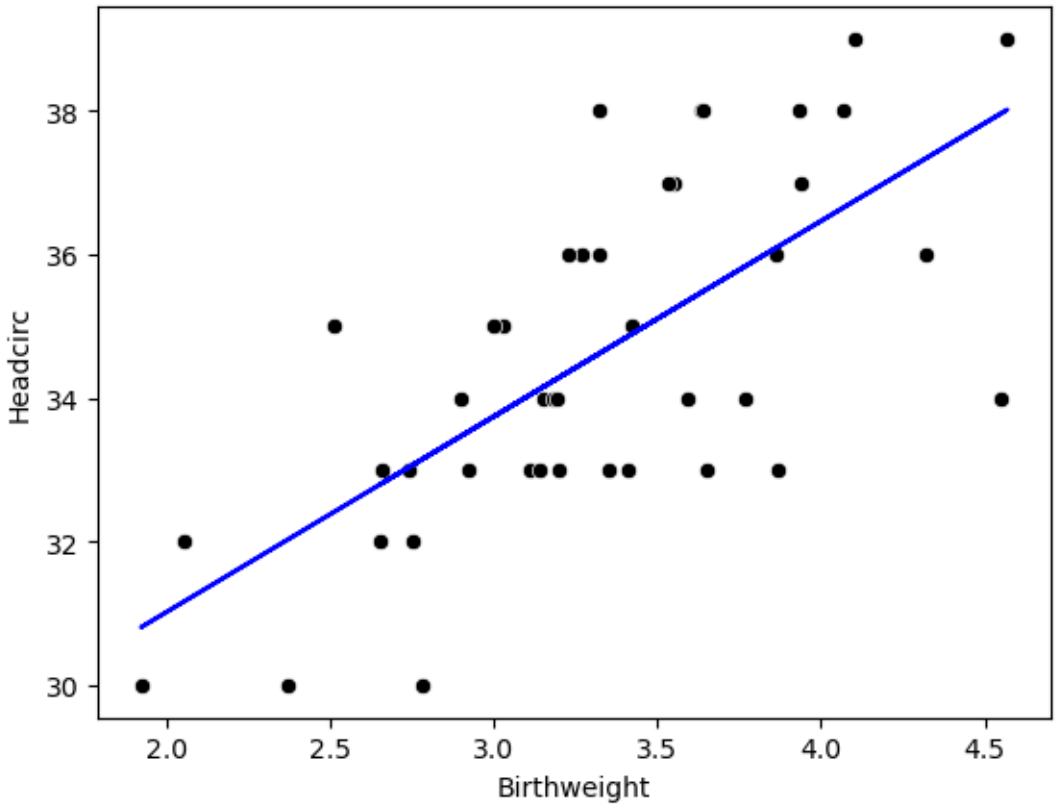
```
from sklearn import linear_model
```

```
my_model = linear_model.LinearRegression()
results = my_model.fit(df[['Birthweight']], df[['Headcirc']])
print("The linear model is: Y = {:.5} + {:.5}X".format(results.intercept_[0],
                                                       results.coef_[0][0]))
```

```
The linear model is: Y = 25.582 + 2.7206X
```

```
# the results are the same, but the sklearn package has some other features
predictions = results.predict(df[['Birthweight']])
```

```
fig, ax = plt.subplots()
# we can add the predicted values to the original plot
ax = sns.scatterplot(x=df['Birthweight'],
                     y=df['Headcirc'],
                     color="0.0")
ax.plot(df['Birthweight'],
        predictions,
        color="b")
```



11.5 Hypothesis testing

- Null hypothesis 1: The actual intercept is equal to zero
- Null hypothesis 2: The actual slope is equal to zero

Using the `statsmodels.api` package

```
import statsmodels.api as sm
```

Set up the model

```
X = sm.add_constant(df['Birthweight'])
Y = df['Headcirc']
model = sm.OLS(Y, X)  # OLS = ordinary least squares
results = model.fit()
print(results.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:              Headcirc   R-squared:                       0.469
Model:                          OLS   Adj. R-squared:                  0.455
Method:               Least Squares   F-statistic:                     35.29
Date:              Tue, 16 Jul 2024   Prob (F-statistic):           5.73e-07
Time:                      09:12:12   Log-Likelihood:                -82.574
No. Observations:                42   AIC:                             169.1
Df Residuals:                    40   BIC:                             172.6
Df Model:                         1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          25.5824      1.542     16.594      0.000      22.467      28.698
Birthweight     2.7206      0.458      5.940      0.000       1.795       3.646
==============================================================================
Omnibus:                        1.089   Durbin-Watson:                   2.132
Prob(Omnibus):                  0.580   Jarque-Bera (JB):                1.007
Skew:                          -0.193   Prob(JB):                        0.604
Kurtosis:                       2.347   Cond. No.                         20.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

11.5.0.1 The P-value is the answer to the question "how likely is it that we'd get a test statistic t* as extreme as we did if the null hypothesis were true?

Does this output correspond to a one-tailed or two-tailed test?

If we want to test whether the slope is different from 0, we need a two-sided test.

If we want to test a specific direction, we can use a one-sided test. To do this, we need to divide the p-value in the table above in half.

Read more: https://stats.idre.ucla.edu/other/mult-pkg/faq/general/faq-what-are-the-differences-between-one-tailed-and-two-tailed-tests/

```
print(f'p value of the intercept: {results.pvalues.iloc[0]}, p value of the slope: {result
```

```
p value of the intercept: 1.5527553380020346e-19, p value of the slope: 5.734797978444454e-0
```

11.5.0.2 Can we compute the P-value using permutations?.... YES!

12 Groupby

```
import pandas as pd
```

The dataset we're going to use has information on members of the US congress - first and last names - birth date - gender - type ("rep" for House of Representatives or "sen" for Senate) - U.S. state - political party.

```
dtypes = {
    "first_name": "category",
    "gender": "category",
    "type": "category",
    "state": "category",
    "party": "category",
}
df = pd.read_csv(
    "./congress.csv",
    dtype=dtypes,
    usecols=list(dtypes) + ["birthday", "last_name"],
    parse_dates=["birthday"]
)
```

```
df.shape
```

```
(12048, 7)
```

```
df
```

	last_name	first_name	birthday	gender	type	state	party
0	Bassett	Richard	1745-04-02	M	sen	DE	Anti-Administration
1	Bland	Theodorick	1742-03-21	M	rep	VA	NaN
2	Burke	Aedanus	1743-06-16	M	rep	SC	NaN
3	Carroll	Daniel	1730-07-22	M	rep	MD	NaN

	last_name	first_name	birthday	gender	type	state	party
4	Clymer	George	1739-03-16	M	rep	PA	NaN
...	...	...	...	...	...	...	...
12043	Loeffler	Kelly	1970-11-27	F	sen	GA	Republican
12044	Wright	Ron	1953-04-08	M	rep	TX	Republican
12045	Fudge	Marcia	1952-10-29	F	rep	OH	Democrat
12046	Haaland	Debra	1960-12-02	F	rep	NM	Democrat
12047	Hastings	Alcee	1936-09-05	M	rep	FL	Democrat

```
df.dtypes
```

```
last_name              object
first_name           category
birthday      datetime64[ns]
gender               category
type                 category
state                category
party                category
dtype: object
```

What is the count of Congressional members, on a state-by-state basis, over the entire history of the dataset?

You call `.groupby()` and pass the name of the column you want to group on, which is "state". Then, you use `["last_name"]` to specify the columns on which you want to perform the actual aggregation.

```
df.groupby("state")["last_name"].count()
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/3228017122.py:1: FutureWarni
  df.groupby("state")["last_name"].count()
```

```
state
AK      16
AL     209
AR     117
AS       2
AZ      49
CA     367
```

CO	92
CT	240
DC	2
DE	97
DK	9
FL	161
GA	317
GU	4
HI	24
IA	205
ID	59
IL	488
IN	343
KS	143
KY	373
LA	199
MA	427
MD	305
ME	175
MI	296
MN	161
MO	334
MS	155
MT	53
NC	356
ND	44
NE	127
NH	181
NJ	359
NM	57
NV	56
NY	1467
OH	675
OK	93
OL	2
OR	90
PA	1053
PI	13
PR	19
RI	107
SC	251
SD	51
TN	301

```
TX     263
UT      55
VA     433
VI       4
VT     115
WA      96
WI     198
WV     120
WY      40
Name: last_name, dtype: int64
```

You can pass a lot more than just a single column name to `.groupby()` as the first argument. You can also specify any of the following: - A list of multiple column names - A dictionary - A Pandas Series - A NumPy array or Pandas Index

12.1 multi index

Here's an example of grouping jointly on two columns, which finds the count of Congressional members broken out by state and then by gender:

```
df.groupby(["state", "gender"])["last_name"].count()
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/503199787.py:1: FutureWarnin
  df.groupby(["state", "gender"])["last_name"].count()
```

```
state  gender
AK     F            0
       M           16
AL     F            4
       M          205
AR     F            5
                  ...
WI     M          198
WV     F            1
       M          119
WY     F            1
       M           39
Name: last_name, Length: 116, dtype: int64
```

Note: When we use `groupby` on more than one column, we create a `MultiIndex`

```
n_by_state_gender = df.groupby(["state", "gender"])["last_name"].count()
type(n_by_state_gender)
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/197250217.py:1: FutureWarnin
  n_by_state_gender = df.groupby(["state", "gender"])["last_name"].count()
```

```
pandas.core.series.Series
```

```
n_by_state_gender.index
```

```
MultiIndex([('AK', 'F'),
            ('AK', 'M'),
            ('AL', 'F'),
            ('AL', 'M'),
            ('AR', 'F'),
            ('AR', 'M'),
            ('AS', 'F'),
            ('AS', 'M'),
            ('AZ', 'F'),
            ('AZ', 'M'),
            ...
            ('VT', 'F'),
            ('VT', 'M'),
            ('WA', 'F'),
            ('WA', 'M'),
            ('WI', 'F'),
            ('WI', 'M'),
            ('WV', 'F'),
            ('WV', 'M'),
            ('WY', 'F'),
            ('WY', 'M')],
           names=['state', 'gender'], length=116)
```

If we want to keep the data in columns, we can use the `as_index` parameter

```
df.groupby(["state", "gender"], as_index=False)["last_name"].count()
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/1045572165.py:1: FutureWarni
  df.groupby(["state", "gender"], as_index=False)["last_name"].count()
```

	state	gender	last_name
0	AK	F	0
1	AK	M	16
2	AL	F	4
3	AL	M	205
4	AR	F	5
...	...	...	...
111	WI	M	198
112	WV	F	1
113	WV	M	119
114	WY	F	1
115	WY	M	39

You might have noticed that `groupby` automatically sorts the data. We can also change this, if we want.

```
df.groupby("state", sort=False)["last_name"].count()
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/966111394.py:1: FutureWarnin
  df.groupby("state", sort=False)["last_name"].count()
```

```
state
DE      97
VA     433
SC     251
MD     305
PA    1053
MA     427
NJ     359
GA     317
NY    1467
NC     356
CT     240
VT     115
KY     373
RI     107
NH     181
TN     301
OH     675
MS     155
```

```
OL       2
IN     343
LA     199
IL     488
MO     334
AL     209
AR     117
ME     175
FL     161
MI     296
IA     205
WI     198
TX     263
CA     367
OR      90
MN     161
NM      57
NE     127
WA      96
KS     143
UT      55
NV      56
CO      92
WV     120
DK       9
AZ      49
ID      59
MT      53
WY      40
DC       2
ND      44
SD      51
OK      93
HI      24
PR      19
AK      16
PI      13
VI       4
GU       4
AS       2
Name: last_name, dtype: int64
```

12.2 What is actually happening here?

```
by_state = df.groupby("state")
by_state
```

```
/var/folders/wn/2bz1970d2w5182zy7h96yfcc0000gn/T/ipykernel_39943/3630249550.py:1: FutureWarni
  by_state = df.groupby("state")
```

```
<pandas.core.groupby.generic.DataFrameGroupBy object at 0x12f9b0890>
```

```
for state, frame in by_state:
    print(f"First 2 entries for {state!r}")
    print("------------------------")
    print(frame.head(2), end="\n\n")
```

```
First 2 entries for 'AK'
------------------------
     last_name first_name   birthday gender type state        party
6617    Waskey      Frank 1875-04-20      M  rep    AK     Democrat
6645      Cale     Thomas 1848-09-17      M  rep    AK  Independent

First 2 entries for 'AL'
------------------------
    last_name first_name   birthday gender type state       party
911   Crowell       John 1780-09-18      M  rep    AL  Republican
990    Walker       John 1783-08-12      M  sen    AL  Republican

First 2 entries for 'AR'
------------------------
     last_name first_name   birthday gender type state party
1000     Bates      James 1788-08-25      M  rep    AR   NaN
1278    Conway      Henry 1793-03-18      M  rep    AR   NaN

First 2 entries for 'AS'
------------------------
          last_name first_name   birthday gender type state     party
10795         Sunia       Fofó 1937-03-13      M  rep    AS  Democrat
11752  Faleomavaega        Eni 1943-08-15      M  rep    AS  Democrat
```

```
First 2 entries for 'AZ'
------------------------
     last_name first_name   birthday gender type state       party
3672    Poston    Charles 1825-04-20      M  rep    AZ  Republican
3723   Goodwin       John 1824-10-18      M  rep    AZ  Republican

First 2 entries for 'CA'
------------------------
     last_name first_name   birthday gender type state        party
2648   Gilbert     Edward        NaT      M  rep    CA     Democrat
2740    Wright     George 1816-06-04      M  rep    CA  Independent

First 2 entries for 'CO'
------------------------
     last_name first_name   birthday gender type state  \
3612    Bennet      Hiram 1826-09-02      M  rep    CO
3931  Bradford      Allen 1815-07-23      M  rep    CO

                       party
3612  Conservative Republican
3931               Republican

First 2 entries for 'CT'
------------------------
     last_name first_name   birthday gender type state                party
14  Huntington   Benjamin 1736-04-19      M  rep    CT                  NaN
16     Johnson    William 1727-10-07      M  sen    CT  Pro-Administration

First 2 entries for 'DC'
------------------------
      last_name first_name   birthday gender type state       party
4230    Chipman     Norton 1834-03-07      M  rep    DC  Republican
10819  Fauntroy     Walter 1933-02-06      M  rep    DC    Democrat

First 2 entries for 'DE'
------------------------
   last_name first_name   birthday gender type state                 party
0    Bassett    Richard 1745-04-02      M  sen    DE  Anti-Administration
40      Read     George 1733-09-18      M  sen    DE                   NaN

First 2 entries for 'DK'
------------------------
     last_name first_name   birthday gender type state     party
```

```
3640     Jayne    William 1826-10-08      M  rep    DK       NaN
3685      Todd       John 1814-04-04      M  rep    DK  Democrat

First 2 entries for 'FL'
------------------------
      last_name first_name   birthday gender type state party
1030  Hernández     Joseph 1788-05-26      M  rep    FL   NaN
1089       Call    Richard 1792-10-24      M  rep    FL   NaN

First 2 entries for 'GA'
------------------------
   last_name first_name   birthday gender type state                party
8        Few    William 1748-06-08      M  sen    GA  Anti-Administration
20   Mathews     George 1739-08-30      M  rep    GA                  NaN

First 2 entries for 'GU'
------------------------
      last_name first_name   birthday gender type state       party
10702   Won Pat    Antonio 1908-12-10      M  rep    GU    Democrat
10872      Blaz        Ben 1928-02-14      M  rep    GU  Republican

First 2 entries for 'HI'
------------------------
     last_name first_name   birthday gender type state       party
6393    Wilcox     Robert 1855-02-15      M  rep    HI         NaN
7522   Baldwin      Henry 1871-01-12      M  rep    HI  Republican

First 2 entries for 'IA'
------------------------
     last_name first_name   birthday gender type state     party
1930   Chapman    William 1808-08-11      M  rep    IA  Democrat
2390  Hastings   Serranus 1813-11-22      M  rep    IA  Democrat

First 2 entries for 'ID'
------------------------
     last_name first_name   birthday gender type state       party
3691   Wallace    William 1811-07-19      M  rep    ID  Republican
3844  Holbrook     Edward 1836-05-06      M  rep    ID    Democrat

First 2 entries for 'IL'
------------------------
      last_name first_name   birthday gender type state party
595        Bond   Shadrack 1773-11-24      M  rep    IL   NaN
```

```
772  Stephenson   Benjamin        NaT      M  rep    IL   NaN

First 2 entries for 'IN'
------------------------
    last_name first_name   birthday gender type state       party
431     Parke   Benjamin 1777-09-22      M  rep    IN         NaN
984    Taylor     Waller        NaT      M  sen    IN  Republican

First 2 entries for 'KS'
------------------------
      last_name first_name   birthday gender type state       party
3161  Whitfield       John 1818-03-11      M  rep    KS    Democrat
3420    Parrott     Marcus 1828-10-27      M  rep    KS  Republican

First 2 entries for 'KY'
------------------------
    last_name   first_name birthday gender type state                party
55    Edwards         John      NaT      M  sen    KY  Anti-Administration
113   Greenup  Christopher      NaT      M  rep    KY           Republican

First 2 entries for 'LA'
------------------------
     last_name first_name birthday gender type state       party
584  Destréhan       Jean      NaT      M  sen    LA     Unknown
585   Magruder      Allan      NaT      M  sen    LA  Republican

First 2 entries for 'MA'
------------------------
   last_name first_name   birthday gender type state               party
6     Dalton   Tristram 1738-05-28      M  sen    MA  Pro-Administration
12     Grout   Jonathan 1737-07-23      M  rep    MA                 NaN

First 2 entries for 'MD'
------------------------
  last_name first_name   birthday gender type state party
3   Carroll     Daniel 1730-07-22      M  rep    MD   NaN
5    Contee   Benjamin        NaT      M  rep    MD   NaN

First 2 entries for 'ME'
------------------------
     last_name first_name   birthday gender type state       party
1018      Dane     Joseph 1778-10-25      M  rep    ME  Federalist
1028    Harris       Mark 1779-01-27      M  rep    ME  Republican
```

```
First 2 entries for 'MI'
------------------------
     last_name first_name   birthday gender type state party
1061    Sibley    Solomon 1769-10-07      M  rep    MI   NaN
1132   Richard    Gabriel 1767-10-15      M  rep    MI   NaN

First 2 entries for 'MN'
------------------------
      last_name first_name   birthday gender type state     party
2864     Sibley      Henry 1811-02-20      M  rep    MN       NaN
3239  Kingsbury    William 1828-06-04      M  rep    MN  Democrat

First 2 entries for 'MO'
------------------------
     last_name first_name   birthday gender type state party
627  Hempstead     Edward 1780-06-03      M  rep    MO   NaN
712     Easton      Rufus 1774-05-04      M  rep    MO   NaN

First 2 entries for 'MS'
------------------------
    last_name  first_name   birthday gender type state party
256    Greene      Thomas 1758-02-26      M  rep    MS   NaN
260    Hunter  Narsworthy        NaT      M  rep    MS   NaN

First 2 entries for 'MT'
------------------------
      last_name first_name   birthday gender type state     party
3754     McLean     Samuel 1826-08-07      M  rep    MT  Democrat
3939  Cavanaugh      James 1823-07-04      M  rep    MT  Democrat

First 2 entries for 'NC'
------------------------
   last_name first_name   birthday gender type state                party
13   Hawkins   Benjamin 1754-08-15      M  sen    NC  Pro-Administration
17  Johnston     Samuel 1733-12-15      M  sen    NC  Pro-Administration

First 2 entries for 'ND'
------------------------
     last_name first_name   birthday gender type state       party
5382     Casey      Lyman 1837-05-06      M  sen    ND  Republican
5477    Pierce    Gilbert 1839-01-11      M  sen    ND  Republican
```

```
First 2 entries for 'NE'
------------------------
     last_name first_name   birthday gender type state     party
2952  Giddings   Napoleon 1816-01-02      M  rep    NE  Democrat
3066   Chapman       Bird 1821-08-24      M  rep    NE  Democrat

First 2 entries for 'NH'
------------------------
    last_name first_name   birthday gender type state       party
99    Wingate      Paine 1739-05-14      M  rep    NH         NaN
120   Langdon       John 1741-06-26      M  sen    NH  Republican

First 2 entries for 'NJ'
------------------------
   last_name first_name   birthday gender type state              party
7      Elmer   Jonathan 1745-11-29      M  sen    NJ  Pro-Administration
23  Paterson    William 1745-12-24      M  sen    NJ  Pro-Administration

First 2 entries for 'NM'
------------------------
      last_name first_name   birthday gender type state     party
2883  Weightman    Richard 1816-12-28      M  rep    NM  Democrat
3418      Otero     Miguel 1829-06-21      M  rep    NM       NaN

First 2 entries for 'NV'
------------------------
        last_name first_name   birthday gender type state       party
3508  Cradlebaugh       John 1819-02-22      M  rep    NV         NaN
3662         Mott     Gordon 1812-10-21      M  rep    NV  Republican

First 2 entries for 'NY'
------------------------
         last_name first_name   birthday gender type state party
9            Floyd    William 1734-12-17      M  rep    NY   NaN
26  Van Rensselaer   Jeremiah 1738-08-27      M  rep    NY   NaN

First 2 entries for 'OH'
------------------------
    last_name first_name   birthday gender type state       party
226  McMillan    William 1764-03-02      M  rep    OH         NaN
254   Fearing       Paul 1762-02-28      M  rep    OH  Federalist

First 2 entries for 'OK'
```

```
------------------------
     last_name first_name   birthday gender type state        party
5599    Harvey      David 1845-03-20      M  rep    OK   Republican
6058  Callahan      James 1852-12-19      M  rep    OK  Free Silver

First 2 entries for 'OL'
------------------------
    last_name first_name   birthday gender type state party
404     Clark     Daniel        NaT      M  rep    OL   NaN
503   Poydras     Julien 1740-04-03      M  rep    OL   NaN

First 2 entries for 'OR'
------------------------
     last_name first_name   birthday gender type state     party
2726  Thurston     Samuel 1816-04-15      M  rep    OR  Democrat
3396      Lane     Joseph 1801-12-14      M  sen    OR  Democrat

First 2 entries for 'PA'
------------------------
   last_name first_name   birthday gender type state                party
4     Clymer     George 1739-03-16      M  rep    PA                  NaN
19    Maclay    William 1737-07-20      M  sen    PA  Anti-Administration

First 2 entries for 'PI'
------------------------
             last_name first_name   birthday gender type state party
6833            Ocampo      Pablo 1853-01-25      M  rep    PI   NaN
6937  Legarda Y Tuason     Benito 1853-09-27      M  rep    PI   NaN

First 2 entries for 'PR'
------------------------
      last_name first_name   birthday gender type state       party
6424    Degetau   Federico 1862-12-05      M  rep    PR  Republican
6809  Larrinaga      Tulio 1847-01-15      M  rep    PR    Unionist

First 2 entries for 'RI'
------------------------
    last_name first_name   birthday gender type state       party
61   Bradford    William 1729-11-04      M  sen    RI  Federalist
105    Bourne   Benjamin 1755-09-09      M  rep    RI  Federalist

First 2 entries for 'SC'
------------------------
```

```
   last_name first_name   birthday gender type state               party
2      Burke    Aedanus 1743-06-16      M  rep    SC                 NaN
15     Izard      Ralph        NaT      M  sen    SC  Pro-Administration

First 2 entries for 'SD'
------------------------
     last_name first_name   birthday gender type state       party
5421   Gifford      Oscar 1842-10-20      M  rep    SD  Republican
5460     Moody     Gideon 1832-10-16      M  sen    SD  Republican

First 2 entries for 'TN'
------------------------
    last_name first_name   birthday gender type state       party
141     White      James 1749-06-16      M  rep    TN         NaN
142    Blount    William 1749-03-26      M  sen    TN  Republican

First 2 entries for 'TX'
------------------------
     last_name first_name   birthday gender type state     party
2567  Pilsbury    Timothy 1789-04-12      M  rep    TX  Democrat
2669   Kaufman      David 1813-12-18      M  rep    TX  Democrat

First 2 entries for 'UT'
------------------------
      last_name first_name   birthday gender type state     party
3482  Bernhisel       John 1799-07-23      M  rep    UT      Whig
3645     Kinney       John 1816-04-02      M  rep    UT  Democrat

First 2 entries for 'VA'
------------------------
   last_name  first_name   birthday gender type state                party
1      Bland  Theodorick 1742-03-21      M  rep    VA                  NaN
11   Grayson     William        NaT      M  sen    VA  Anti-Administration

First 2 entries for 'VI'
------------------------
      last_name first_name   birthday gender type state       party
10494     Evans     Melvin 1917-08-07      M  rep    VI  Republican
11086   de Lugo        Ron 1930-08-02      M  rep    VI    Democrat

First 2 entries for 'VT'
------------------------
   last_name first_name   birthday gender type state               party
```

```
41  Robinson      Moses 1741-03-22      M  sen    VT  Anti-Administration
86     Niles  Nathaniel 1741-04-03      M  rep    VT                  NaN

First 2 entries for 'WA'
------------------------
      last_name first_name   birthday gender type state     party
2977  Lancaster   Columbia 1803-08-26      M  rep    WA  Democrat
3050   Anderson      James 1822-02-16      M  rep    WA  Democrat

First 2 entries for 'WI'
------------------------
     last_name first_name   birthday gender type state     party
2409    Martin     Morgan 1805-03-31      M  rep    WI  Democrat
2502   Darling      Mason 1801-05-18      M  rep    WI  Democrat

First 2 entries for 'WV'
------------------------
       last_name first_name   birthday gender type state  \
3613       Blair      Jacob 1821-04-11      M  rep    WV
3688  Van Winkle      Peter 1808-09-07      M  sen    WV

                      party
3613  Unconditional Unionist
3688              Republican

First 2 entries for 'WY'
------------------------
     last_name first_name   birthday gender type state       party
4007  Nuckolls    Stephen 1825-08-16      M  rep    WY    Democrat
4136     Jones    William 1842-02-20      M  rep    WY  Republican
```

When you use `groupby`, you automatically create a dictionary with the different group labels.

```
by_state.groups["PA"]
```

```
Index([    4,    19,    21,    27,    38,    57,    69,    76,    84,    88,
       ...
       11838, 11862, 11871, 11873, 11883, 11887, 11926, 11938, 11952, 11965],
      dtype='int64', length=1053)
```

12.3 get group

You can also use `.get_group()` as a way to drill down to the sub-table from a single group:

```
by_state.get_group("PA")
```

	last_name	first_name	birthday	gender	type	state	party
4	Clymer	George	1739-03-16	M	rep	PA	NaN
19	Maclay	William	1737-07-20	M	sen	PA	Anti-Administration
21	Morris	Robert	1734-01-20	M	sen	PA	Pro-Administration
27	Wynkoop	Henry	1737-03-02	M	rep	PA	NaN
38	Jacobs	Israel	1726-06-09	M	rep	PA	NaN
...	...	...	...	...	...	...	...
11887	Brady	Robert	1945-04-07	M	rep	PA	Democrat
11926	Shuster	Bill	1961-01-10	M	rep	PA	Republican
11938	Rothfus	Keith	1962-04-25	M	rep	PA	Republican
11952	Costello	Ryan	1976-09-07	M	rep	PA	Republican
11965	Marino	Tom	1952-08-15	M	rep	PA	Republican

This is virtually equivalent to using `.loc[]`. You could get the same output with something like `df.loc[df["state"] == "PA"]`.

13 Multiple hypothesis correction

Bonferroni correction for multiple hypotheses

13.1 Bonferroni correction for multiple hypotheses

The Bonferroni correction is a multiple-comparison correction used when several dependent or independent statistical tests are being performed simultaneously (since while a given alpha value alpha may be appropriate for each individual comparison, it is not for the set of all comparisons). In order to avoid a lot of spurious positives, the alpha value needs to be lowered to account for the number of comparisons being performed.

In multiple hypothesis testing there are two kinds of errors that must be to considered: 1. Type I error: The rejection of a true null hypothesis (also known as a "false positive" finding or conclusion; example: "an innocent person is convicted") 2. Type II error (False negative): The non-rejection of a false null hypothesis (also known as a "false negative" finding or conclusion; example: "a guilty person is not convicted"

https://mathworld.wolfram.com/BonferroniCorrection.html

13.1.1 Example from gene editing

Background: A researcher analyzes thousands of genese to identify "differentially expressed genes" between two groups (e.g., normal vs. treated), which could alter biological mechanisms of a species in response to a particular treatment.

If we analyze 10,000 results with a signifance level of 0.05, then we should expect hundreds of false positives.

To control false discoveries from multiple hypothesis testing, it is imperative to adjust the significance level ( ) to reduce the probability of getting Type I error.

https://www.reneshbedre.com/blog/multiple-hypothesis-testing-corrections.html

Import packages

```
import numpy as np
from statsmodels.stats.multitest import multipletests
```

Generate 500 random p-values

```
rand_num = np.random.random(500)
```

Set alpha value to 0.05

```
# alpha value
alpha = 0.05
```

What would happen if we didn't correct alpha value?

```
# without correction, how many times would we reject the null hypothesis?
print(len(rand_num[np.where(rand_num<alpha)]))
```

```
25
```

With Bonferroni correction

```
p_adjusted = multipletests(pvals=rand_num, alpha=alpha, method='bonferroni')
print(len(p_adjusted[1][np.where(p_adjusted[1]<alpha)]))
```

```
0
```

13.2 Benjamini-Hochberg correction for multiple hypotheses

```
p_adjusted = multipletests(pvals=rand_num, alpha=alpha, method='fdr_bh')
print(len(p_adjusted[1][np.where(p_adjusted[1]<alpha)]))
```

```
0
```

14 Logistic regression

In statistics, the logistic model (or logit model) is used to model the probability of a certain class or event existing such as pass/fail, win/lose, alive/dead or healthy/sick. This can be extended to model several classes of events such as determining whether an image contains a cat, dog, lion, etc. Each object being detected in the image would be assigned a probability between 0 and 1, with a sum of one.

https://en.wikipedia.org/wiki/Logistic_regression

14.1 What's the difference between linear regression and logistic regression?

Logistic regression analysis is used to examine the association of (categorical or continuous) independent variable(s) with one dichotomous dependent variable. This is in contrast to linear regression analysis in which the dependent variable is a continuous variable.

https://www.javatpoint.com/linear-regression-vs-logistic-regression-in-machine-learning

14.2 Logistic regression in Python

https://realpython.com/logistic-regression-python/

https://towardsdatascience.com/building-a-logistic-regression-in-python-step-by-step-becd4d56c9c8

14.3 Logistic regression: simple example

Import packages

```
import numpy as np
from sklearn.linear_model import LogisticRegression
from scipy.special import expit
```

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
import seaborn as sns
```

Make up data

```
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 0, 0, 0, 1, 1, 1, 1, 1, 1])
```

```
x
```

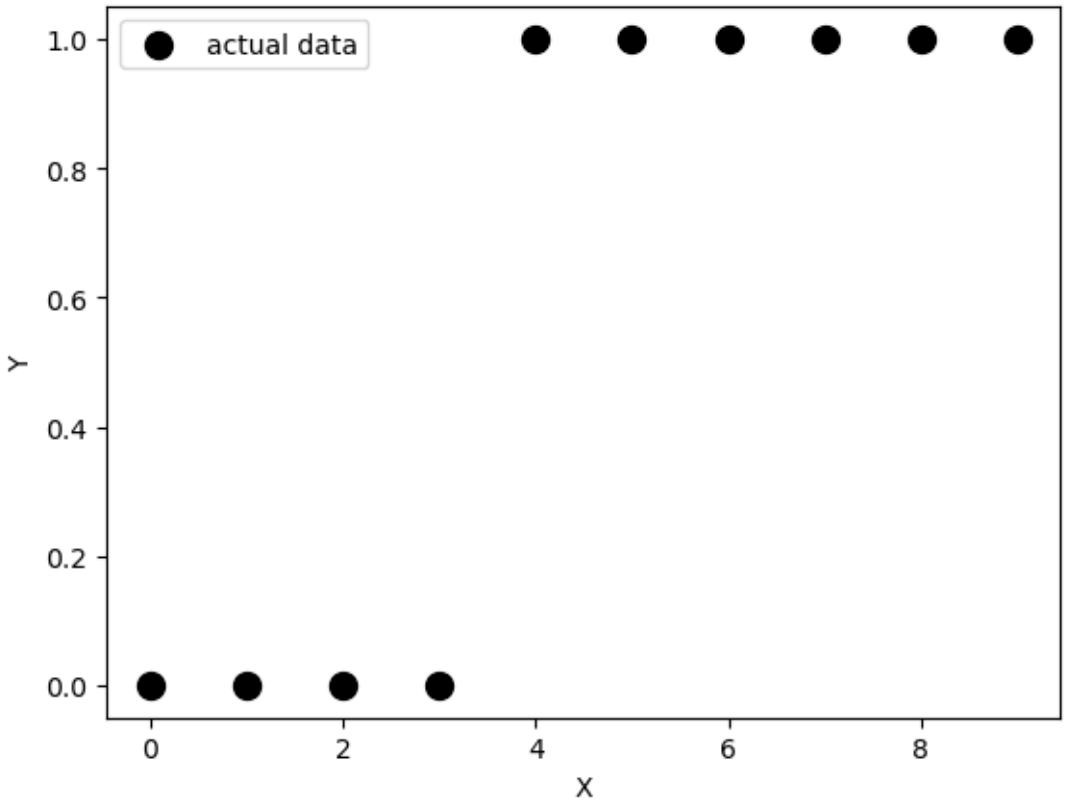
```
array([[0],
       [1],
       [2],
       [3],
       [4],
       [5],
       [6],
       [7],
       [8],
       [9]])
```

Note: we use reshape on `x` because when using the `LogisticRegression` function the x array must be two-dimensional.

Using `reshape()` with the arguments -1, 1 gives us as many rows as needed and one column.

```
fig, ax = plt.subplots()
ax.scatter(x, y,
           color='black',
           s=100,
           label="actual data")
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x128e72bd0>
```



Create the model

```
model = LogisticRegression(solver='liblinear')
```

Train the model

```
model.fit(x, y)
```

```
LogisticRegression(solver='liblinear')
```

Alternatively, we can create and fit the model in just one step

```
model = LogisticRegression(solver='liblinear', random_state=0).fit(x, y)
```

Our potential y-values... not very surprising

```
model.classes_
```

```
array([0, 1])
```

The model's intercept

```
model.intercept_
```

```
array([-1.04608067])
```

The model's coefficient

```
model.coef_
```

```
array([[0.51491375]])
```

Evaluate the model

```
model.predict_proba(x)
```

```
array([[0.74002157, 0.25997843],
       [0.62975524, 0.37024476],
       [0.5040632 , 0.4959368 ],
       [0.37785549, 0.62214451],
       [0.26628093, 0.73371907],
       [0.17821501, 0.82178499],
       [0.11472079, 0.88527921],
       [0.07186982, 0.92813018],
       [0.04422513, 0.95577487],
       [0.02690569, 0.97309431]])
```

This returns the matrix of probabilities that the predicted output is equal to zero or one. The first column is the probability of the predicted output being zero, that is 1 - ( ). The second column is the probability that the output is one, or ( ).

You can get the actual predictions, based on the probability matrix and the values of ( ), with .predict(). This function returns the predicted output values as a one-dimensional array.

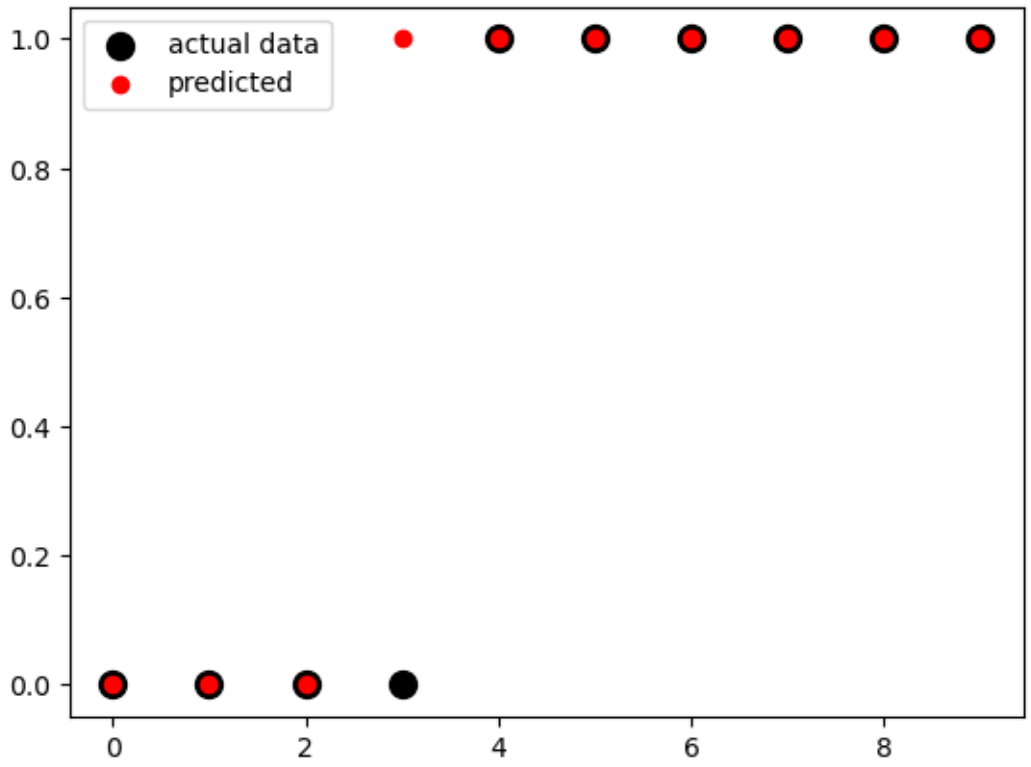
```
model.predict(x)
```

```
array([0, 0, 0, 1, 1, 1, 1, 1, 1, 1])
```

Plot the results

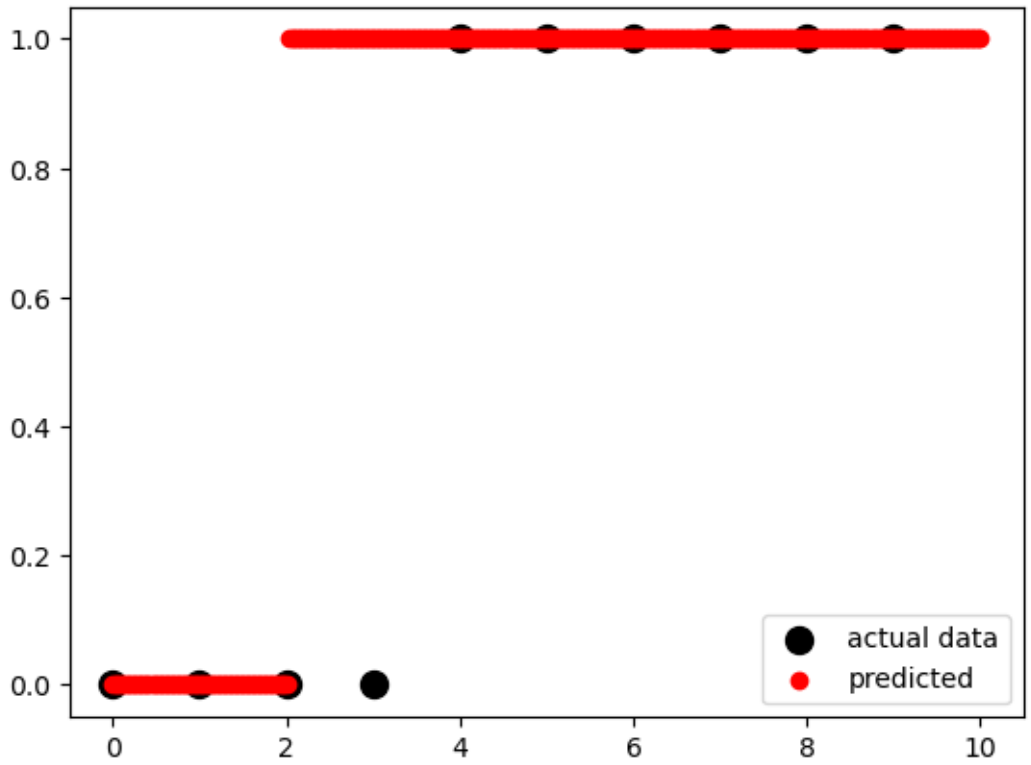
```
fig, ax = plt.subplots()
ax.scatter(x, y,
           color='black',
           s=100,
           label="actual data")
ax.scatter(x, model.predict(x),
           color='red',
           label="predicted")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x12b0c6010>
```



```
x_test = np.linspace(0, 10, 300).reshape(-1, 1)

fig, ax = plt.subplots()
ax.scatter(x, y,
           color='black',
           s=100,
           label="actual data")
ax.scatter(x_test,
           model.predict(x_test),
           color='red',
           label="predicted")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x12b0b7c10>
```

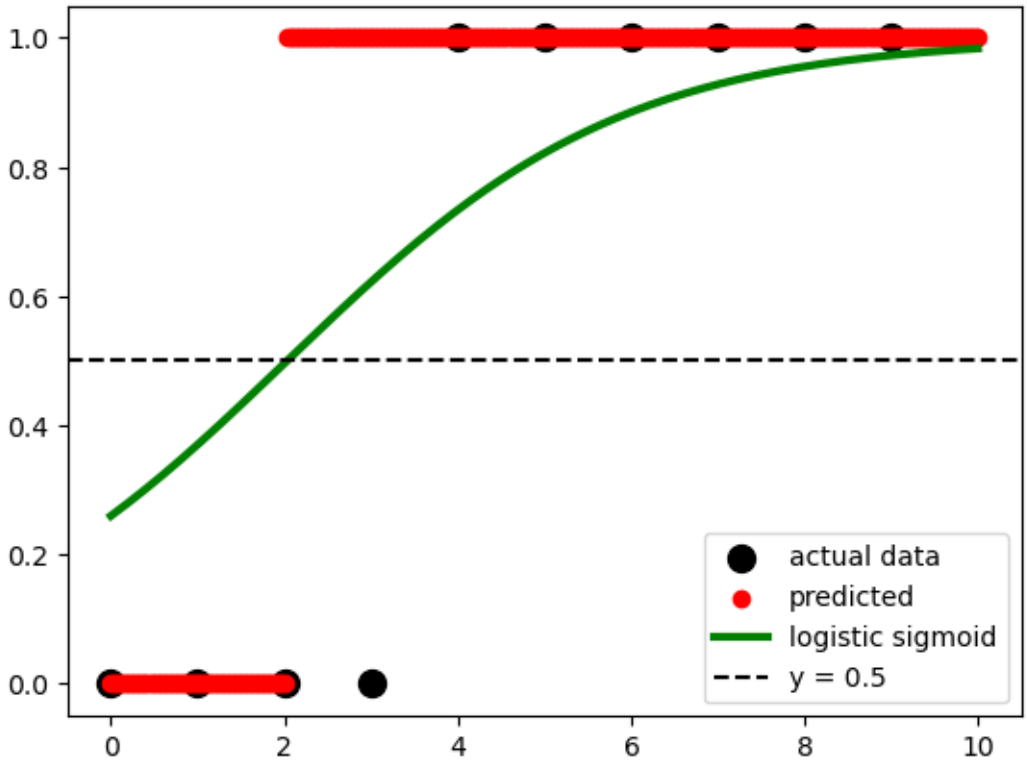


According to the logistic function:
$p = \frac{1}{1+e^{-(a+bx)}}$ * `expit()` This is a function from the SciPy library, specifically from

the scipy.special module. It calculates the sigmoid function, which is defined as `1 / (1 + exp(-x))`.
* `.ravel()` This function is called on the result of `expit()` to flatten or reshape the output as a 1-dimensional array. It converts a potentially multi-dimensional array into a contiguous flattened array.

```
sigmoid = expit(x_test * model.coef_ + model.intercept_).ravel()

fig, ax = plt.subplots()
ax.scatter(x, y,
           color='black',
           s=100,
           label="actual data")
ax.scatter(x_test,
           model.predict(x_test),
           color='red',
           label="predicted")
ax.plot(x_test,
        sigmoid,
        color='green',
        linewidth=3,
        label='logistic sigmoid')
ax.axhline(y=0.5,
           color='black',
           ls='--',
           label='y = 0.5')
ax.legend()
```

```
<matplotlib.legend.Legend at 0x12b192990>
```



Get the model score

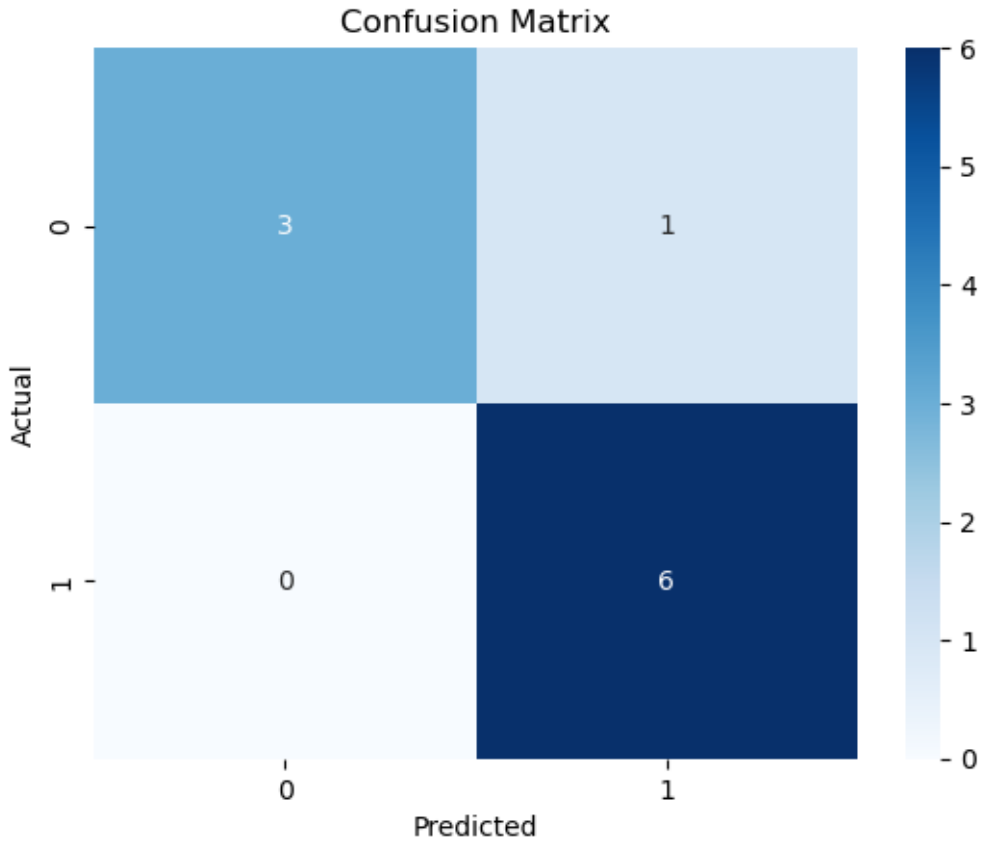
```
model.score(x, y)
```

```
0.9
```

```
# get the predicted values
y_pred = model.predict(x)

# create the confusion matrix
cm = confusion_matrix(y, y_pred)

# plot the confusion matrix using fig, ax method
fig, ax = plt.subplots()

sns.heatmap(cm, annot=True, cmap='Blues', ax=ax)
ax.set_xlabel('Predicted')
```

```
ax.set_ylabel('Actual')
ax.set_title('Confusion Matrix')
```

```
Text(0.5, 1.0, 'Confusion Matrix')
```



.score() takes the input and output as arguments and returns the ratio of the number of correct predictions to the number of observations.

14.3.1 We can also use the `StatsModels` packages, which provides some more statistical details

```
# import packages
import statsmodels.api as sm

# create data
x = np.arange(10).reshape(-1, 1)
y = np.array([0, 1, 0, 0, 1, 1, 1, 1, 1, 1])
x = sm.add_constant(x)

# create model
model = sm.Logit(y, x)

# fit model
result = model.fit()

# get results
result.params
```

```
Optimization terminated successfully.
         Current function value: 0.350471
         Iterations 7
```

```
array([-1.972805   ,  0.82240094])
```

```
print(result.summary())
```

```
                           Logit Regression Results
==============================================================================
Dep. Variable:                      y   No. Observations:                   10
Model:                          Logit   Df Residuals:                        8
Method:                           MLE   Df Model:                            1
Date:                Mon, 22 Jul 2024   Pseudo R-squ.:                  0.4263
Time:                        09:07:51   Log-Likelihood:                -3.5047
converged:                       True   LL-Null:                       -6.1086
Covariance Type:            nonrobust   LLR p-value:                   0.02248
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
const          -1.9728      1.737     -1.136      0.256      -5.377       1.431
x1              0.8224      0.528      1.557      0.119      -0.213       1.858
==============================================================================
```

15 Multiple linear regression

15.1 Multiple Linear Regression

Multiple Linear Regression Multiple or multivariate linear regression is a case of linear regression with two or more independent variables.

If there are just two independent variables, the estimated regression function is (,) = + + . It represents a regression plane in a three-dimensional space. The goal of regression is to determine the values of the weights , , and such that this plane is as close as possible to the actual responses and yield the minimal SSR.

The case of more than two independent variables is similar, but more general. The estimated regression function is (, …,) = + + + , and there are + 1 weights to be determined when the number of inputs is .

https://realpython.com/linear-regression-in-python/#multiple-linear-regression

https://datatofish.com/multiple-linear-regression-python/

Required packages

```
import pandas as pd
from sklearn import linear_model
import statsmodels.api as sm
import seaborn as sns
```

We will use the birthweight dataset again

```
# import dataset
file = "./birthweight.csv"
df = pd.read_csv(file)
df.head()
```

	Length	Birthweight	Headcirc	Gestation	smoker	mage	mnocig	mheight	mppwt	fage	fedyrs
0	56	4.55	34	44	0	20	0	162	57	23	10
1	53	4.32	36	40	0	19	0	171	62	19	12
2	58	4.10	39	41	0	35	0	172	58	31	16

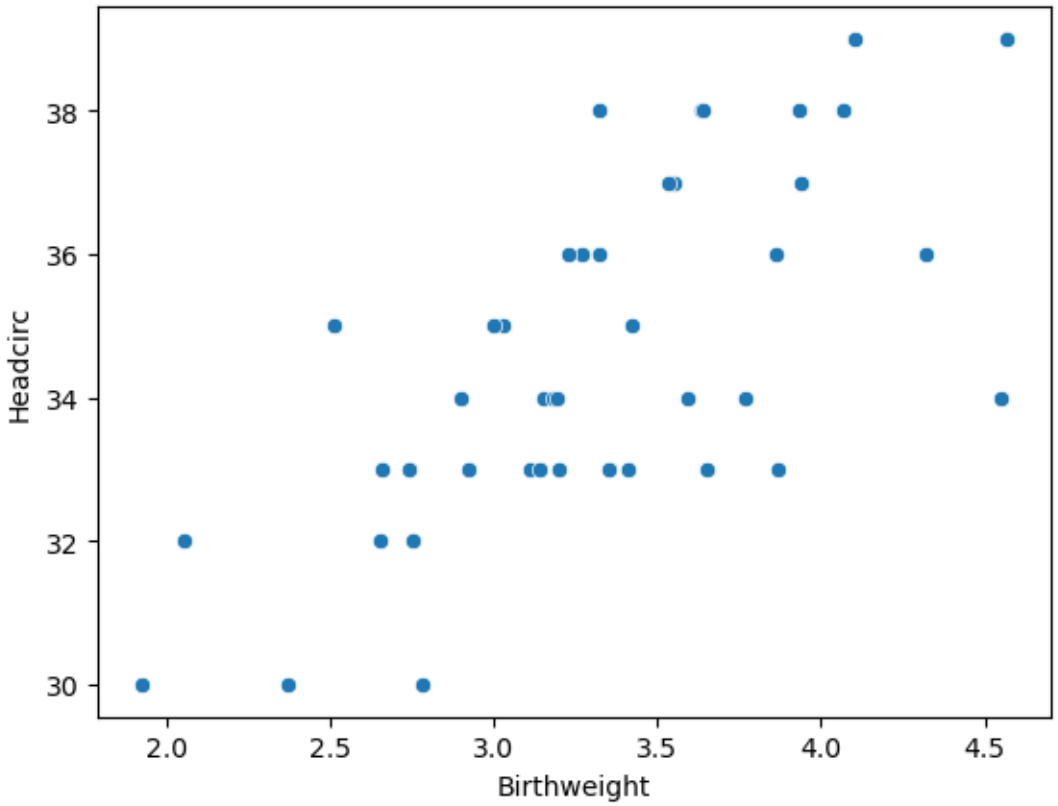
	Length	Birthweight	Headcirc	Gestation	smoker	mage	mnocig	mheight	mppwt	fage	fedyrs
3	53	4.07	38	44	0	20	0	174	68	26	14
4	54	3.94	37	42	0	24	0	175	66	30	12

15.1.1 Refresher

Last time we examined the relationship between head circumfrence and birthweight

```
sns.scatterplot(x=df.Birthweight, y=df.Headcirc)
```

```
<Axes: xlabel='Birthweight', ylabel='Headcirc'>
```




```
X = sm.add_constant(df.Birthweight)
Y = df.Headcirc
model = sm.OLS(Y, X)
```

```
results = model.fit()
print(results.summary())
```

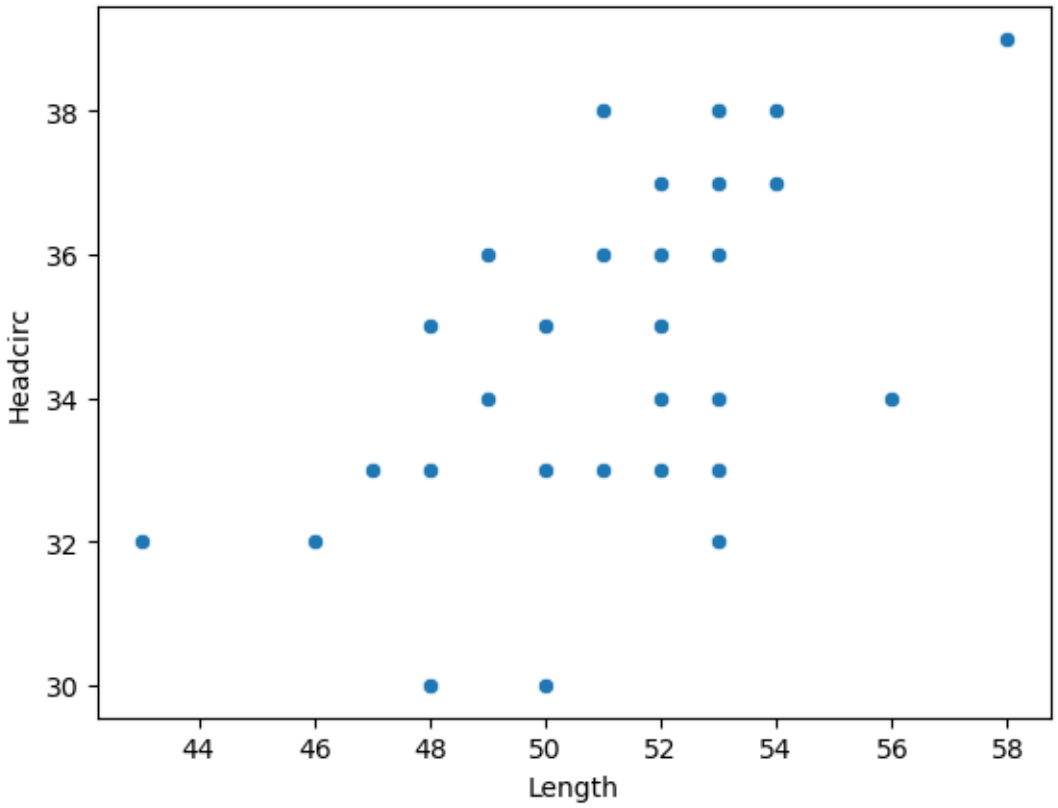
```
                            OLS Regression Results
==============================================================================
Dep. Variable:               Headcirc   R-squared:                       0.469
Model:                           OLS   Adj. R-squared:                  0.455
Method:                Least Squares   F-statistic:                     35.29
Date:               Mon, 22 Jul 2024   Prob (F-statistic):           5.73e-07
Time:                       09:09:45   Log-Likelihood:                -82.574
No. Observations:                 42   AIC:                             169.1
Df Residuals:                     40   BIC:                             172.6
Df Model:                          1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const          25.5824      1.542     16.594      0.000      22.467      28.698
Birthweight     2.7206      0.458      5.940      0.000       1.795       3.646
==============================================================================
Omnibus:                        1.089   Durbin-Watson:                   2.132
Prob(Omnibus):                  0.580   Jarque-Bera (JB):                1.007
Skew:                          -0.193   Prob(JB):                        0.604
Kurtosis:                       2.347   Cond. No.                         20.6
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

15.2 Now, let's add another predicting variable

What's the relationship between length and head circumfrence?

```
sns.scatterplot(x=df.Length, y=df.Headcirc)
```

```
<Axes: xlabel='Length', ylabel='Headcirc'>
```



Now we can combine our two 'predictor' variables

```
# group x variables
X = df[['Birthweight', 'Length']]

# y variable
Y = df['Headcirc']
```

```
# using statsmodels
X = sm.add_constant(X)
model = sm.OLS(Y, X).fit()
predictions = model.predict(X)
print_model = model.summary()
print(print_model)
```

```
                            OLS Regression Results
==============================================================================
```

```
Dep. Variable:                Headcirc   R-squared:                       0.478
Model:                             OLS   Adj. R-squared:                  0.451
Method:                  Least Squares   F-statistic:                     17.84
Date:                 Mon, 22 Jul 2024   Prob (F-statistic):           3.14e-06
Time:                         09:10:16   Log-Likelihood:                -82.211
No. Observations:                   42   AIC:                             170.4
Df Residuals:                       39   BIC:                             175.6
Df Model:                            2
Covariance Type:             nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         21.0794      5.673      3.716      0.001       9.605      32.554
Birthweight    2.3191      0.670      3.464      0.001       0.965       3.673
Length         0.1136      0.138      0.825      0.414      -0.165       0.392
==============================================================================
Omnibus:                        1.252   Durbin-Watson:                   2.115
Prob(Omnibus):                  0.535   Jarque-Bera (JB):                1.122
Skew:                          -0.224   Prob(JB):                        0.571
Kurtosis:                       2.337   Cond. No.                     1.07e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 1.07e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

```
# with sklearn
regr = linear_model.LinearRegression()
regr.fit(X, Y)

print('Intercept: ', regr.intercept_)
print('Coefficients: ', regr.coef_)
```

```
Intercept:  21.079365879118836
Coefficients:  [0.         2.319074   0.11363204]
```

So, which package should we use?

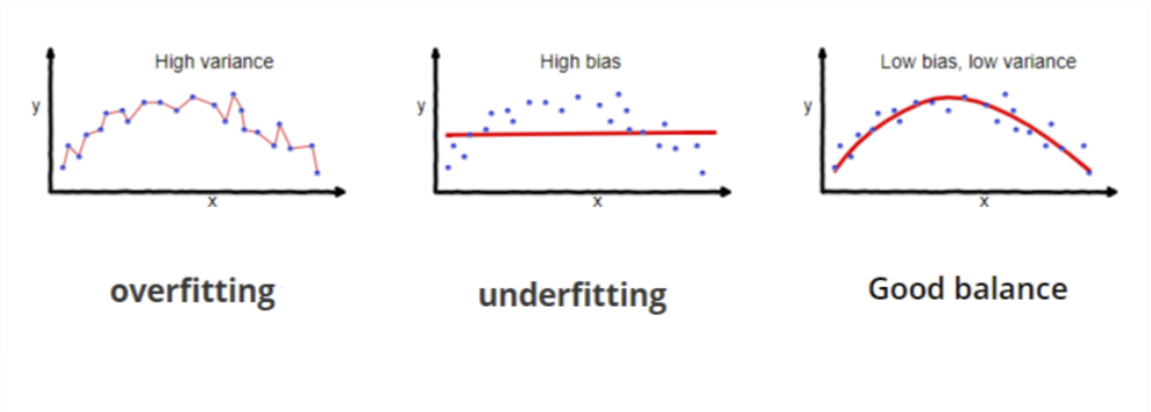
In general, scikit-learn is designed for machine-learning, while statsmodels is made for rigorous statistics.

https://medium.com/(**hsrinivasan2/linear-regression-in-scikit-learn-vs-statsmodels-568b60792991?**)

16 Regularization

16.1 Why is it important?

```
%matplotlib inline
from IPython.display import Image
Image('image_1.png')
```



16.1.0.1 In simple terms

A technique that is used to limit model overfitting by shrinking the coefficient estimates towards zero. Our goal is optimized prediction and not inference.

16.1.0.2 In more exact terms

Regularized least squares (RLS) is a family of methods for solving the least-squares problem while using regularization to further constrain the resulting solution.

RLS is used for two main reasons. The first comes up when the number of variables in the linear system exceeds the number of observations. In such settings, the ordinary least-squares

problem is ill-posed and is therefore impossible to fit because the associated optimization problem has infinitely many solutions. RLS allows the introduction of further constraints that uniquely determine the solution.

The second reason that RLS is used occurs when the number of variables does not exceed the number of observations, but the learned model suffers from poor generalization. RLS can be used in such cases to improve the generalizability of the model by constraining it at training time. This constraint can either force the solution to be "sparse" in some way or to reflect other prior knowledge about the problem such as information about correlations between features. A Bayesian understanding of this can be reached by showing that RLS methods are often equivalent to priors on the solution to the least-squares problem.

https://en.wikipedia.org/wiki/Regularized_least_squares

https://scikit-learn.org/stable/modules/linear_model.html

16.1.1 We will focus on two methods of regularized regression

16.1.1.1 (1) Ridge

- Performs L2 regularization, i.e. adds penalty equivalent to square of the magnitude of coefficients
- Minimization objective = LS Obj + * (sum of square of coefficients)

https://en.wikipedia.org/wiki/Ridge_regression

16.1.1.2 (2) Lasso

https://en.wikipedia.org/wiki/Lasso_(statistics)

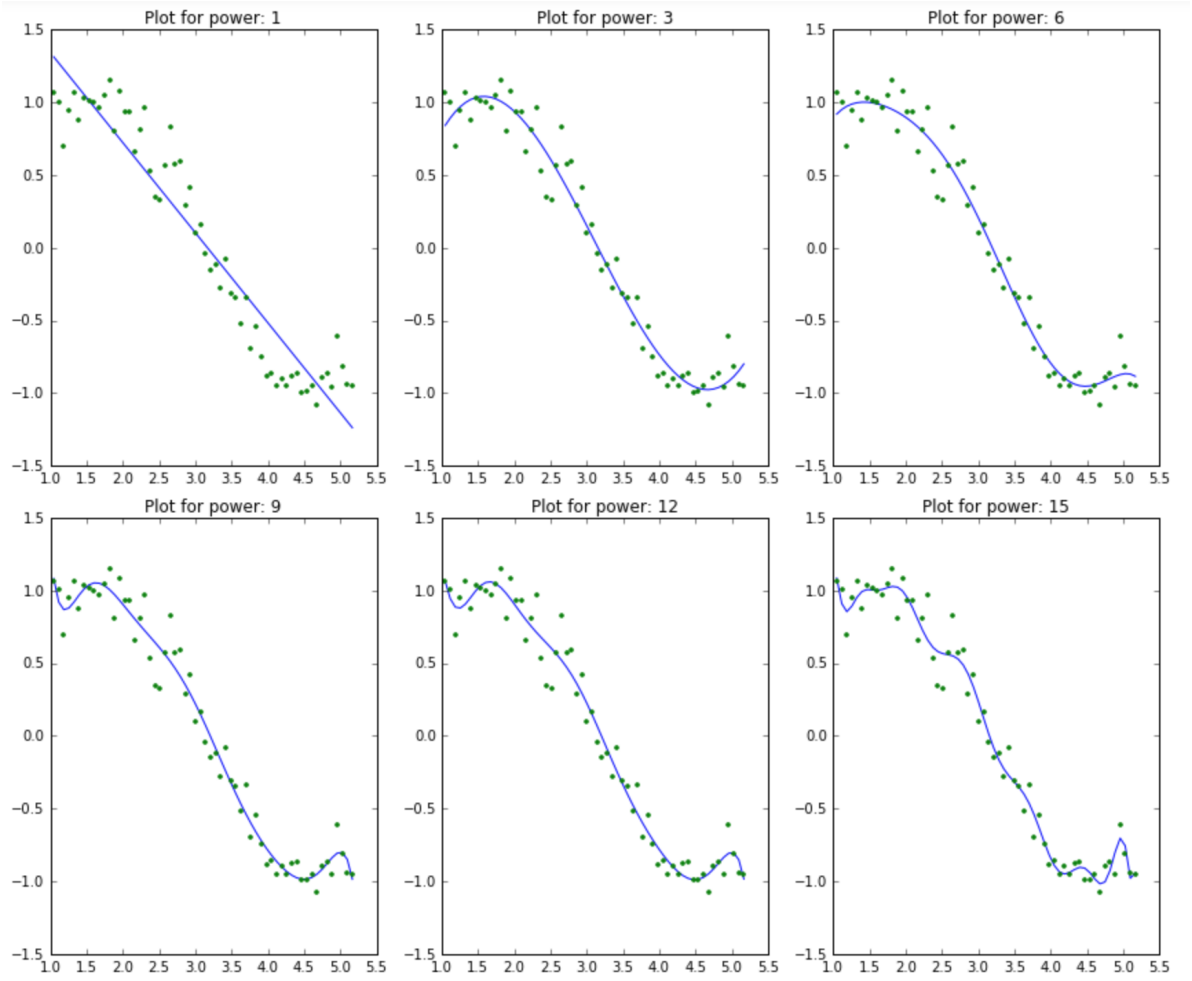
- Performs L1 regularization, i.e. adds penalty equivalent to absolute value of the magnitude of coefficients
- Minimization objective = LS Obj + * (sum of absolute value of coefficients)

16.1.2 Some notes on usage

For both ridge and lasso you have to set a so-called "meta-parameter" that defines how aggressive regularization is performed. Meta-parameters are usually chosen by cross-validation. For Ridge regression the meta-parameter is often called "alpha" or "L2"; it simply defines regularization strength. For LASSO the meta-parameter is often called "lambda", or "L1". In contrast to Ridge, the LASSO regularization will actually set less-important predictors to 0 and help you with choosing the predictors that can be left out of the model.

16.1.3 What happens when we use more and more predictors?

```
Image('image_2.png')
```



16.2 Example

We will use data from the boston house prediction dataset.

In this dataset, each row describes a Boston town or suburb.

There are 506 rows and 13 attributes (features) with a target column (price).

```
# import libraries
import pandas as pd
import numpy as np
from sklearn import datasets
from sklearn.linear_model import LinearRegression, Ridge, Lasso
from sklearn.model_selection import train_test_split
import seaborn as sns
from matplotlib import pyplot as plt
```

```
# # load dataset
# boston_dataset = datasets.fetch_california_housing()
# # boston_dataset = datasets.load_diabetes()
# print(boston_dataset.DESCR)

# import dataset
file = "./BostonHousing.csv"
data = pd.read_csv(file)
data.head()
```

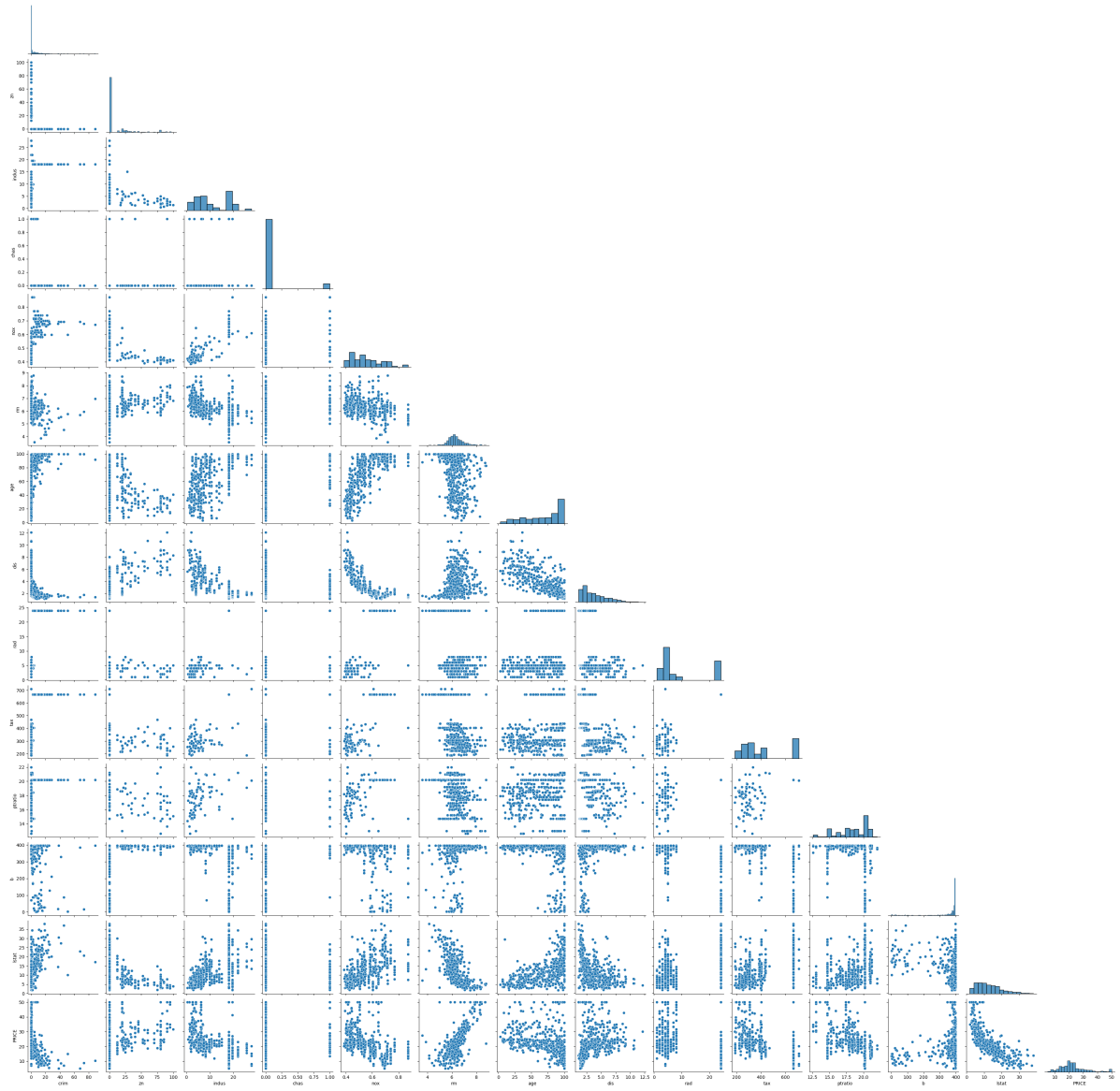
	crim	zn	indus	chas	nox	rm	age	dis	rad	tax	ptratio	b	lstat	PRICE
0	0.00632	18.0	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0.0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0.0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0.0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0.0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
# Check for missing values
data.isnull().sum()
```

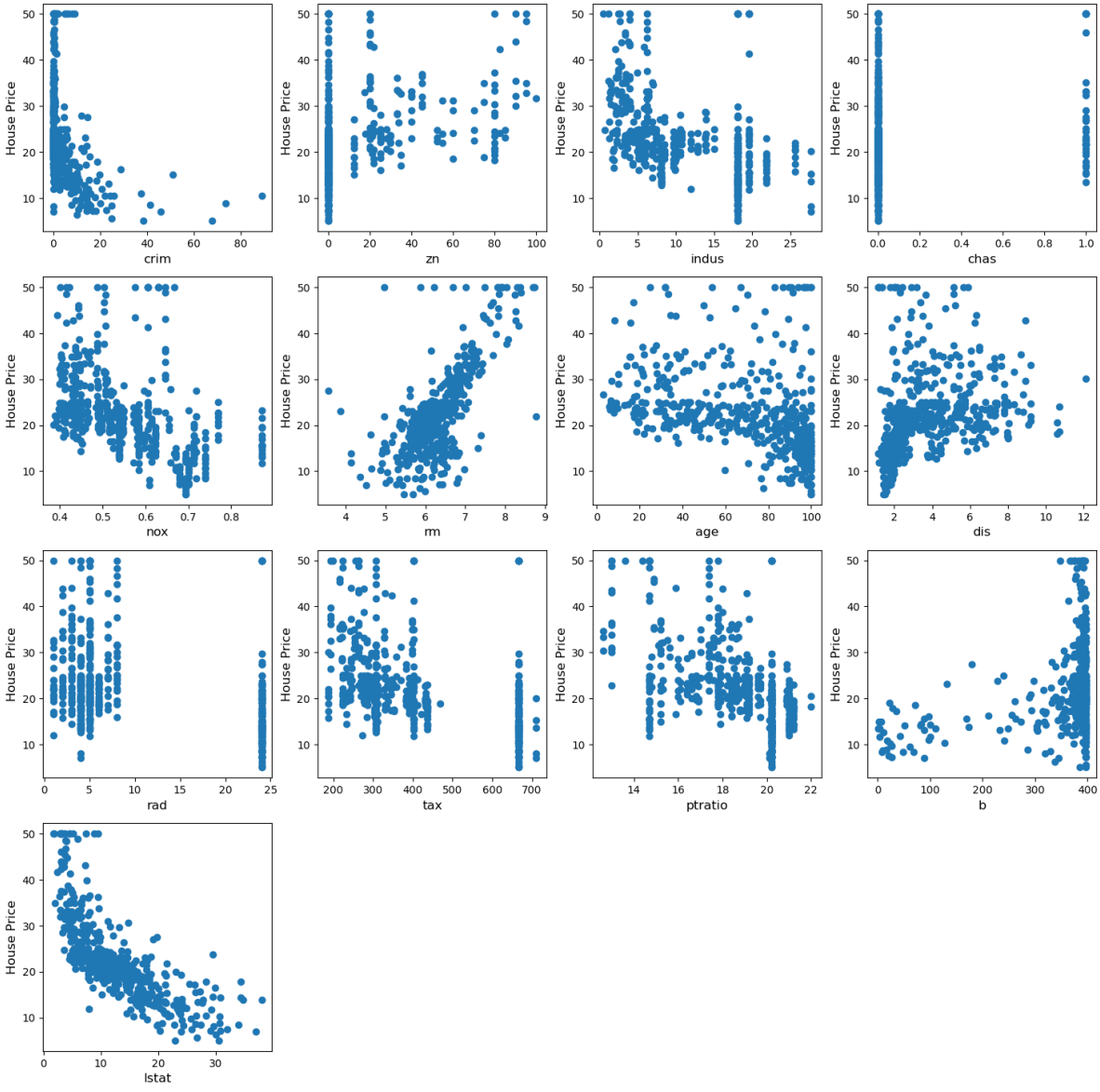
```
crim       0
zn         0
indus      0
chas       0
nox        0
rm         0
age        0
dis        0
rad        0
tax        0
ptratio    0
```

```
b           0
lstat       0
PRICE       0
dtype: int64
```

```
# take a look at the data
sns.pairplot(data, corner = "True")
```



```
fig = plt.figure(figsize = (18, 18))

for ii, feature_name in enumerate(data.columns[:-1]):
    ax = fig.add_subplot(4, 4, ii + 1)
    ax.scatter(data.iloc[:, ii], data.iloc[:,-1])
    ax.set_ylabel('House Price', size = 12)
    ax.set_xlabel(feature_name, size = 12)
```



16.2.0.1 We can observe from the above scatter plots that some of the independent variables are highly correlated (either positively or negatively) with the target variable. What will happen to these variables when we perform the regularization?

16.3 Start the regression :)

16.3.1 Super important!!! standardize the data before applying regularization

```
# standardize the data by computing the z score
stand_data = data.apply(lambda x:(x-x.mean())/x.std(), axis=0)

# input
X = stand_data.iloc[:, :-1]

#output
Y = stand_data.iloc[:, -1]
```

16.3.1.1 Split the data into test and training groups

```
# split the data into training and testing groups

(x_train,
 x_test,
 y_train,
 y_test) = train_test_split(X, Y, test_size = 0.25)

print("Train data shape of X = % s and Y = % s : "%(
    x_train.shape, y_train.shape))

print("Test data shape of X = % s and Y = % s : "%(
    x_test.shape, y_test.shape))
```

```
Train data shape of X = (379, 13) and Y = (379,) :
Test data shape of X = (127, 13) and Y = (127,) :
```

16.4 Multiple Linear Regression

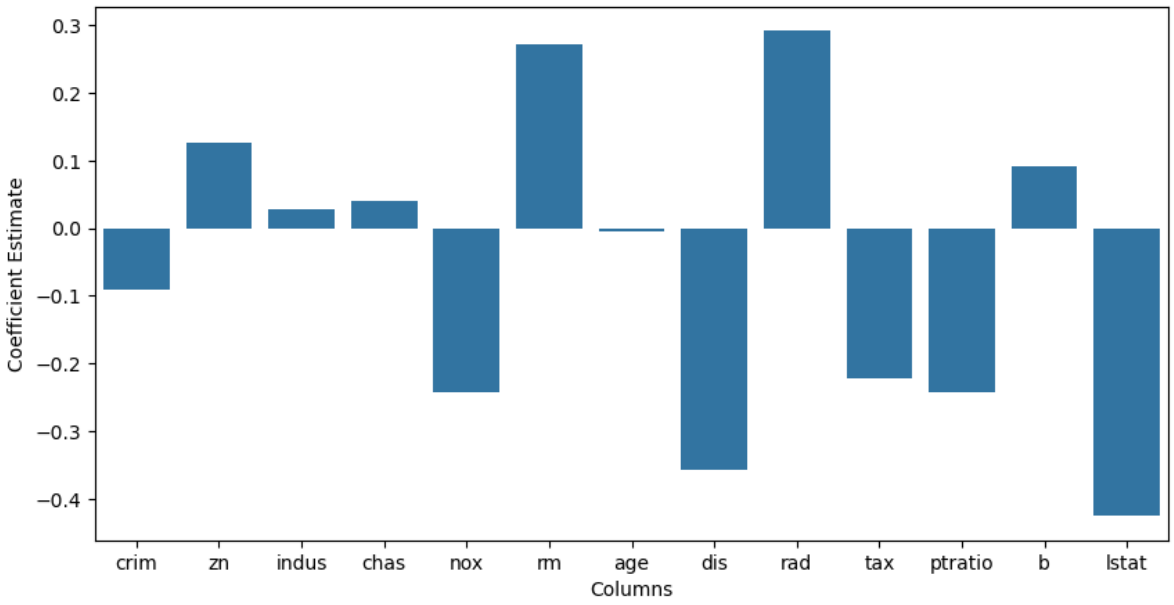
First, let's try multiple linear regression, so that we have a point of comparison

```
# Apply multiple Linear Regression Model
lreg = LinearRegression()
lreg.fit(x_train, y_train)

# Generate Prediction on test set
lreg_y_pred = lreg.predict(x_test)

# calculating Mean Squared Error (mse)
mean_squared_error = np.mean((lreg_y_pred - y_test)**2)
print("Mean squared Error on test set : ", mean_squared_error)

# Putting together the coefficients and their corresponding variable names
lreg_coefficient = pd.DataFrame()
lreg_coefficient["Columns"] = x_train.columns
lreg_coefficient['Coefficient Estimate'] = pd.Series(lreg.coef_)
lreg_coefficient['Type'] = 'Linear (MSE: '+ str(np.around(mean_squared_error,2))+')'
```

```
Mean squared Error on test set :  0.16355502227534496
```

```
# plotting the coefficient score
fig, ax = plt.subplots(figsize =(10, 5))

sns.barplot(x = lreg_coefficient["Columns"],
            y = lreg_coefficient['Coefficient Estimate'],
            hue = None)
```

```
<Axes: xlabel='Columns', ylabel='Coefficient Estimate'>
```



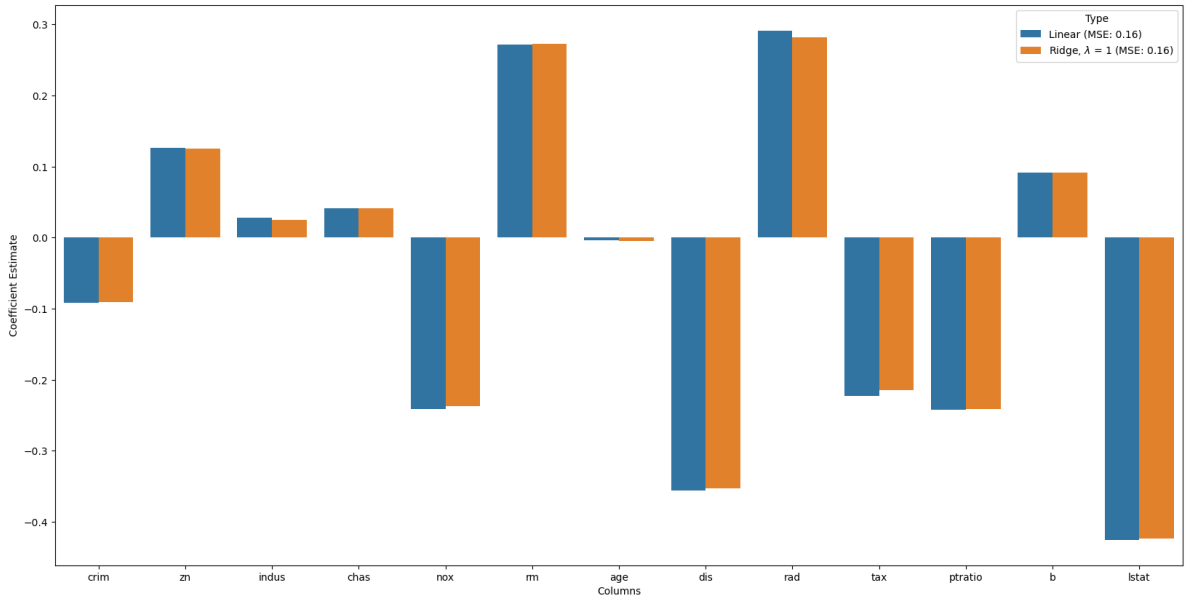
16.4.1 Now, let's try Ridge regression

```
# Train the model
alpha = 1
ridgeR = Ridge(alpha = alpha)
ridgeR.fit(x_train, y_train)
y_pred = ridgeR.predict(x_test)

# calculate mean square error
mean_squared_error_ridge = np.mean((y_pred - y_test)**2)
print("Mean squared Error on test set : ", mean_squared_error_ridge)

# get ridge coefficient
ridge_coefficient = pd.DataFrame()
ridge_coefficient["Columns"]= x_train.columns
ridge_coefficient['Coefficient Estimate'] = pd.Series(ridgeR.coef_)
ridge_coefficient['Type'] = r'Ridge, $\lambda$ = '+ f'{alpha} (MSE: {str(np.around(mean_sq
```

```
Mean squared Error on test set :  0.16328472190858787
```

```
# merge dataframes
frames = [lreg_coefficient,
          ridge_coefficient]

all_coefs = pd.concat(frames)

# plotting the coefficient scores
fig, ax = plt.subplots(figsize =(20, 10))

sns.barplot(x = all_coefs["Columns"],
            y = all_coefs['Coefficient Estimate'],
            hue = all_coefs['Type'])
```

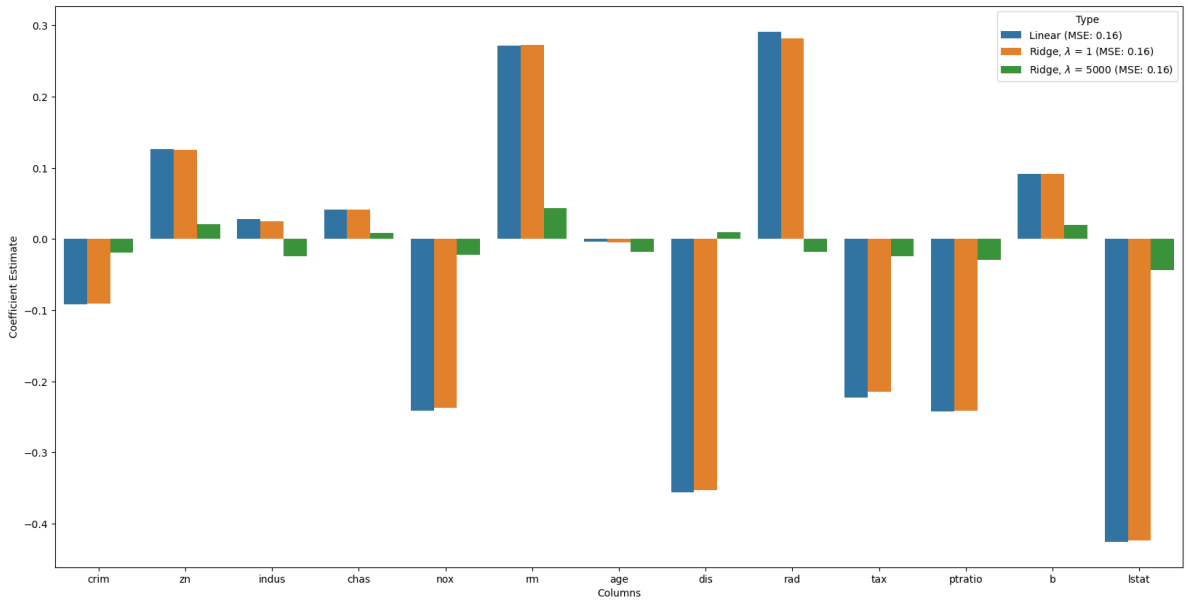
```
<Axes: xlabel='Columns', ylabel='Coefficient Estimate'>
```




```
# Train the model
alpha = 5000
ridgeR = Ridge(alpha = alpha)
ridgeR.fit(x_train, y_train)
y_pred = ridgeR.predict(x_test)

# calculate mean square error
```

```
mean_squared_error_ridge = np.mean((y_pred - y_test)**2)

# get ridge coefficient
ridge_coefficient_10 = pd.DataFrame()
ridge_coefficient_10["Columns"]= x_train.columns
ridge_coefficient_10['Coefficient Estimate'] = pd.Series(ridgeR.coef_)
# r'Ridge, $\lambda$ = '+ f'{alpha} (MSE: {str(np.around(mean_squared_error,2))})'
ridge_coefficient_10['Type'] = r'Ridge, $\lambda$ = '+ f'{alpha} (MSE: {str(np.around(mean

# merge dataframes
frames = [lreg_coefficient,
          ridge_coefficient,
          ridge_coefficient_10]

all_coefs = pd.concat(frames)

# plotting the coefficient scores
fig, ax = plt.subplots(figsize =(20, 10))

sns.barplot(x = all_coefs["Columns"],
            y = all_coefs['Coefficient Estimate'],
            hue = all_coefs['Type'])
```

```
<Axes: xlabel='Columns', ylabel='Coefficient Estimate'>
```

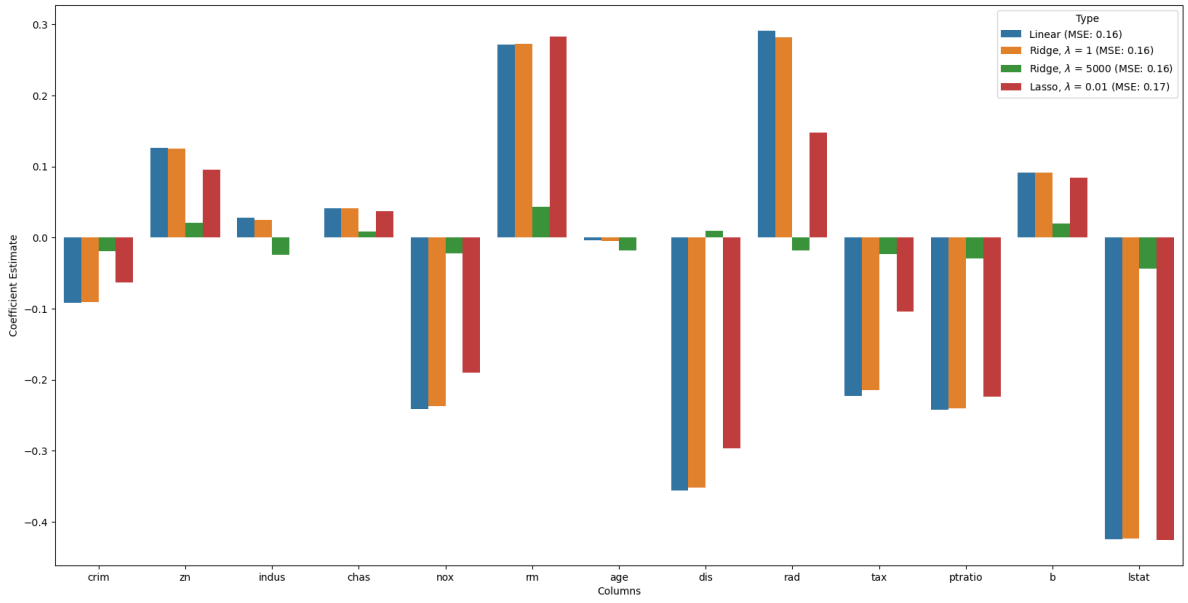


16.4.2 Now, let's try Lasso

```
# Train the model
alpha = 0.01
lasso = Lasso(alpha = alpha)
lasso.fit(x_train, y_train)
y_pred1 = lasso.predict(x_test)

# Calculate Mean Squared Error
mean_squared_error = np.mean((y_pred1 - y_test)**2)

# Put in dataframe
lasso_coeff = pd.DataFrame()
lasso_coeff["Columns"] = x_train.columns
lasso_coeff['Coefficient Estimate'] = pd.Series(lasso.coef_)
lasso_coeff['Type'] = r'Lasso, $\lambda$ = '+ f'{alpha} (MSE: {str(np.around(mean_squared_

# merge dataframes
frames = [lreg_coefficient,
          ridge_coefficient,
          ridge_coefficient_10,
          lasso_coeff]
```

```
all_coefs = pd.concat(frames)

# plotting the coefficient scores
fig, ax = plt.subplots(figsize =(20, 10))

sns.barplot(x = all_coefs["Columns"],
            y = all_coefs['Coefficient Estimate'],
            hue = all_coefs['Type'])
```

```
<Axes: xlabel='Columns', ylabel='Coefficient Estimate'>
```




```
# Train the model
alpha = 0.1
lasso = Lasso(alpha = alpha)
lasso.fit(x_train, y_train)
y_pred1 = lasso.predict(x_test)

# Calculate Mean Squared Error
mean_squared_error = np.mean((y_pred1 - y_test)**2)

# Put in dataframe
lasso_coeff_10 = pd.DataFrame()
```

```
lasso_coeff_10["Columns"] = x_train.columns
lasso_coeff_10['Coefficient Estimate'] = pd.Series(lasso.coef_)
lasso_coeff_10['Type'] = r'Lasso, $\lambda$ = '+ f'{alpha} (MSE: {str(np.around(mean_squar

# merge dataframes
frames = [lreg_coefficient,
          ridge_coefficient,
          ridge_coefficient_10,
          lasso_coeff,
          lasso_coeff_10]

all_coefs = pd.concat(frames)

# plotting the coefficient scores
fig, ax = plt.subplots(figsize =(20, 10))

sns.barplot(x = all_coefs["Columns"],
            y = all_coefs['Coefficient Estimate'],
            hue = all_coefs['Type'])
```

```
<Axes: xlabel='Columns', ylabel='Coefficient Estimate'>
```