Python/R for Data Science

Lecture notes for 2022 Fall at Arkansas Tech University

Xinli Xiao

2022-11-03T00:19:42-05:00

Table of contents

Preface

This is the lecture notes for STAT 2304 Programming languages for Data Science 2022 Fall at ATU. If you have any comments/suggetions/concers about the notes please contact me at my email xxiao@atu.edu.

Part I

Part I: Python

Part II

Preliminaries

1 Why Python?

1.1 Python is easy to learn and use

1.2 Python is easy to read

1.3 Python Community is mature and supportive

2 Hello world!

2.1 Setup the Python environment

In this section we are going to setup the Python developing environment.

2.1.1 VS Code + Anaconda

Click Appendix A to see the detailed steps for VS Code and Anaconda. You may also check out the official document. It contains more features but less details.

We will talk about the relation between Python and Anaconda and more about packages sometime later.

2.1.2 Google Colab

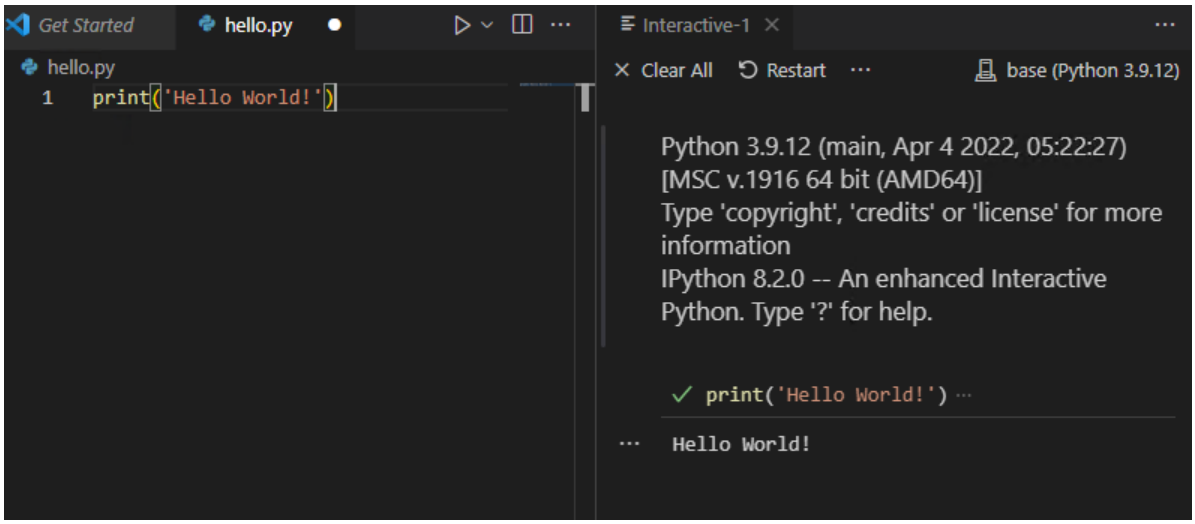
Click Appendix B for more details.

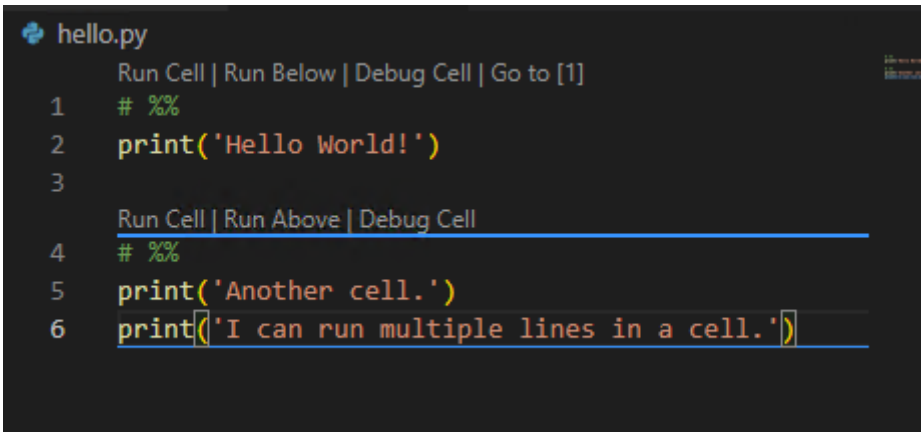
2.2 `Hello World!`

Take VS Code as an example. In the editor window, type in the code, and run the file in the interactive window.

```
print('Hello World!')
```

If you see a small green check mark in the interactive window and also the output `Hello World!`, you are good to go!

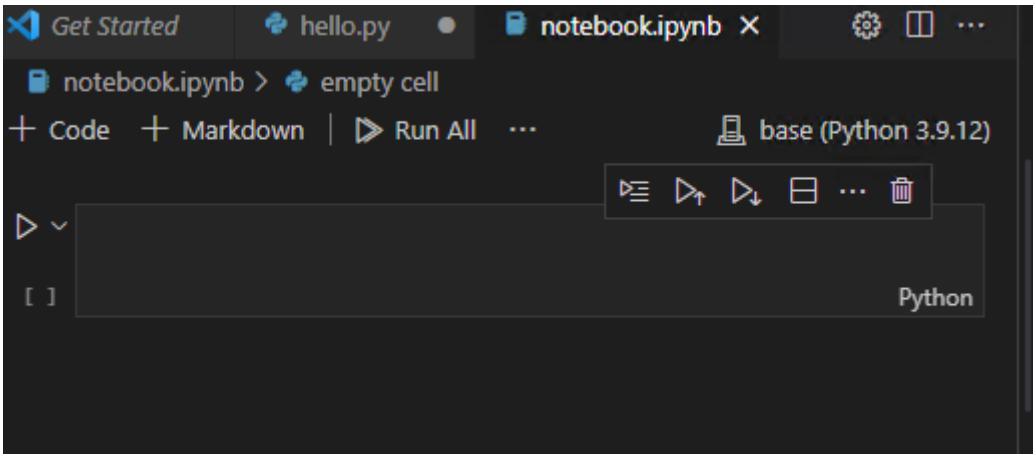

Get Started
hello.py
hello.py
1 print('Hello World!')
Interactive-1
Clear All
Restart
base (Python 3.9.12)
Python 3.9.12 (main, Apr 4 2022, 05:22:27)
[MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.2.0 -- An enhanced Interactive Python. Type '?' for help.
print('Hello World!')
Hello World!


hello.py
Run Cell | Run Below | Debug Cell | Go to [1]
1 # %%
2 print('Hello World!')
3
Run Cell | Run Above | Debug Cell
4 # %%
5 print('Another cell.')
6 print('I can run multiple lines in a cell.')

2.3 Python code cells and Notebooks

In VS Code you can run codes cell by cell. Each cell is separated by the symbol `# %%`. Each cell may contain multiple lines. You may click the small button on top of the cell or use keybindings.

This feature actually mimic the notebook. We may start a real Python Notebook file by directly creating a file with extension `.ipynb`.



The layout is straightforward.

2.4 Linters

2.5 IPython and Jupyter

3 Projects

Exercise 3.1 (Hello world!)**.** Please set up a Python developing environment, including for `.py` file and for notebook, that will be used across the semester. Then print `Hello World!`.

Exercise 3.2 (Define a function and play with `time`)**.** Please play with the following codes in a Jupyter notebook. We haven't talked about any of them right now. Try to guess what they do and write your guess in markdown cells.

```
import time

def multistr(x, n=2):
    return x * n

t0 = time.time()
x = 'Python'
print(multistr(x, n=10))
t1 = time.time()
print("Time used: ", t1-t0)
```

Exercise 3.3 (Fancy Basketball plot)**.** Here is an example of the data analysis. We pull data from a dataset, filter the data according to our needs and plot it to visualize the data. This is just a show case. You are encouraged to play the code, make tweaks and see what would happen. You don't have to turn in anything.

The data we choose is Stephen Curry's shots data in 2021-2022 regular season. First we need to load the data. The data is obtained from `nba.com` using `nba_api`.

```
from nba_api.stats.static import players
from nba_api.stats.endpoints import shotchartdetail
```

```
player_dict = players.get_players()
```

The shots data we need is in `shotchartdetail`. However to use it we need to know the id of `Stephen Curry` using the dataset `player_dict`.

```
for player in player_dict:
    if player['full_name'] == 'Stephen Curry':
        print(player['id'])
```

```
201939
```

So the id of `Stephen Curry` is `201939`. Let's pull out his shots data in 2021-2022 season.

```
results = shotchartdetail.ShotChartDetail(
            team_id = 0,
            player_id = 201939,
            context_measure_simple = 'FGA',
            season_nullable = '2021-22',
            season_type_all_star = 'Regular Season')
df = results.get_data_frames()[0]
df.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_ID
0	Shot Chart Detail	0022100002	26	201939	Stephen Curry	161061274
1	Shot Chart Detail	0022100002	34	201939	Stephen Curry	161061274
2	Shot Chart Detail	0022100002	37	201939	Stephen Curry	161061274
3	Shot Chart Detail	0022100002	75	201939	Stephen Curry	161061274
4	Shot Chart Detail	0022100002	130	201939	Stephen Curry	161061274

`df` is the results we get in terms of a `DataFrame`, and we show the first `5` records as an example.

These are all attempts. We are interested in all made. By looking at all the columns, we find a column called `SHOT_MADE_FLAG` which shows what we want. Therefore we will use it to filter the records.

```
df_made = df[df['SHOT_MADE_FLAG']==1]
df_made.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	GRID_TYPE	GAME_ID	GAME_EVENT_ID	PLAYER_ID	PLAYER_NAME	TEAM_I
2	Shot Chart Detail	0022100002	37	201939	Stephen Curry	16106127
6	Shot Chart Detail	0022100002	176	201939	Stephen Curry	16106127
9	Shot Chart Detail	0022100002	352	201939	Stephen Curry	16106127
16	Shot Chart Detail	0022100002	510	201939	Stephen Curry	16106127
18	Shot Chart Detail	0022100002	642	201939	Stephen Curry	16106127

We also notice that there are two columns `LOC_X` and `LOC_Y` shows the coordinates of the attempts. We will use it to draw the heatmap. The full code for drawing out the court `draw_court` is folded below. It is from Bradley Fay GitHub.

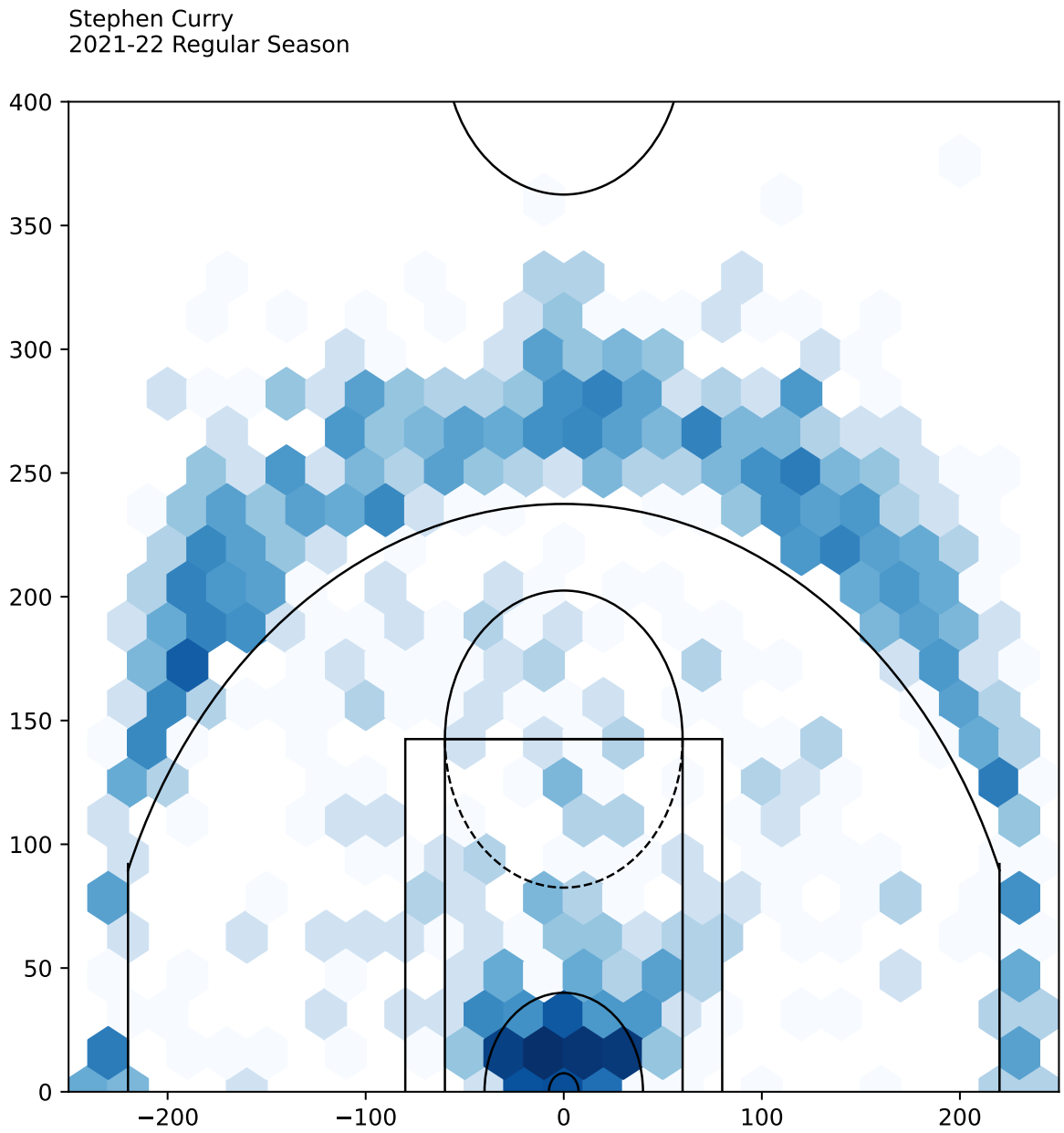
> **Note**
>
> Note that, although `draw_cort` is long, it is not hard to understand. It just draws a court piece by piece.

```
from matplotlib.patches import Circle, Rectangle, Arc
import matplotlib.pyplot as plt


def draw_court(ax=None, color='gray', lw=1, outer_lines=False):
    """
    Returns an axes with a basketball court drawn onto to it.

    This function draws a court based on the x and y-axis values that the NBA
    stats API provides for the shot chart data.  For example, the NBA stat API
    represents the center of the hoop at the (0,0) coordinate.  Twenty-two feet
    from the left of the center of the hoop in is represented by the (-220,0)
    coordinates.  So one foot equals +/-10 units on the x and y-axis.
    """
    if ax is None:
        ax = plt.gca()

    # Create the various parts of an NBA basketball court
```

```
# Create the basketball hoop
hoop = Circle((0, 0), radius=7.5, linewidth=lw, color=color, fill=False)

# Create backboard
backboard = Rectangle((-30, -7.5), 60, -1, linewidth=lw, color=color)

# The paint
# Create the outer box 0f the paint, width=16ft, height=19ft
outer_box = Rectangle((-80, -47.5), 160, 190, linewidth=lw, color=color,
                      fill=False)
# Create the inner box of the paint, widt=12ft, height=19ft
inner_box = Rectangle((-60, -47.5), 120, 190, linewidth=lw, color=color,
                      fill=False)

# Create free throw top arc
top_free_throw = Arc((0, 142.5), 120, 120, theta1=0, theta2=180,
                     linewidth=lw, color=color, fill=False)
# Create free throw bottom arc
bottom_free_throw = Arc((0, 142.5), 120, 120, theta1=180, theta2=0,
                        linewidth=lw, color=color, linestyle='dashed')
# Restricted Zone, it is an arc with 4ft radius from center of the hoop
restricted = Arc((0, 0), 80, 80, theta1=0, theta2=180, linewidth=lw,
                 color=color)

# Three point line
# Create the right side 3pt lines, it's 14ft long before it arcs
corner_three_a = Rectangle((-220, -47.5), 0, 140, linewidth=lw,
                           color=color)
# Create the right side 3pt lines, it's 14ft long before it arcs
corner_three_b = Rectangle((220, -47.5), 0, 140, linewidth=lw, color=color)
# 3pt arc - center of arc will be the hoop, arc is 23'9" away from hoop
three_arc = Arc((0, 0), 475, 475, theta1=22, theta2=158, linewidth=lw,
                color=color)

# Center Court
center_outer_arc = Arc((0, 422.5), 120, 120, theta1=180, theta2=0,
                       linewidth=lw, color=color)
center_inner_arc = Arc((0, 422.5), 40, 40, theta1=180, theta2=0,
                       linewidth=lw, color=color)

# List of the court elements to be plotted onto the axes
```

```
    court_elements = [hoop, backboard, outer_box, inner_box, top_free_throw,
                      bottom_free_throw, restricted, corner_three_a,
                      corner_three_b, three_arc, center_outer_arc,
                      center_inner_arc]

    if outer_lines:
        # Draw the half court line, baseline and side out bound lines
        outer_lines = Rectangle((-250, -47.5), 500, 470, linewidth=lw,
                                color=color, fill=False)
        court_elements.append(outer_lines)

    # Add the court elements onto the axes
    for element in court_elements:
        ax.add_patch(element)

    return ax
```

```
# Create figure and axes
fig = plt.figure(figsize=(6, 6))
ax = fig.add_axes([0, 0, 1, 1])

# Plot hexbin of shots
ax.hexbin(df['LOC_X'], df['LOC_Y'], gridsize=(30, 30), extent=(-300, 300, 0, 940), bins='l
ax = draw_court(ax, 'black')

# Annotate player name and season
ax.text(0, 1.05, 'Stephen Curry\n2021-22 Regular Season', transform=ax.transAxes, ha='left

# Set axis limits
_ = ax.set_xlim(-250, 250)
_ = ax.set_ylim(0, 400)
```



Click for Hint.

```
Solution (Hint).
from nba_api.stats.static import players
from nba_api.stats.endpoints import shotchartdetail
player_dict = players.get_players()
```

These lines import some packages and get player information and save them into `player_dict`.

```
for player in player_dict:
    if player['full_name'] == 'Stephen Curry':
        print(player['id'])
```

Go through all records in `player_dict`. If the name of a player is `Stephen Curry`, get his `id`. Then we will know the `id` of `Stephen Curry`.

To be omitted.

Part III

Python Basics

4 Built-in Types: numeric types and `str`

This section is based on [1].

There are several built-in data structures in Python. Here is an (incomplete) list:

- `None`
- Boolean – `True`, `False`
- Numeric Types — `int`, `float`, `complex`
- Text Sequence Type — `str`
- Sequence Types — `list`
- Map type - `dict`

We will cover numeric types and strings in this section. The rests are either simple that are self-explained, or not simple that will be discussed later.

4.1 Numeric types and math expressions

Numeric types are represented by numbers. If there are no confusions, Python will automatically detect the type.

```
x = 1 # x is an int.
y = 2.0 # y is a float.
```

Python can do math just like other programming languages. The basic math operations are listed as follows.

- `+`, `-`, `*`, `/`, `>`, `<`, `>=`, `<=` works as normal.
- `**` is the power operation.
- `%` is the mod operation.
- `!=` is `not equal`

4.2 `str`

Scalars are represented by numbers and strings are represented by quotes. Example:

```
x = 1       # x is a scalar.
y = 's'     # y is a string with one letter.
z = '0'     # z loos like a number, but it is a string.
w = "Hello" # w is a string with double quotes.
```

Here are some facts.

1. For strings, you can use either single quotes `'` or double quotes `"`.
2. `\` is used to denote escaped words. You may find the list Here.
3. There are several types of scalars, like `int`, `float`, etc.. Usually Python will automatically determine the type of the data, but sometimes you may still want to declare them manually.
4. You can use `int()`, `str()`, etc. to change types.

Although `str` is a built-in type, there are tons of tricks with `str`, and there are tons of packages related to strings. Generally speaking, to play with strings, we are interested in two types of questions.

- Put information together to form a string.
- Extract information from a string. We briefly talk about these two tasks.

> **Note**
>
> There is a very subtle relations between the variable / constant and the name of the variable / constant. We will talk about these later.

Example 4.1. Here is an example of playing with strings. Please play with these codes and try to understand what they do.

```
import re

def clean_strings(strings):
    result = []
    for value in strings:
        value = value.strip()
        value = re.sub('[!#?]', '', value)
        value = value.title()
        result.append(value)
    return result

states = [' Alabama ', 'Georgia!', 'Georgia', 'georgia', 'FlOrIda',
```

```
            'south carolina##', 'West virginia?']
print(clean_strings(states))
```

```
['Alabama', 'Georgia', 'Georgia', 'Georgia', 'Florida', 'South Carolina', 'West Virginia']
```

5 Fundamentals

This section is mainly based on [2].

5.1 Indentation

One key feature about Python is that its structures (blocks) is determined by **Indentation**.

Let's compare with other languages. Let's take `C` as an example.

```
/*This is a C function.*/
int f(int x){return x;}
```

The block is defined by `{}` and lines are separated by `;`. `space` and `newline` are not important when `C` runs the code. It is recommended to write codes in a "beautiful, stylish" format for readibility, as follows. However it is not mandatary.

```
/*This is a C function.*/
int f(int x) {
   return x;
}
```

In Python, blocks starts from `:` and then are determined by indents. Therefore you won't see a lot of `{}` in Python, and the "beautiful, stylish" format is mandatary.

```
# This is a Python function.
def f(x):
    return x
```

The default value for indentation is 4 spaces, which can be changed by users. We will just use the default value in this course.

> **Note**
>
> It is usually recommended that one line of code should not be very long. If you do have one, and it cannot be shortened, you may break it into multiline codes directly in Python.

However, since indentation is super important in Python, when break one line code into multilines, please make sure that everything is aligned perfectly. Please see the following example.

```
results = shotchartdetail.ShotChartDetail(
            team_id = 0,
            player_id = 201939,
            context_measure_simple = 'FGA',
            season_nullable = '2021-22',
            season_type_all_star = 'Regular Season')
```

5.2 Binary operators and comparisons

Most binary operators behaves as you expected. Here I just want to mention `==` and `is`.

- `==` is testing whehter these two objects have the same value.
- `is` is testing whether these two objects are exactly the same.

> **Note**
>
> You may use `id(x)` to check the id of the object `x`. Two objects are identical if they have the same id.

5.3 `import`

In Python a module is simply a file with the .py extension containing Python code. Assume that we have a Python file `example.py` stored in the folder `assests/codes/`. The file is as follows.

```
# from assests/codes/example.py

def f(x):
    print(x)

A = 'You get me!'
```

You may get access to this function and this string in the following way.

```
from assests.codes import example

example.f(example.A)
```

```
You get me!
```

5.4 Comments

Any text preceded by the hash mark (pound sign) `#` is ignored by the Python interpreter. In many IDEs you may use hotkeys to directly toggle multilines as comments. For example, in VS Code the default setting for toggling comments is `ctrl+/`.

5.5 Dynamic references, strong types

In some programming languages, you have to declare the variable's name and what type of data it will hold. If a variable is declared to be a number, it can never hold a different type of value, like a string. This is called *static typing* because the type of the variable can never change.

Python is a *dynamically typed* language, which means you do not have to declare a variable or what kind of data the variable will hold. You can change the value and type of data at any time. This could be either great or terrible news.

On the other side, "dynamic typed" doesn't mean that types are not important in Python. You still have to make sure that the types of all variables meet the requirements of the operations used.

```
a = 1
b = 2
b = '2'
c = a + b
```

```
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

In this example, `b` was first assigned by a number, and then it was reassigned by a `str`. This is totally fine since Python is dynamically types. However later when adding `a` and `b`, the type error occurs since you cannot add a number and a `str`.

> **Note**
>
> You may always use `type(x)` to detect the type of the object `x`.

5.6 Everything is an object

Every number, string, data structure, function, class, module, and so on exists in the Python interpreter in its own "box", which is referred to as a *Python object*.

Each object has an associated type (e.g., string or function) and internal data. In practice this makes the language very flexible, as even functions can be treated like any other object.

Each object might have attributes and/or methods attached.

5.7 Mutable and immutable objects

An object whose internal state can be changed is *mutable*. On the other hand, *immutable* doesn't allow any change in the object once it has been created.

Some objects of built-in type that are mutable are:

- Lists
- Dictionaries
- Sets

Some objects of built-in type that are immutable are:

- Numbers (Integer, Rational, Float, Decimal, Complex & Booleans)
- Strings
- Tuples

Example 5.1 (Tuples are not really "immutable")**.** You can treat a tuple as a container, which contains some objects. The relations between the container and its contents are immutable, but the objects it holds might be mutable. Please check the following example.

```
container = ([1], [2])
print('This is `container`: ', container)
print('This is the id of `container`: ', id(container))
print('This is the id of the first list of `container`: ', id(container[0]))
```

```
container[0].append(2)
print('This is the new `container`: ', container)
print('This is the id of the new `container`: ', id(container))
print('This is the id of the first list (which is updated) of the new `container`: ', id(c
```

```
This is `container`:  ([1], [2])
This is the id of `container`:  1295204025600
This is the id of the first list of `container`:  1295204023936
This is the new `container`:  ([1, 2], [2])
This is the id of the new `container`:  1295204025600
This is the id of the first list (which is updated) of the new `container`:  1295204023936
```

You can see that the tuple `container` and its first object stay the same, although we add one element to the first object.

6 Flows and Logic

6.1 `for` loop

- `range(10)`
- `list`

6.2 `if` conditional control

7 list

> **Note**
>
> In Python, a list is an ordered sequence of object types and a string is an ordered sequence of characters.

- Access to the data
- Slicing
- Methods
  - `append` and `+`
  - `extend`
  - `pop`
  - `remove`
- `in`
- `for`
- `list()`
- `sorted`
- `str.split`
- `str.join`

7.1 List Comprehension

List Comprehension is a convenient way to create lists based on the values of an existing list. It cannot provide any real improvement to the performance of the codes, but it can make the codes shorter and easier to read.

The format of list Comprehension is

```
newlist = [expression for item in iterable if condition == True]
```

8 `dict`

- Access to the data
- Methods
  - directly add items
  - `update`
  - `get`
  - `keys`
  - `values`
  - `items`
- `dict()`
- dictionary comprehension

9 Exercises

Most problems are based on [3], [1] and [4].

Exercise 9.1 (Indentation)**.** Please tell the differences between the following codes. If you don't understand `for` don't worry about it. Just focus on the indentation and try to understand how the codes work.

```
for i in range(5):
    print('Hello world!')
print('Hello world!')
```

```
for i in range(5):
    print('Hello world!')
    print('Hello world!')
```

```
for i in range(5):
print('Hello world!')
print('Hello world!')
```

```
for i in range(5):
    pass
print('Hello world!')
print('Hello world!')
```

Exercise 9.2 (Play with built-in data types)**.** Please first guess the results of all expressions below, and then run them to check your answers.

```
print(True and True)
print(True or True)
print(False and True)
```

```
print((1+1>2) or (1-1<1))
```

Exercise 9.3 (== vs is)**.** Please explain what happens below.

```
a = 1
b = 1.0
print(type(a))
print(type(b))

print(a == b)
print(a is b)
```

```
<class 'int'>
<class 'float'>
True
False
```

Exercise 9.4 (Play with strings)**.** Please excute the code below line by line and explain what happens in text cells.

```
# 1
answer = 10
wronganswer = 11
text1 = "The answer to this question is {}. If you got {}, you are wrong.".format(answer,
print(text1)

# 2
var = True
text2 = "This is {}.".format(var)
print(text2)

# 3
word1 = 'Good '
word2 = 'buy. '
text3 = (word1 + word2) * 3
print(text3)
```

```
# 4
sentence = "This is\ngood enough\nfor a exercise to\nhave so many parts. " \
           "We would also want to try this symbol: '. " \
           "Do you know how to type \" in double quotes?"
print(sentence)
```

```
The answer to this question is 10. If you got 11, you are wrong.
This is True.
Good buy. Good buy. Good buy.
This is
good enough
for a exercise to
have so many parts. We would also want to try this symbol: '. Do you know how to type " in do
```

Exercise 9.5 (`split` and `join`)**.** Please excute the code below line by line and explain what happens in text cells.

```
sentence = 'This is an example of a sentence that I expect you to split.'

wordlist = sentence.split(' ')

newsentence = '\n'.join(wordlist)
print(newsentence)
```

Exercise 9.6 (List reference)**.** Please finish the following tasks.

1. Given the list `a`, make a new reference `b` to `a`. Update the first entry in `b` to be `0`. What happened to the first entry in `a`? Explain your answer in a text block.
2. Given the list `a`, make a new copy `b` of the list `a` using the function `list`. Update the first entry in `b` to be `0`. What happened to the first entry in `a`? Explain your answer in a text block.

Exercise 9.7 (List comprehension)**.** Given a list of numbers, use list comprehension to remove all odd numbers from the list:

```
numbers = [3,5,45,97,32,22,10,19,39,43]
```

Exercise 9.8 (More list comprehension)**.** Use list comprehension to find all of the numbers from 1-1000 that are divisible by 7.

Exercise 9.9 (More list comprehension)**.** Count the number of spaces in a string.

Exercise 9.10 (More list comprehension)**.** Use list comprehension to get the index and the value as a tuple for items in the list `['hi', 4, 8.99, 'apple', ('t,b', 'n')]`. Result would look like `[(index, value), (index, value), ...]`.

Exercise 9.11 (More list comprehension)**.** Use list comprehension to find the common numbers in two lists (without using a tuple or set) `list_a = [1, 2, 3, 4]`, `list_b = [2, 3, 4, 5]`.

Exercise 9.12 (Probability)**.** Compute the probability that two people out of 23 share the same birthday. The math formula for this is

$$1 - \frac{365!/(365-23)!}{365^{23}} = 1 - \frac{365}{365} \cdot \frac{365-1}{365} \cdot \frac{365-2}{365} \cdot \ldots \cdot \frac{365-22}{365}.$$

1. To directly use the formula we have to use a high performance math package, e.g. `math`. Please use `math.factorial` to compute the above formula.
2. Please use the right hand side of the above formula to compute the probability using the following steps.
   a. Please use the list comprehension to create a list $[\frac{365}{365}, \frac{365-1}{365}, \frac{365-2}{365}, \ldots, \frac{365-22}{365}]$.
   b. Use `numpy.prod` to compute the product of elements of the above list.
   c. Compute the probability by finishing the formula.
3. Please use `time` to test which method mentioned above is faster.

10 Projects

Most projects are based on [2], [5].

Exercise 10.1 (Determine the indefinite article)**.** Please finish the following tasks.

1. Please construct a list `aeiou` that contains all vowels.
2. Given a word `word`, we would like to find the indefinite article `article` before `word`. (Hint: the article should be `an` if the first character of `word` is a vowel, and `a` if not.)

Click for Hint.

Solution. Consider `in`, `.lower()` and `if` structure.

Exercise 10.2 (Datetime and files names)**.** We would like to write a program to quickly generate `N` files. Every time we run the code, `N` files will be generated. We hope to store all files generated and organize them in a neat way. To achieve this, one way is to create a subfolder for each run and store all files generated during that run in the particular subfolder. Since we would like to make it fast, the real point of this task is to find a way to automatically generate the file names for the files generated and the folder names for the subfolders generated. You don't need to worry about the contents of the files and empty files are totally fine for this problem.

Click for Hint.

Solution. One way to automatically generate file names and folder names is to use the date and the time when the code is run. Please check `datetime` package for getting and formatting date/time, and `os` packages for playing with files and folders.

Exercise 10.3 (Color the Gnomic data)**.** We can use ASCII color codes in the string to change the color of strings, as an example `\033[91m` for red and `\033[94m` for blue. See the following example.

```
print('\033[91m'+'red'+'\033[92m'+'green'+'\033[94m'+'blue'+'\033[93m'+'yellow')
```

Consider an (incomplete) Gnomic data given below which is represented by a long sequence of `A`, `C`, `T` and `G`. Please color it using ASCII color codes.

```
Gnomicdata = 'TCGATCTCTTGTAGATCTGTTCTCTAAACGAACTTTAAAATCTGTGTGGCTGTCACTCGG'\
             'CTGCATGCTTAGTGCACTCACGCAGTATAATTAATAACTAATTACTGTCGTTGACAGGAC'\
             'ACGAGTAACTCGTCTATCTTCTGCAGGCTGCTTACGGTTTCGTCCGTGTTGCAGCCGATC'\
             'ATCAGCACATCTAGGTTTTGTCCGGGTGTGACCGAAAGGTAAGATGGAGAGCCTTGTCCC'\
             'TGGTTTCAACGAGAAAACACACGTCCAACTCAGTTTGCCTGTTTTACAGGTTCGCGACGT'\
             'GCTCGTACGTGGCTTTGGAGACTCCGTGGAGGAGGTCTTATCAGAGGCACGTCAACATCT'\
             'TAAAGATGGCACTTGTGGCTTAGTAGAAGTTGAAAAAGGCGTTTTGCCTCAACTTGAACA'\
             'GCCCTATGTGTTCATCAAACGTTCGGATGCTCGAACTGCACCTCATGGTCATGTTATGGT'\
             'TGAGCTGGTAGCAGAACTCGAAGGCATTCAGTACGGTCGTAGTGGTGAGACACTTGGTGT'\
             'CCTTGTCCCTCATGTGGGCGAAATACCAGTGGCTTACCGCAAGGTTCTTCTTCGTAAGAA'\
             'CGGTAATAAAGGAGCTGGTGGCCATAGTTACGGCGCCGATCTAAAGTCATTTGACTTAGG'\
             'CGACGAGCTTGGCACTGATCCTTATGAAGATTTTCAAGAAAACTGGAACACTAAACATAG'
```

Click for Hint.

Solution (Hint). You may use `if` to do the conversion. Or you may use `dict` to do the conversion.

Exercise 10.4 (`sorted`)**.** Please read through the Key funtions in this article, and sort the following two lists.

1. Sort `list1 = [[11,2,3], [2, 3, 1], [5,-1, 2], [2, 3,-8]]` according to the sum of each list.
2. Sort `list2 = [{'a': 1, 'b': 2}, {'a': 3, 'b': 4},{'a': 5, 'b': 2}]` according to the `b` value of each dictionary.

Exercise 10.5 (Fantasy Game Inventory)**.** You are creating a fantasy video game. The data structure to model the player's inventory will be a dictionary where the keys are string values describing the item in the inventory and the value is an integer value detailing how many of that item the player has. For example, the dictionary value `{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}` means the player has 1 rope, 6 torches, 42 gold coins, and so on.

Write a program to take any possible `inventory` and display it like the following:

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger
Total number of items: 62
```

Part IV

Package: `numpy`

The main reference for this chapter is [2].

11 Basics

The basic data structure for `numpy` is `numpy.ndarray`. You may treat it as a generalized version of lists. However it can do so much more than the build-in `list`.

To use `numpy`, we just import it. In most cases you would like to use the alias `np`.

```
import numpy as np
```

> **Note**
>
> In many cases, `numpy.ndarray` is a huge object since it stores tons of data. Therefore many of the operations related to `numpy.ndarray` are "in-place" by default. This means that if you don't explicitly ask for a copy, there will be only one copy of the array and all later operations make changes to the original one.
> However there are many cases that

12 Create `np.ndarray`

- convert a list into a numpy array.
- `np.zeros`, `np.zeros_like`
- `np.ones`, `np.ones_like`
- `np.eye`
- `np.random.rand`
- `np.arange`
- `np.linspace`

> **Note**
>
> Please be very careful about the format of the input. For example, when you want to specify the dimension of the array, using `np.zeros`, you need to input a `tuple`. On the other hand, when using `np.random.rand`, you just directly input the dimensions one by one.
>
> ```
> import numpy as np
>
> np.zeros((3, 2))
> np.random.rand(3, 2)
> ```
>
> In this case, the official documents are always your friend.

13 Mathematical and Statistical Methods

- `+`, `-`, `*`, `/`, `**`, etc..
- `np.sin`, `np.exp`, `np.sqrt`, etc..
- `mean`, `sum`, `std`, `var`, `cumsum`
- `max` and `min`
- `maximum` and `minimum`
- `argmin` and `argmax`
- `np.sort`
- `np.unique`, `np.any`
- `np.dot`: Matrix multiplication
- `np.concatenate`
- Broadcast

Example 13.1 (Axis)**.** Given `A = np.array([[1,2],[3,4]])` and `B = np.array([[5,6],[7,8]])`, please use `np.concatenate` to concatencate these two matrices to get a new matrix, in the order:

- `A` left, `B` right
- `A` right, `B` left
- `A` up, `B` down
- `A` down, `B` up

14 Common attributes and methods

- `shape`
- `dtype`
- `ndim`
- Any arithmetic operations between equal-size arrays applies the operation element-wise.

Example 14.1. `MNIST` is a very famous dataset of hand written images. Here is how to load it. Note that in this instance of the dataset the data are stored as `numpy` arraies.

```
import tensorflow as tf

(X_train, y_train), (X_test, y_test) = tf.keras.datasets.mnist.load_data()
X_train.shape
```

15 Basic indexing and slicing

First see the following example.

Example 15.1

```
import numpy as np
arr = np.arange(10)

print(arr[5])
print(arr[5:8])

arr[5:8] = 12
print(arr)

print(arr[5:8:2])
print(arr[8:5:-1])
print(arr[::-1])
```

```
5
[5 6 7]
[ 0  1  2  3  4 12 12 12  8  9]
[12 12]
[ 8 12 12]
[ 9  8 12 12 12  4  3  2  1  0]
```

To do slicing in higher dimensional case, you may either treat a `numpy` array as a nested list, or you may directly work with it with multiindexes.

Example 15.2

```
import numpy as np
arr3d = np.arange(12).reshape(2, 2, 3)

print('case 1:\n {}'.format(arr3d))
```

```
  print('case 2:\n {}'.format(arr3d[0, 1, 2]))
  print('case 3:\n {}'.format(arr3d[:, 0: 2, 1]))
  print('case 4:\n {}'.format(arr3d[:, 0: 2, 1:2]))
```

```
case 1:
 [[[ 0  1  2]
  [ 3  4  5]]

 [[ 6  7  8]
  [ 9 10 11]]]
case 2:
 5
case 3:
 [[ 1  4]
 [ 7 10]]
case 4:
 [[[ 1]
  [ 4]]

 [[ 7]
  [10]]]
```

16 Boolean Indexing

`numpy` array can accept index in terms of numpy arries with boolean indexing.

Example 16.1

```
import numpy as np
a = np.arange(4)
b = np.array([True, True, False, True])
print(a)
print(b)
print(a[b])
```

```
[0 1 2 3]
[ True  True False  True]
[0 1 3]
```

We could combine this way with the logic computation to filter out the elements we don't want.

Example 16.2. Please replace the odd number in the array by its negative.

```
import numpy as np
arr = np.arange(10)
odd = arr %2 == 1
arr[odd] = arr[odd] * (-1)

print(arr)
```

```
[ 0 -1  2 -3  4 -5  6 -7  8 -9]
```

17 Fancy indexing

Fancy indexing is a term adopted by NumPy to describe indexing using integer arrays.

Example 17.1

```
import numpy as np

arr = np.zeros((8, 4))
for i in range(8):
    arr[i] = i

arr[[4, 3, 0, 6]]
```

```
array([[4., 4., 4., 4.],
       [3., 3., 3., 3.],
       [0., 0., 0., 0.],
       [6., 6., 6., 6.]])
```

Example 17.2

```
import numpy as np

arr = np.arange(32).reshape((8, 4))
print(arr)
print(arr[[1, 5, 7, 2], [0, 3, 1, 2]])
print(arr[[1, 5, 7, 2]][:, [0, 3, 1, 2]])
```

```
[[ 0  1  2  3]
 [ 4  5  6  7]
 [ 8  9 10 11]
 [12 13 14 15]
 [16 17 18 19]
 [20 21 22 23]]
```

```
 [24 25 26 27]
 [28 29 30 31]]
[ 4 23 29 10]
[[ 4  7  5  6]
 [20 23 21 22]
 [28 31 29 30]
 [ 8 11  9 10]]
```

18 Copies and views

The view of an numpy array is a way to get access to the array without copying internel data. When operating with a view, the original data as well as all other views of the original data will be modified simutanously.

The default setting for copies and views is that, basic indexing and slicing will make views, and advanced indexing and slicing (e.g. boolean indexing, fancy indexing, etc.) will make copies. For other operations, you need to check the documents to know how they work. For example, `np.reshape` creates a view where possible, and `np.flatten` always creates a copy.

You may use `np.view()` or `np.copy()` to make views or copies explicitly. ::: {#exm-}

```
import numpy as np
arr = np.arange(10)
b = arr[5:8]
print('arr is {}'.format(arr))
print('b is {}'.format(b))

b[0] = -1
print('arr is {}'.format(arr))
print('b is {}'.format(b))


arr[6] = -2
print('arr is {}'.format(arr))
print('b is {}'.format(b))

print('The base of b is {}'.format(b.base))
```

```
arr is [0 1 2 3 4 5 6 7 8 9]
b is [5 6 7]
arr is [ 0  1  2  3  4 -1  6  7  8  9]
b is [-1  6  7]
arr is [ 0  1  2  3  4 -1 -2  7  8  9]
b is [-1 -2  7]
The base of b is [ 0  1  2  3  4 -1 -2  7  8  9]
```

:::

The way to make explicit copy is `.copy()`.

Example 18.1

```
import numpy as np
arr = np.arange(10)
b = arr[5:8].copy()
print('arr is {}'.format(arr))
print('b is {}'.format(b))

b[0] = -1
print('arr is {}'.format(arr))
print('b is {}'.format(b))


arr[6] = -2
print('arr is {}'.format(arr))
print('b is {}'.format(b))

print('The base of b is {}'.format(b.base))
```

```
arr is [0 1 2 3 4 5 6 7 8 9]
b is [5 6 7]
arr is [0 1 2 3 4 5 6 7 8 9]
b is [-1  6  7]
arr is [ 0  1  2  3  4  5 -2  7  8  9]
b is [-1  6  7]
The base of b is None
```

19 More commands

- `.T`
- `axis=n` is very important.
- `np.reshape()`
- `np.tile()`
- `np.repeat()`

20 More advanced commands

- `np.where()`
- `np.any()`
- `np.all()`
- `np.argsort()`

Example 20.1. Get the position where elements of `a` and `b` match.

```
a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])

np.where(a == b)
```

```
(array([1, 3, 5, 7], dtype=int64),)
```

Example 20.2.

```
a = np.array([1,2,3,2,3,4,3,4,5,6])
b = np.array([7,2,10,2,7,4,9,4,9,8])

np.where(a == b, a*2, b+1)
```

```
array([ 8,  4, 11,  4,  8,  8, 10,  8, 10,  9])
```

Example 20.3 (Playing with axis)**.**

```
import numpy as np
a = np.array([[[1,2],[3,4]],[[5,6],[7,8]]])

np.any(a==1, axis=0)
```

```
np.any(a==1, axis=1)
np.any(a==1, axis=2)


np.any(a==2, axis=0)
np.any(a==2, axis=1)
np.any(a==2, axis=2)

np.any(a==5, axis=0)
np.any(a==5, axis=1)
np.any(a==5, axis=2)
```

```
array([[False, False],
       [ True, False]])
```

21 Examples

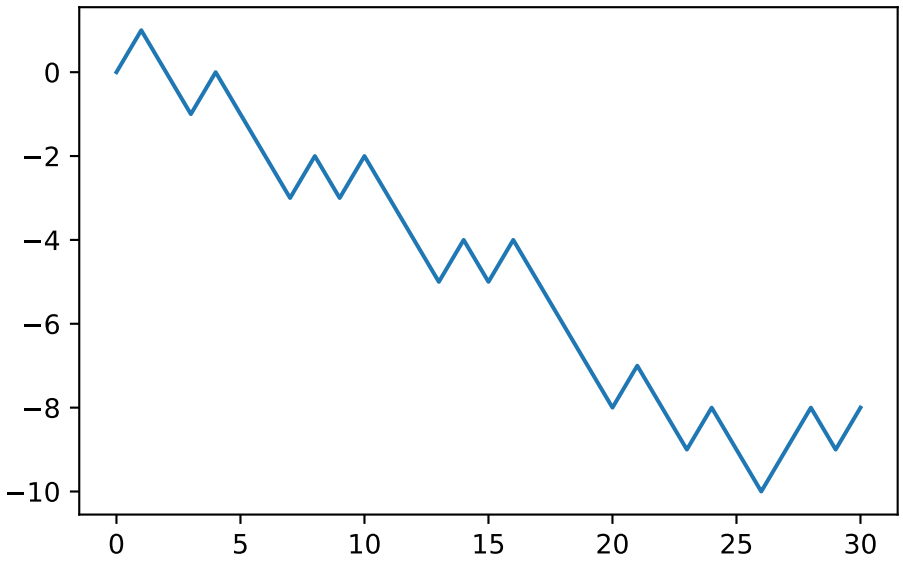
Example 21.1 (Random walks)**.** Adam walks randomly along the axis. He starts from `0`. Every step he has equal possibility to go left or right. Please simulate this process.

Use `choices` to record the choice of Adam at each step. We may generate a random array where `0` represents left and `1` represents right.

Use `positions` to record the position of Adam at each step. Using `choices`, the position is `+1` if we see a `1` and the position is `-1` if we see a `0`. So the most elegent way to perform this is to

1. Convert `choices` from `{0, 1}` to `{-1, 1}`.
2. To record the starting position, we attach `0` to the beginning of the new `choices`.
3. Apply `cumsum` to `choices` to get `positions`.

```
import numpy as np

step = 30
choices = np.random.randint(2, size=step)
choices = choices * 2 - 1
choices = np.concatenate(([0], choices))
positions = choices.cumsum()

import matplotlib.pyplot as plt
plt.plot(positions)
```



Example 21.2 (Many random walks)**.** We mainly use `numpy.ndarray` to write the code in the previous example. The best part here is that it can be easily generalized to many random walks.

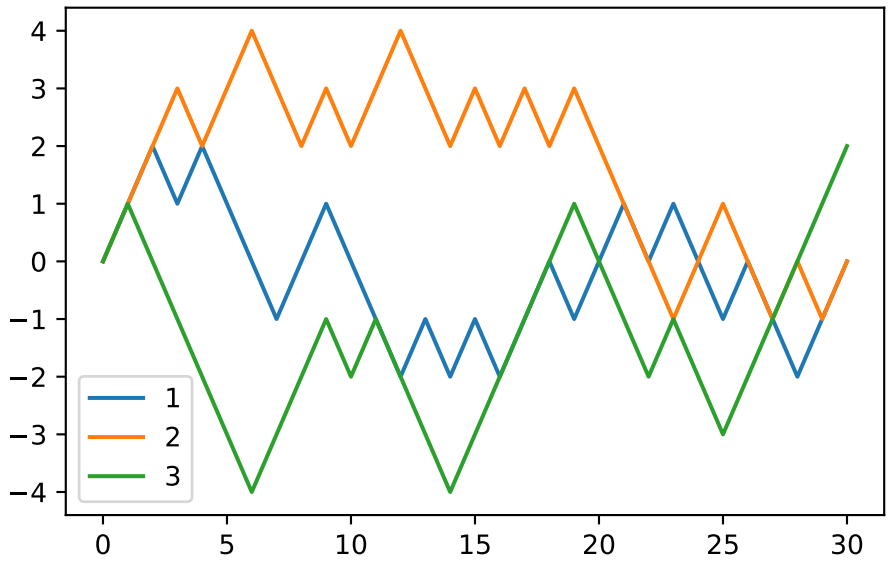
Still keep `choices` and `positions` in mind. Now we would like to deal with multiple people simutanously. Each row represents one person's random walk. All the formulas stay the same. We only need to update the dimension setting in the previous code.

- Update `size` in `np.random.randint`.
- Update `[0]` to `np.zeros((N, 1))` in `concatenate`.
- For `cumsum` and `concatenate`, add `axis=1` to indicate that we perform the operations along `axis 1`.
- We plot each row in the same figure. `plt.legend` is used to show the label for each line.

```
import numpy as np

step = 30
N = 3
choices = np.random.randint(2, size=(N, step))
choices = choices * 2 - 1
choices = np.concatenate((np.zeros((N, 1)), choices), axis=1)
positions = choices.cumsum(axis=1)
```

```
import matplotlib.pyplot as plt
for row in positions:
    plt.plot(row)
plt.legend([1, 2, 3])
```

```
<matplotlib.legend.Legend at 0x2066cf0ea00>
```



Example 21.3 (Analyze `positions`)**.** We play with the numpy array `positions` to get some information about the random walks of three generated in the previous example.

- The maximal position:

```
positions.max()
```

```
4.0
```

- The maximal position for each one:

```
positions.max(axis=1)
```

```
array([2., 4., 2.])
```

- The maximal position across all three for each step:

```
positions.max(axis=0)
```

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

- Check whether anyone once got to the position 3:

```
(positions>=3).any(axis=1)
```

```
array([False,  True, False])
```

- The number of people who once got to the position 3:

```
(positions>=3).any(axis=1).sum()
```

```
1
```

- Which step for each one gets to the right most position:

```
positions.argmax(axis=1)
```

```
array([ 2,  6, 30], dtype=int64)
```

22 Exercises

Many exercises are from [6].

Exercise 22.1 (array)**.** Write a NumPy program to create a 3×3 matrix with values ranging from 2 to 10.

Exercise 22.2 (array)**.** Write a NumPy program to create a null vector of size 10 and update sixth value to 11.

Exercise 22.3 (array)**.** Write a NumPy program to reverse an array (first element becomes last).

Exercise 22.4 (array)**.** Write a NumPy program to create a 10×10 2D-array with 1 on the border and 0 inside.

Exercise 22.5 (repeat and tile)**.** Given `a = np.array([1,2,3])`, please get the desired output `array([1, 1, 1, 2, 2, 2, 3, 3, 3, 1, 2, 3, 1, 2, 3, 1, 2, 3])`.

Exercise 22.6 (Compare two `numpy` arraies)**.** Consider two `numpy` arraies `x` and `y`. Compare them entry by entry. We would like to know how many are the same.

Click to expand.

Solution. Note that `bool` values `True` and `False` can be treated as numbers `1` and `0`.

```
import numpy as np

x = np.array([1, 2, 3, 4, 5])
y = np.array([2, 1, 4, 4, 5])

numofsame = np.sum(x == y)
print(numofsame)
```

```
2
```

Exercise 22.7. Get all items between `5` and `10` from an array `a = np.array([2, 6, 1, 9, 10, 3, 27])`.

Exercise 22.8. Swap rows `1` and `2` in the array `arr = np.arange(9).reshape(3,3)`.

Exercise 22.9. Please finish the following tasks.

1. Reverse the rows of a 2D array `arr = np.arange(9).reshape(3,3)`.
2. Reverse the columns of a 2D array `arr = np.arange(9).reshape(3,3)`.

Exercise 22.10. Create a 2D array of shape `5x3` to contain random decimal numbers between `5` and `10`.

Exercise 22.11. Use the following code to get the dataset `iris`.

```
import numpy as np

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_1d = np.genfromtxt(url, delimiter=',', dtype=None, encoding=None)
```

1. `iris_1d` is a 1D numpy array that each item is a tuple. Please construct a new 1D numpy array that each item is the last componenet of each tuple in `iris_1d`.
2. Convert `iris_1d` into a 2D array `iris_2d` by omitting the last field of each item.

Exercise 22.12 (Normalization)**.** Use the following code to get an 1D array `sepallength`.

```
import numpy as np

url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
sepallength = np.genfromtxt(url, delimiter=',', dtype='float', usecols=[0],
                            encoding=None)
```

Please normalize it such that the values of each item is between `0` and `1`.

Exercise 22.13. `np.isnan()` is a function to check whether each entry of a numpy array is `nan` or not. Please use this as well as `np.where` to find all `nan` entries in an array.

You may use the following array `iris_2d` to test your code.

```
import numpy as np
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_2d = np.genfromtxt(url, delimiter=',', dtype='float', encoding=None)
iris_2d[np.random.randint(150, size=20), np.random.randint(4, size=20)] = np.nan
```

Exercise 22.14. Select the rows of `iris_2d` that does not have any nan value.

```
import numpy as np
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_2d = np.genfromtxt(url, delimiter=',', dtype='float', usecols=[0,1,2,3],
                        encoding=None)
iris_2d[np.random.randint(150, size=20), np.random.randint(4, size=20)] = np.nan
```

Exercise 22.15. Replace all `nan` with `0` in numpy array `iris_2d`.

```
import numpy as np
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_2d = np.genfromtxt(url, delimiter=',', dtype='float', usecols=[0,1,2,3],
                        encoding=None)
iris_2d[np.random.randint(150, size=20), np.random.randint(4, size=20)] = np.nan
```

Exercise 22.16. Consider `x = np.array([1, 2, 1, 1, 3, 4, 3, 1, 1, 2, 1, 1, 2])`. Please find the index of 5th repetition of number `1` in `x`.

23 Projects

Exercise 23.1 (Adding one axis)**.** Please download this file.

1. Please use `matplotlib.pyplot.imread()` to read the file as a 3D numpy array.
2. Check the shape of the array.
3. Add one additional axis to it as axis 0 to make it into a 4D array.

Exercise 23.2 (Random)**.** Please finish the following tasks.

1. Use the package `np.random` to flip a coin 100 times and record the result in a list `coin`.
2. Assume that the coin is not fair, and the probability to get `H` is `p`. Write a code to flip the coin 100 times and record the result in a list `coin`, with a given parameter `p`. You may use `p=.4` as the first choice.
3. For each list `coin` created above, write a code to find the longest `H` streak. We only need the biggest number of consecutive `H` we get during this 100 tosses. It is NOT necessary to know when we start the streak.

Click for Hint.

Solution. The following ideas can be used to solve the problem.

- `np.where`
- string, `split` and `join`

Exercise 23.3 (Bins)**.** Please read the document of `np.digitize`, and use it to do the following task.

Set the following bins:

- Less than `3`: `small`
- `3-5`: `medium`

- Bigger than 5: `large`

Please transform the following data `iris_2c` into texts using the given bins.

```
import numpy as np
url = 'https://archive.ics.uci.edu/ml/machine-learning-databases/iris/iris.data'
iris_2c = np.genfromtxt(url, delimiter=',', dtype='object')[:, 2].astype('float')
```

Exercise 23.4. Consider a 2D numpy array `a`.

```
import numpy as np
a = np.random.rand(5, 5)
```

1. Please sort it along the 3rd column.
2. Please sort it along the 2nd row.

Click for Hint.

Solution. Please use `np.argsort` for the problem.

Exercise 23.5 (One-hot vector)**.** Compute the one-hot encodings of a given array. You may use the following array as a test example. In this example, there are `3` labels. So the one-hot vectors are 3 dimensional vectors.

For more infomation about one-hot encodings, you may check the Wiki page. You are not allowed to use packages that can directly compute the one-hot encodings for this problem.

```
import numpy as np
arr = np.random.randint(1,4, size=6)
```

Exercise 23.6. From the given 1d array `arr = np.arange(15)`, generate a 2d matrix using strides, with a window length of `4` and strides of `2`, like `[[0,1,2,3], [2,3,4,5], [4,5,6,7]..]`.

Part V

Package: `pandas`

The basic data structure for `pandas` is `pandas.DataFrame`. You may treat it as a generalized version of tables.

To use `pandas`, we just import it. In most cases you would like to use the alias `pd`.

```
import pandas as pd
```

Since `DataFrame` is more like a table, the biggest questions here is not to do computations (which is still very important), but to retrieve, search, sort, merge, etc.. those data.

24 Basic pandas

24.1 Series and DataFrame

A *Series* is a 1-d array-like object which has index. The default index is starting from 0. You may change the index to be something assigned by you. Thus it can be treated as a generalization of a dict.

```
obj = pd.Series([3, 1, 2, 4])
obj
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
0	3
1	1
2	2
3	4

```
obj2 = pd.Series([3, 1, 2, 4], index=['a', 'b', 'c', 'd'])
obj2
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
a	3
b	1
c	2
d	4

```
data3 = {'a': 3, 'b': 1, 'c': 2, 'd': 4}
obj3 = pd.Series(data3)
obj3
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
a	3
b	1
c	2
d	4

A *DataFrame* represents a rectangular table of data and contains an ordered collection of columns, each of which can be a different value type. The DataFrame has both a row and column index; it can be thought of as a dict of Series all sharing the same index. When displaying a DataFrame, we may use `.head()` to just display the first few rows for efficicy.

```
import pandas as pd

data = {'a': [1, 2, 3, 4, 5, 6, 7],
        'b': [1.1, 2.1, 3.1, 4.1, 5.1, 6.1, 7.1],
        'c': ['a', 'b', 'c', 'd', 'e', 'f', 'g']}
df = pd.DataFrame(data)
df.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	a	b	c
0	1	1.1	a
1	2	2.1	b
2	3	3.1	c
3	4	4.1	d
4	5	5.1	e

Note

We may use the setting `columns=` or `index=` as well as the methods `.rename(columns=, index=)` to change the column names and the index names. See the following example.

```
import numpy as np
import pandas as pd
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
```

24.2 Accessing data

- A column in a DataFrame can be retrieved as a Series either by dict-like notation or by attribute. What one gets from this is a Series object.
  - dict-like notation: `df['a']`
  - by attribute: `df.a`. Note that if the name of the column is not suitable for attribute names, this method doesn't work.
- Rows are retrieved by `.loc` if using the row index, and by `.iloc` if using the row number.

24.3 Updating data

- Assign values to a column of a DataFrame will update that column. If the column doesn't exist, new column will be created.
- When assign values with non-existent row index, that part of the data will be discarded.
- Any time if there are no values with a specific column and row, it will show as `NaN`.

Example 24.1.

```
import pandas as pd

data = {'a': [1, 2, 3, 4],
        'b': [1.1, 2.1, 3.1, 4.1],
        'c': ['a', 'b', 'c', 'd']}
df = pd.DataFrame(data)

newcol = {1: 'good', 3: 'better', 5: 'best'}
```

```
df['d'] = pd.Series(newcol)
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	a	b	c	d
0	1	1.1	a	NaN
1	2	2.1	b	good
2	3	3.1	c	NaN
3	4	4.1	d	better

24.4 Indexing, Selection, and Filtering

- Series indexing `(obj[...])` works analogously to NumPy array indexing, except you can use the Series's index values instead of only integers.
- We can use logical expresssion to filter DataFrame.

```
import pandas as pd

data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
data[data['one']>5]
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	one	two	three	four
Utah	8	9	10	11
New York	12	13	14	15

- `.loc`, `.iloc`

```
import pandas as pd
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])
print(data.loc['Colorado', ['two', 'three']])
print(data.iloc[2, [3, 0, 1]])
```

```
two      5
three    6
Name: Colorado, dtype: int32
four    11
one      8
two      9
Name: Utah, dtype: int32
```

- Slicing with labels behaves differently than normal Python slicing in that the endpoint is inclusive.

```
import pandas as pd

obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
obj['b':'c']
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
b	1.0
c	2.0

- Reindex `.reindex()`:

```
import pandas as pd
data = pd.DataFrame(np.arange(16).reshape((4, 4)),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])

data.reindex(index = ['Colorado', 'Arkansas', 'New York'],
             columns = ['three', 'five', 'one'])
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	three	five	one
Colorado	6.0	NaN	4.0
Arkansas	NaN	NaN	NaN
New York	14.0	NaN	12.0

> **Note**
>
> `.loc` and `.reindex` are very similar to each other. The main difference between theses two is that `.loc` will return a view and `.reindex` will return a copy in most cases.

> **Note**
>
> When locate data using indexes, duplicate labels will return all results.

24.5 Essential functions

- Arithmetic and Data Alignment Elements of the same index and columns will be computed. By default, if any entry is `nan`, the answer will be `nan`. You may use `fill_value` argument to fill the empty slots.

Example 24.2.

```
Example 24.2
import pandas as pd
import numpy as np
df1 = pd.DataFrame(np.arange(12.).reshape((3, 4)), columns=list('abcd'))
df2 = pd.DataFrame(np.arange(20.).reshape((4, 5)), columns=list('abcde'))
df2.loc[1, 'b'] = np.nan

df1.add(df2, fill_value=0)
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

Relatedly, when reindexing a Series or DataFrame, you can also specify a `fill_value`.

24.6 Function Application and Mapping

We may apply functions to each row/column of a DataFrame. If the function is built-in function that is compatible with DataFrame, you can directly call the function that it will be applied automatically to each row/column. If it is not, we can call `apply` to get the desired result.

Example 24.3.

```
import pandas as pd
data = pd.DataFrame(np.random.rand(4, 4),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])

f = lambda x: x.max() - x.min()

print(data.apply(f))
print(data.apply(f, axis='columns'))
```

```
one      0.391043
two      0.444902
three    0.645077
four     0.384059
dtype: float64
Ohio        0.832982
Colorado    0.729477
Utah        0.373919
New York    0.345299
dtype: float64
```

We can use more complicated function to get more complicated result.

Example 24.4.

```
import pandas as pd
data = pd.DataFrame(np.random.rand(4, 4),
                    index=['Ohio', 'Colorado', 'Utah', 'New York'],
                    columns=['one', 'two', 'three', 'four'])

f = lambda x: pd.Series([x.max(), x.min()], index=['max', 'min'])

print(data.apply(f))
```

```
          one       two     three      four
max  0.944360  0.985511  0.661402  0.513607
min  0.184477  0.327038  0.028321  0.140388
```

24.7 Sorting and Ranking

- .sort_values(by=)
- .rank(ascending=, method=)

24.8 Summarizing and Computing Descriptive Statistics

- sum, cumsum
- mean, median
- .describe()
- .cov, .corr

24.9 Unique Values, Value Counts, and Membership

- unique
- value_counts

24.10 Reading and Writing Data in Text Format

- read_csv
- read_excel
- df.to_csv

24.11 Copies and views

- `inplace`

25 Data cleaning

25.1 Handling Missing Data

- np.nan, pd.NA
- pd.isnull(), np.isnan()
- dropna, fillna

Example 25.1.

```
import pandas as pd
import numpy as np

data = pd.DataFrame([[1., 6.5, 3.], [1., np.nan, np.nan],
                     [np.nan, np.nan, np.nan], [np.nan, 6.5, 3.]])
cleaned = data.dropna()
cleanedrow = data.dropna(how='all')
data
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
cleaned
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2
0	1.0	6.5	3.0

```
cleanedrow
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
3	NaN	6.5	3.0

```
data[4] = np.nan
cleaned1 = data.dropna(axis=1, how='all')
cleanedthresh = data.dropna(thresh=2)
data
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

```
cleaned1
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2
0	1.0	6.5	3.0
1	1.0	NaN	NaN
2	NaN	NaN	NaN
3	NaN	6.5	3.0

```
cleanedthresh
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
3	NaN	6.5	3.0	NaN

```
fill0 = data.fillna(0)
filldict = data.fillna({1: 0.5, 2: -0.1})
data
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	NaN	NaN	NaN
2	NaN	NaN	NaN	NaN
3	NaN	6.5	3.0	NaN

```
fill0
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2	4
0	1.0	6.5	3.0	0.0
1	1.0	0.0	0.0	0.0
2	0.0	0.0	0.0	0.0
3	0.0	6.5	3.0	0.0

```
filldict
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0	1	2	4
0	1.0	6.5	3.0	NaN
1	1.0	0.5	-0.1	NaN
2	NaN	0.5	-0.1	NaN
3	NaN	6.5	3.0	NaN

25.2 Data Transformation

- `.duplicated()`, `drop_duplicates()`

Example 25.2

```
import numpy as np
import pandas as pd

data = pd.DataFrame({'k1': ['one', 'two'] * 3 + ['two'],
                     'k2': [1, 1, 2, 3, 3, 4, 4]})
data.drop_duplicates(['k1'], keep='last')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	k1	k2
4	one	3
6	two	4

- pd.Series.map(), pd.DataFrame.apply()

Example 25.3.

```
import pandas as pd
import numpy as np

data = pd.DataFrame({'food': ['bacon', 'pulled pork', 'bacon',
                      'Pastrami', 'corned beef', 'Bacon',
                      'pastrami', 'honey ham', 'nova lox'],
                      'ounces': [4, 3, 12, 6, 7.5, 8, 3, 5, 6]})

meat_to_animal = {
    'bacon': 'pig',
    'pulled pork': 'pig',
    'pastrami': 'cow',
    'corned beef': 'cow',
    'honey ham': 'pig',
    'nova lox': 'salmon'
    }

data['animal'] = data['food'].str.lower().map(meat_to_animal)

data['food'].map(lambda x: meat_to_animal[x.lower()])
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	food
0	pig
1	pig
2	pig
3	cow
4	cow
5	pig
6	cow
7	pig
8	salmon

- replace

- `rename`
- `describe`
- `permutation`
- `sample`
- dummy variables

25.3 Example: Movies

Below we explore the MovieLens 1M datasets. You may download it from this link.

```
import pandas as pd
import numpy as np
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('assests/datasets/movies.dat', sep='::',
                       header=None, names=mnames, engine="python",
                       encoding='ISO-8859-1')

all_genres = list()
movies['genres'].map(lambda x: all_genres.extend(x.split('|')))

genres = pd.unique(all_genres)

dummies = pd.DataFrame(np.zeros((len(movies), len(genres))), columns=genres)

for i, gen in enumerate(movies.genres):
    indices = dummies.columns.get_indexer(gen.split('|'))
    dummies.iloc[i, indices] = 1

movies_windic = movies.join(dummies.add_prefix('Genre_'))
```

25.4 String Manipulation

The key idea in this section is that, all methods in `pd.Series.str` will be applied to each entry of the Series.

Example 25.4.

```
import pandas as pd
import numpy as np
```

```
s = pd.Series(["A ", " B ", "C", "Aaba", " Baca ", np.nan, "CABA", "dog", "cat"])

s.str.lower()
s.str.split('a')
s.str.len()
s.str.strip()
s.str.replace("A", '1')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
0	1
1	B
2	C
3	1aba
4	Baca
5	NaN
6	C1B1
7	dog
8	cat

Example 25.5. We could also use `.str` to play with column names and row indexes.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.randn(3, 2),
                  columns=[" Column A ", " Column B "], index=range(3))

df.columns = df.columns.str.strip().str.lower().str.replace(' ', '_')
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	column_a	column_b
0	0.290302	1.738902
1	1.718592	0.028324
2	0.206837	0.588591

25.5 Regular expression

Regular expressions provide a flexible way to search or match string patterns in text. A single expression, commonly called a *regex*, is a string formed according to the regular expression language. Python's built-in `re` module is responsible for applying regular expressions to strings.

For details of the regular expression language in Python, please read the official documents from here. There are also many great websites for learning regex. This is one example.

We will briefly mentioned a few rules here.

- `.`: matches any character except a newline.
- `\d`: matches any digit. It is the same as `[0-9]`.
- `\w`: matches any alphabatic or numeric character. It is the same as `[a-zA-Z0-9_]`.
- `\s`: matches any whitespaces. It is the same as `[\t\n\r\f\v]`.
- `*`: Causes the resulting RE to match 0 or more repetitions of the preceding RE, as many repetitions as are possible.
- `+`: Causes the resulting RE to match 1 or more repetitions of the preceding RE, as many repetitions as are possible.
- `?`: Causes the resulting RE to match 0 or 1 repetitions of the preceding RE.
- `*?`, `+?`, `??`: The `*`, `+`, and `?` qualifiers are all greedy; they match as much text as possible. Adding `?` after the qualifier makes it perform the match in non-greedy or minimal fashion; as few characters as possible will be matched.
- `{m}`: Specifies that exactly m copies of the previous RE should be matched.
- `{m,n}`: Causes the resulting RE to match from `m` to `n` repetitions of the preceding RE, attempting to match as many repetitions as possible.
- `{m,n}?`: Causes the resulting RE to match from `m` to `n` repetitions of the preceding RE, attempting to match as few repetitions as possible.
- `[]`: Used to indicate a set of characters.
- `()`: set groups.

Example 25.6.

```
text = "foo bar\t baz \tqux"
```

```
pattern = '\s+'
regex = re.compile(pattern)
regex.split(text)
```

```
['foo', 'bar', 'baz', 'qux']
```

- `.match()`
- `.search()`
- `.findall()`
- `.split()`
- `.sub()`

We can use `()` to specify groups, and use `.groups()` to get access to the results.

Example 25.7.

```
pattern = r'([A-Z0-9._%+-]+)@([A-Z0-9.-]+)\.([A-Z]{2,4})'
regex = re.compile(pattern, flags=re.IGNORECASE)
m = regex.match('wesm@bright.net')
m.groups()
```

```
('wesm', 'bright', 'net')
```

To use regex to DataFrame and Series, you may directly apply `.match`, `.findall`, `.replace` after `.str`, with the regex pattern as one of the arguments.

`.extract` is a method that is not from `re`. It is used to extract the matched groups and make them as a DataFrame.

Example 25.8.

```
import pandas as pd
import numpy as np
mnames = ['movie_id', 'title', 'genres']
movies = pd.read_table('assests/datasets/movies.dat', sep='::',
                       header=None, names=mnames, engine="python",
                       encoding='ISO-8859-1')

pattern = r'([a-zA-Z0-9_\s,.?:;\']+)\((\d{4})\)'
movies = movies.join(movies.title.str.extract(pattern).rename(columns={0: 'movie title', 1
```

26 Data Wrangling

26.1 Hierarchical indexing

Pandas support a more complex indexing system, that the index may have multiple levels. See the following example.

```
import pandas as pd
import numpy as np

data = pd.Series(np.random.randn(9),
                 index = [['a', 'a', 'a', 'b', 'b', 'c', 'c', 'd', 'd'],
                          [1, 2, 3, 1, 2, 3, 1, 2, 3]])
data
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

		0
a	1	1.821327
	2	1.197344
	3	-0.650698
b	1	-0.321392
	2	-0.389620
c	3	-1.434650
	1	-1.013278
d	2	2.409614
	3	-1.204232

You may look at the Series using different levels of indexes.

```
data['a']
```

	0
1	1.821327
2	1.197344
3	-0.650698

```
data.loc[:, 2]
```

	0
a	1.197344
b	-0.389620
d	2.409614

You may use groupby to group by levels and do calculations related to levels. More `.groupby()` will be discussed in the next section.

```
data.groupby(level=1).sum()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	0
1	0.486658
2	3.217338
3	-3.289581

From the example above, you may notice that the 2-level hierarchical indexing for a Series works very similar to a DataFrame. In fact, you may translate it back and forth between a 2-level indexing Series and a DataFrame.

```
df = data.unstack()
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	1	2	3
a	1.821327	1.197344	-0.650698
b	-0.321392	-0.389620	NaN
c	-1.013278	NaN	-1.434650
d	NaN	2.409614	-1.204232

```
df.stack()
```

		0
a	1	1.821327
	2	1.197344
	3	-0.650698
b	1	-0.321392
	2	-0.389620
c	1	-1.013278
	3	-1.434650
d	2	2.409614
	3	-1.204232

For DataFrame the index for both axes can be multiindex. The usual indexing way can be used if you want to start from the first level of the index. The more specific method to extract data is `.xs`.

Example 26.1.

```
Example 26.1.
import pandas as pd

df1 = pd.DataFrame(
    {
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    },
    index=[0, 1, 2, 3],
)

df2 = pd.DataFrame(
    {
        "A": ["A4", "A5", "A6", "A7"],
        "B": ["B4", "B5", "B6", "B7"],
```

```
        "C": ["C4", "C5", "C6", "C7"],
        "D": ["D4", "D5", "D6", "D7"],
    },
    index=[4, 5, 6, 7],
)

df = pd.concat([df1, df2], keys=['x', 'y'])
```

```
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

		A	B	C	D
x	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
	5	A5	B5	C5	D5
	6	A6	B6	C6	D6
	7	A7	B7	C7	D7

```
df['A']
```

		A
x	0	A0
	1	A1
	2	A2
	3	A3
y	4	A4
	5	A5
	6	A6
	7	A7

```
df.loc['x']
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa
```

```
In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df.loc['x',3]
```

	x
	3
A	A3
B	B3
C	C3
D	D3

```
df.xs(3, level=1, drop_level=False)
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

		A	B	C	D
x	3	A3	B3	C3	D3

26.2 Combining and Merging Datasets

26.2.1 `merge()`

Merge combines datasets by linking rows using one or more keys. This is from relational databases (e.g., SQL-based).

Here are some examples.

Example 26.2.

```
import pandas as pd
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df2 = pd.DataFrame({'key': ['a', 'b', 'd'], 'data2': range(3)})
```

The two DataFrames are displayed as follows.

```
df1
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	key	data1
0	b	0
1	b	1
2	a	2
3	c	3
4	a	4
5	a	5
6	b	6

```
df2
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	key	data2
0	a	0
1	b	1
2	d	2

```
pd.merge(df1, df2, on='key')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	key	data1	data2
0	b	0	1
1	b	1	1
2	b	6	1
3	a	2	0
4	a	4	0
5	a	5	0

If the column names are different in each object, you can specify them separately.

```
df3 = pd.DataFrame({'lkey': ['b', 'b', 'a', 'c', 'a', 'a', 'b'],
                    'data1': range(7)})
df4 = pd.DataFrame({'rkey': ['a', 'b', 'd'],
                    'data2': range(3)})
pd.merge(df3, df4, left_on='lkey', right_on='rkey')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	lkey	data1	rkey	data2
0	b	0	b	1
1	b	1	b	1
2	b	6	b	1
3	a	2	a	0
4	a	4	a	0
5	a	5	a	0

By default `merge` does an inner join, that the keys in the result are the interesection found in both tables. Below are different types of `merge`. To specify the method for merge, the option is `how`.

- `inner`
- `left`
- `right`
- `outer`

Let's see the following examples.

```
df1 = pd.DataFrame({'Key': [1, 2], 'A': [0, 2], 'B': [1, 3]})
df1
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	A	B
0	1	0	1
1	2	2	3

```
df2 = pd.DataFrame({'Key': [1, 3], 'C': [0, 2], 'D': [1, 3]})
df2
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	C	D
0	1	0	1
1	3	2	3

```
pd.merge(df1, df2, on='Key', how='inner')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	A	B	C	D
0	1	0	1	0	1

```
pd.merge(df1, df2, on='Key', how='outer')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	A	B	C	D
0	1	0.0	1.0	0.0	1.0
1	2	2.0	3.0	NaN	NaN
2	3	NaN	NaN	2.0	3.0

```
pd.merge(df1, df2, on='Key', how='left')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	A	B	C	D
0	1	0	1	0.0	1.0
1	2	2	3	NaN	NaN

```
pd.merge(df1, df2, on='Key', how='right')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	Key	A	B	C	D
0	1	0.0	1.0	0	1
1	3	NaN	NaN	2	3

Note

If a key combination appears more than once in both tables, the resulting table will have the Cartesian product of the associated data. Here is a very basic example with one unique key combination.

```
df1 = pd.DataFrame({'key': ['b', 'b', 'a', 'c', 'a', 'b'],
                    'data1': range(6)})
df2 = pd.DataFrame({'key': ['a', 'b', 'a', 'b', 'd'],
                    'data2': range(5)})
pd.merge(df1, df2, on='key', how='left')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: Future
```

```
In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `
```

	key	data1	data2
0	b	0	1.0
1	b	0	3.0
2	b	1	1.0
3	b	1	3.0
4	a	2	0.0
5	a	2	2.0
6	c	3	NaN
7	a	4	0.0
8	a	4	2.0
9	b	5	1.0
10	b	5	3.0

Note

If the merge keys in a DataFrame is in its index instead of column(s), we could pass `left_index=True` or `right_index=True` or both instead of setting `left_on`/`right_on`/`on`.

Example 26.3. If we want to really create a Cartesian product, we may use the option `how='cross'`. For example, we would like to generate a deck of cards, we may use the following codes.

```
suit = pd.DataFrame({'suit': ['spades', 'hearts', 'clubs', 'diamonds']})
face = pd.DataFrame({'face': list(range(1, 14))})
deck = pd.merge(suit, face, how='cross')
```

26.2.2 `concat()`

The `concat()` function (in the main pandas namespace) performs concatenation operations along an axis while performing optional set logic (union or intersection) of the indexes (if any) on the other axes.

```
import pandas as pd
```

```
df1 = pd.DataFrame(
    {
        "A": ["A0", "A1", "A2", "A3"],
        "B": ["B0", "B1", "B2", "B3"],
        "C": ["C0", "C1", "C2", "C3"],
        "D": ["D0", "D1", "D2", "D3"],
    },
    index=[0, 1, 2, 3],
)

df2 = pd.DataFrame(
    {
        "A": ["A4", "A5", "A6", "A7"],
        "B": ["B4", "B5", "B6", "B7"],
        "C": ["C4", "C5", "C6", "C7"],
        "D": ["D4", "D5", "D6", "D7"],
    },
    index=[4, 5, 6, 7],
)

df3 = pd.DataFrame(
    {
        "A": ["A8", "A9", "A10", "A11"],
        "B": ["B8", "B9", "B10", "B11"],
        "C": ["C8", "C9", "C10", "C11"],
        "D": ["D8", "D9", "D10", "D11"],
    },
    index=[8, 9, 10, 11],
)

pd.concat([df1, df2, df3], keys=['x', 'y', 'z'])
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

		A	B	C	D
x	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
	5	A5	B5	C5	D5
	6	A6	B6	C6	D6
	7	A7	B7	C7	D7
z	8	A8	B8	C8	D8
	9	A9	B9	C9	D9
	10	A10	B10	C10	D10
	11	A11	B11	C11	D11

The default way of `pd.concat()` is vertically. Note that it will check the column names. If the column names don't match, new columns will be created and `nan` values will be assigned.

If you want to concatenate the DataFrame horizontally you need to add `axis=1` option. Similarly, row index will be checked before concatenating. See the following example.

Example 26.4.

```
pd.concat([df1, df2, df3], axis=1)
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	A	B	C	D	A	B	C	D	A	B	C	D
0	A0	B0	C0	D0	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	A1	B1	C1	D1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
2	A2	B2	C2	D2	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
3	A3	B3	C3	D3	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	NaN	NaN	NaN	NaN	A4	B4	C4	D4	NaN	NaN	NaN	NaN
5	NaN	NaN	NaN	NaN	A5	B5	C5	D5	NaN	NaN	NaN	NaN
6	NaN	NaN	NaN	NaN	A6	B6	C6	D6	NaN	NaN	NaN	NaN
7	NaN	NaN	NaN	NaN	A7	B7	C7	D7	NaN	NaN	NaN	NaN
8	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A8	B8	C8	D8
9	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A9	B9	C9	D9
10	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A10	B10	C10	D10
11	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	A11	B11	C11	D11

Example 26.5. Consider the deck example from Example 26.3. This time we would like to use `pd.concat()` to get the result.

```
suitlist = ['spades', 'hearts', 'clubs', 'diamonds']
facelist = list(range(1, 14))
decklist = [pd.DataFrame({'suit': suit, 'face': facelist}) for suit in suitlist]
deck = pd.concat(decklist, ignore_index=True)
```

27 Data Aggregation and Group Operations

27.1 split-apply-combine model

We would like to apply group operations based on the split-apply-combine model.

- In the first stage of the process, data contained in a pandas object is *split* into groups based on one or more keys that you provide. We then use `.groupby(keys)` to perform the split step. The result is a grouped `groupby` object.
- Once this is done, a function is *applied* to each group, producing a new value.
- Finally the results of all those function applications are combined into a result object. We may apply groupby functions directly as methods to groupby objects.The result is the combined result object.

Example 27.1.

```
import pandas as pd
df = pd.DataFrame({'key1' : ['a', 'a', 'b', 'b', 'a'],
                   'key2' : ['one', 'two', 'one', 'two', 'one'],
                   'data1' : np.random.randn(5),
                   'data2' : np.random.randn(5)})
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	key1	key2	data1	data2
0	a	one	1.303673	-0.545660
1	a	two	0.262172	-1.539773
2	b	one	-0.162941	-0.497047
3	b	two	0.973011	-0.421686
4	a	one	-0.209807	1.132354

Now we want to group `data1` in `df` by `key1`.

```
grouped = df['data1'].groupby(df['key1'])
grouped
```

```
<pandas.core.groupby.generic.SeriesGroupBy object at 0x000002407962A910>
```

What we get is a groupby object and we could apply group functions to it.

The method to look at each group is `.get_group`.

```
grouped.get_group('a')
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	data1
0	1.303673
1	0.262172
4	-0.209807

We may directly apply some group functions to the groupby object.

```
grouped.mean()
```

key1	data1
a	0.452013
b	0.405035

```
grouped.size()
```

key1	data1
a	3
b	2

We could iterate over groups.

```
  for name, group in grouped:
      print('name', name)
      print('group', group)
```

```
name a
group 0    1.303673
1    0.262172
4   -0.209807
Name: data1, dtype: float64
name b
group 2   -0.162941
3    0.973011
Name: data1, dtype: float64
```

We could convert the group object into list and dictionary.

```
  list(grouped)
```

```
[('a',
  0    1.303673
  1    0.262172
  4   -0.209807
  Name: data1, dtype: float64),
 ('b',
  2   -0.162941
  3    0.973011
  Name: data1, dtype: float64)]
```

```
  dict(list(grouped))
```

```
{'a': 0    1.303673
 1    0.262172
 4   -0.209807
 Name: data1, dtype: float64,
 'b': 2   -0.162941
 3    0.973011
 Name: data1, dtype: float64}
```

27.2 More aggregation functions

- `.describe()`
- `.count()`
- `.sum()`
- `.mean()`
- `.median`
- `.std()`, `.var()`
- `.min()`, `.max()`
- `.prod()`
- `first()`, `.last()`
- `.agg()`

27.3 Some examples

Example 27.2. Consider the following DataFrame.

```
import pandas as pd
df = pd.DataFrame({'location': ['East', 'East', 'East', 'East',
                                'West', 'West', 'West', 'West'],
                   'data': np.random.randn(8)},
                   index=['Ohio', 'New York', 'Vermont', 'Florida',
                          'Oregon', 'Nevada', 'California', 'Idaho'])
df.loc[['Vermont', 'Nevada', 'Idaho'], 'data'] = np.nan
```

We would like to fill in NA values with the mean from each group.

```
df.groupby('location').apply(lambda x: x.fillna(x.mean()))
```

```
C:\Users\Xinli\AppData\Local\Temp\ipykernel_6716\2040193686.py:1: FutureWarning:

Dropping of nuisance columns in DataFrame reductions (with 'numeric_only=None') is deprecated

C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	location	data
Ohio	East	-2.201895
New York	East	-0.397776
Vermont	East	-1.152346
Florida	East	-0.857368
Oregon	West	1.454232
Nevada	West	0.623059
California	West	-0.208113
Idaho	West	0.623059

We could also fill in NA values with predefined values, similar to the non-groupby case.

```
df.groupby('location').apply(lambda x: x.fillna({'East': 0.1,
                                                 'West': -0.5}[x.name]))
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	location	data
Ohio	East	-2.201895
New York	East	-0.397776
Vermont	East	0.100000
Florida	East	-0.857368
Oregon	West	1.454232
Nevada	West	-0.500000
California	West	-0.208113
Idaho	West	-0.500000

28 Exercises

Most problems are based on [7].

Exercise 28.1. Please use the following code to generate a series `ser`, and then finish the following tasks.

```
import pandas as pd
import numpy as np


mylist = list('abcedfghijklmnopqrstuvwxyz')
myarr = np.arange(26)
mydict = dict(zip(mylist, myarr))
ser = pd.Series(mydict)
```

1. Convert the series `ser` into a dataframe `df` with its index as another column on the dataframe.
2. Pick the two columns of `df` and set them into two serieses `ser1` and `ser2`.
3. Combine two series `ser1` and `ser2` to form a new dataframe `newdf`, and name their columns `ser1` and `ser2`.

Exercise 28.2. Consider two serieses `ser1` and `ser2`. You may use the following `ser1` and `ser2` as an example. The output of each questions below should be a series. You may want to learn the following commands:

- `np.union1d()`
- `np.intersect1d()`
- `np.isin()`

```
import pandas as pd

ser1 = pd.Series([1, 2, 3, 4, 5])
```

```
ser2 = pd.Series([4, 5, 6, 7, 8])
```

1. Find all the elements from `ser1` that are also in `ser2`.
2. Find all the elements from `ser2` that are also in `ser1`.
3. From `ser1` remove items present in `ser2`.
4. Find the union of `ser1` and `ser2`.
5. Find the intersection of `ser1` and `ser2`.
6. Find all the elemetns that are in either `ser1` or `ser2`, but not both.

Exercise 28.3 (Some statistics)**.** Please check the following commands and answer the following questions.

- `np.percentile()`

How to get the minimum, 25th percentile, median, 75th, and max of a numeric series? You may use the following Series as an example.

```
import pandas as pd
ser = pd.Series(np.random.normal(10, 5, 25))
```

Exercise 28.4. Please use `pd.Series.value_counts()` to calculte the frequency counts of each unique value of the following Series.

```
import pandas as pd
import numpy as np
ser = pd.Series(np.take(list('abcdefgh'), np.random.randint(8, size=30)))
```

Exercise 28.5. Please keep the top 2 most frequent items of `ser` as it is and replace everything else as `Other`.

```
import pandas as pd
import numpy as np
ser = pd.Series(np.take(list('abcdefgh'), np.random.randint(8, size=30)))
```

Exercise 28.6. Please use `pd.cut` or `pd.qcut` to bin the Series `ser` into 10 equal deciles. You may use the following `ser` as an example.

```
import pandas as pd
ser = pd.Series(np.random.random(20))
```

Exercise 28.7. Consider the Series `ser`:

```
import pandas as pd
import numpy as np
ser = pd.Series(np.random.randint(1, 10, 7))
```

Find the positions of numbers that are multiples of 3 from ser.

Exercise 28.8. Compute the mean of `weights` of each `fruit`.

```
import pandas as pd
fruit = pd.Series(np.random.choice(['apple', 'banana', 'carrot'], 10))
weights = pd.Series(np.linspace(1, 10, 10))
df = pd.DataFrame({'fruit': fruit, 'weights': weights})
```

Exercise 28.9. Consider the following DataFrame.

```
import pandas as pd
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/Cars93_miss.cs
```

1. Check if `df` has any missing values.
2. Please count the number of missing values in each column.
3. Please replace all missing values in `Min.Price` and `Max.Price` with their mean respectively.

Exercise 28.10. Replace the spaces in `my_str = 'dbc deb abed gade'` with the least frequent character.

Exercise 28.11. Suppress scientific notations like `e-03` in `df` and print up to 4 numbers after decimal.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.random(4)**10, columns=['random'])
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	random
0	1.050851e-02
1	7.247888e-02
2	4.447110e-09
3	3.983586e-06

Exercise 28.12. Format the values in column `random` of `df` as percentages.

```
import pandas as pd
import numpy as np
df = pd.DataFrame(np.random.random(4), columns=['random'])
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	random
0	0.435058
1	0.554946
2	0.864754
3	0.665629

Exercise 28.13 (Regular expressions)**.** Please use regular expressions to finish the following tasks.

1. Match a string that has an `a` followed by zero or more `b`'s.
2. Match a string that has an `a` followed by one or more `b`'s.
3. Match a string that has an `a` followed by zero or one `b`.
4. Match a string that has an `a` followed by three `b`'s.

Exercise 28.14 (More regex)**.** Find all words starting with `a` or `e` in a given string:

```
text = "The following example creates an ArrayList with a capacity of 50 elements. Four el
```

Exercise 28.15 (More regex)**.** Write a Python code to extract year, month and date from a `url1`:

```
url1= "https://www.washingtonpost.com/news/football-insider/wp/2016/09/02/odell-beckhams-f
```

Exercise 28.16 (More regex)**.** Please use regex to parse the following str to create a dictionary.

```
text = r'''
{
    name: Firstname Lastname;
    age: 100;
    salary: 10000
}
'''
```

Exercise 28.17. Consider the following DataFrame.

```
data = [['Evert van Dijk', 'Carmine-pink, salmon-pink streaks, stripes, flecks.  Warm pink
        ['Every Good Gift', 'Red.  Flowers velvety red.  Moderate fragrance.  Average diam
        ['Evghenya', 'Orange-pink.  75 petals.  Large, very double bloom form.  Blooms in
```

```
        ['Evita', 'White or white blend.  None to mild fragrance.  35 petals.  Large, full
        ['Evrathin', 'Light pink. [Deep pink.]  Outer petals white. Expand rarely.  Mild f
        ['Evita 2', 'White, blush shading.  Mild, wild rose fragrance.  20 to 25 petals.

df = pd.DataFrame(data, columns = ['NAME', 'BLOOM'])
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	NAME	BLOOM
0	Evert van Dijk	Carmine-pink, salmon-pink streaks, stripes, fl...
1	Every Good Gift	Red. Flowers velvety red. Moderate fragrance...
2	Evghenya	Orange-pink. 75 petals. Large, very double b...
3	Evita	White or white blend. None to mild fragrance....
4	Evrathin	Light pink. [Deep pink.] Outer petals white. ...
5	Evita 2	White, blush shading. Mild, wild rose fragran...

Please use regex methods to find all the `()` in each columns.

Exercise 28.18. Get the last two rows of `df` whose row sum is greater than 100.

```
import pandas as pd
df = pd.DataFrame(np.random.randint(10, 40, 60).reshape(-1, 4))
```

Exercise 28.19. The groupby object `df_grouped` is given below.

```
import pandas as pd
import numpy as np
df = pd.DataFrame({'fruit': ['apple', 'banana', 'orange'] * 3,
                   'price': np.random.rand(9),
                   'taste': np.random.randint(0, 11, 9)})

df_grouped = df.groupby(['fruit'])
```

1. Get the group belonging to `apple` as a DataFrame.
2. Find the second largest value of `taste` for `banana`.
3. Compute the mean `price` for every `fruit`.

Exercise 28.20. Join `df1` and `df2` by `fruit`/`pazham` and `weight`/`kilo`.

```
df1 = pd.DataFrame({'fruit': ['apple', 'banana', 'orange'] * 3,
                    'weight': ['high', 'medium', 'low'] * 3,
                    'price': np.random.randint(0, 15, 9)})

df2 = pd.DataFrame({'pazham': ['apple', 'orange', 'pine'] * 2,
                    'kilo': ['high', 'low'] * 3,
                    'price': np.random.randint(0, 15, 6)})
```

29 Projects

Exercise 29.1. Extract the valid emails from the series `emails`. The regex `pattern` for valid emails is provided as reference.

```
import pandas as pd
emails = pd.Series(['buying books at amazom.com',
                    'rameses@egypt.com',
                    'matt@t.co',
                    'narendra@modi.com'])
pattern = '[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\\.[A-Za-z]{2,4}'
```

Exercise 29.2. Consider the following DataFrame.

```
import pandas as pd
import numpy as np
df = pd.read_csv('https://raw.githubusercontent.com/selva86/datasets/master/Cars93_miss.cs
```

1. Replace `NaN` with string `missing` in columns `Manufacturer`, `Model` and `Type`.
2. Create an index as a combination of these three columns.

Exercise 29.3. Given the following DataFrame.

```
import pandas as pd
df = pd.DataFrame({
    'name': ['James', 'Jane', 'Melissa', 'Ed', 'Neil'],
    'age': [30, 40, 32, 67, 43],
    'score': ['90%', '95%', '100%', '82%', '87%'],
    'age_missing_data': [30, 40, 32, 67, None],
    'income':[100000, 80000, 55000, 62000, 120000]
```

```
})
```

- Please use `.map` to create a new column `numeric_score` whose value is the number version of `score`.
- Please use `.apply` to create a new column `numeric_score` whose value is the number version of `score`.

Exercise 29.4. From `ser = pd.Series(['Apple', 'Orange', 'Plan', 'Python', 'Money'])`, find the words that contain at least 2 vowels.

Exercise 29.5. Please download the given file with sample emails, and use the following code to load the file and save it to a string `content`.

```
with open('assests/datasets/test_emails.txt', 'r') as f:
    content = f.read()
```

Please use regex to play with `content`.

1. Get all valid email address in `content`, from both the header part or the body part.
2. There are two emails in `content`. Please get the sender's email and the receiver's email from `content`.
3. Please get the sender's name.
4. Please get the subject of each email.

Exercise 29.6. The following DataFrame is given.

```
import pandas as pd
df = pd.DataFrame(["STD, City    State",
                   "33, Kolkata    West Bengal",
                   "44, Chennai    Tamil Nadu",
                   "40, Hyderabad    Telengana",
                   "80, Bangalore    Karnataka"],
                   columns=['row'])
```

1. Split the columns into a list with 3 entries.
2. Make the first row (row 0) into a header.
3. Create a new DataFrame out of the data.

Part VI

Visualization

The main reference for this Chapter is [2].

30 matplotlib.pyplot

matplotlib is a modern and classic plot library. Its main features are inspired by MATLAB. In this book we mostly use pyplot package from matplotlib. We use the following import convention:

```
import matplotlib.pyplot as plt
```

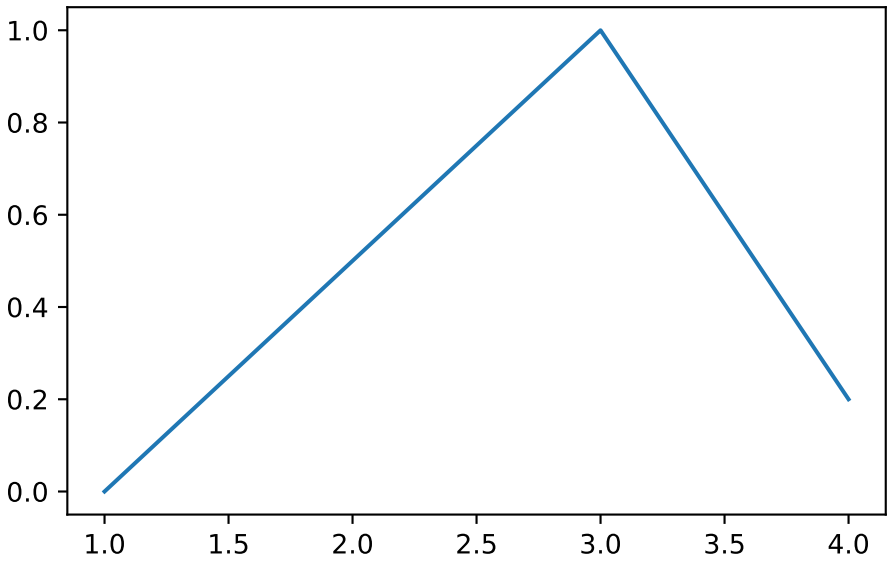
30.1 matplotlib interface

matplotlib has two major application interfaces, or styles of using the library:

- An explicit Axes interface that uses methods on a Figure or Axes object to create other Artists, and build a visualization step by step. You may treat this Figure object as a canvas, and Axes as plots on a canvas. There might be one or more plots on one canvas. This has also been called an *object-oriented* interface.
- An implicit pyplot interface that keeps track of the last Figure and Axes created, and adds Artists to the object it thinks the user wants.

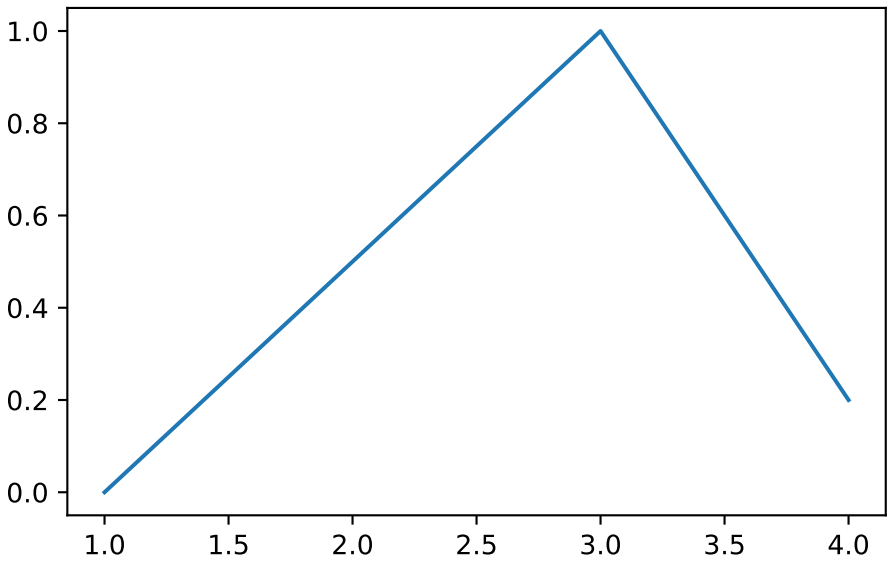
Here is an example of an explicit interface.

```
fig = plt.figure()
ax = fig.subplots()
ax.plot([1, 2, 3, 4], [0, 0.5, 1, 0.2])
```



Here is an example of an implicit interface.

```
plt.plot([1, 2, 3, 4], [0, 0.5, 1, 0.2])
```



> **Note**
>
> If the plot is not shown, you may want to type `plt.show()` to force the plot being rendered. However, to make `plt.show()` work is related to switching `matplotlib` backends, and is sometimes very complicated.

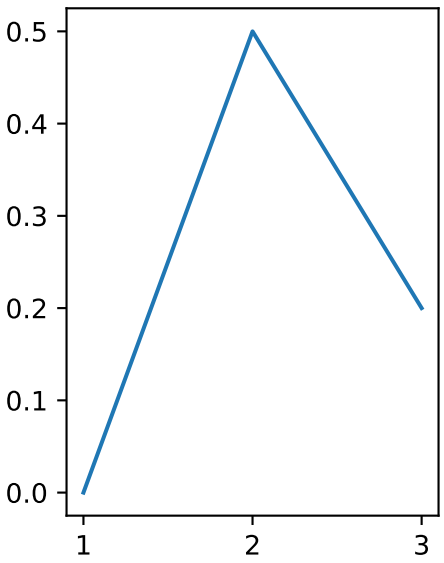
The purpose to explicitly use `fig` and `ax` is to have more control over the configurations. The first important configuration is subplots.

- `.subplot()`
- `.subplots()`
- `.add_subplot()`

Please see the following examples.

Example 30(1.

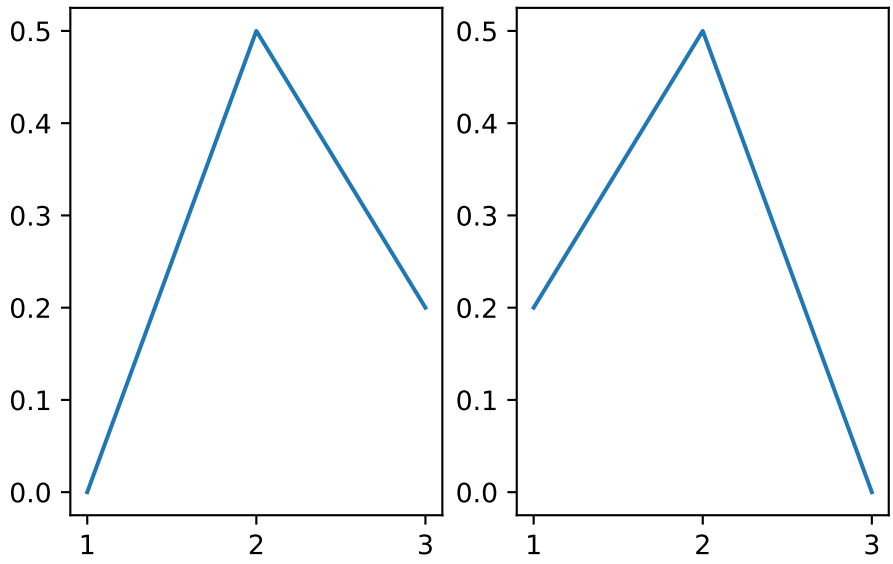
```
plt.subplot(1, 2, 1)
plt.plot([1, 2, 3], [0, 0.5, 0.2])
```



Example 30(2.

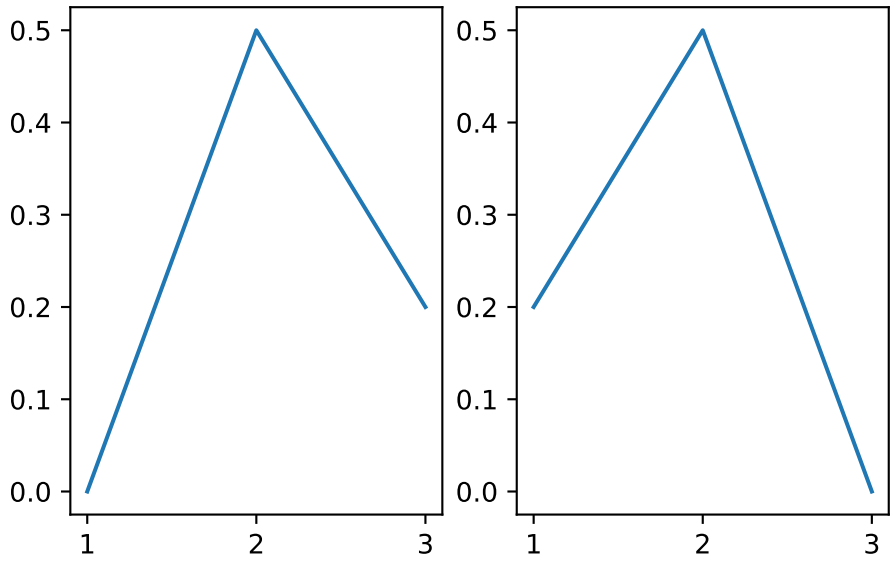
```
plt.subplot(1, 2, 1)
plt.plot([1, 2, 3], [0, 0.5, 0.2])
```

```
plt.subplot(1, 2, 2)
plt.plot([3, 2, 1], [0, 0.5, 0.2])
```



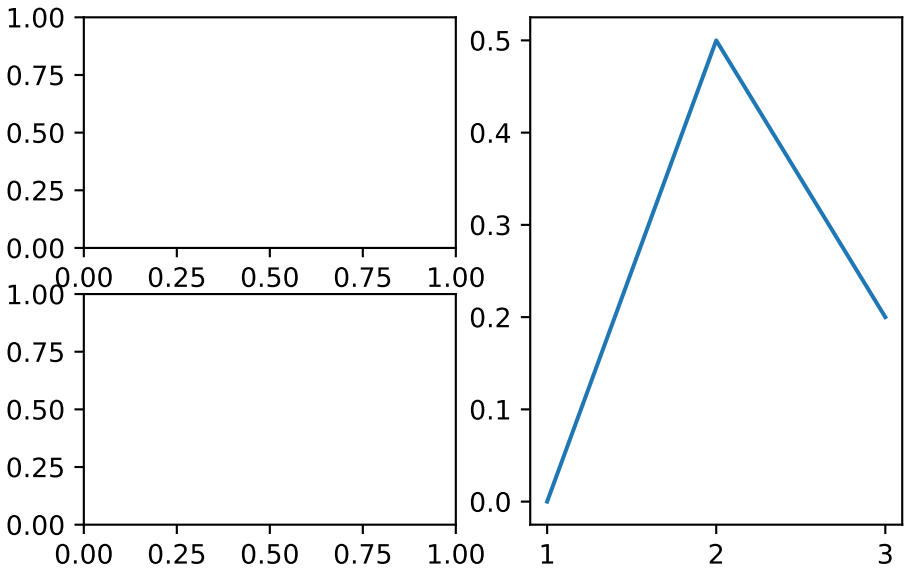
Example 30.3:

```
fig, axs = plt.subplots(1, 2)
axs[0].plot([1, 2, 3], [0, 0.5, 0.2])
axs[1].plot([3, 2, 1], [0, 0.5, 0.2])
```



Example 304

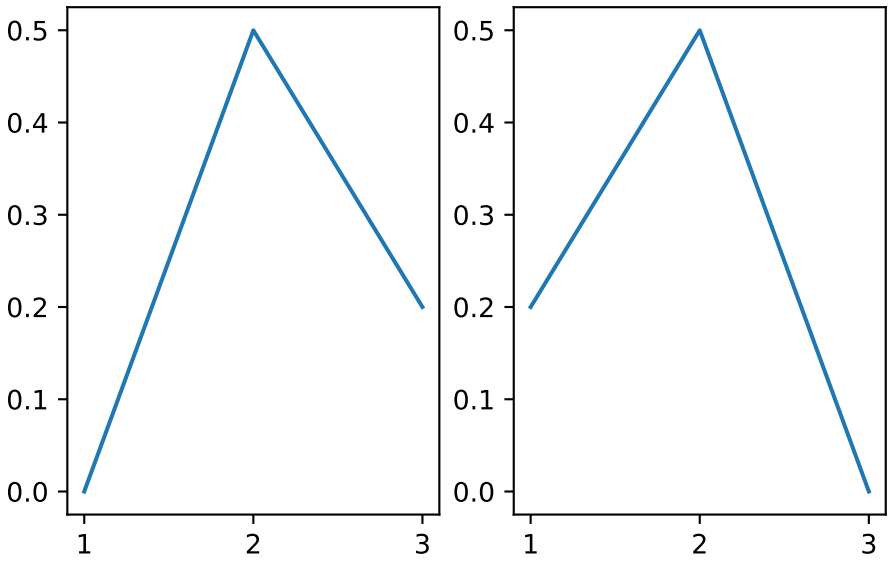
```
Expand import numpy as np
fig = plt.figure()
ax1 = fig.add_subplot(2, 2, 1)
ax2 = fig.add_subplot(2, 2, 3)
ax3 = fig.add_subplot(1, 2, 2)

ax3.plot([1, 2, 3], [0, 0.5, 0.2])
```



The auguments `2, 2, 1` means that we split the figure into a `2x2` grid and the axis `ax1` is in the 1st position. The rest is understood in the same way.

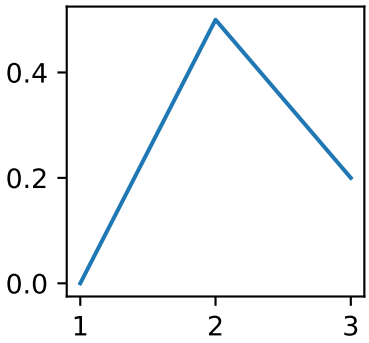
Example 30.5. If you don't explicitly initialize `fig` and `ax`, you may use `plt.gcf()` and `plt.gca()` to get the handles for further operations.

```
plt.subplot(1, 2, 1)
ax = plt.gca()
ax.plot([1, 2, 3], [0, 0.5, 0.2])

plt.subplot(1, 2, 2)
ax = plt.gca()
ax.plot([3, 2, 1], [0, 0.5, 0.2])
```



The purpose to explicitly use `fig` and `ax` is to have more control over the configurations. For example, when generate a `figure` object, we may use `figsize=(3, 3)` as an option to set the figure size to be `3x3`. `dpi` is another commonly modified option.

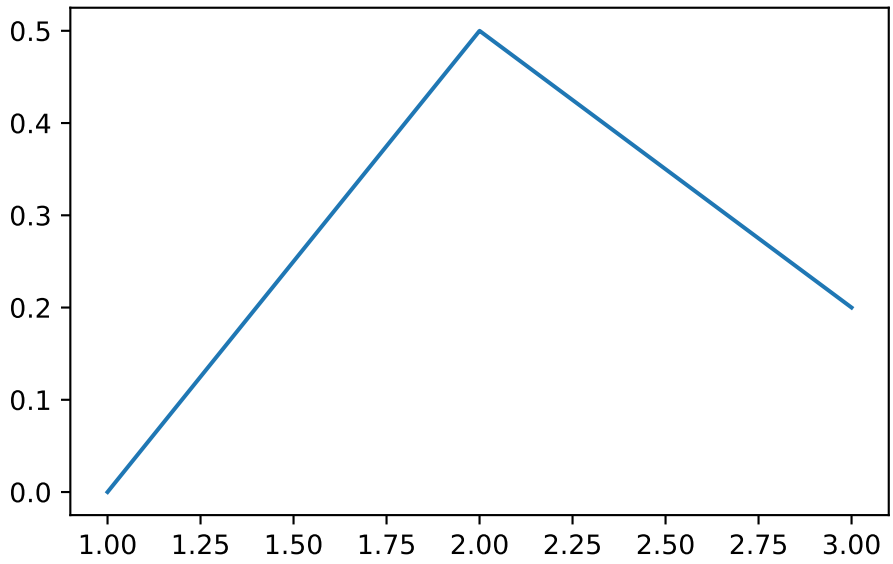
```
fig = plt.figure(figsize=(2, 2), dpi=50)
plt.plot([1, 2, 3], [0, 0.5, 0.2])
```



If you would like to change this setting later, you may use the following command before plotting.

```
fig.set_size_inches(10, 10)
fig.set_dpi(300)
```

```
plt.plot([1, 2, 3], [0, 0.5, 0.2])
```



You may use `fig.savefig('filename.png')` to save the image into a file.

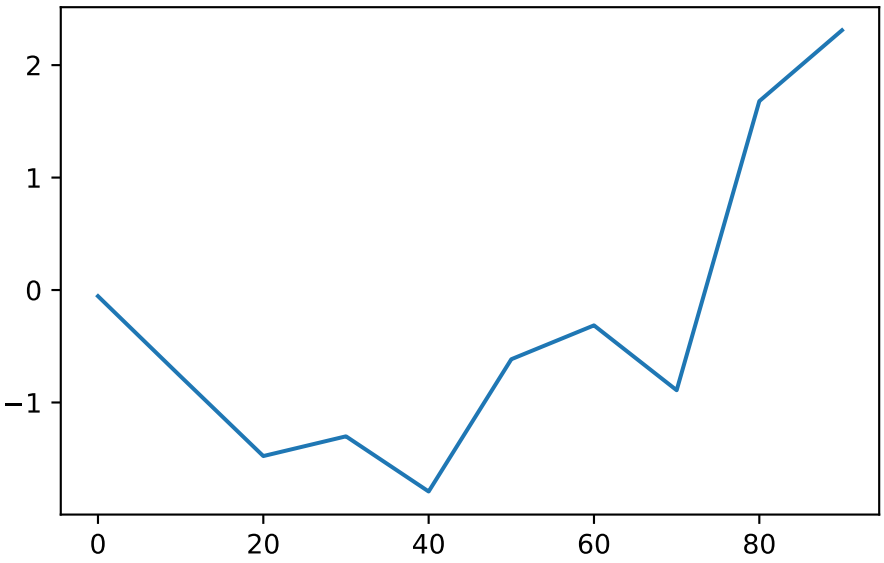
30.2 Downstream packages

There are multiple packages depending on `matplotlib` to provide plotting. For example, you may directly plot from a Pandas DataFrame or a Pandas Series.

Example 30.6.

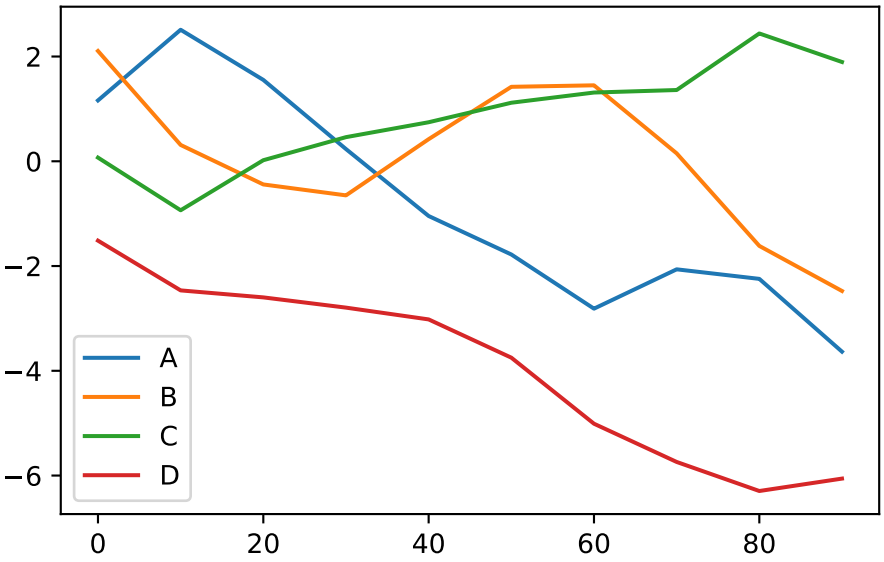
```
import pandas as pd
import numpy as np
s = pd.Series(np.random.randn(10).cumsum(), index=np.arange(0, 100, 10))
s.plot()
```

```
<AxesSubplot:>
```




```
df = pd.DataFrame(np.random.randn(10, 4).cumsum(0),
                  columns=['A', 'B', 'C', 'D'],
                  index=np.arange(0, 100, 10))
df.plot()
```

```
<AxesSubplot:>
```



30.3 plotting

30.3.1 `plt.plot()`

This is the command for line plotting. You may use `linestyle='--'` and `color='g'` to control the line style and color. The style can be shortened as `g--`.

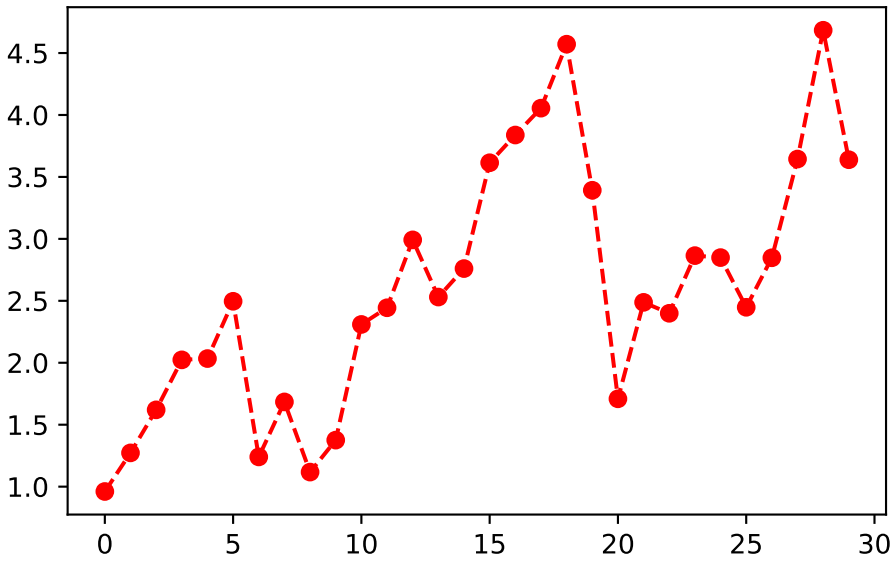
Here is a list of commonly used linestyles and colors.

- line styles
  - `solid` or `-`
  - `dashed` or `--`
  - `dashdot` or `-.`
  - `dotted` or `:`
- marker styles
  - `o` as circle markers
  - `+` as plusses
  - `^` as triangles
  - `s` as squares
- colors
  - `b` as blue
  - `g` as green
  - `r` as red
  - `k` as black
  - `w` as white

The input of `plt.plot()` is two lists `x` and `y`. If there is only one list inputed, that one will be recognized as `y` and the index of elements of `y` will be used as the dafault `x`.

Example 30.7

```
plt.plot(np.random.randn(30).cumsum(), color='r', linestyle='--', marker='o')
```



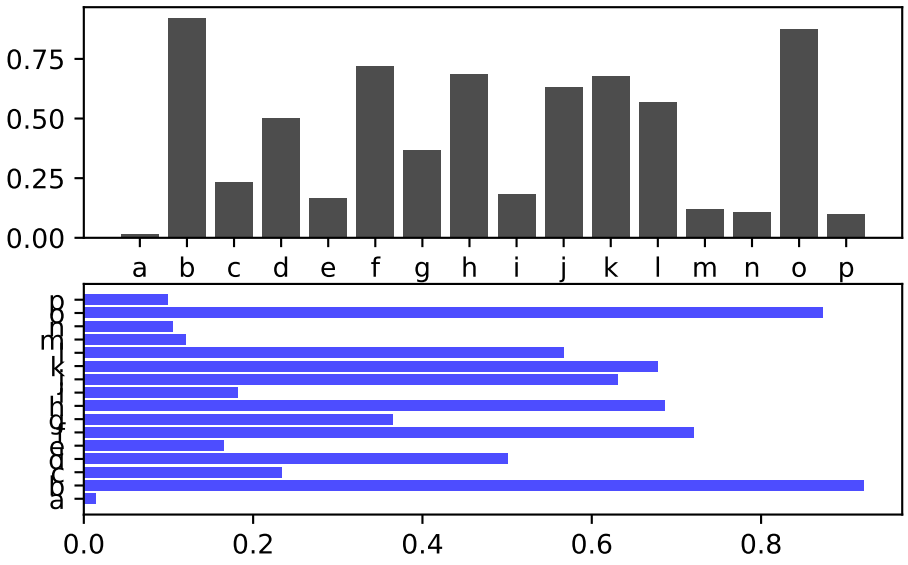
You may compare it with this Example for the purpose of `seaborn` from next Section.

30.3.2 `plt.bar()` and `plt.barh()`

The two commands make vertical and horizontal bar plots, respectively. ::: {#exm-}

```
import pandas as pd
data = pd.Series(np.random.rand(16), index=list('abcdefghijklmnop'))

fig, axes = plt.subplots(2, 1)
axes[0].bar(x=data.index, height=data, color='k', alpha=0.7)
axes[1].barh(y=data.index, width=data, color='b', alpha=0.7)
```

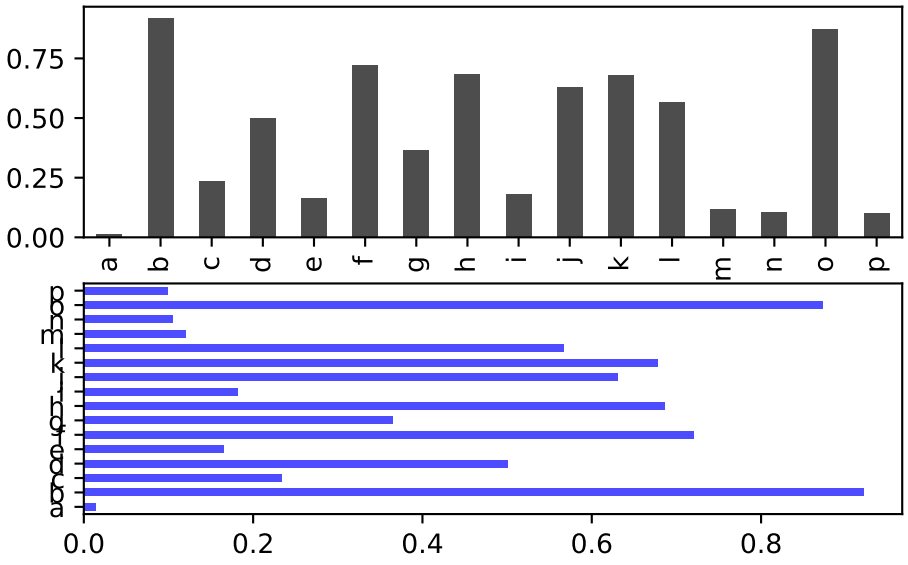
```
<BarContainer object of 16 artists>
```



We may also directly plot the bar plot from the Series.

```
fig, axes = plt.subplots(2, 1)
data.plot.bar(ax=axes[0], color='k', alpha=0.7)
data.plot.barh(ax=axes[1], color='b', alpha=0.7)
```

```
<AxesSubplot:>
```



:::

With a DataFrame, bar plots group the values in each row together in a group in bars. This is easier if we directly plot from the DataFrame.

Example 30.8

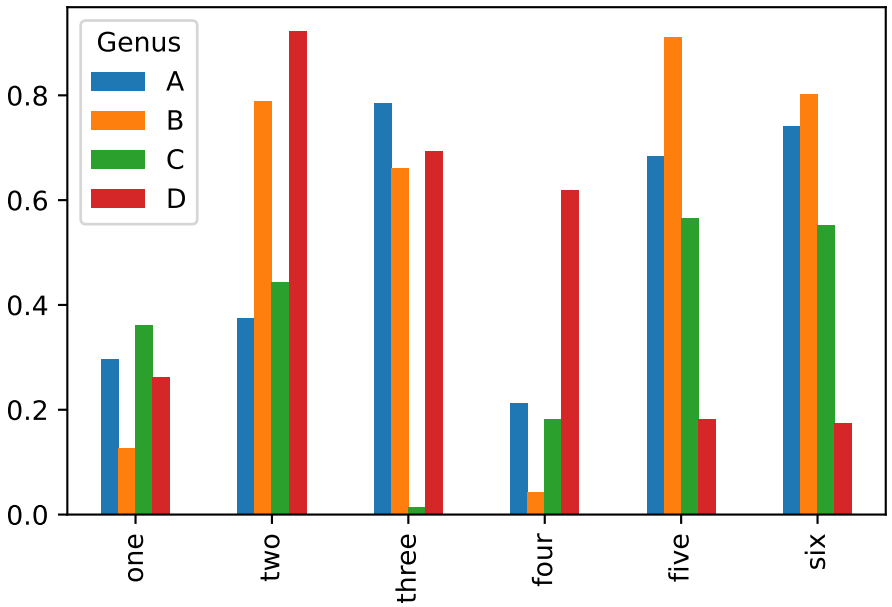
```
df = pd.DataFrame(np.random.rand(6, 4),
                  index=['one', 'two', 'three', 'four', 'five', 'six'],
                  columns=pd.Index(['A', 'B', 'C', 'D'], name='Genus'))
df
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

Genus	A	B	C	D
one	0.297528	0.126427	0.360820	0.261972
two	0.375168	0.789787	0.443735	0.922090
three	0.784327	0.660308	0.014270	0.693136
four	0.212438	0.042673	0.182186	0.619401
five	0.683391	0.910806	0.565555	0.183063
six	0.741157	0.802791	0.552459	0.173806

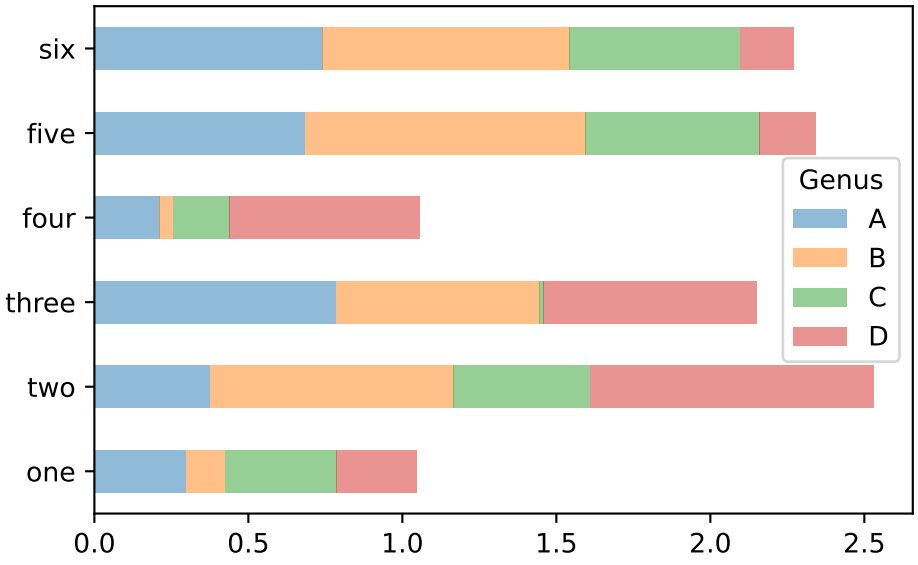
```
df.plot.bar()
```

```
<AxesSubplot:>
```




```
df.plot.barh(stacked=True, alpha=0.5)
```

```
<AxesSubplot:>
```

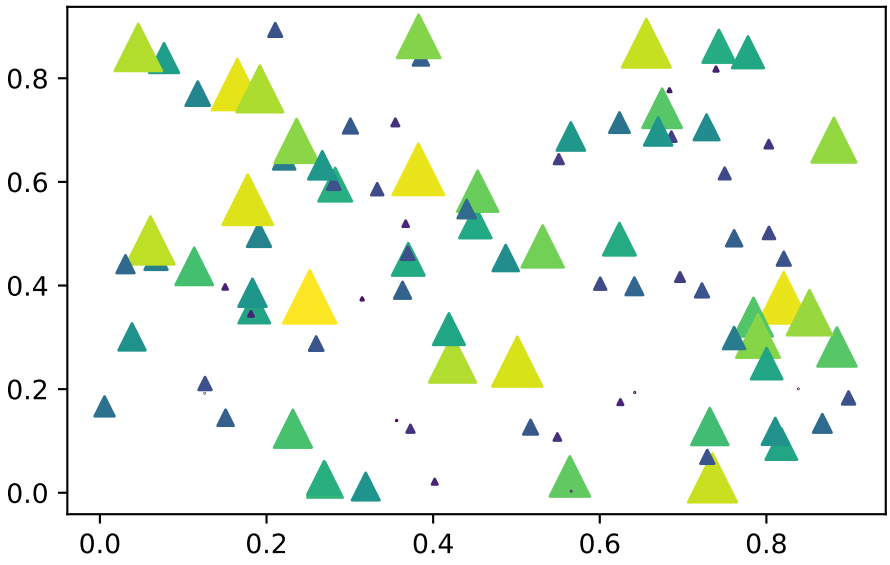


30.3.3 `plt.scatter()`

Example 30.9

```
import numpy as np

N = 100
data = 0.9 * np.random.rand(N, 2)
area = (20 * np.random.rand(N))**2
c = np.sqrt(area)
plt.scatter(data[:, 0], data[:, 1], s=area, marker='^', c=c)
```

```
<matplotlib.collections.PathCollection at 0x1416bc42370>
```

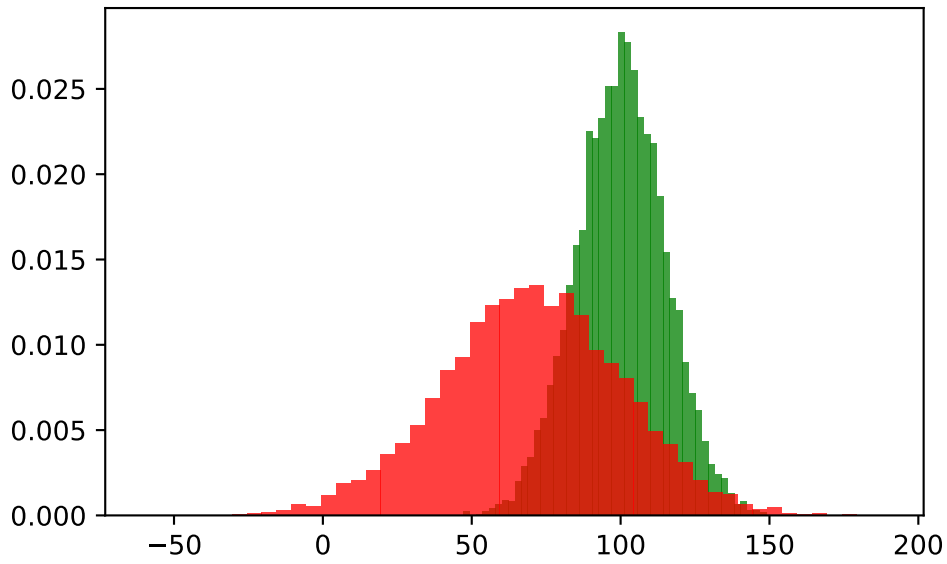


30.3.4 `plt.hist()`

Here are two plots with build-in statistics. The plot command will have statistics as outputs. To disable it we could send the outputs to a temporary variable _. ::: {#exm-histogram1}

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
y = mu-30 + sigma*2 * np.random.randn(10000)
```

```
_ = plt.hist(x, 50, density=True, facecolor='g', alpha=0.75)
_ = plt.hist(y, 50, density=True, facecolor='r', alpha=0.75)
```

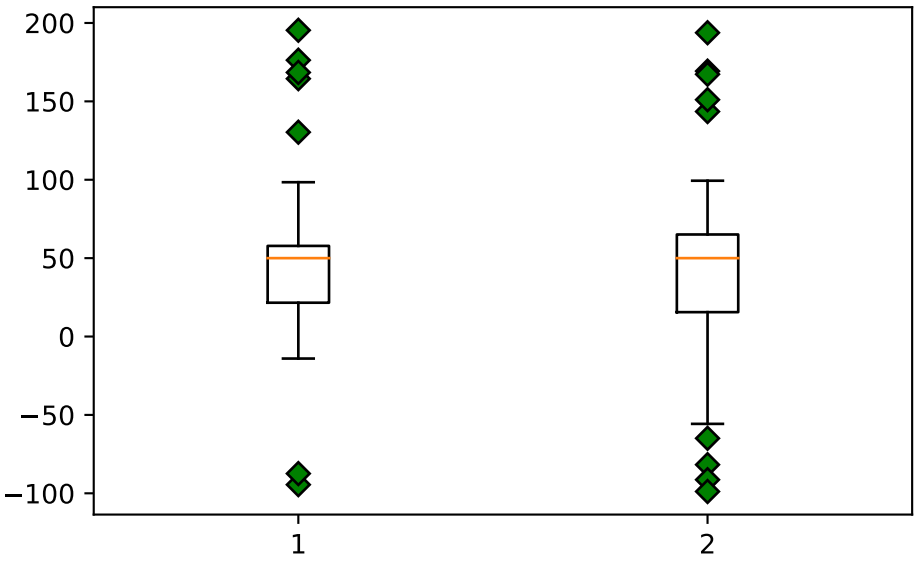


:::

30.4 `plt.boxplot()`

Example 30.10

```
spread = np.random.rand(50) * 100
center = np.ones(30) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low)).reshape(50, 2)

_ = plt.boxplot(data, flierprops={'markerfacecolor': 'g', 'marker': 'D'})
```



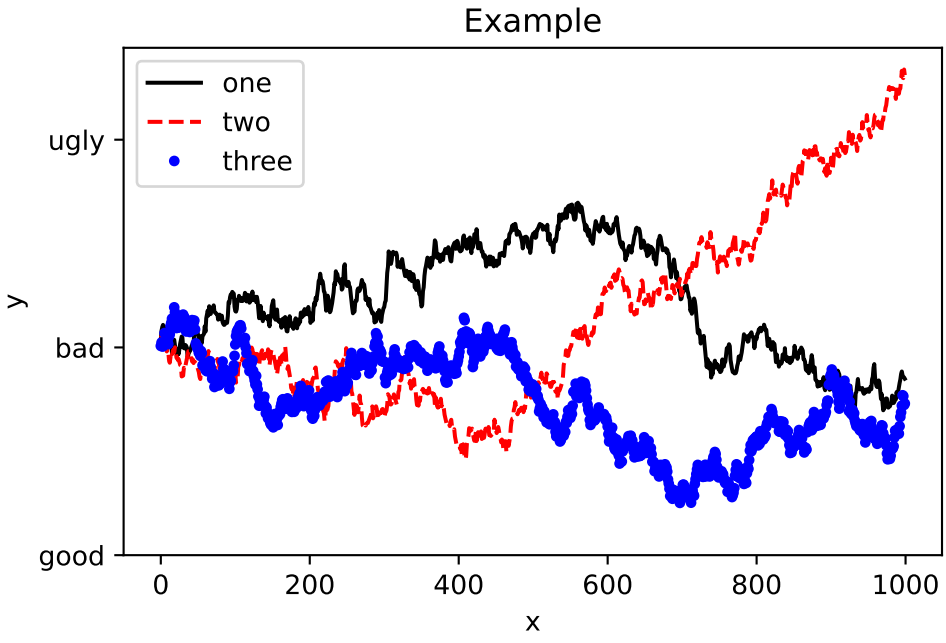
30.5 Titles, labels and legends

- Titles
  - `plt.title(label)`, `plt.xlabel(label)`, `plt.ylabel(label)` will set the title/xlabel/ylabel.
  - `ax.set_title(label)`, `ax.set_xlabel(label)`, `ax.set_ylabel(label)` will do the same thing.
- Labels
  - `plt` methods
    - `xlim()`, `ylim()`, `xticks()`, `yticks()`, `xticklabels()`, `yticklabels()`
    - all the above with arguments
  - `ax` methods
    - `get_xlim()`, `get_ylim()`, etc..
    - `set_xlim()`, `set_ylim()`, etc..
- Legneds
  - First add `label` option to each piece when plotting, and then add `ax.legends()` or `plt.legends()` at the end to display the legends.
  - You may use `handles, labels = ax.get_legend_handles_labels()` to get the handles and labels of the legends, and modify them if necessary.

Example 30.11

```
import numpy as np
fig, ax = plt.subplots(1, 1)
ax.plot(np.random.randn(1000).cumsum(), 'k', label='one')
ax.plot(np.random.randn(1000).cumsum(), 'r--', label='two')
ax.plot(np.random.randn(1000).cumsum(), 'b.', label='three')

ax.set_title('Example')
ax.set_xlabel('x')
ax.set_ylabel('y')

ax.set_yticks([-40, 0, 40])
ax.set_yticklabels(['good', 'bad', 'ugly'])

ax.legend(loc='best')
```

```
<matplotlib.legend.Legend at 0x1416be601f0>
```



30.6 Annotations

- The command to add simple annotations is `ax.text()`. The required auguments are the coordinates of the text and the text itself. You may add several options to modify the

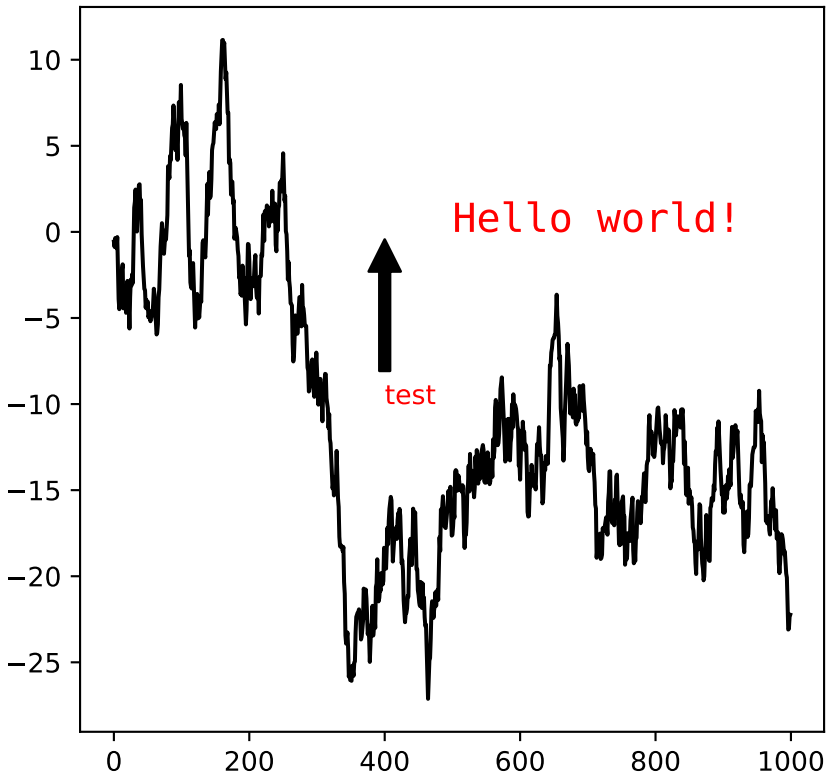
style.

- If arrows are needed, we may use `ax.annotation()`. Here an arrow will be shown from `xytext` to `xy`. The style of the arrow is controlled by the option `arrowprops`.

Example 3012

```
fig, ax = plt.subplots(figsize=(5, 5))
ax.plot(np.random.randn(1000).cumsum(), 'k', label='one')
ax.text(500, 0, 'Hello world!', family='monospace', fontsize=15, c='r')
ax.annotate('test', xy=(400, 0), xytext=(400, -10), c='r',
            arrowprops={'facecolor': 'black',
                        'shrink': 0.05})
```

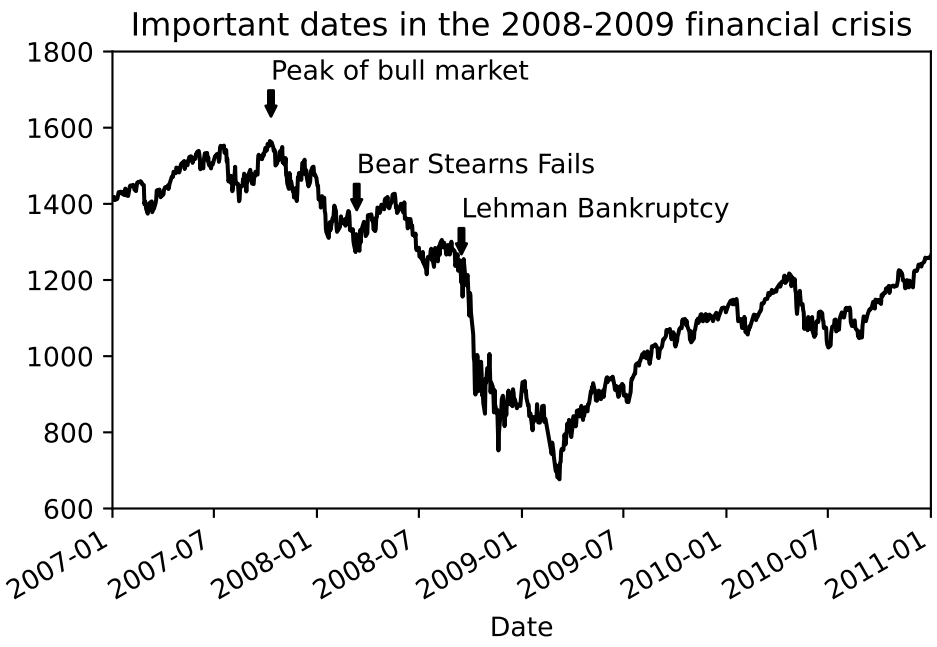
```
Text(400, -10, 'test')
```



30.7 Example

Example 30.13. The stock data can be downloaded from here.

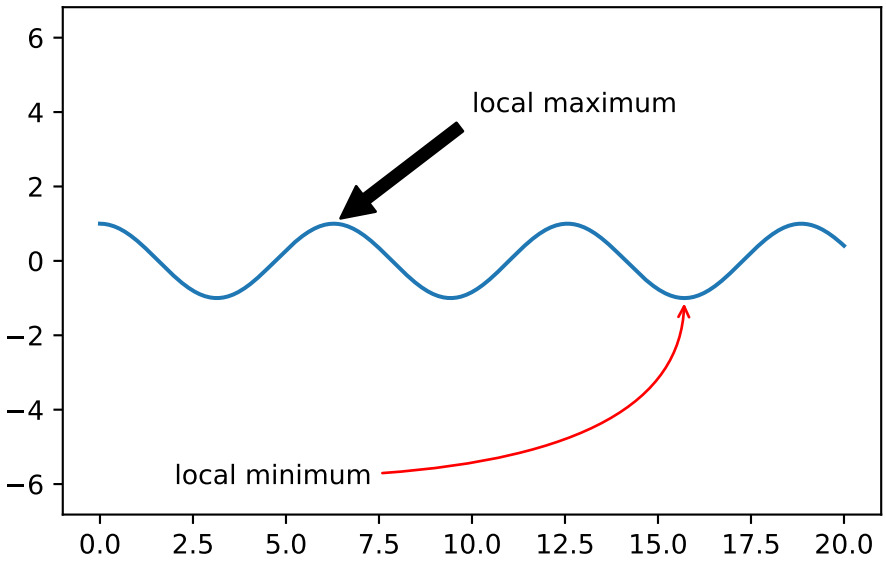
```
from datetime import datetime
fig, ax = plt.subplots()
data = pd.read_csv('assests/datasets/spx.csv', index_col=0, parse_dates=True)
spx = data['SPX']
spx.plot(ax=ax, style='k-')
crisis_data = [(datetime(2007, 10, 11), 'Peak of bull market'),
               (datetime(2008, 3, 12), 'Bear Stearns Fails'),
               (datetime(2008, 9, 15), 'Lehman Bankruptcy')]
for date, label in crisis_data:
    ax.annotate(label, xy=(date, spx.asof(date) + 75),
                xytext=(date, spx.asof(date) + 225),
                arrowprops=dict(facecolor='black', headwidth=4, width=2,
                                headlength=4),
                horizontalalignment='left', verticalalignment='top')
ax.set_xlim(['1/1/2007', '1/1/2011'])
ax.set_ylim([600, 1800])
_ = ax.set_title('Important dates in the 2008-2009 financial crisis')
```



Example 30.14. Here is an example of arrows with different shapes. For more details please read the official document.

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

x = np.linspace(0, 20, 1000)
ax.plot(x, np.cos(x))
ax.axis('equal')

ax.annotate('local maximum', xy=(6.28, 1), xytext=(10, 4),
            arrowprops=dict(facecolor='black', shrink=0.05))

ax.annotate('local minimum', xy=(5 * np.pi, -1), xytext=(2, -6),
            arrowprops=dict(arrowstyle="->",
                            connectionstyle="angle3,angleA=0,angleB=-90",
                            color='r'))
```

```
Text(2, -6, 'local minimum')
```


local maximum
local minimum
6
4
2
0
−2
−4
−6
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0

31 seaborn

There are some new libraries built upon `matplotlib`, and `seaborn` is one of them. `seaborn` is for statistical graphics.

`seaborn` is used imported in the following way.

```
import seaborn as sns
```

`seaborn` also modifies the default `matplotlib` color schemes and plot styles to improve readability and aesthetics. Even if you do not use the `seaborn` API, you may prefer to import `seaborn` as a simple way to improve the visual aesthetics of general `matplotlib` plots.

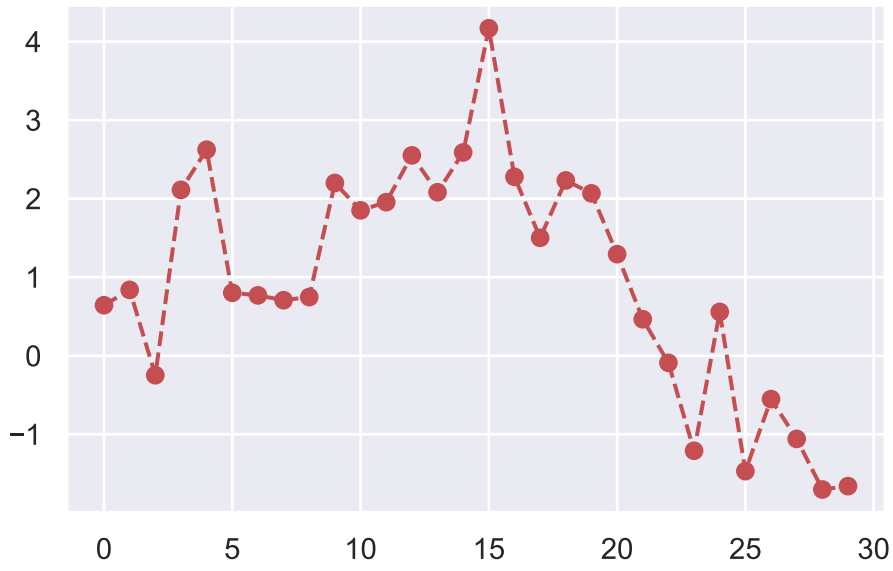
To apply `sns` theme, run the following code.

```
sns.set_theme()
```

Let us directly run a few codes from the last section and compare the differences between them.

Example 31.1

```
plt.plot(np.random.randn(30).cumsum(), color='r', linestyle='--', marker='o')
```



Please compare the output of the same code with the previous example

31.1 Scatter plots with `relplot()`

The basic scatter plot method is `scatterplot()`. It is wrapped in `relplot()` as the default plotting method. So here we will mainly talk about `relplot()`. It is named that way because it is designed to visualize many different statistical relationships.

The idea of `relplot()` is to display points based on the variables `x` and `y` you choose, and assign different properties to alter the apperance of the points.

- `col` will create multiple plots based on the column you choose.
- `hue` is for color encoding, based on the column you choose.
- `size` will change the marker area, based on the column you choose.
- `style` will change the marker symbol, based on the column you choose.

Example 31.2. Consider the following example. `tips` is a DataFrame, which is shown below.

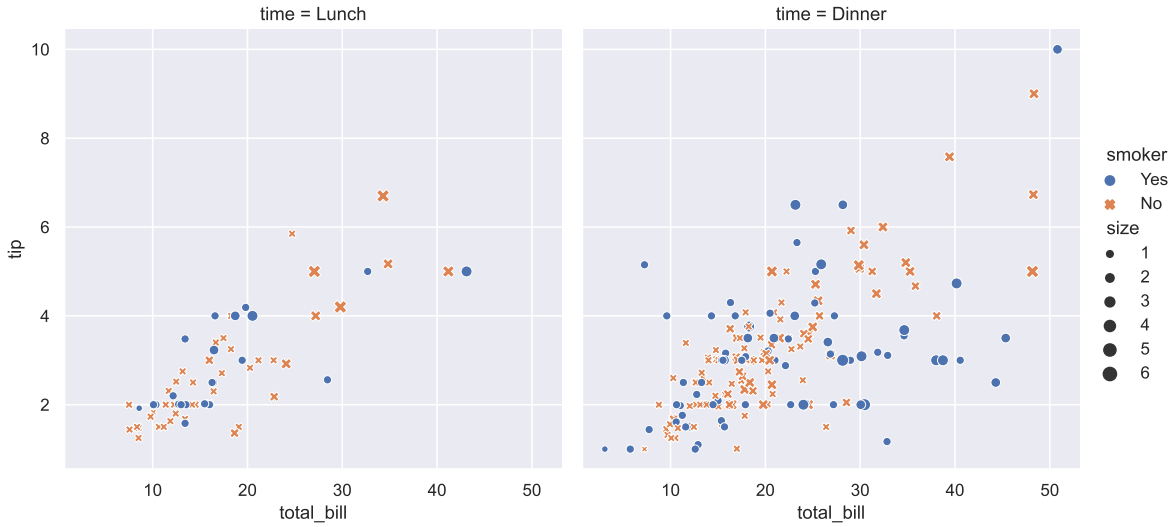
```
import seaborn as sns
tips = sns.load_dataset("tips")
tips
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4
5	25.29	4.71	Male	No	Sun	Dinner	4
6	8.77	2.00	Male	No	Sun	Dinner	2
7	26.88	3.12	Male	No	Sun	Dinner	4
8	15.04	1.96	Male	No	Sun	Dinner	2
9	14.78	3.23	Male	No	Sun	Dinner	2
10	10.27	1.71	Male	No	Sun	Dinner	2
11	35.26	5.00	Female	No	Sun	Dinner	4
12	15.42	1.57	Male	No	Sun	Dinner	2
13	18.43	3.00	Male	No	Sun	Dinner	4
14	14.83	3.02	Female	No	Sun	Dinner	2
15	21.58	3.92	Male	No	Sun	Dinner	2
16	10.33	1.67	Female	No	Sun	Dinner	3
17	16.29	3.71	Male	No	Sun	Dinner	3
18	16.97	3.50	Female	No	Sun	Dinner	3
19	20.65	3.35	Male	No	Sat	Dinner	3
20	17.92	4.08	Male	No	Sat	Dinner	2
21	20.29	2.75	Female	No	Sat	Dinner	2
22	15.77	2.23	Female	No	Sat	Dinner	2
23	39.42	7.58	Male	No	Sat	Dinner	4
24	19.82	3.18	Male	No	Sat	Dinner	2
25	17.81	2.34	Male	No	Sat	Dinner	4
26	13.37	2.00	Male	No	Sat	Dinner	2
27	12.69	2.00	Male	No	Sat	Dinner	2
28	21.70	4.30	Male	No	Sat	Dinner	2
29	19.65	3.00	Female	No	Sat	Dinner	2
30	9.55	1.45	Male	No	Sat	Dinner	2
31	18.35	2.50	Male	No	Sat	Dinner	4
32	15.06	3.00	Female	No	Sat	Dinner	2
33	20.69	2.45	Female	No	Sat	Dinner	4
34	17.78	3.27	Male	No	Sat	Dinner	2
35	24.06	3.60	Male	No	Sat	Dinner	3
36	16.31	2.00	Male	No	Sat	Dinner	3
37	16.93	3.07	Female	No	Sat	Dinner	3
38	18.69	2.31	Male	No	Sat	Dinner	3
39	31.27	5.00	Male	No	Sat	Dinner	3
40	16.04	2.24	Male	No	Sat	Dinner	3
41	17.46	2.54	Male	No	Sun	Dinner	2
42	13.94	3.06	Male	No	Sun	Dinner	2
43	9.68	1.32	Male	No	Sun	Dinner	2
44	30.40	5.60	Male	No	Sun	Dinner	4
45	18.29	3.00	Male	No	Sun	Dinner	2
46	22.23	5.00	Male	No	Sun	Dinner	2
47	32.40	6.00	Male	No	Sun	Dinner	4
48	28.55	2.05	Male	No	Sun	Dinner	3
49	18.04	3.00	Male	No	Sun	Dinner	2
50	12.54	2.50	Male	No	Sun	Dinner	2
51	10.29	2.60	Female	No	Sun	Dinner	2

```
sns.relplot(data=tips,
            x="total_bill", y="tip", col="time",
            hue="smoker", style="smoker", size="size")
```

```
<seaborn.axisgrid.FacetGrid at 0x14167414250>
```

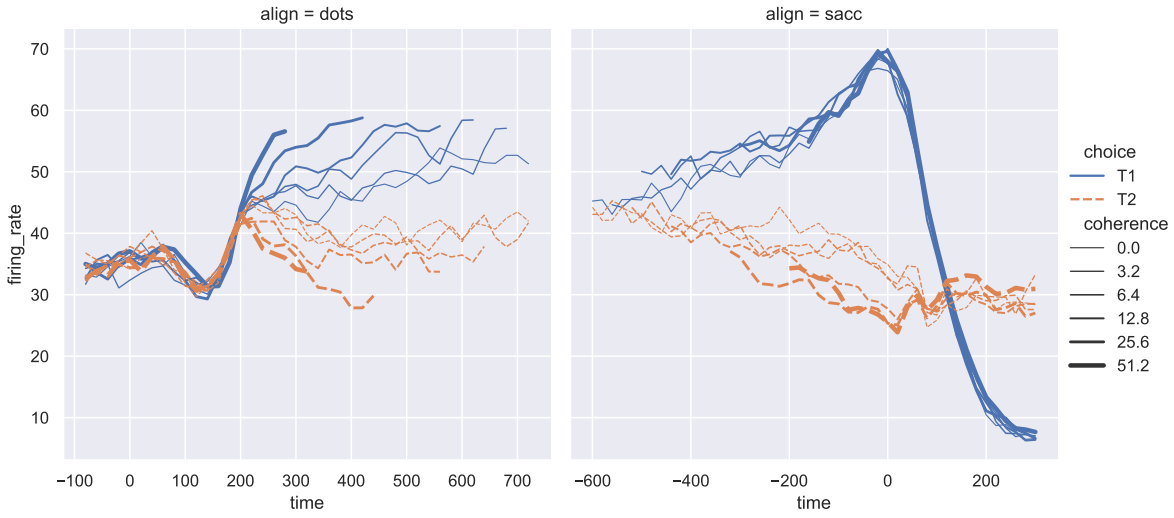


The default type of plots for `relplot()` is scatter plots. However you may change it to line plot by setting `kind='line'`.

Example 113

```
dots = sns.load_dataset("dots")
sns.relplot(data=dots, kind="line",
            x="time", y="firing_rate", col="align",
            hue="choice", size="coherence", style="choice",
            facet_kws=dict(sharex=False))
```

```
<seaborn.axisgrid.FacetGrid at 0x1416e6a5880>
```



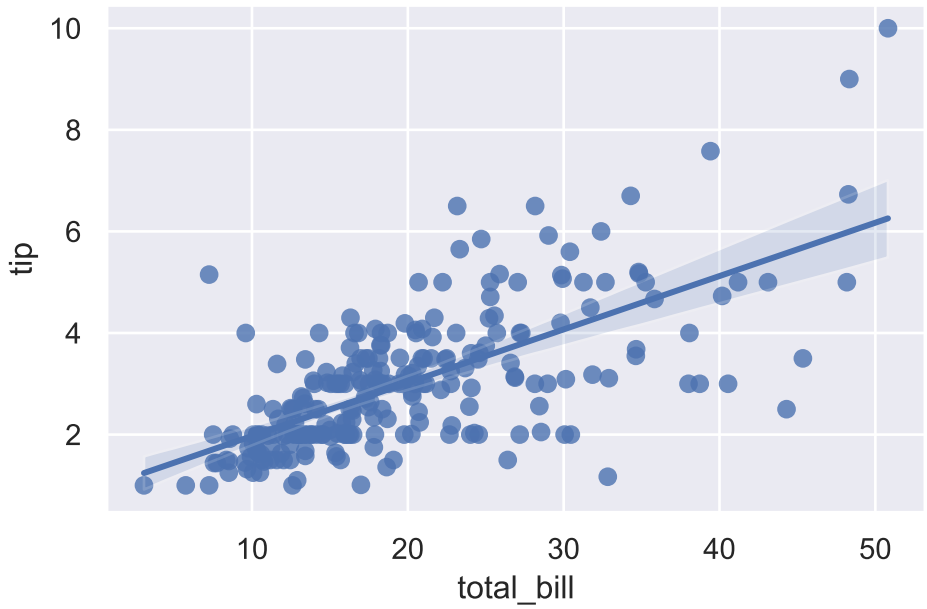
31.2 `regplot()`

This method is a combination between scatter plots and linear regression.

Example 31.4. We still use `tips` as an example.

```
sns.regplot(x='total_bill', y='tip', data=tips)
```

```
<AxesSubplot:xlabel='total_bill', ylabel='tip'>
```



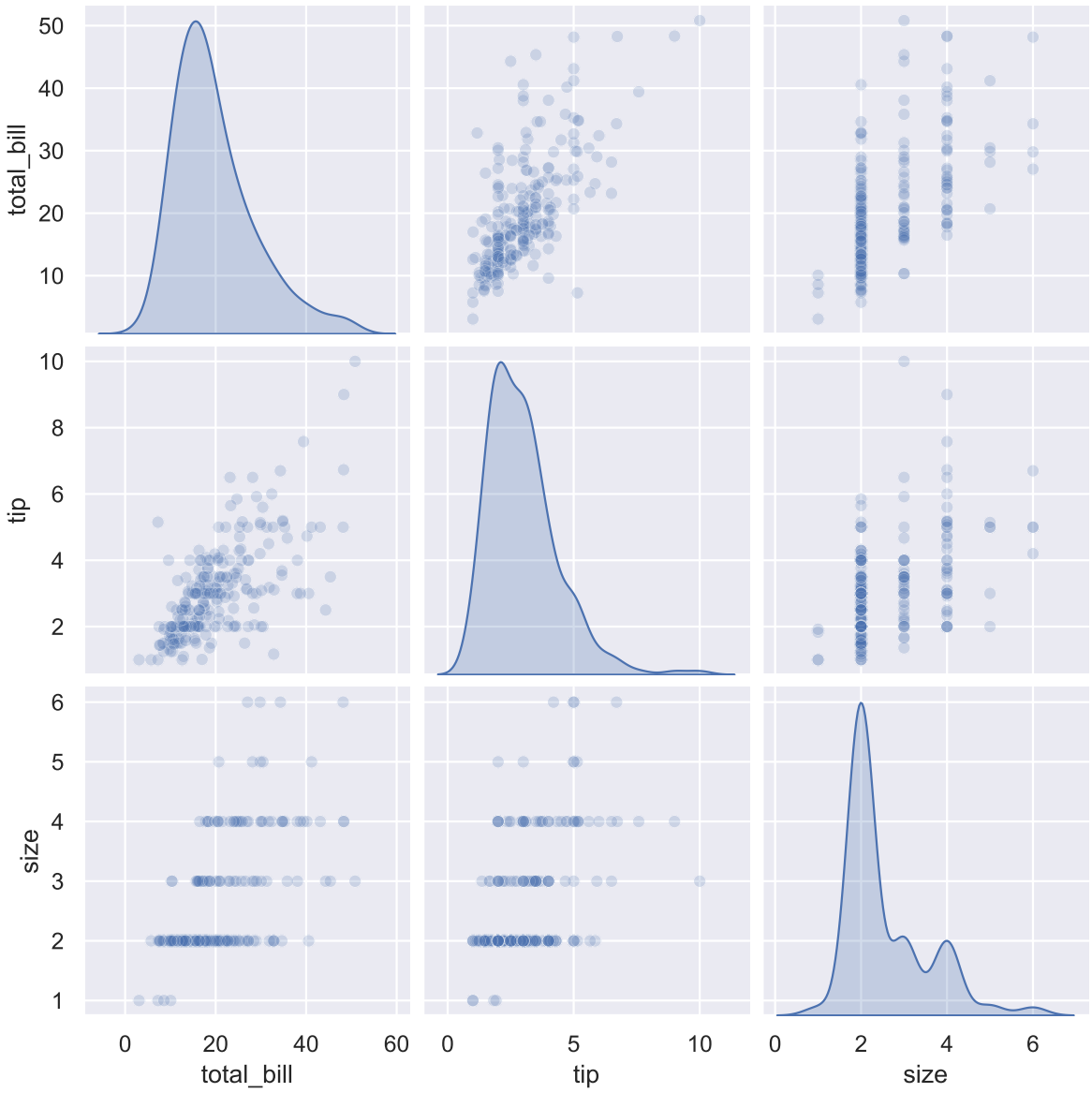
31.3 `pairplot()`

This is a way to display the pairwise relations among several variables.

Example 31.5. The following code shows the pairplots among all numeric data in `tips`.

```
sns.pairplot(tips, diag_kind='kde', plot_kws={'alpha': 0.2})
```

```
<seaborn.axisgrid.PairGrid at 0x1416bcb8e20>
```

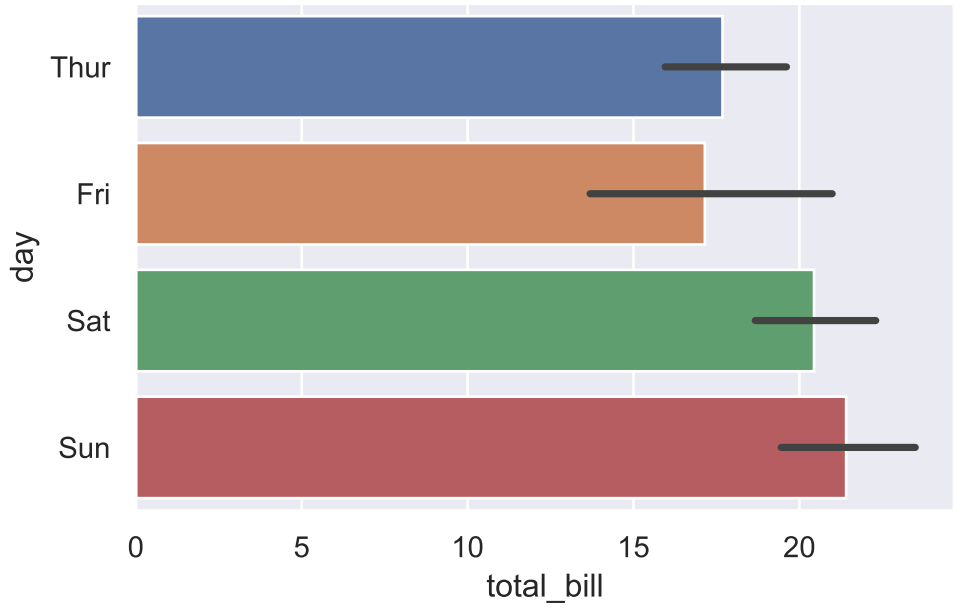


31.4 barplot

Example 31.6.

```
sns.barplot(x='total_bill', y='day', data=tips, orient='h')
```

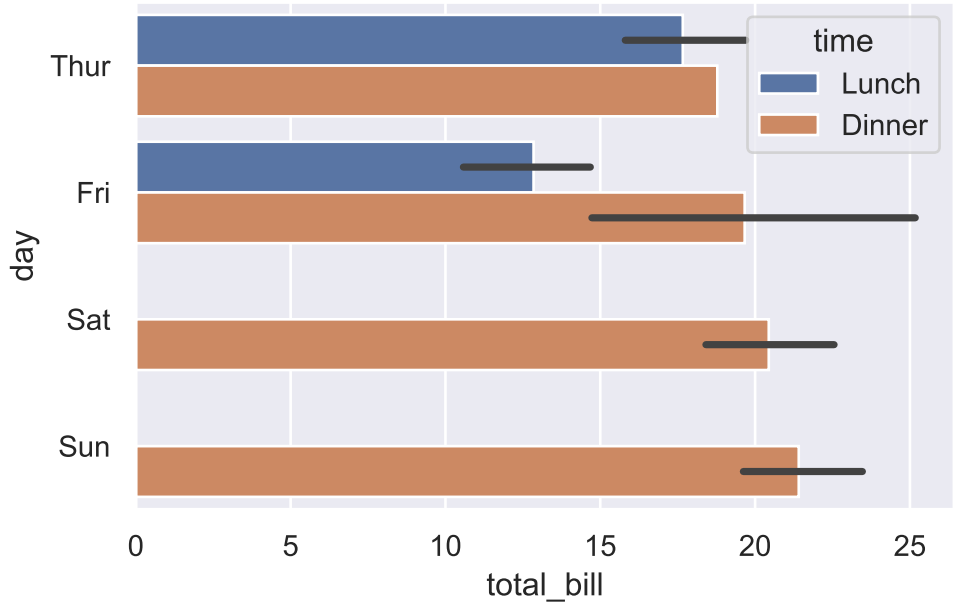
```
<AxesSubplot:xlabel='total_bill', ylabel='day'>
```



In the plot, there are several `total_bill` during each `day`. The value in the plot is the average of `total_bill` in each `day`, and the black line stands for the `95%` confidence interval.

```
sns.barplot(x='total_bill', y='day', hue='time', data=tips, orient='h')
```

```
<AxesSubplot:xlabel='total_bill', ylabel='day'>
```



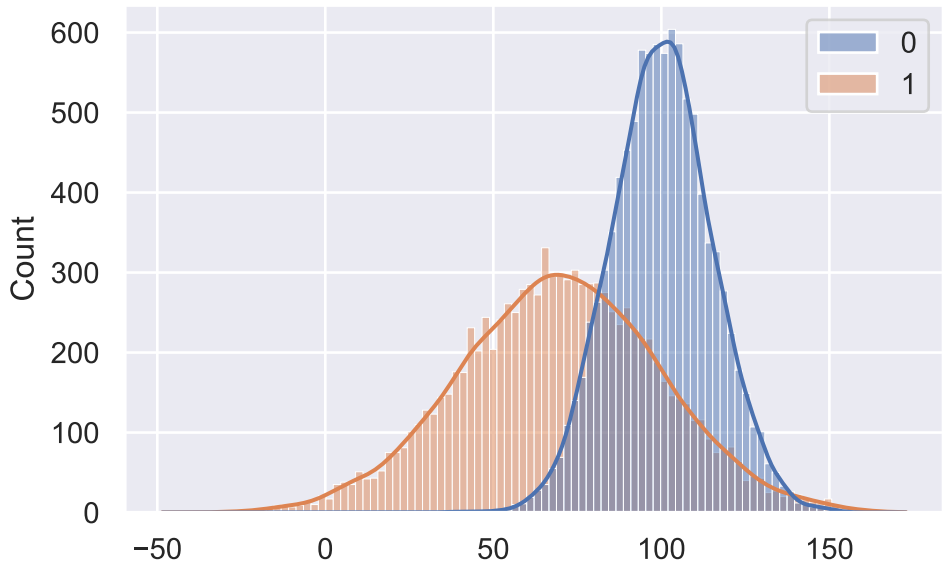
In this plot, lunch and dinner are distinguished by colors.

31.5 Histogram

Example 31.7

```
mu, sigma = 100, 15
x = mu + sigma * np.random.randn(10000)
y = mu-30 + sigma*2 * np.random.randn(10000)
df = pd.DataFrame(np.array([x,y]).T)
sns.histplot(df, bins=100, kde=True)
```

```
<AxesSubplot:ylabel='Count'>
```



Please compare this plot with this Example

32 Examples

32.1 Example 1: `USA.gov` Data From `Bitly`

In 2011, URL shortening service `Bitly` partnered with the US government website `USA.gov` to provide a feed of anonymous data gathered from users who shorten links ending with `.gov` or `.mil`. The data is gotten from [2].

The data file can be downloaded from here. The file is mostly in JSON. It can be converted into a DataFrame by the following code.

```
import pandas as pd
import numpy as np
import json
path = 'assests/datasets/example.txt'
df = pd.DataFrame([json.loads(line) for line in open(path)])
```

We mainly use `tz` and `a` columns. So let us clean it.

```
df['tz'] = df['tz'].fillna('Missing')
df['tz'][df['tz'] == ''] = 'Unknown'
df['a'] = df['a'].fillna('Missing')
df['a'][df['a'] == ''] = 'Unknown'
```

We first want to extract the timezone infomation from it. The timezone info is in the column `tz`.

```
tzone = df['tz']
tvc = tzone.value_counts()
tvc
```

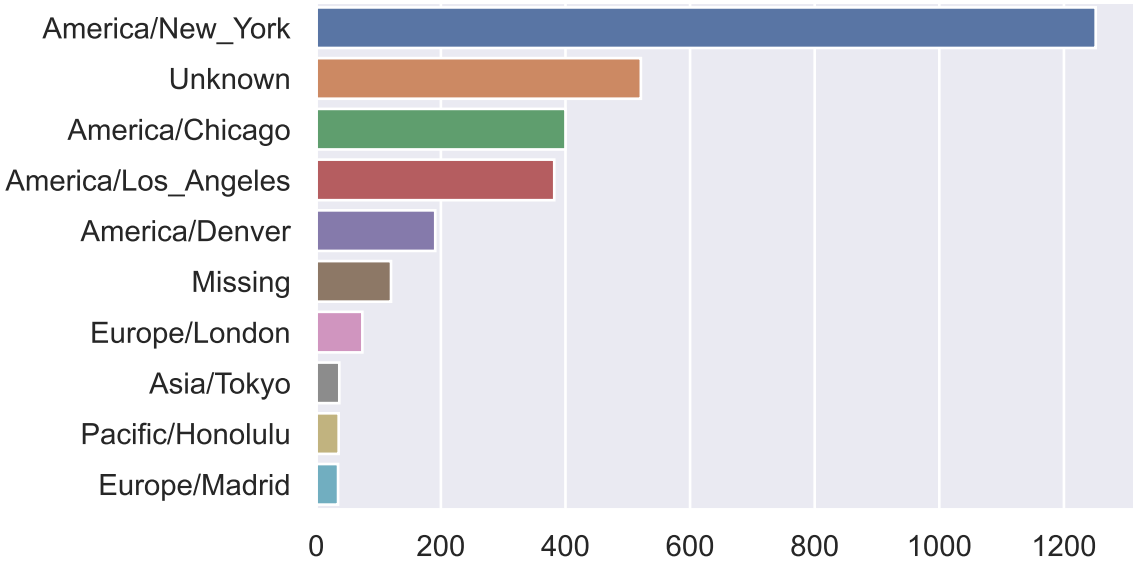
```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	tz
America/New_York	1251
Unknown	521
America/Chicago	400
America/Los_Angeles	382
America/Denver	191
Missing	120
Europe/London	74
Asia/Tokyo	37
Pacific/Honolulu	36
Europe/Madrid	35
America/Sao_Paulo	33
Europe/Berlin	28
Europe/Rome	27
America/Rainy_River	25
Europe/Amsterdam	22
America/Indianapolis	20
America/Phoenix	20
Europe/Warsaw	16
America/Mexico_City	15
Europe/Stockholm	14
Europe/Paris	14
America/Vancouver	12
Pacific/Auckland	11
Europe/Moscow	10
Europe/Helsinki	10
Europe/Oslo	10
Europe/Prague	10
Asia/Hong_Kong	10
America/Puerto_Rico	10
Asia/Calcutta	9
America/Montreal	9
Asia/Istanbul	9
Europe/Lisbon	8
America/Edmonton	6
Chile/Continental	6
Australia/NSW	6
Europe/Vienna	6
Asia/Bangkok	6
Europe/Athens	6
Asia/Seoul	5
Europe/Budapest	5
America/Anchorage	5
Europe/Copenhagen	5
Asia/Dubai	4
Asia/Beirut	4
America/Halifax	4
Europe/Zurich	4
Europe/Bucharest	4
Europe/Brussels	4
America/Winnipeg	4
Asia/Jakarta	3
Asia/Harbin	3

After cleaning data, we would like to visulize the value counts.

```
import seaborn as sns
sns.barplot(x=tvc[:10].values, y=tvc[:10].index)
```

```
<AxesSubplot:>
```



We then would like to extract information from the column `a`. This column is about the agent of the connection. The important info is the part before the space `' '`.

```
agent = df['a']
agent = agent.str.split(' ').str[0]
avc = agent.value_counts()
avc[:10]
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	a
Mozilla/5.0	2594
Mozilla/4.0	601
GoogleMaps/RochesterNY	121
Missing	120
Opera/9.80	34
TEST_INTERNET_AGENT	24
GoogleProducer	21
Mozilla/6.0	5
BlackBerry8520/5.0.0.681	4
BlackBerry8520/5.0.0.592	3

Now let us assume that, if `Windows` appears in column `a` the user is using `Windows` os, if not then not. In this case, the os can be detected by the following code.

```
df['os'] = np.where(df['a'].str.contains('Windows'), 'Windows', 'Not Windows')
```

Now we can make a bar plot about the counts based on `os` and `timezone`.

```
tz_os_counts = df.groupby(['tz', 'os']).size().unstack().fillna(0)
tz_os_counts.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

os tz	Not Windows	Windows
Africa/Cairo	0.0	3.0
Africa/Casablanca	0.0	1.0
Africa/Ceuta	0.0	2.0
Africa/Johannesburg	0.0	1.0
Africa/Lusaka	0.0	1.0

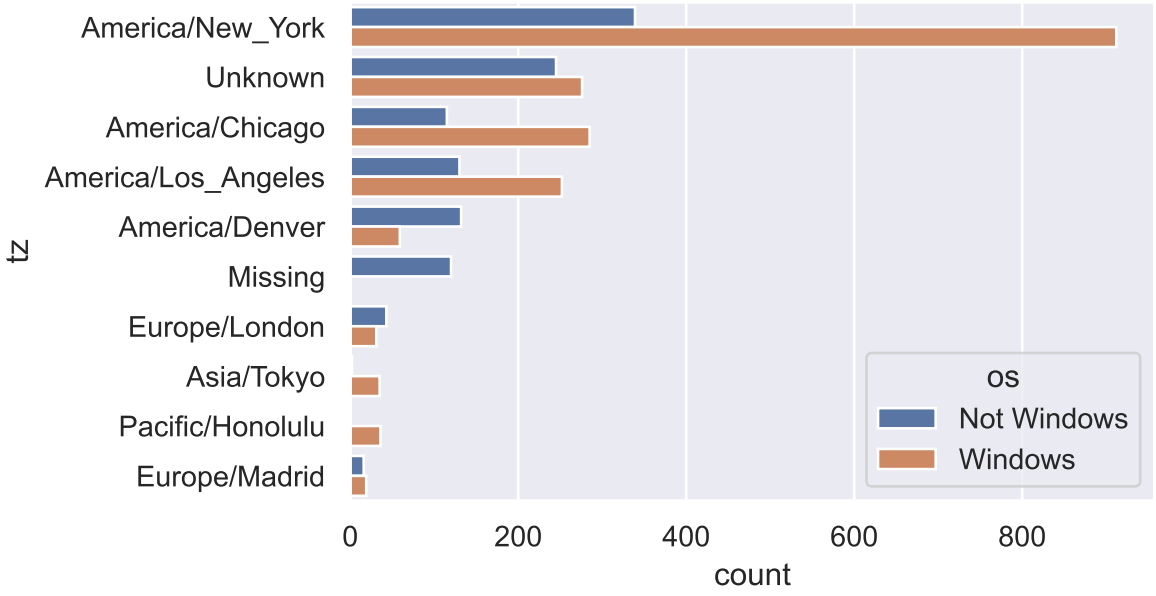
We then turn it into a DataFrame using the `.stack()`, `.unstack()` tricks.

```
tovc = tz_os_counts.stack()[tz_os_counts.sum(axis=1).nlargest(10).index]
tovc.name = 'count'
dftovc = pd.DataFrame(tovc).reset_index()
```

Finally we may draw the bar plot.

```
sns.barplot(x='count', y='tz', hue='os', data=dftovc)
```

```
<AxesSubplot:xlabel='count', ylabel='tz'>
```



32.2 Example 2: US Baby Names 1880–2010

The United States Social Security Administration (SSA) has made available data on the frequency of baby names from 1880 through the present. Hadley Wickham, an author of several popular R packages, has often made use of this dataset in illustrating data manipulation in `R`. The dataset can be downloaded from here as a zip file. Please unzip it and put it in your working folder.

In the folder there are 131 `.txt` files. The naming scheme is `yob` + the year. Each file contains 3 columns: `name`, `gender`, and `counts`. We would like to add a column `year`, and combine all files into a single DataFrame. In our example, the year is from 1880 to 2010.

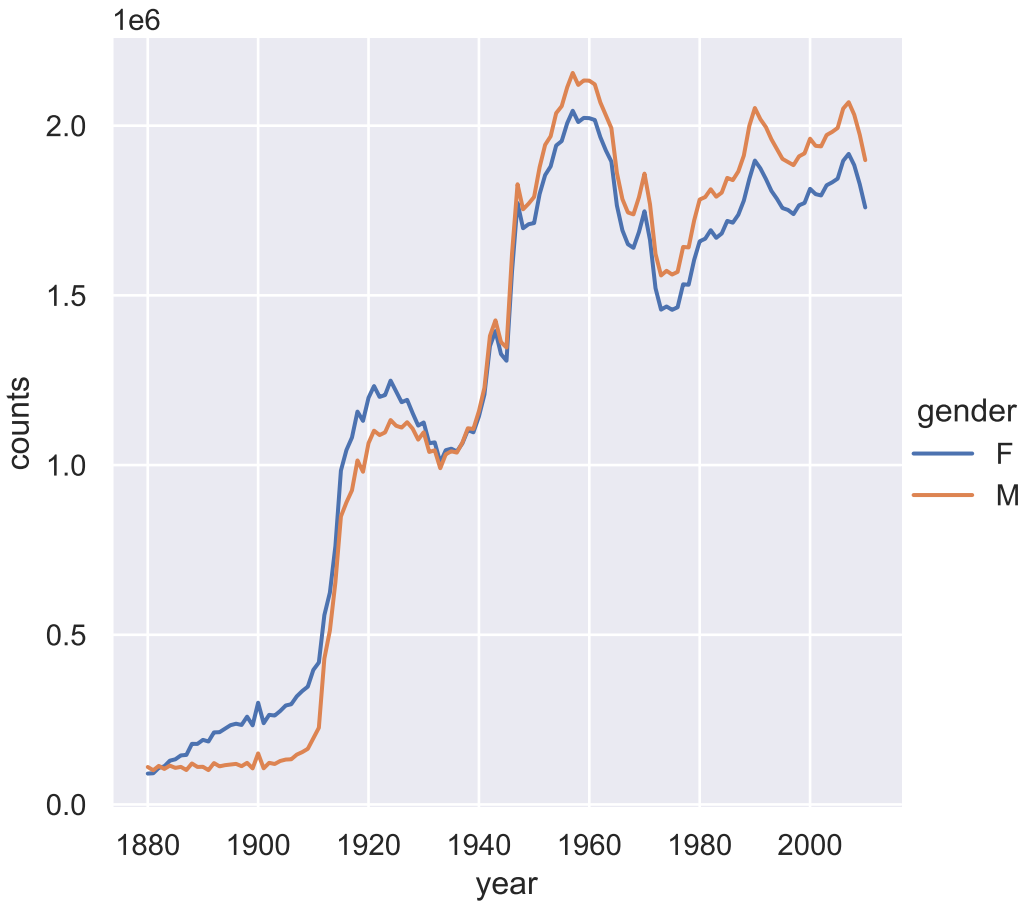
```
import pandas as pd

path = 'assests/datasets/babynames/'
dflist = list()
for year in range(1880, 2011):
```

```
    filename = path + 'yob' + str(year) + '.txt'
    df = pd.read_csv(filename, names=['name', 'gender', 'counts'])
    df['year'] = year
    dflist.append(df)
df = pd.concat(dflist, ignore_index=True)
```

We can plot the total births by sex and year.

```
import seaborn as sns

sns.relplot(data=df.groupby(['gender', 'year']).sum().reset_index(),
            x='year', y='counts', hue='gender', kind='line')
```

```
<seaborn.axisgrid.FacetGrid at 0x14176193580>
```



For further analysis, we would like to compute the proportions of each name relative to the total number of births per year per gender.

```
def add_prop(group):
    group['prop'] = group.counts / group.counts.sum()
    return group

df = df.groupby(['gender', 'year']).apply(add_prop)
df.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

	name	gender	counts	year	prop
0	Mary	F	7065	1880	0.077643
1	Anna	F	2604	1880	0.028618
2	Emma	F	2003	1880	0.022013
3	Elizabeth	F	1939	1880	0.021309
4	Minnie	F	1746	1880	0.019188

Now we would like to keep the first 100 names in each year, and save it as a new DataFrame `top100`.

```
top100 = (
    df.groupby(['year', 'gender'])
    .apply(lambda x: df.loc[x['counts'].nlargest(100).index])
    .drop(columns=['year', 'gender'])
    .reset_index()
    .drop(columns='level_2')
)
top100.head()
```

```
C:\Users\Xinli\anaconda3\envs\ml22\lib\site-packages\IPython\core\formatters.py:343: FutureWa

In future versions `DataFrame.to_latex` is expected to utilise the base implementation of `St
```

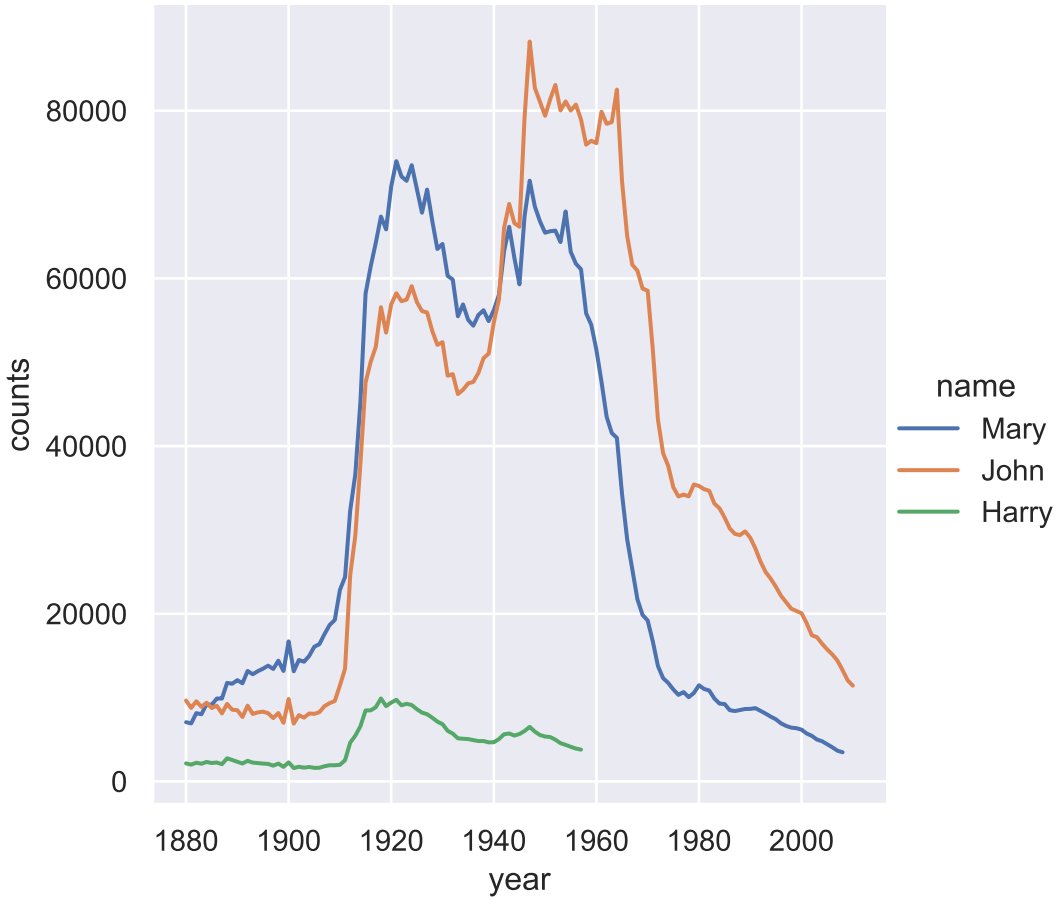
	year	gender	name	counts	prop
0	1880	F	Mary	7065	0.077643
1	1880	F	Anna	2604	0.028618
2	1880	F	Emma	2003	0.022013
3	1880	F	Elizabeth	1939	0.021309
4	1880	F	Minnie	1746	0.019188

Note that `level_2` is related to the original index after `reset_index()`. That's why we don't need it here.

Now we would like to draw the trend of some names.

```
namelist = ['John', 'Harry', 'Mary']
sns.relplot(data=top100[top100['name'].isin(namelist)],
            x='year', y='counts', hue='name', kind='line')
```

```
<seaborn.axisgrid.FacetGrid at 0x14174724e20>
```

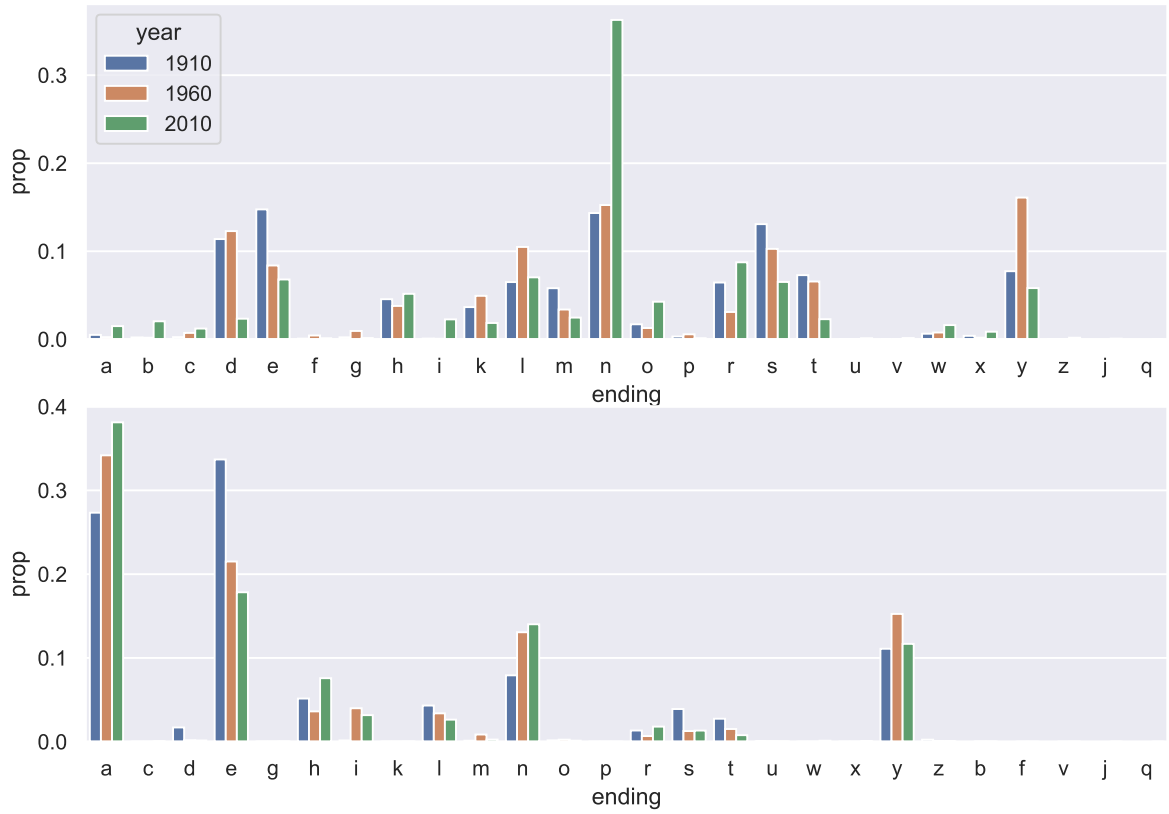


Now we would like to analyze the ending of names.

```
df['ending'] = df['name'].str[-1]
endingcount = df.groupby(['gender', 'year', 'ending']).sum().reset_index()
```

We would like to draw barplots to show the distributions in year 1910, 1960 and 2010.

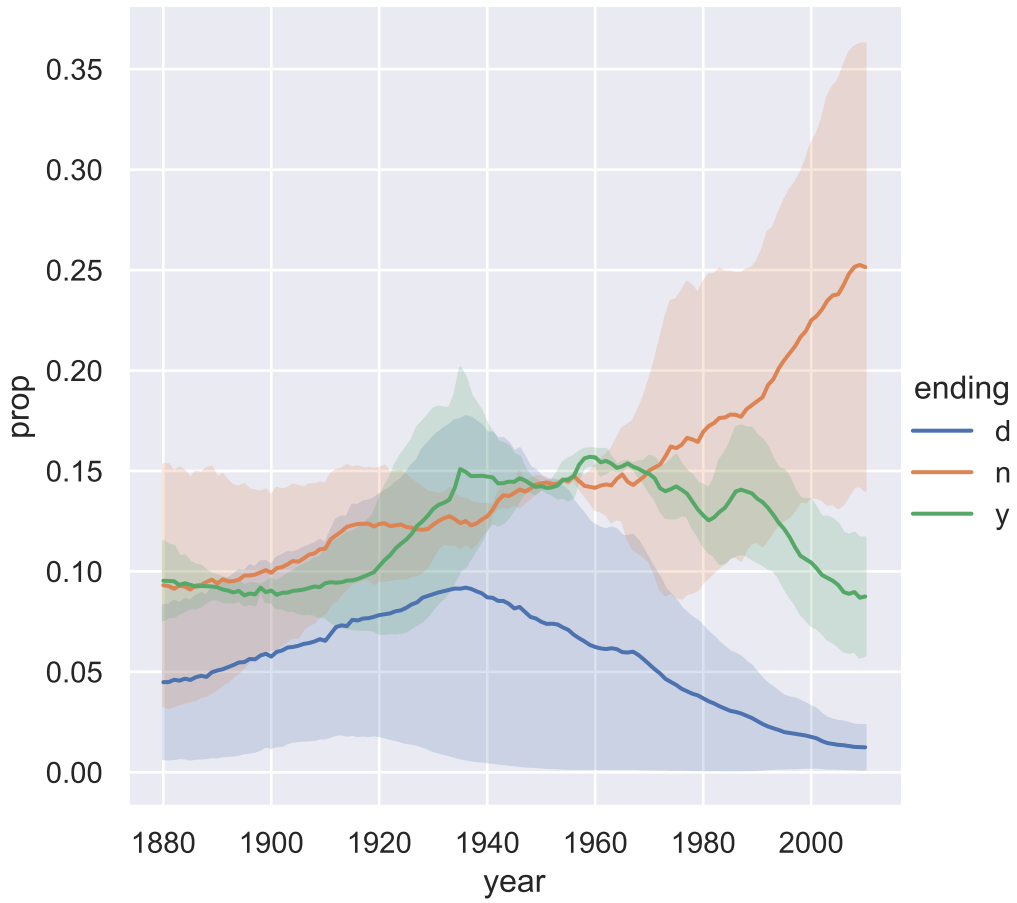
```
certainyear = endingcount[endingcount['year'].isin([1910, 1960, 2010])]
import matplotlib.pyplot as plt

fig, axs = plt.subplots(2, 1, figsize=(10,7))
sns.barplot(data=certainyear[endingcount['gender']=='M'],
            x='ending', y='prop', hue='year', ax=axs[0])
sns.barplot(data=certainyear[endingcount['gender']=='F'],
            x='ending', y='prop', hue='year', ax=axs[1]).legend_.remove()
```



We would also like to draw the line plot to show the trending of certain letters through years.

```
sns.relplot(data=endingcount[endingcount.ending.isin(['d', 'n', 'y'])],
            x='year', y='prop', hue='ending', kind='line')
```

```
<seaborn.axisgrid.FacetGrid at 0x14175ffcc70>
```


0.35
0.30
0.25
0.20
0.15
0.10
0.05
0.00
prop
1880
1900
1920
1940
1960
1980
2000
year
ending
d
n
y

33 Exercises

Exercise 33.1. Please download the `mtcars` file from here and read it as a DataFrame. Then create a scatter plot of the `drat` and `wt` variables from `mtcars` and color the dots by the `carb` variable.

Exercise 33.2. Please consider the baby name dataset. Please draw the trends of counts of names ending in `a`, `e`, `n` across years for each gender.

34 Projects

Exercise 34.1. Please read the file as a DataFrame from here. This is the Dining satisfaction with quick service restaurants questionare data provided by Dr. Siri McDowall, supported by DART SEED grant.

1. Please pick out all rating columns. Excluding `last.visit`, `visit.again` and `recommend`, compute the mean of the rest and add it to the DataFrame as a new column.
2. Use a plot to show the relations among these four columns: `last.visit`, `visit.again`, `recommend` and `mean`.
3. Look at the column `Profession`. Keep `Student`, and change everything else to be `Professional`, and add it as a new column `Status` to the DataFrame.
4. Draw the histogram of `mean` with respect to `Status`.
5. Find the counts of each `recommend` rating for each `Status` and draw the barplot. Do the same to `last.visit`/`Status` and `visit.again`/`Status`.
6. Exploer the dataset and draw one plot.

Exercise 34.2. Please use the baby name dataset. We would like to consider the diversity of the names. Please compute the number of popular names in top 50% for each year each gender. Draw a line plot to show the trend and discuss the result.

Part VII

Classes/Packages for Python

35 Functions

Functions are declared with the `def` keyword and returned from the `return` keyword.

Example 35.1

```
def my_function(x, y, z=1.5):
    if z > 1:
        return z * (x + y)
    else:
        return z / (x + y)
```

Each function can have *positional* arguments and *keyword* arguments.

- Keyword arguments are most commonly used to specify default values.
- If no keywords are given, all arguments will be recognized by the positions.
- If both positional arguments and keyword arguments are given, positional arguments have to be in front.
- The order of keyword arguments are not important.

> **Note**
>
> Although there are global variable, it is always ecouraged to use local variables only. This means that the variables in and out of a function (as well as classes that we will talk about later) are not the same, even if they have the same name.

Example 35.2 (Mutable objects as default value)**.** It is highly recommended NOT to set any mutatable objects as the default value of an input of a function. The reason is that this default object is initialized when the function is defined, not when the function is called. Then all function calls will share the same default object.

A typical example is an empty list. If you use an empty list as the defaul value, that list will be passed to the next function call, which is no longer empty. Please see the following example.

```
def add(x=[]):
    x.append(1)
    return x

add()
```

```
[1]
```

```
add()
```

```
[1, 1]
```

```
add()
```

```
[1, 1, 1]
```

Every time the function is called with no arguments, the default value is used, which is the same list initialized at the beginning. The list at the begining is an empty list. But after we put things inside, it is no longer empyt.

If you want to set a mutable object as a default, the way is as follows:

```
def add(x=None):
    if x is None:
        x = list()
    x.append(1)
    return x

add()
```

```
[1]
```

```
add()
```

```
[1]
```

```
add()
```

```
[1]
```

36 Classes

A class is an abstract structure that can be used to hold both variables and functions. Variables in a class are called *attributes*, and functions in a class are called *methods*.

A class is defined in the following way.

```
class Circle:
    def __init__(self, radius=1):
        self.radius = radius

    def area(self):
        return self.radius**2*3.14
```

In this example, we define a class `Circle`, which represents a circle. There is one attribute `radius`, and one method `area`. When define a cirlce, we need to specify its radius, and we could use the method `area` to compute the area of the circle.

```
cir1 = Circle()
cir2 = Circle(radius=5)

cir1.area()
```

```
3.14
```

```
cir2.area()
```

```
78.5
```

Here we define two circles. The first circle `cir1` is of radius `1`. This `1` comes from the default value. Check the definition of `Circle.__init__()`.

The second circle `cir2` is of radius `5`, and this number is specified when we initialize the `Circle` instance.

Then we compute the areas of these two circles by calling the `area()` method. You can also use `cir1.radius` to get access the radius of the circle. The syntax difference between attributes and methods is the `()` at the end.

36.1 `self`

You may notice the `self` variable in the definition of the classes. The `self` is used to refered to the class its. When you want to get access to the class attributes or the class methods, you may use `self`.

Take the code as an example.

```
class Circle:
    def __init__(self, radius=1):
        self.radius = radius
```

In the `__init__` function, there are two `radius`.

1. `radius` is the local variable that is used by the function. It is also the input argument.
2. `self.radius` is the class attribute, that is shared by all class methods. For example, we may add another class method to the class `Circle`.

```
class Circle:
    def __init__(self, radius=1):
        self.radius = radius

    def area(self):
        return self.radius**2*3.14

    def perimeter(self):
        return self.radius*3.14*2
```

Both `area()` and `perimeter()` use the same `self.radius`.

> **Note**
>
> Class attributes are defined in the `__init__()` function.

36.2 A design example

Assume that we live in a world without Pandas, and we would like to design a table object. Then what do we need?

A table should have multiple rows and multiple columns. We should be able to get access entries by columns and row index. We should also be able to display the table by using the `print` funciton.

> **Note**
>
> The `.__str__()` method will be called when you try to `print` the object. If you don't explicitly override it, the type of the object will be shown.

Therefore we may write the following class.

```
class myTableClass():
    def __init__(self, listoflist=None):
        if listoflist is None:
            listoflist = [[]]
        self.nrows = len(listoflist)
        self.ncols = len(listoflist[0])
        self.data = listoflist
        self.shape = (self.nrows, self.ncols)

    def get(self, i, j):
        return self.data[i][j]

    def __str__(self):
        tmp = [' '.join([str(x) for x in row]) for row in self.data]
        return '\n'.join(tmp)
```

This is a very brief table object. We may add more things to it. For example, we could talk about column names.

```
class myTableClass():
    def __init__(self, listoflist=None, columns=None):
        if listoflist is None:
            listoflist = [[]]
        if columns is None:
            columns = list()
        self.nrows = len(listoflist)
        self.ncols = len(listoflist[0])
        self.data = listoflist
        self.shape = (self.nrows, self.ncols)
        self.columns = columns

    def get(self, i, j):
        return self.data[i][j]

    def rename(self, columns=None):
```

```
        if columns is not None:
            self.columns = columns

    def __str__(self):
        tmp = [' '.join([str(x) for x in row]) for row in self.data]
        if len(self.columns) != 0:
            tmp.insert(0, self.columns)
        return '\n'.join(tmp)
```

> **Note**
>
> In Jupyter notebook or similar environment, we might directly call `df` to show a DataFrame and the shown DataFrame is rendered very pretty. This is due to the `IPython.display.display()` method, and is part of IPython console components.

37 Inheritance

One of the most important feature of classes is inheritance. Attributes and methods can be passed from parents to children, and child classes can override those attributes and methods if needed.

For example, we would like to first write a `people` class.

```
class people():
    def __init__(self, name='default', age=20):
        self.name = name
        self.age = age

    def eat(self):
        print('eat something.')
```

This `people` class defines a people who can eat. Then using this `people` class, we could build a children class: `student`.

```
class student(people):
    pass
```

```
stu1 = student('name1', 10)
stu1.eat()
stu1.name
```

```
eat something.
```

```
'name1'
```

```
type(stu1)
```

```
__main__.student
```

Now you can see that this `stu1` is a `student`, but it has all attributes and methods as a `people`. However at current stage `student` and `people` are exactly the same since we don't have any new codes for `student`. Let us improve it a little bit.

```
class student(people):
    def __init__(self, name='default', age=20, grade=1):
        super().__init__(name, age)
        self.grade = grade

    def eat(self):
        print('eat in the cafe.')

stu1 = student('name1', 10)
stu1.eat()
```

```
eat in the cafe.
```

Now `student` class override the `eat()` method from `people`. If someone is a `student`, he or she will eat in the cafe instead of just eat something.

In addition, you may also notice that the `__init__()` constructor function is also overriden. The first part is `super().__init__(name, age)` which is just call the `people`'s constructor function. The second part is new in `student`, that we add a new attribute `grade` to it. Now `stu1` have attributes from `people` and the new attribute defined in `student`.

```
stu1.name, stu1.age
```

```
('name1', 10)
```

```
stu1.grade
```

```
1
```

38 packages / modules

Main reference is RealPython and [8].

38.1 `import`

In most cases we won't only write one single Python file. If we want to use codes from other files, we need to `import`.

- If both files are in the same folder, e.g. `file1.py` and `file2.py`, you may just put `import file2` in `file1.py`, and use `file2.myfunction()` to call functions or variables defined in `file2.py`.
- If both files are in the same folder, and you just want to use one function from `file1.py` in `file2.py`, you may `from file1 import myfunction()`, and then directly write `myfunction()` in `file2.py`.

Example 38.1. This is from `file1.py`.

```
s = "This is from file1.py."
a = [100, 200, 300]
print(s)

def foo(arg):
    print(f'arg = {arg}')

class Foo:
    pass
```

```
This is from file1.py.
```

In `file2.py`, we could get access to these variables and functions and classes as follows.

```
import file1
file1.s
```

```
'This is from file1.py.'
```

```
file1.a
```

```
[100, 200, 300]
```

```
file1.foo(file1.a)
```

```
arg = [100, 200, 300]
```

```
file1.Foo()
```

```
<file1.Foo at 0x1e2203fb6d0>
```

> **Note**
>
> An alternative way is to use `from <module> import <names>` to directly use the names without the `file1.` prefix.

Please see the following Example to get a feel about how namespace works.

Example 38.2.

```
# file1
a = ['foo', 'bar', 'baz']

from file1 import s as string, a as alist
s
```

```
'foo'
```

```
string
```

```
'This is from file1.py.'
```

```
a
```

```
['foo', 'bar', 'baz']
```

```
alist
```

```
[100, 200, 300]
```

We may use `dir()` to look at all objects in the current namespace.

38.2 __name__

`__name__` is a variable to tell you want is the current active namespace. See the following example.

Example 383.

```
file1.__name__
```

```
'file1'
```

The result `file1` means that the codes in `file1.py` are now treated as a package and are imported into other files.

```
__name__
```

```
'__main__'
```

The result `__main__` means that the codes we are writing now are treated as in the "active" enviroment.

You may see the following codes in a lot of places.

```
if __name__ == '__main__':
    pass
```

It means that the following codes will only be run in the "active" mode. If you import the codes as a package, these part of codes won't be run.

38.3 Packages

Pacages is a way to group and organize mutliple modules. It allow for a hierachical structuring of the module namespace using dot notation.

Creating a package is straightforward, since it makes use of the operating system's inherent hierarchical file structure.

Python defines two types of packages, regular packages and namespace packages. The above package is the regular one. Namespace packages allow codes are spread among different folders. We won't talk about it in this course.

To create a regular package, what you need to do is to organize the files in suitable folders, and then add an `__init__.py` in each folder. The file can be empty, or you could add any initialization codes for the package which is represented by the folder.

> **Note**
>
> In the past `__init__.py` is required for a package. After Python 3.3 the namespace package is introduced, the `__init__.py` is not required (but recommended) for regular packages, and cannot be used for namespace packages.

Let us put the previous `file1.py` and `file2.py` into subfolder `assests/codes/`. To make it into a package `assests` and a subpackage `codes`, we need to put `__init__.py` in each folder.

```
import assests.codes.file1 as f1
f1.s
```

```
'This is from file1.py.'
```

39 Exercieses

Exercise 39.1 (Heron's formula)**.** Consider a triangle whose sides are a, b and c. Heron's formula states that the area of this triangle is

$$\sqrt{s(s-a)(s-b)(s-c)} \quad \text{where } s = \frac{1}{2}(a+b+c).$$

Please write a function that given three points computes the area of the triangle with vertices being the given points. The input is required to be a `list` of three `tuple`s, where each `tuple` contains two numbers representing the 2d-coordinate of a point.

Exercise 39.2 (array)**.** Write a function to reverse an 1D NumPy array (first element becomes last).

Exercise 39.3 (Compare two `numpy` arraies)**.** Consider two `numpy` arraies `x` and `y`. Compare them entry by entry. We would like to know how many are the same.

Write a function that the inputs are `x` and `y`, and the output is the number of the same numbers.

40 Projects

Problems are based on [4].

Exercise 40.1 (Comma Code)**.** Say you have a list value like this: `spam = ['apples', 'bananas', 'tofu', 'cats']`.

Write a function that takes a list value as an argument and returns a string with all the items separated by a comma and a space, with `and` inserted before the last item. For example, passing the previous `spam` list to the function would return 'apples, bananas, tofu, and cats'. But your function should be able to work with any list value passed to it. Be sure to test the case where an empty list `[]` is passed to your function.

Exercise 40.2 (Fantasy Game Inventory)**.** You are creating a fantasy video game. The data structure to model the player's inventory will be a dictionary where the keys are string values describing the item in the inventory and the value is an integer value detailing how many of that item the player has. For example, the dictionary value `{'rope': 1, 'torch': 6, 'gold coin': 42, 'dagger': 1, 'arrow': 12}` means the player has 1 rope, 6 torches, 42 gold coins, and so on.

Write a function named `displayInventory()` that would take any possible `inventory` and display it like the following:

```
Inventory:
12 arrow
42 gold coin
1 rope
6 torch
1 dagger
Total number of items: 62
```

Note that this is the function version of Exercise 10.5.

Exercise 40.3. Create a Car class with two instance attributes:

1. `.color`, which stores the name of the car's color as a string.
2. `.mileage`, which stores the number of miles on the car as an integer.

Then instantiate two Car objects — a blue car with 20,000 miles and a red car with 30,000 miles — and print out their colors and mileage. Your expected output are below:

```
car1 = mycar(color='blue', mileage=20000)
car2 = mycar(color='red', mileage=30000)

print(car1)
print(car2)
```

```
A blue car with 20000 mileage.
A red car with 30000 mileage.
```

Exercise 40.4. Create a `GoldenRetriever` class that inherits from the `Dog` class. Give the `sound` argument of `GoldenRetriever.speak()` a default value of `Bark`. Use the following code for your parent `Dog` class:

```
class Dog:
    species = "Canis familiaris"

    def __init__(self, name, age):
        self.name = name
        self.age = age

    def __str__(self):
        return f"{self.name} is {self.age} years old"

    def speak(self, sound):
        return f"{self.name} says {sound}"
```

Part VIII

Part II: R

Part IX

R Fundamentals

[9]

A few advantages about R:

- Free and open source comparing to some other tools like Excel and SPSS.
- Optimized with vectorization.

41 Hello world for R

```
print('Hello world!')
```

```
[1] "Hello world!"
```

42 Essential concepts

- In R, assignments is `<-`, not `=`. `=` actually works, but it may cause confusions. So it is always recommended to use `<-`. The R Studio keybinding for `<-` is `alt+-`.
- `.` is NOT a special character in R, and can be used in variable names. So `is.na()` simply means a function called `is.na`. It is not a function `na` in a package `is` as in Python.
- In R, the block is defined by `{}`. Indentation is not that important.
- R has a better package management system than Python, and therefore in most cases you don't need virtual environment for R.

42.1 R Markdown / Quarto

The counterpart of Jupyter notebook in R is `.rmd/.qmd` file. Similar to a notebook, in a R Markdown / Quarto file, there is a so-called code block that can run the codes inside to produce documents with both texts and codes and codes outputs.

In the following two sections about R, you are supposed to submit `.rmd/.qmd` file.

> **Note**
>
> Quarto is an extension/continuation of R Markdown. Most R Markdown file can be directly translated to a Quarto file without many modifications. The main difference between R Markdown and Quarto is that Quarto has better support for other languages such as Python and Julia. You may go to its homepage for more details.
> This note is produced by Quarto.

The most import part of R Markdown / Quarto is the code block, that is

````
```{r}
print('Hello world!')
```
````

In Quarto, you may also write

````
```{python}
print('Hello world!')
```
````

There are many options to adjust how the code blocks are excacuted. You don't need to worry about them right now. Currently just try to write your report together with code blocks.

42.2 Development tools

42.2.1 R Studio

For R, the almost definite choice of IDE is R Studio. You may download and install it from the homepage.

Note that R Studio will soon be renamed to `posit`. Please keep an eye on it if it will make any differences.

42.2.2 Google Colab

You may use R in Google Colab. The link is colab.to/r. After you open the notebook, you may go to `Edit->Notebook settings` to change Runtime type to be `R`.

The rest is similar to Jupyter notebook, while the codes are now R codes.

42.2.3 Kaggle notebook

Actually there are a lot of online notebook services that you can use. Kaggle is another good choice. It is a little bit better than Google Colab since its R choice is explicitly listed in the settings: `File->Language`.

43 R Basics

Main reference here is [10] and [9].

43.1 Data structures

43.1.1 Vectors

Vector is one of the basic data structure in R. It is created by `c()` function. You may store any data types in it. The data type inside a vector can be checked by `typeof` function.

```
die <- c(1, 2, 3, 4, 5, 6)
typeof(die)
```

```
[1] "double"
```

For consecutive numbers, an easier way to create vector is to use `:`.

```
die <- 1:6
```

> **Note**
>
> Note that vector index starts from 1 in R, while list index starts from 0 in Python.

```
die[1]
```

```
[1] 1
```

When slicing with vectors, don't forget to use `c()`.

```
die[c(2, 3)]
```

```
[1] 2 3
```

```
die[2:3]
```

```
[1] 2 3
```

You may use `length()` function to get its length.

```
length(die)
```

```
[1] 6
```

43.1.2 Attributes

R objects may have attributes. Attributes won't be shown by default when you show the object. You may find the attributes of a R object by calling the `attributes()` function.

The following example show that the vector `die` defined in Section 43.1.1 doesn't have attributes.

```
attributes(die)
```

```
NULL
```

The most common attributes of R objects are `names`. We could manually input the attributes.

```
names(die) <- c('one', 'two', 'three', 'four', 'five', 'six')
die
```

```
  one   two three  four  five   six
    1     2     3     4     5     6
```

After the assignment, each value is assigned a name. Now `die` has an attribute `names`. It is still a vector despite the fact that it now has names.

```
attributes(die)
```

```
$names
[1] "one"   "two"   "three" "four"  "five"  "six"
```

```r
names(die)
```

```
[1] "one"   "two"   "three" "four"  "five"  "six"
```

```r
is.vector(die)
```

```
[1] TRUE
```

43.1.3 matrices

```r
m <- matrix(c(1,2,3,4,5,6), nrow=2)
m[1, ]
```

```
[1] 1 3 5
```

A matrix has `dim` attribute.

```r
dim(m)
```

```
[1] 2 3
```

Note that by assigning and removing `dim` attribute, you may change the object between vectors and matrices.

Example 43.1.

```
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6
```

```r
is.matrix(m)
```

```
[1] TRUE
```

```r
is.vector(m)
```

```
[1] FALSE
```

```r
dim(m)
```

```
[1] 2 3
```

```r
dim(m) <- NULL
m
```

```
[1] 1 2 3 4 5 6
```

```r
is.matrix(m)
```

```
[1] FALSE
```

```r
is.vector(m)
```

```
[1] TRUE
```

> **Note**
>
> The `dim` of a matrix/vector can be a length 3 vector. In this case, it will become an array. For simplicity we won't talk about array here.

43.1.4 Lists

List is very similar to a vector. The main difference is that vector can only store values, while list can store objects. The most typical example of objects is another vector. Please see the following example.

Example 43.2.

```
c(1:2, 3:4)
```

```
[1] 1 2 3 4
```

```
list(1:2, 3:4)
```

```
[[1]]
[1] 1 2

[[2]]
[1] 3 4
```

43.1.5 `data.frame`

Data Frame is a 2d version of a list. You may think about it in terms of tables.

```
df <- data.frame(face = c("ace", "two", "six"),
                 suit = c("clubs", "clubs", "clubs"),
                 value = c(1, 2, 3))
df
```

```
  face  suit value
1  ace clubs     1
2  two clubs     2
3  six clubs     3
```

43.1.6 Examples

Example 43.3. Consider a date.frame representing a deck of cards. Here we use `expand.grid()` to perform the Cartesian product.

```
suit <- c('spades', 'hearts', 'clubs', 'diamonds')
face <- 1:13
deck <- expand.grid(suit, face)
head(deck)
```

```
      Var1 Var2
1   spades    1
2   hearts    1
3    clubs    1
4 diamonds    1
5   spades    2
6   hearts    2
```

We may assign names to change the column names.

```
names(deck) <- c('suit', 'face')
head(deck)
```

```
      suit face
1   spades    1
2   hearts    1
3    clubs    1
4 diamonds    1
5   spades    2
6   hearts    2
```

Note that since `suit` and `face` are two vectors, `merge()` can also do the Cartesian product. `expand.grid()` is good for both vectors and data.frame.

```
merge(suit, face)
```

```
          x  y
1    spades  1
2    hearts  1
3     clubs  1
4  diamonds  1
5    spades  2
6    hearts  2
7     clubs  2
8  diamonds  2
9    spades  3
10   hearts  3
11    clubs  3
12 diamonds  3
13   spades  4
14   hearts  4
```

```
15    clubs  4
16 diamonds  4
17   spades  5
18   hearts  5
19    clubs  5
20 diamonds  5
21   spades  6
22   hearts  6
23    clubs  6
24 diamonds  6
25   spades  7
26   hearts  7
27    clubs  7
28 diamonds  7
29   spades  8
30   hearts  8
31    clubs  8
32 diamonds  8
33   spades  9
34   hearts  9
35    clubs  9
36 diamonds  9
37   spades 10
38   hearts 10
39    clubs 10
40 diamonds 10
41   spades 11
42   hearts 11
43    clubs 11
44 diamonds 11
45   spades 12
46   hearts 12
47    clubs 12
48 diamonds 12
49   spades 13
50   hearts 13
51    clubs 13
52 diamonds 13
```

43.2 Load data

43.2.1 build-in datasets

R has many build-in datasets. You may use `data()` to see all of them. Here are a few common datasets.

- `mtcars`: Motor Trend Car Road Tests: The data was extracted from the 1974 Motor Trend US magazine, and comprises fuel consumption and 10 aspects of automobile design and performance for 32 automobiles (1973–74 models)

```
data(mtcars)
```

- `iris`: iris data set gives the measurements in centimeters of the variables sepal length, sepal width, petal length and petal width, respectively, for 50 flowers from each of 3 species of iris. The species are Iris setosa, versicolor, and virginica.

```
data(iris)
```

- `ToothGrowth`: ToothGrowth data set contains the result from an experiment studying the effect of vitamin C on tooth growth in 60 Guinea pigs.

```
data(ToothGrowth)
```

- `PlantGrowth`: Results obtained from an experiment to compare yields (as measured by dried weight of plants) obtained under a control and two different treatment condition.

```
data(PlantGrowth)
```

- `USArrests`: This data set contains statistics about violent crime rates by us state.

```
data(USArrests)
```

43.2.2 Read from files

The build-in `read.csv()` function can directly read `.csv` file into a data.frame.

Example 43.4. We use the file `yob1880.txt` from Chapter 5 here. Put the file in the working folder and run the following code.

```
df <- read.csv('yob1880.txt', header = FALSE)
head(df)
```

We may also manually assign columns names.

```
names(df) <- c('name', 'sex', 'counts')
head(df)
```

```
      name sex counts
1     Mary   F   7065
2     Anna   F   2604
3     Emma   F   2003
4 Elizabeth   F   1939
5    Minnie   F   1746
6  Margaret   F   1578
```

43.3 Flow control

43.3.1 `for` loop

Example 43.5

```
for(x in 1:10){
    print(x)
}
```

```
[1] 1
[1] 2
[1] 3
[1] 4
[1] 5
[1] 6
[1] 7
[1] 8
[1] 9
[1] 10
```

43.3.2 if-else

Example 43.6.

```
a <- 200
b <- 33

if (b > a) {
  print("b is greater than a")
} else if (a == b) {
  print("a and b are equal")
} else {
  print("a is greater than b")
}
```

```
[1] "a is greater than b"
```

43.3.3 Functions

Example 43.7.

```
myfunction <- function() {
    die <- 1:6
    sum(die)
}

myfunction()
```

```
[1] 21
```

44 Exercises

45 Projects

Exercise 45.1. Start a R Markdown / Quarto file. In the first section write a R code block to print `Hello world!`.

References

[1] Youens-Clark, K. (2020). *Tiny python projects*. Manning Publications.

[2] McKinney, W. (2017). *Python for data analysis: Data wrangling with pandas, NumPy, and IPython*. O'Reilly Media.

[3] Shaw, Z. A. (2017). *Learn python 3 the hard way*. Addison Wesley.

[4] Sweigart, A. (2020). *Automate the boring stuff with python, 2nd edition practical programming for total beginners: Practical programming for total beginners*. No Starch Press.

[5] Klosterman, S. (2021). *Data science projects with python: A case study approach to gaining valuable insights from real data with machine learning*. Packt Publishing, Limited.

[6] Prabhakaran, S. (2018). 101 NumPy exercises for data analysis (python).

[7] Prabhakaran, S. (2018). 101 pandas exercises for data analysis.

[8] Beuzen, T. and Timbers, T. (2022). *Python packages*. Taylor & Francis Group.

[9] Wickham, H. and Grolemund, G. (2017). *R for data science: Import, tidy, transform, visualize, and model data*. O'Reilly Media.

[10] Grolemund, G. (2014). *Hands-on programming with r: Write your own functions and simulations*. O'Reilly Media.

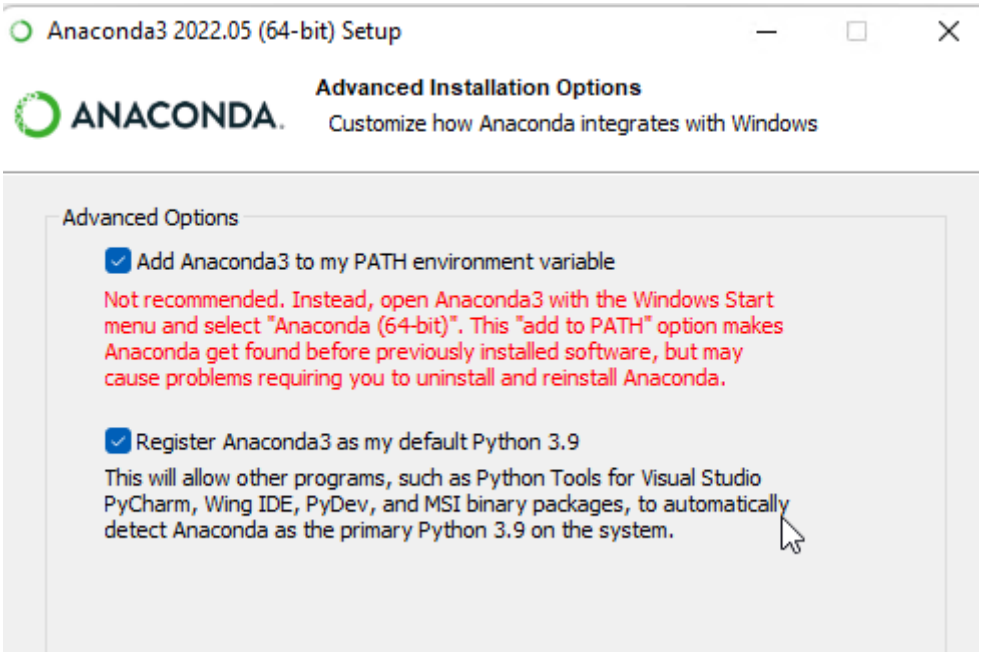
Part X

Setup

A VS Code + Anaconda

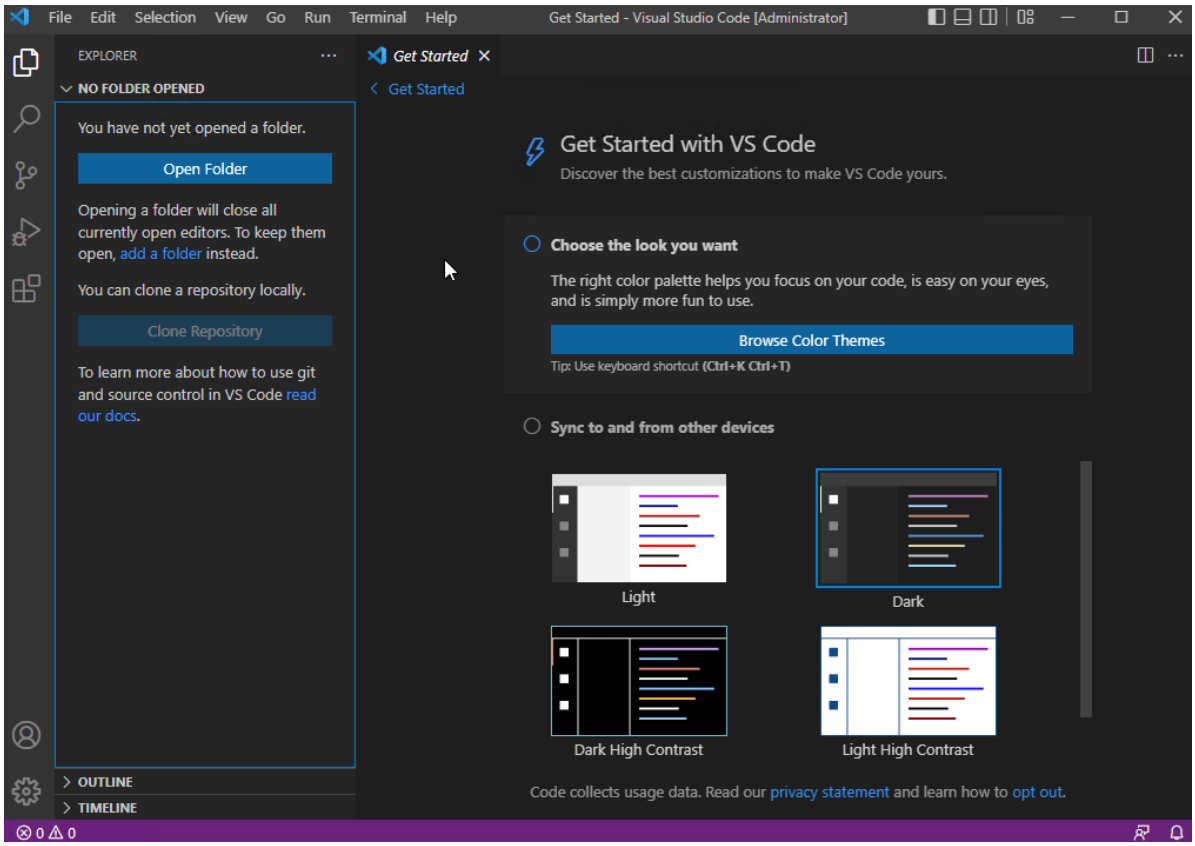
Note that all the following steps are tested in Windows 10/11. If you use other operation systems please contact me.

1. Go to Anaconda download page. Download and install Anaconda.
2. Go to VS Code download page. Download and install VS Code. Actually Anaconda contains one copy of VS Code. Here I just assume that some of you intall VS Code before Anaconda.
3. When installing VS Code, you may accept all default settings. When installing Anaconda, please pay attention to the `PATH` setting.

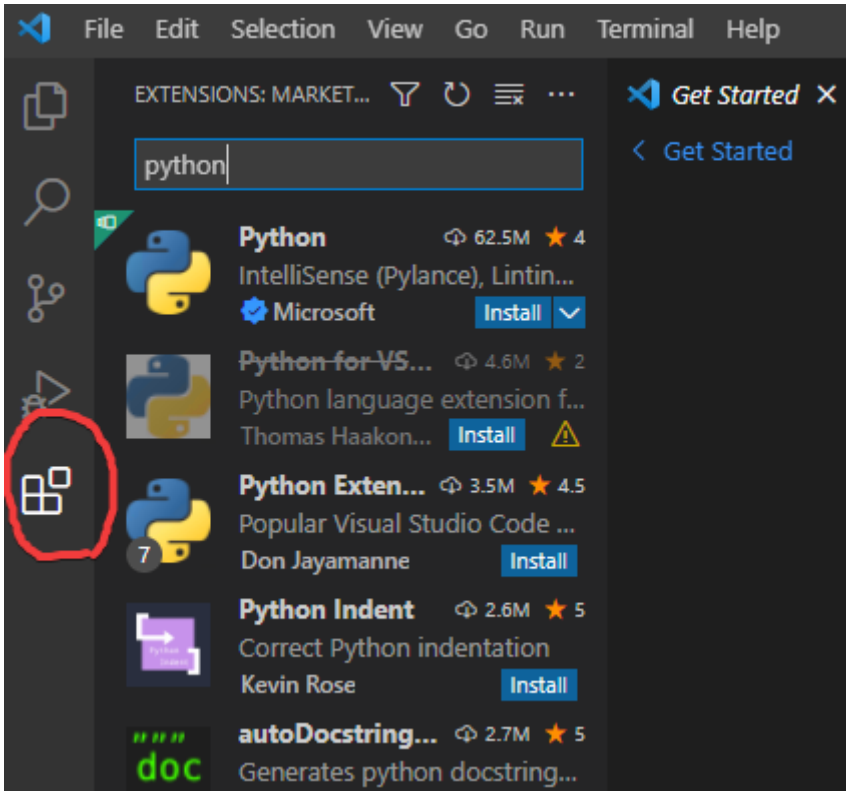


The first box is unchecked by default. This setting is related to the ability to easily run Python code in Terminals. I recommend you to check it. If you don't check it during this step, you may add it to the system environment variable `PATH` manually later.

4. The UI of VS Code looks as follows.


File Edit Selection View Go Run Terminal Help
Get Started - Visual Studio Code [Administrator]
EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all currently open editors. To keep them open, add a folder instead.
You can clone a repository locally.
Clone Repository
To learn more about how to use git and source control in VS Code read our docs.
OUTLINE
TIMELINE
Get Started
Get Started
Get Started with VS Code
Discover the best customizations to make VS Code yours.
Choose the look you want
The right color palette helps you focus on your code, is easy on your eyes, and is simply more fun to use.
Browse Color Themes
Tip: Use keyboard shortcut (Ctrl+K Ctrl+T)
Sync to and from other devices
Light
Dark
Dark High Contrast
Light High Contrast
Code collects usage data. Read our privacy statement and learn how to opt out.
0 0

Please look at the fifth tab from the left sidebar. It is the Extension tab.



Please search for `python` and install the first Python extension from Microsoft. It will actually install five extensions. These are all we need for now.

5. After all are installed, go to the first Explorer tab on the left side bar, and `Open Folder`. This is the working directory for your project.

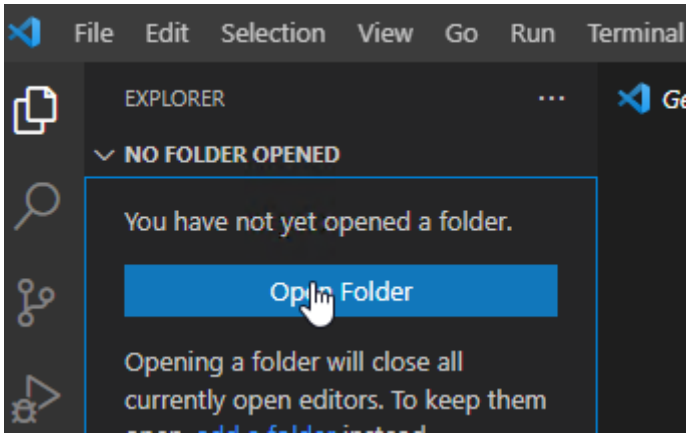
Choose one folder and start a new `.py` file.

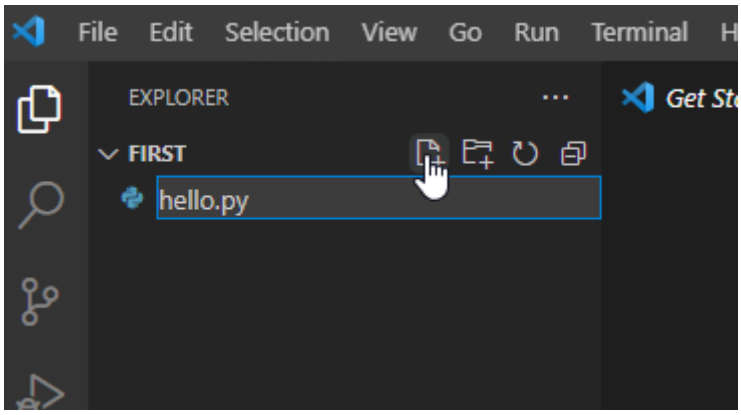
6. If everything is setup correctly, you may see the Python version and environment name at the right lower corner. In our case the environment name is `base`. We will need it in the future.

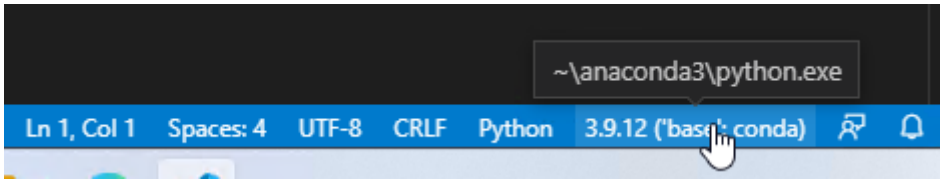
Note that we are not looking at the `Python` for Language Mode. If you see `Select Interpreter` there, it means that VS Code doesn't find your Python interpreter. Please restart VS Code or select it manually, or check whether Anaconda is installed correctly.

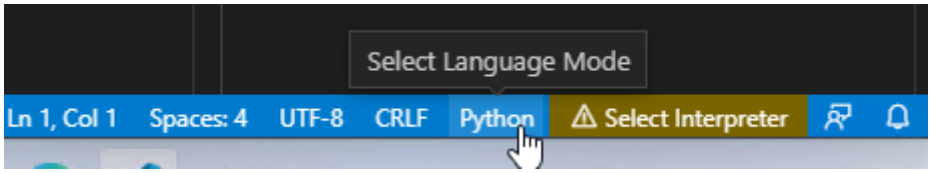
To check whether everything is setup correctly, please run the following tests.

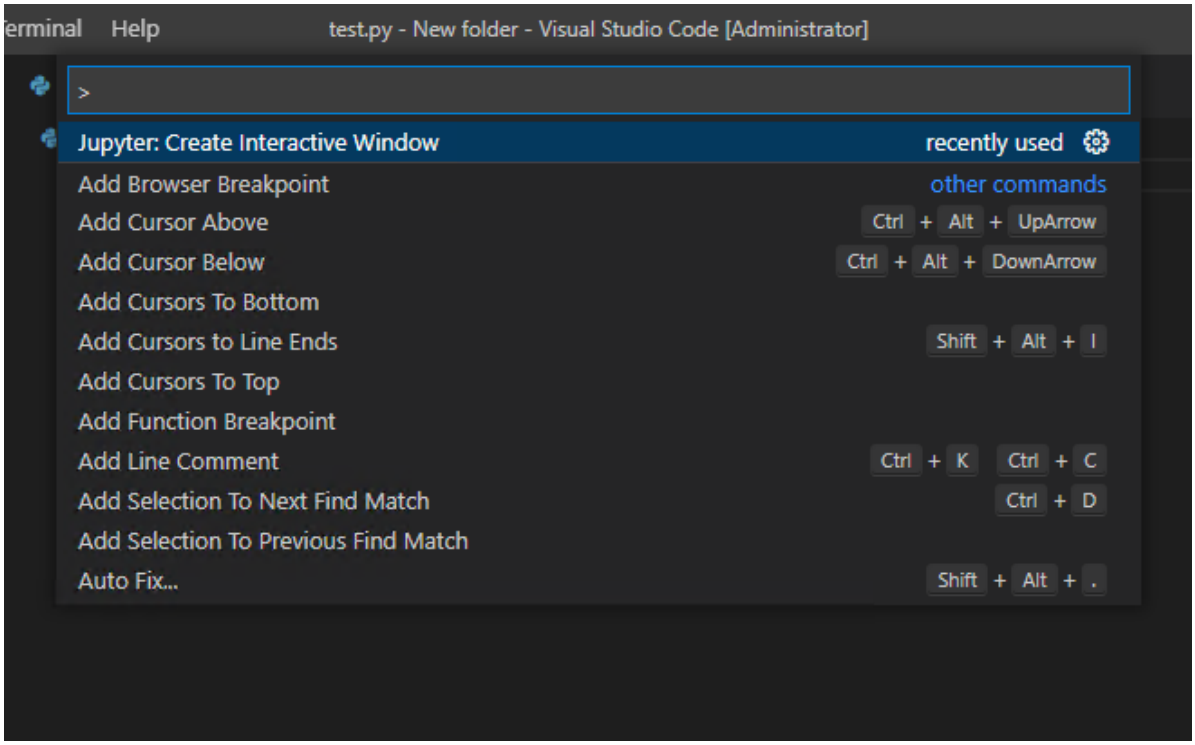
1. Use `ctrl+shift+p` to open the Command Palette, type "Jupyter: Create Interactive Window" and press `enter` to open the Jupyter interactive window.


File Edit Selection View Go Run Terminal
EXPLORER
NO FOLDER OPENED
You have not yet opened a folder.
Open Folder
Opening a folder will close all
currently open editors. To keep them


File Edit Selection View Go Run Terminal
EXPLORER
FIRST
hello.py


~\anaconda3\python.exe
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
Python


Select Language Mode
Ln 1, Col 1
Spaces: 4
UTF-8
CRLF
Python
Select Interpreter

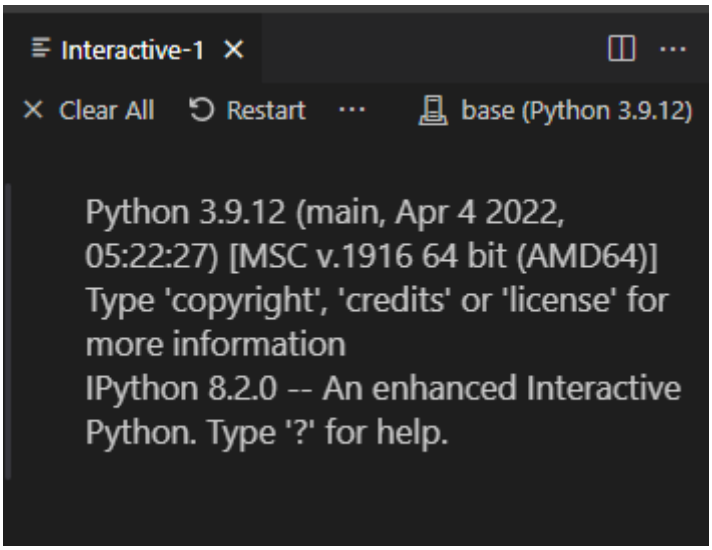


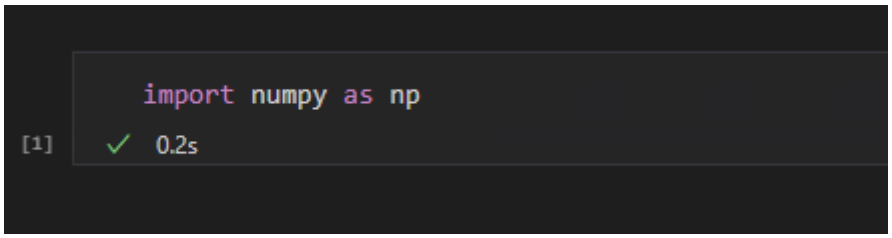
If the interactive window starts and you see the loading infomation of your kernel as follows, especially you see the environment name on the right upper corner, then you get everything correctly. However we will still do more tests.

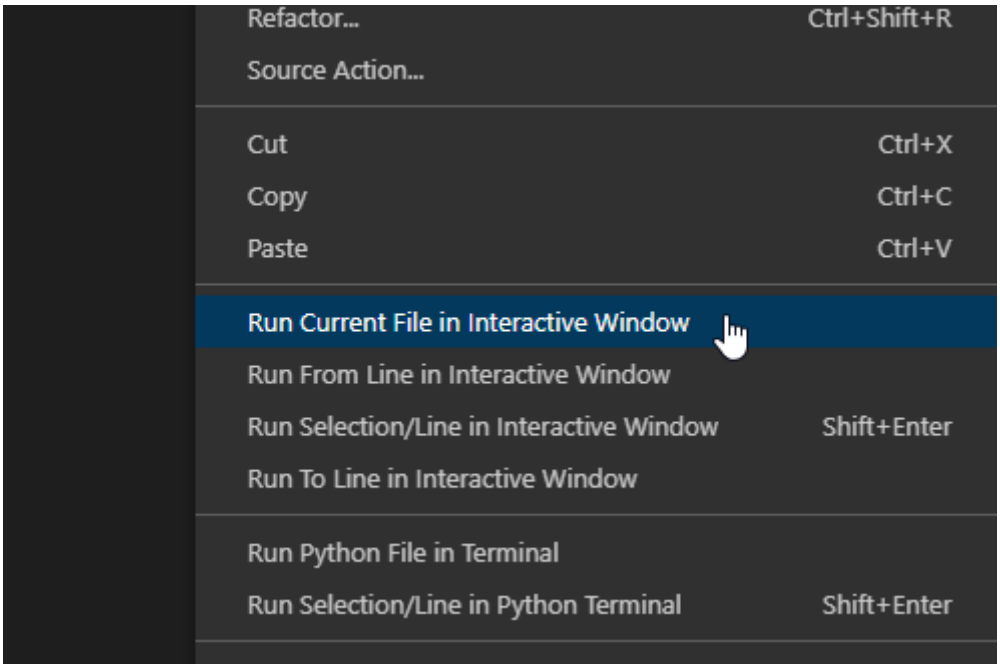
2. In the window type `import numpy as np` to test whether you are able to import packages. If you don't see any error messages then it means good.

3. In the editor window, type `import numpy as np` and right click the body to choose `Run Current File in Interactive Window`, and see whether it runs in interactive window.

4. Open the terminal. Please use `Command Prompt` instead of `Powershell`. Activate the conda environment by type the command `conda activate base` in the example above. Please change the name to match your own environment. If `conda` cannot be recognized, please register Python and Anaconda to the system environment path. Please see the next Appendix for details.


Interactive-1
Clear All
Restart
base (Python 3.9.12)
Python 3.9.12 (main, Apr 4 2022, 05:22:27) [MSC v.1916 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 8.2.0 -- An enhanced Interactive Python. Type '?' for help.


import numpy as np
[1]
0.2s


Refactor... Ctrl+Shift+R
Source Action...
Cut Ctrl+X
Copy Ctrl+C
Paste Ctrl+V
Run Current File in Interactive Window
Run From Line in Interactive Window
Run Selection/Line in Interactive Window Shift+Enter
Run To Line in Interactive Window
Run Python File in Terminal
Run Selection/Line in Python Terminal Shift+Enter

B Google Colab

Google Colab is a product from Google Research, that allows anybody to write and execute arbitrary Python code through the browser, and is especially well suited to machine learning, data analysis and education.

Here is the link to Google Colab. To use it you should have a Google account. Otherwise it is very simple to start, since a lot of packages for our course are already installed.

B.1 Install packages

If you would like to install more packages, you can type `%pip install` + package name in a code cell and execute it.

The drawback here is that Google Colab can only stay for 24 hours. After that, all additionaly installed packages will be earsed. However you can put `%pip install` + package name at the beginning of your notebook and these packages will be installed every time you run the notebook.

B.2 Upload files

You may directly upload files to the working directory of Google Colab. This has to be done in the browser. When working with these files, you may just use relative paths.

The drawback here is that Google Colab can only stay for 24 hours. After that, although your `.ipynb` files will be stores, all other files will be earsed.

B.3 Mount Google Drive

One way to let the uploaded files stay in cloud is to upload them to Google Drive, and then load your Google Drive contents from Google Colab.

Goole Drive is a cloud storage service provided by Google. When you register a Google account you will be automatically assigned a Google Drive account. You may get access to it from this link.

Here are the steps to mount Google Drive:

1. Upload your files to your Google Drive.
2. Run the following codes in Colab code cells before you are loading the uploaded files:

```
from google.colab import drive
drive.mount('/content/gdrive')
```

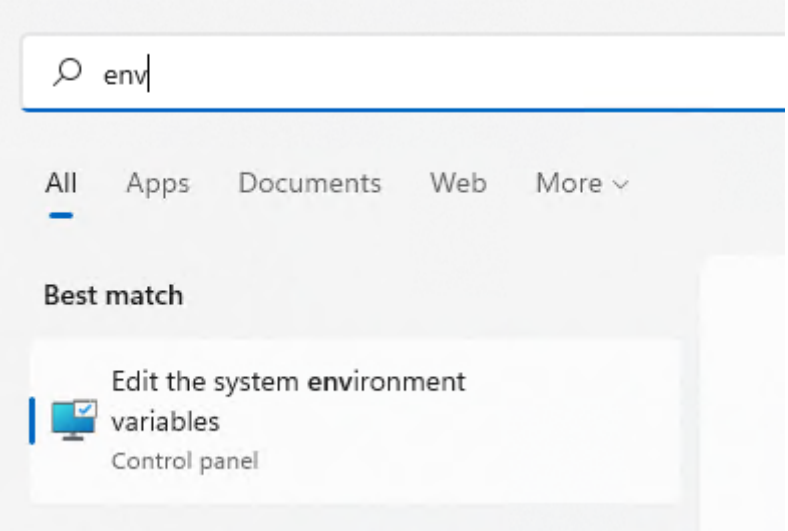
3. A window pop up asking you about the permission. Authorize and the drive is mounted.
4. To work in directories, the most popular commands are
    - `%ls`: list all files and folders in the working directory.
    - `%cd` + folder name: Get into a specific folder.
    - `%cd..`: Get into the parent folder. Then use these commands to find the files your just uploaded.
5. Finally you may directly get access to those files just like they are in the working directory.

Part XI

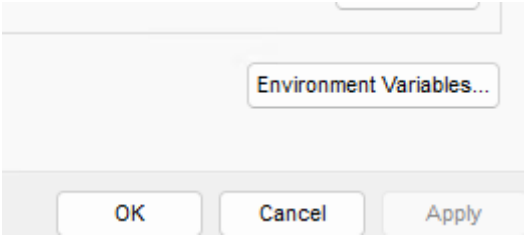
PATH

Here are the steps to edit the system environment variables in Windows 10/11.

1. First in the start menu search for `Edit the system environment variables`.

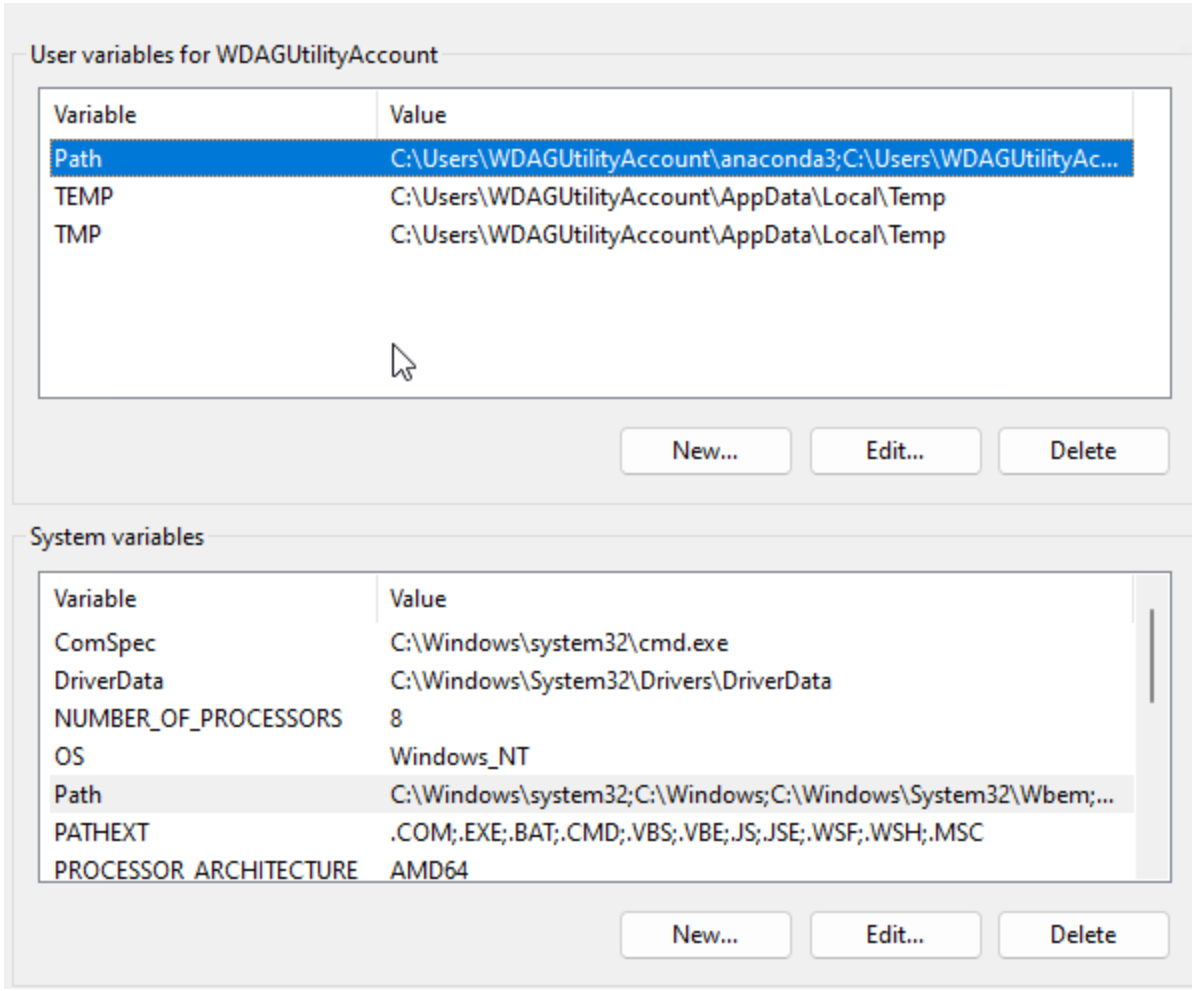


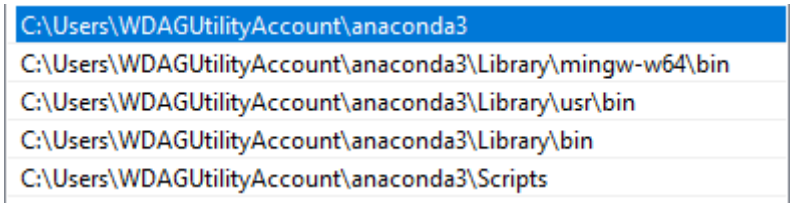
2. Then click the `Environment Variables...` button at the right lower corner.



3. Find the `Path` variable in either the upper window or the lower window. Use which one depends on whether you want to register the variable for the user or for the machine. In this example I add for the user.

4. Finally double click the variable and add the following path to it. You need to make changes according to your installation. I recommend you to locate your Anaconda installation first to get the path.


User variables for WDAGUtilityAccount
Variable
Value
Path
C:\Users\WDAGUtilityAccount\anaconda3;C:\Users\WDAGUtilityAc...
TEMP
C:\Users\WDAGUtilityAccount\AppData\Local\Temp
TMP
C:\Users\WDAGUtilityAccount\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS
8
OS
Windows_NT
Path
C:\Windows\system32;C:\Windows;C:\Windows\System32\Wbem;...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
New...
Edit...
Delete


C:\Users\WDAGUtilityAccount\anaconda3
C:\Users\WDAGUtilityAccount\anaconda3\Library\mingw-w64\bin
C:\Users\WDAGUtilityAccount\anaconda3\Library\usr\bin
C:\Users\WDAGUtilityAccount\anaconda3\Library\bin
C:\Users\WDAGUtilityAccount\anaconda3\Scripts

Part XII

Virtual environments