type(x) 输出 x 的类别

isinstance(x,y) 输出 x 是否是 y 的类别

- Use the following functions to conduct logical test and coerce objects.

Type	Checking	Coercing
numeric	`isinstance(var, (int, float))`	`float(var)` or `int(var)`
logical	`isinstance(var, bool)`	`bool(var)`
character	`isinstance(var, str)`	`str(var)`
categorical	`isinstance(var, pd.Categorical)`	`pd.Categorical(var)`

x.lower() 输出 x 的全部字符小写

x.split(sep="*") 输出 array，里面元素是 x 的分割,里面参数是分隔符

(a:b) b is not included

Np.random.randint(起始范围，结束范围，size=样本数据数量)

```
# Simulate two rolls of a six-sided die
roll = np.random.randint(1, 7, size = 2)
print(roll)
```

关于 counter 函数的用法

Python标准库 collections 里的 counter() 函数是一个计数器工具，用于统计可迭代对象中元素出现的次数，并返回一个字典（key-value）key 表示元素，value 表示各元素 key 出现的次数，可为任意整数 (即包括0与负数)。

可接受参数： 任何可迭代对象，如列表、元组、字符串、字典等。

A Counter is a dict subclass for counting hashable objects. It is an unordered collection where elements are stored as dictionary keys and their counts are stored as dictionary values. Counts are allowed to be any integer value including zero or negative counts.

一、创建 Counter()

```
# 创建一个新的空Counter
c = collections.Counter()

# 或者
from collections import Counter
c = Counter()
```

复制

当参数是列表，字符串，映射关系，关键字参数时：

```
# 参数是字符串
c = collections.Counter("abracadabra")
print(c)  # Counter({'a': 5, 'b': 2, 'r': 2, 'c': 1, 'd': 1})

# 参数是映射
d = collections.Counter({"red":4,"yellow":2,"blue":5})
print(d) # Counter({'blue': 5, 'red': 4, 'yellow': 2})

# 参数是关键字参数
e = collections.Counter(cats=2,dogs=5)
print(e) # Counter({'dogs': 5, 'cats': 2})

# 参数是列表，并访问 key
c = collections.Counter(["apple","pear","apple"])
print(c)           # Counter({'apple': 2, 'pear': 1})
print(c["orange"]) # 0
```

Np.median

Np.arange(起始，结束，间隔）：注意起始到结束是左闭右开区间

np.linspace()

To repeat a sequence, we use the np.tile() function.

```
seq4 = np.tile(seq3, 2)
print(seq4)
```

```
[1 1 1 1 1 1 1 1 1 1]
```

Define a function , we need to clarify three parts of our function : the argument , the default argument ,and the output of the function.

Pd.read_excel(file_name,sheet_name)

file_name 可以是相对目录下的文件名，也可以是绝对目录下的文件名

Lambda 函数

Lambda函数的一般语法很简单，lambda关键字定义，后面跟着参数列表和一个表达式。

```
lambda arguments: expression
```

运行

其中：

- lambda是定义Lambda函数的关键字。
- arguments是Lambda函数的参数列表，可以有零个或多个参数，多个参数之间用逗号分隔。
- expression是Lambda函数的表达式，即函数的具体实现逻辑。

简单的用法：

```
# 定义一个简单的Lambda函数，对传入的参数求平方
square = lambda x: x * x

# 调用Lambda函数
result = square(5)
print(result)  # 输出: 25
```

运行

Apply 函数

apply(self, func, axis=0, raw=False, result_type=None, args=(), **kwds):

- **func:** 应用于每一列或每一行的函数，这个函数可以是Python内置函数、Pandas或其他库中的函数、自定义函数、匿名函数。
- **axis:** 设置批处理函数按列还是按行应用，0或index表示按列应用函数，1或columns表示按行应用函数，默认值为0。
- **raw:** 设置将列/行作为Series对象传递给函数，还是作为ndarray对象传递给函数。raw是bool类型，默认为False。

False: 将列/行作为Series对象传递给函数。

True: 将列/行作为ndarray对象传递给函数。apply中的func函数将接收ndarray对象，如果应用numpy中的函数，这样可以提升性能。

- **result_type:** 当axis=1时，设置返回结果的类型和样式，支持{'expand', 'reduce', 'broadcast', None}四种类型，默认为None。

expand: 列表式的结果将被转化为列。

reduce: 如果可能的话，返回一个Series，而不是返回列表式的结果。这与expand相反。

broadcast: 结果将被广播成DataFrame的原始形状，DataFrame的原始索引和列将被保留。

None: 结果取决于应用函数的返回值，列表式的结果将以Series形式返回，如果应用函数返回Series将会扩展到列。

- **args:** 传给应用函数func的位置参数，args接收的数据类型为元组，如果只有一个位置参数要注意加逗号。
- ****kwds:** 如果func中有关键字参数，可以传给**kwds。

raw和result_type通常不需要自己设置，保持默认即可。

raw和result_type通常不需要自己设置，保持默认即可。

下面依次介绍apply()函数的各种用法。

传入不同类型的函数

```
import numpy as np

df = pd.DataFrame({'Col-1': [1, 3, 5], 'Col-2': [2, 4, 6], 'Col-3': [9, 8, 7], 'Col-4': [3, 6, 9]},
                  index=['A', 'B', 'C'])
print(df)
df1 = df.apply(max)  # python内置函数
print('-' * 30, '\n', df1, sep='')
df2 = df.apply(np.mean)  # numpy中的函数
print('-' * 30, '\n', df2, sep='')
df3 = df.apply(pd.DataFrame.min)  # pandas中的方法
print('-' * 30, '\n', df3, sep='')

```

```
   Col-1  Col-2  Col-3  Col-4
A      1      2      9      3
B      3      4      8      6
C      5      6      7      9
------------------------------
Col-1    5
Col-2    6
Col-3    9
Col-4    9
dtype: int64
------------------------------
Col-1    3.0
Col-2    4.0
Col-3    8.0
Col-4    6.0
dtype: float64
------------------------------
Col-1    1
Col-2    2
Col-3    7
Col-4    3
dtype: int64
```

map 函数

Melt 函数，将数据表转化为 tidy 的格式。

Pd.DataFrame.melt(id_var ,var_name ,value_name)

id_var 是转化后索引

var_name 是堆到列上的变量名

value_name 是要填在值上的变量名

例如：	

pivot 函数

conditional_join

```
import janitor as jn
sales.conditional_join(promos, ("sales_date", "promo_date", ">="),
                       how = "inner")
```

```
  sales_date promo_date  promo_price
0 2024-09-14 2024-09-09          179
1 2024-09-17 2024-09-09          179
2 2024-09-17 2024-09-15          179
```

Np.select

- We can use `.select()` to classify flight departure status based on `dep_delay`.

```
df["dep_status"] = np.select(
    [df["dep_delay"] > 0, df["dep_delay"] < 0, df["dep_delay"] == 0],
    ["delayed", "early", "on time"],
    default = "unknown"
)
df[["dep_delay", "dep_status"]]
```

Df.dtypes

```
df.dtypes
```

```
month              int64
day                int64
dep_time         float64
sched_dep_time     int64
dep_delay        float64
carrier           object
origin            object
dest              object
sched_hour         int64
dep_status        object
dtype: object
```

Df.select_dtypes()

```
df.select_dtypes("int")
```

	month	day	sched_dep_time	sched_hour
0	9	2	2255	22
1	9	2	2359	23
2	9	2	545	5
3	9	2	545	5
4	9	2	600	6
..	...	...	...	...
303	9	2	2035	20
304	9	2	1835	18
305	9	2	1935	19
306	9	2	1645	16
307	9	2	1940	19

[308 rows x 4 columns]

df.columns.str.startswith()

- We can get all columns that begins with "sched":

```
df.loc[:, df.columns.str.startswith("sched")]
```

	sched_dep_time	sched_hour
0	2255	22
1	2359	23
2	545	5
3	545	5
4	600	6
..	...	...
303	2035	20
304	1835	18
305	1935	19
306	1645	16
307	1940	19

df.filter(regex = "^sched")

- Alternatively, use the `.filter()` method with regular expression.

```
df.filter(regex = "^sched")     # Replace the ^ symbol manually in VSCode
```

```
      sched_dep_time  sched_hour
0               2255          22
1               2359          23
2                545           5
3                545           5
4                600           6
..               ...         ...
303             2035          20
304             1835          18
305             1935          19
306             1645          16
307             1940          19

[308 rows x 2 columns]
```

df.columns.str.replace("_time", "")

Pip install command : py -m pip install jupyterlab notebook (windows)

Two principles:

1.Everything in Python is an object.

2.Every object has a type (class).

Avoid the predefined words

Python 区分大小写

pandas 加减是 based on the index

有两种方式来设定 pd.DataFrame

1.from a list

```
# Method 1: From a list
df = pd.DataFrame([["Tom", "Python", 5],
                   ["Mike", "Python", 4],
                   ["Tiffany", "R", 7]],
                   columns = ["Name", "Language", "Courses"])
print(df)
```

```
      Name Language  Courses
0      Tom   Python        5
1     Mike   Python        4
2  Tiffany        R        7
```

2.from a dictionary

```
# Method 2: From a dictionary
df = pd.DataFrame({"Name": ["Tom", "Mike", "Tiffany"],
                   "Language": ["Python", "Python", "R"],
                   "Courses": [5, 4, 7]})
print(df)
```

```
      Name Language  Courses
0      Tom   Python        5
1     Mike   Python        4
2  Tiffany        R        7
```

从 DataFrame 索引，有以下几个方式：

There are several ways to select data from a DataFrame:

- [] and [[]]
- .loc[] and .iloc[]
- Boolean indexing
- query()

1.[]和[[]]

```
# returns a DataFrame with one column
df[["Name"]]
```

```
      Name
0      Tom
1     Mike
2  Tiffany
```

```
# returns a DataFrame with two columns
df[["Name", "Language"]]
```

```
      Name Language
0      Tom   Python
1     Mike   Python
2  Tiffany        R
```

如果要索引好几个列的话就用列表形式来框选出来

2.iloc 和 loc

.iloc[] and loc[] are more flexible alternatives for accessing data.

- .iloc[] accepts **integer(s)** as references to rows/columns.

```
df.iloc[0]
```

```
Name           Tom
Language    Python
Courses          5
Name: 0, dtype: object
```

```
df.iloc[0:2]
```

```
   Name Language  Courses
0   Tom   Python        5
1  Mike   Python        4
```

iloc 是用 index 索引的，用数字索引

- .loc[] accepts **labels** as references to rows/columns.

```
df.loc[:, "Name"]
```

```
0        Tom
1       Mike
2    Tiffany
Name: Name, dtype: object
```

```
df.loc[:, "Name":"Language"]
```

```
      Name Language
0      Tom   Python
1     Mike   Python
2  Tiffany        R
```

loc 是用 labels 来索引，用 str 来索引

第一个参数是 row（行参数），第二个参数是 column（列参数）

3.boolean 索引

输入一个的 bool 列表来的进行索引

Just as with series, we can select data based on **boolean conditions**:

```
df[df["Language"] == "Python"]
```

```
   Name Language  Courses
0   Tom   Python        5
1  Mike   Python        4
```

```
df[(df["Language"] == "Python") & (df["Courses"] > 4)]
```

```
  Name Language  Courses
0  Tom   Python        5
```

其中 df["language"] == "Python" 返回的是一个 bool 列表

4.query 索引

.query() is a powerful tool for filtering data (similar to SQL).

- It accepts a **string expression** to evaluate the conditions.

```
df.query("Courses > 4 & Language == 'Python'")
```

```
  Name Language  Courses
0  Tom   Python        5
```

函数里面输入字符串

- .query() also allows us to reference variable in the current workspace.

```
language_choice = "Python"
df.query("Language == @language_choice")
```

```
   Name Language  Courses
0   Tom   Python        5
1  Mike   Python        4
```

- The @ symbol indicates that language_choice is defined outside the query string.

以上字符串里面和外界变量互动

Df.replace("before",'after')替换值

Df.map

```
client[col] = client[col].map({"yes": 1, "no": 0, "unknown": 0})
```

用 map 函数进行全局操作

```
pd.to_datetime(campaign["last_contact_date"], format="%Y-%b-%d")
```

一、pd.to_datetime基本使用

pd.to_datetime是Pandas库中的函数，可以将字符串转换为Timestamp格式，并且可以转换多种格式的时间字符串。

```
pd.to_datetime(arg, format=None, errors='raise', dayfirst=False, yearfirst=False, utc=None, box=True, exact=True, ur
```

- arg表示待转换的字符串；
- format表示待转换的字符串的格式描述；
  在format的格式字符串中，各种时间元素都有特定的字符表示，例如：
  %Y表示四位数的年份，
  %m表示两位数的月份，
  %d表示两位数的日期，
  %H表示小时数（24小时格式），
  %M表示分钟数，
  %S表示秒数。
- errors表示如果出现无法解析的字符串时会发生什么，raise表示引发异常，coerce表示转换为NaT（Not-a-Time）对象，ignore表示返回原始值。
- unit 参数用于指定输入数据的时间单位。它允许将输入的数据解释为不同时间精度的时间戳。下面是一些常用的 unit 参数选项及其含义：
  'ns'：纳秒（默认值）
  'us'：微秒
  'ms'：毫秒
  's'：秒
  'm'：分钟
  'h'：小时
  'D'：天
  'M'：月
  'Y'：年
  使用 unit 参数，可以确保将输入数据正确地解释为相应的时间单位。例如，如果输入数据是以秒为单位的时间戳，可以通过将 unit='s' 传递给 pd.to_datetime() 来指定单位为秒。

二、获取当前日期时间戳

```
from datetime import datetime
import pandas as pd
now = pd.Timestamp(datetime.now())
print(now)
```

输出结果：

```
2023-07-15 21:12:52.952046
```

三、时间差计算

```
import pandas as pd
date1 = pd.Timestamp('2022-12-01 12:30:00')
date2 = pd.Timestamp('2023-5-03 07:08:14')
print(date2 - date1)
```

输出结果：

```
152 days 18:38:14
```

Df.drop()

campaign.drop(columns=["month", "day", "year"], inplace=True)

Df.to_csv()
client.to_csv("client.csv", index=False)

Index = False 可以设置不传输索引

```
# Replace specific parts of column names
df.columns = df.columns.str.replace("_time", "")
df.head(3)
```

```
   month  day    dep  sched_dep  ...  origin dest sched_hour dep_status
0      9    2    8.0       2255  ...     JFK  BUF         22    delayed
1      9    2   15.0       2359  ...     JFK  SJU         23    delayed
2      9    2  537.0        545  ...     JFK  MIA          5      early

[3 rows x 10 columns]
```

df.reindex(sorted(df.columns), axis = 1)

```
# Sort columns in alphabetical order
df.reindex(sorted(df.columns), axis = 1)
```

```
    carrier  day    dep  dep_delay  ... month origin  sched_dep sched_hour
0        B6    2    8.0       73.0  ...     9    JFK       2255         22
1        B6    2   15.0       16.0  ...     9    JFK       2359         23
2        AA    2  537.0       -8.0  ...     9    JFK        545          5
3        B6    2  542.0       -3.0  ...     9    JFK        545          5
4        B6    2  557.0       -3.0  ...     9    JFK        600          6
..      ...  ...    ...        ...  ...   ...    ...        ...        ...
303      9E    2    NaN        NaN  ...     9    JFK       2035         20
304      9E    2    NaN        NaN  ...     9    JFK       1835         18
305      9E    2    NaN        NaN  ...     9    JFK       1935         19
306      MQ    2    NaN        NaN  ...     9    JFK       1645         16
307      MQ    2    NaN        NaN  ...     9    JFK       1940         19
```

df.groupby("carrier")[["dep_delay"]].mean()

```
df.groupby("carrier")[["dep_delay"]].mean()
```

```
         dep_delay
carrier
9E       56.304348
AA       18.142857
B6       36.484127
DL       34.950820
EV      125.333333
HA       -4.000000
MQ       56.235294
UA       16.818182
```

The syntax is `df.groupby("group_var")[["col_name"]].agg("agg_function")`.

The common functions we can pass to `agg()` are the following:

Aggregation function	Description
size()	Number of items
first() , last()	First and last item
mean() , median()	Mean and median
min() , max()	Min and max
std() , var()	Standard deviation and variance
sum()	Sum of all items

Aggregation summarizes each group down to one row.

with open(file_path, "r") as file:

Let's read in real-time data on minute-by-minute bus arrival times from every bus stop in Singapore.

- The data were obtained from the LTA Data Mall and available as `wk3_BusArrival.json` on Canvas.

```
import json
file_path = "../data/wk3_BusArrival.json"
# Read the JSON file
with open(file_path, "r") as file:
    bus_arrival = json.load(file)
# Type of the parsed data
type(bus_arrival)
```

```
<class 'dict'>
```

pd.json_normalize(data = bus_arrival, record_path = ["Services"])

- Extract information and convert it into a useful data frame.

```
df_bus_arrival = pd.json_normalize(data = bus_arrival,
                                   record_path = ["Services"])
df_bus_arrival.head()
```

```
  ServiceNo Operator  ... NextBus.Lat NextBus.Long
0       176     SMRT  ...    1.301219   103.762202
1        78      TTS  ...     1.30693    103.73333

[2 rows x 7 columns]
```

用 urllib 进行数据请求

```
from urllib.request import urlopen

with urlopen('http://localhost:3000/lyrics/') as response:

  # Use the correct function to read the response data from the response object
  data = response.read()
  encoding = response.headers.get_content_charset()

  # Decode the response data so you can print it as a string later
  string = data.decode(encoding)

  print(string)
```

结果如下

```
from urllib.request import urlopen

with urlopen('http://localhost:3000/lyrics/') as response:

  # Use the correct function to read the response data from the response object
  data = response.read()
  encoding = response.headers.get_content_charset()

  # Decode the response data so you can print it as a string later
  string = data.decode(encoding)

  print(string)

N' I never miss Cause I'm a problem child - AC/DC, Problem Child
```

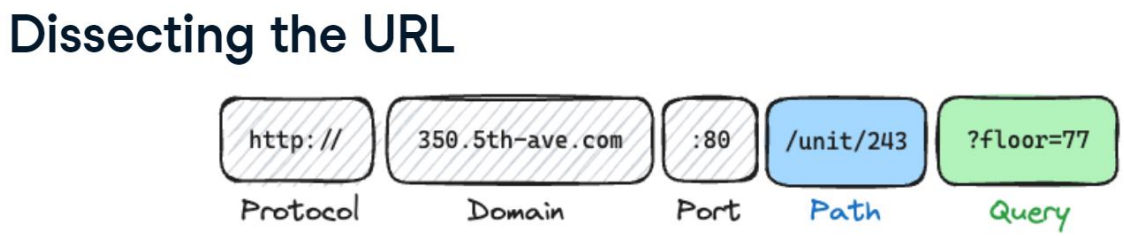
用 request 包进行 api 调用

```python
# Import the requests package
import requests

# Pass the API URL to the get function
response = requests.get('http://localhost:3000/lyrics')

# Print out the text attribute of the response object
print(response.text)
```

结果与 urllib 包调用相似

可注意到的是，urllib 包使用类似于 open 函数进行的数据读取的操作，需要用 with 句式创造一个 response 对象(仅仅是名称，不是类别)来进行操作。而 request 包则更加简单，直接操作即可。

URL 格式



- **Protocol** = the means of transportation
- **Domain** = the street address of the office building
- **Port** = the gate or door to use when entering the building
- **Path** = the specific office unit inside the building
- **Query** = any additional instructions

Adding query parameters with requests

```
# Append the query parameter to the URL string
response = requests.get('http://350.5th-ave.com/unit/243?floor=77&elevator=True')
print(response.url)
```

```
http://350.5th-ave.com/unit/243?floor=77&elevator=True
```

- Use the `params` argument to add query parameters

```
# Create dictionary
query_params = {'floor': 77, 'elevator': True}
# Pass the dictionary using the `params` argument
response = requests.get('http://350.5th-ave.com/unit/243', params=query_params)
print(response.url)
```

在调用 URL 的时候，可以直接键入对应的 URl 地址，也可以用 params 参数进行调用

返回第一个n行排序columns降序排列。

返回第一个n在中具有最大值的行columns，按降序排列。未指定的列也将返回，但不用于排序。

此方法等效于df.sort_values(columns, ascending=False).head(n)，但性能更高。

参数：

n：int要返回的行数。

columns：label 或 list of labels要排序的列标签。

keep：{‘first’, ‘last’, ‘all’}, 默认为 ‘first’有重复值的地方：

first：优先处理首次出现的事件

last：优先排列最后一次出现的

alldo not drop any duplicates, even it means选择多个n项目。