



Data analysis for Machine Learning





Disclaimer

This document is a training document. Therefore, it cannot be considered a specification for the system.

Due to ongoing advances in functions, design and manufacturing, the contents of this document are subject to revision without notice.

Ericsson accepts no responsibility for any errors or damages arising from the use of this document.

This document is not intended to replace the technical documentation provided with your system. Always refer to this technical documentation during operation and maintenance .

© Ericsson AB 2020

This document is produced by Ericsson plc.

This document is for training purposes only and may not be used in any way without the express written permission of Ericsson.

Copy, disclose or distribute this document.

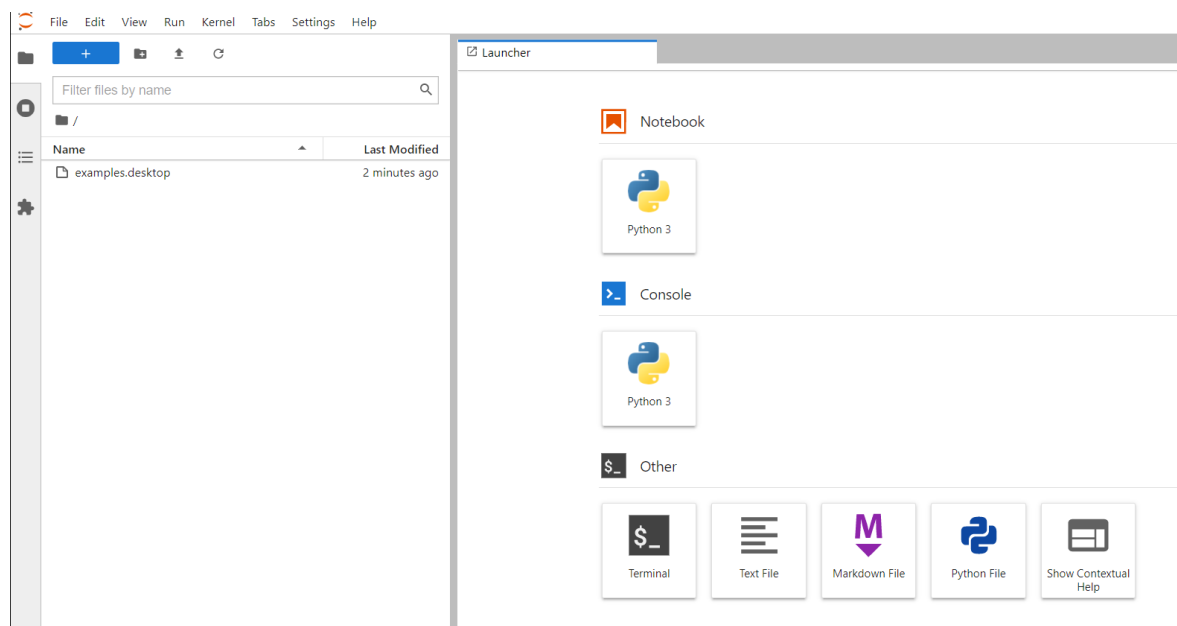


1. Lab environment access and confirmation

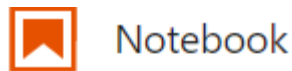
- Open the browser and access the Jupyter Hub address provided by the instructor.
- Enter your assigned account and password

The image shows a JupyterHub sign-in page. At the top is an orange header with the text "Sign in". Below this is a yellow warning box that reads: "Warning: JupyterHub seems to be served over an unsecured HTTP connection. We strongly recommend enabling HTTPS for JupyterHub." Under the warning, there are two input fields: "Username:" and "Password:". Below the password field is an orange "Sign in" button.

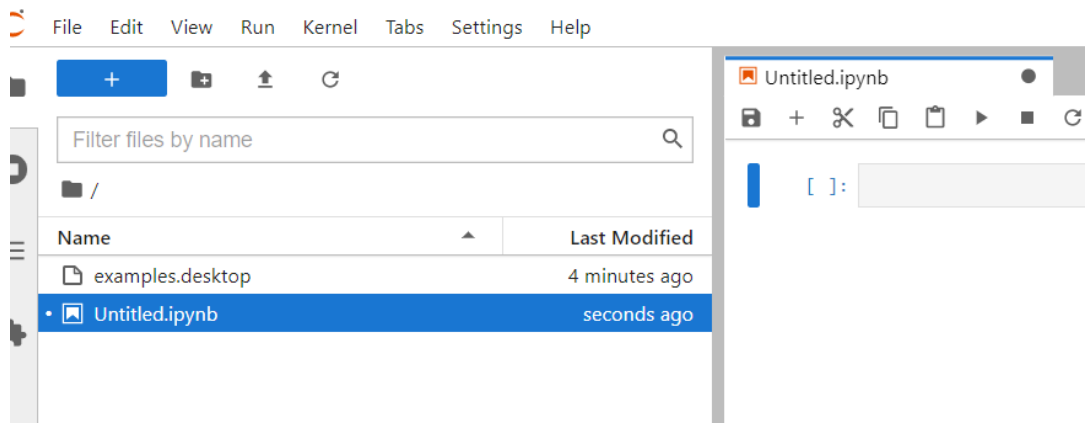
- After successfully logging in, you can see the following interface:



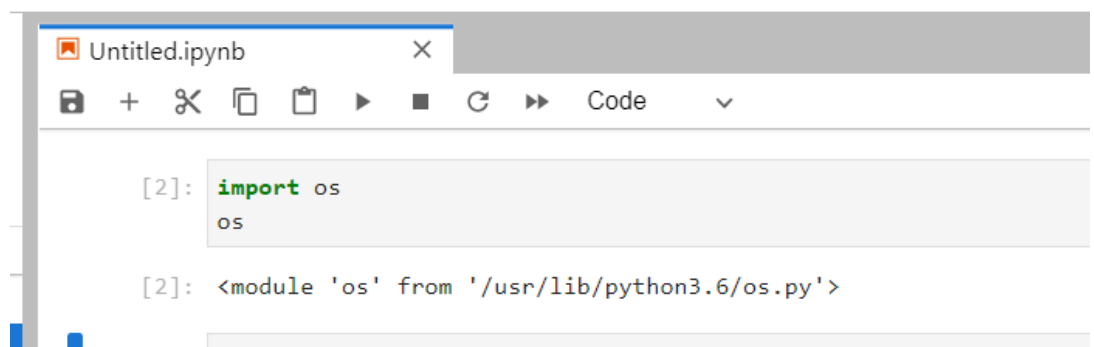
d) Click Notebook on the right Icon attempts to create a notebook



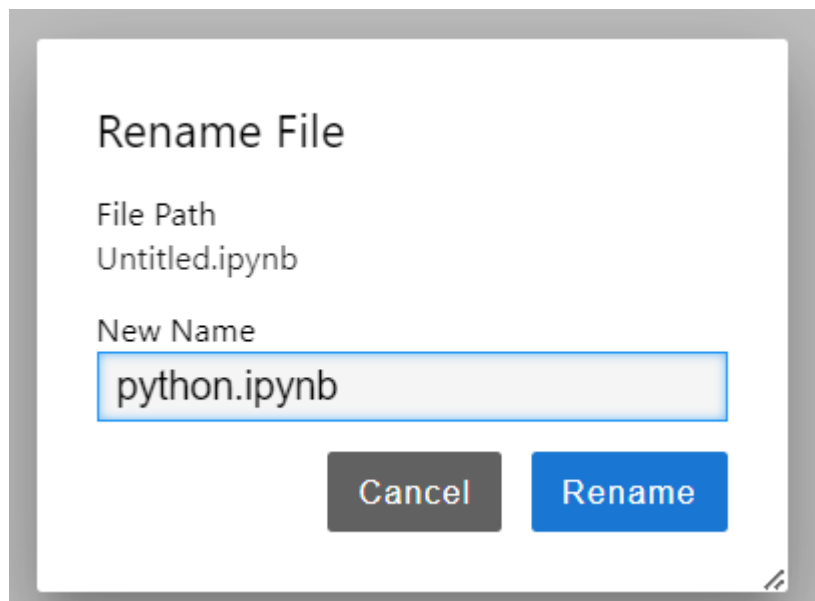
e) You will see a name: Untitled.ipynb files are newly created



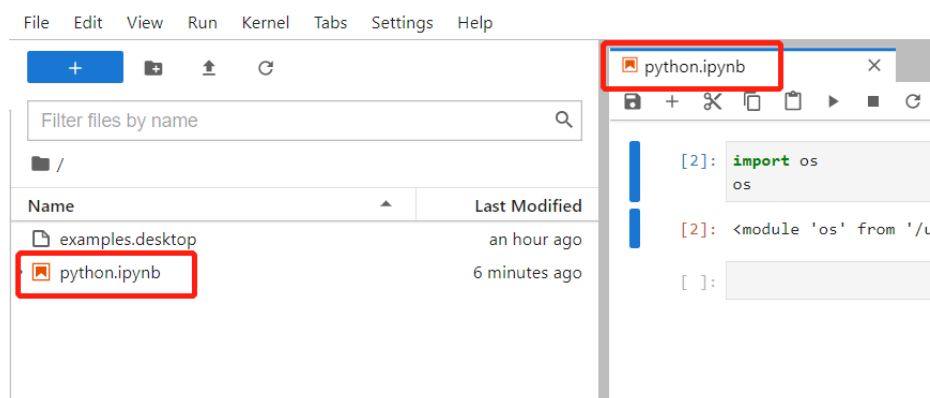
f) Enter the following code in the first gray box of the file and press Shift+Enter to execute it. If you get similar output, it proves that the Jupyter Notebook environment is running normally:



g) Click the File – Rename Notebook.. button in the upper left corner of the page to modify the current Notebook file name for the next part "Python Basic" exercises.



- h) Check the file list on the left to confirm that the file name has been successfully modified:



- i) For subsequent exercises, if no special instructions are given, you need to enter the code in the Notebook and execute it to obtain the output results.
- j) This document will only give the questions and results of the exercises and will not give specific code implementations. The reference code implementation will be given by the instructor at the end of the course.

2. Python Basic

LIST

- a) Create a list object: fruits, which includes the names of three fruits: apple , banana, orange
- b) Use print output all member's values of fruit list at once
- c) Print the second member value of fruits list
- d) Update the first member value of fruits list to: apple-new
- e) Print all member values of fruit list again to confirm that the update is successful.
- f) Two new members are added: pear and grape
- g) Print 2-4 member values

Output result :

```
['apple', 'banana', 'orange']  
banana  
['apple-new', 'banana', 'orange']  
['banana', 'orange', 'pear']
```

TUPLE

- a) Using the fruits from the previous exercise list to create an tuple object: fruits_tuple
- b) Print fruits_tuple all members' value
- c) Use packing style to create a tuple object fruits_tuple_new whose member value is the same as fruits_tuple
- d) Use == to compare two tuples to see if the result is True or False.
- e) Use unpacking to assign the values of fruits_tuple_new to the five variables a, b, c, d, and e respectively
- f) Then use a for loop to print the values of these five variables respectively.



- g) Use range to create an object my_range containing the integer number from 1 to 100
- h) Print the length of my_range object
- i) Try changing the value of the last element of my_range to hundred and see the error output.

Output result :

```
('apple-new', 'banana', 'orange', 'pear', 'grape')
True
apple-new
banana
orange
pear
grape
100
100
```

```
-----
TypeError                                Traceback (most recent call last)
Cell In[26], line 11
      9 print(len(my_range))
     10 print(my_range[-1])
--> 11 my_range[-1]="hundred"
```

```
TypeError: 'range' object does not support item assignment
```

ITERATE SEQUENCE DATA

- a) Use the for loop and enumerate function to iterate the previously created fruits_tuple_new object, obtain the index position information and corresponding values, and try to print each element sequentially in a format like the following statement:
 " Here is number 1 object with value apple-new"
- b) Create a new list object index_list, in which the value of the member is the position number of the fruits_tuple_new index, starting from 1 and the length is consistent with fruits_tuple_new.
- c) Use the for loop and zip function to iterate the two objects index_list and fruits_tuple_new at the same time and print their values in pairs.
- d) Use the range function to create an object the same as index_list, named index_range, to achieve the same paired printing as in the previous step.

Output result:

```
Here is number 0 object with value apple-new
Here is number 1 object with value banana
Here is number 2 object with value orange
Here is number 3 object with value pear
Here is number 4 object with value grape
(1, 'apple-new')
(2, 'banana')
(3, 'orange')
(4, 'pear')
(5, 'grape')
(1, 'apple-new')
(2, 'banana')
(3, 'orange')
(4, 'pear')
(5, 'grape')
```

DICTIONARY

- a) Create a new dict object, fruit_dict, based on the value of the fruits_tuple_new object, where the key is the value of each member of fruits_tuple_new and the value is its position number value in fruits_tuple_new, such as **apple-new: 1**.

You can try to use the zip function to achieve this.

- b) Use key to print the value with key **orange** in fruit_dict.
- c) Print fruit_dict all keys information.
- d) Use for loop to iterate fruit_dict members' key and value, print sequentially

Output result:

```
3
dict_keys(['apple-new', 'banana', 'orange', 'pear', 'grape'])
apple-new----1
banana----2
orange----3
pear----4
grape----5
```




3. Numpy Basic operations

PREPARATION

- a) Click on JupyterHub File – New – Notebook in the upper left corner of the page button to create a new Notebook Used to save the content of this exercise
- b) Select Kernel for Python 3
- c) As in the previous exercise, modify the notebook File name: numpy.ipynb
- d) Enter ***import numpy as np*** in the first code box and try to execute it to confirm that there are no errors, indicating that the numpy library has been loaded successfully. In subsequent exercises, ***np*** will be used instead of ***numpy*** as an abbreviation.

CREATE NUMPY ARRAY

- a) Create a list object containing 3 member of integer type, named: list_integer
- b) Create a tuple object, which contains 3 floating-point type members with a decimal point of 1 digit, named tuple_float
- c) Use the np.array method to construct two new ndarray objects based on the two newly created objects, named: array_list and array_tuple
- d) During the creation process, try to use the TAB completion function to achieve automatic completion of methods and variables.
- e) Then, print the basic results of the created objects separately, for example:

```
array_list,array_tuple  
  
(array([1, 2, 3]), array([1.1, 2.2, 3.3]))
```

- f) View the basic information of the newly created ndarray objects through the type function, shape and dtype attributes respectively:

```
(numpy.ndarray, numpy.ndarray, dtype('int32'), dtype('float64'), (3,), (3,))
```

- g) Use the `np.arange` method to create an ndarray object, which is all even numbers from 1 to 20 :

```
array([ 2,  4,  6,  8, 10, 12, 14, 16, 18, 20])
```

- h) Use `list_integer` and `tuple_float` to create a 2*3 array and use the **shape** attribute to confirm the shape of the result.:

```
(array([[1. , 2. , 3. ],
        [1.1, 2.2, 3.3]]),
 (2, 3))
```

- i) View the array just created through the **type** attribute. Please confirm the data type after combining integer data and floating point data.

```
(array([1, 2, 3]), array([1.1, 2.2, 3.3]), dtype('int32'), dtype('float64'))
```

```
(array([[1. , 2. , 3. ],
        [1.1, 2.2, 3.3]]),
 dtype('float64'))
```

- j) Recreate `array_list` object and set its data type to float 64 to see the changes in the results:

```
(array([1., 2., 3.]), dtype('float64'))
```

- k) Use the `np.ones` and multiplication function to create an ndarray with a shape of 4 * 4 and an element value of 10 :

```
array([[10., 10., 10., 10.],
       [10., 10., 10., 10.],
       [10., 10., 10., 10.],
       [10., 10., 10., 10.]])
```



BASIC OPERATIONS

- a) Create a one-dimensional array a with member values ranging from 1 to 10 integers
- b) Create a one-dimensional array b with member values ranging from -10 to -1 integers
- c) create array c, its value is a+b and check the value content
- d) create array d, whose value is a * b and check the value content

```
array([-9, -7, -5, -3, -1, 1, 3, 5, 7, 9])
```

```
array([-10, -18, -24, -28, -30, -30, -28, -24, -18, -10])
```

- e) Use sum function to calculate the mean of array a
5.5
- f) Use the comparison way to find the elements of the b array that are greater than -5 and count the numbers.

```
(array([False, False, False, False, False, False, True, True, True,
        True]),
4)
```

INDEXING AND DATA ACCESS

- a) Using np.arange to create a new array a with member values ranging from 1 to 10 integers
- b) Access the 3rd, 5th-10th, and penultimate data respectively

```
(3, [5, 6, 7, 8, 9, 10], 9)
```

- c) Multiply values in the even-numbered index of array a by 2 and print the result

```
array([ 4, 8, 12, 16, 20])
```

- d) Using `np.arange` to create a new array `b` with member values ranging from 1 to 24 .
- e) Use the `reshape` function to convert it into 3 rows and 8 columns array.

```
array([[ 1,  2,  3,  4,  5,  6,  7,  8],
       [ 9, 10, 11, 12, 13, 14, 15, 16],
       [17, 18, 19, 20, 21, 22, 23, 24]])
```

- f) Use the `reshape` function to convert it into 4 rows and 6 columns, and then assign it to array `c`.
- g) Get the third row data of array `c`.

```
array([13, 14, 15, 16, 17, 18])
```

- h) Get the third column data of array `c`.

```
array([ 3,  9, 15, 21])
```

- i) Create a new array: **names**, fill in 4 member variables: 'Bob', 'Will', 'Joe', 'Mary '
- j) Assume that each row of data in `c` corresponds to each of the 4 members in `names` (for example, `c[0]` is Bob's data, and so on), then please take out the data of Bob and Joe based on the name as a condition at one time:

```
array([[ 1,  2,  3,  4,  5,  6],
       [13, 14, 15, 16, 17, 18]])
```

CHANGE ARRAY SHAPE

- a) Use the ***flat*** attribute and ***ravel*** function of the array to "flatten" the members in the array `c` from the previous exercise respectively and then iterate and print to see the flattened results and output order:



```
array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24])
```

```
1
2
3
4
5
6
7
8
9
10
11
12
13
14
```

- b) Change array c's shape to 6*4 and assign it to the new array d.
- c) Use the T attribute and the transpose function to create the transposition of the d array respectively and compare whether the two results are consistent (same shape, same elements).
- d) Use vstack and hstack to stack c and d vertically and horizontally respectively (note: c and d have different shapes):

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 7,  8,  9, 10, 11, 12],
       [13, 14, 15, 16, 17, 18],
       [19, 20, 21, 22, 23, 24],
       [ 1,  5,  9, 13, 17, 21],
       [ 2,  6, 10, 14, 18, 22],
       [ 3,  7, 11, 15, 19, 23],
       [ 4,  8, 12, 16, 20, 24]])
```

```
array([[ 1,  2,  3,  4,  5,  6,  1,  5,  9, 13, 17, 21],
       [ 7,  8,  9, 10, 11, 12,  2,  6, 10, 14, 18, 22],
       [13, 14, 15, 16, 17, 18,  3,  7, 11, 15, 19, 23],
       [19, 20, 21, 22, 23, 24,  4,  8, 12, 16, 20, 24]])
```

- a) Using np.sort function sorts the stacked array in previous step and will get the following results:

```
array([[ 1,  2,  3,  4,  5,  6],
       [ 1,  5,  9, 10, 11, 12],
       [ 2,  6,  9, 13, 17, 18],
       [ 3,  7, 10, 14, 17, 21],
       [ 4,  8, 11, 15, 18, 22],
       [ 7,  8, 12, 16, 19, 23],
       [13, 14, 15, 16, 20, 24],
       [19, 20, 21, 22, 23, 24]])
```

```
array([[ 1,  1,  2,  3,  4,  5,  5,  6,  9, 13, 17, 21],
       [ 2,  6,  7,  8,  9, 10, 10, 11, 12, 14, 18, 22],
       [ 3,  7, 11, 13, 14, 15, 15, 16, 17, 18, 19, 23],
       [ 4,  8, 12, 16, 19, 20, 20, 21, 22, 23, 24, 24]])
```

UFUNC OPERATIONS

- a) Use the np.random.randn function to create two arrays x and y respectively, with the number of members being 10. Because they are random values, the output of the following examples will be different from your running results, for reference only.
- b) Use the np.maximum method to find the maximum value of the element at the same index position in x, y, similar to the output:

```
array([ 0.33851654,  0.52283582, -0.60165383, -0.12435201,  0.80477296,
        0.12523513,  0.43381472,  0.55926481,  1.47143048, -0.85568811])
```

- b) Then use the np.abs method to take the absolute value of the value in x, y, and then use the np.maximum method again to find the maximum value of the element at the same index position in x, y.

```
array([2.13188235, 1.22151757, 0.13738144, 0.74742505, 0.62555621,
        1.26229279, 1.32221022, 1.95550421, 1.18047047, 0.23983974])
```



- c) Use `np.equal` to determine the difference between the two maximum values and calculate the different amounts:

```
(array([False, False, False, False, False, True, True, True, False,
        True])),
4)
```

ARRAY ELEMENT OPERATIONS

- a) Using `np.random.randn` function creates a 3*4 array x and use `sum` and `mean` function to calculate:
- Sum and average of this array as a whole
 - Sum and average of each row
 - sum and mean of each column

Whole array

```
(4.8521789180629185, 0.40434824317190987)
```

Each Row

```
(array([ 2.76198296,  1.3405538 ,  2.9629198 , -2.21327764]),
 array([ 0.92066099,  0.44685127,  0.98763993, -0.73775921]))
```

Each Column

```
(array([0.21943539, 0.97683881, 3.65590472]),
 array([0.05485885, 0.2442097 , 0.91397618]))
```

- b) Using `np.ones` and `reshape` to create a 2*2 array a with all member values 1.
- c) Use `np.insert` change array a shape to 2*3 keeping all member values 1 and assign this new array to b.
- d) Use `np.insert` again change array b to 3*3 keeping the all member values as 1 and assign this new array to c.

```
a,b,c
```

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

- e) Using np.append change array c shape to 3 * 4 setting the value of the new column member to 2 and assign it to array d

```
array([[1., 1., 1., 2.],  
       [1., 1., 1., 2.],  
       [1., 1., 1., 2.]])
```

- f) Using np.append again change array d shape to 6*4 by adding 3 new lines to it and assigned to array e.

```
array([[1., 1., 1., 2.],  
       [1., 1., 1., 2.],  
       [1., 1., 1., 2.],  
       [1., 1., 1., 2.],  
       [1., 1., 1., 2.],  
       [1., 1., 1., 2.]])
```

- g) Utilize np.unique function prints the unique value information of the entire array e, rows, and columns respectively.



```
(array([1., 2.]),
 array([[1., 2.],
        [1., 2.],
        [1., 2.],
        [1., 2.],
        [1., 2.],
        [1., 2.]]),
 array([[1., 1., 1., 2.]])
```

4. Pandas operations

PREPARATION

- Click on JupyterHub File – New – Notebook in the upper left corner of the page button to create a new Notebook Used to save the content of this part of the exercise
- Select Kernel for Python 3
- As in the previous exercise, modify the notebook File name: pandas.ipynb
- Enter the import numpy and pandas statements in the first cell for subsequent exercises, and give them new names np and pd respectively.

CREATE DATA STRUCTURE

- Create a List object: list_s_1, fill the objects from a to e, 5 lowercase English letters.
- Using pd.Series create a Series named s_1 based on list_s_1 as data source.

```
s_1
```

```
0    a
1    b
2    c
3    d
4    e
```

```
dtype: object
```

- c) Then use `np.arange` to provide data for `pd.Series` and create an `s_2` object with an integer content of 1 to 5

```
s_2
```

```
0    1
1    2
2    3
3    4
4    5
```

```
dtype: int32
```

- d) Create a List object: `list_index_1`, fill the object with 5 uppercase English letters from A to E. Then use it as the index data of `pd.Series` and `np.random` as the data to create a Series with 5 random numbers: `s_3`



```
s_3
```

```
A    0.689584
B    0.112340
C    0.654600
D    0.120214
E    0.018220
dtype: float64
```

- e) Try using Series index property assignment to modify its Index information, to unify the index information of s_1 and s_2 into list_index_1 , keeping it consistent with s_3 :

```
s_1, s_2
```

```
(A    a
  B    b
  C    c
  D    d
  E    e
  dtype: object,
A    1
  B    2
  C    3
  D    4
  E    5
  dtype: int32)
```

- f) Try to form a List from s_1, s_2 and s_3 and provide it to pd.DataFrame as data to create a DataFrame and check the result:

	A	B	C	D	E
0	a	b	c	d	e
1	1	2	3	4	5
2	0.689584	0.11234	0.6546	0.120214	0.01822

- g) Create s_1, s_2 and s_3 respectively as column data of DataFrame. Try to create them in dict form and set the column names to s1, s2 and s3 respectively. Assign the newly created DataFrame to d_1.

d_1

	s1	s2	s3
A	a	1	0.689584
B	b	2	0.112340
C	c	3	0.654600
D	d	4	0.120214
E	e	5	0.018220


- h) Try to use the list_s_1 created earlier as index data, create DataFrame d_2 with the same data as in the previous step, and check the creation results.:



	s1	s2	s3
a	NaN	NaN	NaN
b	NaN	NaN	NaN
c	NaN	NaN	NaN
d	NaN	NaN	NaN
e	NaN	NaN	NaN

- i) As you can see, the output data are all NaN , try to adjust the list of index data to the index information of the input data, such as ['B', 'C '] , and check the output results.

READING AND WRITING DATA BASED ON FILES

- a) From Jupyter Hub in the upper left corner of the interface, click  the button and select the exams.csv file distributed to you to upload.
- b) After the upload is successful, you will see this new file appear in the directory.
- c) Back to Notebook after that, use read _csv function reads the file just uploaded and assigns the result data to new d_3 object.
- d) Use the **info** function to view the basic information of d_3 and confirm the column information, row number information, data type, etc.

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                1000 non-null   object
1   race/ethnicity                        1000 non-null   object
2   parental level of education          1000 non-null   object
3   lunch                                1000 non-null   object
4   test preparation course              1000 non-null   object
5   math score                           1000 non-null   int64
6   reading score                        1000 non-null   int64
7   writing score                         1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB

```

- e) You can check whether the number of lines and information read are the same as the original file by opening the file.
- f) Using the same function to read CSV file, this time add the parameter **nrows** to set the number of rows to be read, such as 100 , assign it to the new variable d_4 and check whether the result is same as the specified number of rows:

```
d_4.info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                100 non-null   object
1   race/ethnicity                        100 non-null   object
2   parental level of education          100 non-null   object
3   lunch                                100 non-null   object
4   test preparation course              100 non-null   object
5   math score                           100 non-null   int64
6   reading score                        100 non-null   int64
7   writing score                         100 non-null   int64
dtypes: int64(3), object(5)
memory usage: 6.4+ KB

```



- g) Based on d_4 , select three columns of data related to scores: math score, reading score, writing score and assign them to the new variable d_5

```
d_5.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 3 columns):
#   Column          Non-Null Count  Dtype
---  -
0   math score      100 non-null   int64
1   reading score   100 non-null   int64
2   writing score    100 non-null   int64
dtypes: int64(3)
memory usage: 2.5 KB
```

- h) Using to _csv function outputs d_5 to a new file, exams_scoring.csv and replaces the data delimiter with # :

```
!more exams_scoring.csv
```

```
#math score#reading score#writing score
0#67#67#63
1#40#59#55
2#59#60#50
3#77#78#68
4#78#73#68
5#63#77#76
6#62#59#63
7#93#88#84
8#63#56#65
9#17#17#15
```

ACCESS BASIC INFORMATION (PROPERTIES) OF THE DATA STRUCTURE

- a) Based on the result of the previous exercise d_1 object, view its property information. First, view its row and column Index information:

```
(  s1  s2      s3
A   a   1  0.689584
B   b   2  0.112340
C   c   3  0.654600
D   d   4  0.120214
E   e   5  0.018220,
Index(['A', 'B', 'C', 'D', 'E'], dtype='object'),
Index(['s1', 's2', 's3'], dtype='object'))
```

- b) Based on the first row of d_1 , view the index information of the data.

```
(s1      a
s2      1
s3      0.689584
Name: A, dtype: object,
Index(['s1', 's2', 's3'], dtype='object'))
```

- c) Based on column s1 of d_1 , view the index information of the data.

```
(A   a
B   b
C   c
D   d
E   e
Name: s1, dtype: object,
Index(['A', 'B', 'C', 'D', 'E'], dtype='object'))
```




- d) Use info Function to view the overall information of d_1 :

```
<class 'pandas.core.frame.DataFrame'>
Index: 5 entries, A to E
Data columns (total 3 columns):
#   Column  Non-Null Count  Dtype
---  -
0   s1      5 non-null      object
1   s2      5 non-null      int32
2   s3      5 non-null      float64
dtypes: float64(1), int32(1), object(1)
memory usage: 312.0+ bytes
```

ACCESS DATA DATAFRAME BASED ON COLUMNS

- a) Utilize d_4 data to implement column-based data access. First, realize the simultaneous display of information from the three columns " gender ", " race/ethnicity " and " math score ".

	gender	race/ethnicity	math score
0	male	group A	67
1	female	group D	40
2	male	group E	59
3	male	group B	77
4	male	group F	70

- b) Create a new column for d_4: avg_score, with the value being the average of the three score column values:

```
d_4["avg_score"].head()
```

```
0    65.666667
1    51.333333
2    56.333333
3    74.333333
4    73.000000
Name: avg_score, dtype: float64
```

- c) Use the sum method to count the number of avg_score values greater than 60, and the output value is 65
- d) Use a similar method to find the gender information of records with avg_score value greater than 60 :

```
0      male
3      male
4      male
5    female
6    female
...
92   female
93     male
94     male
95   female
96   female
Name: gender, Length: 65, dtype: object
```

- e) Use the drop function to delete the avg_score column, restore d_4 to its original state:

```
d_4.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                100 non-null    object
1   race/ethnicity                        100 non-null    object
2   parental level of education          100 non-null    object
3   lunch                                100 non-null    object
4   test preparation course              100 non-null    object
5   math score                           100 non-null    int64
6   reading score                        100 non-null    int64
7   writing score                         100 non-null    int64
8   avg_score                            100 non-null    float64
dtypes: float64(1), int64(3), object(5)
```



```
d_4.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 100 entries, 0 to 99
Data columns (total 8 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                100 non-null    object
1   race/ethnicity                        100 non-null    object
2   parental level of education           100 non-null    object
3   lunch                                 100 non-null    object
4   test preparation course               100 non-null    object
5   math score                           100 non-null    int64
6   reading score                        100 non-null    int64
7   writing score                         100 non-null    int64
dtypes: int64(3), object(5)
memory usage: 6.4+ KB
```

ROW-BASED ACCESS DATAFRAME

- a) Implement row-based data access using d_1 data. Use the loc attribute to access rows with index values in C and E:

	s1	s2	s3
C	c	3	0.65460
E	e	5	0.01822

- b) Similarly, access all rows starting from index B:

	s1	s2	s3
B	b	2	0.112340
C	c	3	0.654600
D	d	4	0.120214
E	e	5	0.018220

- c) Access all rows starting with index A and ending with D:

	s1	s2	s3
A	a	1	0.689584
B	b	2	0.112340
C	c	3	0.654600
D	d	4	0.120214

d) Access row index values is B of columns s 1 and s 2 :

```
s1      b
s2      2
Name: B, dtype: object
```

e) Access rows index value starting from B of columns s 1 and s 2 :

	s1	s2
B	b	2
C	c	3
D	d	4
E	e	5

f) From this step, using d_5 as data source with iloc function to access data. First access the first 50 rows of data:



```
(  math score  reading score  writing score
0      67      67      63
1      40      59      55
2      59      60      50
3      77      78      68
4      78      73      68,
    math score  reading score  writing score
45      60      46      51
46      64      63      57
47      65      76      78
48      68      72      73
49      60      63      57)
```

- g) Using `numpy.arange` function to access rows whose index positional value is odd :

	math score	reading score	writing score
1	40	59	55
3	77	78	68
5	63	77	76
7	93	88	84
9	47	42	45
11	80	87	90
13	74	74	73
15	69	61	57
17	54	62	65
19	39	32	31
21	58	52	55
23	44	55	54
25	51	45	42

- h) Access the first 2 columns of the first 10 rows of data:

	math score	reading score
0	67	67
1	40	59
2	59	60
3	77	78
4	78	73
5	63	77
6	62	59
7	93	88
8	63	56
9	47	42

- i) Combine loc and iloc to access the "writing score " column data of the first 10 rows of data.

```
0    63
1    55
2    50
3    68
4    68
5    76
6    63
7    84
8    65
9    45
```

Name: writing score, dtype: int64

OTHER WAYS TO ACCESS DATA

- a) This part is based on d_4 object, using loc and judgment conditions to implement data access. First, get all rows of gender is male:



	gender	race/ethnicity	parental level of education	lunch	test preparation course
0	male	group A	high school	standard	completed
2	male	group E	some college	free/reduced	none
3	male	group B	high school	standard	none
4	male	group E	associate's degree	standard	completed
7	male	group E	some college	standard	completed
8	male	group D	high school	standard	none
9	male	group C	some college	free/reduced	none
10	male	group E	some college	standard	completed
12	male	group D	associate's degree	standard	completed
13	male	group C	high school	standard	completed
14	male	group E	some high school	standard	completed
15	male	group E	associate's degree	free/reduced	none
16	male	group B	high school	standard	none

b) Get all rows of gender for female and also in group E:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score
47	female	group E	some high school	standard	completed	65
61	female	group E	some high school	standard	none	58
96	female	group E	associate's degree	standard	completed	95

c) Get reading score greater than 80 or writing score greater than 80:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
7	male	group E	some college	standard	completed	93	88	84
10	male	group E	some college	standard	completed	99	83	85
11	female	group D	high school	standard	completed	80	87	90
12	male	group D	associate's degree	standard	completed	77	87	85
14	male	group E	some high school	standard	completed	81	87	85
20	female	group C	associate's degree	standard	none	83	76	88
28	female	group B	high school	standard	completed	74	89	89
34	female	group B	bachelor's degree	standard	completed	79	88	89
39	male	group E	associate's degree	standard	none	98	81	77
42	male	group E	high school	standard	completed	87	79	82
44	male	group F	high school	standard	none	100	88	87

d) Get all rows of data not in Group E:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	ma
0	male	group A	high school	standard	completed	
1	female	group D	some high school	free/reduced	none	
3	male	group B	high school	standard	none	
5	female	group D	high school	standard	none	
6	female	group A	bachelor's degree	standard	none	
...
94	male	group B	master's degree	free/reduced	none	

e) Get all rows of data not in Group C And gender is female:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing sc
1	female	group D	some high school	free/reduced	none	40	59	
5	female	group D	high school	standard	none	63	77	
6	female	group A	bachelor's degree	standard	none	62	59	
11	female	group D	high school	standard	completed	80	87	
23	female	group B	some college	free/reduced	none	44	55	
28	female	group B	high school	standard	completed	74	89	
29	female	group A	some high school	free/reduced	none	33	54	
33	female	group B	associate's degree	standard	completed	65	65	
34	female	group B	bachelor's degree	standard	completed	79	88	

f) Get all rows with math score in: 40 , 60 , 80 :

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
1	female	group D	some high school	free/reduced	none	40	59	55
11	female	group D	high school	standard	completed	80	87	90
32	male	group A	some high school	free/reduced	completed	40	41	39
45	male	group C	bachelor's degree	standard	none	60	46	51
49	male	group C	associate's degree	free/reduced	completed	60	63	57
69	female	group C	high school	standard	none	60	64	57
99	male	group B	associate's degree	standard	none	60	54	53

g) Use query function to redo a to f tasks.

h) Based on f The output of the step task is iterated using the `iteruples` function, and then the value of math score is stored in `math_score_list` accordingly:

```
math_score_list
```

```
[40, 80, 40, 60, 60, 60, 60]
```




GET STATISTICS ABOUT DATA

SIMPLE DATA PROCESSING FUNCTION

- First, copy d_4 to a new data d_7. Based on the math score column of d_4, set the condition that the value less than 40 is NaN (using where), and assign the result to the math score column of d_7
- Use the isnull and sum methods to count how many null values there are in the math score column of d_7. The result should be: 8
- On the contrary, use the notnull and sum methods to count how many non-null values there are in the math score column of d_7. The result should be: 92
- Get a complete record of all data with null values in the math score column:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
1	female	group D	some high school	free/reduced	none	NaN	59	55
18	female	group C	associate's degree	free/reduced	none	NaN	44	44
19	male	group C	some college	free/reduced	none	NaN	32	31
29	female	group A	some high school	free/reduced	none	NaN	54	51
32	male	group A	some high school	free/reduced	completed	NaN	41	39
53	female	group C	high school	free/reduced	none	NaN	53	49
81	female	group D	some high school	free/reduced	none	NaN	38	42
84	female	group B	associate's degree	free/reduced	none	NaN	41	39

- Try using fillna to fill the data with an empty math score column to the average of the math score (sum of non-null values/number of non-null values):

```
d_7_math_avg, (d_7["math score"].isnull()).sum()
```

```
(67.3695652173913, 0)
```

- Create a new column gender_number for d_7, whose value comes from the gender column, and then use the replace method to convert the character information in gender_number into digits: male is 0, female is 1 :

d_7									
	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	gender_number
0	male	group A	high school	standard	completed	67.000000	67	63	0
1	female	group D	some high school	free/reduced	none	67.369565	59	55	1
2	male	group E	some college	free/reduced	none	59.000000	60	50	0
3	male	group B	high school	standard	none	77.000000	78	68	0
4	male	group E	associate's degree	standard	completed	78.000000	73	68	0
...
95	female	group C	high school	standard	completed	58.000000	63	67	1

IMPLEMENT DATA GROUPING, AGGREGATION AND TRANSFORMATION

- a) This part of the exercise is based on d_3 data . First of all, before processing grouped data, you need to know which columns of data are suitable for grouping. You can use info function obtains the attributes of the data and makes the initial judgment.

```
d_3.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 8 columns):
#   Column                                     Non-Null Count  Dtype
---  -
0   gender                                     1000 non-null   object
1   race/ethnicity                             1000 non-null   object
2   parental level of education                1000 non-null   object
3   lunch                                       1000 non-null   object
4   test preparation course                    1000 non-null   object
5   math score                                 1000 non-null   int64
6   reading score                             1000 non-null   int64
7   writing score                              1000 non-null   int64
dtypes: int64(3), object(5)
memory usage: 62.6+ KB
```

- b) Suppose, first select gender as grouped data, you can pass np.unique function to quickly obtain the unique data information of this column

```
array(['female', 'male'], dtype=object)
```

- c) And if you check unique data for data that is not suitable for grouping, you will get a lot of results, so grouping is meaningless.



```
array([ 13, 23, 25, 26, 28, 29, 30, 31, 32, 33, 34, 35, 36,
       37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49,
       50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62,
       63, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75,
       76, 77, 78, 79, 80, 81, 82, 83, 84, 85, 86, 87, 88,
       89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100],
      dtype=int64)
```

- d) Next, you may need to know the basic information after the data is grouped, such as the grouping situation, that is, which data is divided into which group. You can view it based on the groups attribute of the groupby result.

```
{'female': [1, 5, 6, 11, 17, 18, 20, 23, 26, 28, 29, 33, 34, 35, 36, 38, 40, 43, 47, 48, 51, 52, 53, 55, 56, 58, 61, 65, 67,
9, 70, 72, 73, 75, 76, 77, 80, 81, 82, 84, 85, 86, 88, 92, 95, 96, 104, 105, 110, 111, 112, 116, 119, 120, 121, 122, 123, 1
129, 130, 137, 139, 141, 142, 143, 145, 146, 148, 150, 151, 152, 154, 155, 157, 158, 160, 162, 163, 165, 167, 168, 169, 170
1, 172, 175, 176, 178, 180, 181, 182, 183, 184, 185, 191, 193, 194, 195, 198, 200, ...], 'male': [0, 2, 3, 4, 7, 8, 9, 10,
13, 14, 15, 16, 19, 21, 22, 24, 25, 27, 30, 31, 32, 37, 39, 41, 42, 44, 45, 46, 49, 50, 54, 57, 59, 60, 62, 63, 64, 66, 68,
74, 78, 79, 83, 87, 89, 90, 91, 93, 94, 97, 98, 99, 100, 101, 102, 103, 106, 107, 108, 109, 113, 114, 115, 117, 118, 125, 1
127, 128, 131, 132, 133, 134, 135, 136, 138, 140, 144, 147, 149, 153, 156, 159, 161, 164, 166, 173, 174, 177, 179, 186, 187
8, 189, 190, 192, 196, 197, ...]}
```

- e) The results show two sets of data based on gender and the value of the data is the Index value. At the same time, you can obtain the grouped result data of the specified group based on get_group, such as the data of the male group:

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	male	group A	high school	standard	completed	67	67	63
2	male	group E	some college	free/reduced	none	59	60	50
3	male	group B	high school	standard	none	77	78	68
4	male	group E	associate's degree	standard	completed	78	73	68
7	male	group E	some college	standard	completed	93	88	84
...
992	male	group C	some college	standard	none	69	63	66
...

- f) The grouped data can be subjected to corresponding aggregation operations. The simplest method is to use the count method to count the quantity.

	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
gender							
female	483	483	483	483	483	483	483
male	517	517	517	517	517	517	517

- g) It can be found that the aggregated result will use gender as an index. If you do not want this to happen, you can add as_index=False during groupby to achieve this.

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	female		483	483	483	483	483	483
1	male		517	517	517	517	517	517

- h) At the same time, you can see that the value of each column in the output is actually the same. If you only want to see a certain column, such as lunch, you can use [] to specify the specific column name information you want to see:

	gender	lunch
0	female	483
1	male	517

- i) General aggregation operations are applicable to numeric data, such as sums, averages, and other operations. For example, you want to get the average information of three test scores based on gender grouping:

	gender	math score	reading score	writing score
0	female	63.196687	71.888199	71.708075
1	male	69.384913	66.305609	64.029014

- j) Of course, applying the **describe** function to grouped data can obtain information on multiple statistical analysis values at once:

	math score								reading score					writing score							
	count	mean	std	min	25%	50%	75%	max	count	mean	...	75%	max	count	mean	std	min	25%	50%	75%	max
0	483.0	63.196687	15.490079	13.0	53.0	63.0	75.0	100.0	483.0	71.888199	...	83.0	100.0	483.0	71.708075	15.238072	23.0	62.0	72.0	83.0	100.0
1	517.0	69.384913	14.719365	29.0	59.0	70.0	79.0	100.0	517.0	66.305609	...	77.0	100.0	517.0	64.029014	15.025983	23.0	54.0	64.0	74.0	100.0

- k) Use agg Functions can implement more general aggregation tasks. For example, multiple aggregation functions that need to be used can be provided at the same time, such as np.sum , np.mean , to complete multiple aggregation operations at one time.



	math score		reading score		writing score	
	sum	mean	sum	mean	sum	mean
gender						
female	30524	63.196687	34722	71.888199	34635	71.708075
male	35872	69.384913	34280	66.305609	33103	64.029014

- l) At the same time, you can use the agg function to provide data in dict form and specify different columns to apply different aggregation operations.

	gender	math score	reading score
0	female	63.196687	14.660470
1	male	69.384913	14.305073

- m) Different from the aggregation operation, the transform operation performs a separate operation on each grouped data. A simple understanding is that the shape of the grouped data will not change. Use apply you can implement conversion operations with a simple lambda function . For example, if you want to know the difference between the scores of three exams and the average (lambda x: x- x.mean ()):

```
>>> .groupby(..., group_keys=True)
d_3.groupby(["gender"])[["math score", "re
```

	math score	reading score	writing score
0	-2.384913	0.694391	-1.029014
1	-23.196687	-12.888199	-16.708075
2	-10.384913	-6.305609	-14.029014
3	7.615087	11.694391	3.970986
4	8.615087	6.694391	3.970986
...
995	3.615087	3.694391	0.970986
996	15.615087	24.694391	27.970986
997	-31.196687	-36.888199	-30.708075
998	9.803313	2.111801	10.291925
999	-4.384913	-6.305609	-2.029014

- n) When executing apply based on groupby, some warning messages will pop up, indicating that the group_keys parameter needs to be set for groupby. The default value is False, then the grouping key will not appear in the output. Try setting it to True to display the grouping key:



		math score	reading score	writing score
gender				
female	1	-23.196687	-12.888199	-16.708075
	5	-0.196687	5.111801	4.291925
	6	-1.196687	-12.888199	-8.708075
	11	16.803313	15.111801	18.291925
	17	-9.196687	-9.888199	-6.708075
...
male	992	-0.384913	-3.305609	1.970986
	994	16.615087	15.694391	10.970986
	995	3.615087	3.694391	0.970986
	996	15.615087	24.694391	27.970986
	999	-4.384913	-6.305609	-2.029014

HANDLE CATEGORICAL DATA

- a) This part will be based on d_3 data. Create a new column named group_cate, and the data comes from the race/ethnicity column.

```
d_3.info(), d_3["group_cate"].head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                1000 non-null   object
1   race/ethnicity                        1000 non-null   object
2   parental level of education           1000 non-null   object
3   lunch                                 1000 non-null   object
4   test preparation course               1000 non-null   object
5   math score                            1000 non-null   int64
6   reading score                         1000 non-null   int64
7   writing score                         1000 non-null   int64
8   group_cate                           1000 non-null   object
dtypes: int64(3), object(6)
memory usage: 70.4+ KB

(None,
0    group A
1    group D
2    group E
3    group B
4    group E
Name: group_cate, dtype: object)
```

- b) Based on new group_cate , first use astype function changes its type to "category"

```
d_3.info(),d_3["group_cate"].head()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 9 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   gender                                1000 non-null   object
1   race/ethnicity                        1000 non-null   object
2   parental level of education           1000 non-null   object
3   lunch                                 1000 non-null   object
4   test preparation course               1000 non-null   object
5   math score                            1000 non-null   int64
6   reading score                         1000 non-null   int64
7   writing score                         1000 non-null   int64
8   group_cate                           1000 non-null   category
dtypes: category(1), int64(3), object(5)
memory usage: 63.8+ KB

(None,
0    group A
1    group D
2    group E
3    group B
4    group E
Name: group_cate, dtype: category
Categories (5, object): ['group A', 'group B', 'group C', 'group D', 'group E'])
```




- c) The categorical information is automatically recognized and created. At this time, you can try to execute the describe method on the new column to compare the output with the numerical data:

```
count      1000
unique         5
top      group C
freq      323
Name: group_cate, dtype: object
```

- d) Categorical data can generally be understood as a column in a DataFrame, which will automatically obtain two attribute objects: `cat.categories` and `cat.ordered` (`cat.ordered` will be omitted later). Among them, the categories information is the category group information, that is, how many categories this category data has. You can try to use a for loop to iterate it and sequentially print the categories information of the `group_cate` just created.:

```
for group in d_3["group_cate"].cat.categories:
    print(group)
```

```
group A
group B
group C
group D
group E
```

- e) If order of categorical is important, the definition order of categories and ordered can be used to express the order relationship of categories.

For example, the previous group A-E is a primary to advanced categories, then it is understandable that students in Group E will perform better than students in Group A. By default, `ordered` is False and cannot be modified. The data when `ordered` is True can be obtained through the `cat.as_ordered` method:

```

0      group A
1      group D
2      group E
3      group B
4      group E
...
995    group C
996    group D
997    group C
998    group C
999    group A
Name: group_cate, Length: 1000, dtype: category
Categories (5, object): ['group A' < 'group B' < 'group C' < 'group D' < 'group E']

```

- f) On the contrary, if it is sorted categorical data, use `cat.as_unordered` function can get Data when ordered is False.
- g) The main uses of sequential categorical data are: sorting and comparison. First, assign the `group_cate` column data with order to `s_group_cate` and then use the `sort_values` method to sort it to see if the output is arranged in the preset order.:

```

0      group A
369    group A
376    group A
388    group A
399    group A
...
752    group E
777    group E
783    group E
273    group E
536    group E
Name: group_cate, Length: 1000, dtype: category
Categories (5, object): ['group A' < 'group B' < 'group C' < 'group D' < 'group E']

```

- h) At this time, use the `cat.rename_categories` method to adjust the category name to: 'group B' < 'group C' < 'group A' < 'group E' < 'group D', and then check whether the result after `sort_values` follows changes due to sequence adjustment:

```

s_group_cate.sort_values()

0      group B
369    group B
376    group B
388    group B
399    group B
...
752    group D
777    group D
783    group D
273    group D
536    group D
Name: group_cate, Length: 1000, dtype: category
Categories (5, object): ['group B' < 'group C' < 'group A' < 'group E' < 'group D']

```



- i) It can be seen that `rename_categories` only changes the original category group name, but the actual sorting does not change (can be determined according to the index serial number). Experience the sorting and look at the comparison results. Currently `group_cate` is still in the order of Group A to D, then the elements belonging to Group D should be about the same as the elements of Group A:

```
(0      group A
1      group D
2      group E
3      group B
4      group E
...
995    group C
996    group D
997    group C
998    group C
999    group A
Name: group_cate, Length: 1000, dtype: category
Categories (5, object): ['group A' < 'group B' < 'group C' < 'group D' < 'group E'],
True,
True)
```

- j) As data increases or decreases, new category types may appear, such as Group F, etc.; or disappearing classification types need to be deleted. Category types can be added and deleted respectively through the `add_categories` and `remove_categories` methods (without directly modifying the original data). Note: The new category key cannot be the same as the existing key.

```
0      NaN
1      group D
2      group E
3      group B
4      group E
...
995    group C
996    group D
997    group C
998    group C
999      NaN
Name: group_cate, Length: 1000, dtype: category
Categories (4, object): ['group B' < 'group C' < 'group D' < 'group E']
```

```

0      group A
1      group D
2      group E
3      group B
4      group E
...
995    group C
996    group D
997    group C
998    group C
999    group A
Name: group_cate, Length: 1000, dtype: category
Categories (6, object): ['group A' < 'group B' < 'group C' < 'group D' < 'group E' < 'group F']

```

- k) Group statistics based on categorical data are more convenient, and the method of use has not changed. Based on the `value_counts` method, the number of grouped data can be quickly counted:

```
s_group_cate.value_counts()
```

```

group C    323
group D    262
group B    205
group E    131
group A     79
Name: group_cate, dtype: int64

```

PROCESSING TIME SERIES DATA

- a) First, use the `pd.date_range` method to create a time series data, `s_date_1`, with a start date of 2023-1-1, a frequency (freq) of day (D), and a time span (period) of 100. Check out its basic information below and confirm that it is separated by actual month lengths.



```
s_date_1.info(),s_date_1[29:33],s_date_1[57:61]
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 100 entries, 0 to 99
Series name: None
Non-Null Count  Dtype
-----
100 non-null    datetime64[ns]
dtypes: datetime64[ns](1)
memory usage: 928.0 bytes
```

```
(None,
 29    2023-01-30
 30    2023-01-31
 31    2023-02-01
 32    2023-02-02
dtype: datetime64[ns],
 57    2023-02-27
 58    2023-02-28
 59    2023-03-01
 60    2023-03-02
dtype: datetime64[ns])
```

- b) Using `pd.Period` function creates a time span (Period) object `s_date_2` with a start date of 2023-1-1 and a frequency (freq) of day (D) . Try to implement an addition operation on it to increase the date:

```
s_date_2 + 1
```

```
Period('2023-01-02', 'D')
```

- c) Using this operation, you can easily calculate the difference between two time spans, such as creating a new Period with a start date of 2023-7-1 `s_date_3` , try to calculate the difference between it and 2023-1-1 : 181 days
- d) Timestamp or time span objects have many practical attributes, such as `dayofweek` which indicates the day of the week (starting from 0). So try to calculate how many weekends there are between `s_date_2` and `s_date_3` ?

5. Matplotlib Operations

PREPARATION

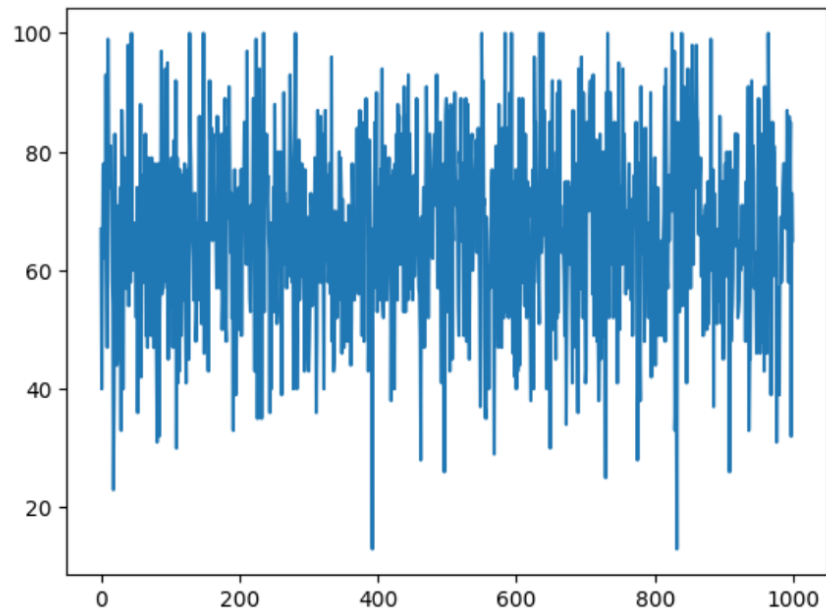
- a) Click on JupyterHub File – New – Notebook in the upper left corner of the page button to create a new Notebook Used to save the content of this part of the exercise
- b) Select Kernel for Python 3
- c) As in the previous exercise, modify the notebook File name: matplotlib.ipynb
- d) Enter import in the first cell numpy and pandas The statements are used in subsequent exercises and are given new names np. and pd.
- e) At the same time, type import matplotlib.pyplot as plt and %matplotlib inline to reference the libraries required by matplotlib

DRAW A LINE CHART

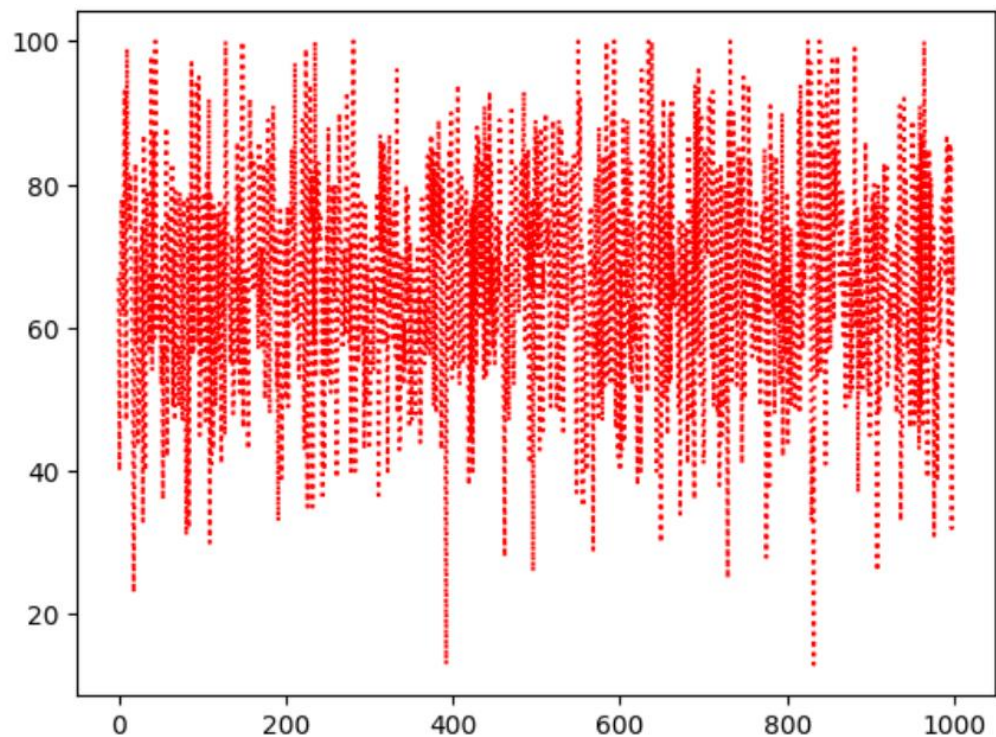
- a) The following drawing exercises will continue to be based on the data in the exams.csv file. Please create a data variable exam_data first to save the read data for subsequent use.
- b) Different types of graphics are suitable for describing different types of data. The drawing exercises in these parts are mainly to master the drawing functions, and the applicability of the data may not be particularly concerned. First, start with the simplest way and plot the data in the math score column separately (Y-axis). The X-axis is the index number of the row.



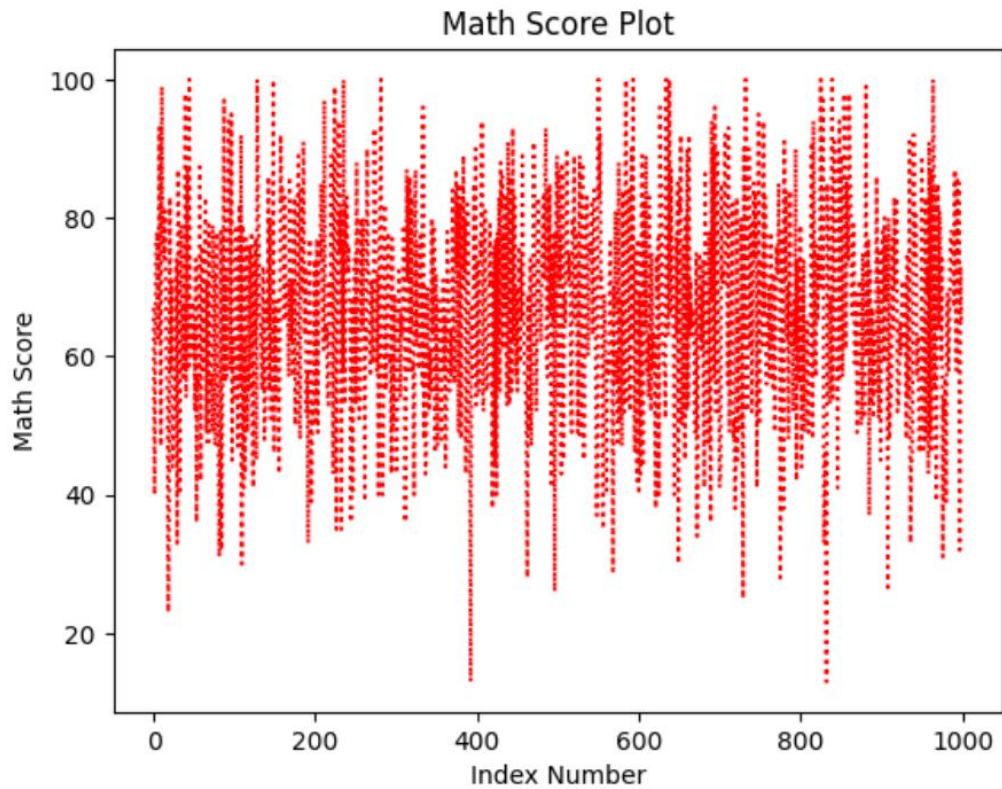
```
fig, ax = plt.subplots()
ax.plot(exam_data["math score"])
plt.show()
```



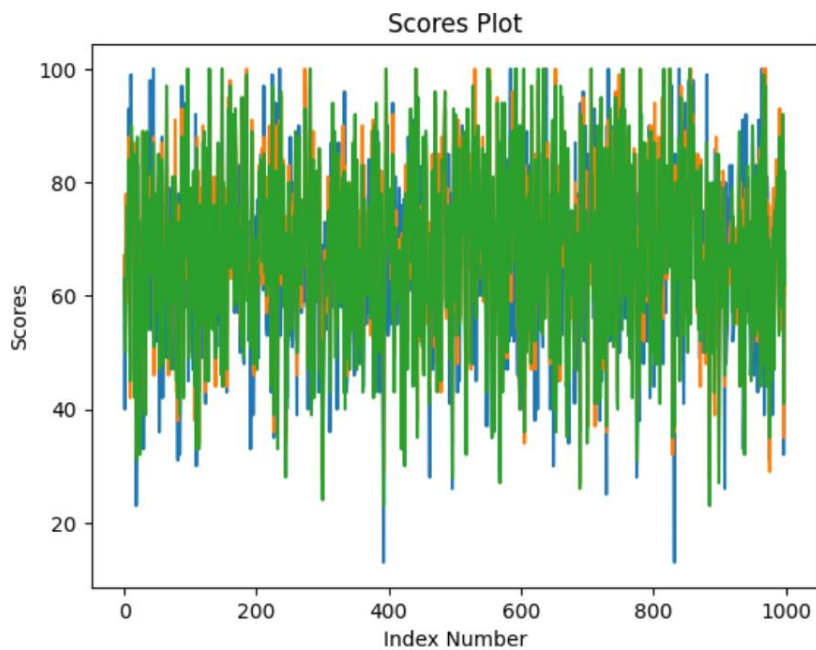
- c) Then, adjust the line color (red) and add the color attribute setting; at the same time, set the line style (linestyle) to: dotted.



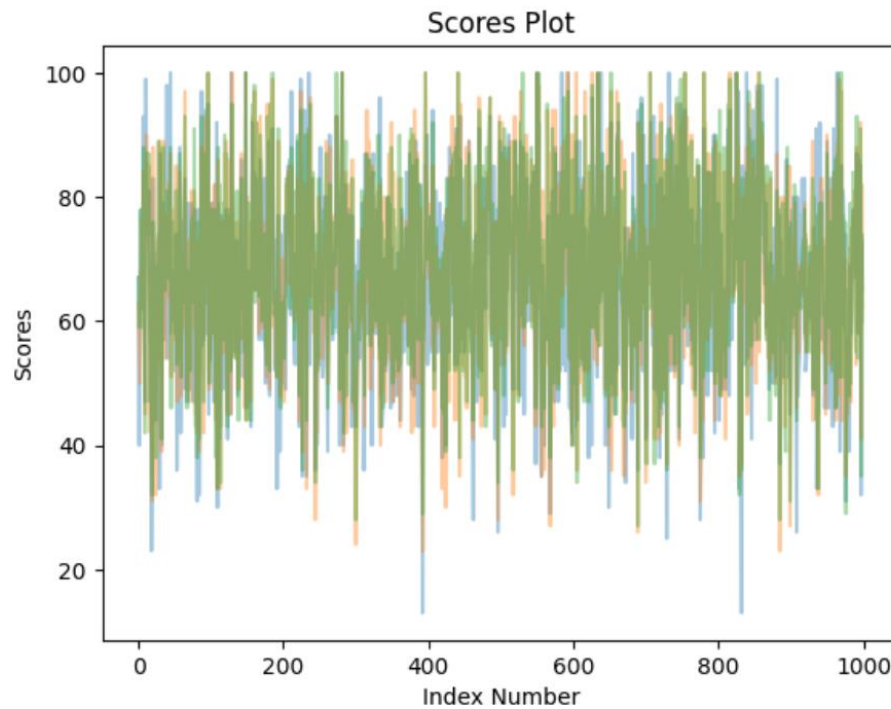
- d) Use the `ax.set_x(y)label` method to set the text description of the X and Y axes, and use `ax.set_title` to set the title of the picture..



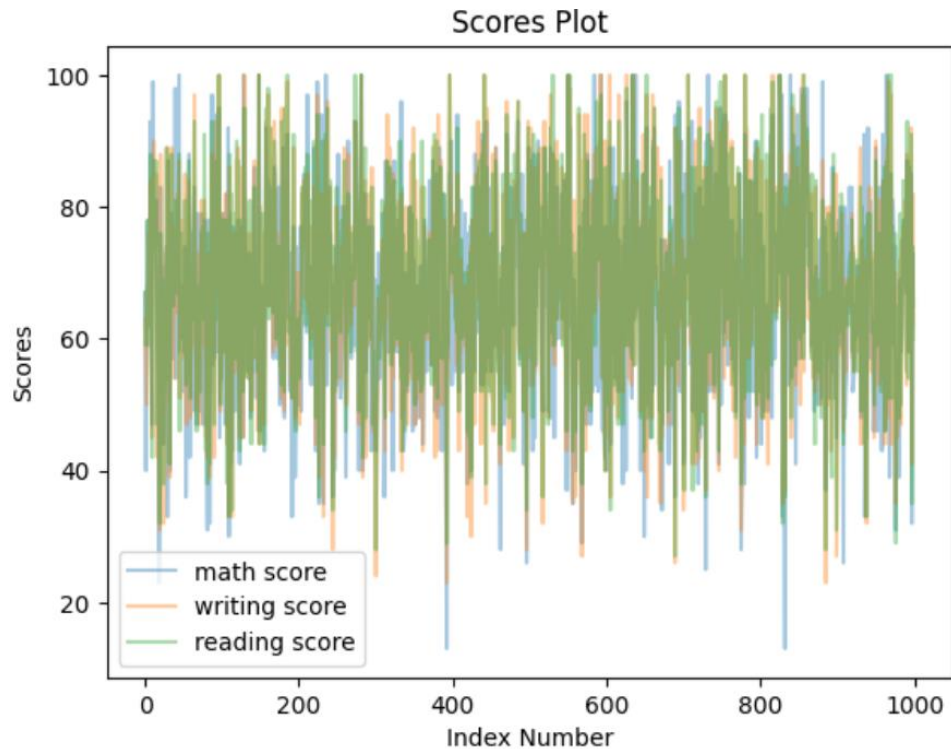
- e) Try to adjust the method in step b) to draw the data of three test scores on one graph at the same time:



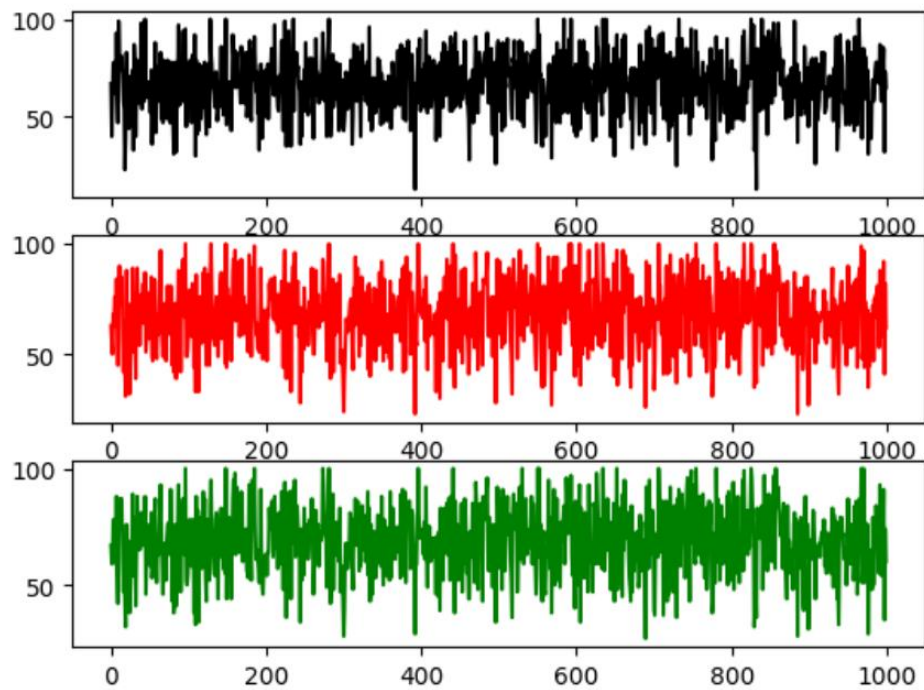
- f) As you can see, the lines are all superimposed together, making it difficult to see clearly. Try using three plot statements respectively to adjust the alpha value to adjust the transparency value of the lines, so that the superimposed lines look layered.



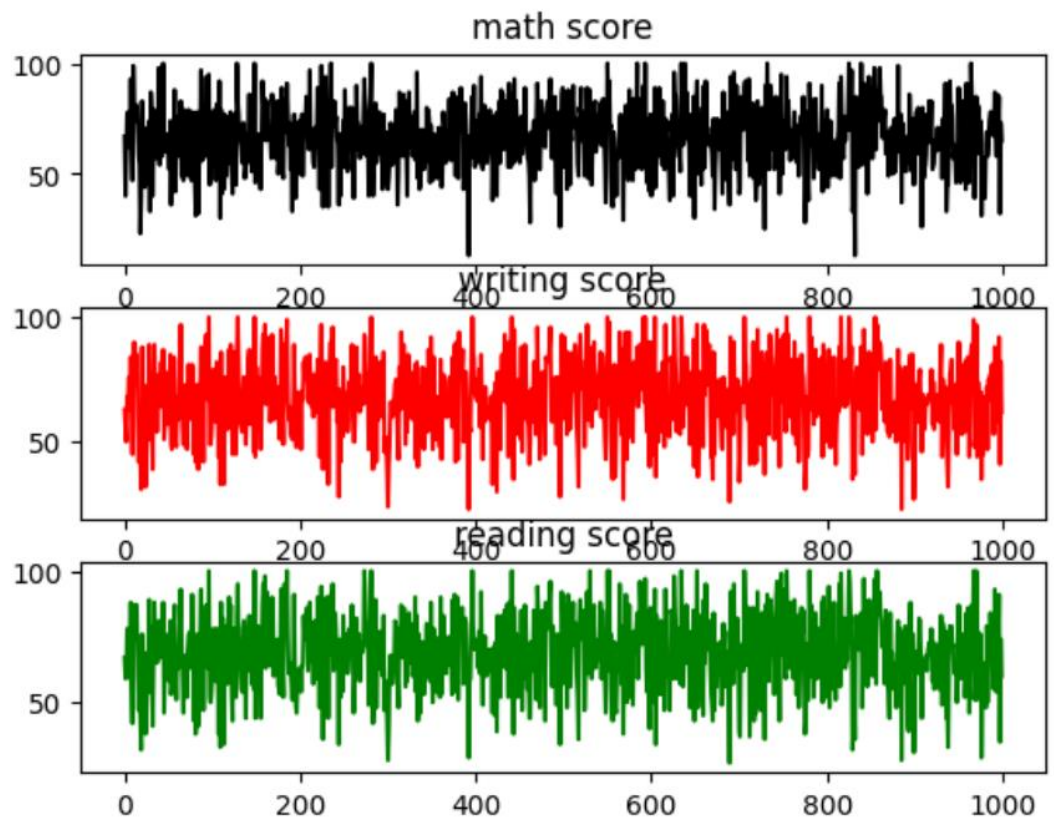
- g) At present, it is not possible to distinguish which color belongs to which test score. You can try to plot function to add label property to set its information when displaying the legend. Then add the `ax.legend` function to realize the legend display:



- h) It can be seen that even if the transparency is adjusted, the display of the three test scores is not very ideal. Now try to draw them on different graphs. Use the `nrows` and `ncol` parameters of `plt.subplots` to create a Figure object with 3 rows and 1 column, and then draw the data of three test scores in it and set different color values. To understand, the `ax` object returned by `plt.subplots` can be used as a list object to save the returned 3*1 drawing object (subplot).



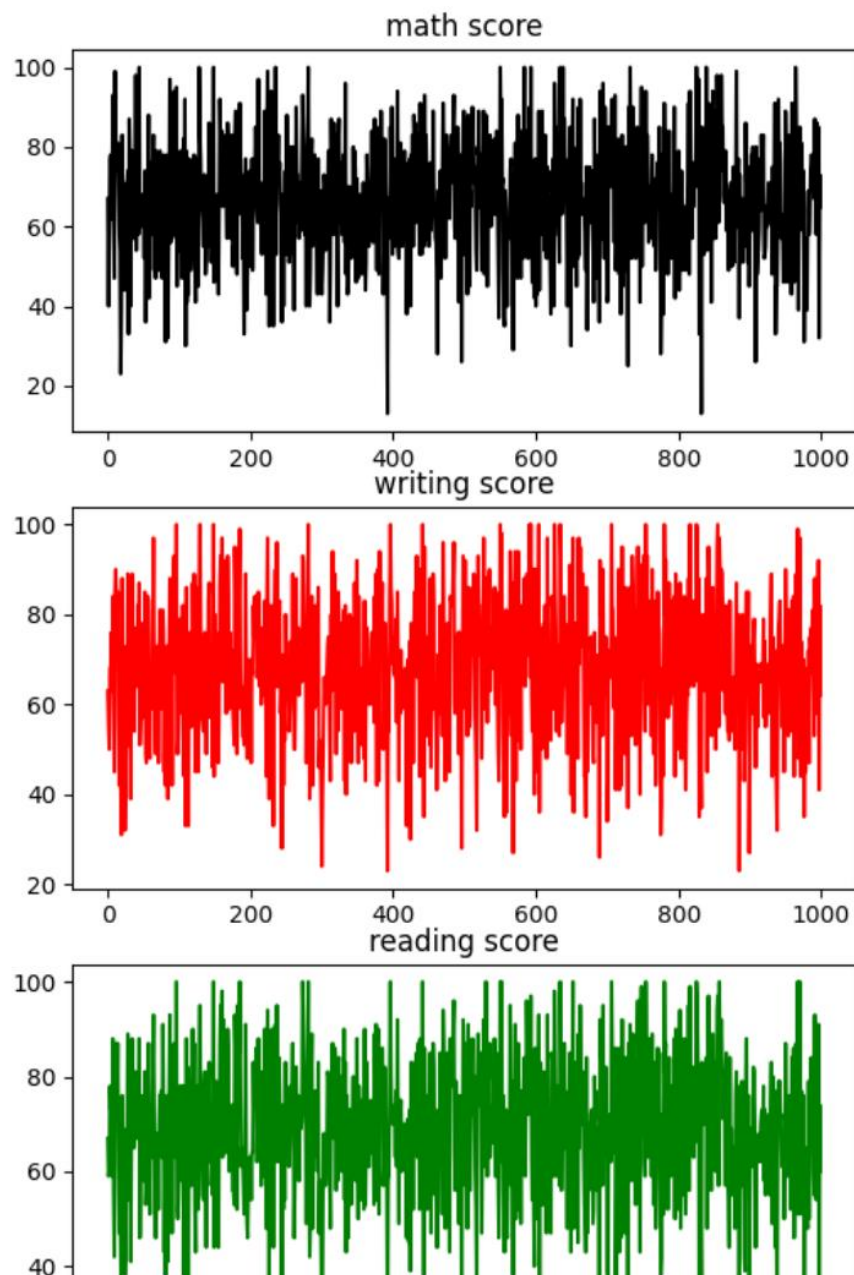
- i) Then, set the title of each sub-figure and display the corresponding test score name.

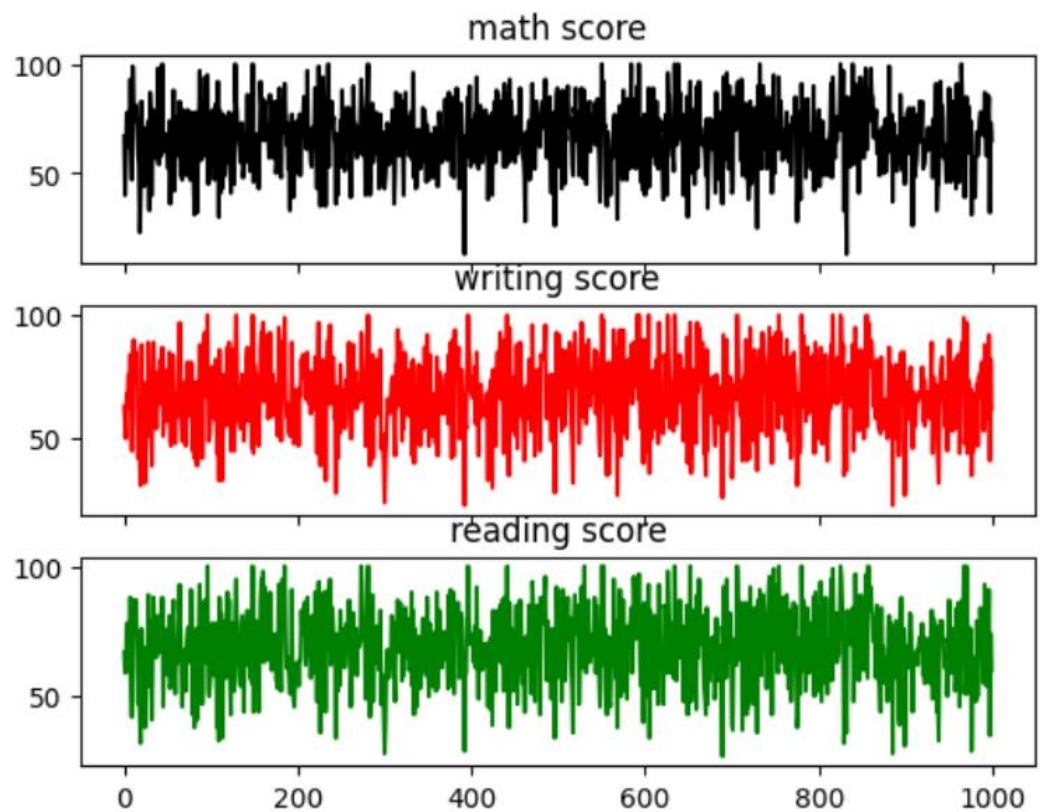


- j) However, you can see that the title of the next subgraph is obscured by the X-axis information of the previous one. There are currently two solutions available.

First, you can try to resize the entire graphics area, setting the X and Y sizes through the figsize property of the subplot.

In addition, occlusion can be eliminated by unifying the X-axis of all graphs and placing them on the bottom subgraph. Of course, these two methods can be applied at the same time to adjust the display effect to the best.

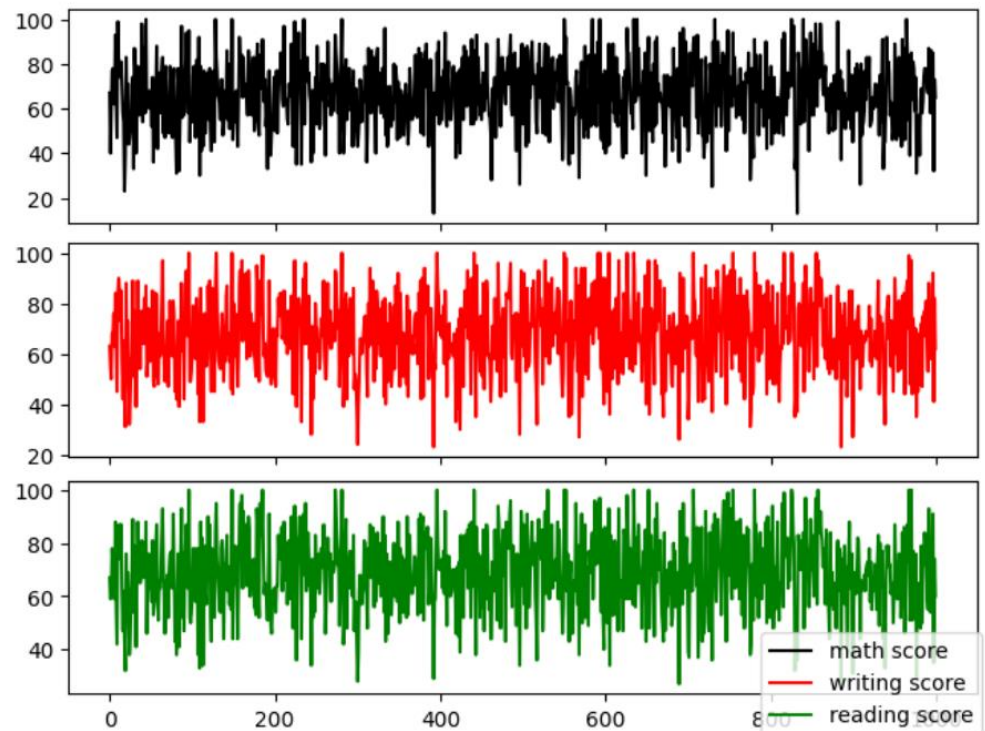




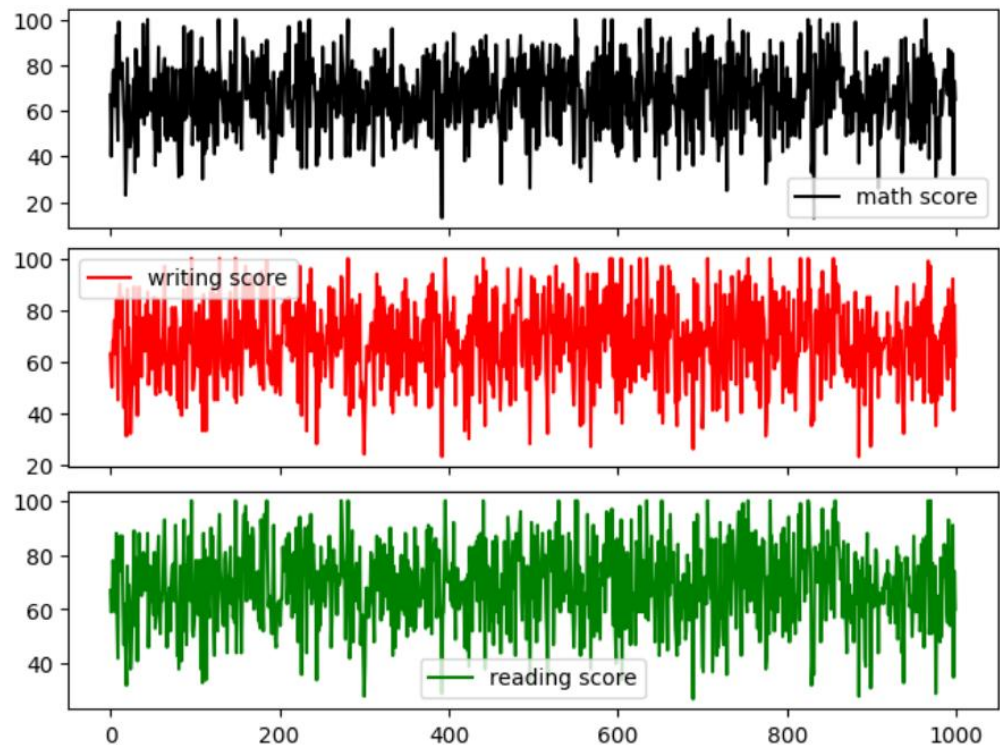
- k) Of course, in addition to titles, legends can also be used to represent the content information of each figure. In the case of multiple subplots, you can directly apply the legend method to the first fig object returned by subplot (make sure there is label information in each plot method) and then a unified legend information can be generated.

Its position can be adjusted through the legend's loc information. Supported settings: 'best', 'upper right', 'upper left', 'lower left', 'lower right', 'right', 'center left', 'center right', 'lower center', 'upper center', 'center':


```
fig.legend(loc="lower right")  
plt.show()
```



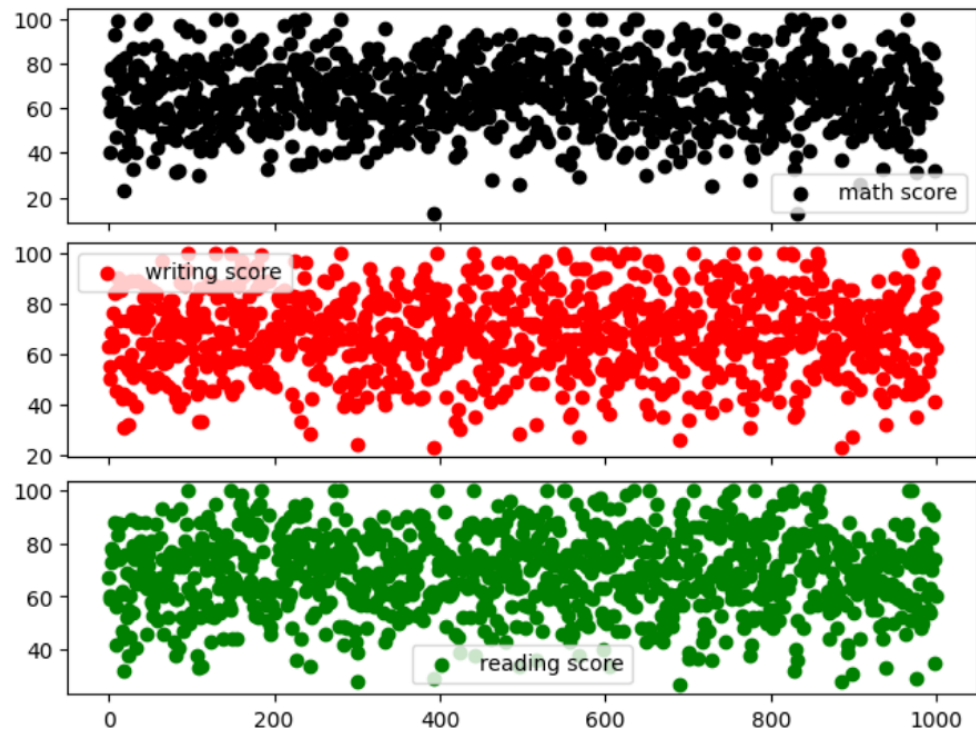
- l) Of course, similar legend settings can also be implemented for each subgraph and the legend display mode can be set separately.



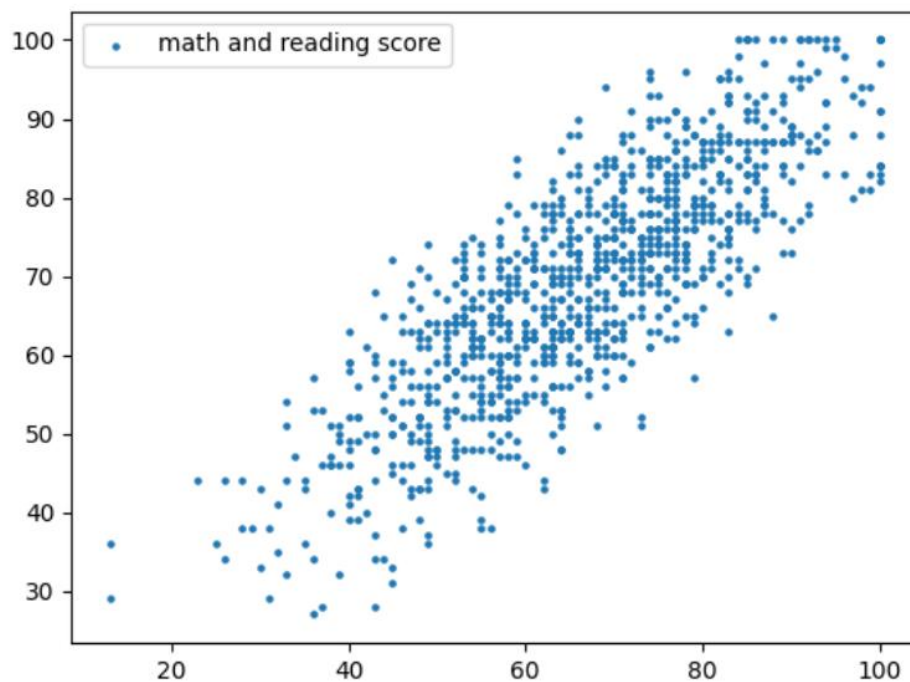


DRAW A SCATTER PLOT

- a) Scatter plots are suitable for representing discrete data. In fact, the previous test scores are more suitable to be represented by scatter plots. You can simply replace plot with scatter to realize the display method of scatter plots. Note: scatter needs to explicitly specify X and Y information.



- b) Scatter plots are also suitable for understanding the correlation between two variables by viewing graphics. For example, if we want to know that there is a direct correlation between math score and reading score, we can try to use the two variables as the X and Y axes respectively. to draw and view the drawing results. s can control the size of the point, and c can control the color.

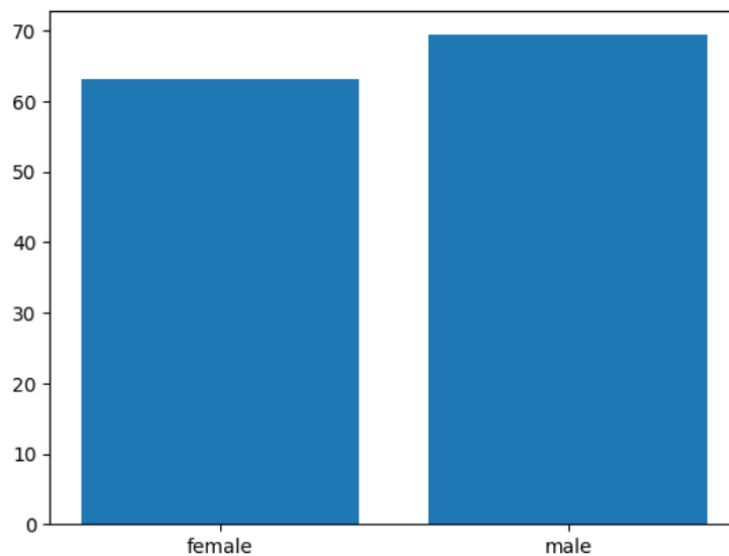


- c) It can be clearly seen that there is a positive correlation between math score and reading score. You can try to draw correlation graphs between math score and writing score and reading score and writing score.

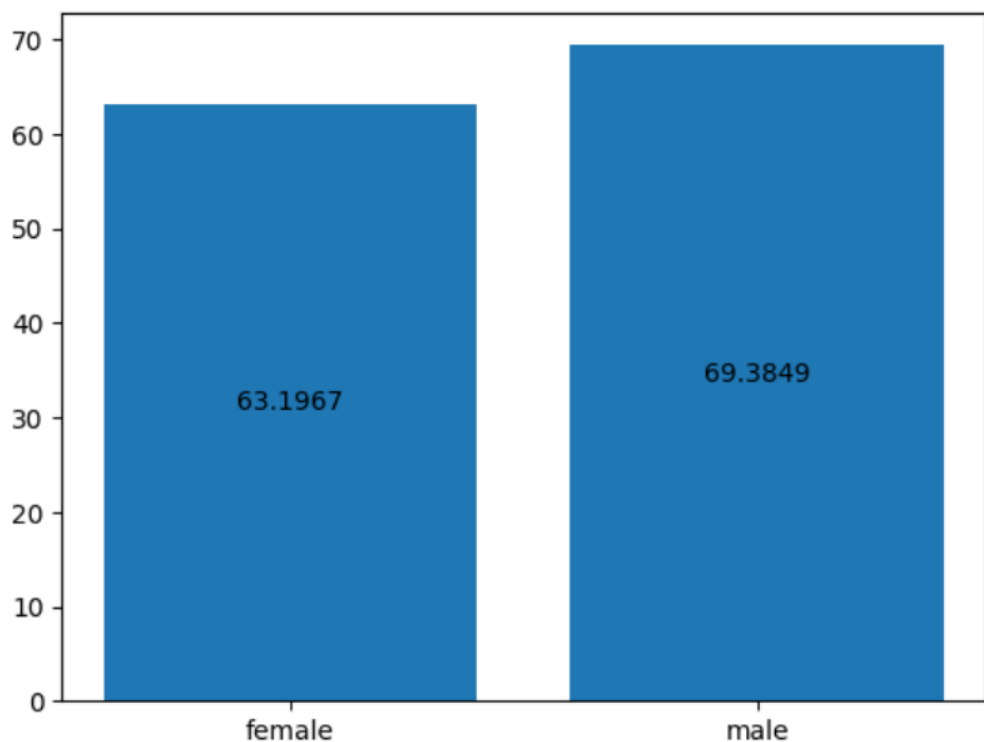
DRAW A BAR CHART

- a) Bar chart are more suitable for displaying aggregated data after grouping. One column represents a certain aspect of value information for a classified data. First, try to plot gender as the grouping key and display the average information of math score. Here you need to use the convergence method in the previous exercise.

Tip: The aggregated results have their own index and value information, which can be used as the values of the X and Y axes.

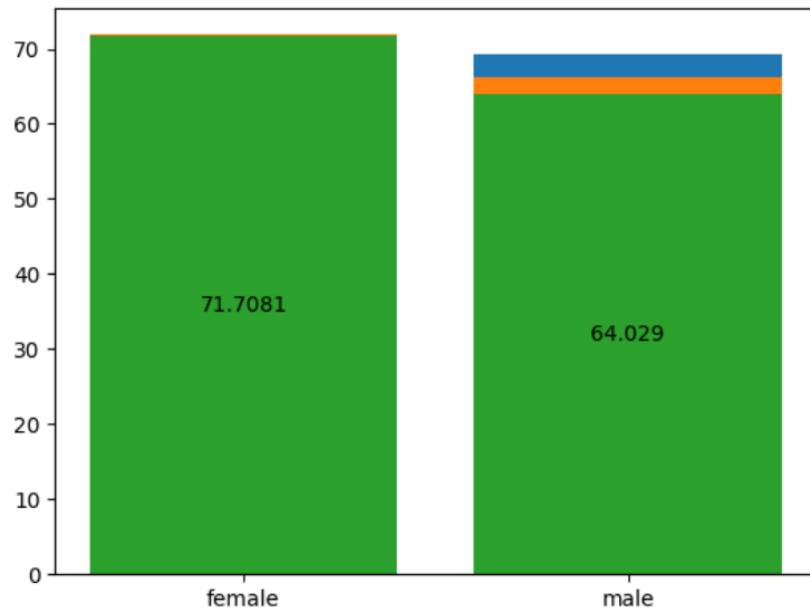


- b) The display of the value of the bar can be set through the `ax.bar_label` method. The first parameter of `bar_label` is the bar subgraph object just created. The `label_type` parameter can specify the position where the numerical value is displayed, such as center.



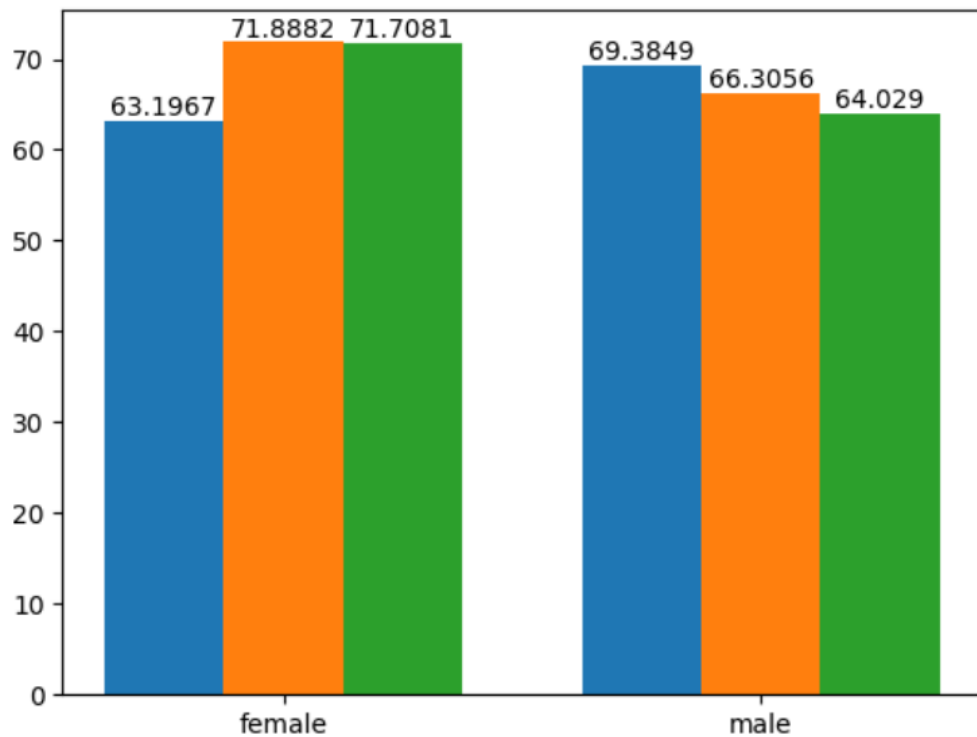
- c) If you want to display all three test scores in bar form at the same time, you need to reset the X-axis, because if the previous index information is used, the generated images will overlap:

```
gender = exam_data.groupby("gender")[["math score","reading score","writing score"]].mean()
fig, ax = plt.subplots()
bar = ax.bar(gender.index,gender["math score"].values)
bar = ax.bar(gender.index,gender["reading score"].values)
bar = ax.bar(gender.index,gender["writing score"].values)
ax.bar_label(bar,label_type='center')
plt.show()
```

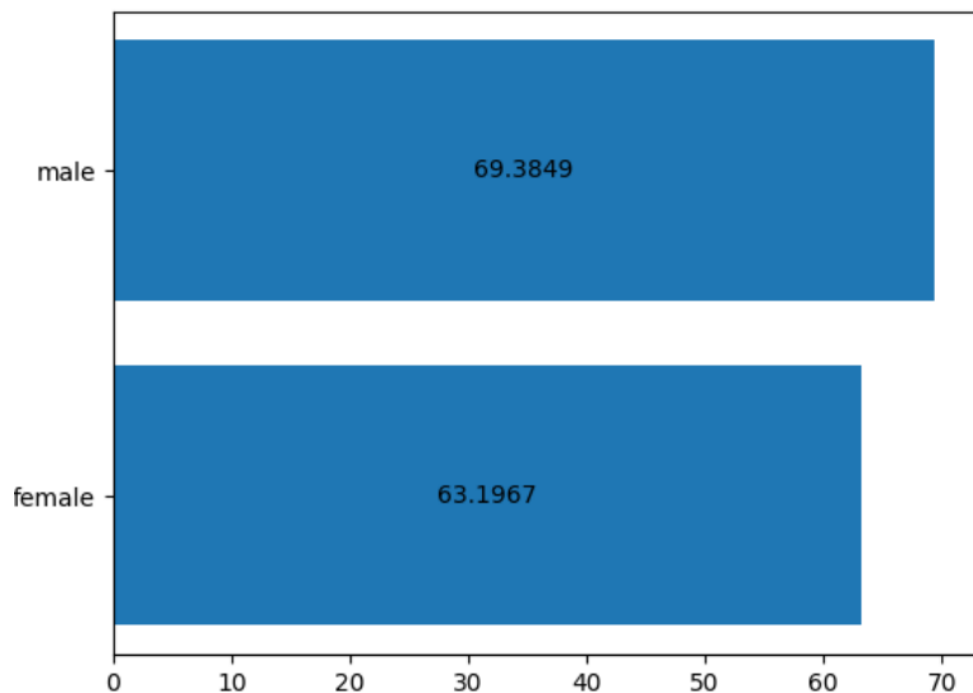


- d) Therefore, you need to adjust the value of the X-axis so that they are displayed side by side without overlapping. At the same time, you need to control the width of the column, and set and adjust the Tick setting of the X-axis. For specific implementation, please refer to:

https://matplotlib.org/stable/gallery/lines_bars_and_markers/barchart.html

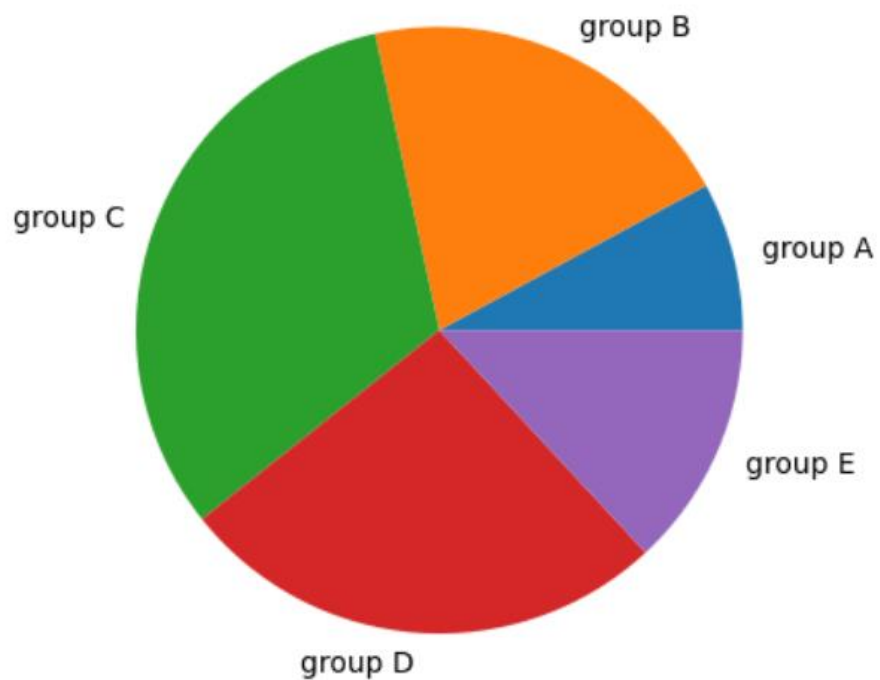


- o) The corresponding exercise of step b can also be displayed horizontally. Try to use `barh` instead of `bar` to realize the content of step b exercise. Note that the parameters of `barh` are inversely related to the parameters of `bar`: `Axes.barh(y, width, height=0.8, left=None, *, align='center')`

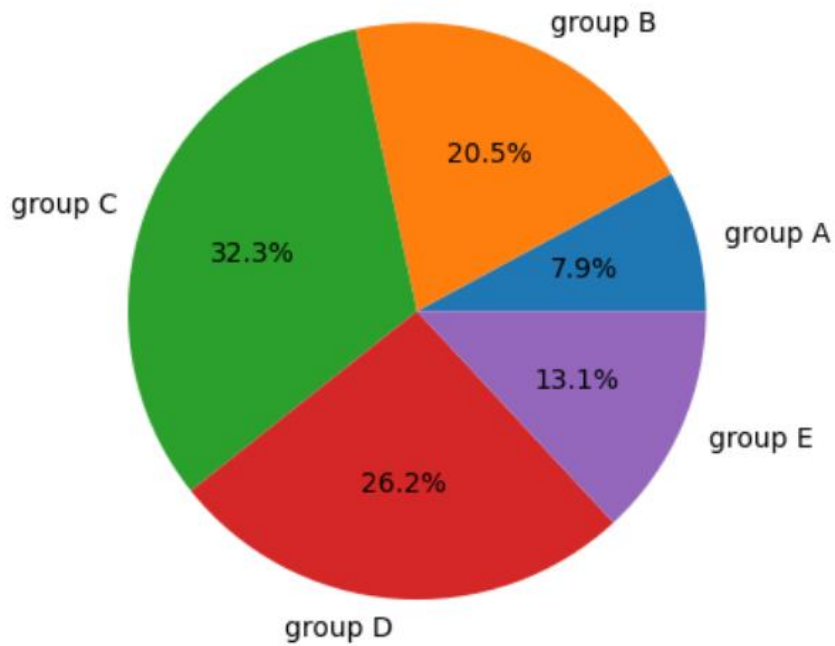


DRAW A PIE CHART

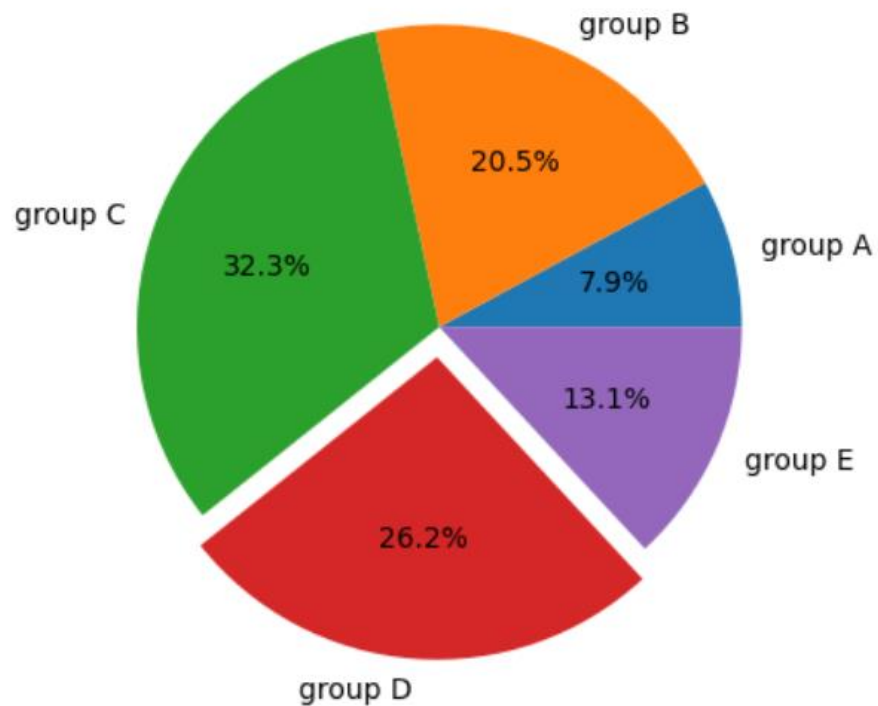
- a) Pie charts are more suitable for displaying the proportion of each grouped data in the overall data. This time you can choose exams " race/ethnicity " in the data to show the number ratio of each group. The name information of each group can be displayed through the label's information of the pie function.



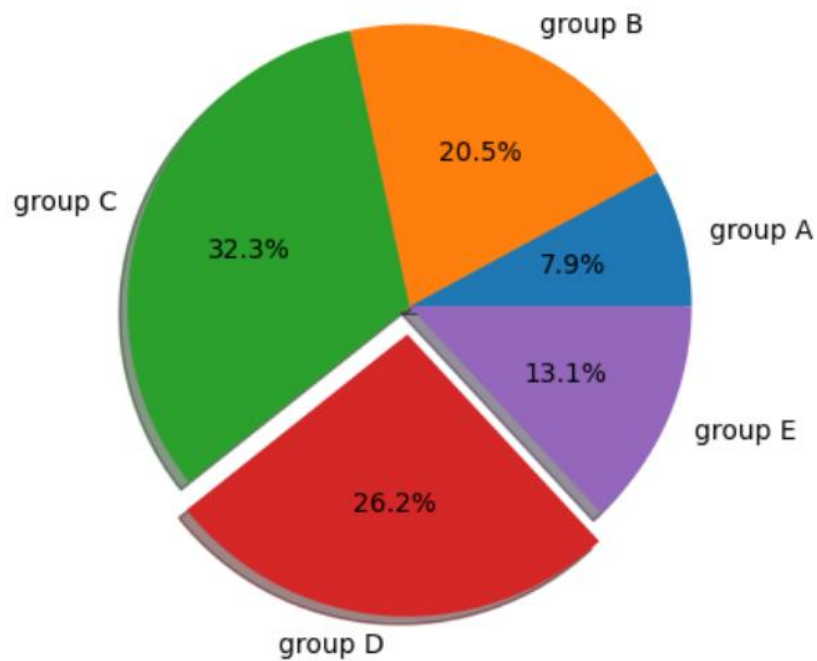
- b) By adding autopct information, you can set the display format of the value of each piece of pie, such as: `%.1f%%` displays the value with %



- c) By providing a tuple of information for the explode parameter, indicate the information, and highlight ratio of the pie that you want to highlight.

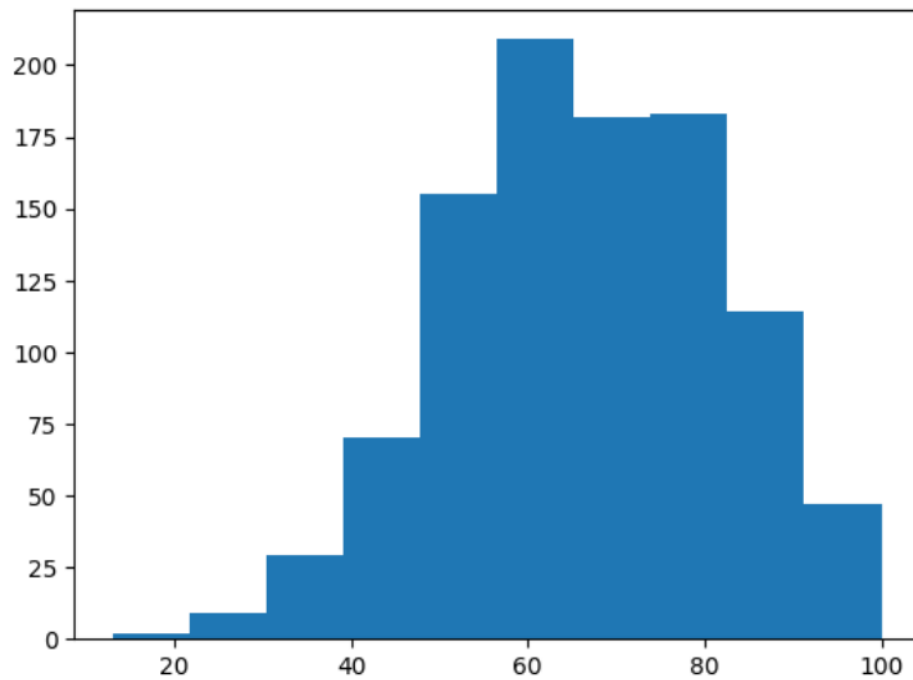


- d) Set shadow to True to achieve shadow effect.

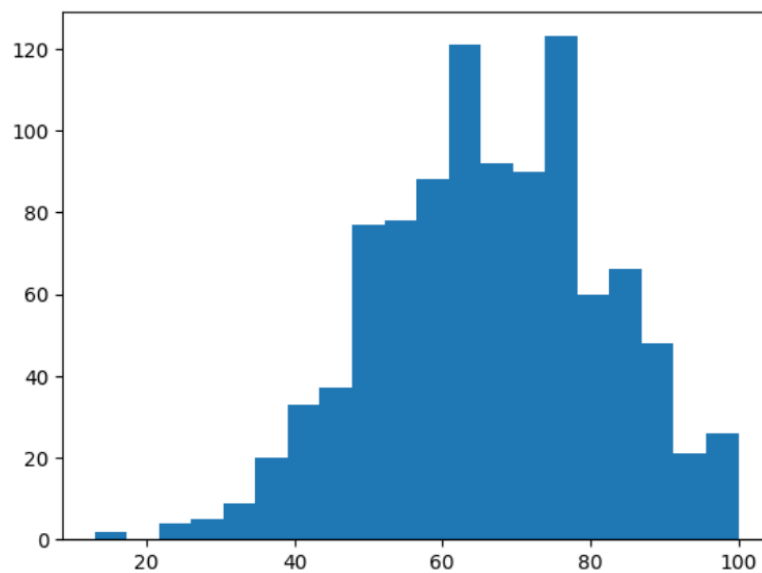


DRAW A HISTOGRAM

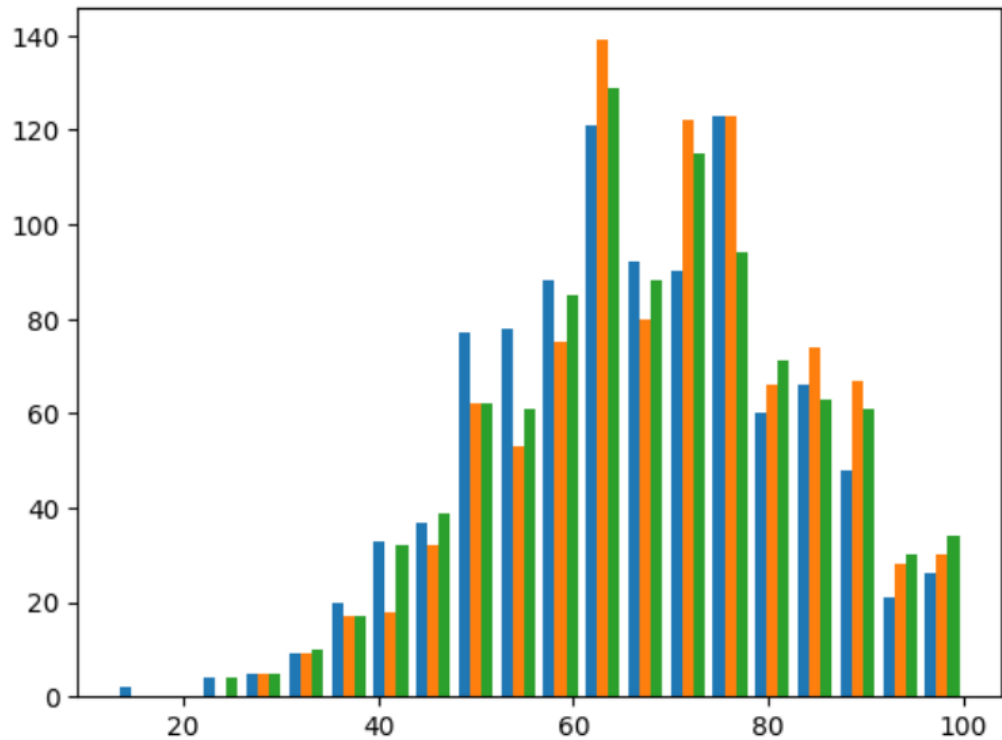
- a) Histograms are somewhat similar to pie charts. They also display grouped data of the overall data. However, the grouping method is not based on categorical data, but on the numerical data itself. It is mainly used to observe the distribution of data (probability distribution). First, implement a simple histogram display based on math score .



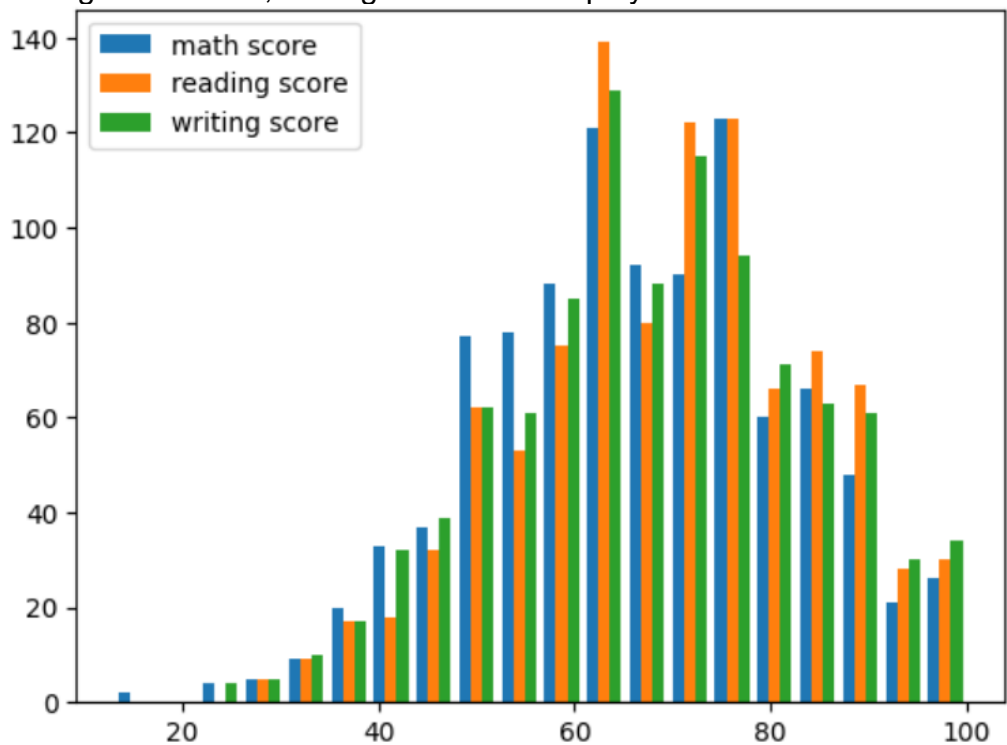
- b) The hist method is divided into 10 bins by default for display. You can adjust the desired bin value by setting the value of bins, such as 20.



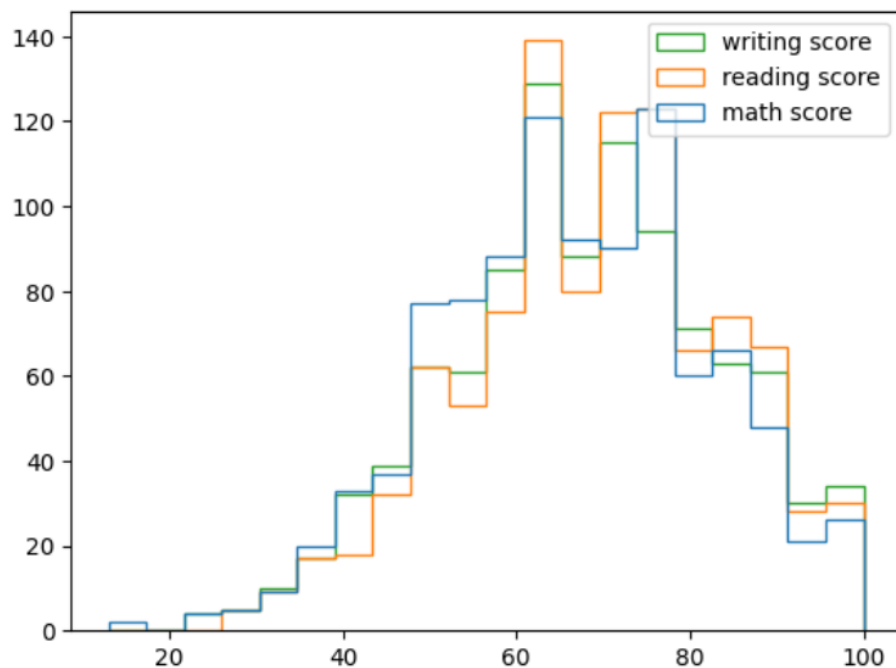
- c) When 3 test scores are provided for hist at the same time, their histograms can be displayed together at the same time.



- d) At the same time, by setting label information for hist and adding the legend method, the legend can be displayed.



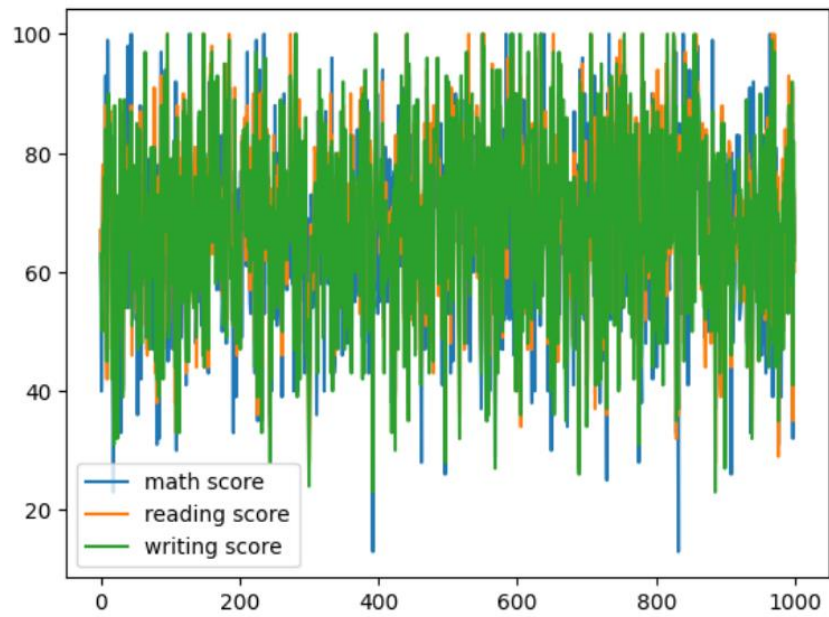
- e) The default type of hist method is bar. You can display histogram data in other ways by modifying histtype to other values, such as step.



USING PANDAS DRAWING

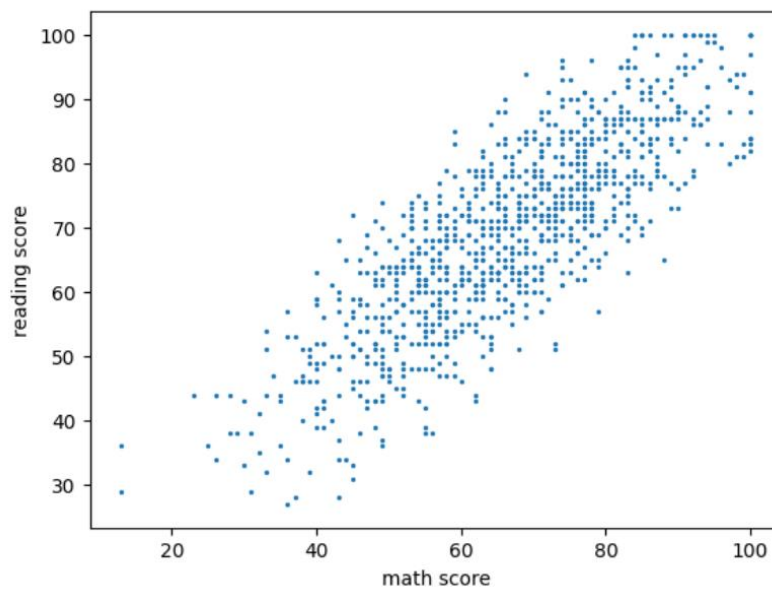
- a) Pandas itself inherits the function of drawing using libraries such as matplotlib.

Taking matplotlib as an example, the syntax is basically the same and can realize drawing more quickly, but the support for some detailed parameters is not perfect enough. Based on the exams data, you can quickly draw the line chart implemented earlier by using the plot method of the data.:

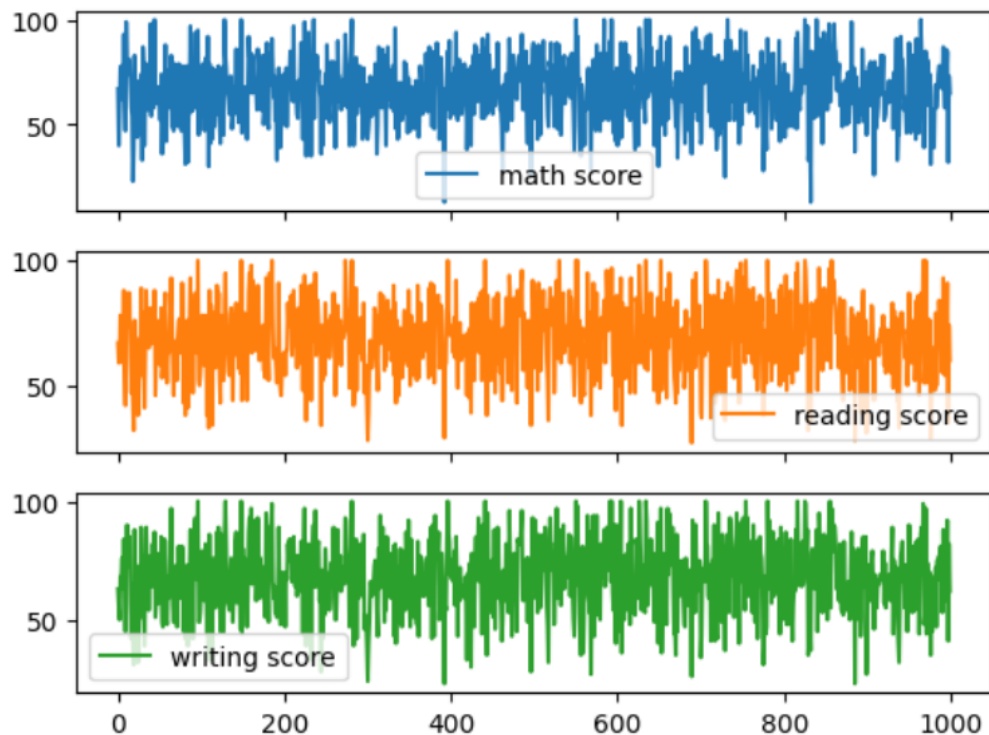


b) Scatter plot

```
exam_data[["math score","reading score"]].plot.scatter(x="math score",y="reading score",size=2)
<AxesSubplot: xlabel='math score', ylabel='reading score'>
```



c) Sub-figures



6.SQL operation

PREPARATION

- SQL for this section Related exercises, continue using Jupyter Notebook To be done.
- Click on JupyterHub File – New – Notebook in the upper left corner of the page button to create a new Notebook Used to save the content of this part of the exercise
- Select Kernel for Python 3
- As in the previous exercise, modify the notebook File name : sql.ipynb

Enter the following content to introduce the libraries required for subsequent exercises
`import pandas as pd`

```
import numpy as np

from sqlalchemy import create_engine

from sqlalchemy import text
```

PREPARE DATA

- a) First, it needs to be based on notebook Environment and SQL Statements to connect to the database and create tables required for subsequent exercises. Enter the following statement in a new cell, replacing PORT in the first statement For the port number assigned to you by your instructor:

```
conn_string="postgresql+psycopg2://sqltest:sql-  
test@10.163.213.2 : PORT /sqltest"
```

```
engine = create_engine ( conn_string )
```

```
conn = engine.connect ()
```

- b) Each student will receive a separate PostgreSQL database instance and a database named sqltest The database is used for exercises. The above connection information: conn_string sets the client's connection configuration through the corresponding driver and address information, and finally conn It is an object after connecting to the database, and the subsequent SQL This connection object is used indirectly or indirectly between executions.

- c) Currently, no tables exist in the database. You need to create a new table first. Of course, you first need to fill in the table with corresponding data. We continue to use the contents of the exams . csv data file. First , read_csv Read data into df_students_exam variables, then, utilized to _sql function, insert it into the database you just connected, where to _sql The function parameters are as follows:

```
df_students_exam.to_sql(name="exam ",con  
=conn,if_exists="replace")
```



- d) The `name` parameter is the name of the database table, `conn` Points to the connection just created, `if_exists="replace"` means if name The set table already exists and will be replaced. If this function is used in a production environment, it needs to be executed with caution and the database should be backed up in advance. At the same time, you need to add `conn.commit()` after this statement to indicate that this change is actually executed in the database. If there is no abnormal output, the database table has been created successfully. You can then start executing SQL .
- e) Behind, SQL The execution time is based on Pandas interfaces and sqlalchemy library, the basic syntax is:

```
pd.read_sql(text(" SQL _STATEMENT"),conn)
```

- f) Where `SQL _STATEMENT` is the SQL to be executed Statements must be enclosed in double quotes and end with `;` . Try to execute the following SQL to query the exam The first 5 rows of data in the table can be compared with the data set before inserting into the database:

```
select * from exam limit 5;
```

```
pd.read_sql(text("select * from exam limit 5;"),conn)
```

index	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score	
0	0	male	group A	high school	standard	completed	67	67	63
1	1	female	group D	some high school	free/reduced	none	40	59	55
2	2	male	group E	some college	free/reduced	none	59	60	50
3	3	male	group B	high school	standard	none	77	78	68
4	4	male	group E	associate's degree	standard	completed	78	73	68

```
df_students_exam.head(5)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	male	group A	high school	standard	completed	67	67	63
1	female	group D	some high school	free/reduced	none	40	59	55
2	male	group E	some college	free/reduced	none	59	60	50
3	male	group B	high school	standard	none	77	78	68
4	male	group E	associate's degree	standard	completed	78	73	68

- g) It can be found that the difference between the two results is that there will be one more index in the database Column, which comes from Pandas Index, if you do not want the index to be inserted into the database, you can use `to_sql` Set `index` in function is `False`. Try recreating the database table without inserting the index into the database as a column:

```
pd.read_sql(text("select * from exam limit 5;"),conn)
```

	gender	race/ethnicity	parental level of education	lunch	test preparation course	math score	reading score	writing score
0	male	group A	high school	standard	completed	67	67	63
1	female	group D	some high school	free/reduced	none	40	59	55
2	male	group E	some college	free/reduced	none	59	60	50
3	male	group B	high school	standard	none	77	78	68
4	male	group E	associate's degree	standard	completed	78	73	68

BASIC SQL OPERATE

a) The previous step was to use Pandas to_sql The statement creates a database table directly based on the existing data set information. SQL The create table statement in is a more standard function of creating a database table. Based on the content of this exercise book, simply design a table including the following fields:

- id : integer type, primary key
- chapter_name : text type, unique
- keywords : text type
- subchapter_num : integer type
- description : text type

Try based on create table syntax creates the corresponding SQL statement

```
CREATE TABLE {table_name} (
    {column_name_1} {data_type_1} {column_constraint_1},
    ...
    {column_name_last} {data type last} {column constraint last}, );
```

b) the read_sql just now Function to execute multiple rows of SQL in the target database statement, it is recommended to first convert the SQL The statement is saved in a variable, and multi-line text is set in three quotes, and then executed, as in the following example:



```
sql_text="""select * from
              exercise_book"""
pd.read_sql(text(sql_text),conn)
```

- c) Use read_sql Execute create table The statement will cause an error similar to the following, which can be ignored:

```
1345          "This result object does not return rows. "
1346          "It has been closed automatically."
1347      ) from err
```

ResourceClosedError: This result object does not return rows. It has been closed automatically.

```
conn.commit()
```

- d) conn. commit () needs to be executed in a separate cell at the back to ensure that the creation is actually submitted to the database. Subsequently, you can pass b Check the query results by following the statement in the example to determine whether the database was created successfully:

```
sql_text="""select * from
              exercise_book"""
pd.read_sql(text(sql_text),conn)
```

id	chapter_name	keywords	subchapter_num	description
----	--------------	----------	----------------	-------------

- e) Subsequently, you can prepare to insert data, using the insert into syntax:

```
INSERT INTO {table_name} ({column_1}, {column_2}, ...{column_last})
VALUES ({column_value_1}, {column_value_2}, ... {column_value_last});
```

Note that column names do not need to be enclosed in quotation marks, but literal values must be enclosed in single quotation marks (double quotation marks are not allowed).

The following is an example, please refer to:

```
INSERT INTO
exercise_book( id,chapter_name ,keywords,subchapter_num,description)
```

VALUES (1, 'env verification', 'env,verification ', 1, 'env verification');

Similar errors with the previous table creation will also occur, please ignore it, and remember to execute conn.commit () in an independent cell .

Please try to create 4 records.

	id	chapter_name	keywords	subchapter_num	description
0	1	env verification	env,verification	1	env verification
1	2	Python basic	Python,basic	4	Python basic
2	3	Numpy basic	Numpy,basic	7	Numpy basic
3	4	Pandas basic	Pandas,basic	12	Pandas basic

- f) At this time, if you find a problem with a certain record, you can update it Operation to update it, of course you need to cooperate with where to filter the rows that need to be updated. Basic syntax:

```
UPDATE {table_name}
SET {column_1} = {column_value_1},
{column_2} = {column_value_2},
WHERE
{conditional};
```

andas basic information of the 4th record to Pandas operation.

	id	chapter_name	keywords	subchapter_num	description
0	1	env verification	env,verification	1	env verification
1	2	Python basic	Python,basic	4	Python basic
2	3	Numpy basic	Numpy,basic	7	Numpy basic
3	4	Pandas operation	Pandas,operation	12	Pandas operation

- g) Try using DELETE below Delete id For rows 2 and 4, you need to use where To set conditions, DELETE Syntax:



DELETE FROM {table_name}

WHERE {conditional};

```
sql_text="""select * from
            exercise_book"""
pd.read_sql(text(sql_text),conn)
```

	id	chapter_name	keywords	subchapter_num	description
0	1	env verification	env,verification	1	env verification
1	3	Numpy basic	Numpy,basic	7	Numpy basic

- h) Finally, try using DROP TABLE command to delete this newly created table. Trying the query again will get the error output that the table (relation) does not exist.

```
sql_text="""select * from
            exercise_book;"""
pd.read_sql(text(sql_text),conn)
```

```
-----
UndefinedTable                                Traceback (most recent call last)
File C:\Python310\lib\site-packages\sqlalchemy\engine\base.py:1964, in Con
t, statement, parameters)
    1963     if not evt_handled:
-> 1964         self.dialect.do_execute(
    1965             cursor, str_statement, effective_parameters, context
    1966         )
    1968 if self._has_events or self.engine._has_events:
```

```
File C:\Python310\lib\site-packages\sqlalchemy\engine\default.py:747, in D
arameters, context)
    746 def do_execute(self, cursor, statement, parameters, context=None):
-> 747     cursor.execute(statement, parameters)
```

```
UndefinedTable: relation "exercise_book" does not exist
LINE 2:         exercise_book;
```