tutor_week3

生成随机数 伪数据

import numpy as np
np.random.rand(d1,d2,d3...dn)

作用：生成一个给定形状的数组，数组中的元素随机采样自[0，1)的均匀分布。d是数组的维度。

np.random.rand(2,2)生成一个2行2列的随机数组

np.random.random(size)

作用：random函数与rand类似，生成随机数或随机数数组，元素值也是从[0，1)的均匀分布中抽取。

size默认返回一个单一的随机数。size = (3,2)

np.random.random((2,2))生成一个2行2列的随机数组

```
import numpy as np

print(np.random.rand(4))
print(np.random.rand(2,2))
```

✓ 0.0s

```
[0.28535276 0.02651113 0.62201206 0.75807965]
[[0.14780734 0.00157377]
 [0.15766652 0.3811228 ]]
```

```
import numpy as np
randnum1 =  np.random.random(4)
randnum2 = np.random.random((2,2))
print(randnum1)
print(randnum2)
```

✓ 0.0s

```
[0.46475813 0.93478973 0.36696959 0.84188119]
[[0.09480598 0.41575565]
 [0.38377707 0.58768621]]
```

(0,1)分布的均匀分布

定义：一个随机变量X的取值范围在区间(0,1)内，并且在该区间内的每个数值都有相同的概率密度函数。也就是说，任何一个值X在(0,1)区间内取到的概率都是相等的，即概率密度函数为常数。

1 定义

如果一个随机变量X的密度函数为：

$$f(x)=\begin{cases}1 & 0<x<1\\ 0 & \text{others}\end{cases} \tag{1}$$

则称随机变量X在$(0,1)$区间上均匀分布。因为$f(x)\geq 0$，且$\int_{-\infty}^{\infty}f(x)dx=1$，所以$f(x)$是概率密度函数。因为仅当$x\in(0,1)$才有$f(x)>0$，所以$X$必然取值在$0,1$之间。又因为在$(0,1)$之间时，$f(x)$是常数，所以$X$等概的取$(0,1)$的值。

一般来讲，我们称X为区间(α,β)上服从均匀分布的随机变量，如果它的密度函数为：

$$f(x)=\begin{cases}\frac{1}{\beta-\alpha} & \alpha<x<\beta\\ 0 & \textbf{others}\end{cases} \tag{2}$$

生成随机数

import numpy as np

np.random.standard_normal(size=None)

作用：standard_normal函数生成具有标准正态分布（均值为0，标准差为1）的随机数或随机数数组。size参数指定了输出数组的形状。例如：size = (2,3)

np.random.normal(loc,scale,size)

作用：normal函数生成具有正态分布的随机数或随机数数组，可以指定均值（loc）、标准差(scale)和输出数组的形状（size）

import random

random.random()

作用：每次调用生成一个[0，1)范围内的随机浮点数。没有参数输入

```
import numpy as np
randnum1 =  np.random.normal(0,2,(1,2))
randnum2 = np.random.standard_normal((1,2))
print(randnum1)
print(randnum2)
```

✓ 0.0s

```
[[-0.29150065 -0.7601947 ]]
[[-1.42654386 -0.67205759]]
```

```
import random
print(random.random())
```

✓ 0.0s

```
0.6228304705613547
```

生成日期

```
import pandas as pd
pd.date_range(startDate, periods=N, freq="D")
```

作用：生成日期范围的函数，适合于创建时间序列数据

startDate：起始日期
periods：时间序列长度，N为int
freq：生成日期的频率，"H"1小时，"D" 日频，"W" 周频，"M" 月频

```
import pandas as pd
startDate1 = '20231204'
pd.date_range(startDate1, periods=5, freq="D")
```

✓ 0.6s

```
DatetimeIndex(['2023-12-04', '2023-12-05', '2023-12-06', '2023-12-07',
               '2023-12-08'],
              dtype='datetime64[ns]', freq='D')
```

```
import pandas as pd
startDate2 = pd.to_datetime('2023-12-04 00:00:00')
print(type(startDate2))
pd.date_range(startDate2, periods=5, freq="s")
```

[26] ✓ 0.0s

```
<class 'pandas._libs.tslibs.timestamps.Timestamp'>

DatetimeIndex(['2023-12-04 00:00:00', '2023-12-04 00:00:01',
               '2023-12-04 00:00:02', '2023-12-04 00:00:03',
               '2023-12-04 00:00:04'],
              dtype='datetime64[ns]', freq='S')
```

生成csv文件

```
import pandas as pd
test_df.to_csv("data/apple.csv")
test_df.to_csv(path_or_buf=None, sep=',', na_rep = ' ', columns=None, header=True,
index=True)
```

作用：生成csv

path_or_buffer: 保存文件的路径
sep:指定分隔符,默认为逗号。假如是分号分隔， sep = ';'
na_rep :缺失值默认是空。na_rep='NA'缺失值保存为NA,
columns： 默认导出 DataFrame 的全部数据,可选择列保存。columns = ['time','close']
header: 是否保留列名，默认保存True
index: 是否保留行索引，默认保存True

读取csv文件

pd.read_csv(filepath_or_buffer, sep=',', header='infer', names=None, index_col=None, dtype=None)

作用：读取csv

filepath_or_buffer: 读取的文件路径
sep: 指定分隔符, 默认为逗号。假如是分号分隔， sep = ';'
index_col:指定某列为索引, 可以指定单列或者多列， index_col = ['id','name']
dtype: 在读取数据时，设定字段类型。 例如：学号id是：0000321，如果默认读取的时候，会显示为321，所以这个时候要转为字符串类型，才能正常显示. dtype = {"id": str}

标签索引与位置索引

位置索引：

df.iloc[x_int,y_int], df.iloc[x_int], df.iloc[:,y_int]

作用：基于整数位置的索引，使用数据的整数索引来选择数据

df.iat[x_int,y_int]

作用：与.iloc相似，但专门用于选取单个元素，更快。

```
import pandas as pd
test_stock_dict = {
    "time": ["2024-03-16 09:30:00", "2024-03-16 10:00:00", "2024-03-16 10:30:00",
             "2024-03-16 11:00:00", "2024-03-16 11:30:00"],
    "price": [100.5, 101.2, 100.8, 99.9, 100.1],
    "amount": [500, 600, 550, 650, 600]
}
test_df = pd.DataFrame(test_stock_dict)
test_df
```

✓ 0.0s

	time	price	amount
0	2024-03-16 09:30:00	100.5	500
1	2024-03-16 10:00:00	101.2	600
2	2024-03-16 10:30:00	100.8	550
3	2024-03-16 11:00:00	99.9	650
4	2024-03-16 11:30:00	100.1	600

```
open_info =  test_df.iloc[0]
open_info
```

✓ 0.0s

```
time      2024-03-16 09:30:00
price                   100.5
amount                    500
Name: 0, dtype: object
```

```
price_col = test_df.iloc[:,1]
price_col
```

✓ 0.0s

```
0    100.5
1    101.2
2    100.8
3     99.9
4    100.1
Name: price, dtype: float64
```

```
open_price = test_df.iloc[0,1]
open_price
```

✓ 0.0s

```
100.5
```

标签索引与位置索引

标签索引:

df.loc[x,y], df.loc[x],df.loc[:,y]

作用:标签的索引，即使用数据的索引名或列名来选择数据

df.at[x,y]

作用:与.loc相似，但专门用于选取单个元素，更快。

```python
import pandas as pd
test_stock_dict = {
    "time": ["2024-03-16 09:30:00", "2024-03-16 10:00:00", "2024-03-16 10:30:00",
             "2024-03-16 11:00:00", "2024-03-16 11:30:00"],
    "price": [100.5, 101.2, 100.8, 99.9, 100.1],
    "amount": [500, 600, 550, 650, 600]
}
test_df = pd.DataFrame(test_stock_dict)
newlist = [item for item in range(10,15)] # [10, 11, 12, 13, 14]
test_df.index = newlist
test_df
```

✓ 0.0s

	time	price	amount
10	2024-03-16 09:30:00	100.5	500
11	2024-03-16 10:00:00	101.2	600
12	2024-03-16 10:30:00	100.8	550
13	2024-03-16 11:00:00	99.9	650
14	2024-03-16 11:30:00	100.1	600

```python
open_info =  test_df.loc[10]
open_info
```

✓ 0.0s

```
time      2024-03-16 09:30:00
price                   100.5
amount                    500
Name: 10, dtype: object
```

```python
price_col =  test_df.loc[:,'price']
price_col
```

✓ 0.0s

```
10    100.5
11    101.2
12    100.8
13     99.9
14    100.1
Name: price, dtype: float64
```

```python
open_price = test_df.loc[10,'price']
open_price
```

✓ 0.0s

```
100.5
```

标签索引与位置索引

标签索引和位置索引的作用：用于行或列的选择，可以选取单行、单列、多行、多列，或是行列交叉的区域

(.at,.iat) 和 (.loc,.iloc)的主要区别：df.iat 和 df.at 不能切片，只能取一个数，所以需要传入[x, y]两个参数。使用频率很少，推荐大家使用df.iloc & df.loc

```
open_price =  test_df.at[10,'price']
open_price
```

✓ 0.0s

```
100.5
```

```
open_price =  test_df.iat[0,1]
open_price
```

✓ 0.0s

```
100.5
```

```
price_col =  test_df.at[:,'price']
price_col
```

51] ⊗ 0.0s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
c:\Users\86182\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:

c:\Users\86182\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

c:\Users\86182\anaconda3\lib\site-packages\pandas\_libs\index.pyx in pandas._libs.index.IndexEngine.get_loc()

TypeError: 'slice(None, None, None)' is an invalid key

During handling of the above exception, another exception occurred:

InvalidIndexError                         Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\3732588704.py in <module>
----> 1 price_col =  test_df.at[:,'price']
      2 price_col

c:\Users\86182\anaconda3\lib\site-packages\pandas\core\indexing.py in __getitem__(self, key)
   2273             return self.obj.loc[key]
   2274
-> 2275         return super().__getitem__(key)
   2276
...
-> 5651             raise InvalidIndexError(key)
   5652
   5653     @cache_readonly

InvalidIndexError: slice(None, None, None)
Output is truncated. View as a scrollable element or open in a text editor. Adjust cell output settings...
```

homework_week3

1. Suppose you have a DataFrame named sales_data containing information about sales transactions with columns Date, Product, and Revenue. You need to calculate the total revenue generated by each product category. Which of the following code snippets accomplishes this task?

a) sales_data['Product'].groupby(sales_data['Revenue']).sum()

b) total_revenue = sales_data.groupby('Product')['Revenue'].sum()

c)total_revenue = sales_data.groupby('Revenue').sum('Product')

d) total_revenue = sales_data.groupby('Product')['Revenue'].mean()

```python
import pandas as pd
data = {
    'Date': ['2023-03-01', '2023-03-01', '2023-03-02', '2023-03-02', '2023-03-03'],
    'Product': ['Product A', 'Product B', 'Product A', 'Product C', 'Product B'],
    'Revenue': [200, 150, 190, 300, 230]
}

sales_data = pd.DataFrame(data)
sales_data
```

✓ 0.0s

	Date	Product	Revenue
0	2023-03-01	Product A	200
1	2023-03-01	Product B	150
2	2023-03-02	Product A	190
3	2023-03-02	Product C	300
4	2023-03-03	Product B	230

```python
print('A', sales_data['Product'].groupby(sales_data['Revenue']).sum())
print('B', sales_data.groupby('Product')['Revenue'].sum())
print('C', sales_data.groupby('Revenue').sum('Product'))
print('D', sales_data.groupby('Product')['Revenue'].mean())
```

✓ 0.0s

```
A Revenue
150    Product B
190    Product A
200    Product A
230    Product B
300    Product C
Name: Product, dtype: object
B Product
Product A    390
Product B    380
Product C    300
Name: Revenue, dtype: int64
C Empty DataFrame
Columns: []
Index: [150, 190, 200, 230, 300]
D Product
Product A    195.0
Product B    190.0
Product C    300.0
Name: Revenue, dtype: float64
```

2. Consider the traffic example from the lecture. Which of the following stations has most traffic on average?

A) 42 ST-PORT AUTH

B) EAST BROADWAY

C) BROOKLYN BRIDGE

D) 34 ST-PENN STA

```
res = df_traffic.groupby('station').apply(lambda x: x['traffic'].mean()).sort_values()
res
```

[148] ✓ 0.0s

```
station
ORCHARD BEACH     5.628571e+00
BROAD CHANNEL     7.128571e+01
TOMPKINSVILLE     8.367347e+01
BEACH 105 ST      1.367714e+02
BEACH 44 ST       1.844000e+02
                      ...
BROAD ST          8.249579e+04
JOURNAL SQUARE    8.428302e+04
61 ST WOODSIDE    1.597926e+05
HUNTS POINT AV    6.208102e+06
182-183 STS       1.557213e+08
Length: 373, dtype: float64
```

3. Suppose you have a DataFrame named employee_data with columns Name, Department, and Salary. You need to filter this DataFrame to include only the employees who belong to the "Finance" department and have a salary greater than $50,000. Which of the following code snippets achieves this task?

a) filtered_data = employee_data[(employee_data['Department'] == 'Finance') & (employee_data['Salary'] > 50000)]

b) filtered_data = employee_data((employee_data['Department'] == 'Finance') & (employee_data['Salary'] > 50000))

c) filtered_data = employee_data['Department == "Finance" and Salary > 50000']

d) filtered_data = employee_data('Department == "Finance" and Salary > 50000')

```
# T3
import pandas as pd
data = {
    'Name': ['Alice', 'Bob', 'Charlie', 'David', 'Eve','Tom'],
    'Department': ['HR', 'IT', 'Finance', 'Marketing', 'IT','Finance'],
    'Salary': [70000, 80000, 90000, 60000, 85000, 45000]
}

employee_data = pd.DataFrame(data)

print(employee_data)
```

✓ 0.0s

```
      Name Department  Salary
0    Alice         HR   70000
1      Bob         IT   80000
2  Charlie    Finance   90000
3    David  Marketing   60000
4      Eve         IT   85000
5      Tom    Finance   45000
```

```
print('A', employee_data[(employee_data['Department'] == 'Finance') & (employee_data['Salary'] > 50000)])
```

✓ 0.0s

```
A       Name Department  Salary
2  Charlie    Finance   90000
```

```
print('B', employee_data((employee_data['Department'] == 'Finance') & (employee_data['Salary'] > 50000)))
```

⊗ 0.0s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\1888660377.py in <module>
----> 1 print('B', employee_data((employee_data['Department'] == 'Finance') & (employee_data['Salary'] > 50000)))

TypeError: 'DataFrame' object is not callable
```

```
print('C', employee_data['Department == "Finance" and Salary > 50000'])
```

⊗ 0.0s

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\Users\86182\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
```

```
print('D', employee_data('Department == "Finance" and Salary > 50000'))
```

⊗ 0.0s

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\3497107408.py in <module>
----> 1 print('D', employee_data('Department == "Finance" and Salary > 50000'))

TypeError: 'DataFrame' object is not callable
```

4. Suppose you have a DataFrame named inventory_data containing columns Product, Quantity, and Price. You need to calculate the total value of each product in inventory by multiplying the quantity of each product by its price and then adding a 10% tax on the total value. Which of the following code snippets correctly computes this?

a) inventory_data['Total_Value'] = (inventory_data['Quantity'] * inventory_data['Price']) * 1.10

b) inventory_data['Total_Value'] = (inventory_data['Quantity'*'Price']) * 1.10

c) inventory_data['Total_Value'] = (inventory_data['Quantity'] * inventory_data['Price']) * 0.10
inventory_data['Total_Value'] = inventory_data + (inventory_data['Quantity'] * inventory_data['Price'])

d) inventory_data['Total_Value'] = (inventory_data['Quantity'] * inventory_data['Price']) * 1.10
inventory_data['Total_Value'] += (inventory_data['Quantity'] * inventory_data['Price']) * 0.10

```
# T4
data = {
    'Product': ['Laptop', 'Mouse', 'Keyboard', 'Monitor', 'Printer'],
    'Quantity': [50, 150, 100, 75, 40],
    'Price': [1200, 20, 80, 250, 150]
}

# Create the DataFrame
inventory_data = pd.DataFrame(data)

# Display the DataFrame
print(inventory_data)
```

40] ✓ 0.0s

```
     Product  Quantity  Price
0     Laptop        50   1200
1      Mouse       150     20
2   Keyboard       100     80
3    Monitor        75    250
4    Printer        40    150
```

```
print('A', (inventory_data['Quantity'] * inventory_data['Price']) * 1.10 )
print('B', (inventory_data['Quantity'*'Price']) * 1.10)
```

41] ⊗ 0.0s

```
A 0    66000.0
1     3300.0
2     8800.0
3    20625.0
4     6600.0
dtype: float64

---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\2641065313.py in <module>
      1 print('A', (inventory_data['Quantity'] * inventory_data['Price']) * 1.10 )
----> 2 print('B', (inventory_data['Quantity'*'Price']) * 1.10)
      3 print('C', inventory_data + (inventory_data['Quantity'] * inventory_data['Price']))

TypeError: can't multiply sequence by non-int of type 'str'
```

```
print('C', inventory_data + (inventory_data['Quantity'] * inventory_data['Price']))
```

5] ✓ 0.0s

```
C     0   1   2   3   4  Price Product  Quantity
0 NaN NaN NaN NaN NaN    NaN     NaN       NaN
1 NaN NaN NaN NaN NaN    NaN     NaN       NaN
2 NaN NaN NaN NaN NaN    NaN     NaN       NaN
3 NaN NaN NaN NaN NaN    NaN     NaN       NaN
4 NaN NaN NaN NaN NaN    NaN     NaN       NaN
```

```
# D
inventory_data['Total_Value'] += (inventory_data['Quantity'] * inventory_data['Price']) * 0.10
inventory_data['Total_Value']
```

3] ⊗ 0.0s

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
c:\Users\86182\anaconda3\lib\site-packages\pandas\core\indexes\base.py in get_loc(self, key, method, tolerance)
   3628             try:
-> 3629                 return self._engine.get_loc(casted_key)
   3630             except KeyError as err:
```

5. Suppose you have a DataFrame named customer_data with columns Customer_ID, First_Name, Last_Name, and Email. You need to add a new column called Full_Name which concatenates the first and last names. Which of the following code snippets accomplishes this task?

a) customer_data['Full_Name'] = customer_data(row: row['First_Name'] + ' ' + row['Last_Name'], axis=1)

b) customer_data['Full_Name'] = pd.merge([customer_data['First_Name'], customer_data['Last_Name']], axis=1)

c) customer_data['Full_Name'] = customer_data['First_Name'] + ' ' + customer_data['Last_Name']

d) customer_data['Full_Name'] = ([customer_data['First_Name'], customer_data axis=1).join(axis=1)

```python
# T5
data = {
    'Customer_ID': [1, 2, 3, 4, 5],
    'First_Name': ['John', 'Jane', 'Jim', 'Jill', 'Jack'],
    'Last_Name': ['Doe', 'Doe', 'Smith', 'Jones', 'Brown'],
    'Email': [
        'john.doe@example.com',
        'jane.doe@example.com',
        'jim.smith@example.com',
        'jill.jones@example.com',
        'jack.brown@example.com'
    ]
}
customer_data = pd.DataFrame(data)
print(customer_data)
```

✓ 0.0s

	Customer_ID	First_Name	Last_Name	Email
0	1	John	Doe	john.doe@example.com
1	2	Jane	Doe	jane.doe@example.com
2	3	Jim	Smith	jim.smith@example.com
3	4	Jill	Jones	jill.jones@example.com
4	5	Jack	Brown	jack.brown@example.com

```python
print('A', customer_data(lambda row: row['First_Name'] + ' ' + row['Last_Name'], axis=1))
```

⊗ 0.0s

```
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\1411829169.py in <module>
----> 1 print('A', customer_data(lambda row: row['First_Name'] + ' ' + row['Last_Name'], axis=1))

TypeError: 'DataFrame' object is not callable
```

```python
pd.merge([customer_data['First_Name'], customer_data['Last_Name']], axis=1)
```

⊗ 0.0s

```
TypeError                                 Traceback (most recent call last)
~\AppData\Local\Temp\ipykernel_27160\2230436288.py in <module>
----> 1 pd.merge([customer_data['First_Name'], customer_data['Last_Name']], axis=1)

TypeError: merge() got an unexpected keyword argument 'axis'
```

```python
customer_data['Full_Name'] = customer_data['First_Name'] + ' ' + customer_data['Last_Name']
customer_data['Full_Name']
```

✓ 0.0s

```
0      John Doe
1      Jane Doe
2     Jim Smith
3    Jill Jones
4    Jack Brown
Name: Full_Name, dtype: object
```

```python
customer_data['Full_Name'] = ([customer_data['First_Name'], customer_data['Last_Name']], axis=1).join(axis=1)
```

⊗ 0.0s

```
  File "C:\Users\86182\AppData\Local\Temp\ipykernel_27160\4279354664.py", line 1
    customer_data['Full_Name'] = ([customer_data['First_Name'], customer_data['Last_Name']], axis=1).join(axis=1)
                                                                                                   ^

SyntaxError: invalid syntax
```