In [3]:
```
import pandas as pd
```

In [2]:
```
# create dataframe from list of lists
# DataFrame should have two columns, student_id and age
data = [
  [1, 15],
  [2, 11],
  [3, 11],
  [4, 20]
]
df = pd.DataFrame(data, columns=['student_id', 'age'])
df['name'] = ['a','b','c','d']
df
```

Out[2]:

	student_id	age	name
0	1	15	a
1	2	11	b
2	3	11	c
3	4	20	d

In [3]:
```
df.shape
```

Out[3]:
```
(4, 3)
```

In [4]:
```
df.loc[df["student_id"] == 2, ["name", "age"]]
```

Out[4]:

	name	age
1	b	11

Remove duplicate rows and keep only the first occurrence:

In [ ]:
```
+-------------+---------+---------------------+
| customer_id | name    | email               |
+-------------+---------+---------------------+
| 1           | Ella    | emily@example.com   |
| 2           | David   | michael@example.com |
| 3           | Zachary | sarah@example.com   |
| 4           | Alice   | john@example.com    |
| 5           | Finn    | john@example.com    |
| 6           | Violet  | alice@example.com   |
+-------------+---------+---------------------+

df = df.drop_duplicates(subset='email', keep='first', inplace=False, ignore_index=False)
```

In [ ]:
```
# axis=0 by row
df.dropna(axis=0, how=_NoDefault.no_default, thresh=_NoDefault.no_default, subset=None, inplace=False, ig
# how:
# ‘any’ : If any NA values are present, drop that row or column.
# ‘all’ : If all values are NA, drop that row or column.
# subset: label or list of labels
```

In [ ]:
```
# rename column names accordingly
df.rename(columns={
    'id': 'student_id',
    'first': 'first_name',
    'last': 'last_name',
    'age': 'age_in_years'
}, inplace=True)
```

In [ ]:
```
# change data type
df['age'] = df['age'].astype(int)
```

Fillna

In [ ]:
```
# fill missing data
df.fillna(value=None, method=None, axis=None, inplace=False, limit=None, downcast=_NoDefault.no_default)
# value: scalar, dict, Series, or DataFrame
# Value to use to fill holes (e.g. 0), alternately a dict/Series/DataFrame of values specifying which valu
# for each index (for a Series) or column (for a DataFrame). Values not in the dict/Series/DataFrame will

# Method to use for filling holes in reindexed Series:

# ffill: propagate last valid observation forward to next valid.
# bfill: use next valid observation to fill gap.
```

Concat

In [ ]:
```
df = pd.concat([df1,df2],axis=0) # 按行拼接
```

Pivot table

In [9]:
```
Input:
+--------------+----------+-------------+
| city         | month    | temperature |
+--------------+----------+-------------+
| Jacksonville | January  | 13          |
| Jacksonville | February | 23          |
| Jacksonville | March    | 38          |
| Jacksonville | April    | 5           |
| Jacksonville | May      | 34          |
| ElPaso       | January  | 20          |
| ElPaso       | February | 6           |
| ElPaso       | March    | 26          |
| ElPaso       | April    | 2           |
| ElPaso       | May      | 43          |
+--------------+----------+-------------+
Output:
+----------+--------+--------------+
| month    | ElPaso | Jacksonville |
+----------+--------+--------------+
| April    | 2      | 5            |
| February | 6      | 23           |
| January  | 20     | 13           |
| March    | 26     | 38           |
| May      | 43     | 34           |
+----------+--------+--------------+
import pandas as pd

df = pd.DataFrame({
    'city': [
        'Jacksonville', 'Jacksonville', 'Jacksonville', 'Jacksonville','Jacksonville',
        'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso'
        ],
    'month': [
        'January', 'February', 'March', 'April', 'May',
        'January', 'February', 'March', 'April', 'May'
        ],
    'temperature': [13, 23, 38, 5, 34, 20, 6, 26, 2, 43]
})
df
```

Out[9]:

	city	month	temperature
0	Jacksonville	January	13
1	Jacksonville	February	23
2	Jacksonville	March	38
3	Jacksonville	April	5
4	Jacksonville	May	34
5	ElPaso	January	20
6	ElPaso	February	6
7	ElPaso	March	26
8	ElPaso	April	2
9	ElPaso	May	43

In [10]:
```
df.pivot_table(values='temperature', index='month', columns='city')
```

Out[10]:

city	ElPaso	Jacksonville
month		
April	2	5
February	6	23
January	20	13
March	26	38
May	43	34

In [5]:
```
import pandas as pd

df1 = pd.DataFrame({
    'city': [
        'Jacksonville', 'Jacksonville', 'Jacksonville', 'Jacksonville','Jacksonville',
        'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso'
        ],
    'month': [
        'January', 'February', 'March', 'April', 'May',
        'January', 'February', 'March', 'April', 'May'
        ],
    'temperature': [13, 23, 38, 5, 34, 20, 6, 26, 2, 43]
})

df2 = pd.DataFrame({
    'city': [
        'Jacksonville', 'Jacksonville', 'Jacksonville', 'Jacksonville','Jacksonville',
        'Jacksonville',
        'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso', 'ElPaso'
        ],
    'month': [
        'January', 'February', 'March', 'April', 'May',
        'January',
        'January', 'February', 'March', 'April', 'May'
        ],
    'temperature': [13, 23, 38, 5, 34, 230, 20, 6, 26, 2, 43]
})

# 1. Cases for pandas.DataFrame.pivot:
# Case 1.1: no duplicates in table
df1_pivot = df1.pivot(index='month', columns='city', values='temperature')
print('Use pandas.DataFrame.pivot with table without duplicates:')
print(df1_pivot)
# Case 1.2: with duplicates in table
print('Using pandas.DataFrame.pivot with duplicates contained in the table:')
try:
    df2_pivot = df2.pivot(index='month', columns='city', values='temperature')
except ValueError:
    print('ValueError: Index contains duplicate entries, cannot reshape')
else:
```

```
    print(df2_pivot)

# 2. Cases for pandas.DataFrame.pivot_table:
# Case 2.1: no duplicates in table
df1_pivot_table = df1.pivot_table(
        values='temperature',
        index='month',
        columns='city',
        aggfunc='sum'
    )
print('Use pandas.DataFrame.pivot_table with table without duplicates:')
print(df1_pivot_table)
# Case 2.2: with duplicates in table
df2_pivot_table = df2.pivot_table(
        values='temperature',
        index='month',
        columns='city',
        aggfunc='sum'
    )
print('Use pandas.DataFrame.pivot_table with duplicates contained in the table:')
print(df2_pivot_table)
print('OMG! I never go to Jacksonville in January! 243 is too much! :)')
```

```
Use pandas.DataFrame.pivot with table without duplicates:
city      ElPaso  Jacksonville
month
April          2             5
February       6            23
January       20            13
March         26            38
May           43            34
Using pandas.DataFrame.pivot with duplicates contained in the table:
ValueError: Index contains duplicate entries, cannot reshape
Use pandas.DataFrame.pivot_table with table without duplicates:
city      ElPaso  Jacksonville
month
April          2             5
February       6            23
January       20            13
March         26            38
May           43            34
Use pandas.DataFrame.pivot_table with duplicates contained in the table:
city      ElPaso  Jacksonville
month
April          2             5
February       6            23
January       20           243
March         26            38
May           43            34
OMG! I never go to Jacksonville in January! 243 is too much! :)
```

Melt

In [ ]:

```
# melt Unpivot a DataFrame from wide to long format
Input:
+-------------+-----------+-----------+-----------+-----------+
| product     | quarter_1 | quarter_2 | quarter_3 | quarter_4 |
+-------------+-----------+-----------+-----------+-----------+
| Umbrella    | 417       | 224       | 379       | 611       |
| SleepingBag | 800       | 936       | 93        | 875       |
+-------------+-----------+-----------+-----------+-----------+
Output:
+-------------+-----------+-------+
| product     | quarter   | sales |
+-------------+-----------+-------+
| Umbrella    | quarter_1 | 417   |
| SleepingBag | quarter_1 | 800   |
| Umbrella    | quarter_2 | 224   |
| SleepingBag | quarter_2 | 936   |
| Umbrella    | quarter_3 | 379   |
| SleepingBag | quarter_3 | 93    |
| Umbrella    | quarter_4 | 611   |
| SleepingBag | quarter_4 | 875   |
+-------------+-----------+-------+
```

```
pd.melt(report, id_vars=['product'], var_name='quarter', value_name='sales')
```

Write a solution to list the names of animals that weigh strictly more than 100 kilograms.

Return the animals sorted by weight in descending order.

In [ ]:
```
Input:
DataFrame animals:
+----------+---------+-----+--------+
| name     | species | age | weight |
+----------+---------+-----+--------+
| Tatiana  | Snake   | 98  | 464    |
| Khaled   | Giraffe | 50  | 41     |
| Alex     | Leopard | 6   | 328    |
| Jonathan | Monkey  | 45  | 463    |
| Stefan   | Bear    | 100 | 50     |
| Tommy    | Panda   | 26  | 349    |
+----------+---------+-----+--------+
Output:
+----------+
| name     |
+----------+
| Tatiana  |
| Jonathan |
| Tommy    |
| Alex     |
+----------+

animals[animals['weight'] > 100].sort_values(by='weight',ascending=False)[['name']]
```

In [14]:
```
type(df[['city']])
```

Out[14]:
```
pandas.core.frame.DataFrame
```

In [15]:
```
type(df['city'])
```

Out[15]:
```
pandas.core.series.Series
```

Merge

```
pd.merge(left, right, how = 'inner', on = None, left_on = None, right_on = None,
    left_index = False, right_index = False, sort = True,
    suffixes = ('_x', '_y'), copy = True, indicator = False, validate=None)
```

参数on

Column or index level names to join on. These must be found in both DataFrames. If on is None and not merging on indexes then this defaults to the intersection of the columns in both DataFrames. 若没有相同名称的列，也可以用left_on和right_on来确定

参数how how参数控制拼接方式

how{‘left’, ‘right’, ‘outer’, ‘inner’, ‘cross’}, default ‘inner’ Type of merge to be performed.

left: use only keys from left frame, similar to a SQL left outer join; preserve key order.

right: use only keys from right frame, similar to a SQL right outer join; preserve key order.

outer: use union of keys from both frames, similar to a SQL full outer join; sort keys lexicographically.

inner: use intersection of keys from both frames, similar to a SQL inner join; preserve the order of the left keys.

cross: creates the cartesian product from both frames, preserves the order of the left keys.

In [35]:
```
customers = pd.DataFrame({'id':[1,2,3,4], 'name':['Joe','Henry','Sam','Max']})
orders = pd.DataFrame({'id':[1,2], 'customer_id':[2,4]})
```

```
customers
```

Out[35]:

	id	name
0	1	Joe
1	2	Henry
2	3	Sam
3	4	Max

In [36]:
```
orders
```

Out[36]:

	id	customer_id
0	1	2
1	2	4

In [39]:
```
df =  pd.merge(customers,orders,left_on='id',right_on='customer_id',how='left')
df
```

Out[39]:

	id_x	name	id_y	customer_id
0	1	Joe	NaN	NaN
1	2	Henry	1.0	2.0
2	3	Sam	NaN	NaN
3	4	Max	2.0	4.0

In [40]:
```
df[(df.id_y.isna())][['name']].rename(columns={'name':'Customers'})
```

Out[40]:

	Customers
0	Joe
2	Sam

In [1]:
```
##########################################################
```

In [ ]:
```
df['column'].str.len() # get length of each str in the column
```

In [3]:
```
| employee_id | name    | salary |
| ----------- | ------- | ------ |
| 2           | Meir    | 3000   |
| 3           | Michael | 3800   |
| 7           | Addilyn | 7400   |
| 8           | Juan    | 6100   |
| 9           | Kannon  | 7700   |
df = pd.DataFrame({'employee_id':[2,3,7,8,9],'name':['Meir','Michael','Addilyn','Juan','Kannon'],'salary'
df
```

Out[3]:

	employee_id	name	salary
0	2	Meir	3000
1	3	Michael	3800
2	7	Addilyn	7400
3	8	Juan	6100
4	9	Kannon	7700

df筛选: | 或 & 与 ~ 取反

In [19]:
```
df['temp'] = (df['employee_id'] % 2 == 1) & (~ df['name'].str.startswith('M'))
df
```

Out[19]:

	employee_id	name	salary	temp
0	2	Meir	3000	False
1	3	Michael	3800	False
2	7	Addilyn	7400	True
3	8	Juan	6100	False
4	9	Kannon	7700	True

In [20]:
```
df['bonus'] = df['salary'] * df['temp']
df
```

Out[20]:

	employee_id	name	salary	temp	bonus
0	2	Meir	3000	False	0
1	3	Michael	3800	False	0
2	7	Addilyn	7400	True	7400
3	8	Juan	6100	False	0
4	9	Kannon	7700	True	7700

In [25]:
```
################################################
df['name'].str.contains('M')
```

Out[25]:
```
0     True
1     True
2    False
3    False
4    False
Name: name, dtype: bool
```

Map, Apply, Applymap

In [42]:
```
import numpy as np

boolean=[True,False]
gender=["男","女"]
color=["white","black","yellow"]
data=pd.DataFrame({
    "height":np.random.randint(150,190,100),
    "weight":np.random.randint(40,90,100),
    "smoker":[boolean[x] for x in np.random.randint(0,2,100)],
    "gender":[gender[x] for x in np.random.randint(0,2,100)],
    "age":np.random.randint(15,90,100),
    "color":[color[x] for x in np.random.randint(0,len(color),100) ]
}
)
data
```

Out[42]:

	height	weight	smoker	gender	age	color
0	176	88	False	男	27	white
1	181	54	True	男	42	black
2	185	51	False	男	64	yellow
3	175	41	True	男	57	white
4	177	70	True	女	59	yellow
...	...	...	...	...	...	...
95	168	47	False	男	67	white
96	173	56	False	女	62	black
97	180	51	True	女	22	white
98	157	75	True	女	31	black
99	151	46	False	男	47	white

100 rows × 6 columns

In [ ]:
```
# gender列的男替换为1, 女替换为0

# 使用字典进行映射
data["gender"] = data["gender"].map({"男":1, "女":0})
# 使用函数
def gender_map(x):
    gender = 1 if x == "男" else 0
    return gender
#注意这里传入的是函数名, 不带括号
data["gender"] = data["gender"].map(gender_map)
```

In [ ]:
```
def apply_age(x,bias):
    return x+bias

# apply以元组的方式传入额外的参数
data["age"] = data["age"].apply(apply_age,args=(-3,))
```

In [45]:
```
# 当沿着轴0（axis=0）进行操作时, 会将各列(columns)默认以Series的形式作为参数,
# 传入到你指定的操作函数中, 操作后合并并返回相应的结果。
# 沿着0轴求和
data[["height","weight","age"]].apply(np.sum, axis=0)

# 沿着0轴取对数
data[["height","weight","age"]].apply(np.log, axis=0)
```

Out[45]:

	height	weight	age
0	5.170484	4.477337	3.295837
1	5.198497	3.988984	3.737670
2	5.220356	3.931826	4.158883
3	5.164786	3.713572	4.043051
4	5.176150	4.248495	4.077537
...	...	...	...
95	5.123964	3.850148	4.204693
96	5.153292	4.025352	4.127134
97	5.192957	3.931826	3.091042
98	5.056246	4.317488	3.433987
99	5.017280	3.828641	3.850148

100 rows × 3 columns

In [ ]:
```
def BMI(series):
    weight = series["weight"]
    height = series["height"]/100
    BMI = weight/height**2
    return BMI

data["BMI"] = data.apply(BMI, axis=1)
```

applymap的用法比较简单，会对DataFrame中的每个单元格执行指定函数的操作

Groupby

In [46]:
```
company=["A","B","C"]
data=pd.DataFrame({
    "company":[company[x] for x in np.random.randint(0,len(company),10)],
    "salary":np.random.randint(5,50,10),
    "age":np.random.randint(15,50,10)
}
)
data
```

Out[46]:

	company	salary	age
0	C	35	35
1	C	49	48
2	C	20	21
3	A	49	18
4	B	49	48
5	B	26	26
6	A	41	20
7	A	29	20
8	B	40	32
9	C	46	40

In [47]:
```
list(data.groupby("company"))
```

Out[47]:
```
[('A',
    company  salary  age
  3       A      49   18
  6       A      41   20
  7       A      29   20),
 ('B',
    company  salary  age
  4       B      49   48
  5       B      26   26
  8       B      40   32),
 ('C',
    company  salary  age
  0       C      35   35
  1       C      49   48
  2       C      20   21
  9       C      46   40)]
```

agg 聚合操作

In [48]:
```
data.groupby("company").max()
data.groupby('company').agg({'salary':'median','age':'mean'})
```

Out[48]:

company	salary	age
A	49	20
B	49	48
C	49	48

In [ ]:
```
Input:
Activities table:
+------------+------------+
| sell_date  | product     |
+------------+------------+
| 2020-05-30 | Headphone  |
| 2020-06-01 | Pencil     |
| 2020-06-02 | Mask       |
| 2020-05-30 | Basketball |
| 2020-06-01 | Bible      |
| 2020-06-02 | Mask       |
| 2020-05-30 | T-Shirt    |
+------------+------------+
Output:
+------------+----------+------------------------------+
| sell_date  | num_sold | products                     |
+------------+----------+------------------------------+
| 2020-05-30 | 3        | Basketball,Headphone,T-shirt |
| 2020-06-01 | 2        | Bible,Pencil                 |
| 2020-06-02 | 1        | Mask                         |
+------------+----------+------------------------------+
```

In [ ]:
```
activities.groupby('sell_date')['product'].agg(
    [('num_sold', 'nunique'), ('products', lambda x: ','.join(sorted(x.unique())))]
).reset_index()
```

transform

In [ ]:
```
# 求各个公司平均工资并加入原df

# 不用transform:
salary_dict = data.groupby('company')['salary'].mean().to_dict()
data['avg_salary'] = data['company'].map(avg_salary_dict)
# 用transform:
data['avg_salary'] = data.groupby('company')['salary'].transform('mean')
```

apply

In [49]:
```
# 获取各个公司年龄最大的员工的数据

def get_oldest_staff(x):
    df = x.sort_values(by = 'age',ascending=False)
    return df.iloc[0,:]
data.groupby('company',as_index=False).apply(get_oldest_staff)
```

Out[49]:

	company	salary	age
0	A	41	20
1	B	49	48
2	C	49	48

In [ ]:
```
## 例子
Employee table:
+----+-------+--------+--------------+
| id | name  | salary | departmentId |
+----+-------+--------+--------------+
| 1  | Joe   | 70000  | 1            |
| 2  | Jim   | 90000  | 1            |
| 3  | Henry | 80000  | 2            |
```

```
| 4  | Sam   | 60000  | 2            |
| 5  | Max   | 90000  | 1            |
+----+-------+--------+--------------+
Department table:
+----+-------+
| id | name  |
+----+-------+
| 1  | IT    |
| 2  | Sales |
+----+-------+
# 取出每个部门工资最高的，结果为:
+------------+----------+--------+
| Department | Employee | Salary |
+------------+----------+--------+
| IT         | Jim      | 90000  |
| Sales      | Henry    | 80000  |
| IT         | Max      | 90000  |
+------------+----------+--------+
```

In [55]:
```
employee = pd.DataFrame({'id':[1,2,3,4,5],'name':['Joe','Jim','Henry','Sam','Max'],'salary':[70000,90000,8
department = pd.DataFrame({'id':[1,2],'name':['IT','Sales']})
temp = pd.merge(left=employee,right=department,how='left',left_on='departmentId',right_on='id')[['name_x'
temp.rename(columns={'name_x':'Employee','name_y':'Department','salary':'Salary'},inplace=True)
temp
```

Out[55]:

	Employee	Salary	Department
0	Joe	70000	IT
1	Jim	90000	IT
2	Henry	80000	Sales
3	Sam	60000	Sales
4	Max	90000	IT

In [59]:
```
temp[temp['Salary'] == temp.groupby('Department')['Salary'].transform(max)]
```

Out[59]:

	Employee	Salary	Department
1	Jim	90000	IT
2	Henry	80000	Sales
4	Max	90000	IT

Rank

```
DataFrame.rank(axis=0, method='average', numeric_only=False, na_option='keep', ascending=True,
pct=False)
```

method{‘average’, ‘min’, ‘max’, ‘first’, ‘dense’}, default ‘average’ How to rank the group of records that have the same value (i.e. ties):

average: average rank of the group

min: lowest rank in the group

max: highest rank in the group

first: ranks assigned in order they appear in the array

dense: like ‘min’, but rank always increases by 1 between groups.

In [ ]:
```
############################################################
```

In [ ]:
```
Input:
Employees table:
+--------+-----------+---------+----------+
```

```
| emp_id | event_day   | in_time | out_time |
+--------+-------------+---------+----------+
| 1      | 2020-11-28 | 4       | 32       |
| 1      | 2020-11-28 | 55      | 200      |
| 1      | 2020-12-03 | 1       | 42       |
| 2      | 2020-11-28 | 3       | 33       |
| 2      | 2020-12-09 | 47      | 74       |
+--------+-------------+---------+----------+
Output:
sum of out_time - in_time
+------------+--------+------------+
| day        | emp_id | total_time |
+------------+--------+------------+
| 2020-11-28 | 1      | 173        |
| 2020-11-28 | 2      | 30         |
| 2020-12-03 | 1      | 41         |
| 2020-12-09 | 2      | 27         |
+------------+--------+------------+
```

In [5]:
```
df = pd.DataFrame({'emp_id':[1,1,1,2,2],'event_day':['2020-11-28','2020-11-28','2020-12-03','2020-11-28',
                   'in_time':[4,55,1,3,47],'out_time':[32,200,42,33,74]})
df
```

Out[5]:

	emp_id	event_day	in_time	out_time
0	1	2020-11-28	4	32
1	1	2020-11-28	55	200
2	1	2020-12-03	1	42
3	2	2020-11-28	3	33
4	2	2020-12-09	47	74

In [15]:
```
df['time'] = df['out_time'] - df['in_time']
df.groupby(['emp_id','event_day'])['time'].agg(sum).reset_index()
```

Out[15]:

	emp_id	event_day	time
0	1	2020-11-28	173
1	1	2020-12-03	41
2	2	2020-11-28	30
3	2	2020-12-09	27

In [ ]:
```
# Count number of distinct elements in specified axis.
DataFrame.nunique(axis=0, dropna=True)
```

In [ ]:
```
DataFrame['column'].mode().to_frame()
```