> pandas是python中一个数据处理和科学分析的工具，它能做的很多，不过最为强大的就是数据处理。由于它可以很轻松的处理excel数据，所以，也是自动化办公的一个神器，也是我学习的主要目标。

安装

使用pip来安装pandas，pip需要先配置国内源，不然下载速度很慢

```
pip install pandas
```

处理excel数据，还需要一个插件（openpyxl）来读取/写入excel

```
# openpyxl (3.0.3)    - A Python library to read/write Excel 2010 xlsx/xlsm files
pip install openpyxl
```

三种数据结构

数据结构	维度	特点
Series	一维	同构（数据类型相同），表示一列column
DataFrame	二维	异构，表示一个表格table
Panel	三维	异构，使用的少

由于具有DataFrame这种二维数据结构，所以可以处理很多类似数据结构的文件，比如：excel，sql，json等

简单读取写入

读取数据的时候可以直接指定index列 `index_col=`

数据如果太长显示不了，可以设置显示宽度 `pandas.options.display.max_columns=num`

excel

```
import pandas as pd

# 写入数据到excel
data = {"name":["韩信","李白","杜甫"], "age":[21,22,24], "score":[80,90,100]}
df = pd.DataFrame(data)
df.to_excel('/home/narcissus/Desktop/1.xlsx')

# 从excel读取数据
pe = pd.read_excel("/home/narcissus/Desktop/1.xlsx", index_col='name')
print(pe)
```

mysql

需要安装数据库连接驱动

```
pip install pymysql
```

```
import pandas
import pymysql

host = '192.168.0.105'
user = 'pi'
password = input("password for mysql connection: ")
database = 'mytestdb'
con = pymysql.connect(host,user,password,database)

# 读取数据
df = pd.read_sql_query('select * from ssacount limit 0,10',con)
print(df)

# 可以将数据库中读取到的数据直接写入excel中
df.to_excel('~/Desktop/1.xlsx')
```

sqlite

```
import sqlite3

con = sqlite3.connect("testdatabase.db")
df = pd.read_sql_query("SELECT * FROM ssacount", con)
print(df)
```

json

```
df = pd.read_json('test.json')
```

csv/tsv/txt

pandas只有一个 `read_csv()` 函数，但是却可以读取这三种文件（文本文件都可以），默认读取csv文件，其分隔符为逗号，所以在读取tsv和txt的时候，要手动设置分隔符 `sep=''`

```
df = pd.read_csv('1.csv')
df = pd.read_csv('1.tsv', sep='\t')
df = pd.read_csv('1.txt', sep='文件中的特殊分隔符号')
```

写入

值得注意的是在写入文件之前要确保去掉默认的index，需要设置成自己指定的index，否则再读取文件时，会有一个unamed的列（之前的index列）。其实可以用后面学习的 `skiprows` 在读取时指定跳过多少行，以及 `usecols` 来跳过指定列

```
import pandas as pd

data = {"name":["韩信",None,"李白","杜甫"], "age":[21,None,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
# 写入文件之前需要去掉默认的index
df.set_index('name', inplace=True)
df.to_excel('~/Desktop/1.xlsx')
# 读取数据时指定index
df = pd.read_excel('~/Desktop/1.xlsx', index_col='name')
print(df)
```

```
df.to_excel('~/Desktop/1.xlsx')

df.to_json('~/Desktop/test.json')

df.to_sql('ssacount', con)
```

PS: 按照目前我的理解，pandas可以完美的完成这些数据文件之间的转换。可以将json，数据库中的数据导入到excel中进行可视化操作。

DataFrame

作为最重要的一个部分，DataFrame包含了几百种方法和操作，可以对数据进行任何的分析与处理。

创建

创建的方式比较多，但通常都是使用 `dict` 来创建。

```
import pandas as pd

data = {
        "name":["韩信","李白","杜甫"],
        "age":[21,22,24],
        "score":[80,90,100]
       }
df = pd.DataFrame(data)
print(df)
```

每一对key-value作为一个Series(column)，key作为表头，value作为数据

自己指定index

默认会自动创建一个index列，为从0开始的连续数据。可以自己指定某列为索引列

```
# 使用set_index()方法
df = df.set_index("name")

# 或者在DataFrame中指定index
df = pd.DataFrame(data, index=['一班','二班','二班'])
```

需要注意的是这个index的数据是可以重复的，并不需要唯一性

通过index找出对应数据

可以通过 `loc` 找出来的条件是设置了index，否则无法找到

```
sear = df.loc['一班']
print(sear)

name     韩信
age      21
score    80
Name: 一班, dtype: object
```

可以看出index与其对应数据也是一个 `dict` 数据类型

即使设置了index，也可以通过默认的index来查找 `iloc`

```
sear = df.iloc[1]
```

指定header

默认header=0,会使用第0行（index为空）/excel中的第一行作为header，即columns

```
df = pd.read_excel('~/Desktop/1.xlsx', header=1)
```

重新自定义header，能够自定义成功的前提是这个文件本身就没有header，否则原来的header会作为一个数据行成为index=0的那一行

```
df = pd.read_excel('~/Desktop/1.xlsx', header=None)
# 取指定列
df = df[[1,3]]
# 设置header
df.columns = [1,2]
```

查看开头和结尾

```
# 默认为5行
df.head()
df.tail()

# 可以自己指定显示多少行
df.head(20)
df.tail(20)
```

查看数据信息

`.info()` 提供了很重要的一些信息，包括行，列，数据是否为空，数据类型，内存占用等

```
import pandas as pd

data = {"name":["韩信","李白","杜甫"], "age":[21,22,24], "score":[80,80,100]}
df = pd.DataFrame(data)
info = df.info()
print(info)

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3 entries, 0 to 2
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   name    3 non-null      object
 1   age     3 non-null      int64
 2   score   3 non-null      int64
dtypes: int64(2), object(1)
memory usage: 200.0+ bytes
None
```

查看行列数

`.shape` 提供了一个简单快速查看总行数，列数的信息(rows,columns)

```
info = df.shape
print(info)

(3,3)
```

添加与去重

添加DataFrame

- df.append()
- pandas.concat([])

`.append()` 可以将已有的DataFrame追加。但是index不会自动变化，要是index自动变化，需要调用 `reset_index(drop=True)`

append如果需要将Series追加到行后面，必须加一个参数 `ignore_index=True`

```
import pandas as pd
import pymysql

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
tmp_df = df.append(df)
print(tmp_df)

    name  age  score
0   韩信   21     80
1   韩信   21     80
2   李白   22     80
3   杜甫   24    100
0   韩信   21     80
1   韩信   21     80
2   李白   22     80
3   杜甫   24    100
```

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
tmp_df = df.append(df).reset_index(drop=True)
print(tmp_df)

    name  age  score
0   韩信   21     80
1   韩信   21     80
2   李白   22     80
3   杜甫   24    100
4   韩信   21     80
5   韩信   21     80
6   李白   22     80
7   杜甫   24    100
```

使用concat默认y轴添加，使用axis=1则x轴添加

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
tmp_df = pd.concat([df,df], axis=0).reset_index(drop=True)
print(tmp_df)
```

添加Series

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
add = pd.Series({'name':'筱倩', 'age':30, 'score':88})
df = df.append(add, ignore_index=True)
print(df)

    name  age  score
0   韩信   21     80
1   韩信   21     80
2   李白   22     80
3   杜甫   24    100
4   筱倩   30     88
```

添加列

```
df['column_name'] = []
tmp_df['add'] = [i for i in range(max(tmp_df.index)+1)]
```

找到重复数据

`df.dumpllicateed(subset='column_name')` 可以找出指定列的重复index，和True/False的Series。那么如何通过index来找到真正的重复数据？可以通过循环，其实更加简单的方式是使用 `df.iloc[index]` 定位

```
import pandas as pd
import pymysql

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
dum = df.duplicated(subset='name')
# 查看是否存在重复数据
print(dum.any())
dum = dum[dum == True] #可以简化为dum = dum[dum]
```

```
info = df.iloc[dum.index]
print(info)

True
    name  age  score
1   韩信   21     80
```

去重

`.drop_duplicates()` 可以将追加的重复数据清除，需要加上 `inplace=True` 这个参数，否则不会生效。

- 默认会对整个表的数据进行去重，如果需要对指定列，那么添加参数 `subset='column_name'`

```
tmp_df.drop_duplicates(subset='score', inplace=True)

    name  age  score
0   韩信   21     80
2   李白   22     80
3   杜甫   24    100
```

`.drop_duplicates()` 的 `keep` 参数：

- first，保留第一次出现的，默认
- last，保留最后一次出现的
- False，去除所有重复数据

```
tmp_df.drop_duplicates(inplace=True, keep='first')

    name  age  score
0   韩信   21     80
2   李白   22     80
3   杜甫   24    100
```

```
tmp_df.drop_duplicates(inplace=True, keep='last')

    name  age  score
1   韩信   21     80
2   李白   22     80
3   杜甫   24    100
```

```
tmp_df.drop_duplicates(inplace=True, keep=False)

Empty DataFrame
Columns: [name, age, score]
Index: []
```

修改

通过df['column_name']/df.column_name可以获取Series，通过Series[index]可以获取到指定的值

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
df.name[1] = '筱倩'
df.age[1] = 99
print(df)

    name  age  score
0   韩信   21     80
1   筱倩   99     80
2   李白   22     80
3   杜甫   24    100
```

替换

使用到了 `df.iloc[index]` 定位，并用Series进行替换

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
alert = pd.Series({'name':'筱倩', 'age':30, 'score':88})
df.iloc[1] = alert
print(df)
```

插入

行

使用到了df[]切片以及追加操作。值得注意的是：在python中可以对list进行切片，但是在pandas中还可以对DataFrame这种二维数据进行切片

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
insert = pd.Series({'name':'筱倩', 'age':30, 'score':88})
part1 = df[:2]
part2 = df[2:]
df = part1.append(insert, ignore_index=True).append(part2).reset_index(drop=True)
print(df)

    name  age  score
0   韩信   21     80
1   韩信   21     80
2   筱倩   30     88
3   李白   22     80
```

```
4   杜甫   24    100
```

列

```
# num为想插入到第几列前面
df.insert(num, column='column_name', value=[])
```

删除

行

两种方法：

- 使用到了 `df.drop(index=[], inplace=True)`
- 可以使用切片，将需要的数据切出来然后append组合

```
import pandas as pd

data = {"name":["韩信","韩信","李白","杜甫"], "age":[21,21,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)
insert = pd.Series({'name':'筱倩', 'age':30, 'score':88})
part1 = df[:1]
part2 = df[2:]
# index后面可以跟一个list集合，也可以是一个range()函数
df.drop(index=2, inplace=True)
# 使用切片然后组合
# df = part1.append(part2).reset_index(drop=True)
print(df)
```

列

```
df.drop(columns=[],inplace=True)
```

columns

`.columns` 可获取所有列名，返回的是一个列表

```
print(df.columns)

Index(['name', 'age', 'score'], dtype='object')
```

`.rename()` 可修改列名

```
df.rename(columns={'name':'Name', 'age':'AGE'}, inplace=True)
```

`col.lower()` 搭配列表生成式可以转换小写

```
df.columns = [col.lower() for col in df.columns]
```

Null数据行清除

`.isnull()` 找出null数据，null数据显示True，非空数据显示False

```
print(df.isnull())
```

`.sum()` 计算总的null数据

```
print(df.isnull().sum())
```

`.dropna()` 删除有null数据的行，只要有一个null数据就会清除

```
df.dropna(inplace=True)
```

可以看到这种清除数据的缺点，无法对指定列的Null数据进行清除，所以需要使用到过滤

```
import pandas as pd

data = {"name":["韩信",None,"李白","杜甫"], "age":[21,21,None,24], "score":
[80,80,80,100]}
# 为了能够找到手动输入的None数据列，对其进行了转换
df = pd.DataFrame(data).fillna(0)

miss = df.loc[df.name == 0 ].index
df.drop(index=miss, inplace=True)
print(df)

    name   age  score
0   韩信  21.0     80
2   李白   0.0     80
3   杜甫  24.0    100
```

Null数据列的清除

`.dropna(axis=1)` 使用了一个参数axis

```
df.dropna(inplace=True, axis=1)
```

为什么是1？在我们使用df.shape的时候返回一个tuple，(rows,columns)，可以看出columns的索引为1

使用mean来填充null数据列

`.fillna(column_name, inplace=True)` 如果将有null的数据列都删除的话， 那么数据的缺失会比较大，可以用平均值来填充

```
import pandas as pd

data = {"name":["韩信",None,"李白","杜甫"], "age":[21,None,22,24], "score":
[80,80,80,100]}
df = pd.DataFrame(data)

# 取出age列
age = df['age']
# 取平均值
age_mean = age.mean()
# 将age列中的null用平均值填充
age.fillna(age_mean, inplace=True)
print(df)
```

获取数据描述

前面用 `.info()` 获取了对DataFrame的描述，但是 `.describe()` 可以获取关于数据的描述：包括最大值，最小值，平均值，列数，行数等。既可以是整个DataFrame，也可以是某一列。

```
# 整个表
print(df.describe())
# age列
print(df['age'].describe())
```

查看某列数据相同数据的个数

`.value_counts()` 指定列的数据出现个数

```
print(df['score'].value_counts())
```

重组DataFrame

`type(df['age'])` 的类型是一个serise，而 `type(df[['age']])` 的类型是一个DataFrame，所以可以使用这种方式来重新选取需要的列构成新的DataFrame

```
new_df = df[['name', 'age']]
print(new_df)
```

数据的过滤

在Series的基础上过滤

用条件判断来过滤数据：`> < == !=` 等

```
info = (df['score'] == 80)
print(info)

# 可以得到true of false的返回值
0     True
1     True
2     True
3    False
```

如果想要直接得到过滤后的数据

```
info = df[df['score'] == 80]
print(info)

   name   age  score
0    韩信  21.0     80
1  None   NaN     80
2    李白  22.0     80
```

还可以使用逻辑判断

```
info = df[(df['score'] > 80) | (df['age'] == 22)]
```

使用 `isin()` 更加简洁

```
info = df[df['age'].isin([21,22])]
```

上面的所有方法都很简洁，但是数据量比较大的时候，效率就比较低

所以我们需要使用函数来进行过滤，使用 `.apply()` 将column传递给函数

```
def filter_score(x):
    if x == 100:
        return True
    else:
        return False
info = df.loc[df['score'].apply(filter_score)]
# 简化
info = df[df['score'].apply(filter_score)]
# 简化，其实我不理解为什么可以这么写
info = df[df.score.apply(filter_score)]
```

可以使用匿名函数简化

```
info = df[df['score'].apply(lambda x : x == 100)]
# 再简化
info = df[df.score.apply(lambda x : x == 100)]
```

在DataFrame基础上过滤

`axis=1` 表示每一行（x轴），`axis=0` 表示每一列（y轴）

```
def filter_score(x):
    if x.score == 100:
        return True
    else:
        return False

info = df.apply(filter_score,  axis=1)
print(info)
```

从指定行或列开始读取

可能表格并不是从第一行第一列开始写的，所以需要指定从哪里开始读取，否则会出现许多的unamed row和Nan值

```
import pandas as pd

df = pd.read_excel('~/Desktop/1.xlsx')
print(df)

# 这里包含了一个空行，和一个index列，读取的时候会出现这些问题
Unnamed: 0 Unnamed: 1 Unnamed: 2 Unnamed: 3
0          NaN       name         age      score
1          0.0         韩信          21         80
2          1.0         韩信          22         80
3          2.0         李白          22         80
4          3.0         杜甫          24        100
```

使用 `skiprows(num)` 和 `usecols=[]` 来选取需要的数据

```
import pandas as pd

df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3])
# 或者使用列名指定
df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=['name','age','score'])
print(df)

# 这样就能读取需要的数据了
    name  age  score
0   韩信   21     80
1   韩信   22     80
2   李白   22     80
3   杜甫   24    100
```

自动填充功能

用python来实现excel中拖拽自动填充的功能，生成序列数据进行填充，比较复杂的是日期的处理

- 连续数据：list赋值
- 交叉数据：三目运算
- 日期数据：自定义函数

```
```

```python
import pandas as pd
from datetime import date,timedelta

data = {'name':['李白','杜甫','白居易','王维'], 'age':[None,None,None,None], 'birthday':
[None,None,None,None], 'drink':[None,None,None,None]}
df = pd.DataFrame(data)
print(df)

==========================================================================================
=
    name   age birthday drink
0   李白  None     None  None
1   杜甫  None     None  None
2  白居易  None     None  None
3   王维  None     None  None
==========================================================================================
=

# 初始化日期
start = date(1600,1,31)
# 定义日期处理函数，传一个开始日期以及添加的月份
def add_month(start_date, add_month):
    tmonth = start_date.month + add_month
    cyear = start_date.year + tmonth // 12
    cmonth = 12 if tmonth % 12  == 0 else tmonth % 12
    cday = start_date.day
    # 由于不同月份的天数不同，所以需要考虑
    if cday > 28:
        if cmonth == 2:
            cday = 29 if cyear % 4 == 0 and cyear % 100 != 0 or cyear % 400 == 0 else
28
        elif cday == 31 and cmonth in [1,3,5,7,8,10,12]:
            cday = 31
        elif cday == 31 and cmonth in [2,4,6,9,11]:
            cday = 30
    return date(cyear,cmonth,cday)

# 对age进行填充，使用到了list的赋值
df['age'] = [i+20 for i in range(df.shape[1])]
# 使用循环index，来对每一项进行赋值
for i in df.index:
    # 用到了三目运算，实现交叉数据填充
    df['drink'][i] = 'like' if i % 2 == 0 else 'hate'

    # day递增的情况：day的相加有对应的函数
    df['birthday'][i] = start + timedelta(days=i)
    # year递增的情况：year的相加可以在year上直接添加
    df['birthday'][i] = date(start.year+i, start.month, start.day)
    # month递增的情况：比较复杂，涉及到year与month进位的问题，调用自定义函数实现
    df['birthday'][i] = add_month(start,i+10)

print(df)

==========================================================================================
=
    name  age    birthday drink
0   李白   20  1600-11-30  like
1   杜甫   21  1601-12-31  hate
```

```
2  白居易   22  1601-01-31  like
3   王维   23  1601-02-28  hate
================================================================================
=
```

计算功能

x轴向乘法

使用到了pandas特有的list相乘，以及list展开相乘（这在python语法中是没有的，乘也是list的复制）

```
sr1 = pd.Series([21,18,22,24])
sr2 = pd.Series([80,80,80,100])
df = pd.DataFrame({'A':sr1, 'B':sr2})
print(df)

# pandas的list乘
df['C'] = df['A'] * df['B']
df['D'] = df['C'] * 100
# 可以使用apply()
df['E'] = df['B'].apply(lambda x:x+2)

print(df)
```

```
A    B
0  21   80
1  18   80
2  22   80
3  24  100
    A    B     C       D    E
0  21   80  1680  168000   82
1  18   80  1440  144000   82
2  22   80  1760  176000   82
3  24  100  2400  240000  102
```

x轴向求和/平均

前面都是使用的Series来进行计算的，如果需要的不止一列的数据呢？比如说需要每一行的和和平均值，当然也可以将所有列写出来计算，但是有点麻烦，其实可以指定轴。它会自动过滤掉文字列，只计算数字列。(索引不会计算)

`axis=1` 指定x轴，按行计算，从上到下

```
import pandas as pd
df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')

table = df1.join(df2, how='left').fillna(0)
table['qq'] = table['qq'].astype(int)

table['sum'] = table.sum(axis=1)
table['mean'] = table.mean(axis=1).astype(int)
print(table)
```

```
    name  age  score student          qq         sum        mean
0   韩信   21     80      韩信   124242423   124242524    62121262
1   赵云   22     10      赵云    12342356    12342388     6171194
2   李白   22     30      李白   574534563   574534615   287267307
3   杜甫   24    100      杜甫     6785842     6785966     3392983
4   曹操   18     75      曹操    34563342    34563435    17281717
5   姜尚   45     75      姜尚    57845262    57845382    28922691
6   阿狗   23     11      阿狗    45645834    45645868    22822934
7   筱倩    3     99      筱倩    58364743    58364845    29182422
8   夫子  123    107      夫子  4824854834  4824855064  2412427532
9   葛聂   33     90       0           0         123          61
```

y轴向求和/平均并添加到DataFrame

- 必须设置 `ignore_index=True` ，否则无法添加

```
import pandas as pd
df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')

table = df1.join(df2, how='left').fillna(0)
table['qq'] = table['qq'].astype(int)

# 这里省略了axis=0,因为这是默认值
total = table.mean()
sum = table.sum()
# total为一个Series，可以设置name值
total['name'] = '平均值'
sum['name'] = '总和'
# 将Nan转为空
table = table.append([total,sum], ignore_index=True).fillna('')
print(table)

name    age  score student           qq
0    韩信   21.0   80.0      韩信  1.242424e+08
1    赵云   22.0   10.0      赵云  1.234236e+07
2    李白   22.0   30.0      李白  5.745346e+08
3    杜甫   24.0  100.0      杜甫  6.785842e+06
4    曹操   18.0   75.0      曹操  3.456334e+07
5    姜尚   45.0   75.0      姜尚  5.784526e+07
6    阿狗   23.0   11.0      阿狗  4.564583e+07
7    筱倩    3.0   99.0      筱倩  5.836474e+07
8    夫子  123.0  107.0      夫子  4.824855e+09
9    葛聂   33.0   90.0       0  0.000000e+00
10  平均值   33.4   67.7          5.739179e+08
11   总和  334.0  677.0          5.739179e+09
```

排序

`.sort_values(by=[],inplace=True,ascending=[False])`

默认生成一个新的DataFrame，使用inplace=True在原DataFrame上修改

默认使用升序排列，用ascending=False来降序排列

1. 对一列数据进行排序

```
import pandas as pd

sr2 = pd.Series([21,18,22,22,24])
sr3 = pd.Series([80,80,60,80,100])
df = pd.DataFrame({'A':sr2, 'B':sr3})
df.set_index('A', inplace=True)
df.sort_values(by='A', inplace=True, ascending=False)
print(df)
```

2. 对多列数据进行排序，而且第一列倒序，第二列升序

```
import pandas as pd

sr2 = pd.Series([21,18,22,22,24])
sr3 = pd.Series([80,80,60,80,100])
df = pd.DataFrame({'A':sr2, 'B':sr3})
df.set_index('A', inplace=True)
df.sort_values(by=['A','B'], inplace=True, ascending=[False,True])
print(df)
```

数据相关性

之前在一篇文章中看到 `df.corr()` 时，没看懂它到底什么意思，有什么作用。今天看了老师讲的视频，才发现这个东西超级无敌厉害。它就是分析一个表中每两列数据之间的相关性的。

- 1.0代表完全相关
- 比例越大，相关性越大

```
import pandas as pd

sr2 = pd.Series([21,18,22,22,24])
sr3 = pd.Series([80,80,60,80,100])
df = pd.DataFrame({'age':sr2, 'score':sr3})
df.corr()


# 可以看出score与age之间的相关性比较小，仅有0.3
            age score
age     1.000000    0.322749
score   0.322749    1.000000
```

从不同sheet页读取

```
import pandas as pd
df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')
```

```
print(df1)
print(df2)
# table = df1.merge(df2, on='name')
# print(table)
    name  age  score
0   韩信   21     80
1   赵云   22     10
2   李白   22     30
3   杜甫   24    100
4   曹操   18     75
5   姜尚   45     75
6   阿狗   23     11
7   筱倩    3     99
8   夫子  123    107
9   葛聂   33     90
  name          qq
0   韩信   124242423
1   赵云    12342356
2   李白   574534563
3   杜甫     6785842
4   曹操    34563342
5   姜尚    57845262
6   阿狗    45645834
7   筱倩    58364743
8   夫子  4824854834
```

多表联合查询

用到了与数据库类似的操作，merge两个DataFrame，默认为inner join,可以设置left/right join

- `df.merge()` 需要指定关联的列，具有内连接，左/右连接
- `df.join()` 默认使用index作为关联的列，不具有内连接，左/右连接
- `pandas.concat([],axis=num)`

```
import pandas as pd
df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')

table = df1.merge(df2, on='name')
print(table)

    name  age  score          qq
0   韩信   21     80   124242423
1   赵云   22     10    12342356
2   李白   22     30   574534563
3   杜甫   24    100     6785842
4   曹操   18     75    34563342
5   姜尚   45     75    57845262
6   阿狗   23     11    45645834
7   筱倩    3     99    58364743
8   夫子  123    107  4824854834
```

可以发现默认是使用的inner join，要df1的数据全部显示出来，使用left join方式

```
table = df1.merge(df2, how='left', on='name')
```

如果要关联的两张表的columns名字不同，那么不能使用 `on` ，而需要使用 `left_on='',right_on=''` 来连接

```
    name  age  score
0   韩信   21     80
1   赵云   22     10
2   李白   22     30
3   杜甫   24    100
4   曹操   18     75
5   姜尚   45     75
6   阿狗   23     11
7   筱倩    3     99
8   夫子  123    107
9   葛聂   33     90

  student          qq
0      韩信   124242423
1      赵云    12342356
2      李白   574534563
3      杜甫     6785842
4      曹操    34563342
5      姜尚    57845262
6      阿狗    45645834
7      筱倩    58364743
8      夫子  4824854834
```

关联之后数据会出现Nan值，使用 `fillna()` 函数来填充null

如果显示的是小数或者科学计数，可以使用 `astype()` 设置为数据类型

```
import pandas as pd

df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')

table = df1.merge(df2, how='left', left_on='name', right_on='student').fillna(0)
table['qq'] = table['qq'].astype(int)
print(table)

    name  age  score student          qq
0   韩信   21     80      韩信   124242423
1   赵云   22     10      赵云    12342356
2   李白   22     30      李白   574534563
3   杜甫   24    100      杜甫     6785842
4   曹操   18     75      曹操    34563342
5   姜尚   45     75      姜尚    57845262
6   阿狗   23     11      阿狗    45645834
7   筱倩    3     99      筱倩    58364743
8   夫子  123    107      夫子  4824854834
9   葛聂   33     90       0           0
```

使用 `join()` 函数

```
import pandas as pd
df1 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet1')
df2 = pd.read_excel('~/Desktop/1.xlsx', sheet_name='Sheet2')
```

```
table = df1.join(df2, how='left').fillna(0)
table['qq'] = table['qq'].astype(int)
print(table)

    name  age  score student          qq
0   韩信   21     80      韩信   124242423
1   赵云   22     10      赵云    12342356
2   李白   22     30      李白   574534563
3   杜甫   24    100      杜甫     6785842
4   曹操   18     75      曹操    34563342
5   姜尚   45     75      姜尚    57845262
6   阿狗   23     11      阿狗    45645834
7   筱倩    3     99      筱倩    58364743
8   夫子  123    107      夫子  4824854834
9   葛聂   33     90       0           0
```

数据列分割

使用到了 `Series.str.split()` 方法，另外 `Series.str` 有很多方法可以用，可以对字符串进行各种操作。

```
import pandas as pd

data = {'A':["hello joyce" for i in range(1,8)]}
df = pd.DataFrame(data)
# 不使用expand，那么数据会作为一个list
info = df.A.str.split(expand=True)
df['first'] = info[0]
df['last'] = info[1]
print(df)
```

行列转换

`df.transpose()` 即可，为了避免旋转后第一行为index，所以在读取的时候需要指定index

```
import pandas as pd

df = pd.read_excel('~/Desktop/1.xlsx', index_col='name')
df = df.transpose()
print(df)
```

提取日期

```
pandas.DatetimeIndex(df['column_name']).year
pandas.DatetimeIndex(df['column_name']).month
pandas.DatetimeIndex(df['column_name']).day
```

分组，聚合

模拟透视表功能

```
import pandas as pd
import numpy as np

df.pivot_table(index='column_name', columns='column_name', value='column_name',
aggfunc=np.sum)
```

分组

```
groups = df.groupby('column_name')
```

聚合

```
S = groups['column_name'].sum()
```

条件颜色

```
import pandas as pd

def colorlize(x):
    colors = 'orange' if x < 60 else 'lime'
    # 这里返回的格式必须要是这样
    return f'color:{colors}'

def another(x):
    # 这里需要返回一个list
    return ['color:red' if v > 90 else 'background-color:black' for v in x]

df = pd.read_excel('~/Desktop/1.xlsx')

# 可直接添加在后面
# df.style.apply(another, subset=['age','score'])
df.style.applymap(colorlize, subset=['age','score']).apply(another, subset=
['age','score'])
```

	name	age	score
0	韩信	21	80
1	赵云	22	10
2	李白	22	30
3	杜甫	24	100
4	曹操	18	75
5	姜尚	45	75
6	阿狗	23	11
7	筱倩	3	99
8	夫子	123	107
9	葛聂	33	90

需要注意的：

- `df.style.applymap()` 获取到的是每一个值
- `df.style.apply()` 获取到的是一个Series，但是需要用for循环读取每一个值，这和之前用的 `df.apply()` 自动循环每一个值不同

颜色条

```
import pandas as pd

df = pd.read_excel('~/Desktop/1.xlsx')
df.style.bar(color='pink', subset=['age','score'])
```

	name	age	score
0	韩信	21	80
1	赵云	22	10
2	李白	22	30
3	杜甫	24	100
4	曹操	18	75
5	姜尚	45	75
6	阿狗	23	11
7	筱倩	3	99
8	夫子	123	107
9	葛聂	33	90

渐变颜色

需要下载seaborn模块

```
pip install seaborn
```

```
import pandas as pd
import seaborn as sn

df = pd.read_excel('~/Desktop/1.xlsx')
color_map = sn.light_palette('pink', as_cmap=True)

df.style.background_gradient(color_map, subset=['age', 'score'])
```

	name	age	score
0	韩信	21	80
1	赵云	22	10
2	李白	22	30
3	杜甫	24	100
4	曹操	18	75
5	姜尚	45	75
6	阿狗	23	11
7	筱倩	3	99
8	夫子	123	107
9	葛聂	33	90

Series

Series序列

用来生成一维数据。类似list数据类型

```
import pandas as pd

data = {"name":["韩信","貂蝉","李白","杜甫"], "age":[21,18,22,24], "score":
[80,80,80,100]}
sr = pd.Series(data)
print(sr)

name      [韩信, 貂蝉, 李白, 杜甫]
age       [21, 18, 22, 24]
score    [80, 80, 80, 100]
```

Series组装DataFrame

1. 使用 `dict` 组装。不指定index时，使用默认的index

```
import pandas as pd

sr1 = pd.Series(["韩信","貂蝉","李白","杜甫"])
sr2 = pd.Series([21,18,22,24])
sr3 = pd.Series([80,80,80,100])
df = pd.DataFrame({'name':sr1, 'age':sr2, 'score':sr3})
print(df)

  name  age  score
0   韩信   21     80
1   貂蝉   18     80
2   李白   22     80
3   杜甫   24    100
```

2. 还可以手动指定index

```
import pandas as pd

sr1 = pd.Series(["韩信","貂蝉","李白","杜甫"], index=(1,2,3,4), name='name')
sr2 = pd.Series([21,18,22,24], index=(1,2,3,4), name='age')
sr3 = pd.Series([80,80,80,100], index=(1,2,3,4))
df = pd.DataFrame({sr1.name:sr1, sr2.name:sr2, sr3.name:sr3})
print(df)

  name  age  NaN
1   韩信   21   80
2   貂蝉   18   80
3   李白   22   80
4   杜甫   24  100
```

3. 使用 `list` 组装。如果不指定index和name，那么会用默认的index来填充columns和index

```
import pandas as pd

sr1 = pd.Series(["韩信","貂蝉","李白","杜甫"],  index=(1,2,3,4), name='name')
sr2 = pd.Series([21,18,22,24], index=(1,2,3,4), name='age')
sr3 = pd.Series([80,80,80,100], index=(1,2,3,4), name='score')
df = pd.DataFrame([sr1, sr2, sr3])
print(df)

        1   2   3    4
name   韩信  貂蝉  李白   杜甫
age    21  18  22   24
score  80  80  80  100
```

可以发现，使用list来组装DataFrame时，会将Series以行的形式展示。如果用dict来组装，那么会将Series以列的形式组装。

DataFrame取出Series

两种方式：

- 这种就是python自身的语法，DataFrame由字典组成，所以使用 `df['column_name']` 即可
- 使用pandas DataFrame数据结构的语法， `df.column_name` 也可以

制图

pandas是构建于matplotlib之上，所以需要先下载

```
# matplotlib (3.2.1)                - Python plotting package
pip install matplotlib
```

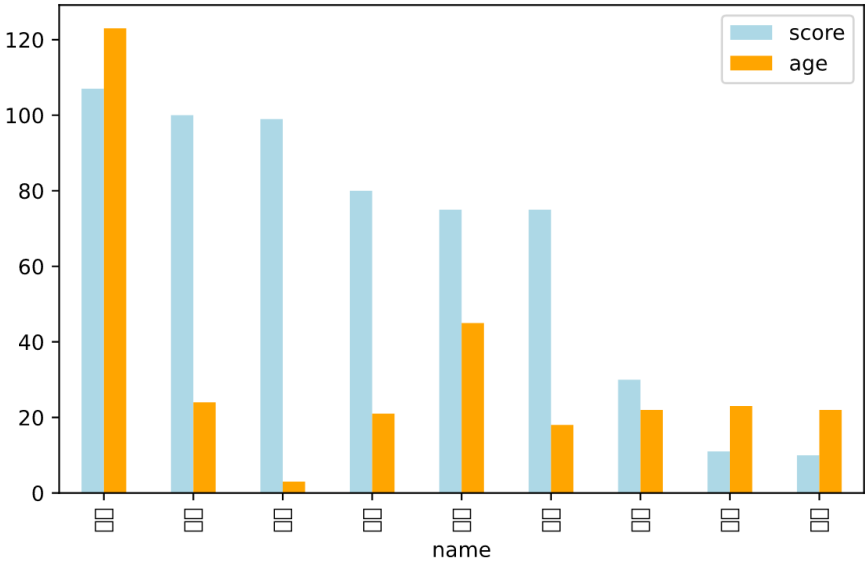
使用到的数据：

	A	B	C	D
1				
2		name	age	score
3	0	韩信	21	80
4	1	韩信	22	10
5	2	李白	22	30
6	3	杜甫	24	100
7	4	曹操	18	75
8	5	姜尚	45	75
9	6	阿狗	23	11
10	7	筱倩	3	99
11	8	夫子	123	107

柱状图

第一个例子：

```
import pandas as pd
import matplotlib.pyplot as plt


df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
# 先排score，再排age，从大到小
df.sort_values(by=['score','age'], inplace=True, ascending=False)
# pandas支持制图
df.plot.bar(x='name', y=['score','age'], color=['lightblue','orange'])
# 紧凑显示
plt.tight_layout()
# 使用matplotlib来渲染
plt.show()
```



中文支持

默认matplotlib无法显示中文，需要从matplotlib导入font_manager

```
from matplotlib import font_manager
```

查看有那些中文字体

```
fc-list :lang=zh

/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc: WenQuanYi Micro Hei,文泉驛微米黑,文泉驛微米黑:style=Regular
/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc: WenQuanYi Micro Hei Mono,文泉驛等寬微米黑,文泉驛等宽微米黑:style=Regular
```

设置font变量

```
font = font_manager.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")
```

在需要显示中文的地方加上fontproperties属性。需要注意只有matplotlib才能加这个属性，所以在需要加这个属性的时候，要用matplotlib来设置

```
plt.xticks(rotation=40, fontproperties=font)
```

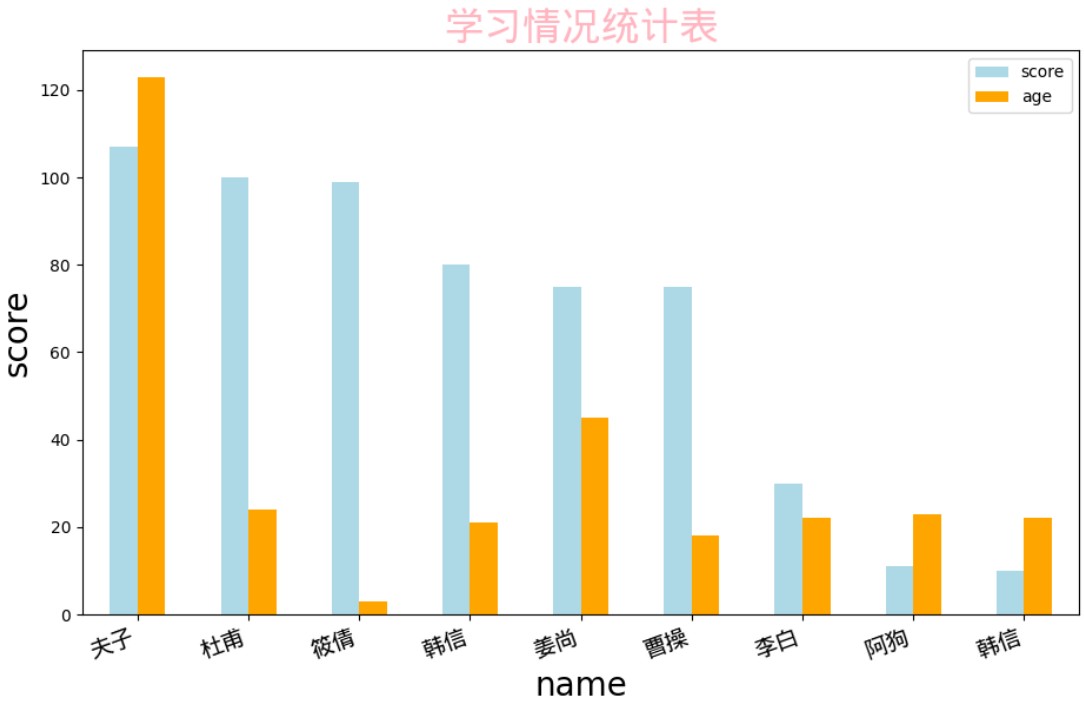
使用matplotlib修饰

matpllotlib比pandas自带的制图功能要多，可定制性更强。导入matplotlib的作用呢就是来补充pandas自身制图功能的一些不足，以及渲染图片。

由于pandas基于matplotlib，所以这两个可完美的搭配使用。在pandas搞不定的地方就用matplotlib来补充

分组柱状图

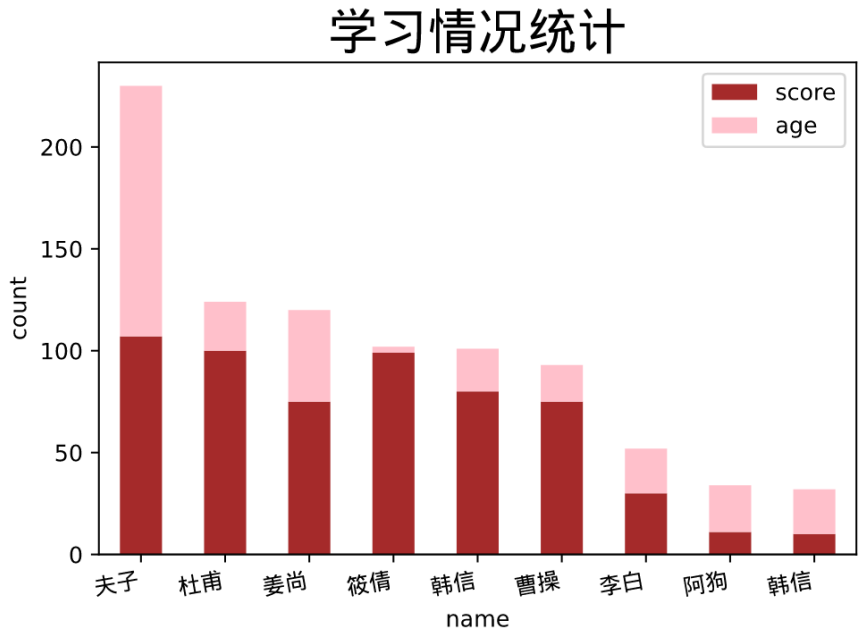
```
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib import font_manager

font = font_manager.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")

# 读取
df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
# 排序
df.sort_values(by=['score','age'], inplace=True, ascending=False)
# 用pandas自身绘制分组柱状图
df.plot.bar(x='name', y=['score','age'], color=['lightblue','orange'])
# pandas自身绘图的标签不支持中文，那么用matplotlib来让标签显示中文，以及设置一定旋转角度
# plt.xticks(rotation=20, fontproperties=font, fontsize=14)
# 上面这个无法设置旋转中心，所以使用下面的
# gca获取x轴，ha='right'设置右端对齐
px = plt.gca()
px.set_xticklabels(df['name'], rotation=20, ha='right', fontproperties=font, fontsize=14)
# gcf获取图形的布局
pf = plt.gcf()
pf.subplots_adjust(left=0.1)
# 用matplotlib来设置x轴名称
plt.xlabel('name', fontsize=20)
# 用matplotlib来设置y轴名称
plt.ylabel('score', fontsize=20)
# 用matplotlib来设置title
plt.title('学习情况统计表', fontproperties=font, fontsize=24, color='lightpink')
plt.tight_layout()
plt.show()
```



叠加柱状图

- 需要自己新计算一个新的列，这个列的值为需要叠加的数据之和
- 使用这个新的列作为排序的标准
- y轴仍然为之前的列
- 只不过在df.plot.bar显示的时候加了一个参数 `stacked=True` ，声明使用叠加的方式

```
import pandas as pd
import matplotlib.pyplot  as plt
from matplotlib import font_manager

font = font_manager.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")

df = pd.read_excel('~/Desktop/1.xlsx', usecols=[1,2,3,4,5], skiprows=1)
df['total'] = df['age'] + df['score']
df.sort_values(by='total', inplace=True, ascending=False)
df.plot.bar(x='name', y=['score', 'age'], color=['brown','pink'], stacked=True)
px = plt.gca()
px.set_xticklabels(df['name'], rotation=10, ha='right', fontproperties=font)
plt.xlabel('name')
plt.ylabel('count')
plt.title('学习情况统计', fontproperties=font, fontsize=22, fontweight='bold')
plt.show()
```

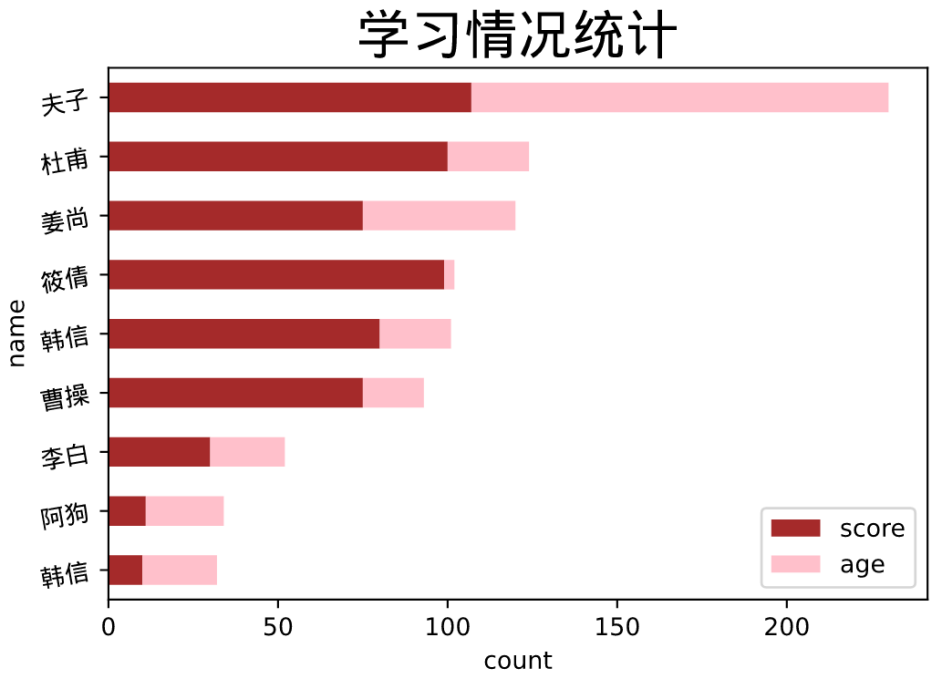


水平叠加柱状图

其实只需要改一个参数就可以了，将 `df.plot.bar` 改为 `df.plot.barh` 就代表水平展示了。

- 这个时候的排序可能需要调整一下
- 如果自定义了x轴，y轴的话，需要交换一下

```
import pandas as pd
import matplotlib.pyplot  as plt
from matplotlib import font_manager

font = font_manager.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")

df = pd.read_excel('~/Desktop/1.xlsx', usecols=[1,2,3,4,5], skiprows=1)
df['total'] = df['age'] + df['score']
df.sort_values(by='total', inplace=True)
df.plot.barh(x='name', y=['score', 'age'], color=['brown','pink'], stacked=True)
px = plt.gca()
px.set_yticklabels(df['name'], rotation=10, ha='right', fontproperties=font)
plt.xlabel('count')
plt.ylabel('name')
plt.title('学习情况统计', fontproperties=font, fontsize=22, fontweight='bold')
plt.show()
```



饼图

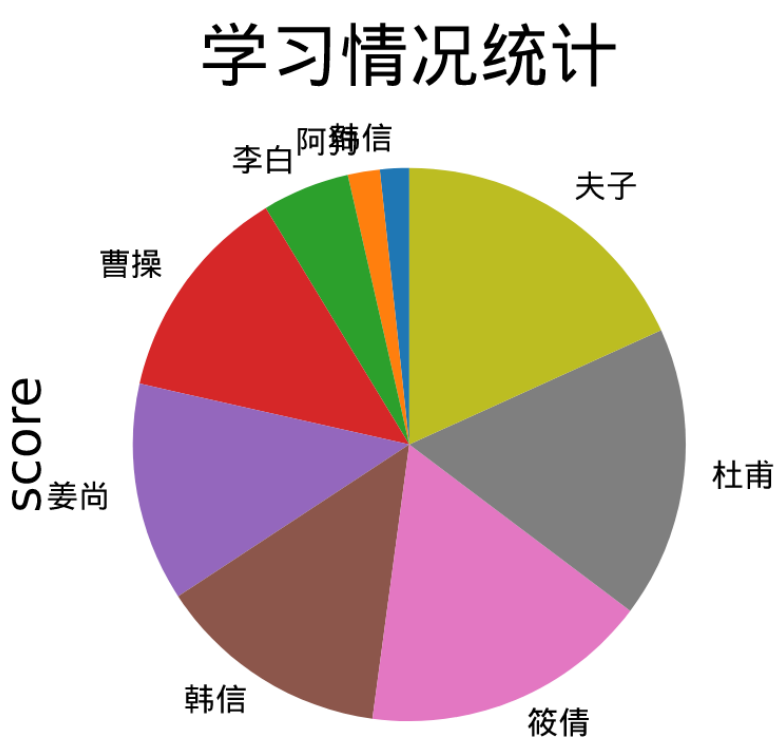
绘制饼图的优点是可以清晰的看出各部分所占的比例

与柱状图不同，饼图需要的数据为一个Series，同时在读取数据的时候指定的index会成为饼图显示的信息

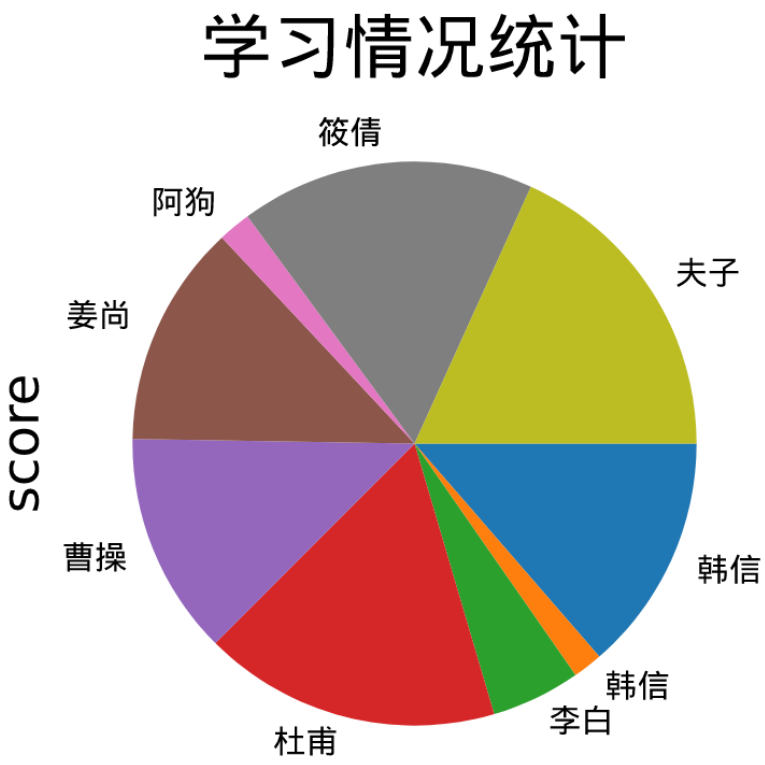
- 饼图注意点还挺多的，最为重要的就是使用pandas来绘制，无法显示中文，所以采用了plt来绘制
- 采用matplotlib时，也无法直接设置字体，需要用循环给具体的对象赋值
- 对序列排序，并通过绘制时的顺时针来设置旋转方向
- 通过startangle来设置最大值从12点方向开始

```
import pandas as pd
from matplotlib import font_manager as fm
from matplotlib import pyplot as plt

font = fm.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-
microhei.ttc")
df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
# 为了顺序显示，进行排序。这里没有从大到小排是因为后面绘制的时候可以使用counterclock来设置顺时针排序
df.sort_values(by='score', inplace=True)
# 使用pandas自身的绘图，无法显示中文，所以不采用
# df['score'].plot.pie(labels=df['name'])
# 使用matplotlib来绘制图形，label为标签名
paint = plt.pie(df['score'], labels=df['name'], counterclock=True, startangle=-270)
# 即便是通过matplotlib来绘制，中文也不好显示，只能通过返回值来找到text所在位置，并循环设置中文字体
for i in df.index:
    paint[1][i].set_fontproperties(font)
plt.title('学习情况统计', fontproperties=font, fontsize=22, fontweight='bold')
# ylabel来设置左边显示文字
plt.ylabel('score', fontsize=16)
plt.show()
```

对数据进行排序之后会出现，比例较小的部分字重叠，这也是一个问题。



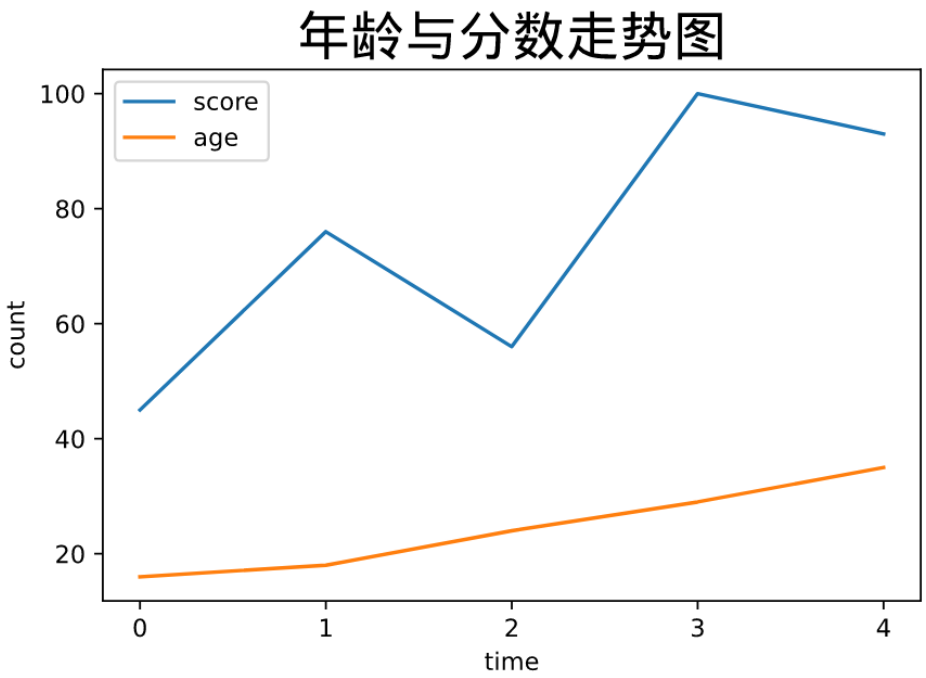
所以不对其进行排序显示的效果可能会更好。



折线图

- 折线图在绘制的时候，只要直接使用 `.plot` 就可以了。需要指定y轴的数据
- x轴的数据可以通过指定index来设置

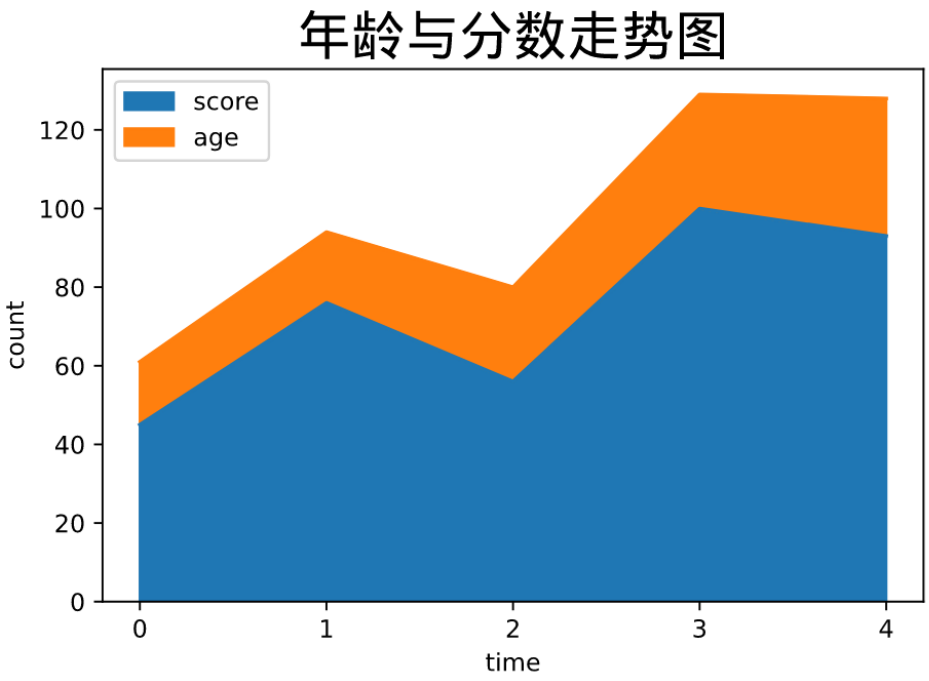
```
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import font_manager as fm

font = fm.FontProperties(fname='/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc')
data = {'score':[45,76,56,100,93], 'age':[16,18,24,29,35]}
df = pd.DataFrame(data)
df.plot(y=['score','age'])

plt.title('年龄与分数走势图',fontproperties=font, fontsize=22)
plt.xlabel('time')
plt.xticks(df.index)
plt.ylabel('count')
plt.show()
```



叠加区域图

只需要在折线图的基础上将 `.plot` 改为 `.plot.area` 即可

```
df.plot.area(y=['score','age'])
```

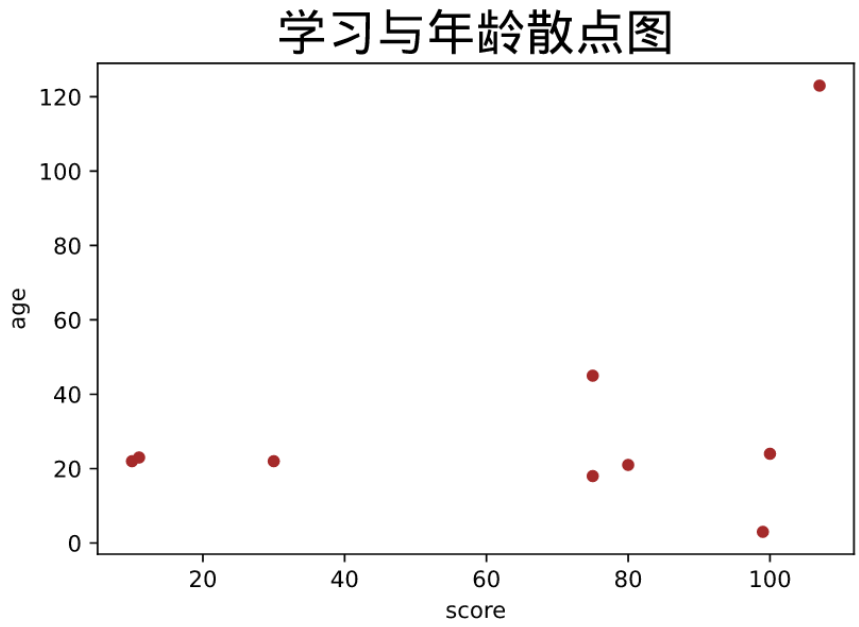


散点图

散点图就可以看出pandas的处理速度完胜excel，同时交换x，y轴数据，pandas可以轻松搞定，而excel在数据量多的时候无法操作。

- 使用到了 `df.plot.scatter()` 来画散点图

```
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import font_manager as fm

font = fm.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")

df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
df.plot.scatter(x='score', y='age', color='brown')
plt.title('学习与年龄散点图', fontproperties=font, fontsize=22)
plt.show()
```

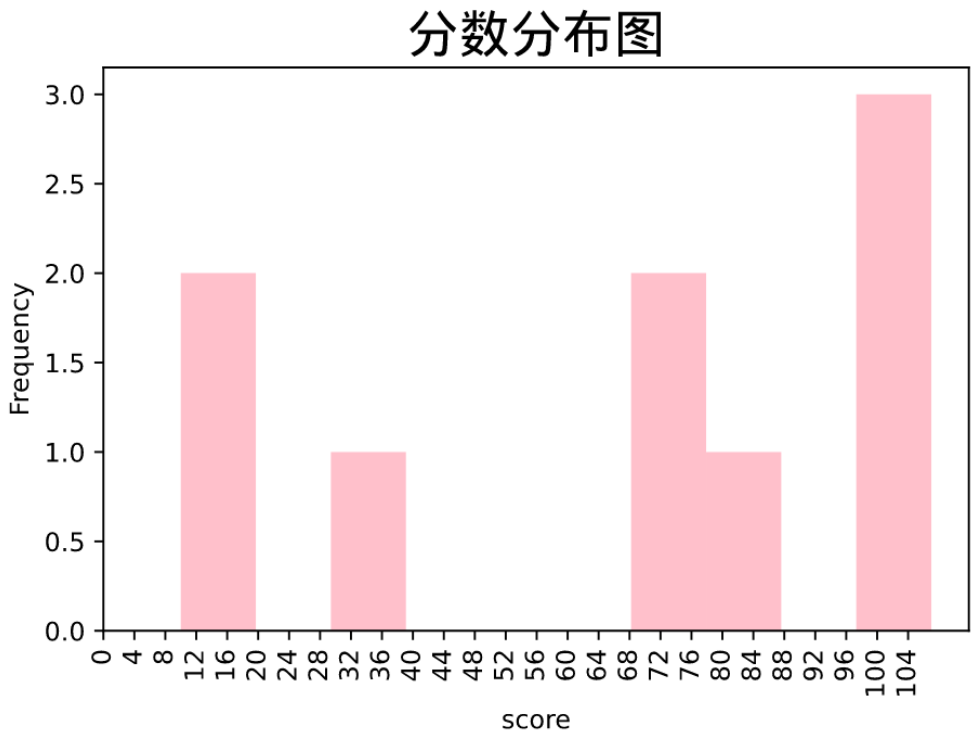


分布图

可以清晰的看出指定数据的分布情况。分布图只需要Series就可以

- 使用了 `Series.plot.hist(bins=num)`
- 参数bins设置划分区间，越多图越精细
- `plt.xticks()` 设置x轴的分段，默认Series分段少，使用range()函数，来设置步长

```
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import font_manager as fm

font = fm.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-microhei.ttc")

df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
df['score'].plot.hist(bins=10)
plt.xticks(range(0,max(df['score']),4), rotation=90)
plt.xlabel('score')
plt.title('分数分布图', fontproperties=font, fontsize=20)
plt.show()
```



密度图

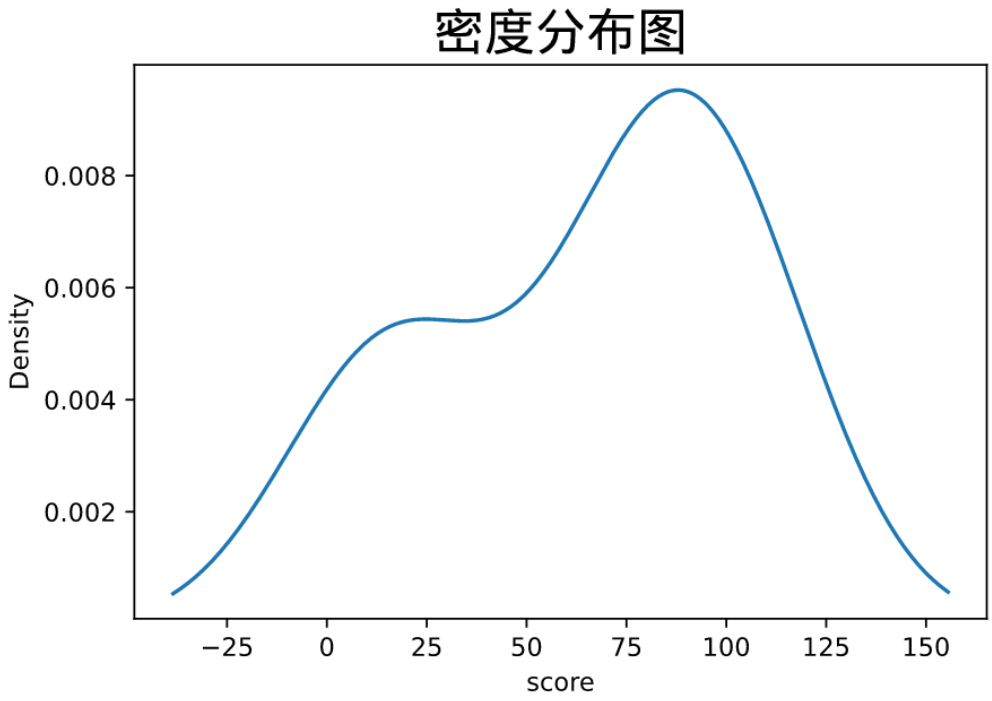
直接使用会报错，提示缺少一个scipy的科学库

```
# scipy (1.4.1)                    - SciPy: Scientific Library for Python
pip install scipy
```

密度图同样使用Series

- 使用 `Series.plot.kde()` ,其它与分布图类似

```
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib import font_manager as fm

font = fm.FontProperties(fname="/usr/share/fonts/wenquanyi/wqy-microhei/wqy-
microhei.ttc")

df = pd.read_excel('~/Desktop/1.xlsx', skiprows=1, usecols=[1,2,3,4])
df['score'].plot.kde()
plt.xlabel('score')
plt.title('密度分布图', fontproperties=font, fontsize=20)
plt.show()
```



终于一口气学完了这个模块，感觉功能确实很多，而且没学到的功能还有很多。但是就目前掌握的这些来说，写个自动化办公的脚本来说已经够了。数据的读取，增删查改，制图等等功能都很好用。虽然是一边看视频，一边敲代码，但是熟悉程度还不够，还的在日常的生活中多用才会熟练。学完一个东西觉得很开心，但是又觉得有点失落，因为我接下来要学什么呢？是个很大的问题。要不学学powershell吧！