开启命令

jupyter notebook

提示:

两种模式通用快捷键

* **Shift+Enter，执行本单元代码，并跳转到下一单元**
* **Ctrl+Enter，执行本单元代码，留在本单元**

**Matplotlib使用**

折线图

import matplotlib.pyplot as plt

#设置图片大小,清晰度

plt.figure(figsize=(10, 10),dpi=80)

#设置xy轴的对应关系

x = range(60)

y\_shanghai = [random.uniform(15, 18) for i in x]

plt.plot(x, y\_shanghai,label=”上海”)

# 使用plot可以画多个折线,设置线条属性

plt.plot(x, y\_beijing, color='r', linestyle='--', label="北京")

# 修改x,y坐标的刻度

x\_ch = ["11点{}分".format(i) for i in x]

y\_ticks = range(40)

plt.xticks(x[::5], x\_ch[::5]) #对应好设置成中文

plt.yticks(y\_ticks[::5])

# 增加标题和xy轴描述

plt.xlabel("时间")

plt.ylabel("温度")

plt.title("中午11点0分到12点之间的温度变化图示")

# 添加图形注释

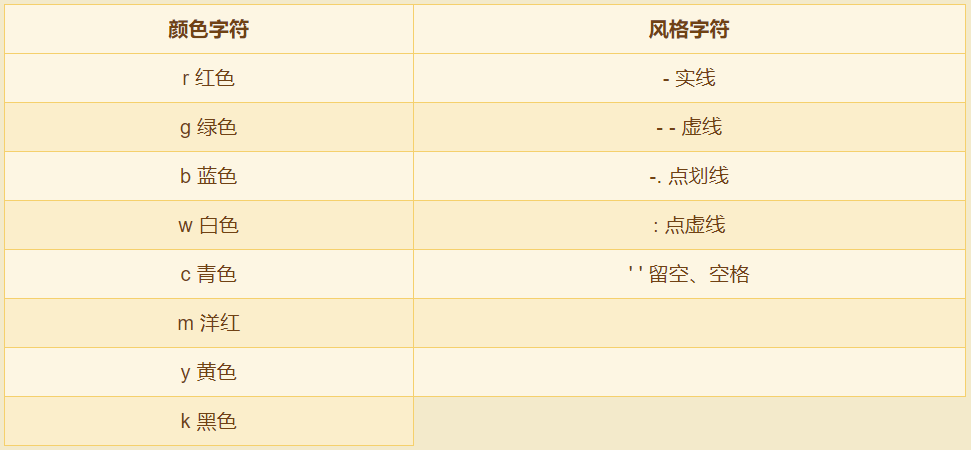
plt.legend(loc="best")

#保存图片

plt.savefig("test.png")

#显示图片

plt.show()



多坐标系

# 创建一个figure,一行两列

fig, ax = plt.subplots(nrows=1, ncols=2, figsize=(20, 8), dpi=80)

# 操作每个坐标系

ax[0].plot()

设置标题等方法不同：

set\_xticks

set\_yticks

set\_xlabel

set\_ylabel

set\_title

柱状图

# 准备电影的名字以及电影的票房数据

movie\_name = ['雷神3：诸神黄昏',…]

y = [73853,57767,…]

# 放进横坐标的数字列表

x = range(len(movie\_name))

# 画出条形图

plt.bar(x,y,width=0.5,color=['b','r',…])

# 修改刻度名称

plt.xticks(x, movie\_name)

再添加一条柱状图

# 画出柱状图

plt.bar(x, first\_day, width=0.2, label="首日票房")

# 首周柱状图显示的位置在首日的位置右边

plt.bar([i+0.2 for i in x], first\_weekend, width=0.2, label="首周票房")

# 显示X轴中文，固定在首日和首周的中间位置

plt.xticks([i+0.1 for i in x], movie\_name)

直方图

# 展现不同电影的时长分布状态

plt.figure(figsize=(20, 8), dpi=100)

# 准备时长数据

time =[131, 98, 125,…]

# 定义一个间隔大小

a = 2

# 得出组数

bins = int((max(time) - min(time)) / a)

# 画出直方图(数据,组数,显示频率)

plt.hist(time, bins, normed=1)

# 指定刻度的范围，以及步长

plt.xticks(list(range(min(time), max(time)))[::a])

plt.xlabel("电影时长大小")

plt.ylabel("电影的数据量")

# 更好的显示(显示虚线, alpha透明度)

plt.grid(True, linestyle='--', alpha=0.5)

plt.show()

饼图

# 展现每部电影的排片的占比

plt.figure(figsize=(20, 8), dpi=100)

# 准备每部电影的名字，电影的排片场次

movie\_name = ['雷神3：诸神黄昏','正义联盟',…]

place\_count = [60605,54546,…]

# 通过pie

plt.pie(place\_count,labels=movie\_name,autopct='%1.2f%%',colors=['b','r',…])

# 指定显示的pie是正圆

plt.axis('equal')

plt.legend(loc='best')

plt.title("排片占比示意图")

plt.show()

其他

# 使用splines以及设置颜色，将上方和右方的坐标去除

ax.spines['right'].set\_color('none')

ax.spines['top'].set\_color('none')

# 将刻度设置为空，去除刻度

plt.xticks([])

plt.yticks([])

numpy

a = numpy.array



dtype是numpy.dtype类型

创建数组

1.0和1

np.ones([3,4])

np.zeros([3,4])

2.从现有的数据中创建

a = np.array([[1,2,3],[4,5,6]])

# 从现有的数组当中创建

a1 = np.array(a)

# 相当于索引的形式，并没有真正的创建一个新的

a2 = np.asarray(a)

3.创建固定范围数组

np.linspace(start, stop, num, endpoint, retstep, dtype)

start 序列的起始值

stop 序列的终止值，

如果endpoint为true，该值包含于序列中

num 要生成的等间隔样例数量，默认为50

endpoint 序列中是否包含stop值，默认为ture

retstep 如果为true，返回样例，

以及连续数字之间的步长

dtype 输出ndarray的数据类型

eg:

np.linspace(0, 100, 10)

np.arange(10, 50, 2)

4.创建随机数组

np.random模块

均匀分布

np.random.rand(10)

np.random.uniform(0,100)

np.random.randint(100)

正态分布

给定均值／标准差／维度的正态分布

np.random.normal(1.75, 0.2, (3,4))

np.random.standard\_normal(size=(3,4))

二维数组索引

stock\_day\_rise[0, 0:10]

形状修改

1.行列互换,没修改形状

stock\_day\_rise.reshape([504, 500])

2.修改形状

stock\_day\_rise.resize([504,500])

3.变为一维

stock\_day\_rise.flatten()

修改类型

stock\_day\_rise.reshape([504, 500]).astype(np.int32)

修改小数位

np.round(stock\_day\_rise[:2, :20], 4)

数组转置

stock\_day\_rise.shape

(500, 504)

stock\_day\_rise.T.shape

(504, 500)

转成二进制

arr.tostring()

复制

temp = stock\_day\_rise[:4, :4].copy()

逻辑运算

把大于0.5的赋值1

temp[temp > 0.5] = 1

通用函数判断

# 对所有的数据进行判断,返回T,F

np.all(temp[0:2,0:5] > 0)

# 返回不重复的值

change\_int = temp[0:2,0:5].astype(np.int)

np.unique(change\_int)

复合逻辑运算

复合逻辑需要结合np.logical\_and和np.logical\_or使用

np.where(np.logical\_and(temp > 0 ,temp<0.5 ), 1, 0)

统计运算

axis = 0对列计算 (1对行计算)

np.max(temp, axis=1) # 最大

np.min(temp, axis=1) # 最小

np.mean(temp, axis=1) # 平均

np.var(temp, axis=1) # 方差,波动

np.std(temp, axis=1) # 标准差

最大,最小值的位置

np.argmax(temp, axis=)

np.argmin(temp, axis=)

数组间的运算

arr + 1

arr / 2

arr \* 3

广播机制(不包含矩阵)

每个对应的维度满足

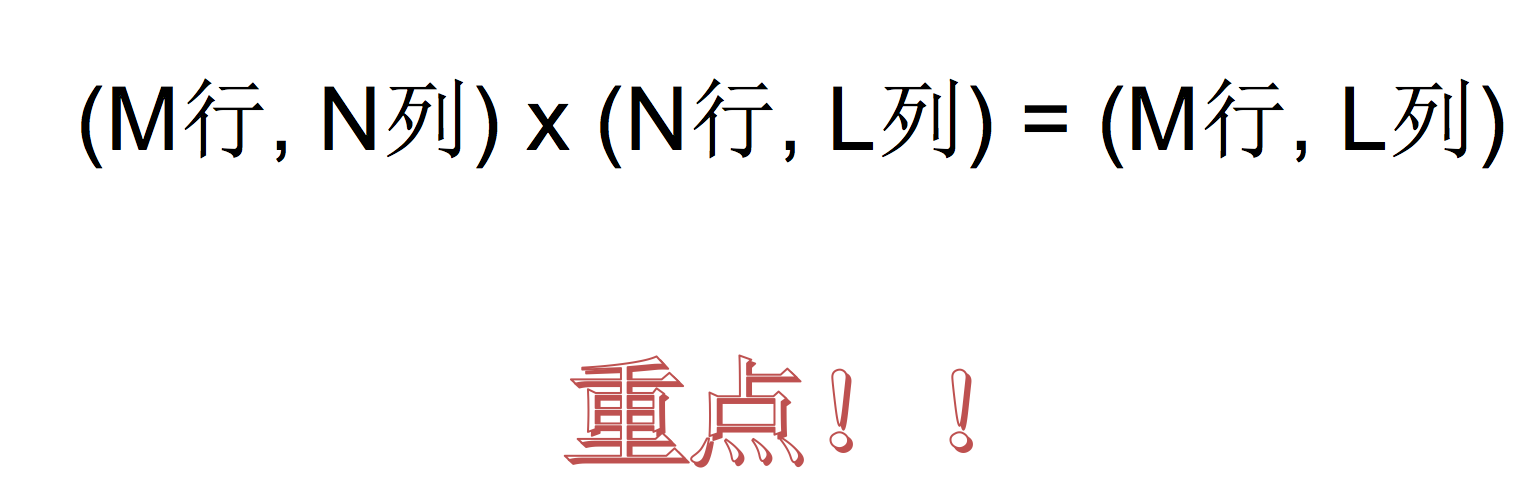
1.数量相等

2.shape（其中相对应的一个地方为1）

矩阵

二维数组

a = np.mat(temp)



矩阵相乘

np.matmul(a,b)

数组合并

# axis=1，按列合并

# axis=0，按行合并

np.concatenate([a, b], axis=0)

np.hstack([a,b]) # 列拼接

np.vstack([a,b]) # 行拼接

数组分割

# 分割成4分,按行分割

np.split(ab, 4, axis=0)

数据读取

# (路径,分隔符',')

test = np.genfromtxt("./data/numpy\_test/test.csv", delimiter=',')

缺失值nan(float类型)

e\*\*2 = np.exp(2)

Pandas

三大数据结构，Series、DataFrame以及Panel

DataFrame

#构造行索引索引序列

stock\_code = ['股票' + str(i) for i in range(stock\_day\_rise.shape[0])]

# 添加行索引

data = pd.DataFrame(stock\_day\_rise, index=stock\_code)

# 生成一个时间的序列，略过周末非交易日

date = pd.date\_range('2017-01-01', periods=stock\_day\_rise.shape[1], freq='B')

# index代表行索引，columns代表列索引

data = pd.DataFrame(stock\_day\_rise, index=stock\_index, columns=date)

DataFrame属性

shape

dtypes

ndim

index 行索引

columns 列索引

values 获取值,不包含索引

T 转置

查询相关

head() 前5行

tail() 后5行

修改索引(必须整体修改)

data.index = [i for i in range(500)]

重置索引(drop丢弃原来的索引)

data.reset\_index(drop=True)

以某列值设置新索引

df = pd.DataFrame({'month':[1,4,7,10], 'year':[1, 1, 2, 2], 'sale':[55, 40, 84, 31]})

# df.set\_index(['month'])# 设置新的索引值，但是返回一个新的dataframe

df = df.set\_index(['month'])

# 设置多重索引 MultiIndex的结构

df.set\_index(['year', df.index])

Series

创建

pd.Series(np.arange(10))

pd.Series([6.7,5.6,3,10,2], index=)

pd.Series({'red':100, ''blue':200, 'green': 500, 'yellow':1000})

属性

index

values

索引操作

直接使用行列索引(先列后行)

结合loc或者iloc使用索引

使用ix组合索引

1. 直接使用行列索引(先列后行)

data[['open'], ['high']]

data['open'][['2018-02-27']]

2.loc,iloc

data.loc['2018-02-27':'2018-02-22', 'open']

data.iloc[0:100, 0:2].head()

3.ix

data.ix[0:10, ['open', 'close']]

# 相当于

data[['close', 'open', 'high']][0:3]

修改值

data['close'] = 1

# 或者

data.close = 1

排序

# 对索引进行排序

data.sort\_index()

# 按照涨跌幅大小进行排序 , 使用ascending指定按照大到小排序

data = data.sort\_values(by='p\_change', ascending=False)

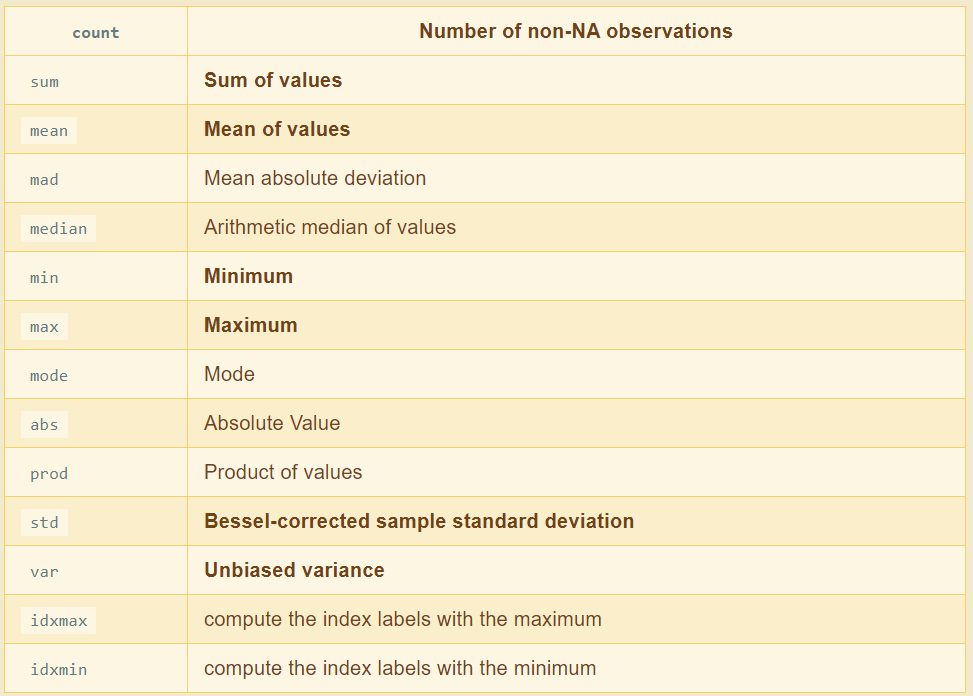
# 按照过个键进行排序

data = data.sort\_values(by=['open', 'high'])

统计运算

# 计算平均值、标准差、最大值、最小值、分位数

data.describe()



# 单独计算

data['close'].max()

# 对所有的列进行计算

data.max(0)

# 对所有的行进行计算

data.max(1)

# 求出最大值的位置

data.idxmax(axis=0)

# 求出最小值的位置

data.idxmin(axis=0)



# 排序之后，进行累计求和

data = data.sort\_index()

#计算累计函数

stock\_rise = data['p\_change']

# plot方法集成了前面直方图、条形图、饼图、折线图

stock\_rise.cumsum()

# 简单画图plot

import matplotlib.pyplot as plt

stock\_rise.cumsum().plot()

plt.show()

逻辑运算

<,>,|,&,

data[(data['p\_change'] > 2) & (data['open'] > 15)]

query用法

data.query(“p\_change > 2 & open > 15”)

isin()

data[data['turnover'].isin([4.19])]

算术运算

add(),sub()

close1 = data['close']

open1 = data['open']

data['m\_price\_change'] = close1.sub(open1)

自定义运算 apply()

data[['open', 'close']].apply(lambda x: x.max() - x.min(), axis=0)

data[['open', 'close']].apply(lambda x: x.max() - x.min(), axis=1)

文件操作

csv读取

pandas.read\_csv(filepath, usecols=[],sep =',' , delimiter = None)

usecols:指定读取的列名，列表形式

csv保存

DataFrame.to\_csv(path=None,sep=',’,columns=None,header=True,index=False, index\_label=None, mode='w', encoding=None)

columns:指定保存列[]

mode:'w'：重写, 'a' 追加

index:是否写进行索引

header:default True,列名是否写进列索引值

h5读取 (推荐)

pandas.read\_hdf(path，key =None，\*\* kwargs)

从h5文件当中读取数据

path:文件路径

key:读取的键

mode:打开文件的模式

h5存储

DataFrame.to\_hdf(path= ,key= )

缺失值处理

np.nan

1. 删除(按行删除)

movi = movie.dropna()

1. 替换(按每列平均,中位值填)

movie['Revenue'].fillna(movie['Revenue'].mean(),inplace=True)

其他字符(?)

1、先替换‘?’为np.nan

df.replace(to\_replace=, value=)

2、在进行缺失值的处理

# 把一些其它值标记的缺失值，替换成np.nan

wis = wis.replace(to\_replace='?', value=np.nan)

wis.dropna()

数据离散化

**连续属性的离散化就是将连续属性的值域上，将值域划分为若干个离散的区间，最后用不同的符号或整数** **值代表落在每个子区间中的属性值。**

# 画图

data = pd.read\_csv("./data/stock\_day/stock\_day.csv")

p\_change= data['p\_change']

p\_change.hist(bins=80)

plt.show()

# 自行分组qcut

qcut = pd.qcut(np.abs(p\_change), 10)

qcut.value\_counts()

# 自定义分组cut

bins = [-100, -7, -5, -3, 0, 3, 5, 7, 100]

p\_counts = pd.cut(p\_change, bins)

哑变量矩阵

dummaries = pd.get\_dummies(p\_counts, prefix="rise")

prefix:分组名字

合并

1. pd.concat实现数据合并

pd.concat([data1, data2], axis=1)

2. pd.merge 按键合并

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)

* 可以指定按照两组数据的共同键值对合并或者左右各自
* left: A DataFrame object
* right: Another DataFrame object
* on: Columns (names) to join on. Must be found in both the left and right DataFrame objects.
* left\_on=None, right\_on=None：指定左右键

result = pd.merge(left, right, on=['key1', 'key2'])

交叉表与透视表

# 寻找星期几跟股票张得的关系

# 1、先把对应的日期找到星期几

date = pd.to\_datetime(data.index).weekday

data['week'] = date

# 2、假如把p\_change按照大小去分个类0为界限

data['posi\_neg'] = np.where(data['p\_change'] > 0, 1, 0)

# 通过交叉表找寻两列数据的关系

count = pd.crosstab(data['week'], data['posi\_neg'])

# 算数运算，先求和

count.sum(axis=1).astype(np.float32)

pro = count.div(count.sum(axis=1).astype(np.float32), axis=0)

透视表

data.pivot\_table(['posi\_neg'], index=['week'])

**分组与聚合**

DataFrame.groupby(key, as\_index=False)

* key:分组的列数据，可以多个

# 分组，求平均值

col.groupby(['color'])['price1'].mean()

col['price1'].groupby(col['color']).mean()

# 分组，as\_index数据的结构不变

col.groupby(['color'], as\_index=False)['price1'].mean()

分组聚合

# 按照国家分组，求出每个国家的星巴克零售店数量

count = starbucks.groupby(['Country']).count()

count['Brand'].plot(kind='bar', figsize=(20, 8))

plt.show()

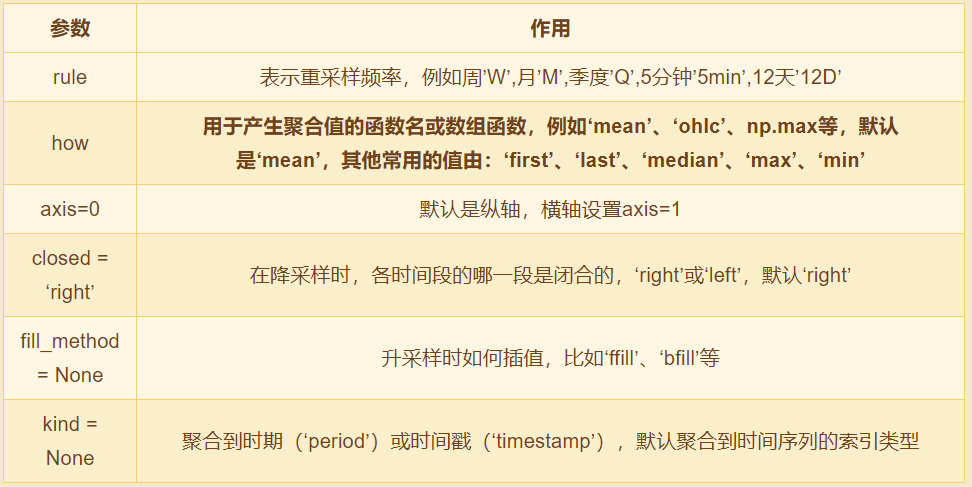
# 设置多个索引，set\_index()

starbucks.groupby(['Country', 'State/Province']).count()

K线

DataFrame.resample(rule,how=None,axis=0,fill\_method=None,closed=None,kind=None,)

* 频率转换和时间序列重采样，对象必须具有类似日期时间的索引（DatetimeIndex，PeriodIndex或TimedeltaIndex）



周K线

stock\_day = pd.read\_csv("./data/stock\_day/stock\_day.csv")

stock\_day = stock\_day.sort\_index()

# 对每日交易数据进行重采样 （频率转换）

stock\_day.index

# 1、必须将时间索引类型编程Pandas默认的类型

stock\_day.index = pd.to\_datetime(stock\_day.index)

# 2、进行频率转换日K---周K,首先让所有指标都为最后一天的价格

period\_week\_data = stock\_day.resample('W').last()

# 分别对于开盘、收盘、最高价、最低价进行处理

period\_week\_data['open'] = stock\_day['open'].resample('W').first()

# 处理最高价和最低价

period\_week\_data['high'] = stock\_day['high'].resample('W').max()

# 最低价

period\_week\_data['low'] = stock\_day['low'].resample('W').min()

# 成交量 这一周的每天成交量的和

period\_week\_data['volume'] = stock\_day['volume'].resample('W').sum()