笔记汇总

利用Python进行数据分析--IPython

`IPython` 的使用

自动补全 `tab`键自动补全，`Enter`键确认

内省

- 在变量前后加入 `?` 函数则会显示 `docstring`
- 如果是 `??` 函数会显示源代码
- 可以用通配符寻找需要要的函数名称

```
re.*find*? # 输出结果
# re.findall
# re.finditer
```

历史命令

- 利用 `hist` 或 `%hist` 查看所有的历史输入
- `_` `__` `___` 和 `_i` `_ii` `_iii` 分别表示最后三个输出和输入

用 `%run` 运行脚本 使用 `%timeit` 命令快速测量代码运行时间 使用 `Pylab` 进行交互式计算

用 `!` 作为调用系统 `shell` 的前缀

利用Python进行数据分析--Numpy

`ndarray` 是一种多维数组对象，可以利用这种数组，对整块数据做数学运算。

```
arr = np.array([1,2,3,4]) # 支持从列表生成
arr1 = np.zeros(10)
arr2 = np.ones(10)
arr3 = np.zeros((2,3))
# 可以创建全0、全1，并且可以规定形状
arr4 = np.empty(10)
# 也可以是全部未知值
np.arange(15)
# 和自带的range函数类似
arr.shape
# 返回形状（2，3）
np.eye()
np.indentity() # 创建对角线为1的n*n矩阵
```

数组之间的运算都会作用在**元素级**，而数组和标量之间的运算则会以**广播的方式**传播。

```
arr * arr # 每个元素相乘
1/arr # 广播
```

索引和切片 基本的索引的切片和列表是一样的，表现的特性也大致相同

```
arr[5:8] =  12 # 广播
```

同时对切片的数据处理会反应在原来的数组上，因为切片只是不同的视图 `View`。 对于多维数组 `arr[0][2]` 和 `arr[0, 2]` 是一样的。

```
arr2d[:2, 1:] # 也可以传入多个切片
arr2d[:2, :] = 0 # 广播
```

布尔型索引

```
# names是一个string数组
names == 'Bob' # 这个运算也是广播，会生成一个等长的bool类型数组
# 这个布尔类型数组可以用于索引
data[names == 'Bob'] # 等长的bool类型数组可以用于筛选
data[names == 'Bob', 2:] # 可以和切片混合使用
data[names == 'Bob', 3]

data[data < 0] = 0 # 利用bool型索引快速处理小于0的值
```

`data[data < 0] = 0` 利用bool索引可以快速简洁的对某一部分进行操作。

花式索引 索引可以传入一个**列表**作为索引，选出特定的行，并且能够按照特定的顺序。

```
arr[[4,3,0,6]] # 选出特定的行按特定的顺序排列
arr[[1,5,7,2], [0,3,1,2]] # 这个操作会把两个列表结合直接把该位置的值取出来
# 注意不是先对行排列再对列排列！
arr[[1,5,7,2]][:, [0,3,1,2]] # 这个才是先对行排列再对列排列
# 或是利用 np.ix_([],[]) 创建一个索引器
```

转置和轴变换

```
arr.T #  转置
arr.swapaxes(1, 2) # 按照某轴进行变换，0表示行，1表示列，2表示...
```

通用函数

```
# 一元函数
arr = np.sqrt(arr)
np.exp(arr)
arr.floor(arr) # 数值变换

# 二元函数
np.add()
np.subtract()
np.equal()
# 一些示例
```

条件逻辑与数组运算 `np.where` 的巧妙运用，类似与 `x if c else y` 的语句块，在这里，更加灵活，对数组进行元素级别的判断和运算。

```
result = np.where(cond, arrx, arry) # 逐个判断
np.where(arr > 0, 2, -2) # 参数可以是标量
```

数学和统计方法 `arr.sum`，`arr.mean`，`arr.min`，`arr.std`，`arr.var` 这些统计上的函数都是可以使用过的。特别的用于bool数组的还有 `bools.any` 和 `bools.all`

排序

```
arr.sort() # 二维数组默认列从小到大排列
arr.sort(1) # 传入轴的参数，行从小到大排列
```

集合逻辑

线性代数

随机 `np.random`

Pandas

基本数据结构

最基本的两个数据结构是 `Series` 和 `Dataframe` ，和数组相似，特别指出是带有索引，并且能够根据索引访问。

```
obj = Series([4,7,5,-3], index=['d', 'a', 'b', 'c'])
obj2 = obj[['a', 'b']]
# 能够选取访问多个索引
obj2[obj2 > 0]
# Numpy的布尔索引，数组元素都保留了，同时还可以看成订场的字典

obj4 = Series(sdata, index=states)
# 可以传入新索引，缺失的值为Na
obj4.isnull()
obj4,notnull()
# 检测缺失值，并且返回布尔数组
```

dataframe

```
frame = DataFrame(data, columns=[], index=[])
# 用columns指定列序列的顺序
# index表示行的索引
# 缺失值都会用NA表示

frame['state']
frame.year
# 两种方位列属性的方式

frame.ix['three']
# 访问行索引的方式
frame['debt'] = 16.5
# 进行赋值时仍然有传播性质
del frame['debt']
# 用del删除
frame.values
# 返回ndarray表示的数据
```

重要方法和基本功能

重索引

```
obj,reindex([]) # 重新创建索引，并根据新索引重排
obj.reindex([], fill_value=0)
# 对于缺失值可以指定填充方式
obj.reindex([], method='ffill')
# 可以指定插值方法
obj.reindex(index=[], columns=[])
# 可以重新索引行或者列，插值只能应用于行
obj.ix([], states)
#  利用.ix实现更加简单
```

丢弃

```
data.drop('a') # 默认是列
data.drop(['a','b']) # 一组列
data.drop(['two','four'], axis=1) # 通过axis参数可以丢弃行
```

索引选取过滤

```
# 对于Series，可以按照整数选取，也可以按照索引名切片
s[[1,3]] # 选取一组
s[1:3]  # 切片
s['a':'b'] # 注意边界包含
s['a':'b']  = 1 # 传播

# 而对于DataFrame，则是索引列
data['two'] # 列或一组列

data[:2] # 如果是切片则是选择行了
data[data['three'] > 5] # 布尔型数组选取行
data[data < 0] = 5 # 布尔型dataframe索引
# 总结起来索引形式有 obj[val]，obj.ix[val]，注意不同表现形式
# 也有icol。irow方法，按照位置选取
```

算术运算和数据对齐

```
df1 +  df2
# 注意加法的时候会对齐
# 会引入NA

# 通过使用算术方法，可以设置天充值
df1.add(df2, fill_value=0)
df1.reindex(columns, fill_value) # 重新索引也可以指定填充

# dataframe和series的运算
arr - arr[0] # 在numpy里面是传播的
frame - frame.ix[0] # 在pandas也是
series = frame['d']
frame.sub(series, axies=0)
# 如果要减去某一列并传播，则必须这样写，axis指定轴
```

函数应用和映射

```
f # 求最大最小的差
frame.apply(f) # 在列上应用，一列一列
frame.apply(f, axis=1) # 在行上应用，一行一行
frame.applymap(f) # 可以在元素级上
frame['a'].map(f) # 在series上是map方法
```

排序

```
frame.sort_index() # 列方向的索引，即行索引排序
frame.sort_index(aixs=1, ascending=False) # 行方向上的索引，即列索引排序，默认升序

series.order() # 按值排series
frame.sort_index(by='b') # 按照某一列的值排序
# 传入索引列表，则会有多个标准
```

汇总描述性统计

```
df.sum(axis=1, skipna=False) # 按照不同方向，是否跳过缺失值
#  还有level参数规定，层级
```

相关系数协方差

```
returns.corr()
returns.cov()
#  自相关矩阵，和协方差矩阵
```

```
returns.corrwith(returns.IBM)
# 逐列求相关系数
```

文件读入

```
df = pd.read_csv('a.csv', names=[], index_col='message')
# 给出读入的数据的列名和索引
# skiprows=[] 跳过某些行
# 那na_values=[]接受表示缺失值的字符串
# header表示用作列名的行号，默认为0
```

可以读入 `json` 格式的数据，注意读入的方式。

数据规整化

dataframe合并

```
pd.merge(df1, df2)
# 默认是去重叠列名
pd.merge(df1, df2, on='key')
# 可以根据一个或者多个key合并
pd.merge(df1, df2, left_on='k1', right_on='k2')
# 可以分别给定左右的键
pf.merge(df1, df2, how='outer')
# 默认是inner，取交集，outer是并集，还可以用left，right
pf.merge(df1, df2, suffixes=('_l', '_r'))
# 可以指定未合并的相同列的后缀名
```

索引上的合并

- `pd.merge`
- `df.join`

```
pd.merge(df1, df2, left_on='k1', right_index=True)
# 用索引，如果是层次式索引，则必须指定相应的列
df1.join(right2, on='key', how='outer')
# 这时默认左边是用索引
```

轴向连接

- `pd.concat`
- `df1.combine_first` ```py

s1，s2，s3都是series

pd.concat([s1, s2, s3], aixs=1) pd.concat([s1, s2, s3], aixs=1, join='inner')

这种方式将去行索引的并集

可以通过join_axes指定行索引

pd.concat([s1, s2, s3], keys=[])

默认列向连接，指定keys相当于增加层次索引

如果axis=1，相当于给定列名

同样的逻辑对于dataframe也是一样的，会增加层及索引

df1.combine_first(df2) pd.w here(pd.isnull(df1), df2, df1)

这两个作用相同

重塑和轴向旋转
+ `stack` 将列旋转为行
+ `unstack` 行旋转为列，并且默认操作的是**最内层**
+ `pivot` 用某一列数据作为索引，分别行索引，列索引，最后一个参数说明数列
```py
# 行索引是不会变的，列索引会变为行索引的最内层
# 如果传入分层级别的编号或者名称
result.unstack(0) # 最外层变为列索引
result.unstack('state') # 等价操作
# 可能引入缺失数据，但可逆
# 可以用 dropna=False 留下na

ldata.pivot('date', 'item', 'value')
# item是列名，一列是重复数据，可以作为索引
# 如果有两个数据列，则会生成层级索引
```

数据转换

- `data.duplicate` 返回布尔型dataframe
- `data.drop_duplicate` 丢掉重复行
- `series.map()`
- `data.replace()`

```
data.drop_duplicate(['k1']) # 只看某个列的重复
data.replace([],[]) # 可以传入两个列表同时转换
data.replace({}) # 可以传入字典
```

重命名索引

- `data.index.map` 要赋值回去完成修改
- `data.rename(index=str.title, columns=str.upper)`

离散化和组划分

- `pd.cut(ages, bins)`

```
cats = pd.cut(ages, bins) # bins是一个列表，给定间隔，左开右闭
cats = pd.cut(ages, bins, labels=group_names) # 可以指定组名
cats = pd.cut(ages, 4,  precision=2) # 均匀分成四组
pd.value_counts(cat) # 分组计数
```

字符串方法

- `data.str.contains` 利用str属性访问pandas的字符串方法，并且是矢量化的字符串操作

数据分组和聚合运算

`拆分-应用-合并` 的思考和运作方式。

分组

groupby技术

- `df.groupby('key')`
- `df.groupby(df['key'])`
- `df.groupby(['key1', 'key2'])`

`goupby` 对象支持迭代，由分组名和数据块构成

```
for name, group in df.groupby('key1'):
    print(name)
    print(group)

for (k1, k2), group in df.groupby(['key1', 'key2']):
    print(k1, k2)
    print(group)

pieces = dict(list(df.groupby('key1')))
# 转换为词典有利于之后的操作
```

选取特定的一个或一组列

- `df.group('key1')['data1']` 一列
- `df.group('key1')[['data1', 'data2']]` 一组列
- 是**语法糖**

用字典和Series等映射也可以完成分组

通过函数分组

- `people.groupby(len).sum()`
- 返回值作为分组的名称

根据索引级别分组

- `data.groupby(level='city', aixs=1).count()`
- 传入level关键词参数，可以是索引名和层级编号

聚合

面向列使用多个自定义函数聚合

- `grouped.agg(f, 'mean')` 用 `'mean'` 表示求均值，传入的 `f` 是自定义的
- 可以自己给定列名，通过传入一个元组列表 `[('foo', 'mean'),('bar', np.std)]`
- 这时前面为**列名**，后面为**具体的函数**

`trasform` **方法** 这个方法把一个函数应用到各个分组，并**返回到原来适当的位置中**去

- `grouped.groupby(key).mean()`
- 输出结果为：

```
group1 1 1 1 1
group2 2 2 2 2
```

- `grouped.groupby(key).transform(np.mean)`
- 输出结果为：

```
group1 1 1 1 1
group2 2 2 2 2
group1 1 1 1 1
group2 2 2 2 2
```

`apply` **方法** 这是 一**般性** 的分组合并方法

```
# 例1，用各组平均值填充各组的缺失值
fill_mean = lambda g:g.fillna(g.mean())
data.groupby(key).apply(fill_mean)
```

时间序列

日期和时间数据类型及工具

```
from datetime import datetime
now = datetime.now()
delta = datetime(2011, 1, 7) - datetime(2008, 6, 24, 8, 15)
delta.days # 可以取出相隔的天数
delta.seconds # 可以取出相隔的秒数
```

字符串和 `datetime` **的相互转换**

- `stamp.strftime('%Y-%m-%d')` 输出
- `datetime.strptime(value, '%Y-%m-%d')` 从字符串 `value` 中读入
- `pd.to_datetime(datestr)` 利用 `pandas` 自带的模块

时间序列基础

索引，切片

- `ts['1/10/2011']` 利用解释为日期的字符串选取
- `longer['2001']` 较长的数据传入年轻松完成**切片**
- `longer['2001-05']` 年月完成切片

日期的范围、频率和移动

- `ts.resample('D')` 按照天重新设定频率
- `pandas.date_range('', '')` 生成指定长度的索引
- `ts.shift(3, freq='D')` ```py

默认freq参数是D

pd.date_range('1/1/2000', '12/1/2000', freq='BM')

BM表示 business end of month，设定了采样频率

freq = 'WOM-3FRI'

每月的第三个周三

ts.shift(3, freq='D')

不指定freq则索引不变，数据移动

指定则索引变化，数据不变

```
**锚点偏移量**
+ `MothEnd(2)` 之后的第二个月末
+ `MonthEnd().rollforward(now)`
偏移量可以**加在** `datetime` 对象上，加的是锚点偏移量，就会是原日期向前或向后滚动到下一个日期。
```py
now + MonthEnd(2)
# 之后的第二个月末

offset = MonthEnd()
offset.rollforward(now) # 向前滚动
offset.rollback(now) # 向后滚动
```

时期及其算术运算

用 `period` 表示时间区间，如某个月份，某个季度，并且可以进行算数运算。

- `p.asfreq('M', how='start')` 低转高
- 高转低时，由子时期所属的位置决定
- `ts.asfreq()` 可以对ts使用 ```py pd.Period(2007, freq='A-DEC')

一个Period对象

rng = pd.period_range('1/1/2000', '6/30/2000', freq='M')

一组Period，可以用作索引

ts.asfreq('M', how ='start')

改变series中的频率

```
**降采样、升采样**
+ 将高频转换为低频，并用某种方式**聚合**
+ `ts.resample('M', how='mean')`
+ `ts.resample('5min', how='ohlc')`
+ 这种重采样聚合方式可以得到**开盘最高最低收盘值**
+ ts.groupby(lambda x:x.month).mean()
+ 也可以利用groupby
+ ts.resample('D', fill_method='ffill')
+ 将低频转为高频叫做升采样， 插值方法和 `fillna` `reindex` 一样

### 移动窗口函数
移动窗口自动排除缺失值。
+ `pd.rolling_mean(series, 250, min_periods=10)`
+ 参数中指定了必须有10个非NA值，这是在求**250日均线**
+ `rolling_count` 非NA观测值
+ `rolling_sum` 移动窗口和
+ `rolling_meadian` 中位数
+ `rolling_apply` 对窗口应用普通数组函数
+ `ewma` 指数加权平均 **赋予近期更大的权数**
+ ...
+ **二元移动窗口函数**
+ `pd.rolling_corr(r1, r2, 125, min_periods=100)`
+ 可以用来计算与标普500的移动相关系数

---
```
```

```
## 金融和经济数据应用

### 数据规整化
**频率不同的时间序列的运算**
+ `resample` 将数据转换到固定频率，适合规整的索引
+ `reindex` 使数据符合一个新索引，适合不规整的索引
```py
ts1.resample('B', method='ffill')
# 规整数据
ts1.reindex(ts2.index, method='ffill')
# ts2的索引不规整
```

如果索引是 `period` 时间数列 此时和 `timestamp` 的时间序列不同， `Period` 索引 **必须** 进行显示转换。

- `infl.asfreq('Q-SEP', how='end')`
- 先将年化的infl索引转换为变成季度数据
- `infl.reindex(gdp.index, method='ffill')`
- 在使用重索引，将两个索引规整化

选取特定时间点

- `ts[time(10,0)]`
- 传入time对象就可以抽取时间点上的值
- `ts.at_time((time(10,0)))`
- 实际上是使用 `at_time` 方法
- 如果没有刚好落在时间点上的数据，就只能取最近的数据
- `selection = pd.date_range('', periods=4, freq='B')`
- `ts.asof(selection)`
- 传入的参数是要选取的日期范围和时间点
- 得到这些时间点或**之前最近**的**有效非NA**数据

拼接多个数据源

- `combine_first` 结合两个数据源取出非NA值
- `spliced.update(data2, overwrite=False)` 可以得出相同结果
- `overwrite=False` 只填补空白NA值
- `concat` 拼接

金融时间序列处理

有用的时间序列函数

- `series.pct_change()` 百分比变化。加一则为收益
- `series.cumprod()` 累计积，用于计算**累计收益**
- `series.cumsum()` 累计和
- `series.diff()` **一阶差分**

分组变化分析

- `zscore = lambda x:(x-x.mean())/x.std()`
- `by_industry.apply(zscore)`
- 分组分析

分组因子暴露

- `factors.corwith(port)`
- `pd.ols(y=port, x=factors).beta`
- 用最小二乘回归计算因子暴露
```