

Python程序设计

陈远祥

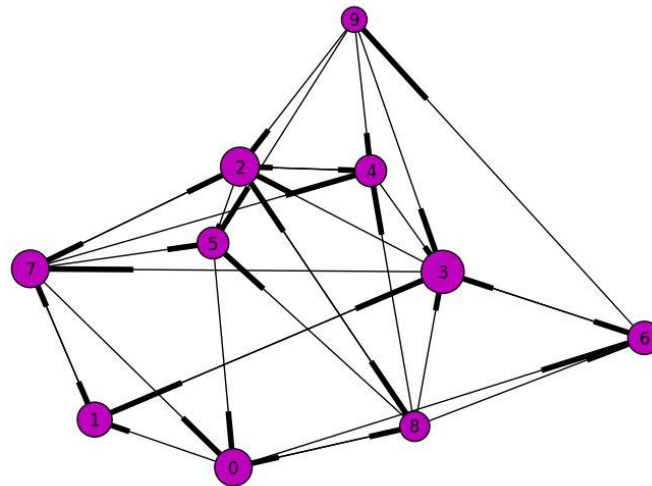
chenyxmail@gmail.com

北京邮电大学 电子工程学院





Pandas



Pandas

- Pandas是基于NumPy的一种工具，该工具是为了解决数据分析任务而创建的
- Pandas纳入了大量库和一些标准的数据模型，提供了高效地操作大型数据集所需的工具
- Pandas提供了大量能使我们快速便捷地处理数据的函数和方法

Pandas

- Pandas是Python的一个数据分析包，最初由AQR Capital Management于2008年4月开发，并于2009年底开源出来，目前由专注于Python数据包开发的PyData开发team继续开发和维护，属于PyData项目的一部分
- Pandas最初被作为金融数据分析工具而开发出来，因此，Pandas为时间序列分析提供了很好的支持

Pandas

- Pandas的名称来自于面板数据 (panel data) 和Python数据分析 (data analysis)
- panel data是经济学中关于多维数据集的一个术语，在Pandas中也提供了panel的数据类型

Pandas

■ Pandas的数据结构

- ✓ **Series**：一维数组，与Numpy中的一维array类似。二者与Python基本的数据结构List也很相近，其区别是：List中的元素可以是不同的数据类型，而Array和Series中则只允许存储相同的数据类型，这样可以更有效的使用内存，提高运算效率

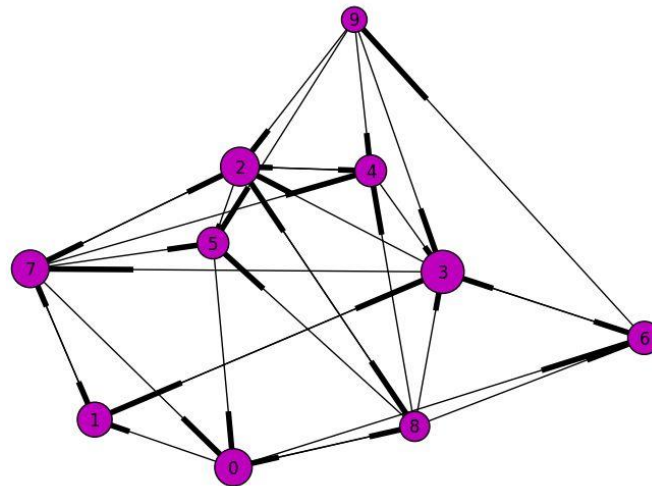
Pandas

■ Pandas的数据结构

- ✓ Time-Series: 以时间为索引的Series
- ✓ DataFrame: 二维的表格型数据结构。可以将DataFrame理解为Series的容器
- ✓ Panel: 三维的数组, 可以理解为DataFrame的容器



Series



Series

■ Series

- ✓ **Series**（序列）是一种类似于一维数组的对象，它由一组数据（各种NumPy数据类型）以及一组与之相关的数据标签（即索引）组成。仅由一组数据即可产生最简单的Series:

```
obj = Series([4, 7, -5, 3])
```

Series

■ Series

- ✓ Series的字符串表现形式为：索引在左边，值在右边。由于没有为数据指定索引，于是会自动创建一个0到N-1 (N为数据的长度) 的整数型索引。可以通过Series的values和index属性获取其数组表示形式和索引对象：
- ✓ obj.values
- ✓ obj.index

Series

■ Series

- ✓ 通常希望所创建的Series带有一个可以对各个数据点进行标记的索引：

```
obj2=Series([4, 2, -5, 3], index=['a', 'b', 'c', 'd'])
```

Series

- 与普通NumPy数组相比，可以通过索引的方式选取Series中的单个或一组值：
 - ✓ `obj2['a']`
 - ✓ `obj2['d'] = 6`
 - ✓ `obj2[['c', 'a', 'd']]`

Series

■ 字典 -> Series:

- ✓ 字典(dict)可以作为输入传递, 如果没有指定索引, 则按排序顺序取得字典键以构造索引。如果传递了索引, 索引中与标签对应的数据中的值将被拉出
- ✓ #dict
- ✓ #dict2

```
import pandas as pd
from pandas import Series, DataFrame
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s=Series(data)
print(s)
```

```
from pandas import Series, DataFrame
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
#索引顺序保持不变，缺少的元素使用NaN(不是数字)填充
s = Series(data,index=['b','c','d','a'])
print(s)
```

Series

■ 标量 -> Series:

- ✓ 如果数据是标量值，则必须提供索引。将重复该值以匹配索引的长度
- ✓ #标量

Series

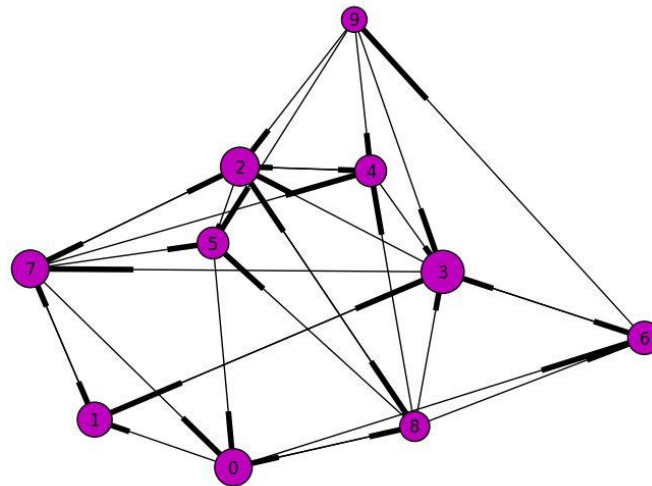
■ ndarray → Series:

- ✓ ndarray可直接转换为Series
- ✓ #ndarray

```
import pandas as pd
import numpy as np
data = np.random.randn(5) # 一维随机数
index = ['a', 'b', 'c', 'd', 'e'] # 指定索引
s = pd.Series(data, index)
#当我们非人为指定索引值时，Pandas会默认从0开始设置索引
#s = pd.Series(data)
print(s)
```




DataFrame



DataFrame

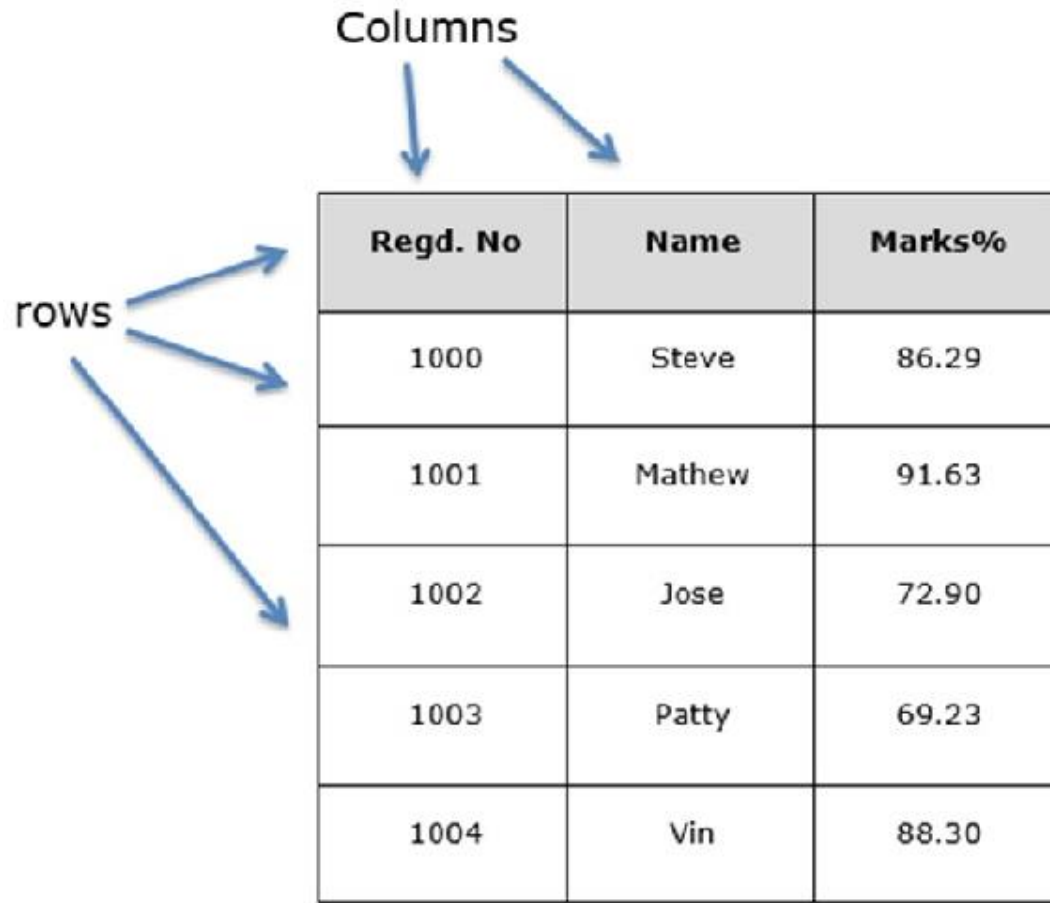
- 数据帧 (DataFrame) 是二维数据结构，即数据以行和列的表格方式排列
- DataFrame 可以被看成是以 Series 组成的字典。它和 Series 的区别在于，不但具有行索引，且具有列索引

DataFrame

- 数据帧 (DataFrame) 的功能特点:
 - ✓ 潜在的列是不同的类型
 - ✓ 大小可变
 - ✓ 标记轴 (行和列)
 - ✓ 可以对行和列执行算术运算

DataFrame

- 一个包含学生数据的数据帧：



The diagram illustrates a DataFrame as a table. The word "Columns" is positioned above the table with two arrows pointing to the "Regd. No" and "Name" headers. The word "rows" is positioned to the left of the table with three arrows pointing to the first three rows of data.

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

DataFrame

- Pandas 中的 DataFrame 可以使用以下构造函数创建：
 - ✓ `pandas.DataFrame(data, index, columns, dtype, copy)`

DataFrame

- Pandas 数据帧 (DataFrame) 可以使用各种输入创建
 - ✓ 列表
 - ✓ 字典
 - ✓ 序列
 - ✓ Numpy ndarrays
 - ✓ 另一个数据帧 (DataFrame)

DataFrame

■ Series字典 -> DataFrame:

✓ #Series字典

```
import pandas as pd
#带Series 的字典
d = {'one' : pd.Series([1., 2., 3.], index=['a', 'b', 'c']), 'two' : pd.Series([1., 2., 3., 4.], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d) # 新建 DataFrame
print(df)
```

DataFrame

■ ndarrays或lists字典 -> DataFrame:

✓ #list_dataframe

```
import pandas as pd
# 列表构成的字典
d = {'one' : [1, 2, 3, 4], 'two' : [4, 3, 2, 1]}
df1 = pd.DataFrame(d) # 未指定索引
df2 = pd.DataFrame(d, index=['a', 'b', 'c', 'd']) # 指定索引
print(df1)
print(df2)
```


DataFrame

■ 带字典的列表 -> DataFrame:

✓ #字典列表

```
import pandas as pd
# 带字典的列表
d = [{'a': 1, 'b': 2}, {'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(d)
print(df)
```

DataFrame

■ 列选择，添加，删除：

✓ #列选择

✓ #列添加列删除

DataFrame

- 将行标签传递给loc()函数来选择行
- 通过将整数位置传递给iloc()函数来选择行
- ✓ #行选择

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),
     'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c', 'd'])}
df = pd.DataFrame(d)
df2=df.loc['b']

print(df.iloc[2])
```

DataFrame

■ 行切片：

- ✓ 可以使用:运算符选择多行
- ✓ #行切片

DataFrame

■ 附加行:

- ✓ 使用 `append()` 函数将新行添加到 DataFrame
- ✓ #附加行

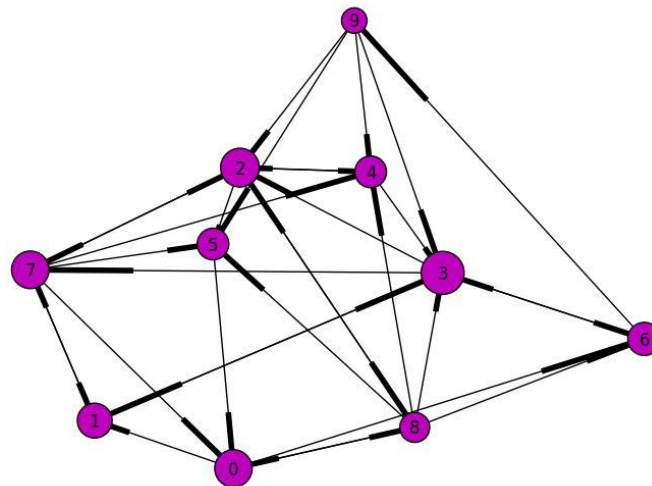
DataFrame

■ 删除行:

- ✓ 使用索引标签从DataFrame中删除或删除行。如果标签重复，则会删除多行
- ✓ #删除行



Panel I



Panel

- Panel（面板数据）：是Pandas中使用频率较低的一种数据结构，但它是三维数据的重要容器
- Panel data是计量经济学中衍生出来的一个概念。在计量经济学中，数据大致可分为三类：截面数据，时间序列数据，以及面板数据

Panel

- 截面数据指在某一时间点收集的不同对象的数据。而时间序列数据是指同一对象在不同时间点所对应的数据集合
- 面板数据即是截面数据与时间序列数据综合起来的一种数据类型

Panel

■ 一个城市和GDP关系

■ 截面数据：

- ✓ 例如城市：北京、上海、重庆、天津在某一年的GDP分别为10、11、9、8（单位亿元）

■ 时间序列数据：

- ✓ 例如：2000、2001、2002、2003、2004 各年的北京市GDP分别为8、9、10、11、12（单位亿元）。

■ 面板数据：

- ✓ 2000、2001、2002、2003、2004 各年中国所有直辖市的GDP分别为（单位亿元）：北京市分别为 8、9、10、11、12；上海市分别为 9、10、11、12、13；天津市分别为 5、6、7、8、9；重庆市分别为 7、8、9、10、11。

Panel

- Panel主要由三个要素构成：
 - ✓ items-axis 0: 每个项目 (item) 对应于内部包含的 DataFrame
 - ✓ major_axis-axis 1: 每个 DataFrame 的索引行
 - ✓ minor_axis-axis 2: 每个 DataFrame 的索引列
- 在Pandas中, 一个 Panel由多个DataFrame组成

Panel

- 可以使用以下构造函数创建面板：

`pandas.Panel (data, items, major_axis, minor_axis, dtype, copy)`

- 可以使用多种方式创建面板：

- ✓ 从ndarrays创建
- ✓ 从DataFrames的dict创建

Panel

■ 从3D ndarray创建

✓ 面板1

```
import pandas as pd
import numpy as np
data = np.random.rand(2,4,5)
p = pd.Panel(data)
print(p)
```

Panel

■ 从DataFrame对象的dict创建面板

✓ 面板2

```
import pandas as pd
import numpy as np
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
        'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print(p)
```

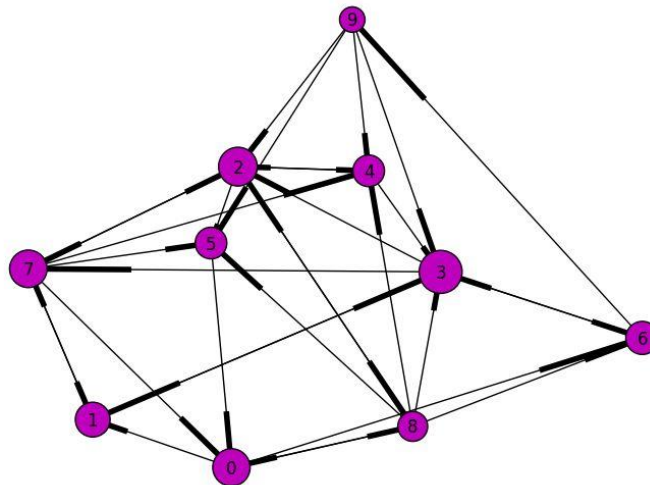
Panel

■ 从面板选择数据

- ✓ Items
- ✓ Major_axis
- ✓ Minor_axis
- ✓ #板选择数据



Pandas常用的基本方法



基本方法

- 数据读取与存储
- Head&Tail
- 统计方法
- 计算方法
- 标签对齐
- 排序

基本方法

- Pandas支持大部分常见数据文件读取与存储。一般情况下，读取文件的方法以`pd.read_`开头，而写入文件的方法以`pd.to_`开头

- #读取文件

文件格式	读取方法	写入方法
CSV	<code>read_csv</code>	<code>to_csv</code>
JSON	<code>read_json</code>	<code>to_json</code>
HTML	<code>read_html</code>	<code>to_html</code>
Local clipboard	<code>read_clipboard</code>	<code>to_clipboard</code>
MS Excel	<code>read_excel</code>	<code>to_excel</code>
HDF5 Format	<code>read_hdf</code>	<code>to_hdf</code>
Feather Format	<code>read_feather</code>	<code>to_feather</code>
Msgpack	<code>read_msgpack</code>	<code>to_msgpack</code>
Stata	<code>read_stata</code>	<code>to_stata</code>
SAS	<code>read_sas</code>	
Python Pickle Format	<code>read_pickle</code>	<code>to_pickle</code>
SQL	<code>read_sql</code>	<code>to_sql</code>
Google Big Query	<code>read_gbq</code>	<code>to_gbq</code>

基本方法

- Pandas提供了几个统计和描述性方法，方便你从宏观的角度去了解数据集：
 - ✓ `idxmin()` 和 `idxmax()` 会计算最小、最大值对应的索引标签
 - ✓ `count()` 用于统计非空数据的数量
 - ✓ `value_counts()` 仅仅针对 `Series`，它会计算每一个值对应的数量统计
 - ✓ #读取文件

基本方法

- 除了统计类的方法，Pandas还提供了很多计算类的方法：
 - ✓ `sum()` 用于计算数值数据的总和
 - ✓ `mean()` 用于计算数值数据的平均值
 - ✓ `median()` 用于计算数值数据的算术中值
 - ✓ ○ ○ ○ ○

基本方法

■ 标签对齐:

- ✓ 索引标签是Pandas中非常重要的特性，有些时候，由于数据的缺失等各种因素导致标签错位的现象，或者想匹配新的标签。于是Pandas提供了索引标签对齐的方法`reindex()`

基本方法

■ 标签对齐作用：

- ✓ 重新排序现有数据以匹配新的一组标签
- ✓ 在没有标签对应数据的位置插入缺失值 (NaN) 标记
- ✓ 特殊情形下，使用逻辑填充缺少标签的数据
(与时间序列数据高度相关)
- ✓ #标签对齐

```
import pandas as pd
#Series数据
s = pd.Series(data=[1, 2, 3, 4, 5], index=['a', 'b', 'c', 'd', 'e'])
print(s)
s1=s.reindex(['e', 'b', 'f', 'd'])

#DataFrame数据
df = pd.DataFrame(data={'one': [1, 2, 3], 'two': [4, 5, 6], 'three': [7, 8, 9]}, index=['a', 'b', 'c'])
print(df)
df1=df.reindex(index=['b', 'c', 'a'], columns=['three', 'two', 'one'])
```

基本方法

■ 排序：

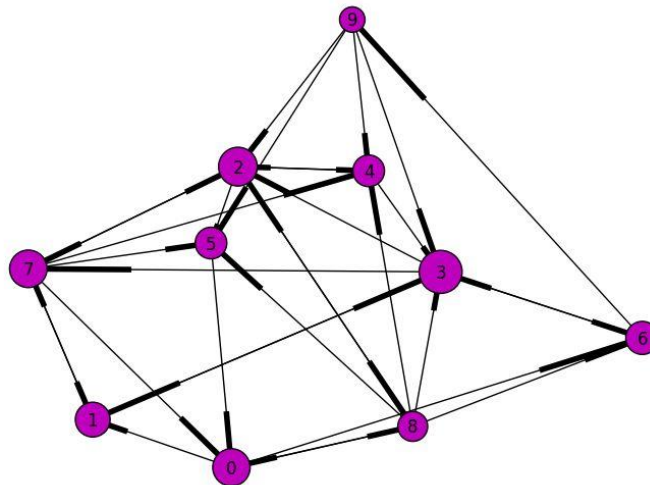
- ✓ 按索引排序
- ✓ 按数值排序
- ✓ #排序

```
import pandas as pd
df = pd.DataFrame(data={'one': [1, 2, 3], 'two': [4, 5, 6], 'three': [7, 8, 9], 'four': [10, 11, 12]}, index=['a', 'c', 'b'])
print(df)
#按索引排序
df1=df.sort_index()
df2=df.sort_index(ascending=False)

#按数值排序
df = pd.DataFrame(data={'one': [1, 2, 3, 7], 'two': [4, 5, 6, 9], 'three': [7, 8, 9, 2], 'four': [10, 11, 12, 5]}, index=['a', 'c', 'b', 'd'])
print(df)
#将第三列按照从小到大排序:
df1=df.sort_values(by='three')
```



Pandas数据选择与过滤



数据选择与过滤

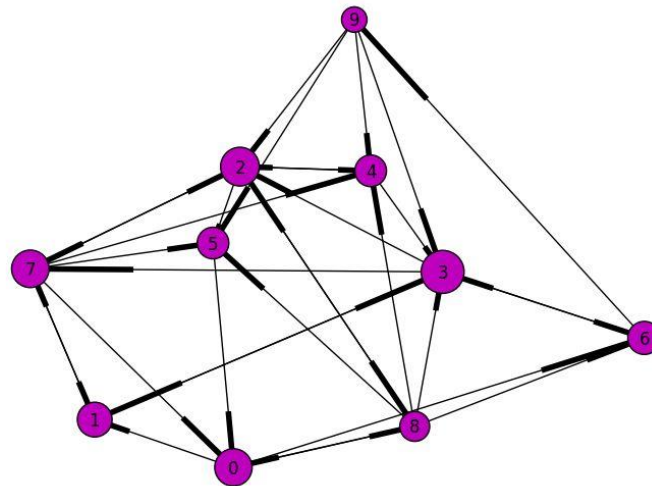
- 在数据预处理过程中，我们往往会对数据集进行切分，只将需要的某些行、列，或者数据块保留下来，输出到下一个流程中去。这也就是这里所说的数据选择

数据选择与过滤

- 基于索引数字选择
- 数据随机取样
- 条件语句选择
- `where()` 方法选择
- `query()` 方法选择



Pandas缺失值处理



缺失值处理

- 缺失值主要是指数据丢失的现象，也就是数据集中的某一块数据不存在。除此之外、存在但明显不正确的数据也被归为缺失值一类。例如，在一个时间序列数据集中，某一段数据突然发生了时间流错乱，那么这一小块数据就是毫无意义的，可以被归为缺失值
- 除了原始数据集就已经存在缺失值以外。当我们用到前面章节中的提到的索引对齐（`reindex()`，选择等）方法时，也容易人为导致缺失值的产生
 - ✓ #缺失值

缺失值处理

■ 缺失值处理包括：

- ✓ 缺失值标记
- ✓ 缺失值填充
- ✓ 缺失值插值

缺失值处理

- Pandas 为了更方便地检测缺失值，将不同类型数据的缺失均采用NaN标记。这里的NaN代表Not a Number，它仅仅是作为一个标记。例外是，在时间序列里，时间戳的丢失采用NaT标记
- Pandas 中用于检测缺失值主要用到两个方法，分别是：`isnull()`和`notnull()`，顾名思义就是「是缺失值」和「不是缺失值」。默认会返回布尔值用于判断
- #缺失值

缺失值处理

■ 插值:

- ✓ 插值是数值分析中一种方法。简而言之，就是借助于一个函数（线性或非线性），再根据已知数据去求解未知数据的值。插值在数据领域非常常见，它的好处在于，可以尽量去还原数据本身

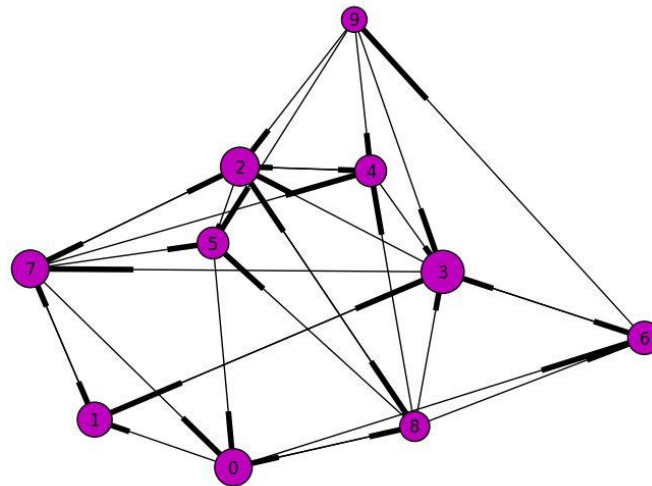
缺失值处理

■ 插值:

- ✓ Pandas中的插值, 通过 `interpolate()` 方法完成, 默认为线性插值, 即 `method='linear'`。除此之外, 还有 { 'linear', 'time', 'index', 'values', 'nearest', 'zero', 'slinear', 'quadratic', 'cubic', 'barycentric', 'krogh', 'polynomial', 'spline', 'piecewise_polynomial', 'from_derivatives', 'pchip', 'akima' } 等插值方法可供选择
- ✓ #插值



Pandas时间序列分析



时间序列分析

- 时间序列（英语：time series）是经济学的一种统计方法，它是采用时间排序的一组随机变量，国内生产总值（GDP）、消费者物价指数（CPI）、股价指数、利率、汇率等等都是时间序列。时间序列的时间间隔可以是分秒（如高频金融数据），可以是日、周、月、季度、年、甚至更大的时间单位

时间序列分析

- Pandas经常被用于处理与时间序列相关的数据，尤其是像财务数据。在处理时间序列数据时，会遇到各类需求，包括：
 - ✓ 生成固定跨度的时期构成时间序列
 - ✓ 将现有的时间序列，转换成需要的时间序列格式
 - ✓ 计算序列中的相对时间，例如：每季度的第一周

时间序列分析

- 时间戳: Timestamp
- 时间索引: DatetimeIndex, 时间戳构成
 - ✓ `date_range()` 创建一系列等间距时间:
 - ✓ `pandas.date_range(start=None, end=None, periods=None, freq='D', tz=None, normalize=False, name=None, closed=None, **kwargs)`
 - `start=`: 设置起始时间
 - `end=`: 设置截至时间
 - `periods=`: 设置时间区间, 若 `None` 则需要单独设置起止和截至时间。
 - `freq=`: 设置间隔周期, 默认为 `D`, 也就是天。可以设置为小时、分钟、秒等。
 - `tz=`: 设置时区。

时间序列分析

- 时间转换`to_datetime`是Pandas用于处理时间序列时的一个重要方法，它可以将实参转换为时间戳

```
pandas.to_datetime(arg, errors='raise', dayfirst=False, yearfirst=False, utc=None, box=True, format=None, exact=True, unit=None, infer_datetime_format=False, origin='unix' )
```

- ✓ #时间序列分析

```
import pandas as pd
t1=pd.Timestamp('2017-10-01')#时间戳

t2=pd.Timestamp('1/10/2017 13:30:59')
rng1 = pd.date_range('1/10/2017', periods=24, freq='H')

#时间转换

#输入标量
t1=pd.to_datetime('1/10/2017 10:00', dayfirst=True)

#输入列表
t2=pd.to_datetime(['1/10/2017 10:00','2/10/2017 11:00','3/10/2017 12:00'])

#输入series
t3=pd.to_datetime(pd.Series(['Oct 11, 2017', '2017-10-2', '3/10/2017']), dayfirst=True)

#输入dataframe
t4=pd.to_datetime(pd.DataFrame({'year': [2017, 2018], 'month': [9, 10], 'day': [1, 2], 'hour': [11, 12]}))

#errors参数
pd.to_datetime(['2017/10/1', 'abc'], errors='raise')
pd.to_datetime(['2017/10/1', 'abc'], errors='ignore')
pd.to_datetime(['2017/10/1', 'abc'], errors='coerce')
```

时间序列分析

■ 时间序列检索优点:

- ✓ 查找和检索特定日期的字段非常快
- ✓ 进行数据对齐时，拥有相同时间间隔的索引的数据将会非常快
- ✓ 可以很方便地通过shift和ishift方法快速移动对象

时间序列分析

■ 时间序列计算：

- ✓ 在Pandas中，包含有很多可以被加入到时间序列计算中去的类，这些被称为Offsets对象
- ✓ #时间序列对象


```
import pandas as pd
import numpy as np
from pandas.tseries import offsets # 载入 offsets
dt = pd.Timestamp('2017-10-1 10:59:59')
dt1=dt + offsets.DateOffset(months=1, days=2, hour=3) # 增加时间
dt2=dt - offsets.Week(3) # 减去 3 个周的时间

#shifting 可以将数据或者时间索引沿着时间轴的方向前移或后移
ts = pd.DataFrame(np.random.randn(7,2), columns=['Value1','Value2' ], index=pd.date_range('20170101', periods=7, freq='T'))
ts.shift(3) # 数据值向后移动了3行
ts.shift(-3) # 添加负号, 使得向前移动

#移动索引tshift()。
ts.tshift(3)
ts.shift(3,freq='D') # 日期向后移动3天
```

时间序列分析

■ 重采样Resample

- ✓ 重采样，即是将序列从一个频率转换到另一个频率的过程。实施重采样的情形如下：
- ✓ 时间序列数据集非常大，比如百万级别甚至更高。如果将全部数据用于后序计算，其实很多情况下是没有必要的。此时，我们可以对原有的时间序列进行降频采样
- ✓ 除了上面的情形，重采样还可以被用于数据对齐。比如，两个数据集，但是时间索引的频率不一致，这时候，可以通过重采样使二者频率一致，方便数据合并、计算等操作

谢谢