输入输出

Pickling

1. read_pickle(path)
   从指定的加载 pickled pandas 对象(或任何其他 pickled 对象)

Flat File

1. read_table(filepath_or_buffer [，sep，...])
   将一般分隔文件读入 DataFrame
2. read_csv(filepath_or_buffer [，sep，...])
   将 CSV(逗号分隔)文件读入 DataFrame
3. read_fwf(filepath_or_buffer [，colspecs，width])
   将固定宽度格式的行的表读入 DataFrame
4. read_msgpack(path_or_buf [，encoding，iterator])
   从指定的加载 msgpack pandas 对象

Clipboard

1. read_clipboard(\ * \ * kwargs)
   从剪贴板读取文本并传递给 read_table。

Excel

1. read_excel(io [，sheetname，headers，...])
   将 Excel 表读入 pandas DataFrame
2. ExcelFile.parse([sheetname，header，...]) （解析）
   将指定的工作表解析到 DataFrame 中

JSON

1. read_json([path_or_buf，orient，typ，dtype，...]) （转换为）
   将 JSON 字符串转换为 pandas 对象
2. json_normalize(data [，record_path，meta，...]) （平面表格）
   将规范化的半结构化 JSON 数据转换为平面表格

HTML

1. read_html(io [，match，flavor，header，...])
   将 HTML 表格读入 DataFrame 对象的 list。
2. read_hdf(path_or_buf [，key])
   从存储中读取，如果文件打开了，就关闭它
3. HDFStore.put(key，value [，format，append])
   存储对象到 HDFStore 中
4. HDFStore.append(key，value [，format，...])
   将内容添加到文件中的表。
5. HDFStore.get(key)
   获取存储在文件中的 pandas 对象
6. HDFStore.select(key [，where，start，stop，...])
   检索存储在文件中的 pandas 对象

SAS

1. read_sas(filepath_or_buffer [，format，...])
   读取以 XPORT 或 SAS7BDAT 格式文件存储的 SAS 文

SQL

1. read_sql_table(table_name，con [，schema，...])
   将 SQL 数据库表读入 DataFrame。
2. read_sql_query(sql，con [，index_col，...])
   将 SQL 查询读入 DataFrame。
3. read_sql(sql，con [，index_col，...])
   将 SQL 查询或数据库表读入一个 DataFrame。

Google BigQuery

1. read_gbq(query [，project_id，index_col，...])
   从 Google BigQuery 加载数据。
2. to_gbq(dataframe，destination_table，project_id)
   将 DataFrame 写入 Google BigQuery 表。

STATA

1. read_stata(filepath_or_buffer [，...])
   将 Stata 文件读入 DataFrame
2. StataReader.data(\ * \ * kwargs)
   DEPRECATED：从 Stata 文件读取观察结果，将它们转换为数据帧
3. StataReader.data_label()
   返回 Stata 文件的数据标签
4. StataReader.value_labels()
   返回一个 dict，关联每个变量名称一个 dict
5. StataReader.variable_labels()
   返回变量标签作为 dict，关联每个变量名
6. StataWriter.write_file()

一般操作

数据操作

1. melt(frame [，id_vars，value_vars，var_name，...])
   将 DataFrame 从宽格式“不透明”为长格式，可选择离开
2. pivot(index, columns, values)
   基于此 DataFrame 的 3 列生成'pivot'表。
3. pivot_table(data [，values，index，columns，...])
   创建一个电子表格样式的数据透视表作为 DataFrame。
4. crosstab(索引，列[，values，rownames，...])　计
   算两个(或更多)因子的简单交叉列表。
5. cut(x，bins [，right，labels，retbins，...])
   返回 x 的每个值所属的半开的索引。
6. qcut(x，q [，labels，retbins，precision])
   基于分位数的离散化函数。
7. merge(left，right [，how，on，left_on，...]
   通过按列或索引执行数据库样式的连接操作来合并 DataFrame 对象。
8. merge_ordered(left，right [，on，left_on，...])
   执行与为时序数据等有序数据设计的可选填充/插值合并。
9. merge_asof(left，right [，on，left_on，...])
   执行 asof 合并。

10. concat(objs [，axis，join，join_axes，...])
    沿着特定轴连接 pandas 对象，沿着其他轴连接可选的设置逻辑。
11. get_dummies(data [，prefix，prefix_sep，...])
    将分类变量转换为虚拟/指示符变量
12. factorize(values [，sort，order，...])
    将输入值编码为枚举类型或类别变量

顶层缺失数据

1. isnull(obj)
   检测缺失值(数值数组中的 NaN，对象数组中的无/ NaN)
2. notnull(obj)
   替换适用于对象数组的 numpy.isfinite / -numpy.isnan。

顶级转化

1. to_numeric(arg [，errors，downcast])
   将参数转换为数字类型。

顶级处理时间

1. to_datetime(\ * args，\ * \ * kwargs)
   将参数转换为 datetime。
2. to_timedelta(\ * args，\ * \ * kwargs)
   将参数转换为 timedelta
3. date_range([start，end，periods，freq，tz，...])
   返回固定频率日期时间索引，将日(日历)作为默认值
4. bdate_range([start，end，periods，freq，tz，...])
   返回固定频率 datetime 索引，以工作日为默认值
5. period_range([start，end，periods，freq，name])
   返回固定频率日期时间索引，将日(日历)作为默认值
6. timedelta_range([start，end，periods，freq，...])
   返回固定频率 timedelta 索引，以天为默认值
7. infer_freq(index [，warn])
   给定输入索引，推断最可能的频率。

顶级评估

1. eval(expr [，parser，engine，truediv，...])
   使用各种后端将 Python 表达式评估为字符串。

测试

1. test
   测试使用鼻子测试模块。

Series

Constructor

1. Series([data，index，dtype，name，copy，...])
   带轴标签的一维参考数组（包括时间序列）。

属性

索引：轴标签

1. Series.values
   返回 ndarray 或类 ndarray 的 Series
2. Series.dtype
   返回底层数据的 dtype 对象
3. Series.ftype
   返回数据是稀疏还是密集
4. Series.shape
   返回基础数据的形状的元组
5. Series.nbytes
   返回底层数据中的字节数
6. Series.ndim
   返回底层数据的维数，
7. Series.size
   返回底层数据中的元素数量
8. Series.strides
   返回基础数据的步幅
9. Series.itemsize
   返回底层数据项的 dtype 的大小
10. Series.base
    如果基础数据的内存是，则返回基础对象
11. Series.T
    返回转置
12. Series.memory_usage([index，deep])
    系列的内存使用情况

转换

1. Series.astype(dtype [，copy，raise_on_error])
   转换对象到输入的 numpy.dtype
2. Series.copy([deep])
   复制此对象数据。
3. Series.isnull()
   返回一个布尔大小相同的对象，指示值是否为 null。
4. Series.notnull()
   返回一个布尔大小相同的对象，指示这些值是否为空。

索引，迭代

1. Series.get(key [，default])
   从给定键的对象获取 item(DataFrame 列，面板切片等)。
2. Series.at
   基于快速标签的标量访问器
3. Series.iat
   快速整数位置标量存取器。
4. Series.ix
   主要是基于标签位置的索引器，具有整数位置后备。
5. Series.loc
   纯标签位置索引器，用于按标签选择。
6. Series.iloc

纯粹基于整数位置的索引，用于按位置选择。

7. Series.__iter__()
   提供对系列的值的迭代
8. Series.iteritems()
   Lazily 迭代(索引，值)元组

二进制运算符函数

1. Series.add(other[，level，fill_value，axis])
   将 Series 和 other 相加。
2. Series.sub(other[，level，fill_value，axis])
   将 series 和 other 相减。
3. Series.mul(other[，level，fill_value，axis])
   将 series 和 other 相乘。
4. Series.div(other[，level，fill_value，axis])
   将 series 和 other 相除。
5. Series.truediv(other [，level，fill_value，axis])
   将 series 和 other 执行浮点除法。
6. Series.floordiv(other[，level，fill_value，axis])
   将 series 和 other 执行整数除法。
7. Series.mod(other[，level，fill_value，axis])
   求模。
8. Series.pow(other[，level，fill_value，axis])
   求 series 和 other 的指数幂。
9. Series.radd(other[，level，fill_value，axis])
   相当于 series.add()，区别不明。
10. Series.rsub(other[，level，fill_value，axis])
    相当于 series.sub()，区别不明
11. Series.rmul(other[，level，fill_value，axis])
    相当于 series.mul()，区别不明
12. Series.rdiv(other[，level，fill_value，axis])
    相当于 series.div()，区别不明
13. Series.rtruediv(other [，level，fill_value，axis])
    相当于 series.truediv()，区别不明
14. Series.rfloordiv(other [，level，fill_value，...])
    相当于 series.floordiv()，区别不明
15. Series.rmod(other[，level，fill_value，axis])
    相当于 series.mod()，区别不明
16. Series.rpow(other[，level，fill_value，axis])
    相当于 series.pow，区别不明
17. Series.combine(other，func [，fill_value])
    使用给定的函数对两个 series 执行元素二进制操作
18. Series.combine_first(other)
    组合 Series 值，首先选择调用 Series 的值。
19. Series.round([decimals])
    将 Series 中的每个值四舍五入为给定的小数位数。
20. Series.lt(other[，level，fill_value，axis])
    Series 小于 other。
21. Series.gt(other[，level，fill_value，axis])
    Series 大于 other。
22. Series.le(other[，level，fill_value，axis])
    Series 小于等于 other。

23. Series.ge(other[，level，fill_value，axis])
    Series 大于等于 other。
24. Series.ne(other[，level，fill_value，axis])
    Series 不等于 other。
25. Series.eq(other [，level，fill_value，axis])
    Series 等于 other。

功能应用，**GroupBy**＆**Window**

1. Series.apply(func [，convert_dtype，args])
   对 Series 的值调用函数。
2. Series.map(arg [，na_action])
   使用输入对应关系(可以是可以是字典，系列或函数）
3. Series.groupby([by，axis，level，as_index，...])
   使用 mapper 的组系列(dict 或 key 函数，将给定函数应用于组，将结果返回为系列)或通过一系列列。
4. Series.rolling(window [，min_periods，freq，...])
   提供滚动窗口计算。
5. Series.expanding([min_periods，freq，...])
   提供扩展转换。
6. Series.ewm([com，span，halflife，alpha，...])
   提供指数加权函数

缺少数据处理

1. Series.dropna([axis，inplace])
   返回无 null 值的 Series
2. Series.fillna([value，method，axis，...])
   使用指定的方法填充 NA / NaN 值
3. Series.interpolate([method，axis，limit，...])
   根据不同的方法内插值。

整形，整理

1. Series.argsort([axis，kind，order])
   覆盖 ndarray.argsort。
2. Series.reorder_levels(order)
   使用输入顺序重新排列索引级别。
3. Series.sort_values([axis，ascending，...])
   按任一轴的值排序
4. Series.sort_index([axis，level，ascending，...])
   按标签(沿轴)对对象排序
5. Series.sortlevel([level，ascending，...])
   按所选级别对 MultiIndex 进行排序。
6. Series.swaplevel([i，j，copy])
   在 MultiIndex 中交换级别 i 和 j
7. Series.unstack([level，fill_value])
   Unstack，a.k.a.
8. Series.searchsorted(v [，side，sorter])
   查找要插入元素以维持顺序的索引。

组合**/**加入**/**合并

1. Series.append(to_append [，ignore_index，...])
   串联两个或更多 Series。
2. Series.replace([to_replace，value，inplace，...])

将'to_replace'中给出的值替换为'value'。

3. Series.update(other)
   使用通过的系列中的非 NA 值修改 Series。

时间序列相关

1. Series.asfreq(freq [，method，how，normalize])
   将 TimeSeries 转换为指定的频率。
2. Series.shift([periods，freq，axis])
   使用可选的时间频率按期望的周期数切换索引
3. Series.first_valid_index()
   返回第一个非 NA /空值的标签
4. Series.last_valid_index()
   返回最后一个非 NA /空值的标签
5. Series.tz_convert(tz [，axis，level，copy])
   将 tz 轴转换为目标时区。
6. Series.tz_localize(\ * args，\ * \ * kwargs)
   将 tz-naive TimeSeries 本地化为目标时区。