```
# -*- coding: utf-8 -*-
"""
程序开头注释
author: xingbuxing
date: 2017年04月23日
功能：本程序主要介绍pandas最最常用的一些方法。这些方法在之后的课程、作业中都会用到。
"""
import pandas as pd  # 将pandas作为第三方库导入，我们一般为pandas取一个别名叫做pd


# =====导入数据
df = pd.read_csv(
    # 该参数为数据在电脑中的路径，可以不填写
    filepath_or_buffer='/Users/jxing/Desktop/201704课程/20170423_class3/data/sz000002.csv',
    # 该参数代表数据的分隔符，csv文件默认是逗号。其他常见的是'\t'
    sep=',',
    # 该参数代表跳过数据文件的的第1行不读入
    skiprows=1,
    # nrows，只读取前n行数据，若不指定，读入全部的数据
    nrows=15,
    # 将指定列的数据识别为日期格式。若不指定，时间数据将会以字符串形式读入。一开始先不用。
    # parse_dates=['交易日期'],
    # 将指定列设置为index。若不指定，index默认为0, 1, 2, 3, 4...
    # index_col=['交易日期'],
    # 读取指定的这几列数据，其他数据不读取。若不指定，读入全部列
    usecols=['交易日期', '股票代码', '股票名称', '收盘价', '涨跌幅', '成交量', '新浪概念', 'MACD_金叉死叉'],
    # 当某行数据有问题时，报错。设定为False时即不报错，直接跳过该行。当数据比较脏乱的时候用这个。
    error_bad_lines=False,
    # 将数据中的null识别为空值
    na_values='NULL',

    # 更多其他参数，请直接搜索"pandas read_csv"，要去逐个查看一下。比较重要的，header等
)

# print df

# 使用read_csv导入数据非常方便

# 导入的数据的数据类型是DataFrame。

# 导入数据主要使用read系列函数
# 还有read_table、read_excel、read_json等，他们的参数内容都是大同小异，可以自行搜索查看。


# =====看数据
# print df.shape  # 输出dataframe有多少行、多少列。
# print df.shape[0]  # 取行数量，相应的列数量就是df.shape[1]
# print df.columns  # 顺序输出每一列的名字，演示如何for语句遍历。

# print df.index  # 顺序输出每一行的名字，可以for语句遍历。
```

```
# print df.dtypes  # 数据每一列的类型不一样，比如数字、字符串、日期等。该方法输出每一列变量类型
# print df.head(3)  # 看前3行的数据，默认是5。与自然语言很接近
# print df.tail(3)  # 看最后3行的数据，默认是5。
# print df.sample(n=3)  # 随机抽取3行，想要去固定比例的话，可以用frac参数
# print df.describe()  # 非常方便的函数，对每一列数据有直观感受；只会对数字类型的列有效

# 对print出的数据格式进行修正
# pd.set_option('expand_frame_repr', False)  # 当列太多时不换行
# pd.set_option('max_colwidth', 8) # 设定每一列的最大宽度，恢复原设置的方法，
pd.reset_option('max_colwidth')
# 更多设置请见http://pandas.pydata.org/pandas-docs/stable/options.html


# =====如何选取指定的行、列
# print df['股票代码']  # 根据列名称来选取，读取的数据是Series类型
# print df[['股票代码', '收盘价']]  # 同时选取多列，需要两个括号，读取的数据是DataFrame类型
# print df[[0, 1, 2]]  # 也可以通过列的position来选取

# loc操作：通过label（columns和index的名字）来读取数据
# print df.loc['12/12/2016']  # 选取指定的某一行，读取的数据是Series类型
# print df.loc['13/12/2016': '06/12/2016']  # 选取在此范围内的多行，和在list中slice操作类似，读取的
数据是DataFrame类型
# print df.loc[:, '股票代码':'收盘价']  # 选取在此范围内的多列，读取的数据是DataFrame类型
# print df.loc['13/12/2016': '06/12/2016', '股票代码':'收盘价']  # 读取指定的多行、多列。逗号之前是
行的范围，逗号之后是列的范围。读取的数据是DataFrame类型
# print df.loc[:, :]  # 读取所有行、所有列，读取的数据是DataFrame类型
# print df.at['12/12/2016', '股票代码']  # 使用at读取指定的某个元素。loc也行，但是at更高效。

# iloc操作：通过position来读取数据
# print df.iloc[0]  # 以index选取某一行，读取的数据是Series类型
# print df.iloc[1:3]  # 选取在此范围内的多行，读取的数据是DataFrame类型
# print df.iloc[:, 1:3]  # 选取在此范围内的多列，读取的数据是DataFrame类型
# print df.iloc[1:3, 1:3]  # 读取指定的多行、多列，读取的数据是DataFrame类型
# print df.iloc[:, :]  # 读取所有行、所有列，读取的数据是DataFrame类型
# print df.iat[1, 1]  # 使用iat读取指定的某个元素。使用iloc也行，但是iat更高效。


# =====列操作
# 行列加减乘除
# print df['股票名称'] + '_地产'  # 字符串列可以直接加上字符串，对整列进行操作
# print df['收盘价'] * 100  # 数字列直接加上或者乘以数字，对整列进行操作。
# print df['收盘价'] * df['成交量']  # 两列之间可以直接操作。收盘价*成交量计算出的是什么？
# 新增一列
# df['股票名称+行业'] = df['股票名称'] + '_地产'

# =====统计函数
# print df['收盘价'].mean()  # 求一整列的均值，返回一个数。会自动排除空值。
# print df[['收盘价', '成交量']].mean()  # 求两列的均值，返回两个数，Series
# print df[['收盘价', '成交量']]
# print df[['收盘价', '成交量']].mean(axis=1)  # 求两列的均值，返回DataFrame。axis=0或者1要搞清楚。
# axis=1，代表对整几列进行操作。axis=0（默认）代表对几行进行操作。实际中弄混很正常，到时候试一下就知道
了。
```

```
# print df['收盘价'].max()  # 最大值
# print df['收盘价'].min()  # 最小值
# print df['收盘价'].std()  # 标准差
# print df['收盘价'].count()  # 非空的数据的数量
# print df['收盘价'].median()  # 中位数
# print df['收盘价'].quantile(0.25)  # 25%分位数
# 肯定还有其他的函数计算其他的指标，在实际使用中遇到可以自己搜索


# =====shift类函数、删除列的方式
# df['昨天收盘价'] = df['收盘价'].shift(-1)  # 读取上一行的数据，若参数设定为3，就是读取上三行的数据；若参数设定为-1，就是读取下一行的数据；
# print df[['收盘价', '昨天收盘价']]
# del df['昨天收盘价']  # 删除某一列的方法

# df['涨跌'] = df['收盘价'].diff(-1)  # 求本行数据和上一行数据相减得到的值
# print df[['收盘价', '涨跌']]
# df.drop(['涨跌'], axis=1, inplace=True)  # 删除某一列的另外一种方式，inplace参数指是否替代原来的df
# print df
# df['涨跌幅_计算'] = df['收盘价'].pct_change(-1)  # 类似于diff，但是求的是两个数直接的比例，相当于求涨跌幅


# =====cum(cumulative)类函数
# df['成交量_cum'] = df['成交量'].cumsum()  # 该列的累加值
# print df[['成交量', '成交量_cum']]
# print (df['涨跌幅'] + 1.0).cumprod()  # 该列的累乘值，此处计算的就是资金曲线，假设初始1元钱。


# =====其他列函数
# df['收盘价_排名'] = df['收盘价'].rank(ascending=True, pct=False)  # 输出排名。ascending参数代表是顺序还是逆序。pct参数代表输出的是排名还是排名比例
# print df[['收盘价', '收盘价_排名']]
# del df['收盘价_排名']
# print df['股票代码'].value_counts()  # 计数。统计该列中每个元素出现的次数。返回的数据是Series


# =====筛选操作，根据指定的条件，筛选出相关拿数据。
# print df['股票代码'] == 'sh000002'  # 判断股票代码是否等于sz000002
# print df[df['股票代码'] == 'sz000002']  # 将判断为True的输出：选取股票代码等于sz000002的行
# print df[df['股票代码'].isin(['sz000002', 'sz000003 ', 'sz000004'])]  # 选取股票代码等于sz000002的行
# print df[df['收盘价'] >= 24.0]  # 选取收盘价大于24的行
# print df[(df.index >= '03/12/2016') & (df.index <= '06/12/2016')]  # 两个条件，或者的话就是|


# =====缺失值处理：原始数据中存在缺失值，如何处理？
# 删除缺失值
# print df.dropna(how='any')  # 将带有空值的行删除。how='any'意味着，该行中只要有一个空值，就会删除，可以改成all。
# print df.dropna(subset=['MACD_金叉死叉', '涨跌幅'], how='all')  # subset参数指定在特定的列中判断空值。

# all代表全部为空，才会删除该行；any只要一个为空，就删除该行。
```

```
# 补全缺失值
# print df.fillna(value='没有金叉死叉')  # 直接将缺失值赋值为固定的值
# df['MACD_金叉死叉'].fillna(value=df['收盘价'], inplace=True)  # 直接将缺失值赋值其他列的数据
# print df.fillna(method='ffill')  # 向上寻找最近的一个非空值，以该值来填充缺失的位置，全称forward fill，非常有用
# print df.fillna(method='bfill')  # 向下寻找最近的一个非空值，以该值来填充确实的位置，全称backward fill

# 找出缺失值
# print df.notnull()  # 判断是否为空值，反向函数为isnull()
# print df[df['MACD_金叉死叉'].notnull()]  # 将'MACD_金叉死叉'列为空的行输出


# =====排序函数
# df.reset_index(inplace=True)
# print df.sort_values(by=['交易日期'], ascending=1)  # by参数指定按照什么进行排序，acsending参数指定是顺序还是逆序，1顺序，0逆序
# print df.sort_values(by=['股票名称', '交易日期'], ascending=[1, 1])  # 按照多列进行排序


# =====两个df上下合并操作，append操作
# df.reset_index(inplace=True)
# df1 = df.iloc[0:10][['交易日期', '股票代码', '收盘价', '涨跌幅']]
# print df1
# df2 = df.iloc[5:15][['交易日期', '股票名称', '收盘价', '涨跌幅']]
# print df2
# print df1.append(df2)  # append操作，将df1和df2上下拼接起来。注意观察拼接之后的index
# df3 = df1.append(df2, ignore_index=True)  # ignore_index参数，用户重新确定index
# print df3


# =====对数据进行去重
# df3中有重复的行数，我们如何将重复的行数去除？
# df3.drop_duplicates(
#     subset=['收盘价', '交易日期'],  # subset参数用来指定根据哪类类数据来判断是否重复。若不指定，则用全部列的数据来判断是否重复
#     keep='first',  # 在去除重复值的时候，我们是保留上面一行还是下面一行？first保留上面一行，last保留下面一行，False就是一行都不保留
#     inplace=True
# )
# print df3


# =====其他常用重要函数
# print df.rename(columns={'MACD_金叉死叉': '金叉死叉', '涨跌幅': '涨幅'})  # rename函数给变量修改名字。使用dict将要修改的名字传给columns参数
# print df.empty  # 判断一个df是不是为空，此处输出不为空
# print pd.DataFrame().empty  # pd.DataFrame()创建一个空的DataFrame，此处输出为空
# print df.T  # 将数据转置，行变成列，很有用



# =====字符串处理
```

```
# print df['股票代码']
# print 'sz000002'[:2]
# print df['股票代码'].str[:2]
# print df['股票代码'].str.upper()  # 加上str之后可以使用常见的字符串函数对整列进行操作
# print df['股票代码'].str.lower()
# print df['股票代码'].str.len()  # 计算字符串的长度,length
# df['股票代码'].str.strip()  # strip操作，把字符串两边的空格去掉
# print df['股票代码'].str.contains('sh')  # 判断字符串中是否包含某些特定字符
# print df['股票代码'].str.replace('sz', 'sh')  # 进行替换，将sz替换成sh
# split操作
# print df['新浪概念'].str.split('；')  # 对字符串进行分割
# print df['新浪概念'].str.split('；').str[:2]  # 分割后取第一个位置
# print df['新浪概念'].str.split('；', expand=True)  # 分割后并且将数据分列
# # 更多字符串函数请见：http://pandas.pydata.org/pandas-docs/stable/text.html#method-summary


# =====时间处理
# 导入数据时将index参数注释掉

# df['交易日期'] = pd.to_datetime(df['交易日期'])  # 将交易日期由字符串改为时间变量
# print df['交易日期']
# print df.iloc[0]['交易日期']
# print df.dtypes
# print pd.to_datetime('1999年01月01日')  # pd.to_datetime函数：将字符串转变为时间变量
# print df.at[0, '交易日期']

# print df['交易日期'].dt.year  # 输出这个日期的年份。相应的month是月份，day是天数，还有hour，minute，
second
# print df['交易日期'].dt.week  # 这一天是一年当中的第几周
# print df['交易日期'].dt.dayofyear  # 这一天是一年当中的第几天
# print df['交易日期'].dt.dayofweek  # 这一天是这一周当中的第几天，0代表星期一
# print df['交易日期'].dt.weekday  # 和上面函数相同，更加常用
# print df['交易日期'].dt.weekday_name  # 和上面函数相同，返回的是星期几的英文，用于报表的制作。
# print df['交易日期'].dt.days_in_month  # 这一天是这一月当中的第几天
# print df['交易日期'].dt.is_month_end  # 这一天是否是该月的开头，是否存在is_month_end？
# print df['交易日期'] + pd.Timedelta(days=1)  # 增加一天，Timedelta用于表示时间差数据
# print (df['交易日期'] + pd.Timedelta(days=1)) - df['交易日期']  # 增加一天然后再减去今天的日期


# =====rolling、expanding操作
# 计算'收盘价'这一列的均值
# print df['收盘价'].mean()
# 如何得到每一天的最近3天收盘价的均值呢？即如何计算常用的移动平均线？
# 使用rolling函数
# df['收盘价_3天均值'] = df['收盘价'].rolling(5).mean()
# print df[['收盘价', '收盘价_3天均值']]
# rolling(n)即为取最近n行数据的意思，只计算这n行数据。后面可以接各类计算函数，例如max、min、std等
# print df['收盘价'].rolling(3).max()
# print df['收盘价'].rolling(3).min()
# print df['收盘价'].rolling(3).std()

# rolling可以计算每天的最近3天的均值，如果想计算每天的从一开始至今的均值，应该如何计算？

# 使用expanding操作
```

```
# df['收盘价_至今均值'] = df['收盘价'].expanding().mean()
# print df[['收盘价', '收盘价_至今均值']]

# expanding即为取从头至今的数据。后面可以接各类计算函数
# print df['收盘价'].expanding().max()
# print df['收盘价'].expanding().min()
# print df['收盘价'].expanding().std()

# rolling和expanding简直是为量化领域量身定制的方法，经常会用到。


# =====输出
# print df
# df.to_csv('output.csv', encoding='gbk', index=False)


# =====文档
# 以上是我认为最常用的函数
# 哪里可以看到全部的函数？http://pandas.pydata.org/pandas-docs/stable/api.html
# 一般的使用方法
```