

棕榈学院

10 天 Python 训练营讲义

第九讲



2018 年 10 月 22 日-2018 年 10 月 31 日

目录

一、从 Alpha Vantage 上获取数据

二、Apply

附：第八讲作业答案



首先载入需要的包：

```
import matplotlib.pyplot as plt
```

```
import seaborn as sns
```

```
import pandas as pd
```

```
import urllib.request
```

```
import json
```

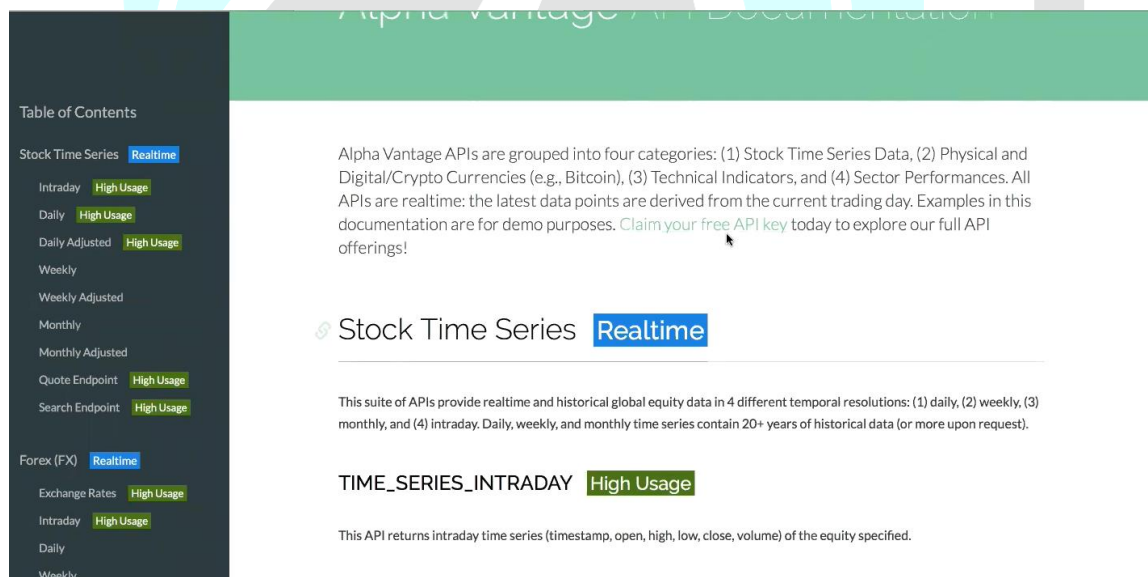
注：

- ① 用 urllib.request 访问某些 url 的数据。
- ② json 是一个常见的数据结构，很多时候我们通过 url 访问到的数据都是 json 结构的。

一、从 Alpha Vantage 上获取数据

数据来源网站：

<https://www.alphavantage.co/documentation/>



Alpha Vantage API Documentation

Table of Contents

- Stock Time Series **Realtime**
 - Intraday **High Usage**
 - Daily **High Usage**
 - Daily Adjusted **High Usage**
 - Weekly
 - Weekly Adjusted
 - Monthly
 - Monthly Adjusted
 - Quote Endpoint **High Usage**
 - Search Endpoint **High Usage**
- Forex (FX) **Realtime**
 - Exchange Rates **High Usage**
 - Intraday **High Usage**
 - Daily
 - Weekly

Alpha Vantage APIs are grouped into four categories: (1) Stock Time Series Data, (2) Physical and Digital/Crypto Currencies (e.g., Bitcoin), (3) Technical Indicators, and (4) Sector Performances. All APIs are realtime: the latest data points are derived from the current trading day. Examples in this documentation are for demo purposes. [Claim your free API key](#) today to explore our full API offerings!

Stock Time Series **Realtime**

This suite of APIs provide realtime and historical global equity data in 4 different temporal resolutions: (1) daily, (2) weekly, (3) monthly, and (4) intraday. Daily, weekly, and monthly time series contain 20+ years of historical data (or more upon request).

TIME_SERIES_INTRADAY **High Usage**

This API returns intraday time series (timestamp, open, high, low, close, volume) of the equity specified.

点击 “Claim your free API key”，获得自己的 API key。

这里可能会出现无法获取 API key 的情况，童鞋们连一下 VPN 就可以获取啦~

我们主要需要两种数据：TIME_SERIES_INTRADAY 和 TIME_SERIES_DAILY。
TIME_SERIES_INTRADAY 是指每一天的所有数据，它会精确到每一分钟。
TIME_SERIES_DAILY 是指每一天的数据。

TIME_SERIES_INTRADAY High Usage

This API returns intraday time series (timestamp, open, high, low, close, volume) of the equity specified.

API Parameters

■ Required: `function`

The time series of your choice. In this case, `function=TIME_SERIES_INTRADAY`

■ Required: `symbol`

询问的股价的名字

The name of the equity of your choice. For example: `symbol=MSFT`

■ Required: `interval` 输入你想要访问多长时间间隔的数据

Time interval between two consecutive data points in the time series. The following values are supported: `1min`, `5min`, `15min`, `30min`, `60min`

■ Optional: `outputsize` 访问的文件的大小 `compact` : 返回最近的100个数据点 ; `full` : 返回全部的数据点

By default, `outputsize=compact`. Strings `compact` and `full` are accepted with the following specifications: `compact` returns only the latest 100 data points in the intraday time series; `full` returns the full-length intraday time series. The "compact" option is recommended if you would like to reduce the data size of each API call.

■ Optional: `datatype` 获得数据的形式，本节课主要学习如何读取json形式的

By default, `datatype=json`. Strings `json` and `csv` are accepted with the following specifications: `json` returns the intraday time series in JSON format; `csv` returns the time series as a CSV (comma separated value) file.

■ Required: `apikey`

Your API key. Claim your free API key [here](#).

例：

```
timeseries = 'TIME_SERIES_INTRADAY'
ticker = 'AAPL'
interval = '1min'
size = 'compact'
api = '.....'
```

```
url =
'https://www.alphavantage.co/query?function=' + timeseries + '&symbol=' + t
icker + '&interval=' + interval + '&outputsize=' + size + '&apikey=' + api
f = urllib.request.urlopen(url)
dat = f.read()
js = dat.decode('utf8')
f.close()
js
```

[illegible]

注：

- ① 注意要切换到英文输入法下。
- ② 我们可以把很多东西都变成可以填入的变量，如 `function` 等，从而使用更加灵活，从而获得更多不同组合的数据。
- ③ 记得在 `api=`后填入自己的 API key
- ④ `f = urllib.request.urlopen(url)`指使用 `request` 函数打开了这个 url。
- ⑤ `dat = f.read()`指用 `.read()`这个函数打开了数据。
- ⑥ `js = dat.decode('utf8')`是指把字符标准化。
- ⑦ 这里还是字符串的形式，

```
parse_data = json.loads(js)
```

parse_data

[illegible]

注：

- ① 这里把字符串变成一个更好看的形式。
- ② json.loads()是指把json的数据读取之后，转化成为一个字典的样子。

```
print(type(parse_data))
```

```
parse_data.keys()
```

```
<class 'dict'>
```

```
Out[10]: dict_keys(['Meta Data', 'Time Series (1min)'])
```

注：

- ① 这里可以看到这个数据的类型已经变成了一个字典。
- ② 通过查看这个数据的键，我们可以发现数据被存为 2 部分，一部分是 Meta Data，另一部分是 Time Series(1min)。

```
ps = parse_data['Time Series (1min)']
```

```
df = pd.DataFrame.from_dict(ps,orient = 'index') #'column'
```

```
df.columns = ['Open','High','Low','Close','Volume']
```

```
df.head()
```

	Open	High	Low	Close	Volume
2018-10-23 14:21:00	220.4910	220.5575	220.3707	220.4400	69926
2018-10-23 14:22:00	220.3600	220.3600	220.1600	220.1600	57322
2018-10-23 14:23:00	220.1502	220.2600	220.1400	220.2500	47705
2018-10-23 14:24:00	220.1900	220.3800	220.1603	220.3100	44611
2018-10-23 14:25:00	220.3200	220.4800	220.2570	220.4423	40678

注：

- ① 这里通过 pd.DataFrame.from_dict()这个函数把字典转化成为一个 DataFrame。
- ② orien=是指字典中，我们要把每一个键当成 index 还是 column。如果不填这个参数，它会默认为 column name。

```
def import_data(timeseries, ticker, interval, size, key_name):
    if timeseries == 'TIME_SERIES_INTRADAY':
        url =
'https://www.alphavantage.co/query?function='+timeseries+'&symbol='+t
icker+'&interval='+interval+'&outputsize='+size+'&apikey='+api
        if timeseries == 'TIME_SERIES_DAILY':
            url =
'https://www.alphavantage.co/query?function='+timeseries+'&symbol='+t
icker+'&outputsize='+size+'&apikey='+api
            f = urllib.request.urlopen(url)
            dat = f.read()
            js = dat.decode('utf8')
            f.close()
            parse_data = json.loads(js)
            ps = parse_data[key_name]
            df = pd.DataFrame.from_dict(ps,orient = 'index') #'column'
            df.columns = ['Open','High','Low','Close','Volume']
            return df
```

注：

- ① 如果我们还需要其他的数据，这个我们可以先把它写成一个 function，后面就可以直接使用了。
- ② 首先我们可以将这个函数命名为 import data，在括号内输入变量的名字。
- ③ 每一天的数据和每一时刻的数据在 url 的写法上有不同，所以在这里我们可以添加一个条件。

```
timeseries = 'TIME_SERIES_DAILY'
ticker = 'AAPL'
interval = '1min'
size = 'compact'
key_name = 'Time Series (Daily)'
#api = '...'
```

```
apple = import_data(timeseries, ticker, interval, size, key_name)
apple.head()
```

	Open	High	Low	Close	Volume
2018-06-04	191.6350	193.4200	191.3500	191.8300	26266174
2018-06-05	193.0650	193.9400	192.3600	193.3100	21565963
2018-06-06	193.6300	194.0800	191.9200	193.9800	20933619
2018-06-07	194.1400	194.2000	192.3350	193.4600	21347180
2018-06-08	191.1700	192.0000	189.7700	191.7000	26656799

注：此时只需填入参数值，就可以获得数据了。

```
ticker = 'MSFT'
msft = import_data(timeseries, ticker, interval, size, key_name)
msft.head()
```

	Open	High	Low	Close	Volume
2018-06-04	101.2600	101.8600	100.8510	101.6700	27281623
2018-06-05	102.0000	102.3300	101.5300	102.1900	23514402
2018-06-06	102.4800	102.6000	101.9000	102.4900	21122917
2018-06-07	102.6500	102.6900	100.3800	100.8800	28232197
2018-06-08	101.0924	101.9500	100.5400	101.6300	22165128

注：这个 DataFrame 和我们前两天学过的 DataFrame 在数据类型上有不同，下面来看一下它的数据类型：

```
type(apple.iloc[0,0])
```

```
str
```

```
apple.iloc[0,0]
```

```
'191.6350'
```

注：里面的数据都还是字符串，index 也不是时间变量。

二、apply

把 DataFrame 里面长得像数字的字符串变成数字。

例 1：

```
df = pd.DataFrame([[1,2,3],[4,5,6],[7,8,9]], columns = ['A','B','C'])
```

df

	A	B	C
0	1	2	3
1	4	5	6
2	7	8	9

注：两个中括号代表矩阵。

```
import numpy as np
```

```
df.apply(np.sqrt)
```

	A	B	C
0	1.000000	1.414214	1.732051
1	2.000000	2.236068	2.449490
2	2.645751	2.828427	3.000000

注：在 apply 括号内填入数据后，这个函数会作用于 DataFrame 里面每一个数据。

```
df.apply(np.sum)
```

```
A    12  
B    15  
C    18  
dtype: int64
```

注：apply 会对每一列的数据进行一列一列的应用。

```
df.apply(np.sum, axis = 1)
```

```
0    6
1   15
2   24
dtype: int64
```

注：通过修改参数，把它应用于每一行。

df.apply(print)

```
0    1
1    4
2    7
Name: A, dtype: int64
0    2
1    5
2    8
Name: B, dtype: int64
0    3
1    6
2    9
Name: C, dtype: int64

Out[25]: A    None
        B    None
        C    None
        dtype: object
```

注：这里我们可以看清楚 apply 的应用方式。

df['A'].apply(print)

```
1
4
7

Out[26]: 0    None
        1    None
        2    None
        Name: A, dtype: object
```

注：对 DataFrame 里某单个的一列 apply，这里就是针对于每一个数据来执行了。

def diff(x):

 return max(x)-min(x)

注：我们也可以把 apply 放到一个自定义的函数里面。

df.apply(diff)

```
A    6
B    6
C    6
dtype: int64
```

df['A'].apply(diff)

```
TypeError                                Traceback (most recent call last)
<ipython-input-29-4dd2bd83c0b7> in <module>()
--> 1 df['A'].apply(diff)

/anaconda/lib/python3.6/site-packages/pandas/core/series.py in apply(self, func, convert_dtype, args, **kwargs)
   3192         else:
   3193             values = self.astype(object).values
-> 3194             mapped = lib.map_infer(values, f, convert=convert_dtype)
   3195
   3196             if len(mapped) and isinstance(mapped[0], Series):

pandas/_libs/src/inference.pyx in pandas._libs.lib.map_infer()

<ipython-input-27-88294b788af8> in diff(x)
     1 def diff(x):
--> 2     return max(x)-min(x)

TypeError: 'int' object is not iterable
```

注：把它填成单独的一列时，就出现了错误。因为这个函数需要输入的是一串数，如果输入的是一列数就没有意义了。

df['A'].map(np.sqrt)

```
0    1.000000
1    2.000000
2    2.645751
Name: A, dtype: float64
```

注：

- ① map 这个函数和 apply 用法很相似。
- ② map 和 apply 区别：map() is for Series (i.e. single columns) and operates on one cell at a time, while apply() is for DataFrame, and operates on a whole row at a time. map 更适用于 series，apply 更适用于 DataFrame.

apply (lambda x:)

例 2：

```
df.apply(lambda x: x+1)
```

	A	B	C
0	2	3	4
1	5	6	7
2	8	9	10

注：

- ① `apply (lambda x:)`是指在 `apply` 里面自定义方程。
- ② `x` 指的是每一列而不是每一个数字。

```
df[['A','B']] = df[['A','B']].apply(lambda x: x *2)
```

df

	A	B	C
0	2	4	3
1	8	10	6
2	14	16	9

注：更改其中的两列数字。

```
df.apply(lambda x: max(x) - min(x))
```

```
A      12  
B      12  
C       6  
dtype: int64
```

对 DataFrame 内每一个数字应用一些方程时，用 `apply` 更加方便快捷。

这里我们要把 DataFrame 里的字符串改成数字：

```
cols = apple.columns
```

```
apple[cols] = apple[cols].apply(pd.to_numeric, errors = 'coerce')
```

```
apple.index = pd.to_datetime(apple.index, format = '%Y-%m-%d')
```

```
apple.head()
```

	Open	High	Low	Close	Volume
2018-06-04	191.635	193.42	191.350	191.83	26266174
2018-06-05	193.065	193.94	192.360	193.31	21565963
2018-06-06	193.630	194.08	191.920	193.98	20933619
2018-06-07	194.140	194.20	192.335	193.46	21347180
2018-06-08	191.170	192.00	189.770	191.70	26656799

注：当我们应用 apply 时，所有要被填在函数里面的参数，要在 apply 括号内并列地填在函数的后面。pd.to_numeric, errors = 'coerce'本来应该是 pd.to_numeric(x,errors = 'coerce')。coerce 表示当有填不出的数据时，直接当成缺失值处理。

```
type(apple.iloc[0,0])
numpy.float64
```

注：这里检查一下，它已经变成一个字符串了。

```
def data_clean(df):
    cols = df.columns
    df[cols] = df[cols].apply(pd.to_numeric, errors = 'coerce')
    df.index = pd.to_datetime(df.index, format = '%Y-%m-%d')
    return df
```

注：这里我们可以把它设置成一个 format，后续用起来就更加方便了。

msft

Out[41]:

	Open	High	Low	Close	Volume
2018-06-04	101.2600	101.8600	100.8510	101.67	27281623
2018-06-05	102.0000	102.3300	101.5300	102.19	23514402
2018-06-06	102.4800	102.6000	101.9000	102.49	21122917
2018-06-07	102.6500	102.6900	100.3800	100.88	28232197
2018-06-08	101.0924	101.9500	100.5400	101.63	22165128
2018-06-11	101.3700	101.5900	100.6700	101.05	23490894
2018-06-12	101.1000	101.4493	100.7500	101.31	18325228
2018-06-13	101.7200	102.0100	100.5600	100.85	29492875
2018-06-14	101.6500	102.0300	101.0000	101.42	25691811
2018-06-15	101.5100	101.5300	100.0700	100.13	65738585

注：由于篇幅原因，这里没有截取全部数据。

```
m = data_clean(msft)
```

```
timeseries = 'TIME_SERIES_DAILY'
```

```
ticker_list = ['AAPL','MSFT','GOOG','FB']
```

```
interval = '1min'
```

```
size = 'compact'
```

```
key_name = 'Time Series (Daily)'
```

```
stock_list = {}
```

```
for ticker in ticker_list:
```

```
    temp_dat = import_data(timeseries, ticker, interval, size, key_name)
```

```
    stock_list[ticker] = data_clean(temp_dat)
```

注：

- ① 如果我们要读取同样类型的数据，可以把它写成一个循环。
- ② 如果我们需要应用了写好了的 format，第一步把它读进来，第二步填到 stock_list 里面。
- ③ 采用字典的原因是字典是一个很好的调用数据以及建立复杂数据的方式。

下面如果我们想要看 Facebook 的话，我们就直接填进去就可以了。

```
stock_list['FB']
```

```
Out[46]:
```

	Open	High	Low	Close	Volume
2018-06-04	191.8400	193.9800	191.4700	193.28	18939795
2018-06-05	194.3000	195.0000	192.6200	192.94	15544294
2018-06-06	191.0252	192.5300	189.1100	191.34	22558920
2018-06-07	190.7500	190.9700	186.7700	188.18	21503171
2018-06-08	187.5300	189.4754	186.4300	189.10	12677092
2018-06-11	188.8100	192.6000	188.8000	191.54	12928907
2018-06-12	192.1700	193.2800	191.5600	192.40	11562704
2018-06-13	192.7400	194.5000	191.9100	192.41	15853821
2018-06-14	193.1000	197.2800	192.9100	196.81	19120866
2018-06-15	195.7900	197.0700	194.6400	195.85	21860931
2018-06-18	194.8000	199.5800	194.1300	198.31	16826023
2018-06-19	196.2352	197.9600	193.7900	197.49	19993996
2018-06-20	199.1000	203.5500	198.8050	202.00	28230933

注：由于篇幅原因，这里没有截取全部数据。

今日作业：

```
import matplotlib.pyplot as plt
import seaborn as sns
import pandas as pd
import urllib.request
import json
```

Q1.

使用 api 从网上获得 AAPL 的 intraday data，时间间隔是 5min。并且把 index 变成时间变量。

```
apple.head()
# 这里同学们的数据在时间上可能跟我的不一样，因为是实时的。
```

	Open	High	Low	Close	Volume
2018-10-25 14:10:00	219.0700	219.7300	219.0300	219.6100	238556
2018-10-25 14:15:00	219.6900	219.8000	219.5600	219.7112	230211
2018-10-25 14:20:00	219.7500	219.9300	219.6800	219.8600	228136
2018-10-25 14:25:00	219.8500	219.9300	219.5900	219.8100	230370
2018-10-25 14:30:00	219.7978	220.2600	219.7978	220.1500	270164

Q2.

新建两列分别是日期，和时间。并且把 volume 这一列的变量类型变成数字

```
apple.head()
```

	Open	High	Low	Close	Volume	Date	Time
2018-10-25 14:10:00	219.0700	219.7300	219.0300	219.6100	238556	2018-10-25	14:10:00
2018-10-25 14:15:00	219.6900	219.8000	219.5600	219.7112	230211	2018-10-25	14:15:00
2018-10-25 14:20:00	219.7500	219.9300	219.6800	219.8600	228136	2018-10-25	14:20:00
2018-10-25 14:25:00	219.8500	219.9300	219.5900	219.8100	230370	2018-10-25	14:25:00
2018-10-25 14:30:00	219.7978	220.2600	219.7978	220.1500	270164	2018-10-25	14:30:00

Q3.

使用.pct_change()来得到交易量在下一个时间点和上一个时间点的百分比变化，并命名为新的一列['Volume_pct1'] 2.观察.shift(1) 和.shift(-1) 的用法，换一种方法得到交易量的百分比变化，并命名为['Volume_pct2'] 3.使用 apply(lambda x:)的写法，同样求交易量百分比的变化，并命名['Volume_pct3']

```
apple.head()
```

	Open	High	Low	Close	Volume	Date	Time	Volume_pct1	Volume_pct2	Volume_pct3
2018-10-25 14:10:00	219.0700	219.7300	219.0300	219.6100	238556	2018-10-25	14:10:00	NaN	NaN	NaN
2018-10-25 14:15:00	219.6900	219.8000	219.5600	219.7112	230211	2018-10-25	14:15:00	-0.034981	-0.034981	-0.034981
2018-10-25 14:20:00	219.7500	219.9300	219.6800	219.8600	228136	2018-10-25	14:20:00	-0.009013	-0.009013	-0.009013
2018-10-25 14:25:00	219.8500	219.9300	219.5900	219.8100	230370	2018-10-25	14:25:00	0.009792	0.009792	0.009792
2018-10-25 14:30:00	219.7978	220.2600	219.7978	220.1500	270164	2018-10-25	14:30:00	0.172740	0.172740	0.172740

注：如果有童鞋无法获得 API key，可以只做后两题，用第八讲发的数据就可以~

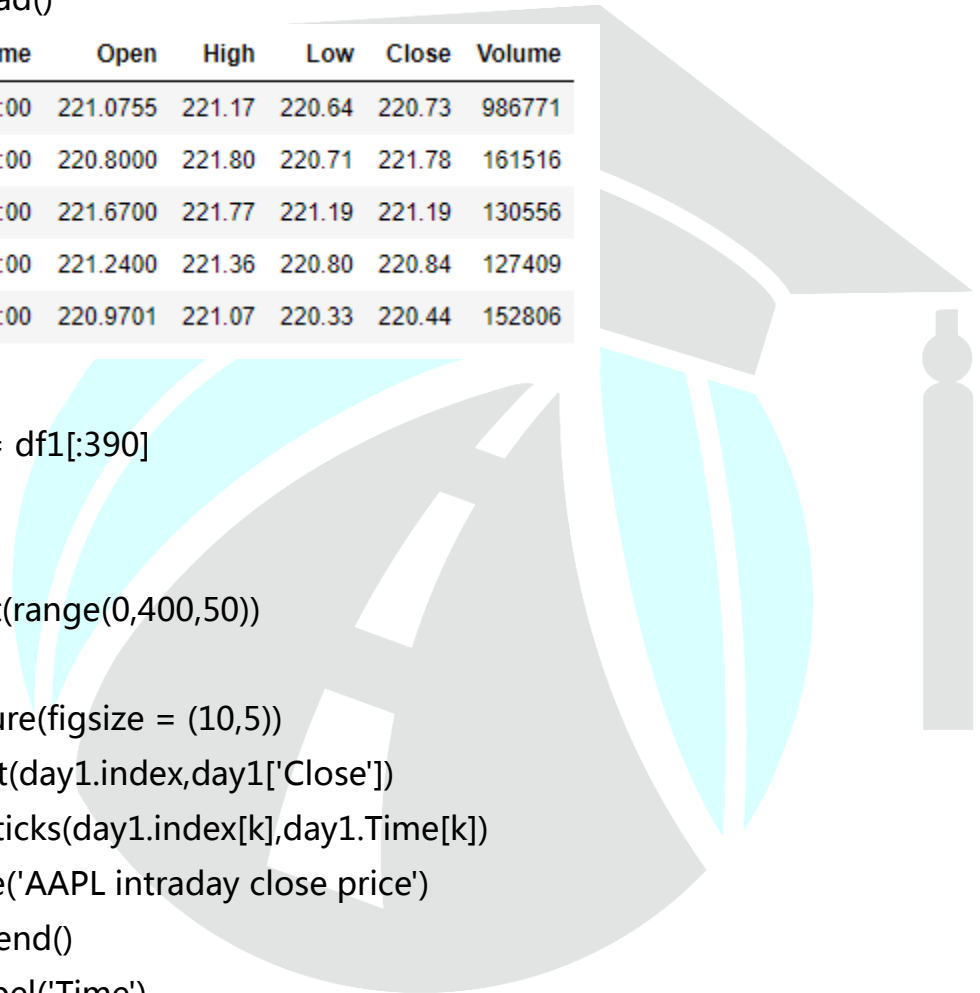
第八讲作业答案：

Q1.

```
df1 = pd.read_csv('AAPL_intraday_data.csv')
```

```
df1.columns = ['Time', 'Open', 'High', 'Low', 'Close', 'Volume']
```

```
df1.head()
```



	Time	Open	High	Low	Close	Volume
0	9:31:00	221.0755	221.17	220.64	220.73	986771
1	9:32:00	220.8000	221.80	220.71	221.78	161516
2	9:33:00	221.6700	221.77	221.19	221.19	130556
3	9:34:00	221.2400	221.36	220.80	220.84	127409
4	9:35:00	220.9701	221.07	220.33	220.44	152806

```
day1 = df1[:390]
```

Q2.

```
k = list(range(0,400,50))
```

```
plt.figure(figsize = (10,5))
```

```
plt.plot(day1.index,day1['Close'])
```

```
# plt.xticks(day1.index[k],day1.Time[k])
```

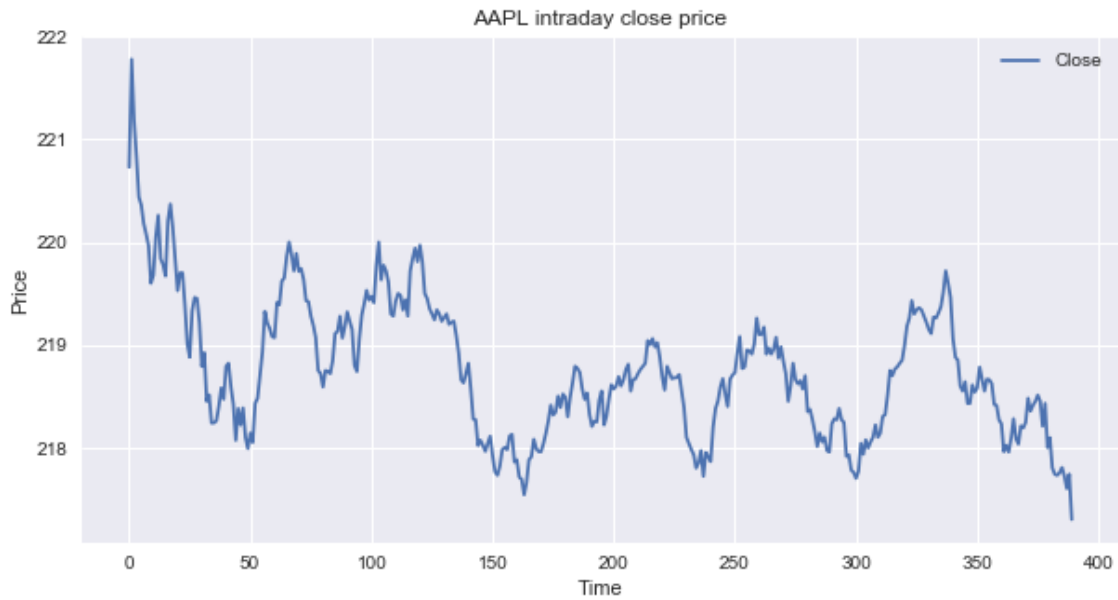
```
plt.title('AAPL intraday close price')
```

```
plt.legend()
```

```
plt.xlabel('Time')
```

```
plt.ylabel('Price')
```

```
plt.show()
```



```
day1['Time'] = pd.to_datetime(day1['Time'], format='%H:%M:%S').dt.time
```

```
datetime.time(9, 31)
```

```
plt.figure(figsize = (10,5))
plt.plot(day1['Time'],day1['Close'])
plt.title('AAPL intraday close price')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Price')
plt.show()
```



Q3.

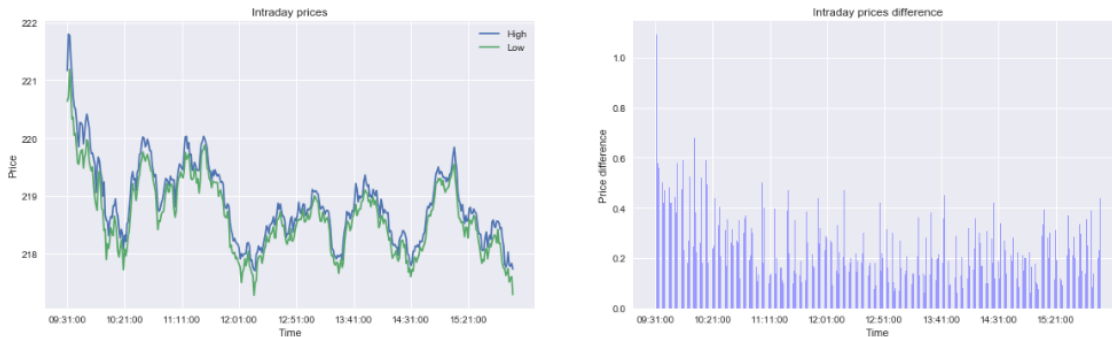
```
k = list(range(0,400,50))
fig = plt.figure(figsize = (20,12))

ax = fig.add_subplot(2,2,1)
day1.High.plot(label = 'High')
day1.Low.plot(label = 'Low')
plt.title('Intraday prices')
plt.xticks(day1.index[k],day1.Time[k])
plt.legend()
plt.xlabel('Time')
plt.ylabel('Price')
```

```
day1.diff = day1.High-day1.Low
ax = fig.add_subplot(2,2,2)
plt.bar(day1.index,day1.diff,color = '#9999ff',width = 0.5)
plt.title('Intraday prices difference')
plt.xticks(day1.index[k],day1.Time[k])
```

```
plt.xlabel('Time')  
plt.ylabel('Price difference')
```

```
plt.show()
```



注：

```
day1 = df1[:390]  
day1['Time'] = pd.to_datetime(day1['Time'], format = '%H:%M:%S').dt.time
```

```
k = list(range(0,400,50))  
fig = plt.figure(figsize = (20,12))
```

```
ax = fig.add_subplot(2,2,1)  
day1.High.plot(label = 'High')  
day1.Low.plot(label = 'Low')  
plt.title('Intraday prices')  
plt.xticks(day1.index[k],day1.Time[k])  
plt.legend()  
plt.xlabel('Time')  
plt.ylabel('Price')
```

```
day1.Diff = day1.High - day1.Low  
ax = fig.add_subplot(2,2,2)  
plt.bar(day1.Time,day1.Diff,color = '#9999ff',width = 0.5)  
plt.title('Intraday prices difference')
```

```
# plt.xticks(day1.index[k],day1.Time[k])
plt.xlabel('Time')
plt.ylabel('Price difference')
```

```
plt.show()
```

```
TypeError                                Traceback (most recent call last)
<ipython-input-32-212c7fd7c7eb> in <module>()
    17 day1.Diff = day1.High - day1.Low
    18 ax = fig.add_subplot(2,2,2)
--> 19 plt.bar(day1.Time,day1.Diff,color = '#9999ff',width = 0.5)
    20 plt.title('Intraday prices difference')
    21 # plt.xticks(day1.index[k],day1.Time[k])

/anaconda/lib/python3.6/site-packages/matplotlib/pyplot.py in bar(left, height, width, bottom, hold, data, **kwargs)
    2703     try:
    2704         ret = ax.bar(left, height, width=width, bottom=bottom, data=data,
-> 2705                     **kwargs)
    2706     finally:
    2707         ax._hold = washold

/anaconda/lib/python3.6/site-packages/matplotlib/_init__.py in inner(ax, *args, **kwargs)
    1890         warnings.warn(msg % (label_namer, func.__name__),
    1891                       RuntimeWarning, stacklevel=2)
-> 1892     return func(ax, *args, **kwargs)
    1893     pre_doc = inner.__doc__
    1894     if pre_doc is None:

/anaconda/lib/python3.6/site-packages/matplotlib/axes/_axes.py in bar(self, left, height, width, bottom, **kwargs)
    2103         if align == 'center':
    2104             if orientation == 'vertical':
-> 2105                 left = [left[i] - width[i] / 2. for i in xrange(len(left))]
    2106             elif orientation == 'horizontal':
    2107                 bottom = [bottom[i] - height[i] / 2.

/anaconda/lib/python3.6/site-packages/matplotlib/axes/_axes.py in <listcomp>(.0)
    2103         if align == 'center':
    2104             if orientation == 'vertical':
-> 2105                 left = [left[i] - width[i] / 2. for i in xrange(len(left))]
    2106             elif orientation == 'horizontal':
    2107                 bottom = [bottom[i] - height[i] / 2.

TypeError: unsupported operand type(s) for -: 'datetime.time' and 'float'
```

- ① 这里，有很多同学出现了上述问题，原因是，这里的 `datetime.time` 不可以被直接画图。我的理解是，没有日期的时间变量不可以被准确排序。
- ② 因为，只有时间的话，今天的下午 2 点和明天的下午 1 点无法比较。所以这里出现了这个问题。对于这样的问题，我觉得最简单的方法是把 `Time` 赋值给 `index`。
- ③ 这里，也有同学提问，为什么变成 `Index` 就可以了呢？因为 `index` 有明显的顺序，所以是可以被画在横坐标的。
- ④ 当然如果直接赋值成 `Index`，显示的时间因为是 Python 自己调节的，所以可能出现时间显示不完整的情况。如果希望显示好看的横坐标的话，我自己还是用

xticks 觉得比较方便。因为这样的话就是先把图画好，然后再把横坐标改成自己喜欢的就行了。

- ⑤ 另外，还有一种把 `datetime.time` 这个变量变成数字的方法，就是使用 `date2num` 这个函数。如果大家有兴趣的话可以了解一下~



扫码关注棕榈学院，解锁更多精彩课程