2022-2023学年度第一学期课程

Python and Data Science

Python与数据科学

新疆农业大学计算机与信息工程学院
cs.xjau.edu.cn

第9章　时间序列

教学内容

授课要点

- 时间序列介绍
- 时间序列的基本操作
- 时序分析模型ARIMA

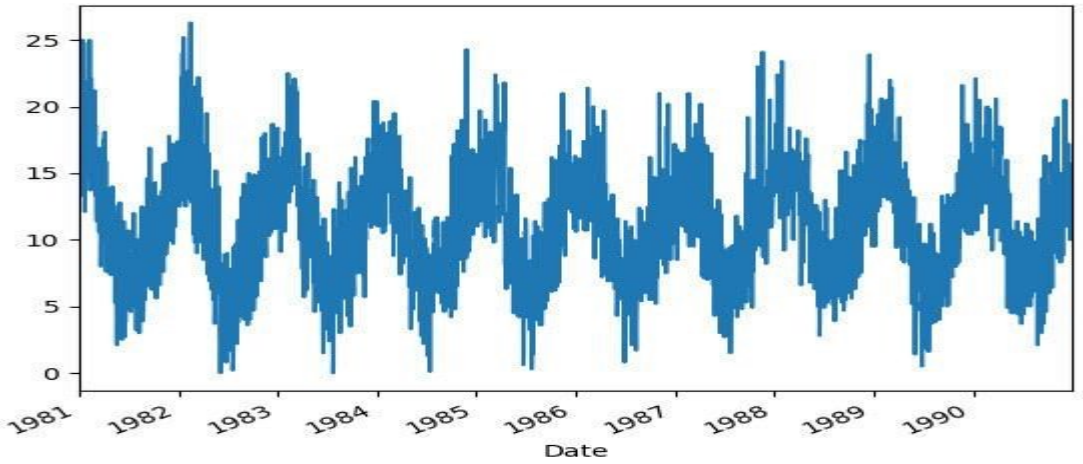
9.1 时间序列介绍

9.1.1 Python中的日期和时间数据类型

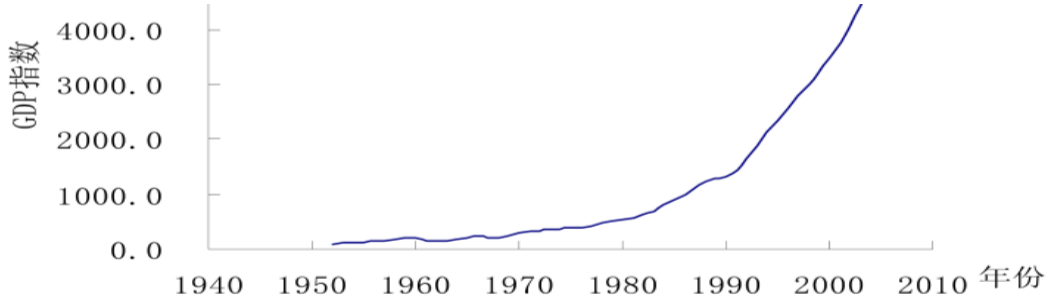
9.1.2 几种常用的时间序列

9.1 时间序列介绍

也称动态数列，它是把同一个统计指标的数值根据其发生的时间先后顺序进行排列而生成的数列。



十年间数千辆轻型汽车的销量的月度采样数据



1995年至2005年我国GDP指数的时间序列图。

9.1.1 Python中的日期和时间数据类型

在Python中，用来表示日期(date)和时间(time)数据的数据类型都包含在标准库中，常用的模块有time、datetime、calendar等。

date日期存储年、月、日信息；
time时间存储小时、分钟、秒、毫秒信息;

除此之外，pandas提供了许多内置的时间序列处理工具和数据算法。

9.1.2 常用的时间序列

时间序列可以是固定频率的，也可以是不定期的，这种数据的意义主要取决于具体的应用场景，通常有以下几种常用的时间序列：

- 时间戳 timestamp
- 时期 period
- 时间间隔 interval

9.1.2 常用的时间序列

- **时间戳 timestamp**

时间戳是指特定的时刻，这个特定时刻并不是我们日常生活中常描述的可以精确到秒的一个时间，而是一个float型数据。它表示从格林威治时间1970年1月1日(00:00:00 GMT)至当前时间的总毫秒数。

```
time.time()
```

9.1.2 常用的时间序列

- **时期period**

时期表示的是时间区间，通常描述为数日、数月、数季、数年等。

```
pd.Period(timestamp，freq)
```

◆功能：创建一个时期

◆参数1：表示一个时间戳

◆参数2freq：用于指明该时间段的长度

9.1.2 常用的时间序列

```
import pandas as pd
#生成一个以2022-01开始，月为频率的时间构造器
print( pd.Period( '2022', freq = 'M' ))
```

运行结果：

2022-01

选项	说明
B	工作日
C	用户自定义工作日
D	日历日
W	周，另外W还可以和SUN、MON、TUE、WED、THU、FRI、SAT组合，比如W-TUE表示以周二为一个星期的最后一天

9.1.2 常用的时间序列

- **时间间隔 interval**

通常用在表示实验或过程持续的时间。 时期period可以被看做间隔的特例。可以计算两个日期之间相差的时间。

```
import datetime
d1 = datetime.datetime.strptime('2022-03-05 17:41:20',
                                '%Y-%m-%d %H:%M:%S')
d2 = datetime.datetime.strptime('2022-03-02 17:41:20',
                                '%Y-%m-%d %H:%M:%S')
delta = d1 - d2
print(delta.days)
```

运行结果：

3

9.2 常用时间序列分析工具

9.2.1 time模块

9.2.2 datetime模块

9.2.3 pandas处理时间序列

9.3.4 其它时间处理模块

9.2.1 time模块

time 模块中只有time.struct_time一个类。

struct_time是一个转换秒数得到的结构化的时间对象，可以通过下标或属性名称获取对象的年、月、日、时、分、秒等属性。

调用gmtime()，localtime()，strptime()等方法可得到struct_time实例。

9.2.1 time模块

```
import time
st = time.localtime()
print(st)
#输出时间元组的一项tm_mon
print("tm_mon = ",st.tm_mon)
#可利用元组下标获取指定数据项，下标序号为[0，8]
print("st[1] = ",st[1])
```

9.2.1 time模块

- strftime()函数

strftime(format)

- 功能：将日期和时间对象转换为字符串形式
- 参数format：格式化代码输入（一个或多个）
- 返回字符串表示形式

9.2.1 time模块

格式代码列表

通配符	含义
%y	两位数的年份表示（00-99）
%Y	四位数的年份表示（000-9999）
%m	月份（01-12）
%d	月内中的一天（0-31）
%H	24小时制小时数（0-23）
%I	12小时制小时数（01-12）
%M	分钟数（00=59）
%S	秒（00-59）
%a	本地简化星期名称
%A	本地完整星期名称
%b	本地简化的月份名称
%B	本地完整的月份名称
%c	本地相应的日期表示和时间表示
%j	年内的一天（001-366）
%p	本地A.M.或P.M.的等价符
%U	一年中的星期数（00-53）星期天为星期的开始
%w	星期（0-6），星期天为星期的开始
%W	一年中的星期数（00-53）星期一为星期的开始
%x	本地相应的日期表示
%X	本地相应的时间表示
%Z	当前时区的名称
%%	%号本身

9.2.1 time模块

- strptime()函数

```
time.strptime(string ,format)
```

  - 功能：按照特定时间格式将字符串转换为时间类型
  - 参数1：要转成日期的字符串
  - 参数2：可缺省，表示字符串显示的格式

9.2.1 time模块

```
import time
# 格式化成2022-08-31 09:42:51形式
print('当前时间：',time.strftime("%Y-%m-%d %H:%M:%S",
      time.localtime()))
# 格式化成Wed Aug 31 09:42:51 2022形式
print('当前时间：',time.strftime("%a %b %d %H:%M:%S %Y",
      time.localtime()))
# 将格式字符串转换为时间戳
a = "Wed Oct 27 19:56:36 2021"
print(time.mktime(time.strptime(a,%a %b %d %H:%M:%S %Y")))
```

例9-2

9.2.2 datetime模块

datetime是Python处理日期和时间的标准库，其作用是对日期和时间类型的数据进行操作。包含5个类。

类名	功能
date	表示以公历形式存储的日期（年、月、日）
time	表示时间（时、分、秒、毫秒）
datetime	表示时间和日期
timedelta	表示两个datetime对象的差值
tzinfo	表示时区的相关信息

9.2.2 datetime模块——date类

- date类

date类有三个参数,datetime.date(year,month,day)，分别表示年、月、日，返回year-month-day。

```
import datetime
t = datetime.date(2022,7,26)
print(t)
print('年：',t.year,'月：',t.month,'日：',t.day)
```

运行结果:

2022-07-26

年： 2022 月： 7 日： 26

9.2.2 datetime模块——date类

- date类

常用方法：

- date.ctime()
- date.isoformat()
- date.strftime(format)
- date.fromtimestamp(timestamp)
- date.isocalendar()
- date.isoweekday()
- date.weekday()
- date.replace(year,month,day)
- date.timetuple()

9.2.2 datetime模块——date类

- date.ctime()：以历元以来的秒为参数，返回一个表示本地时间的字符串。
- date.isoformat()：返回标准日期格式。
- date.strftime(format)：将日期按照给定的format进行格式化。
- date.fromtimestamp(timestamp)：根据给定的时间戳，返回一个date对象，与datetime.date.today()作用相同。
- date.replace(year,month,day)：替换给定日期，但不改变原日期。
- date.timetuple()：返回日期对应的time.struct_time对象。
- date.isocalendar()：此方法的返回类型是一个元组，它告诉我们指定日期的年份，第几周和该周的第几个工作日。
- date.isoweekday()：返回日期是星期几，返回值是[1, 7], 1表示星期一。
- datetime.date.weekday()：返回日期是星期几，返回值是[0, 6]，0表示星期一。

9.2.2 datetime模块——time类

（2）time类

datetime.time(hour,minute,second,microsecond,tzinfo)

- hour　时
- minute　分
- second　秒
- microsecond　微秒
- tzinfo　返回传递给time构造方法的tzinfo对象，如果该参数未给出，则返回None

```
import datetime
time = datetime.time(12,6,9)
print('时:',time.hour,' 分:',time.minute,' 秒:',time.second)
```

运行结果:

时: 12　分: 6　秒: 9

9.2.2 datetime模块——time类

常用方法：

- time.replace()
- time.strftime(format)
- time.isoformat()
- time.tzname()

- time.replace()：生成一个新的时间对象，用参数指定的时、分、秒、微秒代替原有对象中的属性。(原有对象仍保持不变)
- time.strftime(format):按照format格式返回时间。
- time.isoformat()：返回标准时间格式。

9.2.2 datetime模块——time类

```
import datetime
time = datetime.time(12,6,9)
new_time = time.replace(13,5,8)
print('原时间：',time)
print('新时间：',new_time)
print('指定格式：',new_time.strftime("%H %M %S"))
print('标准格式：',time.isoformat())
```

运行结果：

原时间： 12:06:09

新时间： 13:05:08

指定格式： 13 05 08

标准格式： 12:06:09

9.2.2 datetime模块——time类

- time.tzname()：用于以字符串形式返回传递的datetime对象的时区名称。

返回无tzinfo() 的时区信息

为亚洲/上海添加时区并获取时区详细信息

```
import datetime
import pytz
dt = datetime.datetime.now()
print(dt)
print(dt.tzinfo)
print("时区:", dt.tzname())
print()
timezone = pytz.timezone("Asia/ShangHai")
mydt = timezone.localize(dt)
print(mydt)
print("Tzinfo:", mydt.tzinfo)
print("时区:", mydt.tzname())
```

运行结果：

2022-09-01 13:51:54.853444

None

时区: None

2022-09-01 13:51:54.853444+08:00

Tzinfo: Asia/Shanghai

时区: CST

9.2.3 datetime模块——datetime类

（3）datetime类

```
import datetime
dt_c = datetime.datetime.today()
print(dt_c)
print('年:', dt_c.year, '月:', dt_c.month, '日:', dt_c.day)
print('时:', dt_c.hour, '分:', dt_c.minute,
      '秒:', dt_c.second, '微秒:', dt_c.microsecond)
print('时区信息:', dt_c.tzinfo)
```

- second　秒
- microsecond　微秒
- tzinfo　返回传递给time构造方法
  则返回None

运行结果：

2022-09-02 20:23:54.908927

年: 2022　月: 9　日: 2

时: 20　分: 23　秒: 54　微秒: 908927

时区信息: None

9.2.3 datetime模块——datetime类

（3）datetime类

常用方法：

- datetime.ctime()
- datetime.fromtimestamp(timestamp)
- datetime.replace()
- datetime.strftime()
- datetime.now()
- datetime.now().date()
- datetime.now().time()

9.2.3 datetime模块——datetime类

（3）datetime类

```
import datetime
date = datetime.datetime(2022,7,17,17,58,34)
date_array = date.ctime()
print('字符串类型：',date_array)
time = datetime.datetime.fromtimestamp(1658051914)
print('时间戳“1658051914”所对应的datetime对象：')
print(time)
new_time = time.replace(2022
print('原时间：',time)
print('新时间：',new_time)
new_time2 = new_time.strftim
print('新时间的指定格式：',new_
```

运行结果：

字符串类型： Sun Jul 17 17:58:34 2022

时间戳“1658051914”所对应的datetime对象：

2022-07-17 17:58:34

原时间： 2022-07-17 17:58:34

新时间： 2022-08-30 14:12:12

新时间的指定格式： Aug 30 2022 14:12:12

9.2.3 datetime模块——datetime类

（3）datetime类

- datetime.now()：获取当前日期和时间
- datetime.now().date()：获取当前日期时间的日期部分
- datetime.now().time()：获取当前日期时间的时间部分

```
import datetime
date_time = datetime.datetime.now()
print('当前日期时间：',date_time)
date = datetime.datetime.now().date()
print('日期部分：',date)
time = datetime.datetime.now().time()
print('时间部分：',time)
```

运行结果：

当前日期时间： 2022-09-01 14:05:44.117343

日期部分： 2022-09-01

时间部分： 14:05:44.117343

9.2.4 datetime模块——timedelta类

（4）timedelta类

datetime.timedelta表示两个时间的差值。
实例化timedelta类时参数默认是0，依次是：

- days 时间差的天数
- seconds 时间差的秒数
- milliseconds 时间差的毫秒数
- microseconds 时间差的微秒数
- minutes 时间差的分钟数
- hours 时间差的小时数
- weeks 时间差的周数

9.2.4 datetime模块——timedelta类

（4）timedelta类

```
from datetime import datetime, timedelta
current_datetime = datetime.now()
print('当前时间:', current_datetime)
# 过去的时间
three_days_before_date = current_datetime - timedelta(days=3)
print('三天前时间:', three_days_before_date)
# 未来的时间
one_year_future_date = curren
four_weeks_future_date = curr
three_minutes_future_date = c
                            t
print('一年后时间:', one_year_f
print('三周后时间',four_weeks_
print('三分钟后时间',three_min
```

运行结果：

当前时间: 2022-08-31 12:35:15.640002

三天前时间: 2022-08-28 12:35:15.640002

一年后时间: 2023-08-31 12:35:15.640002

三周后时间 2022-09-21 12:35:15.640002

三分钟后时间 2022-08-31 12:38:15.640002

9.2.4 datetime模块——tzinfo类

（5）tzinfo类

tzinfo 是作为时区的类，在python中用pytz来转换模块。

```
import pytz
dt =
datetime.datetime.fromtimestamp(1405938446,
                                pytz.timezone('UTC'))
print(dt.tzinfo)
```

运行结果：

UTC

9.2.3 PANDAS处理时间序列

pandas库是处理时间序列的利器，pandas有着强大的日期数据处理功能，可以按日期筛选数据、按日期显示数据、按日期统计数据。常用的方法有：

pd.to_datetime()

pd.to_period()

pd.date_range()

pd.period_range()

resample()

9.2.3 PANDAS处理时间序列

（1）定义时间戳timestamp格式

```
import pandas as pd
#定义timestamp
t1=pd.Timestamp('2021-01-10')
t2=pd.Timestamp('2022-05-20')
print(f't1= {t1}')
print(f't2= {t2}')
print(f't1与t2时间间隔： {(t1-t2).days}天')
```

运行结果：

t1= 2021-01-10 00:00:00

t2= 2022-05-20 00:00:00

t1与t2时间间隔：-495天

9.2.3 PANDAS处理时间序列

（2）实现datetime加减

对日期和时间进行加减实际上就是把datetime往后或往前计算，得到新的datetime。加减可以直接用+和-运算符。这里会用到pandas模块中的timedelta类。

```
import pandas as pd
inte = pd.Timedelta(
        days=5, minutes=50,
        seconds=20, milliseconds=10,
microseconds=10, nanoseconds=10)
print(inte)
```

运行结果：

5 days 00:50:20.010010010

9.2.3 PANDAS处理时间序列

想从当前日期开始往后推100天，可以直接用加法

```
#计算当前时间往后100天的日期
import pandas as pd
now=pd.datetime.now()
dt=now+pd.Timedelta(days=100)
print(dt.strftime('%Y-%m-%d'))
```

运行结果：

2022-11-27

这表示从当前日期（2022年8月19日）算起，增加100天后的新日期为2022年11月27日。

9.2.3 PANDAS处理时间序列

（3）生成一段时间范围

可以利用date_range方法来生成一段时间。

```
pd.date_range(start=, end=, periods=, freq='D')
```

其中，start、和end和freq配合，可以生成star 和end范围内以频率freq的一组时间索引；另一种情况，start、periods和freq配合能够生成从star 开始的频率为freq的periods个时间索引。

9.2.3 PANDAS处理时间序列

频率的缩写表示

缩写	说明	偏移量类型
D	频率间隔为天，是每个日历日	Day
B	频率间隔为天，是每个工作日	BusinessDay
H	频率间隔为每小时	Hour
T	频率间隔为每分	Minute
S	频率间隔为每秒	Second
ms	频率间隔为每毫秒	Milli
M	频率间隔为每月最后一个工作日	MonthEnd
MS	频率间隔为每月第一个工作日	MonthBegin

9.2.3 PANDAS处理时间序列

（4）显示一段时间

```
import pandas as pd
pd.date_range(start='20210901', end='20210924')
```

运行结果：

```
DatetimeIndex(['2021-09-01', '2021-09-02', '2021-09-03', '2021-09-04',
               '2021-09-05', '2021-09-06', '2021-09-07', '2021-09-08',
               '2021-09-09', '2021-09-10', '2021-09-11', '2021-09-12',
               '2021-09-13', '2021-09-14', '2021-09-15', '2021-09-16',
               '2021-09-17', '2021-09-18', '2021-09-19', '2021-09-20',
               '2021-09-21', '2021-09-22', '2021-09-23', '2021-09-24'],
              dtype='datetime64[ns]', freq='D')
```

9.2.4 其它时间处理模块

➢statsmodels模块

statsmodels是第三方模块，是一个 Python 包，它提供了用于估计各种统计模型以及运行统计测试和统计数据分析的类和函数。

它为时间序列分解及其可视化提供了一种非常方便的方法。使用这个包，可以轻松分解任何时间序列并分析其组成部分，例如趋势、季节性组成部分以及残差或噪声。

statsmodels 库在名为seasonal_decompose()的函数中提供了简单或经典分解方法的实现。它要求你指定模型是加法还是乘法。

seasonal_decompose() 函数返回一个结果对象。结果对象以数组形式提供对趋势和季节性系列的访问。它还提供了对残差的访问，残差是趋势之后的时间序列，并且去除了季节性成分。

9.2.4 其它时间处理模块

➢ dateutil模块

想从Blog的RSS中读取日志内容，再根据日期来确定哪些内容是自己所需要的，这里需要用到两个东西，一个是不同格式日期的解析，一个就是日期的比较了。dateutil模块主要有两个函数，parser和rrule。其中parser是根据字符串解析成datetime，而rrule则是根据定义的规则来生成datetime。

因为是第三方库，所以需要安装. pip install python-dateutil

parser可以把表示日期的字符串解析成datetime，字符串格式可以多样化，可以用时间日期的英文单词，可以用横线、逗号、空格等做分隔符。

9.2.4 其它时间处理模块

➢ arrow模块

arrow是一个专门处理时间和日期的轻量级Python库，它提供了一种合理、智能的方式来创建、操作、格式化、转换时间和日期。它的官方网站上称它可以为创建、操作、格式化和转换日期、时间和时间戳提供一种合理且人性化的方法。它实现和更新datetime类型，填补了功能上的空白，并提供了支持许多常见创建场景的智能模块API。简单地说，它可以帮助您以更少的导入和更少的代码处理日期和时间。https://arrow.readthedocs.io/

因为是第三方库，所以需要安装。 pip install arrow

9.3.1 创建时间序列

（1）Pandas中的Series对象

Pandas的Series对象是能够保存任何类型的数据（整数，字符串，浮点数，Python对象等）的一维数组。其与一般数组的最大区别在于能够定义灵活的索引。与大多数类数组的对象相同，创建Series对象最直接的方法是用一个一维数组作为实例化的参数。

```
from pandas import Series
from numpy import random
import numpy as np
import pandas as pd
from datetime import datetime
dates=[datetime(2022,1,2),datetime(2022,1,5),datetime(2022,1,7),
       datetime(2022,1,8),datetime(2022,1,9),datetime(2022,1,10)]
ts = Series(np.random.randn(6),index = dates)
print(ts)
```

运行结果如下：

```
2022-01-02   -0.281425
2022-01-05    1.078028
2022-01-07   -1.190149
2022-01-08    0.458002
2022-01-09    0.059659
2022-01-10    1.482028
dtype: float64
```

9.3.1 创建时间序列

（2）固定频率date_range()

Pandas提供的date_range()方法，用于生成一个固定频率的DatetimeIndex时间索引。

```
date_range(start=None, end=None, periods=None, freq=None,
tz=None, normalize=False,  name=None, closed=None, **kwargs)
```

参数说明：

- start：表示起始日期，默认为None。
- end：表示终止日期，默认为None。
- periods：表示产生多少个时间戳索引值。
- freq：用来指定计时单位。

使用date_range()方法可以快速创建出一个日期范围。

```
x=pd.date_range(start='2021/2/1', end='2021/2/28', freq='5D')
print(x)
```

运行结果如下：

```
DatetimeIndex(['2021-02-01', '2021-02-06', '2021-02-11', '2021-02-16',
               '2021-02-21', '2021-02-26'],
              dtype='datetime64[ns]', freq='5D')
```

9.3.2 子集的选取

Pandas读取Excel、csv文件中的数据时，获取的大多是表格型的二维数据，在pandas中对应的即为DataFrame数据结构。在处理这类数据时，往往要根据据需求先获取数据中的子集，如某些列、某些行、行列交叉的部分等。可以说子集选取是一个非常基础、频繁使用的操作，而DataFrame的子集选取看似简单却有一定复杂性。

这里涉及几个基本概念。

- 列标签（也叫列名：columns）
- 行标签（也叫行索引：index）默认为(0, 1, 2, …, n)。这里与位置序号恰好一致。

```
# 指定索引为多个日期字符串的列表
date_list = ['2020/06/30', '2020/06/01','2021.6.1', '2021.4.7','2022.9.1',
'2022.1.23']
# 将日期字符串转换为DatetimeIndex
date_index = pd.to_datetime(date_list)
# 创建以DatetimeIndex 为索引的Series对象
date_se = pd.Series(np.arange(6), index=date_index)
print(date_se)
```

运行结果如下：

```
2020-06-30    0
2020-06-01    1
2021-06-01    2
2021-04-07    3
2022-09-01    4
2022-01-23    5
dtype: int32
```

9.3.2 索引与切片

TimeSeries是Series的一个子类，所以Series索引及数据选取方面的方法基本一样；同时TimeSeries通过时间序列有更便捷的方法做索引和切片；时间序列由于按照时间先后排序，故不用考虑顺序问题。

（1）简单索引

```
from datetime import datetime
rng = pd.date_range('2022/1','2022/3')
ts = pd.Series(np.random.rand(len(rng)), index = rng)
print(ts.head())
print(ts[0])
```

运行结果如下：

```
2022-01-01    0.896031
2022-01-02    0.902591
2022-01-03    0.178581
2022-01-04    0.573210
2022-01-05    0.948120
Freq: D, dtype: float64
0.896031195250842
```

9.3.3 索引与切片

（2）时间序列标签索引

```
print(ts['2022/1/2'])
print(ts['20220103'])
print(ts['1/10/2022'])
print(ts[datetime(2022,1,20)])
```

运行结果如下：

```
0.9025909026944346
0.17858111106812002
0.45985833398609
0.5890343430095241
```

（3）切片

切片功能特别实用，可以方便地对时间序列数据根据要求进行筛选。例如，调用某一个月的数据，进行查看或者处理。

```
rng = pd.date_range('2022/1','2022/3',freq = '12H')
ts = pd.Series(np.random.rand(len(rng)), index =
rng)
print(ts[:2])
print(ts['2022/1/5':'2022/1/10'])
print(ts['2022/2'].head())
```

其中，三个不同方式的切片，第一个直接按照行数切片，第二个按照索引切片，第三个直接传入月份进行切片。

9.3.3 索引与切片

（4）重复索引的时间序列

有时候，日期数据里面并不止一个时间标签，结果就是一个标签对应多个数据，检查标签和数值是否重复的方式为：is_unique。

```
dates=pd.DatetimeIndex(['1/1/2022','1/2/2022','1/3/2022'
                        ,'1/4/2022','1/1/2022','1/2/2022'])
ts = pd.Series(np.random.rand(6), index = dates)
print(ts)
print(ts.is_unique,ts.index.is_unique)
```

运行结果如下：

```
2022-01-01    0.980790
2022-01-02    0.724384
2022-01-03    0.435161
2022-01-04    0.717160
2022-01-01    0.590178
2022-01-02    0.314510
dtype: float64
True False
```

当index有重复的索引时候将返回多个值。

```
print(ts['20220101'],type(ts['20220101']))
print(ts['20220104'],type(ts['20220104']))
```

运行结果如下：

```
2022-01-01    0.980790
2022-01-01    0.590178
dtype: float64 <class 'pandas.core.series.Series'>
0.7171604381570228 <class 'numpy.float64'>
```

9.3.4 时间序列的频率和偏移量

Pandas中，默认生成的时间序列数据是按天计算的，即频率为“D”。“D”是一个基础频率，通过用一个字符串的别名表示，比如“D”是“day”的别名。频率是由一个基础频率和一个乘数组成的，比如，“5D”表示每5天。

```
x=pd.date_range(start='2022/2/1', end='2022/2/28', freq='5D')
```

输出x，结果如下所示：

```
DatetimeIndex(['2022-02-01', '2022-02-06', '2022-02-11', '2022-02-16',
               '2022-02-21', '2022-02-26'],
              dtype='datetime64[ns]', freq='5D')
```

每个基础频率还可以跟着一个被称为日期偏移量的DateOffset对象。如果想要创建一个DateOffset对象，则需要先导入pd.tseries.offsets模块后才行。

```
from pandas.tseries.offsets import *
x=DateOffset(months=4, days=5)
print(x)
```

运行结果如下：

```
<DateOffset: days=5, months=4>
```

9.3.4 时间序列的频率和偏移量

可以使用offsets模块中提供的偏移量类型进行创建。例如，创建14天10小时的偏移量，可以换算为两周零十个小时，其中“周”使用Week类型表示的，“小时”使用Hour类型表示，它们之间可以使用加号连接。

```
date_offset  = Week(2) + Hour(10)
z=pd.date_range('2022/3/1', '2022/3/31',
freq=date_offset)
print(z)
```

这里的频率是14天10小时的偏移量。运行结果如下：

```
DatetimeIndex(['2022-03-01 00:00:00', '2022-03-15 10:00:00',
               '2022-03-29 20:00:00'],
              dtype='datetime64[ns]', freq='346H')
```

9.3.5 时期及其算术运算

（1）时期对象的创建

Period类表示一个标准的时间段或时期，比如某年、某月、某日、某小时等。

创建Period类对象的方式比较简单，只需要在构造方法中以字符串或整数的形式传入一个日期即可。

例如，创建一个可以表示从2021-01-01到2021-12-31之间的时间段对象。

9.3.5 时期及其算术运算

（2）时期对象的数学运算

Period对象能够参与数学运算。如果Period对象加上或者减去一个整数，则会根据具体的时间单位进行位移操作。如果具有相同频率的两个Period对象进行数学运算，那么计算结果为它们的单位数量。

```
# Period对象加上一个整数
print(x + 1 )
other_period = pd.Period(201401,
freq='M' )
print(y - other_period)
```

运行结果如下：

2021-07
<5 * MonthEnds>

如果希望创建多个Period对象，且它们是固定出现的，则可以通过period_range()函数实现。

```
period_index = pd.period_range('2022.1.8',
                '2022.5.31', freq='M')
```

输出结果返回一个PeriodIndex对象，它是由一组时期对象构成的索引。结果如下：

PeriodIndex(['2022-01', '2022-02', '2022-03', '2022-04', '2022-05'], dtype='period[M]')

9.3.5 时期及其算术运算

除了使用上述方式创建PeriodIndex外，还可以直接在PeriodIndex的构造方法中传入一组日期字符串。

```
str_list = ['2020', '2021', '2022']
y=pd.PeriodIndex(str_list, freq='A-DEC')
print(y)
```

运行结果如下：

PeriodIndex([‘2020’, ‘2021’, ‘2022’], dtype=‘period[A-DEC]’)

```
period_ser = pd.Series(np.arange(5),
period_index)
print(period_ser)
```

运行结果如下：

```
2022-01    0
2022-02    1
2022-03    2
2022-04    3
2022-05    4
Freq: M, dtype: int32
```

9.3.6 时区处理

（1）pytz库的使用

时区是以UTC偏移量的形式表示的。在Python中，时区信息来自第三方库pytz，它使Python可以使用Olson数据库（汇 编了世界时区信息）。这对历史数据非常重要，这是因为由于各地政府的各种突发奇想，夏令时转变日期（甚至UTC偏移量）已经发生过多次改变了。要从pytz中获取时区对象，要使用pytz.timezones。

```
import pytz
tz = pytz.common_timezones[-5:]
print(tz)
```

运行结果如下：

```
['US/Eastern', 'US/Hawaii', 'US/Mountain', 'US/Pacific', 'UTC']
```

（2）时区本地化

默认情况下，pandas中的时间序列是单纯（naive）的时区。

```
 # 生成范围日期时可以加上时区信息
rng = pd.date_range('3/9/2021 22:29',periods=5,freq='D')
ts = pd.Series(np.random.randn(len(rng)), index=rng)
print(ts)
```

运行结果如下：

```
2021-03-09 22:29:00   -3.058159
2021-03-10 22:29:00   -0.477388
2021-03-11 22:29:00    0.409784
2021-03-12 22:29:00    0.163310
2021-03-13 22:29:00    0.478280
Freq: D, dtype: float64
```

可以通过tz_localize方法可以为没有时区的时间序列赋予时区。

```
ts_utc = ts.tz_localize('UTC')
print(ts_utc)
```

运行结果如下：

```
2021-03-09 22:29:00+00:00   -3.058159
2021-03-10 22:29:00+00:00   -0.477388
2021-03-11 22:29:00+00:00    0.409784
2021-03-12 22:29:00+00:00    0.163310
2021-03-13 22:29:00+00:00    0.478280
Freq: D, dtype: float64
```

9.3.6 时区处理

一旦时间序列被本地化到某个特定时区，就可以用tz_convert()方法将其转换到别的时区了。

```
ts_eastern =
ts_utc.tz_convert('US/Eastern')
print(ts_eastern)
```

运行结果如下：

```
2021-03-09 17:29:00-05:00   -0.531917
2021-03-10 17:29:00-05:00    1.004579
2021-03-11 17:29:00-05:00   -0.102343
2021-03-12 17:29:00-05:00   -0.617575
2021-03-13 17:29:00-05:00    0.433114
Freq: D, dtype: float64
```

```
x=ts_eastern.tz_convert('UTC')
print(x)
```

运行结果如下：

```
2021-03-09 22:29:00+00:00   -0.529296
2021-03-10 22:29:00+00:00    1.201424
2021-03-11 22:29:00+00:00   -0.536082
2021-03-12 22:29:00+00:00    1.029342
2021-03-13 22:29:00+00:00   -0.106025
Freq: D, dtype: float64
```

9.3.6 时区处理

tz_localize()和tz_convert()也是DatetimeIndex的实例方法。

```
ts.index.tz_localize('Asia/Shanghai')
```

运行结果如下：

```
DatetimeIndex(['2021-03-09 22:29:00+08:00', '2021-03-10 22:29:00+08:00',
               '2021-03-11 22:29:00+08:00', '2021-03-12 22:29:00+08:00',
               '2021-03-13 22:29:00+08:00'],
              dtype='datetime64[ns, Asia/Shanghai]', freq=None)
```

9.4 重采样及频率转换

9.4.1 重采样介绍

时间序列数据的重采样是对一个常用操作，是将时间序列从一个频率转换为另一个频率的过程，简单来说，就是改变频率。

根据频率改变的情况，重采样可以分为降采样和和升采样两种。

降采样是指从高频率变换到低频率，升采样是指从低频率变换到高频率。

例如，原始的采样频率是按照天来统计，重采样时变为按照月进行统计，这时数据量变少了，这种重采样叫做降采样。

相反，如果采样频率由天变为5小时，那数据量就变多了，这种采样叫做升采样。

升采样会产生缺失值，相比而言，降采样更为常用。

9.4.2 resample方法

pandas模块中的resample()方法是一个常用的重采样方法。这个方法可以将时间序列从一个频率转化为另一个频率，通过修改period来制定新的频率。

该方法使用时需要注意，处理的对象必须具有索引，而且index必须是datetime类型。

resample方法的使用格式如下：

```
DataFrame.resample(rule, axis=0, closed=None, label=None,
                                  convention='start', kind=None,
loffset=None,                     base=None, on=None, level=None,
                                  origin='start_day', offset=None)
```

9.4.2 resample方法

现有乌鲁木齐2019-2021年度空气质量数据文件urumuqi.csv，该数据是一个时间序列，原有采样频率为天。如果统计数据中不同年份空气质量，则需要更换采样频率。如果想在此基础上，统计不同年份空气质量为优的情况，则需要在读取到CSV数据后进行筛选，挑选出空气质量等级为优的数据后，以年为新的采样频率来计数。

urumuqi.csv文件数据内容如下：

```
日期,        质量等级,AQI指数,当天AQI排名,PM25,PM10,So2,No2, Co,  O3
2019-01-01,重度污染,  206,      362,       156,  230,  13,  67, 2.63, 18
2019-01-02,重度污染,  211,      352,       162,  243,   8,  65, 2.67, 19
2019-01-03,重度污染,  211,      331,       164,  244,   9,  69, 2.78, 17
……
```

9.4.2 resample方法

```
import numpy as np
import pandas as pd
df = pd.read_csv("./wulumuqi.csv")
#利用to_datetime()方法将日期列转换为datetime类型
df["日期"] = pd.to_datetime(df["日期"])
#为该序列创建索引
df.set_index("日期", inplace=True)
print(df.head())
print(df.info())
#从序列中挑选空气质量等级为优的记录
sel_data = df[df['质量等级'] == "优"].copy()
print(sel_data)
#以年为频率进行重采样，统计不同年份空气质量为优的次数
count_by_year =sel_data.resample('Y').count()['质量等级']
print(count_by_year)
```

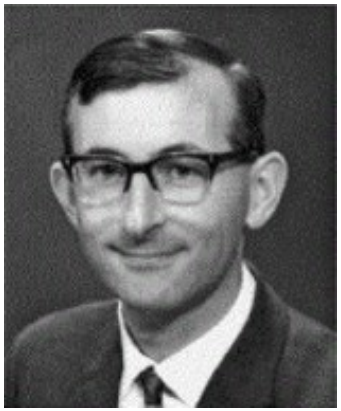
9.5时序分析模型ARIMA

著名时间序列预测方法

➢ 差分整合移动平均自回归模型
(Autoregressive Integrated Moving Average Model, ARIMA)

模型包括：
自回归模型（AR模型）
移动平均模型（MA模型）
自回归-移动平均混合模型（ARMA模型）
差分整合移动平均自回归模型（ARIMA模型）

乔治·博克斯（统计学家）
George E.P. Box

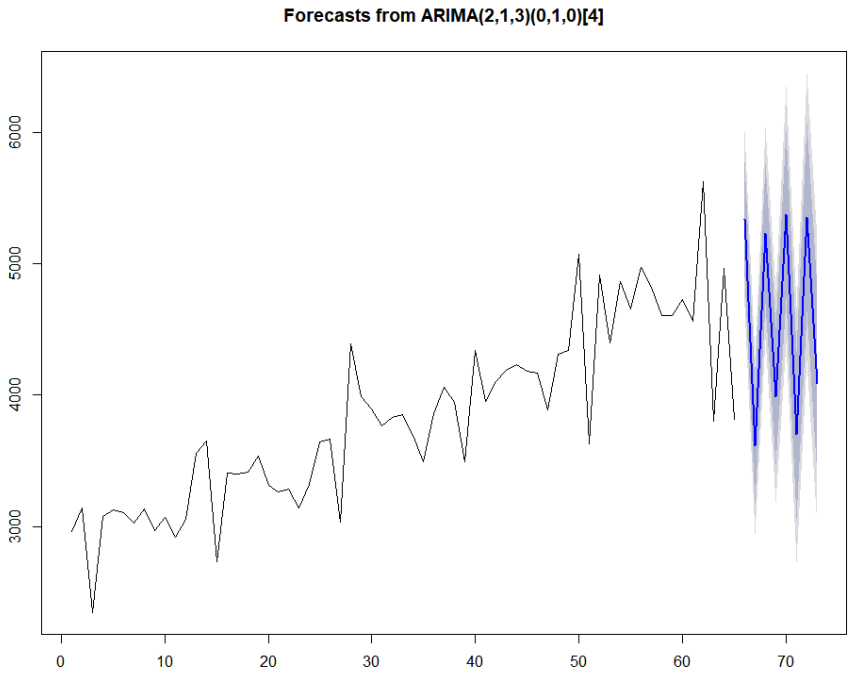


格威利姆·詹金斯（统计学家）
Gwilym M.Jenkins

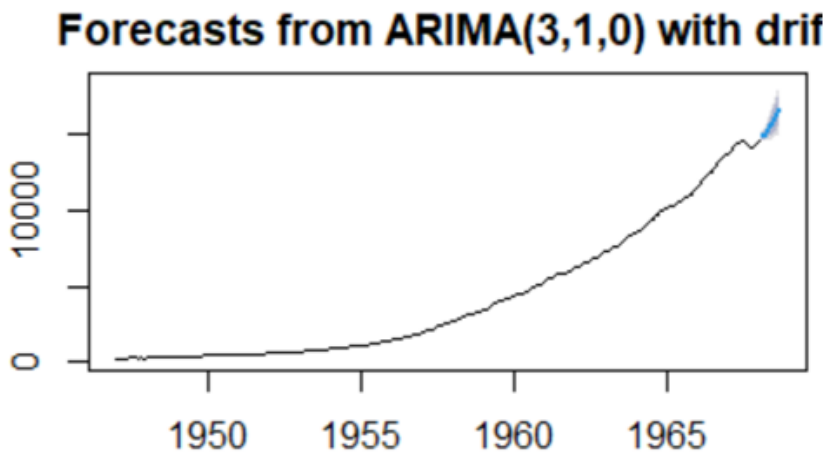
9.5 时序分析模型ARIMA

随着科技的飞速发展，应用ARIMA模型的领域日渐增多，比如了解大豆单产预测、农产品价格预测、降水量的预测等等。

疾病预测门诊量预测图



美国GNP增长率预测图



9.5.1 模型的基本原理

ARIMA通常用于需求预测和规划中，可以用来对付随机过程的特征随着时间变化而非固定，并且导致时间序列非平稳的原因是随机而非确定的。

将非平稳时间序列转化为平稳时间序列，然后将因变量仅对它的滞后值以及随机误差项的现值和滞后值进行回归所建立的模型。

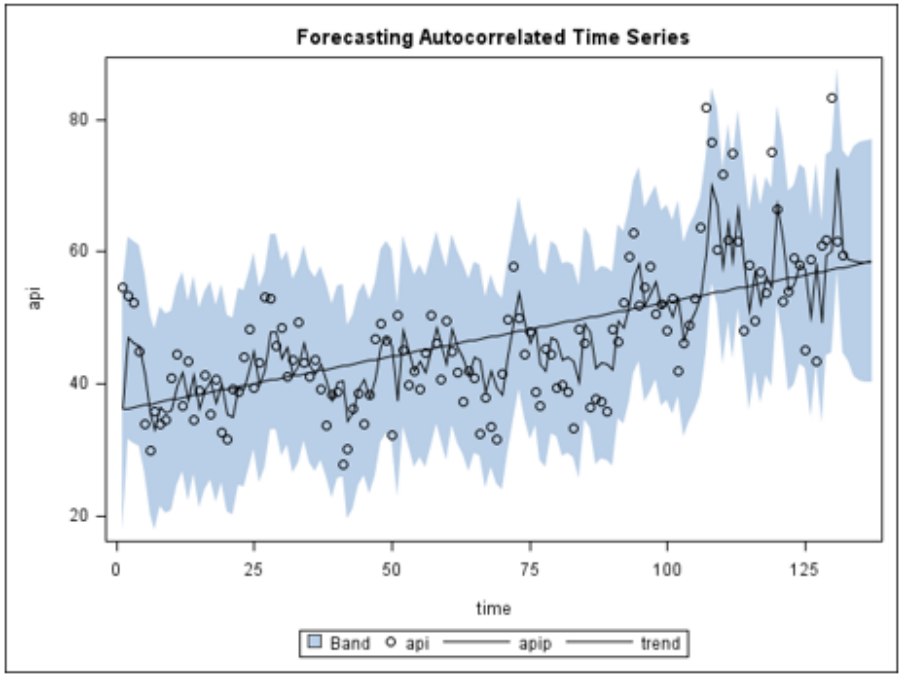
其中ARIMA（p，d，q）模型是ARMA(p，q)模型的扩展，AR是自回归，p为自回归项的阶数；MA为移动平均，q为“移动平均”项的阶数，d为使序列平稳所需的最小差分阶数。

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \xi_t$$

9.5.1 模型的基本原理

自回归模型（英语：Autoregressive model，简称**AR模型**），是统计上一种处理时间序列的方法，用同一变数例如x的之前各期，亦即x_1至x_{t-1}来预测本期x_t的表现，并假设它们为一线性关系。因为这是从回归分析中的线性回归发展而来，只是不用x预测y，而是用x预测x（自己）；所以叫做**自回归**。

自回归模型图列



9.5.1 模型的基本原理

一般的P阶自回归模型AR，描述当前值与历史值之间的关系，用变量自身的历史时间数据对自身进行预测，必须满足平稳性的要求。

自回归模型首先需要确定一个阶数p，表示用几期的历史值来预测当前值。p阶自回归模型的公式定义为：

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \xi_t$$

其中，α是常数项，$\beta_1,\ldots,\beta_p$是模型参数，p为阶数，ξ_t是具备均值为0，方差为σ的白噪声（白噪声是指功率谱密度在整个频域内均匀分布的噪声）。

9.5.1 模型的基本原理

移动平均模型（moving average model; MA model ）是2016年公布的管理科学技术名词。
定义：时间序列当期值为随机误差项以及滞后误差项线性函数所形成的模型。

移动平均模型MA关注的是自回归模型中的误差项的累加，能有效地消除预测中的随机波动，q阶自回归过程的公式定义如下：

$$Y_t = \alpha + \xi_t + \phi_1\xi_{t-1} + \phi_2\xi_{t-2} + \cdots + \phi_q\xi_{t-q}$$

其中，α是常数项，$\phi_1,\ldots,\phi_q$是模型参数，q为阶数，ξ_t是具备均值为0，方差为σ的白噪声。

9.5.1 模型的基本原理

将AR（p）与MA（q）结合，得到一个一般的自回归移动平均模型ARMA（p，q）：

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \xi_t + \phi_1 \xi_{t-1} + \phi_2 \xi_{t-2} + .. + \phi_q \xi_{t-q}$$

其中，α是常数项，$\beta_1, \ldots, \beta_p$，$\phi_1, \ldots, \phi_q$是模型参数，p与q为阶数，ξ_t是具备均值为0，方差为σ的白噪声。

该式表明：

（1）一个随机时间序列可以通过一个自回归移动平均模型来表示，即该序列可以由其自身的过去或滞后值以及随机扰动项来解释。

（2）如果该序列是平稳的，即它的行为并不会随着时间的推移而变化，那么我们就可以通过该序列过去的行为来预测未来。

9.5.1 模型的基本原理

ARIMA建模流程

1）获取被观测系统时间序列数据

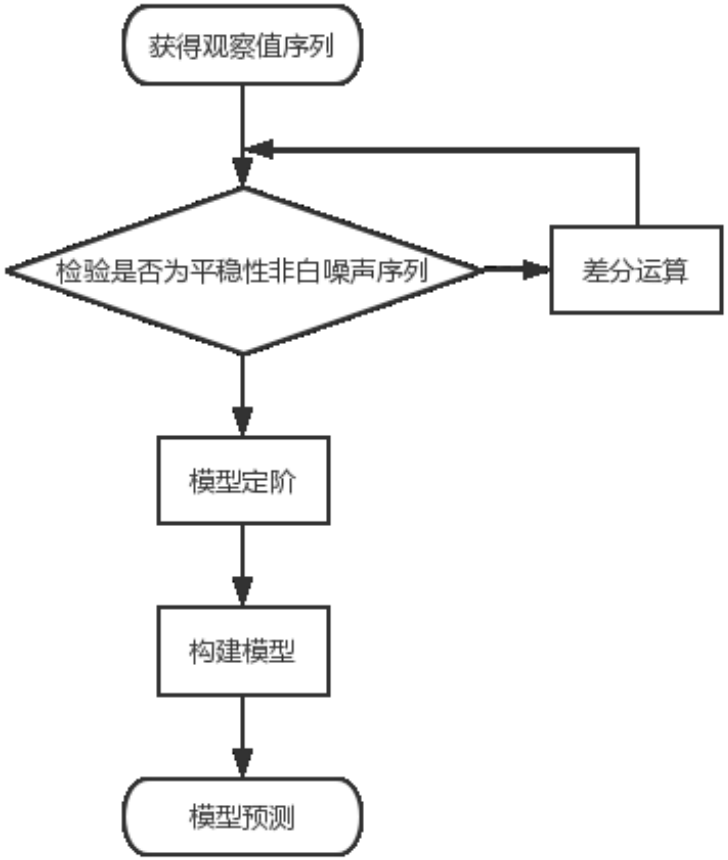
2）时间序列的预处理。时间序列的预处理包括两个方面的检验，平稳性检验和白噪声检验。

3）确定模型的阶数。对平稳时间序列分别绘制其自相关图ACF和偏自相关图PACF，通过图像对p、q值进行估算，利用BIC准则得到最佳的阶层p和阶数q。

4）建立模型。由以上获取的d、p、q，得到ARIMA模型。

5）用建立的ARIMA模型进行预测。

建模流程：



9.5.1 模型的基本原理

在建模时，我们不想保留太多相互关联的特征，因为这会产生多重共线性问题。因此，我们只需要保留相关功能。

如何根据自相关系数和偏自先关系数来判断ARIMA模型中的AR过程和MA过程的阶数？我们可以通过以下规则来进行：

模型	ACF	PACF
AR（p）	拖尾衰减趋于0（几何形或者震荡型）	p阶后截尾
MA（q）	q阶后截尾	拖尾衰减趋于0（几何形或者震荡型）
ARMA(p,q)	q阶拖尾衰减趋于0（几何形或者震荡型）	p阶拖尾衰减趋于0（几何形或者震荡型）

截尾：在大于某个常数k后快速趋于0为k阶截尾

拖尾：始终有非零取值，不会在k大于某个常数后就恒等于零（或在0附近随机波动）

9.5.2模型的Python实现

要求：利用ARIMA模型完成未来五天大众销量的预测

ARIMA模型实现过程包含以下几步：

（1）获取时间序列数据

（2）时序图绘制

（3）平稳性检测

（4）如果数据不平稳，则对数据进行差分，并画出差分后的时序图、自相关图和偏相关图

（5）进行平稳性检测和白噪声检测

（6）对模型进行定阶，得出p、q值，进行预测

（7）对得到的p和q的值建立ARIMA模型，并进行预测。

最终预测得到，未来五天大众销量的数据：

```
预测结果为：  [39500.27657318 41236.97837674 42127.335732   39349.02316569
 41166.34945884]
```

注意事项：

pip install statsmodels==0.10.1

pip install statsmodels

9.6本章实训

实训目的

熟练掌握时序数据的预处理以及利用ARIMA模型对时序数据进行预测。

要求：

大气环境污染对工农业危害十分严重，在国家、政府大力倡导和资金扶持下，大气环境自动监测技术快速发展，一些走在前列的省份已经开始开展借助以往的空气质量指数数据来预测未来的空气质量数据的研究。针对这些研究的数据，我们来进行对时序数据预处理操作。预测未来空气质量。

9.6.2数据的预处理及可视化步骤

数据的来源为“天气后报”的空气质量指数（http://www.tianqihoubao.com/aqi）

预处理及可视化步骤步骤

（1）数据爬取

使用requests模块中的requests.get()方法获取到网页的数据，之后使用bs4模块中的BeautifulSoup()方法对网页进行解析，最后将爬取到的数据写入到fourCityAQI 文件夹中的csv文件中，分别用四大城市的拼音进行命名。

（2）数据清洗

首先将4个城市的空气质量历史数据处理进行联合成一张表，并处理其中的缺失值，最后保存文件到fourcity_aqi.csv。对于缺失值的处理为取上一行的数据进行填充，用的主要是Pandas模块的fillna()方法，有ffill，pad，bfill，backfill四种填充方式。

9.6.2数据的预处理及可视化步骤

（3）数据预处理

对乌鲁木齐的空气质量历史数据进行预处理，向数据中加入季度列，将数据分为四个季节的数据。

（4）数据可视化

将预处理的数据进行更加直观地展示出来，使用按季节分组的直方图来进行展示数据。

9.6.3利用ARIMA模型完成对时序数据的预测步骤

（1） 导入筛选数据

（2） 画出筛选数据的时序图

（3） 对数据进行平稳性检测

（4） 画出数据的自相关性图以及偏相关性图

（5） 对数据进行白噪声检测

（6） 对模型进行定阶

（7） 建立ARIMA模型并利用模型进行预测

结果分析：通过运行结果判断，若预测数据与之前已有数据接近，说明预测结果准确率高，用ARIMA模型进行分析是合理的。出现数据偏差大的原因可能是模型本身的局限性，数据的采集量还不够多，每年的季节变化情况有所不同等。