Python 数据挖掘实践（Data Mining Action）

Chap 7 时间序列分析预测（Time Series Analysis & Prediction）

内容：

- 序列分析概述
- 序列的预处理
  - 平稳性检验
  - 纯随机性检验
- 平稳序列分析
  - AR模型
  - MA模型
  - ARMA模型
- 非平稳序列分析
  - 差分运算
  - ARIMA模型
- Python 主要时序模式算法
- 序列数据的特征工程
- 序列分析模型：AR模型、MA模型、ARMA模型、ARIMA模型
- 应用领域：时间序列（股票、天气），空间序列（地形、生态），金融、经济、生态、神经科学、物理学等

实践：

- 实例1：某公司产品销售数据的序列分析
- 实例2：预测股票的股价
- 实例3：魁北克月度汽车销量预测的特征分析

需要安装并加载StatsModels库

```
conda install statsmodels
```

这节课是在前面数据分析的基础上，对时间序列类型的数据进行分析和建模，并预测序列的未来数值；时间序列数据可以采用序列分析模型，也可以通过特征工程转变为有监督机器学习的方法来预测，本节课通过3个实例来进行时间序列的数据分析、预测建模和特征分析。

准备工作：导入库，配置环境等

In [1]:
```
from __future__ import division
import os, sys

# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

import pandas as pd
import numpy as np
```

In [2]:
```
# 运行代码时会有很多warning输出，如提醒新版本之类的
import warnings
warnings.filterwarnings('ignore','ValueWarning')
```

1. 序列分析概述

在许多问题中，随机数据是依时间先后排成序列的，称为**时间序列**。时间序列分析就是针对这组数列，应用随机过程理论和数理统计方法，研究随机数据序列所遵从的统计规律，以预测未来事物的发展。

经典的统计分析都假定数据序列具有独立性，而时间序列分析则着重研究数据序列的相互依赖关系。后者实际上是对离散指标的随机过程的统计分析，所以又可看作是随机过程统计的一个组成部分。例如,用 $x(t)$ 表示某地区第 t 个月的降雨量，$\{x(t), t = 1, 2, \ldots, T\}$ 是一个时间序列。对 $t = 1, 2, \ldots, T$，记录逐月的降雨量数据 $x(1), x(2), \ldots, x(T)$，称为长度为 T 的样本序列。利用时间序列分析方法，可以对未来几个月的雨量 $x(T + 1), (T = 1, 2, \ldots)$ 进行预报。 时间序列的各种变化都可以归纳成四大类因素综合影响：

- 长期趋势 (trend)：会导致序列出现明显的长期趋势
- 循环波动 (circle)：会导致序列呈现出周期性波动
- 季节性变化 (season)：会导致序列呈现出和季节变化相关的稳定周期波动 .
- 随机波动 (immediate)：纯随机，与时间无关

时间序列分析的基本原理：一是承认事物发展的延续性，应用历史数据，就能推测事物的发展趋势。二是考虑到事物发展的随机性，任何事物发展都可能受偶然因素影响，为此要利用统计分析中加权平均法对历史数据进行处理。因此，如果序列是完全无序的随机波动，可以终止对该序列的分析，这是没有信息可提取的平稳序列。时间序列预测一般反映三种实际变化规律：趋势变化、周期性变化、随机性变化。

时间序列分析是定量预测方法之一，该方法简单易行，便于掌握，但准确性差，一般只适用于短期预测。

根据观测得到的时间序列数据，时间序列分析包括一般统计分析（如自相关分析、谱分析等），统计模型的建立与推断，以及关于随机序列的最优预测、控制和滤波等，通过曲线拟合和参数估计（如非线性最小二乘法）来建立数学模型的理论和方法。

时间序列分析在金融、经济、军事、空间科学和工业自动化等部门的应用广泛，如国民经济宏观控制、区域综合发展规划、企业经营管理、市场潜量预测、气象预报、水文预报、地震前兆预报、农作物病虫灾害预报、环境污染控制、生态平衡、天文学和海洋学等。

2. 序列的预处理

拿到一个观察值序列后，首先要对序列的 **纯随机性** 和 **平稳性** 进行检验。这两个重要的检验称为序列的预处理。根据检验结果可以将序列分为不同的类型，对不同类型的序列采取不同的分析方法。

- 对于 **纯随机序列** ，也称为 **白噪声** 序列，序列的各项之间没有任何相关关系，序列中进行完全无序的随机波动，可以终止对该序列的分析。这是没有信息可提取的平稳序列。
- 对于**平稳非白噪声序列**，它的均值和方差是常数，通常建立一个平稳序列的线性模型来拟合该序列的发展，借此提取该序列的有用信息。ARMA模型是最常用的平稳序列拟合模型。
- 对于**非平稳序列**，由于均值和方差不稳定，处理方法一般是将其转变为平稳序列，然后应用有关平稳时间序列的分析方法，如建立ARMA模型来进行相应的研究。如果一个时间序列经差分运算后具有平稳性，则该序列为差分平稳序列，可以使用ARIMA模型进行分析。

一阶差分就是离散函数中连续相邻两项之差。例如定义函数 $X(k)$,则 $Y(k) = X(k + 1) - X(k)$为此函数的一阶差分；对于 $Y(k)$ 的一阶差分 $Z(k) = Y(k + 1) - Y(k) = X(k + 2) - 2 * X(k + 1) + X(k)$ 为此函数的二阶差分。二阶差分法在工程，电学等方面应用比较广泛的。

1. 平稳性检验

1.） 平稳时间序列的定义

- 对于随机变量 X，可以计算其均值（数学期望） μ 和方差 σ^2
- 对于两个随机变量 X 和 Y， 可以计算它们的协方差 $cov(X,Y)=E[(X-\mu_X)(Y-\mu_Y)]$ 和 相关系数 $\rho(X,Y)=\frac{cov(X,Y)}{\sigma X \sigma Y}$， 它们度量了两个不同事件直接的相互影响程度
- 对于时间序列 $\{X_t, t\in T\}$，任意时刻的序列值 X_t 都是一个随机变量，每一个随机变量都会有均值和方差，记 X_t 的均值为 μ_t， 方差为 σ_t； 任取 $t,s\in T$， 定义序列 $\{X_t\}$ 的自协方差函数 $\gamma(t,s)=E[(X_t-\mu_t)(X_s-\mu_s)]$ 和自相关系数 $\rho(t,s)=\frac{cov(X_t,X_s)}{\sigma_t\sigma_s}$ （特别地，$\gamma(t,t)=\gamma(0)=1$，$\rho_0=1$)）。之所以称它们为 **自协方差函数** 和 **自相关系数**，是因为它们衡量的是同一个事件在两个不同时期（时刻 t 和 s）之间的相关程度，形象地讲，就是度量自己过去的行为对自己现在的影响。

如果时间序列$\{X_t, t\in T\}$在某一常数附近波动且波动范围有限，即有常数均值和常数方差，并且延迟 k 期的序列变量的自协方差和自相关系数是相等的或者说延迟 k 期的序列变量之间的影响程度是一样的，则称 $\{X_t, t\in T\}$为平稳序列。

2.） 平稳性的检验

有两种检验方法。（1）根据时序图和自相关图的特征做出判断的图检验，这个方法操作简单、应用广泛，缺点是带有主观性；（2） 构造检验统计量进行检验的方法，常用的是单位根检验（unit root test）。

> （1）时序图检验。根据平稳时间序列的均值和方差都是常数的性质，平稳序列的时序图显示该序列值始终在一个常数附近随机波动，而且波动的范围有界；如果有明显的趋势性或者周期性，那它通常不是平稳序列。

> （2）自相关图检验。平稳序列具有短期相关性，这个性质表明对平稳序列而言通常只有近期的序列值对现时值的影响比较明显，间隔越远的过去值对现时值的影响越小。随着延迟期数 k 的增加，平稳序列的自相关系数 ρ_k（延迟 k 期） 会比较快的衰减趋向于零，并在零附近随机波动，而非平稳序列的自相关系数衰减的速度比较慢，这就是利用自相关图进行平稳性检验的标准。

> （3）单位根检验。单位根检验是指检验序列中是否存在单位根，如果存在单位根就是非平稳时间序列。

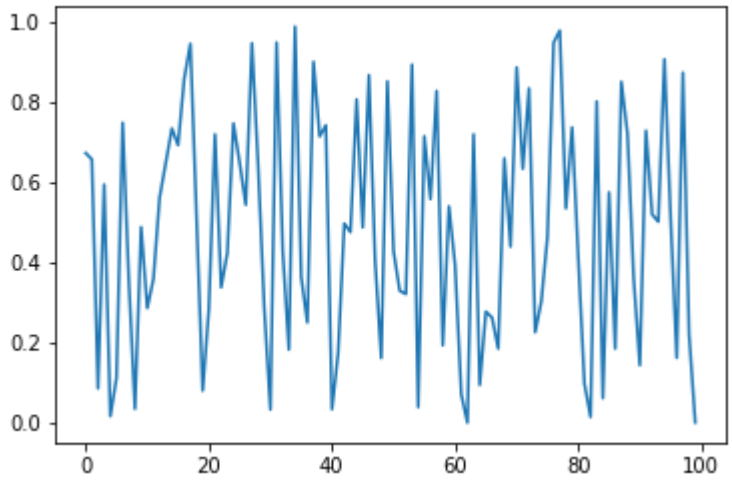
2. 纯随机性检验

如果一个序列是纯随机序列，那么它的序列值之间应该没有任何关系，即满足自协方差函数 $\gamma(k)=0, k\neq 0$。 这是理论上才会出现的理想状态，实际上纯随机序列的样本自相关系数不会绝对为零，但是很接近零，并在零附近随机波动。

纯随机性检验也称白噪声检验，一般是构造检验统计量来检验序列的纯随机性，常用的检验统计量有Q统计量、LB统计量，由样本各延迟期数的自相关系数可以计算得到检验统计量，然后计算出对应的 p 值，如果 p 值显著大于显著性水平 α，则表示该序列不能拒绝纯随机性的原假设，可以停止对该序列的分析。

In [3]:
```
# 1. 生成随机序列
import numpy as np
x = np.random.rand(100)
plt.plot(x)  # 绘图，序列的时序图
```

Out[3]:
```
[<matplotlib.lines.Line2D at 0x7f074f0cd3d0>]
```



对随机生成的序列进行检验：

In [4]:
```
# 2. ADF平稳性检验用来检验金融、经济时间序列数据的平稳性
from statsmodels.tsa.stattools import adfuller as ADF  # 导入ADF检验
print("ADF平稳性检验结果为：") # Augmented Dickey-Fuller unit root test,返回的结果有ADF值、p值等
ADF(x) # 第一个返回值为ADF值，第二个为p值
```

ADF平稳性检验结果为：

Out[4]:
```
(-9.947556575526413,
 2.5655077368830445e-17,
 0,
 99,
 {'1%': -3.498198082189098,
  '5%': -2.891208211860468,
  '10%': -2.5825959973472097},
 35.828511549910246)
```

1) ADF单位根检验进行平稳性检验：

- ADF单位根检验（精确判断）：单位根检验统计量对应的p值(即第二个返回值)显著小于0.05，该序列可认为是平稳序列。(非平稳序列一定不是白噪声序列)
- 对平稳序列还需要进行白噪声检验

In [5]:
```
# 3. 白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("白噪声检验结果为：")  # 返回统计量和p值
acorr_ljungbox(x,lags=1, return_df=False)
```

白噪声检验结果为：

Out[5]:
```
(array([0.04849403]), array([0.82570481]))
```

2）白噪声检验：

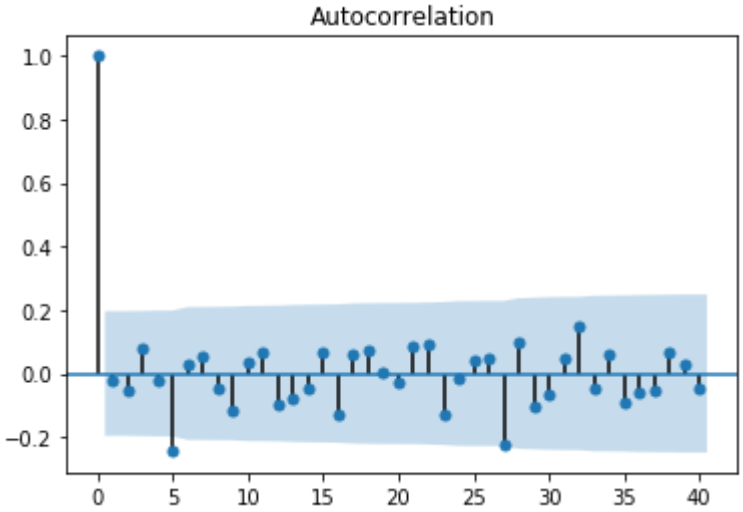
- 统计量的p值(第二个返回值,0.1781) 大于显著性水平0.05，则表示该序列不能拒绝纯随机的原假设，即认为序列为纯随机序列，即白噪声序列，可以停止对该序列的分析。

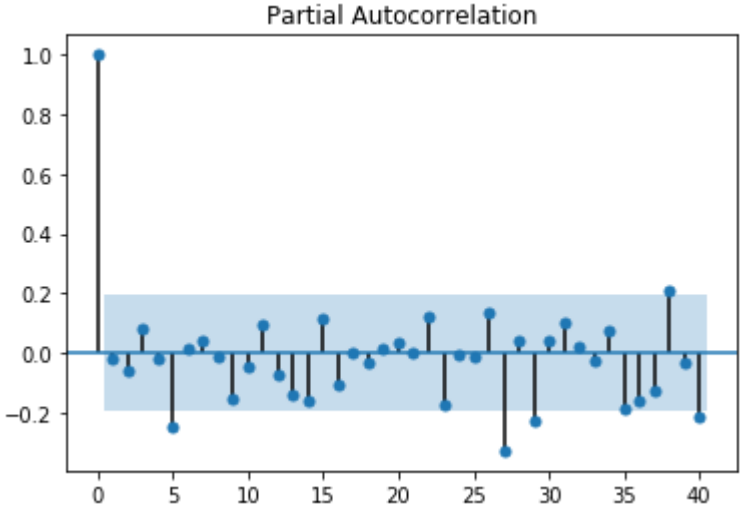
如果白噪声统计量的p值小于显著性水平0.05，则以95%的置信水平拒绝原假设，认为序列为非白噪声序列。再综合ADF检验为平稳序列，则可以认为原序列为平稳非白噪声序列，适用于ARMA模型。

3）自相关和偏自相关图检验

In [6]:
```
# 4. 计算绘制acf自相关系数图和pcaf偏自相关图
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(x,lags = 40).show()  # 绘制原始序列的自相关图，默认lags=20，可以修改，这里最大值99（100个
plot_pacf(x,lags = 40).show()  # 绘制原始序列的偏自相关图
```

```
/home/lanman/anaconda3/lib/python3.7/site-packages/matplotlib/figure.py:445: UserWarning: Matpl
otlib is currently using module://ipykernel.pylab.backend_inline, which is a non-GUI backend, s
o cannot show the figure.
  % get_backend())
```





自相关和偏自相关图检验

- 用于测量当前序列值和过去序列值之间的相关性
- 自相关图/偏自相关图 都是一个平面二维坐标悬垂线图。横坐标表示延迟阶数，纵坐标表示自相关系数/偏自相关系数(y轴上的相关性值在-1到1之间），置信区间被画成圆锥形。默认情况下，置信区间这被设置为95%
- 自相关函数 (ACF)：延迟为 k 时，是相距 k 个时间间隔的序列值之间的相关性。
- 偏自相关函数 (PACF)：延迟为 k 时，是相距 k 个时间间隔的序列值之间的相关性，同时考虑了间隔之间的值。

观察随机序列的自相关图和偏自相关图：

- 本图中 自相关图 和偏自相关图，数据到后面都没有收敛趋势，有拖尾

截尾与拖尾

- 截尾是指时间序列的自相关函数（ACF）或偏自相关函数（PACF）在某阶后均为0的性质，在大于某个常数k后快速趋于0 称为 k阶截尾
- 拖尾是ACF或PACF并不在某阶后均为0的性质，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)

3. 平稳序列分析

某个时间序列经过预处理，被判定为平稳非白噪声序列，就可以利用ARMA模型进行建模。计算出平稳非白噪声序列 { $ X_{t}$ } 的自相关系数和偏自相关系数，再由$ AR(p)$模型、$ MA(q)$ 和$ ARMA(p,q)$ 的自相关系数和偏自相关系数的性质，选择合适的模型。

ARMA模型的全称是自回归移动平均模型(Auto Regressive Integrated Moving Average Model)，是目前最常用的拟合平稳序列的模型。又可以细分为AR模型、MA模型和ARMA三大类，都可以看作是多元线性回归模型。

1). AR模型(Auto Regressive)

t 时刻的随机变量 X_{t} 的取值 x_{t} 是前 p 期 $x_{t-1}, x_{t-2},...,x_{t-p}$ 的多元线性回归，认为 x_{t}主要是受过去 p 期的序列值的影响。具有这样结构的模型称为 $ p$ 阶自回归模型，简记为$ AR(p)$ 。

平稳AR模型的性质见下表：

统计量	性质	统计量	性质
均值	常数均值	自相关系数（ACF)	拖尾
方差	常数方差	偏自相关系数（PACF）	p 阶截尾

自相关系数（ACF)

- 平稳 $ AR(p)$模型的自相关系数 $ p_{k}$ 呈指数的速度衰减，始终有非零取值(不会在k大于某个常数之后就恒等于零)，这个性质就是平稳$ AR(p)$模型的自相关系数 $ p_{k}$ 具有拖尾性。

偏自相关系数（PACF)

- 对于一个平稳 $ AR(p)$模型，求出延迟 k 期自相关系数$ p_{k}$ 时，实际上得到的并不是 X_{t} 与 X_{t-k} 之间单纯的相关关系，因为 X_{t} 同时还会受到中间 $k-1$ 个随机变量$X_{t-1},X_{t-2},...,X_{t-k}$ 的影响，所以自相关系数 $ p_{k}$ 里实际上掺杂了其他变量对 X_{t} 与 X_{t-k} 的相关影响，为了单纯地测度 X_{t-k} 对 X_{t}的影响，引进了偏自相关系数的概念。
- 可以证明平稳 $ AR(p)$ 模型的偏自相关系数具有 $ p$ 阶截尾性。这个性质连同前面的自相关系数的拖尾性是 $ AR(p)$ 模型重要的识别依据。

2). MA模型(Moving Average)

t 时刻的随机变量 X_{t} 的取值 x_{t} 是前 q 期 的随机扰动 $ \epsilon_{t-1}, \epsilon_{t-2},...,\epsilon_{t-q}$ 的多元线性回归，μ 是序列 ${X_{t}}$ 的均值。认为 x_{t}主要是受过去 q 期的误差项的影响。具有这样结构的模型称为 $ q$ 阶自回归模型，简记为$ MA(q)$ 。

平稳MA模型的性质见下表：

统计量	性质	统计量	性质
均值	常数均值	自相关系数（ACF）	q 阶截尾
方差	常数方差	偏自相关系数（PACF）	拖尾

自相关系数（ACF）

- 对于一个平稳 $ MA(q)$模型的自相关系数具有 $ q$ 阶截尾性。

偏自相关系数（PACF）

- 平稳 $ MA(q)$ 模型的偏自相关系数呈指数的速度衰减，始终有非零取值，不会在k大于某个常数之后就恒等于零，这个性质就是平稳$ MA(q)$模型的偏自相关系数具有拖尾性。

3). ARMA模型 (Auto Regressive Moving Average)

ARMA模型的全称是自回归移动平均模型，是目前最常用的拟合平稳序列的模型。

t 时刻的随机变量 X_{t} 的取值 x_{t} 是前 p 期 $x_{t-1}, x_{t-2},...,x_{t-p}$ 和 前 q 期 的随机扰动 $ \epsilon_{t-1}, \epsilon_{t-2},...,\epsilon_{t-q}$ 的多元线性函数，认为 x_{t}主要是受过去 p 期的序列值和过去 q 期的误差项的共同影响。具有这样结构的模型称为自回归移动平均模型，简记为$ ARMA(p,q)$ 。

特殊的，当 $q=0$ 时，是$ AR(p)$ 模型；当 $ p=0$时，是 $ MA(q)$ 模型。

平稳ARMA(p,q)模型的性质见下表：

统计量	性质	统计量	性质
均值	常数均值	自相关系数（ACF）	p阶拖尾
方差	常数方差	偏自相关系数（PACF）	q阶拖尾

自相关系数（ACF）

- 平稳 ARMA模型的自相关系数 呈指数的速度衰减，始终有非零取值(不会在k大于某个常数之后就恒等于零)，这个性质就是平稳 ARMA模型的自相关系数具有 p阶拖尾性。

偏自相关系数（PACF）

- 平稳 ARMA模型的偏自相关系数呈指数的速度衰减，始终有非零取值(不会在k大于某个常数之后就恒等于零)，这个性质就是平稳 ARMA模型的偏自相关系数具有q阶拖尾性。

可以证明 ARMA模型的自相关系数和偏自相关系数都具有拖尾性是 ARMA模型重要的识别依据。

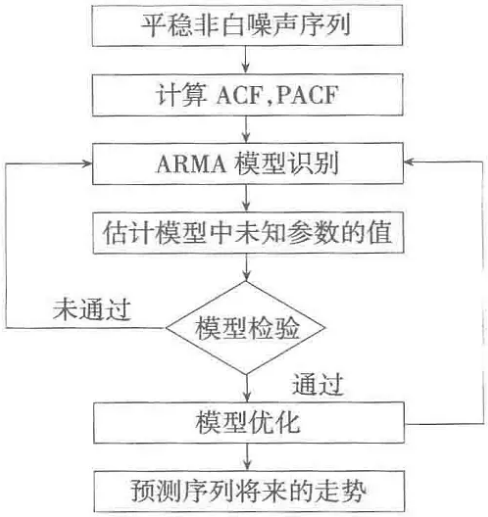
4). 平稳序列建模

平稳时间序列ARMA模型的建模步骤：

- 1）计算ACF和PACF。先计算非平稳白噪声序列的自相关系数（ACF）和偏自相关系数（PACF）。
- 2）ARMA模型识别。也称为模型定阶，由$ AR(p)$模型、$ MA(q)$ 和$ ARMA(p,q)$ 的自相关系数和偏自相关系数的性质，选择合适的模型。识别的原则见下表。模型定阶就是确定 p 和 $ q$。

- 3）估计模型中未知参数的值并根据参数进行检验。
- 4）模型检验。
- 5）模型优化。
- 6）模型应用：进行短期预测。

平稳序列建模步骤如下图所示:



平稳时间序列ARMA模型建模步骤图

下表列出ARMA模型的识别原则

模 型	自相关系数（ACF）	偏自相关系数（PACF）
$ AR$ $ (p) $	拖尾	p 阶截尾
$ MA$ $ (q) $	q 阶截尾	拖尾
$ ARMA$$(p,q)$	p阶拖尾	q阶拖尾

4. 非平稳序列分析

实际上，自然界中绝大部分序列都是非平稳的。因而对非平稳序列的分析更普遍、更重要，创造出来的分析方法也更多。

对非平稳序列的分析方法可以分为确定性因素分解的时序分析和随机时序分析两大类。

确定性因素分解的方法把所有序列的变化都归结为4个因素：

- 长期趋势 (trend)：会导致序列出现明显的长期趋势
- 循环波动 (circle)：会导致序列呈现出周期性波动
- 季节变化 (season)：会导致序列呈现出和季节变化相关的稳定周期波动 .
- 随机波动 (immediate)：纯随机，与时间无关 的综合影响，其中长期趋势（全球气候变暖）和季节变动（雨季和旱季）的规律性信息通常比较容易提取，而由随机因素导致的波动则非常难确定和分析，对随机信息浪费严重，会导致模型拟合精度不够理想。

随机时序分析法的发展是为了弥补确定性因素分解方法的不足。根据时间序列的不同特点，随机时序分析可以建立的模型有ARIMA模型、残差自回归模型、季节模型、异方差模型等。本课重点介绍使用ARIMA模型对非平稳序列进行建模的方法。

几个使原始序列平稳的小技巧：

- 1）如果序列波动很大，也就是方差比较大，可以对序列作对数转换以减缓其波动幅度
- 2）如果序列存在明显趋势，且呈现近似一条直线的趋势，可以对序列作一阶差分，从而消除趋势性
- 3）如果序列存在明显的S期季节性，则可对序列作S阶差分，从而消除季节性

1. 差分运算

1). 1 阶差分

相距一期的两个序列值之间的减法运算称为 1阶差分运算。

2). k 阶差分

相距 k 期的两个序列值之间的减法运算称为 k 阶差分运算。

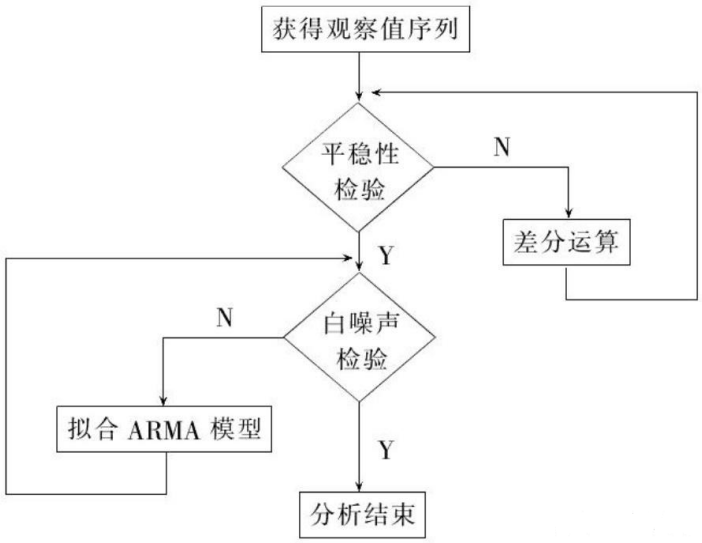
差分是使时间序列得以平稳的一个很重要的过程，但是差分的阶数选择不对，或者差分的次数过多，反而会丢失太多时间序列的信息。

2. ARIMA (Auto Regressive Integrated Moving Average)模型

差分运算具有强大的确定性信息提取能力，许多非平稳序列差分后会显示出平稳序列的性质，这时称这个非平稳序列为差分平稳序列。

对差分平稳序列可以使用ARIMA模型进行拟合。ARIMA模型的实质就是 k阶差分运算与 $ARMA(p,q)$模型的组合，掌握了ARMA模型的建模方法和步骤之后，对序列建立 $ARIMA(p,k,q)$ 模型是比较简单的。

差分平稳序列建模步骤如下图所示。



差分平稳时间序列ARIMA模型建模步骤图

5. Python 主要时序模式算法

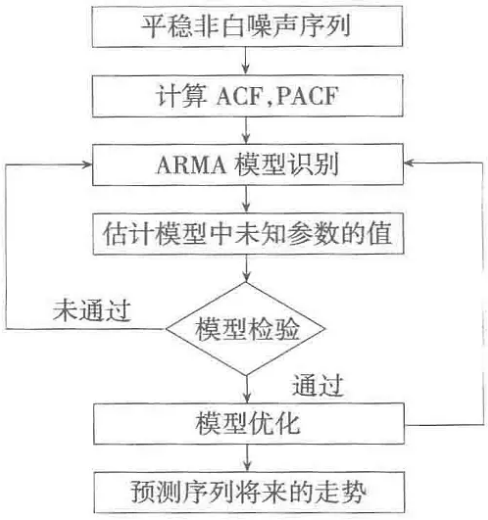
Python实现时序模式的主要库是StatsModels，算法主要是ARIMA模型，在使用该模型进行建模时，需要进行一系列判别操作，主要包含平稳性检验、白噪声检验、是否差分、AIC和BIC指标值、模型定阶，最后再做预测。

时序模式算法函数列表

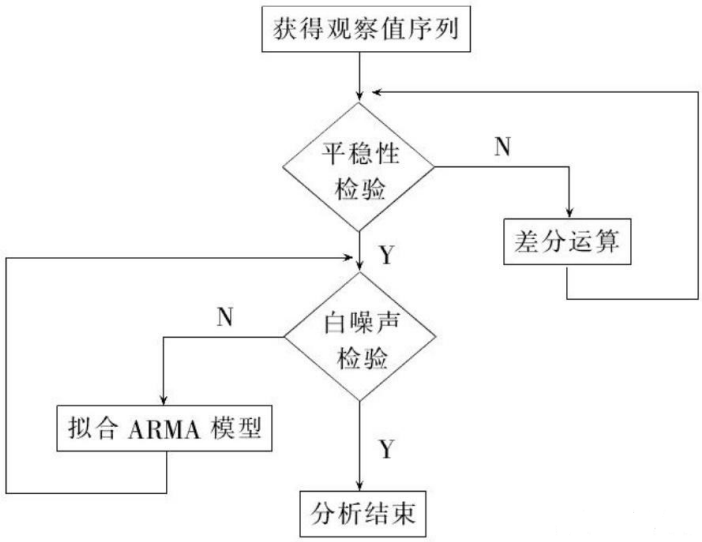
函数名	函数功能	所属工具箱
acf()	计算自相关系数	statsmodels.tsa.stattools
plot_acf()	绘图自相关系数图	statsmodels.graphics.tsaplots
pacf()	计算偏自相关系数	statsmodels.tsa.stattools
plot_pacf()	绘图偏自相关系数图	statsmodels.graphics.tsaplots
adfuller()	对观测值序列进行单位根检验	statsmodels.tsa.stattools
acorr_ljungbox()	Ljung-Box检验，检验是否为白噪声	statsmodels.stats.diagnostic
diff()	对观测值序列进行差分计算	Pandas对象自带的方法

函数名	函数功能	所属工具箱
SARIMAX()	创建一个ARIMA时序模型	statsmodels.tsa.statespace.sarimax
summary()	给出一份ARIMA模型的报告	ARIMA模型对象自带的方法
aic/bic/hqic	计算ARIMA模型的AIC/BIC/HQIC 指标值	ARIMA模型对象自带的变量
forcast()	应用构建的时序模型进行预测	ARIMA模型对象自带的方法

回顾 平稳序列ARMA模型建模和差分平稳序列建模步骤如图



平稳时间序列ARMA建模步骤



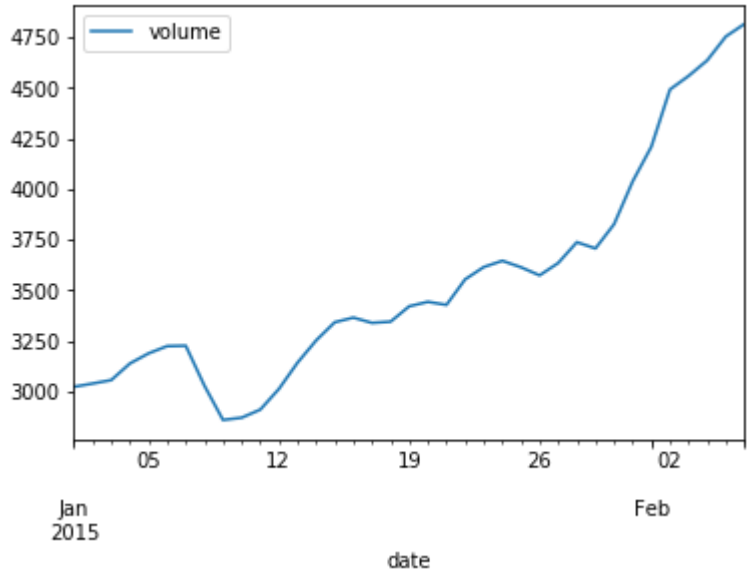
差分平稳时间序列ARIMA建模步骤

实例1：某公司产品销售序列数据的分析

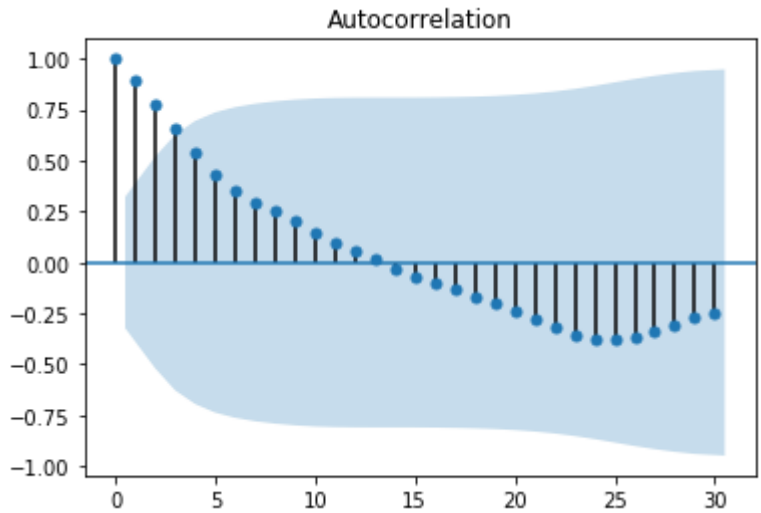
应用以上的理论知识，对某公司的销售数据进行分析和建模。

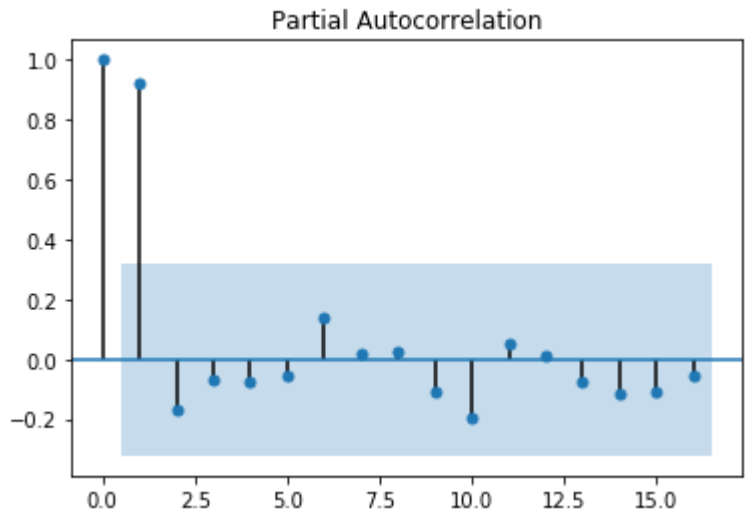
- 1）ADF平稳性检验：不平稳，进行差分运算；平稳，进行第二步白噪声检验
- 2）白噪声检验：纯随机序列，停止分析；非随机序列，进行第三步模型拟合
- 3) 拟合ARMA模型，估计模型中未知参数的值，进行模型检验

In [7]:
```
# 1. 加载数据，绘制原始的时序图
%matplotlib inline
import pandas as pd
data = pd.read_csv('data/arima_data.csv', index_col = 'date')
data.index = pd.to_datetime(data.index)  # 将字符串索引转换成时间索引
data.plot()  # 绘图原始序列的时序图
plt.show()
# 该时序图具有趋势性，不是平稳序列
```



In [8]:
```
# 2. 计算绘制acf自相关系数图和pcaf偏自相关图
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(data,lags = 30).show()  # 绘制原始序列的自相关图
plot_pacf(data).show()  # 绘制原始序列的偏自相关图
```





In [9]:
```
# 3. ADF平稳性检验进行单位根检验，常用来检验金融、经济时间序列数据的平稳性
from statsmodels.tsa.stattools import adfuller as ADF  # 导入ADF检验，对观测值序列进行单位根检验
print("原始序列的ADF检验结果为:",ADF(data['volume']))  # ADF unit root test,返回的结果有ADF值、p值
```

```
原始序列的ADF检验结果为: (1.813771015094526, 0.9983759421514264, 10, 26, {'1%': -3.711212300864
8155, '5%': -2.981246804733728, '10%': -2.6300945562130176}, 299.4698986602418)
```

In [10]:
```
# 4. 白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("原始序列的白噪声检验结果为：",acorr_ljungbox(data['volume'],lags=1, return_df=False))  # 
```

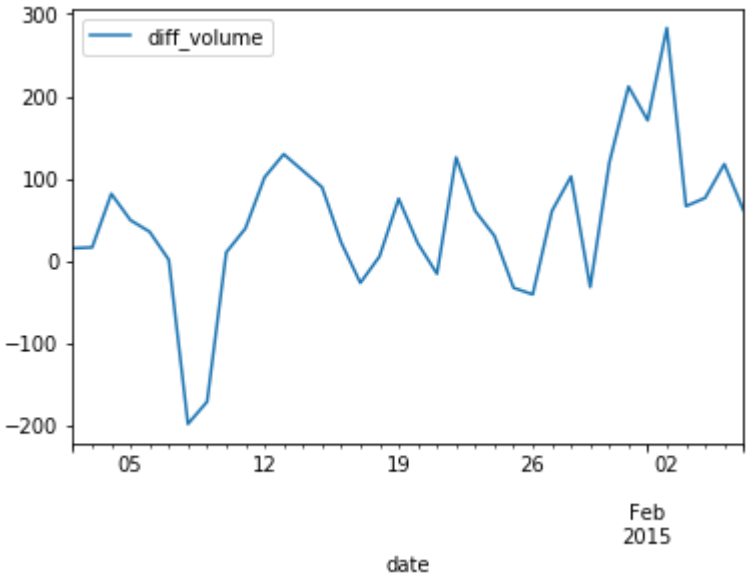
```
原始序列的白噪声检验结果为： (array([32.0111333]), array([1.53291527e-08]))
```

观察：

- 时序图显示该序列具有明显的单调递增趋势，可以判断为非平稳序列。
- 自相关图acf显示自相关系数长期大于零，说明序列间具有很强的长期相关性（不是纯随机序列，即不是白噪声序列）；如果自相关系数会很快衰减向0，可认为是平稳序列。
- ADF单位根检验进行平稳性检验，单位根检验统计量对应的p值（0.9984）显著大于0.05（pValue > 0.05），该序列可以判断为非平稳序列(非平稳序列一定不是白噪声序列)
  - adf = 1.8138
  - pValue = 0.9984
  - cValue = {1%：-3.7112, 5%: -2.9812, 10%: -2.6301}
- 白噪声统计量的p值 （1.53291527e-08） 小于显著性水平0.05，则表示该序列以95%拒绝纯随机的原假设，即认为序列不是纯随机序列，即非白噪声序列。

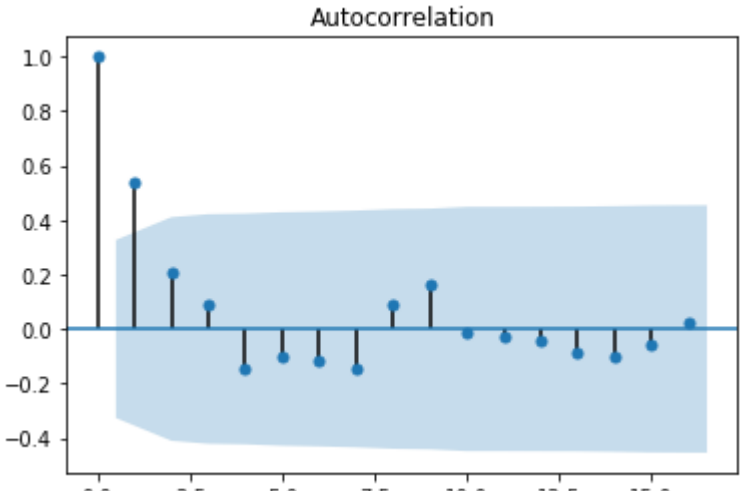
因此，对于非平稳序列，进行一阶差分运算，然后，进行平稳性和白噪声检验。

In [11]:
```
# 5. 差分后的时序图
%matplotlib inline
D_data = data.diff().dropna()
D_data.columns = ['diff_volume']
D_data.plot(); plt.show()  # 差分序列的时序图
```



In [12]:
```
# 6. 差分后结果：自相关图,偏自相关图
%matplotlib inline
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(D_data).show()  # 差分序列的自相关图

from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(D_data).show()  # 差分序列的偏自相关图
```

```
/home/lanman/anaconda3/lib/python3.7/site-packages/matplotlib/figure.py:445: UserWarning: Mat
plotlib is currently using module://ipykernel.pylab.backend_inline, which is a non-GUI backen
d, so cannot show the figure.
  % get_backend())
```



In [13]:
```
# 7. 差分后序列做 ADF平稳检验
print("差分序列的ADF检验结果为：")
ADF(D_data['diff_volume'])  # ADF平稳性检测
```

差分序列的ADF检验结果为：

Out[13]:
```
(-3.1560562366723537,
 0.022673435440048798,
 0,
 35,
 {'1%': -3.6327426647230316,
  '5%': -2.9485102040816327,
  '10%': -2.6130173469387756},
 287.5909090780334)
```

观察： 对原始序列进行一阶差分运算后，再进行平稳性和白噪声检验的过程同上。

- 差分后时序图：不再是单调递增趋势，在均值附近比较平稳的波动
- 差分后序列的自相关图：自相关系数有正有负，不是长期相关性，很强的短期相关性
- 差分后序列的单位根检验ADF的p值为0.0227（小于0.05），判断为平稳序列

因此，一阶差分后的序列是平稳序列，然后做白噪声检验。

In [14]:
```
# 8. 差分后为平稳序列，做白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("差分序列的白噪声检验结果为：",acorr_ljungbox(D_data['diff_volume'],lags=1,return_df=False))
```

差分序列的白噪声检验结果为： (array([11.30402222]), array([0.00077339]))

- 白噪声统计量的p值小于显著性水平0.05，则表示该序列可以95%拒绝纯随机的原假设，即认为序列不是纯随机序列，即非白噪声序列。

因此，对于一阶差分后 ($k=1$)的平稳非白噪声序列拟合ARMA模型。下面进行模型定阶，即确定 p 和 q

模型定阶的方法有三种

1. 人为识别的方法

根据前面ARMA模型的识别原则表进行模型定阶。

模 型	自相关系数（ACF）	偏自相关系数（PACF）
$ AR$ $(p) $	拖尾	p 阶截尾
$ MA$ $ (q) $	q 阶截尾	拖尾
$ARMA (p,q)$	p阶拖尾	q阶拖尾

观察一阶差分后的图:

- 自相关图，显示出 1阶截尾:（1）数据到后面有收敛趋势，因此没有拖尾；（2）自相关1阶拖尾（n从1开始缩至置信区间，即在淡蓝色区域内）
- 偏自相关图，显示出 1阶拖尾性:（1）数据到后面还有增大的情况，没有明显的收敛趋势；（2）偏自相关1阶拖尾（n从1开始缩至置信区间，即在淡蓝色区域内）

所以可以考虑用 $ MA(q=1)$ 模型拟合1阶差分($k=1$)后序列，即对原始序列建立$ARIMA(p,k,q)$模型。

模型参数：$p=0$，$k=1$，$q=1$ ==> 即建立 $ARIMA(0,1,1)$ 模型。

在应对单一序列模型，通过眼观图形和检验参数的显著性来判定，效果佳

2. 相对最优模型识别

使用AIC (Akaike Information Criterion) 或 BIC (Bayesian Information Criterion) 信息准则进行模型选择

用极大似然函数拟合模型(目标是最大化似然函数)，但也需要考虑模型复杂度，所以常常使用AIC和BIC作为模型优劣的衡量标准。 在多个备选模型中，最小AIC或BIC值的模型刻画的真实数据表达的信息损失最小，这是相对指标。

- AIC=-2 ln(L) + 2 k
- BIC=-2 ln(L) + ln(n)*k
  - 其中， k为模型参数个数，n为样本数量，L为似然函数

AIC 偏重拟合效果，BIC 对模型复杂度惩罚更重。

AIC在样本容量很大时，拟合所得数值会因为样本容量而放大(通常超过1000的样本称之为大样本容量)。AIC和BIC作为模型选择的准则，可以有效弥补根据自相关图和偏自相关图定阶的主观性。

计算 $ ARMA(p,q)$。当$p$ 和 q 均小于等于 3 的所有组合的 BIC 信息量，取其中 BIC 信息量达到最小的模型阶数。

计算完成 AIC 和 BIC 信息准则矩阵和各自选择的模型参数如下:

	0	1	2	3
0	436.825374	421.183469	422.903003	420.084145
1	419.650377	421.550767	421.817314	421.829314
2	421.588515	420.326151	422.036225	423.713017
3	423.213238	421.908073	423.782854	421.522162

AIC最小的p值和q值为：1、0

AIC信息准则矩阵

	0	1	2	3
0	438.408893	424.350507	427.653560	426.418221
1	422.817415	426.301323	428.151390	429.746909
2	426.339071	426.660227	429.953819	433.214130
3	429.547314	429.825668	433.283967	432.606794

BIC最小的p值和q值为：1、0

BIC信息准则矩阵

从图中可以看出，当 $p=1$ 和 $q=0$ 时，AIC和BIC最小值分别为：419.65 和 422.81。 p 和 q 定阶完成!

In [15]:
```
# 9. 定阶 (使用AIC和BIC两种度量中任一个即可)
from statsmodels.tsa.statespace.sarimax import SARIMAX

import warnings
warnings.simplefilter('ignore')
from warnings import catch_warnings,filterwarnings

pmax = int(len(D_data)/10) #一般阶数不超过length/10
qmax = int(len(D_data)/10) #一般阶数不超过length/10

data['volume'] = pd.DataFrame(data['volume'],dtype=float)  # 数据从int类型转为float

aic_matrix = [] #aic矩阵
bic_matrix = [] #bic矩阵
for p in range(pmax+1):
    tmpa = []; tmpb = []
    for q in range(qmax+1):
        try: #存在部分报错，所以用try来跳过报错。
            tmpa.append(SARIMAX(data, order=(p,1,q)).fit().aic)
            tmpb.append(SARIMAX(data, order=(p,1,q)).fit().bic)
        except:
            tmpa.append(None); tmpb.append(None)
    aic_matrix.append(tmpa)
    bic_matrix.append(tmpb)
```

In [16]:
```
# 找出AIC信息准则最小对应的p和q
aic_matrix = pd.DataFrame(aic_matrix) #从中可以找出最小值
print(aic_matrix)
p,q = aic_matrix.stack().idxmin() #先用stack展平，然后用idxmin找出最小值位置。
print('AIC最小的p值和q值为：%s、%s' %(p,q))
```

```
            0           1           2           3
0  436.825374  421.183469  422.903003  420.084145
1  419.650377  421.550767  421.817314  421.829314
2  421.588515  420.326151  422.036225  423.713017
3  423.213238  421.908073  423.782854  421.522162
AIC最小的p值和q值为：1、0
```

In [17]:
```
# 找出BIC信息准则最小对应的p和q
bic_matrix = pd.DataFrame(bic_matrix) #从中可以找出最小值
print(bic_matrix)
p,q = bic_matrix.stack().idxmin() #先用stack展平，然后用idxmin找出最小值位置。
print('BIC最小的p值和q值为：%s、%s' %(p,q))
```

```
            0           1           2           3
0  438.408893  424.350507  427.653560  426.418221
1  422.817415  426.301323  428.151390  429.746909
2  426.339071  426.660227  429.953819  433.214130
3  429.547314  429.825668  433.283967  432.606794
BIC最小的p值和q值为：1、0
```

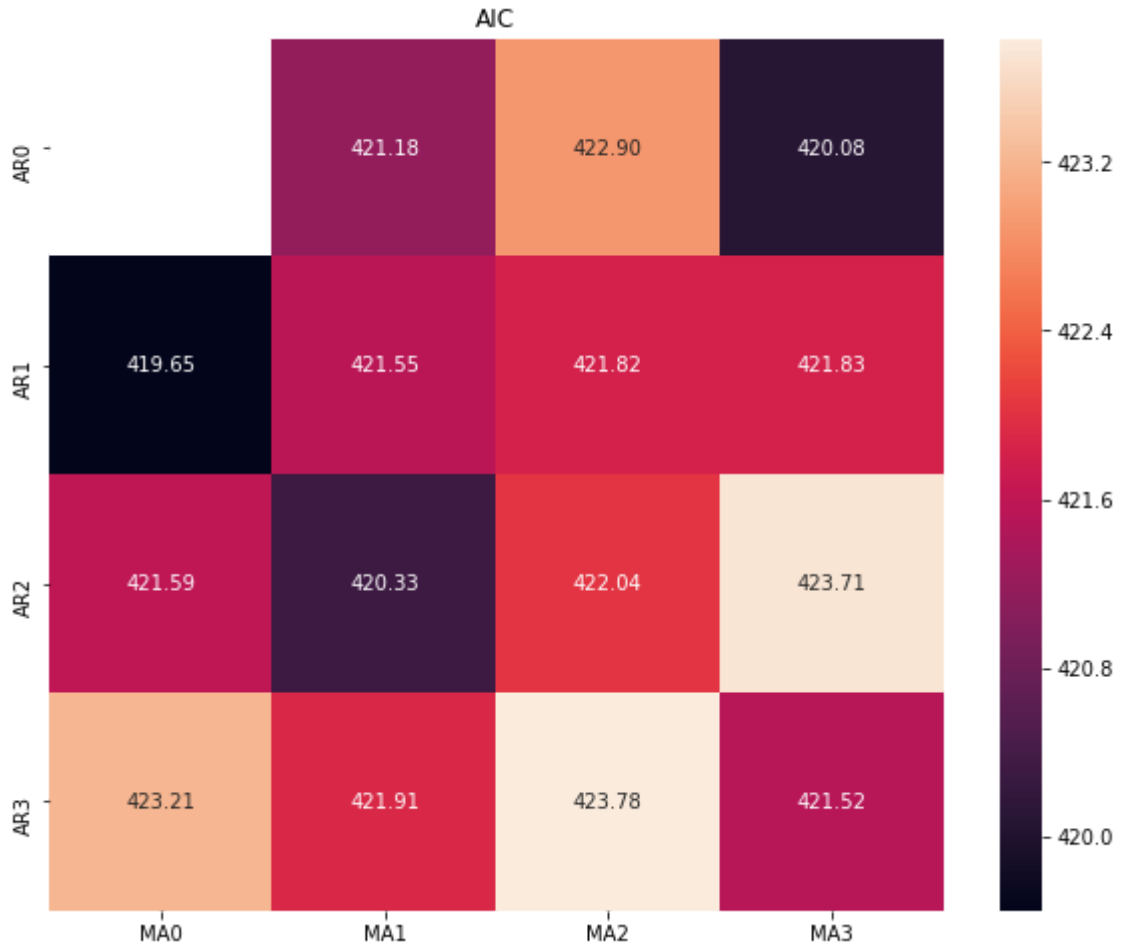
3. 热力图定阶

热力图定阶的方式和（2）信息准则定阶的方式类似，使用用热力图的方式呈现。

AIC值越小越好，颜色越深（黑色）的位置越好，可以看出， p，q取（1,0） （2,1） （0,3）都可以，通常是越小越好。

In [18]:
```
# 9.  3） 热力图定阶
from statsmodels.tsa.statespace.sarimax import SARIMAX
import seaborn as sns #热力图
import itertools

#设置遍历循环的初始条件，以热力图的形式展示，跟AIC定阶作用一样
pmax = 3; qmax = 3

# 创建Dataframe,以BIC准则
aic_matrix = pd.DataFrame(index=['AR{}'.format(i) for i in range(pmax+1)],columns=['MA{}'.format
# itertools.product 返回p,q中的元素的笛卡尔积的元组
for p,q in itertools.product(range(pmax+1), range(qmax+1)):
    if p==0 and q==0:
        aic_matrix.loc['AR{}'.format(p), 'MA{}'.format(q)] = None
        continue
    try:
        aic_matrix.loc['AR{}'.format(p), 'MA{}'.format(q)] = SARIMAX(data, order=(p,1,q)).fit().ai
    except:
        continue
aic_matrix = pd.DataFrame(aic_matrix[aic_matrix.columns], dtype=float) # 数据从int类型转为float
```

In [19]:
```
# 热力图展示AIC信息准则矩阵
fig, ax = plt.subplots(figsize=(10, 8))
ax = sns.heatmap(aic_matrix,
                 #mask=aic_matrix.isnull(),
                 ax=ax,
                 annot=True, #将数字显示在热力图上
                 fmt='.2f',
                 )
ax.set_title('AIC')
plt.show()
```



In [20]:
```
# 10. 给出一份模型报告
from statsmodels.tsa.statespace.sarimax import SARIMAX
ARIMA = SARIMAX(data, order=(p,1,q))
model = ARIMA.fit() # ARIMA(p,i,q)模型
model.summary()
```

Out[20]:

SARIMAX Results

Dep. Variable:	volume	**No. Observations:**	37
Model:	SARIMAX(3, 1, 3)	**Log Likelihood**	-203.761
Date:	Wed, 17 Mar 2021	**AIC**	421.522
Time:	12:56:34	**BIC**	432.607
Sample:	01-01-2015	**HQIC**	425.391
	- 02-06-2015		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4524	0.380	-1.189	0.234	-1.198	0.293
ar.L2	-0.4045	0.282	-1.434	0.152	-0.957	0.148
ar.L3	0.1475	0.364	0.405	0.686	-0.566	0.861
ma.L1	1.3045	2.819	0.463	0.643	-4.220	6.829
ma.L2	1.3912	14.816	0.094	0.925	-27.647	30.430
ma.L3	0.8274	10.271	0.081	0.936	-19.304	20.959
sigma2	4446.2697	5.45e+04	0.082	0.935	-1.02e+05	1.11e+05

Ljung-Box (L1) (Q):	0.13	**Jarque-Bera (JB):**	1.19
Prob(Q):	0.72	**Prob(JB):**	0.55
Heteroskedasticity (H):	1.78	**Skew:**	-0.10
Prob(H) (two-sided):	0.33	**Kurtosis:**	3.87

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

In [21]:
```
model.params
```

Out[21]:
```
ar.L1       -0.452411
ar.L2       -0.404496
ar.L3        0.147470
ma.L1        1.304490
ma.L2        1.391191
ma.L3        0.827425
sigma2    4446.269701
dtype: float64
```

对序列拟合模型进行分析

用 $AR(1)$ 模型拟合一阶差分后的序列，即对原始序列建立 $ ARIMA(1,1,0)$ 模型。 虽然两种方法建立的模型是一样的，但是模型是非唯一的，可以检验 $ ARIMA(0,1,1)$ 和 $ ARIMA(1,1,1)$，后面这两个模型也能通过检验。

下面对一阶差分后的序列拟合 $AR(1)$ 模型进行分析。

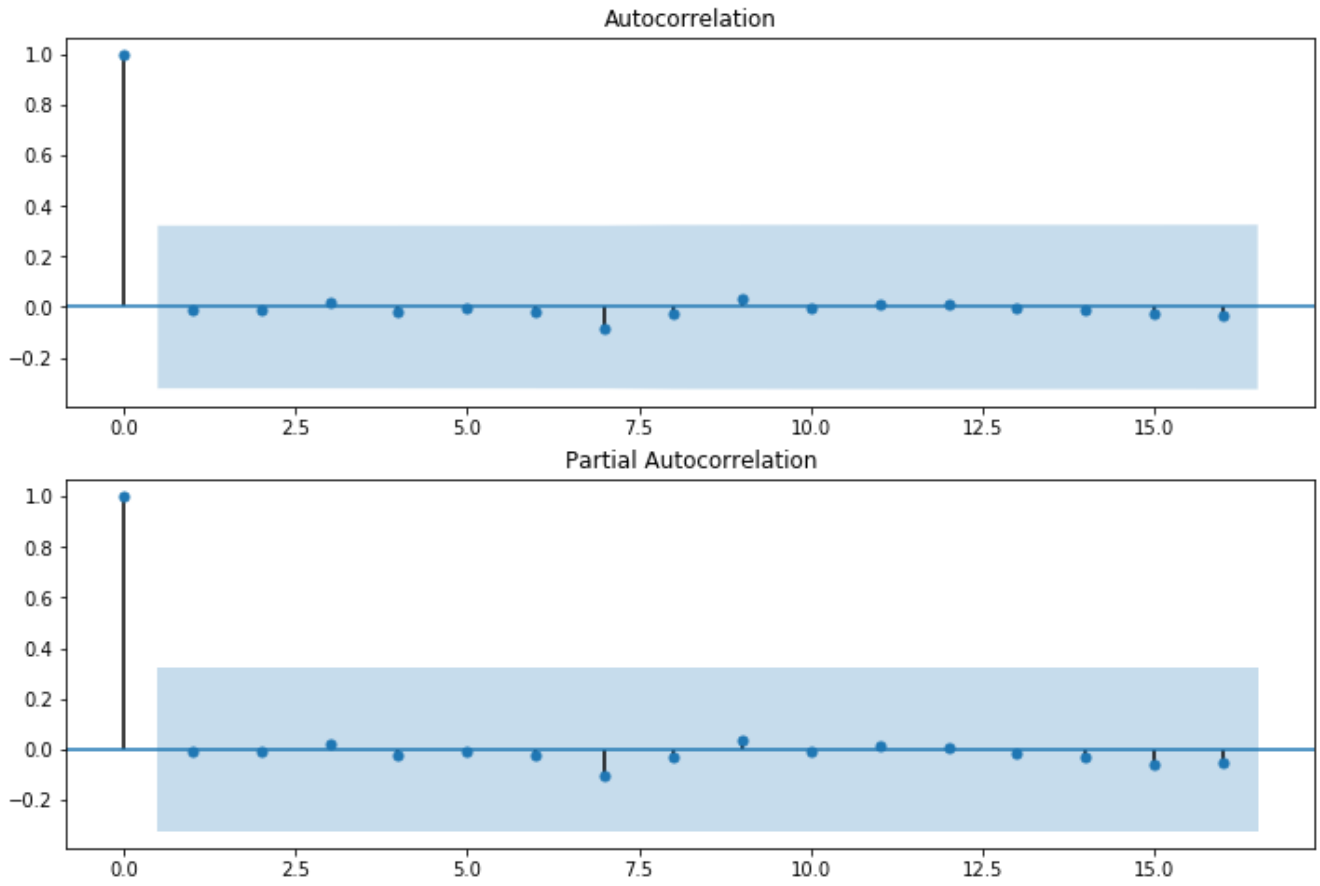
- 模型检验。对残差进行白噪声检验，残差为白噪声序列，p值为：0.9501
- 参数检验和参数估计见下图

	Coef.	Std.Err.	t	P>|t|	[0.025	0.975]
const	49.9564	20.1390	2.4806	0.0182	10.4847	89.4281
ma.L1.D.volume	0.6710	0.1648	4.0712	0.0003	0.3480	0.9941

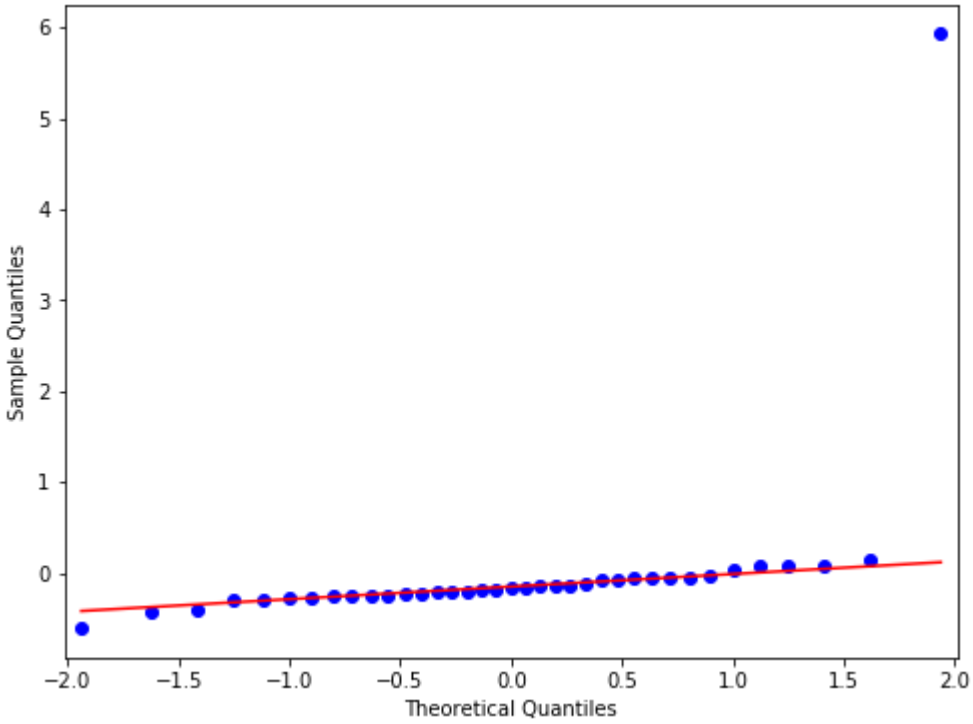
在指数平滑模型下，观察ARIMA模型的残差是否是平均值为0且方差为常数的正态分布（服从零均值、方差不变的正态分布），同时也要观察连续残差是否（自）相关

In [22]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(data, order=(1,1,0)).fit()  # ARIMA(1,1,0)模型
resid = model.resid # ARIMA模型的残差
```

In [23]:
```
# 对ARMA(1,0)模型所产生的残差做自相关图
fig = plt.figure(figsize=(12,8))
ax1 = fig.add_subplot(211)
fig = plot_acf(resid, ax=ax1)
ax2 = fig.add_subplot(212)
fig = plot_pacf(resid, ax=ax2)
```



In [24]:
```
# 观察残差是否符合正态分布
# 这里使用QQ图，它用于直观验证一组数据是否来自某个分布，或者验证某两组数据是否来自同一（族）分布。
# 在教学和软件中常用的是检验数据是否来自于正态分布。
from statsmodels.graphics.api import qqplot
fig = plt.figure(figsize=(8,6))
ax = fig.add_subplot(111)
fig = qqplot(resid, line='q', ax=ax, fit=True)
```



Ljung-Box test 是对randomness的检验,或者说是对时间序列是否存在滞后相关的一种统计检验。对于滞后相关的检验，常采用的方法还包括计算ACF和PCAF并观察其图像，但是无论是ACF还是PACF都仅仅考虑是否存在某一特定滞后阶数的相关。LB检验则是基于一系列滞后阶数，判断序列总体的相关性或者说随机性是否存在。

时间序列中一个最基本的模型就是高斯白噪声序列。而对于ARIMA模型，其残差被假定为高斯白噪声序列，所以当用ARIMA模型去拟合数据时，拟合后还要对残差的估计序列进行LB检验，判断其是否是高斯白噪声，如果不是，那么就说明ARIMA模型也许并不是一个适合样本的模型。

In [25]:
```
from statsmodels.tsa.stattools import acf as ACF
r1,q1,p1 = ACF(resid.values.squeeze(), qstat=True)
tmp = np.c_[list(range(0,36)), r1[1:], q1, p1]
table = pd.DataFrame(tmp, columns=['lag', "AC", "Q", "Prob(>Q)"])
print(table.set_index('lag')[:15])
```

```
            AC         Q  Prob(>Q)
lag
0.0  -0.008717  0.003046  0.955989
1.0  -0.008731  0.006188  0.996911
2.0   0.017997  0.019934  0.999256
3.0  -0.016783  0.032251  0.999871
4.0  -0.007195  0.034585  0.999988
5.0  -0.017643  0.049075  0.999998
6.0  -0.082429  0.375893  0.999786
7.0  -0.022928  0.402051  0.999942
8.0   0.031266  0.452431  0.999980
9.0  -0.005564  0.454086  0.999996
10.0  0.011957  0.462021  0.999999
11.0  0.008047  0.465758  1.000000
12.0 -0.006712  0.468467  1.000000
13.0 -0.010826  0.475820  1.000000
14.0 -0.029747  0.533860  1.000000
15.0 -0.031518  0.602121  1.000000
```

In [26]:
```
# 残差：残差的白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("残差的白噪声检验结果为：",acorr_ljungbox(resid,lags=1))  # 返回统计量和p值
```

```
残差的白噪声检验结果为： (array([0.00304573]), array([0.95598857]))
```

残差检验结果观察：

检验的结果就是看最后一列的检验概率（一般观察滞后1~12阶），如果检验概率小于给定的显著性水平（比如0.05、0.10等），就拒绝原假设（原假设是相关系数为零）。说明 ARIMA 模型也许并不是一个适合样本的模型。

就结果来看，取显著性水平为0.05，检验概率大于0.05，不能拒绝原假设（原假设是相关系数为零），因此，认为相关系数与零没有显著差异，即为白噪声序列。

白噪声检验的p值为： 0.9559， 认为相关系数与零没有显著差异，残差为白噪声序列。

ARIMA 模型预测

销售数据（data/arima_data.csv）包含2015年1月1日到2015年2月6日（共计36天）的销售数据。

应用 $ARIMA(1,1,0)$ 模型 对销售数据（data/arima_data.csv）做为期5天的预测，即预测2015年2月7日到2015年2月11日（共5天）的销售量，结果如下：

日期	2015/2/7	2015/2/8	2015/2/9	2015/2/10	2015/2/11
预测销售量	4874.0	4923.9	4973.9	5023.8	5073.8

说明： 利用模型向前预测的时期越长，预测误差将会越大，这是时间序列预测的典型特点。

In [27]:
```
forecastnum = 5  # 预测窗口
model.forecast(forecastnum, alpha=0.01)  # 作为期5天的预测，返回预测结果、标准误差、置信区间为99%
```

Out[27]:
```
2015-02-07    4856.351924
2015-02-08    4881.328922
2015-02-09    4897.182034
2015-02-10    4907.244138
2015-02-11    4913.630641
Freq: D, Name: predicted_mean, dtype: float64
```

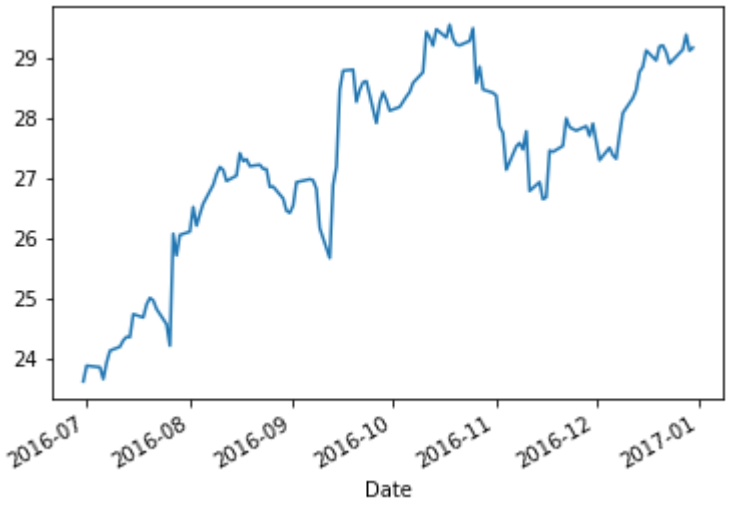
实例2：苹果股票价格预测

采集数据1：苹果股票2017

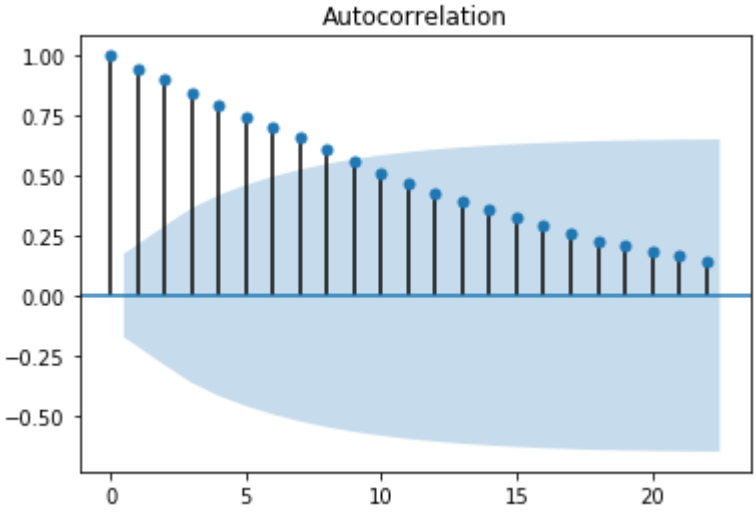
采集数据2：苹果股票2016-2（2016下半年开始）

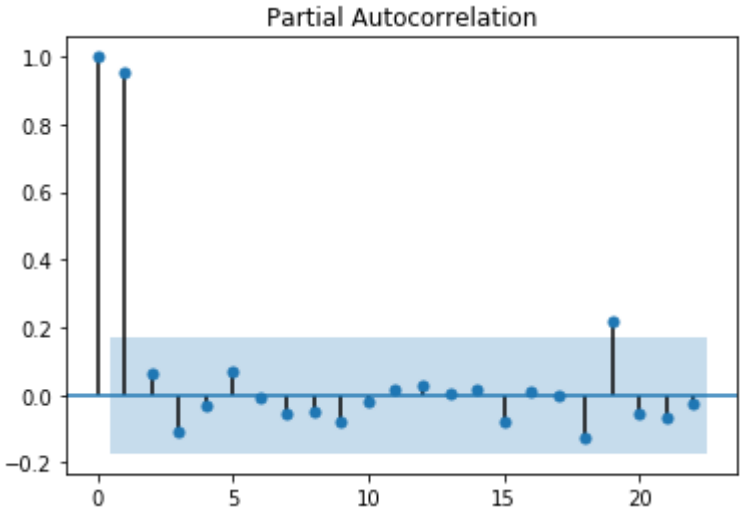
In [28]:
```
# 0. 采集Web股票数据并保存，使用Yahoo Finance的API获取股票数据
import pandas as pd
import numpy as np
from pandas.util.testing import assert_frame_equal
import pandas_datareader as pdr
stock = pdr.get_data_yahoo('AAPL', start = '2016/7/1',end = '2016/12/31') # 苹果股票
out_file = open('data/aapl2016-2.csv', 'w')
stock.to_csv(out_file)
out_file.close()
```

In [29]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt

# 1. 加载数据
import pandas as pd
data = pd.read_csv('data/aapl2016-2.csv', index_col='Date')  # 使用Date列作为行索引
data.index = pd.to_datetime(data.index)  # 将字符串索引转换成时间索引
data['Open'].plot()  # 绘图原始序列的时序图
plt.show()
```



In [30]:
```
# 2. 计算绘制acf自相关系数图和pcaf偏自相关图
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(data['Open']).show()  # 绘制原始序列的自相关图
plot_pacf(data['Open']).show()  # 绘制原始序列的偏自相关图
```





In [31]:
```
# 3. ADF平稳性检验进行单位根检验，常用来检验金融、经济时间序列数据的平稳性
from statsmodels.tsa.stattools import adfuller as ADF  # 导入ADF检验，对观测值序列进行单位根检验
print("原始序列的ADF检验结果为:",ADF(data['Open']))  # ADF unit root test,返回的结果有ADF值、p值等
```

```
原始序列的ADF检验结果为: (-2.193183161680527, 0.20872887258194878, 0, 127, {'1%': -3.482920063655088, '5%': -2.884580323367261, '10%': -2.5790575441750883}, 96.53993577223792)
```

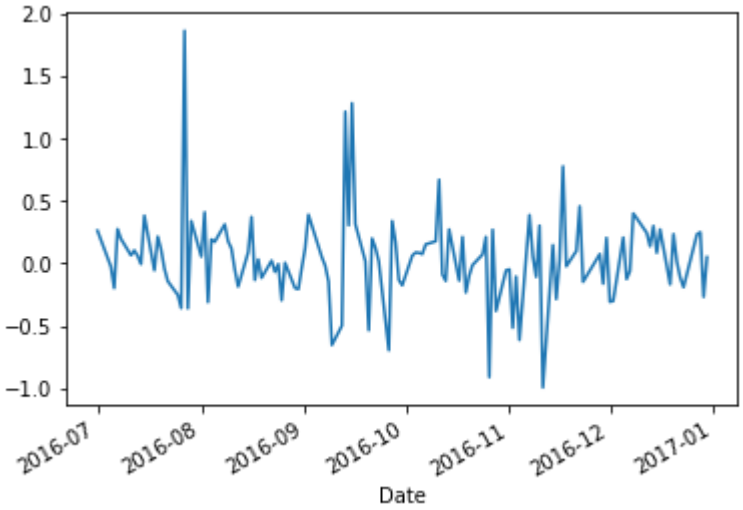
In [32]:
```
# 4. 白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("原始序列的白噪声检验结果为：",acorr_ljungbox(data['Open'],lags=1))  # 返回统计量和p值,如果
```

原始序列的白噪声检验结果为： (array([116.89539636]), array([3.02614415e-27]))

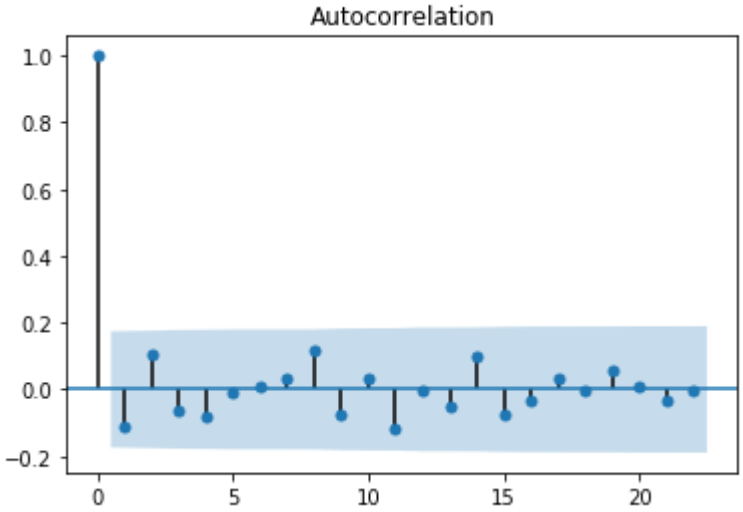
- ADF平稳性检验的p值 大于 显著性水平 0.05,表示该序列为非平稳
- 白噪声统计量的p值 小于 显著性水平0.05，认为序列是非白噪声序列。

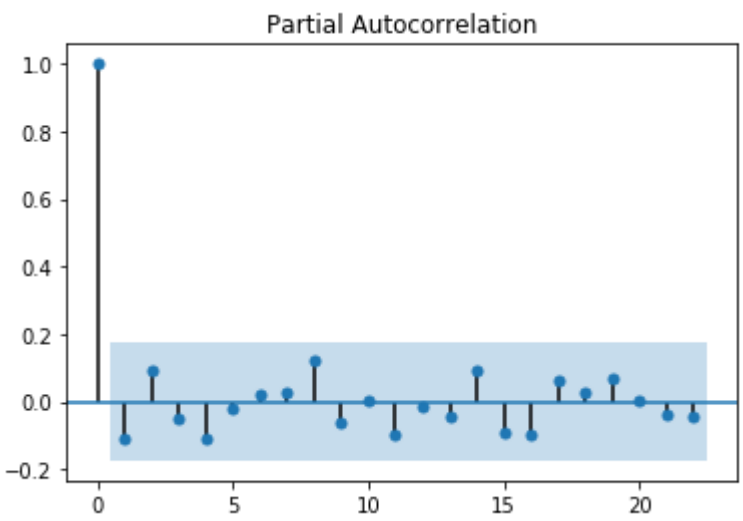
因此，对于非平稳序列，进行一阶差分运算，然后，进行平稳性和白噪声检验。

In [33]:
```
# 5. 差分后结果: 自相关图,偏自相关图和ADF检验
D_data = data['Open'].diff().dropna()
D_data.columns = ['diff_Open']
D_data.plot(); plt.show()  # 差分序列的时序图

from statsmodels.graphics.tsaplots import plot_acf
plot_acf(D_data).show()  # 差分序列的自相关图

#from statsmodels.tsa.stattools import pacf as PACF
#print "差分序列的PACF自相关系数为: ",PACF(D_data['diff_volume'])  # 返回差分序列的PACF自相关系数
from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(D_data).show()  # 差分序列的偏自相关图

print("差分序列的ADF检验结果为: ", ADF(D_data))  # ADF平稳性检验的p值 < 0.05, 表示为平稳序列
```



差分序列的ADF检验结果为: (-12.463998171674886, 3.3570893074352296e-23, 0, 126, {'1%': -3.4833462346078936, '5%': -2.8847655969877666, '10%': -2.5791564575459813}, 100.46776285222103)



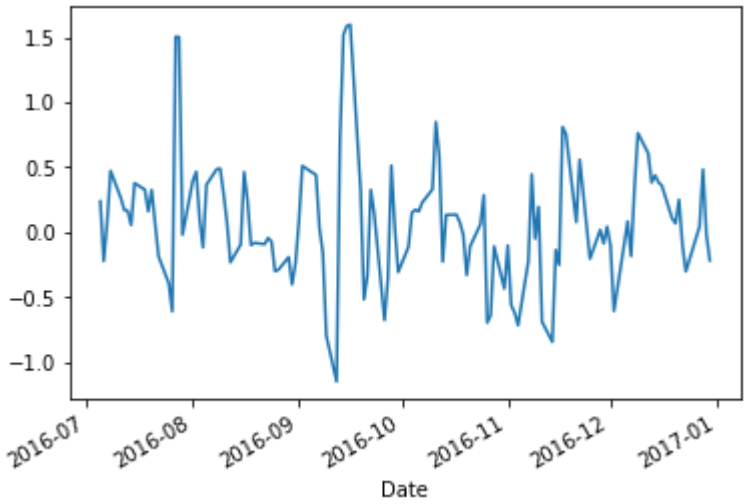


In [34]:
```
# 因此，一阶差分后的序列是平稳序列，然后做白噪声检验。
# 6. 差分后为平稳序列，做白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("差分序列的白噪声检验结果为：",acorr_ljungbox(D_data,lags=1))  # 返回统计量和p值
#差分序列的白噪声检验结果为： (array([ 1.507054]), array([ 0.21958917]))
```

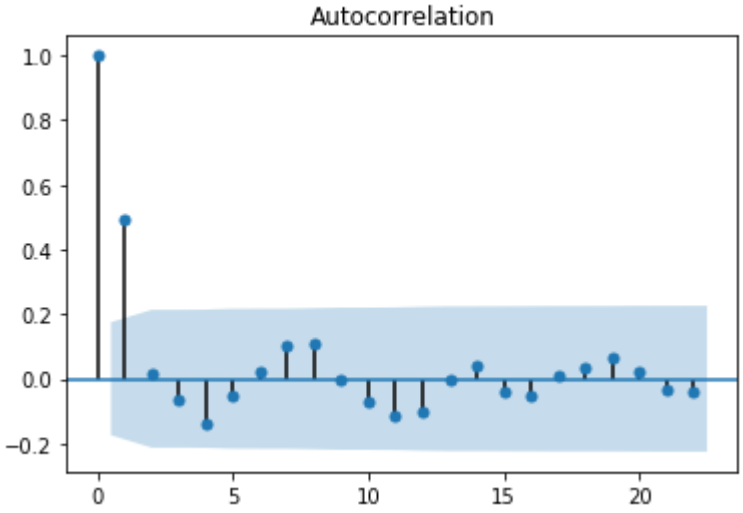
差分序列的白噪声检验结果为： (array([1.5941726]), array([0.206731]))

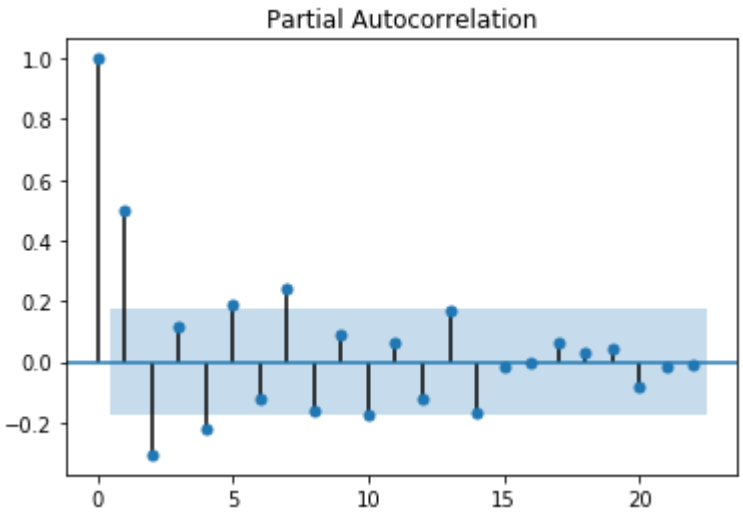
- 白噪声统计量的p值 > 0.05，则表示认为序列是纯随机序列，即白噪声序列。停止分析？！

In [35]:
```
# 7. 二阶差分后结果: 自相关图,偏自相关图和ADF检验
D2_data = data['Open'].diff(periods=2).dropna()
D2_data.columns = ['diff_Open']
D2_data.plot(); plt.show()  # 差分序列的时序图

from statsmodels.graphics.tsaplots import plot_acf
plot_acf(D2_data).show()  # 差分序列的自相关图

from statsmodels.graphics.tsaplots import plot_pacf
plot_pacf(D2_data).show()  # 差分序列的偏自相关图

print("二阶差分序列的ADF检验结果为：", ADF(D2_data))  # ADF平稳性检验的p值 < 0.05，表示为非平稳序列
# 6. 差分后为平稳序列，做白噪声检验
from statsmodels.stats.diagnostic import acorr_ljungbox
print("二阶差分序列的白噪声检验结果为：",acorr_ljungbox(D2_data,lags=1))  # 返回统计量和p值 ,p值 <
#差分序列的白噪声检验结果为： (array([ 1.507054]), array([ 0.21958917]))
```



二阶差分序列的ADF检验结果为： (-3.5665371045196204, 0.0064339508900736414, 7, 118, {'1%': -3.4870216863700767, '5%': -2.8863625166643136, '10%': -2.580009026141913}, 118.05131362276387)
二阶差分序列的白噪声检验结果为： (array([31.63447239]), array([1.86096631e-08]))





观察1：

只观察2017年1月之后的苹果股票数据（data/aapl2017.csv），共计67天的股票数据（截至到4月9日），分析结果：

- 原始序列，ADF平稳性检验 p > 0.05, 不平稳；
- 一阶差分后，ADF平稳性检验的p值 > 0.05，不平稳
- 二阶差分后，ADF平稳性检验的p值 < 0.05，平稳；白噪声检验 p值 < 0.05,非白噪声；$\to$ 采用ARMA模型定阶。
- 三阶差分后，ADF平稳性检验的p值 = 0.2558 > 0.05，不平稳
- 四阶差分后，ADF平稳性检验的p值 = 0.2084 > 0.05，不平稳
- 五阶差分后，ADF平稳性检验的p值 = 0.2475 > 0.05，不平稳
- 六阶差分后，ADF平稳性检验的p值 = 0.4464 > 0.05，不平稳
- 七阶差分后，ADF平稳性检验的p值 = 0.5220 > 0.05，不平稳

看到一阶差分后还是不平稳，二阶差分后还是不平稳，甚至做到七阶、八阶差分还是不平稳，怎么办?

原因：（1）可能数据序列是爆炸性增长的，一般不可持久；（2）还可能是因为序列数据太少或观察值不准确，还不足以反映变化规律，这时要加多观测值；（3）外在其他影响因素。

观察2：

扩大数据采集的范围，从2016年7月之后的苹果股票数据（data/aapl2016-2.csv），共计194天的股票数据（截至到4月9日），分析结果：

- 原始序列，ADF平稳性检验 p = 0.9435 > 0.05, 不平稳；白噪声
- 一阶差分后，ADF平稳性检验的p值 < 0.05，平稳；白噪声检验的p = 0.1274 > 0.05, 白噪声
- 二阶差分后，ADF平稳性检验的p值 = 0.0021 < 0.05，平稳；白噪声检验的p < 0.05, 非白噪声

因此，对于二阶差分后 (k=2)的平稳非白噪声序列拟合ARMA模型。下面进行模型定阶，即确定 p 和 q

In [36]:
```
# 7. 定阶
from statsmodels.tsa.statespace.sarimax import SARIMAX

pmax = 3; qmax = 3
data['Open'] = data['Open'].astype(float)  # 数据从int类型转为float

bic_matrix = [] #bic矩阵
for p in range(pmax+1):
    tmp = []
    for q in range(qmax+1):
        try: #存在部分报错，所以用try来跳过报错。
            tmp.append(SARIMAX(data['Open'], order=(p,2,q)).fit().bic) # ARIMA(p,2,q)模型
        except:
            tmp.append(None)
    bic_matrix.append(tmp)

bic_matrix = pd.DataFrame(bic_matrix) #从中可以找出最小值
print(bic_matrix)

p,q = bic_matrix.stack().idxmin() #先用stack展平，然后用idxmin找出最小值位置。
print('BIC最小的p值和q值为：%s、%s' %(p,q))

# BIC最小(112.7)的p值和q值为：0、1
```

```
            0           1           2           3
0  203.475903  112.756961  116.471587  119.817883
1  153.333070  116.247642  120.224797  125.116853
2  147.814907  119.832844  125.082216  125.774976
3  148.729137  124.483709  129.325860  130.254271
BIC最小的p值和q值为：0、1
```

结果：

- 得到 BIC最小(112.7)的p值和q值为：0、1
- 肉眼观察二阶差分后的自相关图和偏自相关图：
  - 自相关收敛，截尾， 1阶
  - 偏自相关不收敛，拖尾， 6阶

确定，建立模型 $ARIMA(0,2,1)$。

In [37]:
```
# 分割数据构建模型和测试模型
train = data[:-5]['Open']
test = data[-5:]['Open']

from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(train, order=(p,2,q)).fit()
forecastnum = 5  # 预测窗口，未来5天
yHat = model.forecast(forecastnum,alpha=0.01)  # 提高置信区间为99%
```

In [38]:
```
print("预测值：\n", yHat) # 预测结果
print("真实值：\n", test) # 真实值
```

```
预测值：
 123    29.132361
124    29.177223
125    29.222084
126    29.266945
127    29.311807
Name: predicted_mean, dtype: float64
真实值：
 Date
2016-12-23    28.897499
2016-12-27    29.129999
2016-12-28    29.379999
2016-12-29    29.112499
2016-12-30    29.162500
Name: Open, dtype: float64
```

In [39]:
```
# 1. 计算 MSE(均方误差)
def MSE(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    import numpy as np
    return ((yArr-yHatArr)**2).sum() / len(yArr)

# 均方误差
MSE(test.tolist(), yHat)
```

Out[39]: 0.02569468735387922

In [40]:
```
# 2. 调用 MSE(均方误差)
from sklearn.metrics import mean_squared_error
mean_squared_error(yHat, test)
```

Out[40]: 0.02569468735387922

In [41]:
```
# 3. 计算绝对平均误差
def absError(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    return abs(yArr-yHatArr).sum() / len(yArr)

# 平均绝对误差
absError(test.tolist(),yHat)
```

Out[41]: 0.14875062702175654

实例3：以魁北克月度汽车销量为例，探讨序列预测的特征选择

要将机器学习算法应用于时间序列数据，需要特征工程的帮助。例如，单变量的时间序列数据集由一系列观察结果组成，它们必须被转换成输入和输出特征，才能用于监督性学习算法。

但这里有一个问题：针对每个时间序列问题，可以处理的特征类型和数量，却并没有明确的限制。当然，古典的时间序列分析工具（如相关图correlogram）可以帮助评估滞后变量（lag variables），但并不能直接帮助开发者对其他类型的特征进行选择，例如从时间戳（年、月、日）和移动统计信息（如移动平均线moving average）衍生的特征。

因此，第三个实例着重针对时间序列预测的特征选择，探讨如何利用基于特征重要性和特征选择的机器学习工具处理时间序列问题。通过实例3的学习，将了解：

- 如何创建和解释滞后观察的相关图。
- 如何计算和解释时间序列特征的重要性得分。
- 如何对时间序列输入变量进行特征选择。

共分为如下六个部分：

- 1). 载入每月汽车销量数据集：即载入我们将要使用的数据集。
- 2). 平稳化：讲述如何使数据集平稳化，以便于后续的分析和预测。
- 3). 自相关图：讲述如何创建时间序列数据的相关图。
- 4). 时间序列到监督学习：将时间单变量的时间序列转化为监督性学习问题。
- 5). 滞后变量的特征重要性：讲述如何计算和查看时间序列数据的特征重要性得分。
- 6). 滞后变量的特征选择：讲述如何计算和查看时间序列数据的特征选择结果。

1. 载入数据：载入每月汽车销量数据集

魁北克在 1960 到 1968 年的月度汽车销量的原始数据可以在如下链接下载：

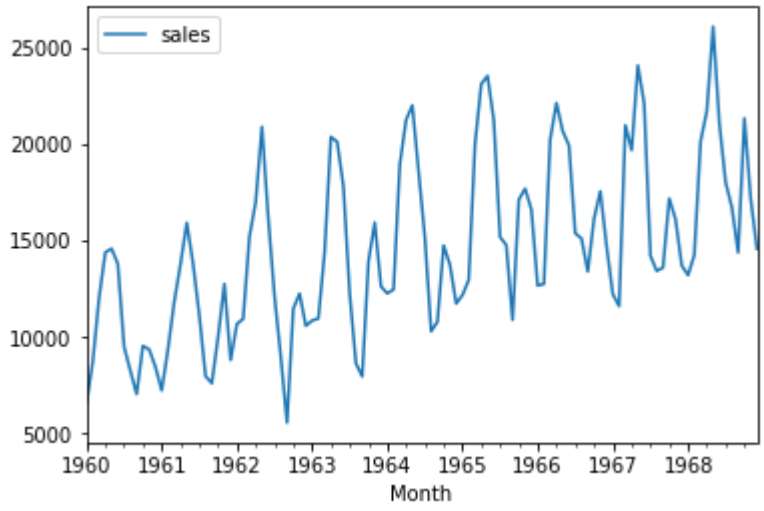
https://datamarket.com/data/set/22n4/monthly-car-sales-in-quebec-1960-1968
(https://datamarket.com/data/set/22n4/monthly-car-sales-in-quebec-1960-1968)

下载后的数据保存为 data/car-sales.csv 文件，同时删去了文件中的脚注信息。

In [42]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 加载该数据集
data = pd.read_csv('data/car-sales.csv', header=0, index_col='Month')  # 使用date列作为行索引
data.index = pd.to_datetime(data.index)
data.columns = ['sales'] # 修改列名 "Monthly car sales in Quebec 1960-1968"
print(data.head()) # display first few rows
data.plot()  # line plot of dataset
```

```
            sales
Month
1960-01-01   6550
1960-02-01   8728
1960-03-01  12026
1960-04-01  14395
1960-05-01  14587
```

Out[42]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f072c6317d0>
```



2. 平稳化: 如何使数据集平稳化，便于后续的分析和预测

从上图可以看到汽车销量数据明显的季节性和日益增长的变化趋势。这种季节性的变化和增长趋势虽然可以作为序列预测的关键特征，但如果需要探索其他的有助于做出序列预测的系统信号，就必须将它们移除。

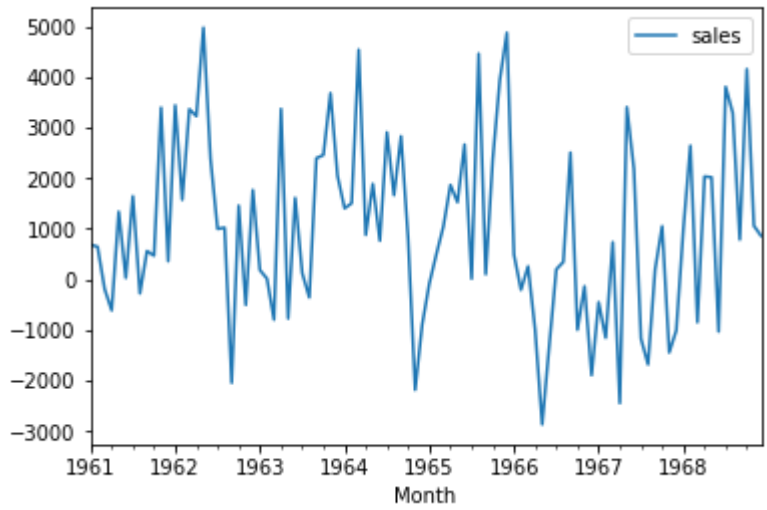
通常，将除去了季节性变化和增长趋势的时间序列称为平稳化序列。

为了消除这种季节性变化，通常采取季节差分的办法，即生成所谓的季节性适配时间序列（seasonally adjusted time series）。本例中季节性变化的变化周期似乎是一年（12个月）。

下面的代码展示了如何计算季节性适配时间序列，并将结果保存到文件 data/seasonally-adjusted.csv 代码中，由于最初的 12 个月没有更早的数据用以差分计算，因此被丢弃。最终得到的季节差分结果如下图所示：

In [43]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 加载原始数据集
data = pd.read_csv('data/car-sales.csv', header=0, index_col='Month')  # 使用date列作为行索引
data.index = pd.to_datetime(data.index)
data.columns = ['sales'] # 修改列名 "Monthly car sales in Quebec 1960-1968"

# 季节性适配时间序列
D_data = data.diff(12).dropna()  # seasonal difference,trim off the first year of empty data
D_data.to_csv('data/seasonally_adjusted.csv') # save differenced dataset to file
D_data.plot()
#代码中，由于最初的 12 个月没有更早的数据用以差分计算，因此被丢弃。最终得到的季节差分结果如下图所
```

Out[43]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x7f072c622e10>
```



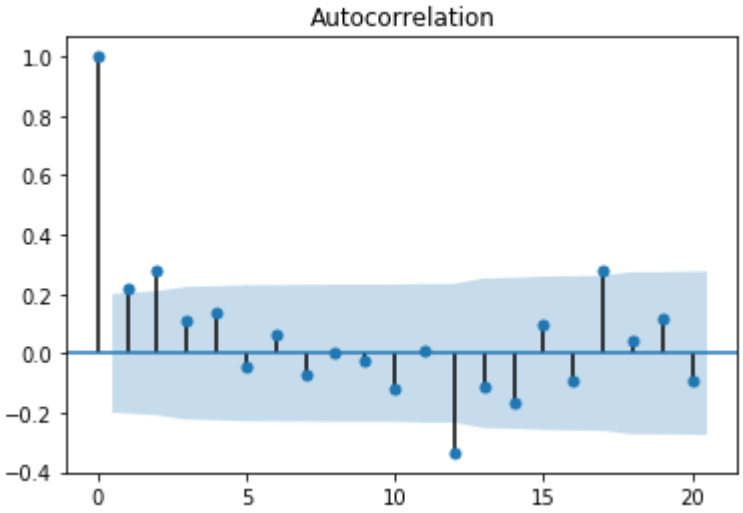
从图中可以看出，通过差分运算消除了季节性变化和增长趋势信息。

3. 自相关图: 讲述如何创建时间序列数据的相关图

通常可以根据与输出变量的相关性来选择时间序列的特征。这被称为自相关（autocorrelation），包括如何绘制自相关图，也称为相关图。 自相关图展示了每个滞后观察结果的相关性，以及这些相关性是否具有统计学的显著性。

下面的代码绘制了月汽车销量数据集中所有滞后变量的相关图。运行后可以得到一张相关图，或自相关函数（ACF）图。

In [44]:
```
import pandas as pd
from statsmodels.graphics.tsaplots import plot_acf

D_data = pd.read_csv('data/seasonally_adjusted.csv', index_col='Month')  # 使用date列作为行索引
D_data.index = pd.to_datetime(D_data.index)
plot_acf(D_data).show()
```



观察：

- 图中 x 轴表示滞后值，y 轴上 -1 和 1 之间则表现了这些滞后值的正负相关性。
- 蓝色区域中的点表示统计学显著性。滞后值为 0 相关性为 1 的点表示观察值与其本身 100% 正相关。
- 可以看到，图中在 1,2,12 和 17 个月显示出了显著的滞后性（即超出图中淡蓝色区域的四个点）。

这个分析为后续的比较过程提供了一个很好的基准。

4. 时间序列到监督学习: 将时间单变量的时间序列转化为监督性学习问题

通过将滞后观察（例如 t-1）作为输入变量，将当前观察（t）作为输出变量，可以将单变量的月度汽车销量数据集转换为监督学习问题。

为了实现这一转换，在下面的代码中调用了 Pandas 库中的 shift 函数，通过 shift 函数可以为转换后的观察值创建新的队列。

下面示例中，创建了一个包含 12 个月滞后值的新时间序列，以预测当前的观察结果。代码中 12 个月的迁移表示前 12 行的数据不可用，因为它们包含 NaN 值。

In [45]:
```
import pandas as pd
# load dataset
D_data = pd.read_csv('data/seasonally_adjusted.csv', index_col='Month')  # 使用date列作为行索引
D_data.index = pd.to_datetime(D_data.index)
# reframe as supervised learning
S_data = pd.DataFrame()
for i in range(12,0,-1):
    S_data['t-'+str(i)] = D_data['sales'].shift(i)
S_data['t'] = D_data.values
S_data = S_data[13:]  # drop 前面12行数据
# save to new file
S_data.to_csv('data/lags_12months_features.csv', index=False)

print(S_data.head())
```

```
             t-12    t-11    t-10     t-9     t-8     t-7     t-6     t-5  \
Month
1962-02-01   646.0  -189.0  -611.0  1339.0    30.0  1645.0  -276.0   561.0
1962-03-01  -189.0  -611.0  1339.0    30.0  1645.0  -276.0   561.0   470.0
1962-04-01  -611.0  1339.0    30.0  1645.0  -276.0   561.0   470.0  3395.0
1962-05-01  1339.0    30.0  1645.0  -276.0   561.0   470.0  3395.0   360.0
1962-06-01    30.0  1645.0  -276.0   561.0   470.0  3395.0   360.0  3440.0

              t-4     t-3     t-2     t-1       t
Month
1962-02-01   470.0  3395.0   360.0  3440.0  1573.0
1962-03-01  3395.0   360.0  3440.0  1573.0  3363.0
1962-04-01   360.0  3440.0  1573.0  3363.0  3226.0
1962-05-01  3440.0  1573.0  3363.0  3226.0  4974.0
1962-06-01  1573.0  3363.0  3226.0  4974.0  2384.0
```

注意：

- 将前 12 行的数据删除，然后将结果保存在 lags_12months_features.csv 文件中。
- 实际上，这个过程可以在任意的时间步长下重复进行，例如 6 或 24 个月，感兴趣的朋友可以自行尝试

5. 滞后变量的特征重要性：如何计算和查看时间序列数据的特征重要性得分

如何计算特征值的重要性得分？

- 各种决策树，例如 bagged 树和随机森林等，都可以用来，这些是机器学习中的常见用法，以便在开发预测模型时有效评估输入特征的相对有效性。

这里可以通过正要性得分，来帮助评估时间序列预测输入特征的相对重要性。这点非常重要，不仅可以设计上述提到的滞后观察特征，还可以设计基于观测时间戳、滚动统计等其他类型的特征。因此，特征重要性是整理和选择特征时非常有效的一种方法。

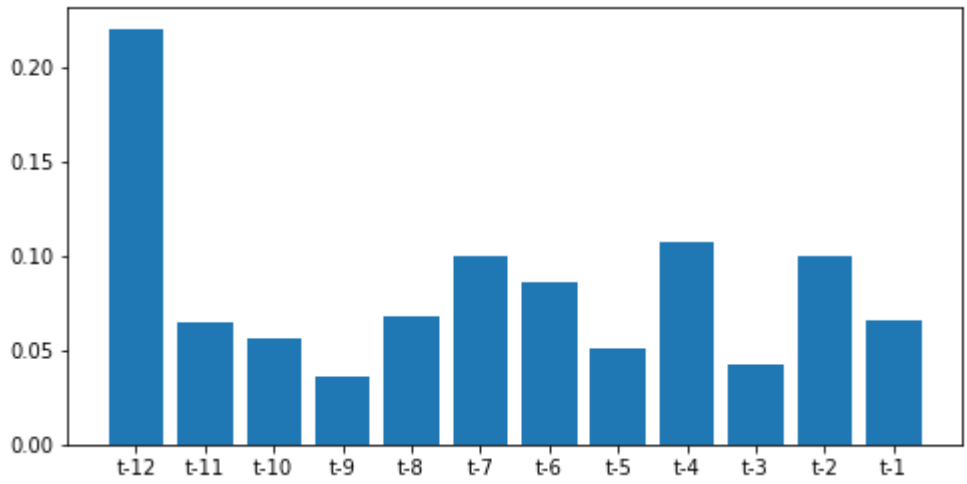
下面的实例中，加载了上面创建的数据集的监督性学习视图，然后利用随机森林模型（代码中为 RandomForestRegressor），总结了 12 个滞后观察中每一个的相对特征重要性得分。

这里使用了大数量的树来保证得分的稳定性。此外，还用到了随机种子初始化（the random number seed is initialized），用以保证每次运行代码时都能获得相同的结果。

运行示例后，首先打印了滞后观察值的重要性得分，然后将得分绘制为条形图，如图所示。

In [46]:
```
import pandas as pd
from sklearn.ensemble import RandomForestRegressor

%matplotlib inline
import matplotlib.pyplot as plt

# load data
S_data = pd.read_csv('data/lags_12months_features.csv', header=0)
array = S_data.values
# split into input and output
X = array[:,0:-1]  # 前12列为12个滞后值，即t-12,t-11,t-10,...,t-1
y = array[:,-1] # 最后一列为t值

# fit random forest model
model = RandomForestRegressor(n_estimators=50, random_state=1)
model.fit(X, y)

# show importance scores
print("滞后观察值的重要性得分:")
print(model.feature_importances_)

# plot importance scores
names = S_data.columns.values[0:-1]
ticks = [i for i in range(len(names))]
plt.rc('figure', figsize=(8,4))
plt.bar(ticks, model.feature_importances_)
plt.xticks(ticks, names)
plt.show()
```

```
滞后观察值的重要性得分:
[0.220692   0.06444966 0.05597269 0.03637144 0.06798764 0.10012901
 0.08632594 0.05120944 0.10782591 0.0422795  0.10044342 0.06631336]
```



观察：

运行示例后，首先打印了滞后观察值的重要性得分，如下所示。然后将得分绘制为条形图，如图所示。

- 图中显示 t-12 观测值的相对重要性最高，其次就是 t-2,t-4 和 t-7。
- 感兴趣的可以仔细研究这个结果与上述自相关图的差异。
- 这里还可以用 gradient boosting，extra trees，bagged decision trees 等代替随机森林模型，同样可以计算特征的重要性得分

6. 滞后变量的特征选择： 如何计算和查看时间序列数据的特征选择结果

接下来，通过特征选择来自动识别并选择出最具预测性的输入特征。

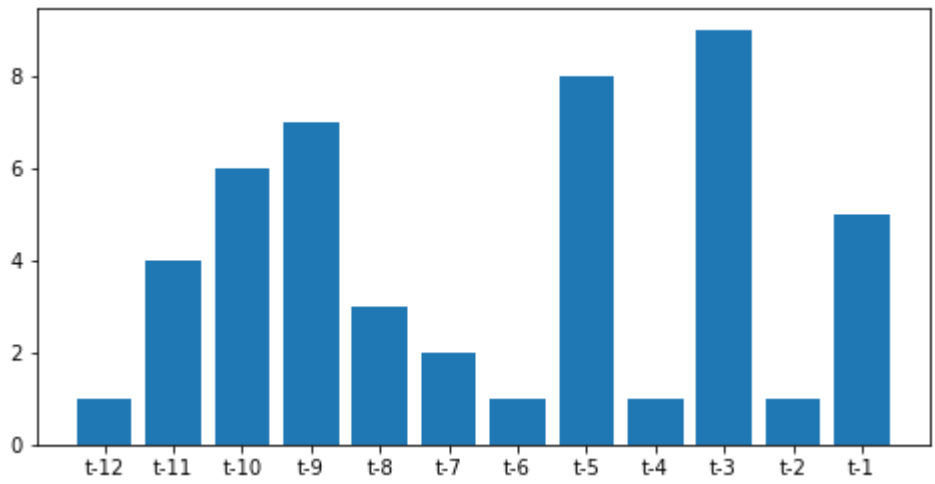
目前，特征选择最流行方法是递归特征选择（Recursive Feature Selection，RFE）。RFE 可以创建预测模型，对特征值赋予不同的权值，并删掉那些权重最小的特征，通过不断重复这一流程，最终就能得到预期数量的特征。

以下示例中演示了如何通过RFE与随机森林模型进行特征选择，注意其中输入特征的预期数量设置的是 4。

运行以上示例后，可以得到如下 4 个待选特征。 Selected Features: t-12 t-6 t-4 t-2 可见，这一结果与上一节由重要性得分得到的结果相一致。

In [47]:
```
import pandas as pd
%matplotlib inline
import matplotlib.pyplot as plt

from sklearn.feature_selection import RFE
from sklearn.ensemble import RandomForestRegressor

# load dataset
S_data = pd.read_csv('data/lags_12months_features.csv', header=0)
array = S_data.values
# separate into input and output variables
X = array[:,0:-1]
y = array[:,-1]

# perform feature selection
rfe = RFE(RandomForestRegressor(n_estimators=500, random_state=1), 4)
fit = rfe.fit(X, y)

# report selected features
print('Selected Features:')
names = S_data.columns.values[0:-1]
for i in range(len(fit.support_)):
    if fit.support_[i]:
        print((names[i]))

# plot feature rank
plt.rc('figure', figsize=(8,4))
ticks = [i for i in range(len(names))]
plt.bar(ticks, fit.ranking_)
plt.xticks(ticks, names)
plt.show()
```

```
Selected Features:
t-12
t-6
t-4
t-2
```



结果观察：

程序创建一个条形图，显示了每个待选输入特征的选择排序（数字越小越好）。同样，感兴趣还可以设置不同的预期特征数量，或者换用随机森林之外的其他模型。

7. 历史汽车销量的敏感性分析

In [48]:
```
# 启动绘图
%matplotlib inline
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np

# 加载原始数据集
data = pd.read_csv('data/car-sales.csv', header=0, index_col='Month')  # 使用date列作为行索引
data.index = pd.to_datetime(data.index)
data.columns = ['sales'] # 修改列名 "Monthly car sales in Quebec 1960-1968"

# 季节性适配时间序列,这也意味着第一年的数据将无法用于建模，因为第一年并没有更早的数据。
D_data = data.diff(12).dropna()  # seasonal difference,trim off the first year of empty data
D_data.tail() # 显示最后5条
```

Out[48]:

Month	sales
1968-08-01	3288.0
1968-09-01	787.0
1968-10-01	4155.0
1968-11-01	1061.0
1968-12-01	864.0

In [49]:
```
# 将数据导入 ARIMA(7,0,0) 模型，并打印输出汇总信息。
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = SARIMAX(D_data, order=(7,0,0)).fit(trend='nc', disp=0)
print(model.summary())

#forecastnum = 5  # 预测窗口，未来5天
#yHat = model.forecast(forecastnum,alpha=0.01)  # 提高置信区间为99%
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                  sales   No. Observations:                   96
Model:               SARIMAX(7, 0, 0)   Log Likelihood                -850.579
Date:                Wed, 17 Mar 2021   AIC                           1717.159
Time:                        12:57:30   BIC                           1737.674
Sample:                    01-01-1961   HQIC                          1725.451
                         - 12-01-1968
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2394      0.118      2.026      0.043       0.008       0.471
ar.L2          0.2633      0.117      2.241      0.025       0.033       0.494
ar.L3          0.0746      0.127      0.589      0.556      -0.174       0.323
ar.L4          0.1020      0.113      0.899      0.369      -0.120       0.324
ar.L5         -0.0853      0.118     -0.725      0.469      -0.316       0.145
ar.L6          0.1183      0.138      0.860      0.390      -0.151       0.388
ar.L7         -0.0117      0.133     -0.088      0.930      -0.272       0.248
sigma2      3.052e+06   5.71e+05      5.348      0.000    1.93e+06    4.17e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.12   Jarque-Bera (JB):                 1.45
Prob(Q):                              0.73   Prob(JB):                         0.48
Heteroskedasticity (H):               1.73   Skew:                             0.03
Prob(H) (two-sided):                  0.12   Kurtosis:                         2.40
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

注意：

我们原本有 9 年的原始数据，但是由于季节性差异处理，因此只有 8 年的实际数据可用。为了进行历史数据大小的敏感性分析，将最后一年的数据作为测试样本，依次选择1年、2年一直到7年的剩余数据为训练样本，步进地进行测试，并逐日记录测试情况。根据记录数据，我们还计算了均方根误差（RMSE）来明确反应模型的性能表现。

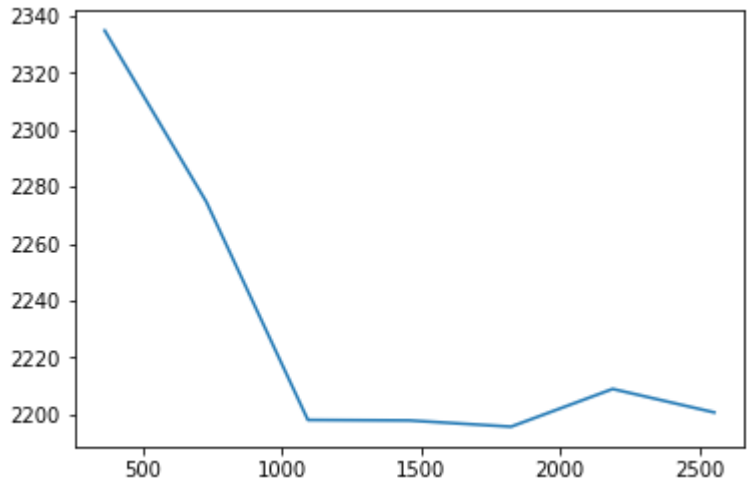
In [50]:
```
# 1. 计算 RMSE(均方根误差)
def RMSE(yArr,yHatArr): #yArr and yHatArr both need to be arrays
    import numpy as np
    return np.sqrt(((yArr-yHatArr)**2).sum() / len(yArr))
```

In [51]:
```
# 将经过季节性调整的数据分为训练数据和测试数据。
train, test = D_data[D_data.index < '1968'], D_data['1968']
years = ['1967', '1966', '1965', '1964', '1963', '1962', '1961']
```

In [55]:
```
#定好了数据之后，下一步是评估 ARIMA 模型。
#具体的步进评估方法是：首先选取一个时间段的数据，并根据选定数据建模，训练，然后对下一段数据进行预测
# 接着，将真实的观察数据加入建模数据，建立新的模型并展开训练，对再下一段数据进行预测，并记录结果。
#最终，预测结果将被集合在一起，与真实观察数据中的最后一年比较，计算出错误情况。在这种情况下，RMSE

from statsmodels.tsa.statespace.sarimax import SARIMAX

rmseList = list()
for year in years:
# select data from 'year' cumulative to 1989
    dataset = train[train.index >= year]
    # walk forward over time steps in test
    values = dataset.values
    history = [values[i] for i in range(len(values))]
    predictions = list()
    test_values = test.values
    for t in range(len(test_values)):
        # fit model
        model = SARIMAX(history, order=(1,0,0)).fit(trend='nc', disp=0)
        # make prediction
        yhat = model.forecast()
        predictions.append(yhat)
        history.append(test_values[t])
    rmse = RMSE(test_values, predictions)
    rmseList.append(rmse)
    print(('%s-%s (%d values) RMSE: %.3f' % (years[0], year, len(values), rmse)))
```

```
1967-1967 (12 values) RMSE: 2334.701
1967-1966 (24 values) RMSE: 2274.660
1967-1965 (36 values) RMSE: 2198.139
1967-1964 (48 values) RMSE: 2197.964
1967-1963 (60 values) RMSE: 2195.780
1967-1962 (72 values) RMSE: 2209.009
1967-1961 (84 values) RMSE: 2200.787
```

In [56]:
```
# 以下代码是根据测试数据绘制曲线图的过程。
from matplotlib import pyplot
x = [365, 730, 1095, 1460, 1825, 2190, 2555] # x轴是时间（天数）
pyplot.plot(x, rmseList)
pyplot.show()
```



结果

从曲线图可以更清晰地看到总体上历史数据越多，预测结果就更精确这一变化趋势。因为历史数据越多，就意味着系数的优化越精确，符合数据变化的内在规律的可能性就越高。

但同时也可以从上图中看到另一个现象：大多数时候人们觉得历史数据越多，模型的表现就越好。但实际上，连续三年以上的数据之间并没有什么根本性的差别，因此灵活选择时间跨度也至关重要。

总结

- 时间序列分析一般只适用于短期预测，准确性差。
- 时间序列数据的分析要检验纯随机性和平稳性检验，必要时进行差分，再选择合适的模型。
- 时间序列数据可以通过机器学习的工具（参照实例3）进行特征设计和选择，转换为适合于机器学习算法的输入数据。