时间序列分析（2） ARIMA 模型


随风
大数据、人工智能

关注他

416 人赞同了该文章

1 前言

从本章开始，将逐一对时间序列相关算法进行讨论。关于时间序列的基础知识，可以参见第一篇文章 时间序列分析（1） 基本概念与实战：

语言：python3

数据集：余额宝在2014-04-01~2014-08-10期间每日申购的总金额（数据来自天池大赛）

数据下载地址：tianchi.aliyun.com/comp

2 AR （Auto Regression）模型

自回归模型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。

一般的P阶自回归模型 AR：

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \ldots + \alpha_p X_{t-p} + u_t$$

如果随机扰动项是一个白噪声（ $u_t = \varepsilon_t$ ），则称为一个纯AR（p）过程，记为：

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \ldots + \alpha_p X_{t-p} + \varepsilon_t$$

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。

自回归模型有很多的限制：

（1）自回归模型是用自身的数据进行预测

（2）时间序列数据必须具有平稳性

（3）自回归只适用于预测与自身前期相关的现象（时间序列的自相关性）

3 MA （Moving Average）模型

在AR模型中，如果 u_t 不是一个白噪声，通常认为它是一个q阶的移动平均。即

$u_t = \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}$ ，其中 ε_t 表示白噪声序列。

特别的，当 $X_t = u_t$ ，即时间序列当前值与历史值没有关系，而只依赖于历史白噪声的线性组合，就得到MA模型：

$$X_t = \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}$$

需要指出一点，AR模型中历史白噪声的影响是间接影响当前预测值的（通过影响历史时序值）。

4 ARMA 模型

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）：

$$X_t = \alpha_1 X_{t-1} + \alpha_2 X_{t-2} + \ldots + \alpha_p X_{t-p} + \varepsilon_t + \beta_1 \varepsilon_{t-1} + \ldots + \beta_q \varepsilon_{t-q}$$

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。

5 ARIMA 模型

将自回归模型（AR）、移动平均模型（MA）和差分法结合，我们就得到了差分自回归移动平均模型 ARIMA（p、d、q），其中 d 是需要对数据进行差分的阶数。

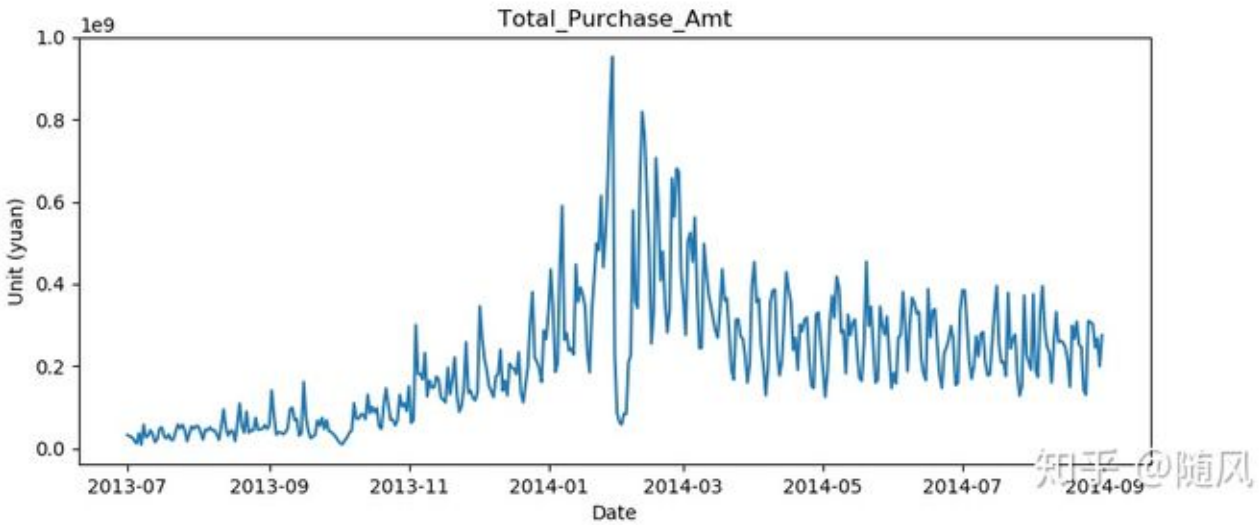
6 ARIMA 实战

生成 ARIMA 模型的基本步骤：

对序列绘图，进行 ADF 检验，观察序列是否平稳；对于非平稳时间序列要先进行 d 阶差分，转化为平稳时间序列；
经过第一步处理，已经得到平稳时间序列。要对平稳时间序列分别求得其自相关系数（ACF）和偏自相关系数（PACF），通过对自相关图和偏自相关图的分析，得到最佳的阶数p、q；
由以上得到的d、q、p，得到 ARIMA 模型。然后开始对得到的模型进行模型检验。

首先我们查看 user_balance_table.csv 文件，显示从2013-07-01到2014-08-31每日申购的总金额。

```
import matplotlib.pyplot as plt
import pandas as pd

user_balance = pd.read_csv('./user_balance_table.csv')

df_tmp = user_balance.groupby(['report_date'])['total_purchase_amt', 'total_redeem_amt
df_tmp.reset_index(inplace=True)

df_tmp['report_date'] = pd.to_datetime(df_tmp['report_date'], format='%Y%m%d')

df_tmp.index = df_tmp['report_date']

total_purchase_amt = plt.figure(figsize=(10, 4))
ax = total_purchase_amt.add_subplot(111)
ax.set(title='Total_Purchase_Amt',
       ylabel='Unit (yuan)', xlabel='Date')
plt.plot(df_tmp['report_date'], df_tmp['total_purchase_amt'])
plt.show()
```



从上图中可以看出，从2013-07~2014-04，余额宝每日的申购金额经历了一个较大的波动过程，从2014-04开始，趋向于稳定的震荡（均值近似为常数）。**ARIMA 模型是通过寻找历史数据之间的自相关性，来预测未来（假设未来将重复历史的走势），要求序列必须是平稳的。**因此我们取2014-04-01~2014-07-31的数据作为训练集，将2014-08-01~2014-08-10的数据作为测试集。

```
import pandas as pd


def generate_purchase_seq():
    dateparse = lambda dates: pd.datetime.strptime(dates, '%Y%m%d')
    user_balance = pd.read_csv('./user_balance_table.csv', parse_dates=['report_date']
```

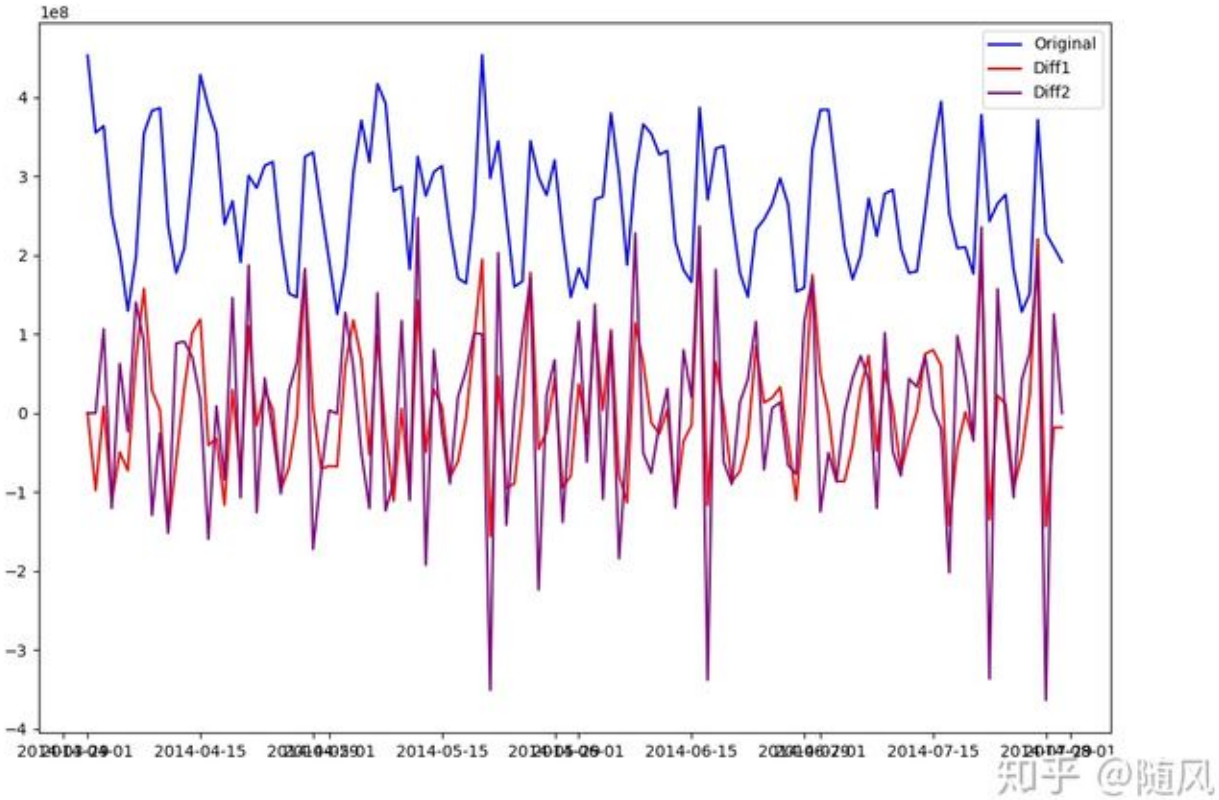
```
                                  index_col='report_date', date_parser=dateparse)

    df = user_balance.groupby(['report_date'])['total_purchase_amt'].sum()
    purchase_seq = pd.Series(df, name='value')

    purchase_seq_train = purchase_seq['2014-04-01':'2014-07-31']
    purchase_seq_test = purchase_seq['2014-08-01':'2014-08-10']

    purchase_seq_train.to_csv(path='./purchase_seq_train.csv', header=True)
    purchase_seq_test.to_csv(path='./purchase_seq_test.csv', header=True)


generate_purchase_seq()
```

查看一下训练集 purchase_seq_train.csv 的差分效果，并对每一次差分结果做 ADF 检验：

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller as ADF


def diff(timeseries):
    timeseries_diff1 = timeseries.diff(1)
    timeseries_diff2 = timeseries_diff1.diff(1)

    timeseries_diff1 = timeseries_diff1.fillna(0)
    timeseries_diff2 = timeseries_diff2.fillna(0)

    timeseries_adf = ADF(timeseries['value'].tolist())
    timeseries_diff1_adf = ADF(timeseries_diff1['value'].tolist())
    timeseries_diff2_adf = ADF(timeseries_diff2['value'].tolist())

    print('timeseries_adf : ', timeseries_adf)
    print('timeseries_diff1_adf : ', timeseries_diff1_adf)
    print('timeseries_diff2_adf : ', timeseries_diff2_adf)

    plt.figure(figsize=(12, 8))
    plt.plot(timeseries, label='Original', color='blue')
    plt.plot(timeseries_diff1, label='Diff1', color='red')
    plt.plot(timeseries_diff2, label='Diff2', color='purple')
```

```
    plt.legend(loc='best')
    plt.show()
```

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

diff(purchase_seq_train)
```

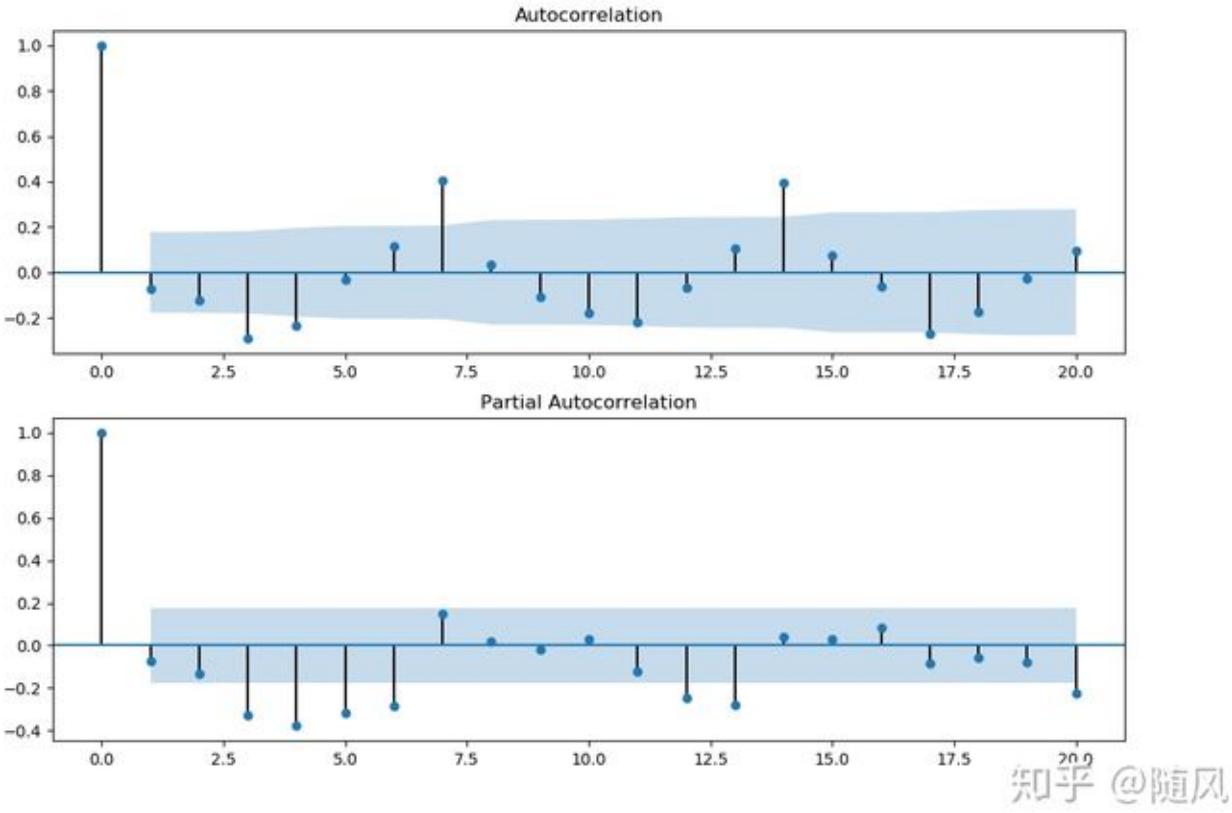



```
timeseries_adf :  (-2.0639747511769864, 0.25924499643351684, 13, 108, {'1%': -3.492401
timeseries_diff1_adf :  (-6.542516143607563, 9.270661450976566e-09, 12, 109, {'1%': -3
timeseries_diff2_adf :  (-5.615545867454514, 1.1766955956627649e-06, 13, 108, {'1%': -
```

从结果来看，要想使得序列变得平稳，需要进行一阶差分。我们对序列进行一阶差分，并查看差分后序列的 ACF、PACF：

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
```

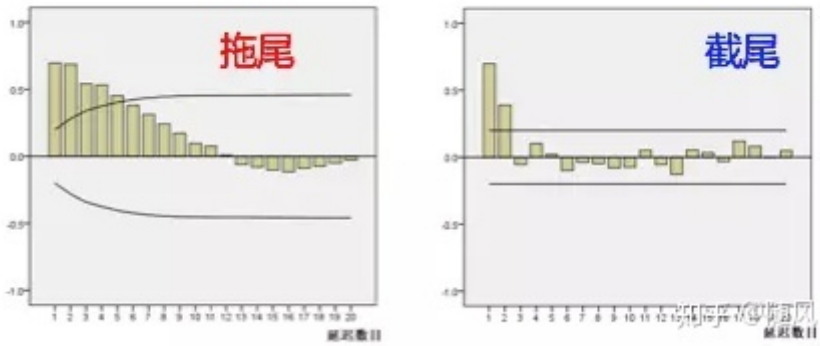
```
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller as ADF


def autocorrelation(timeseries, lags):
    fig = plt.figure(figsize=(12, 8))
    ax1 = fig.add_subplot(211)
    sm.graphics.tsa.plot_acf(timeseries, lags=lags, ax=ax1)
    ax2 = fig.add_subplot(212)
    sm.graphics.tsa.plot_pacf(timeseries, lags=lags, ax=ax2)
    plt.show()


dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

purchase_seq_train_diff = purchase_seq_train.diff(1)
purchase_seq_train_diff = purchase_seq_train_diff.fillna(0)

autocorrelation(purchase_seq_train_diff, 20)
```



如何确定 AR（p）的 p 值，MA（q）的 q 值?

拖尾和截尾

拖尾指序列以指数率单调递减或震荡衰减，而截尾指序列从某个时点变得非常小：



p，q阶数的确定

模型（序列）	AR(p)	MA(q)	ARMA(p, q)
自相关函数	拖尾	第q个后截尾	拖尾
偏自相关函数	第p个后截尾	拖尾	拖尾

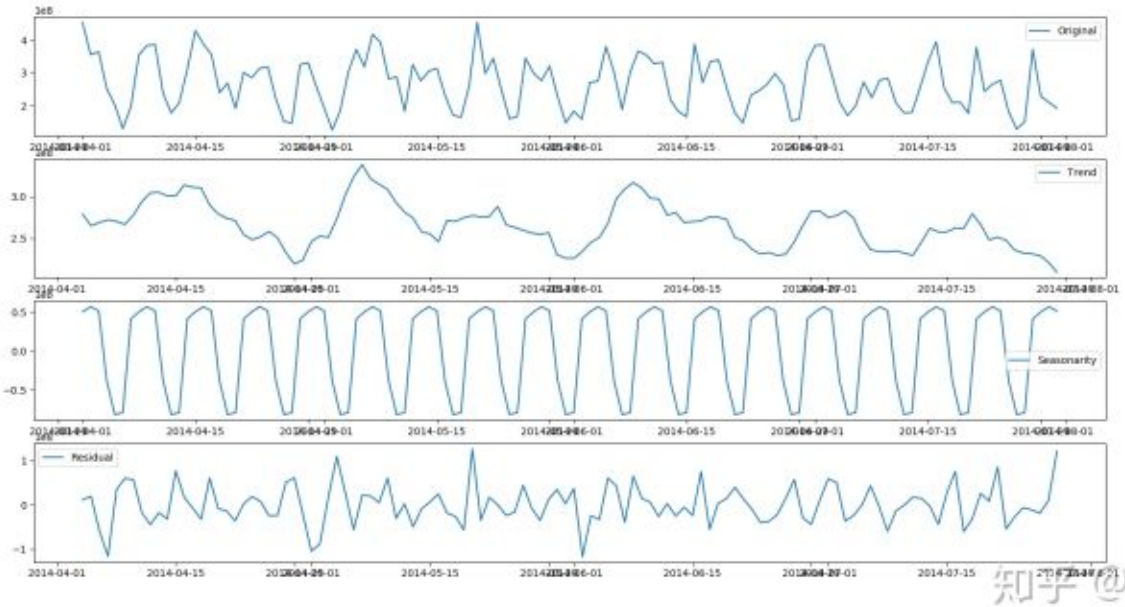
从序列 ACF 、PACF 的图中没有发现明显的拖尾或截尾，说明对于这样的序列，并不适合用 ARIMA 模型来拟合。如何用 ARIMA 模型来拟合这样的曲线呢？**这里我们将先通过对时间序列分解（STL），再采用 ARIMA 模型来拟合趋势序列与残差序列。**

对原序列分解：

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller as ADF


def decomposing(timeseries):
    decomposition = seasonal_decompose(timeseries)
    trend = decomposition.trend
    seasonal = decomposition.seasonal
    residual = decomposition.resid

    plt.figure(figsize=(16, 12))
    plt.subplot(411)
    plt.plot(timeseries, label='Original')
    plt.legend(loc='best')
```

```
    plt.subplot(412)
    plt.plot(trend, label='Trend')
    plt.legend(loc='best')
    plt.subplot(413)
    plt.plot(seasonal, label='Seasonarity')
    plt.legend(loc='best')
    plt.subplot(414)
    plt.plot(residual, label='Residual')
    plt.legend(loc='best')
    plt.show()
```

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

decomposing(purchase_seq_train)
```



从上图中可以看到，原序列有明显的周期性，而且是以七天为一个周期（可以将具体的数值打印出来）。因此我们只对趋势序列和残差序列去拟合，同时认为这样的周期性会延伸至2014-08-01~2014-08-10的测试集上。下面将介绍对趋势序列、残差序列的拟合过程。

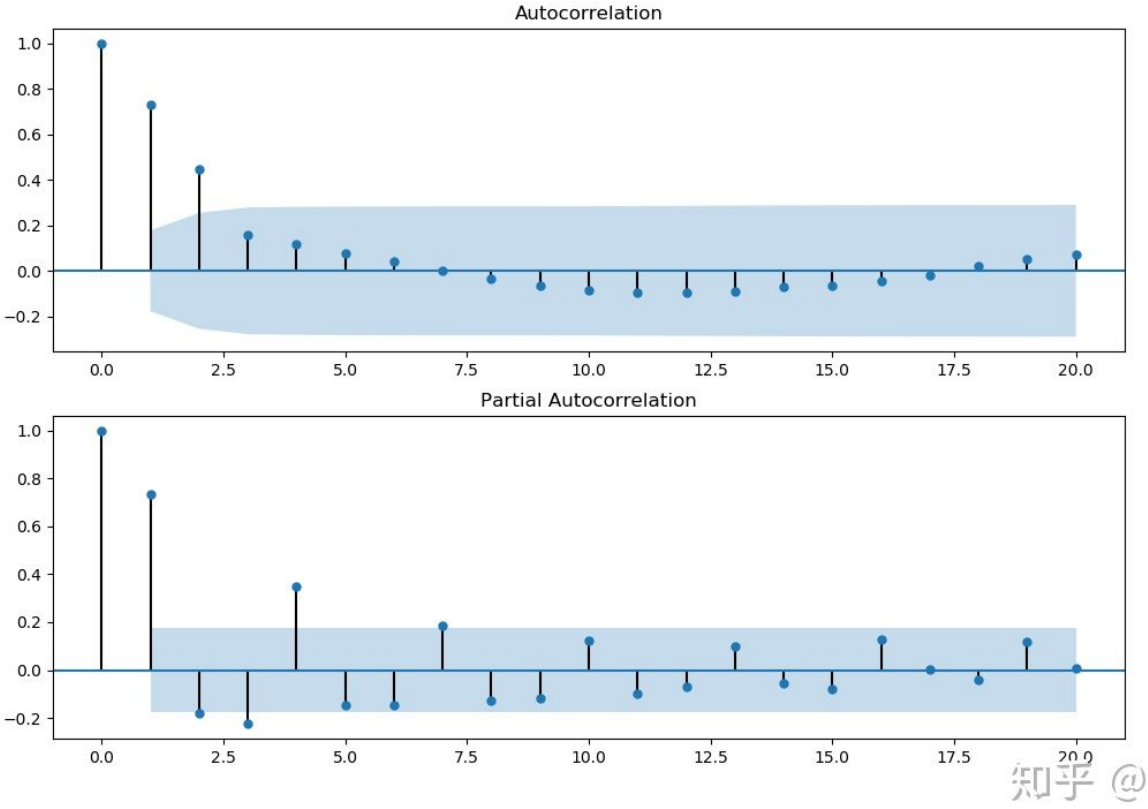
先观察趋势序列、残差序列的差分效果，并对每一次差分结果做 ADF 检验，函数代码请参考前面的内容：

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
```

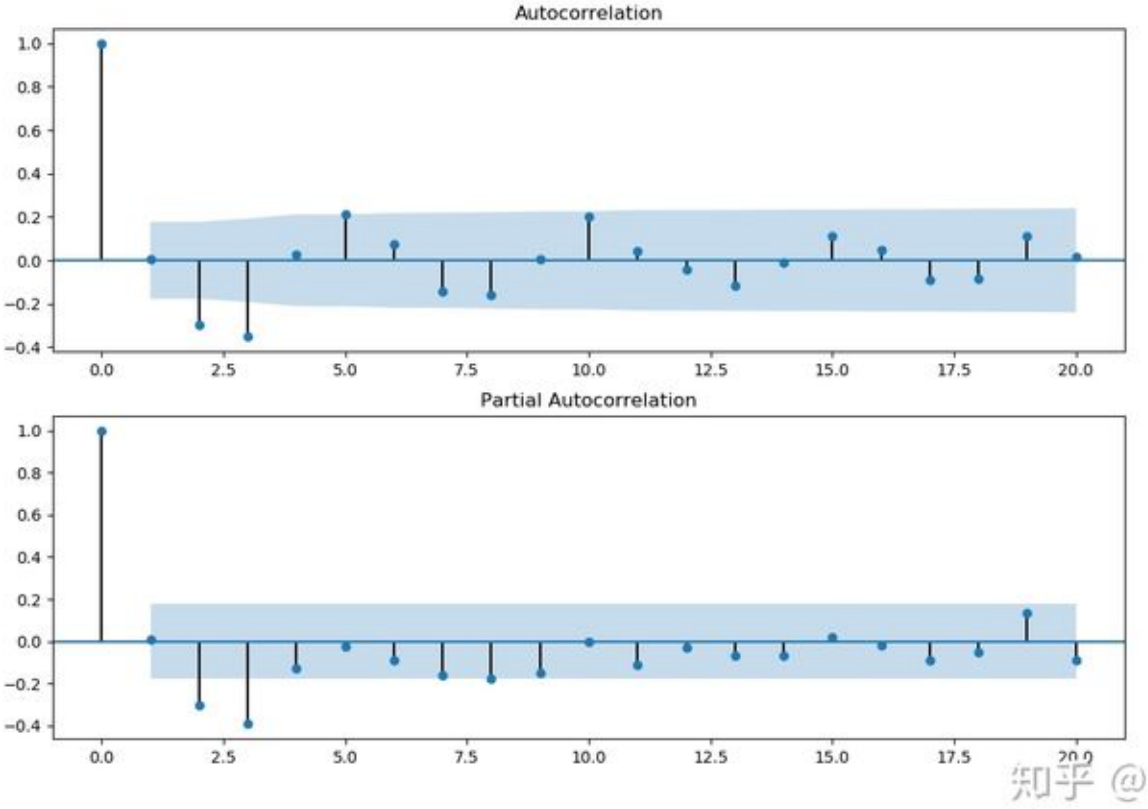
```
                                index_col='report_date', date_parser=dateparse)

decomposition = seasonal_decompose(purchase_seq_train)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

trend = trend.fillna(0)
seasonal = seasonal.fillna(0)
residual = residual.fillna(0)

diff(trend)
timeseries_adf :  (-3.2368487584485877, 0.017948383665881536, 0, 121, {'10%': -2.57967
timeseries_diff1_adf :  (-10.571816201699766, 7.2723798146224e-19, 0, 121, {'10%': -2.
timeseries_diff2_adf :  (-5.522044427904824, 1.868165478739749e-06, 8, 113, {'10%': -2

diff(residual)
timeseries_adf :  (-6.290212104648347, 3.614727756796406e-08, 8, 113, {'10%': -2.58060
timeseries_diff1_adf :  (-5.903150268380962, 2.7477376300421066e-07, 13, 108, {'10%':
timeseries_diff2_adf :  (-6.6447268786449385, 5.3007963991287325e-09, 13, 108, {'10%':
```

从 ADF 的检验结果来看，趋势序列和残差序列都已经比较平稳了，因此不需要进行差分。再看一下 ACF、PACF：

```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                index_col='report_date', date_parser=dateparse)

decomposition = seasonal_decompose(purchase_seq_train)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

trend = trend.fillna(0)
seasonal = seasonal.fillna(0)
residual = residual.fillna(0)

autocorrelation(trend, 20)

autocorrelation(residual, 20)
```



趋势序列 ACF、PACF



残差序列 ACF、PACF

从上图可以看到：

（1）趋势序列 ACF 有 3 阶截尾，PACF 有 2 阶拖尾。因此可以选 p=2， q=3。

（2）残差序列 ACF 有 4 阶拖尾，PACF 有 4 阶截尾。因此可以选 p=4， q=4。

通过拖尾和截尾对模型定阶，具有很强的主观性。回顾一下我们对于模型参数估计得方法，是通过对损失和正则项的加权评估。我们在参数选择的时候，需要平衡预测误差与模型复杂度。我们可以根据信息准则函数法，来确定模型的阶数。这里介绍 AIC、BIC准则。

AIC 准则全称是最小化信息量准则（Akaike Information Criterion）：

$AIC = -2ln(L) + 2K$ ，其中 L 表示模型的极大似然函数， K 表示模型参数个数。

AIC 准则存在一定的不足。当样本容量很大时，在 AIC 准则中拟合误差提供的信息就要受到样本容量的放大，而参数个数的惩罚因子却和样本容量没关系（一直是2），因此当样本容量很大时，使用 AIC 准则的模型不收敛于真实模型，它通常比真实模型所含的未知参数个数要多。BIC（Bayesian InformationCriterion）贝叶斯信息准则弥补了 AIC 的不足：

$BIC = -2ln(L) + Kln(n)$ ，其中 n 表示样本容量。

显然，这两个评价指标越小越好。我们通过网格搜索，确定 AIC、BIC 最优的模型（p、q）。

```
import statsmodels.api as sm


dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

decomposition = seasonal_decompose(purchase_seq_train)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

trend = trend.fillna(0)
seasonal = seasonal.fillna(0)
residual = residual.fillna(0)

trend_evaluate = sm.tsa.arma_order_select_ic(trend, ic=['aic', 'bic'], trend='nc', max
                                             max_ma=4)
print('trend AIC', trend_evaluate.aic_min_order)
print('trend BIC', trend_evaluate.bic_min_order)
```

```
residual_evaluate = sm.tsa.arma_order_select_ic(residual, ic=['aic', 'bic'], trend='nc
                                                max_ma=4)
print('residual AIC', residual_evaluate.aic_min_order)
print('residual BIC', residual_evaluate.bic_min_order)
```

```
trend AIC (1, 0)
trend BIC (1, 0)

residual AIC (2, 1)
residual BIC (2, 1)
```

从评价准则的结果看（这里采用 AIC 结果）：

（1）对趋势序列，p = 1， q = 0

（2）对残差序列，p = 2， q = 1

下面我们将分别训练趋势序列和残差序列的 ARIMA 模型，并结合原序列的周期，拟合训练集数据，并预测测试集数据。

对于训练集，拟合序列 = 周期序列 + 趋势序列（ARIMA拟合） + 残差序列（ARIMA拟合）

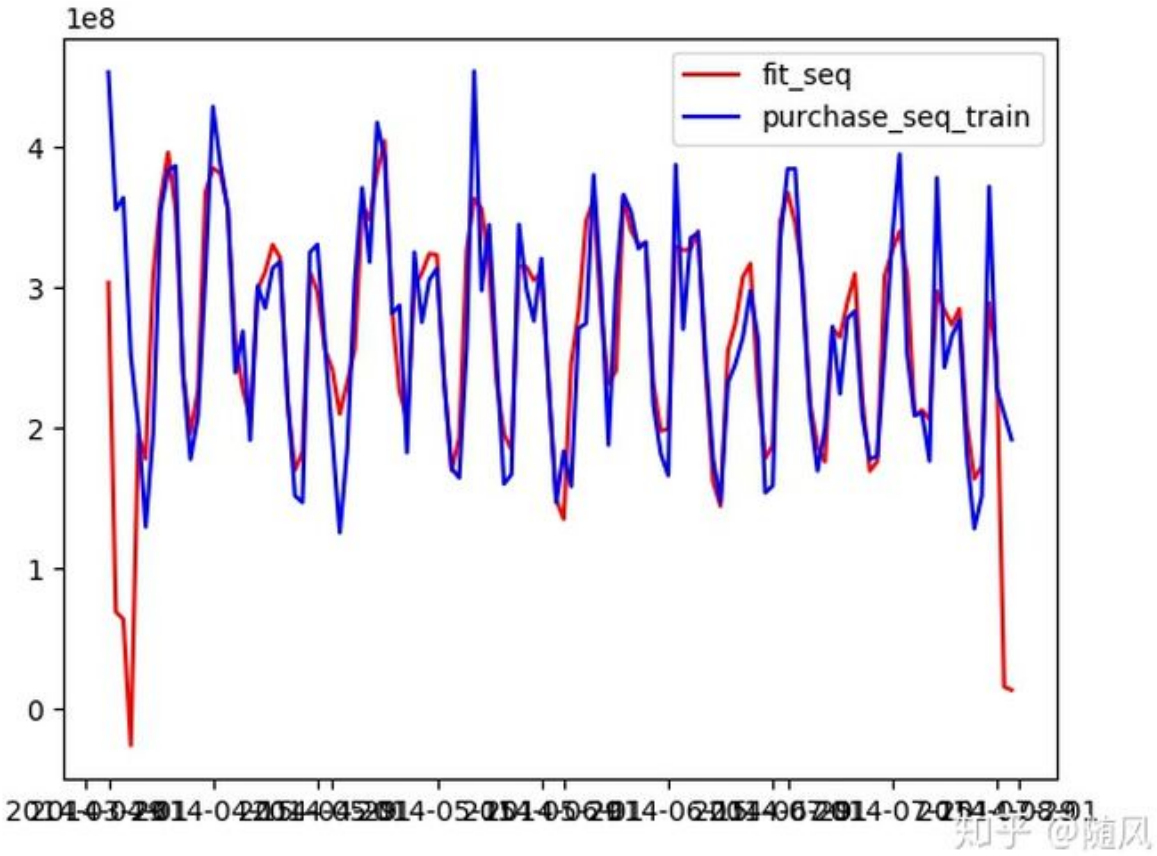
对于测试集，预测序列 = 周期序列 + 趋势序列（ARIMA预测） + 残差序列（ARIMA预测）

```
import matplotlib.pyplot as plt
import pandas as pd
import statsmodels.api as sm
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller as ADF

def ARIMA_Model(timeseries, order):
    model = ARIMA(timeseries, order=order)
    return model.fit(disp=0)
```

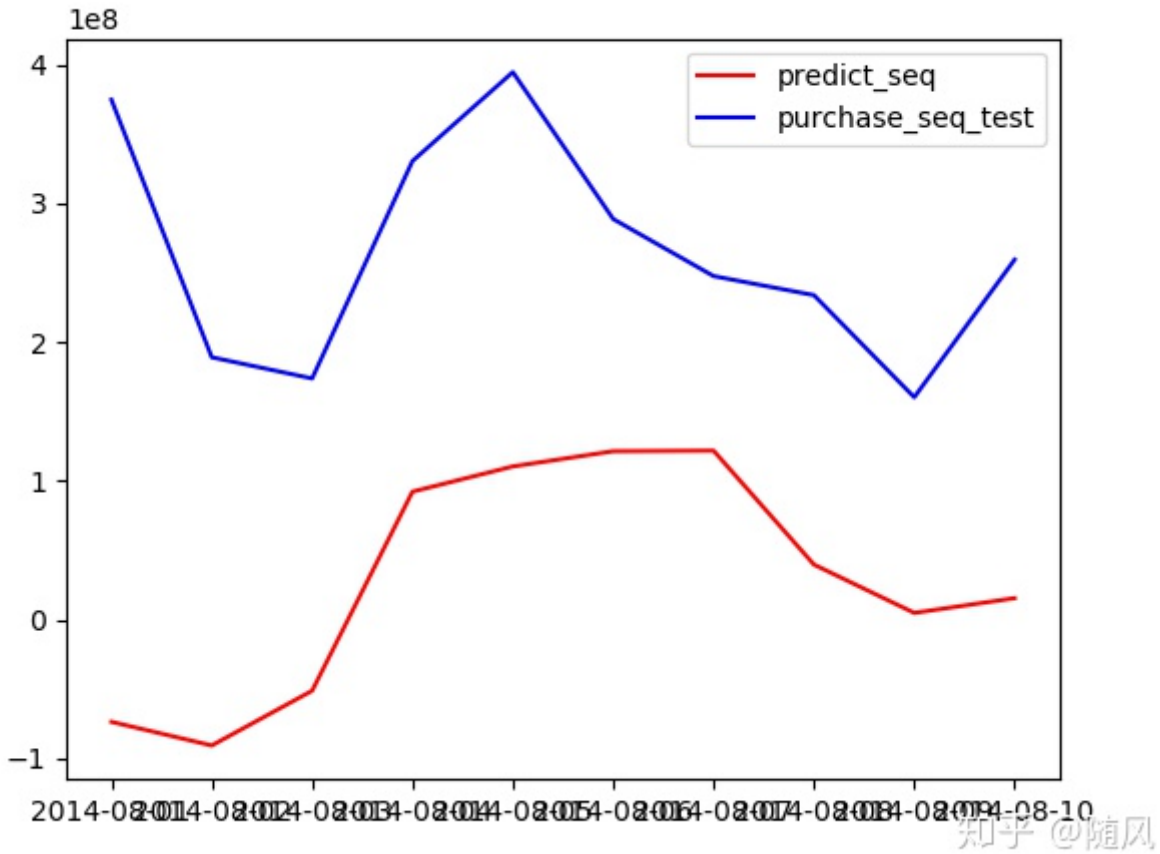
```
dateparse = lambda dates: pd.datetime.strptime(dates, '%Y-%m-%d')
purchase_seq_train = pd.read_csv('./purchase_seq_train.csv', parse_dates=['report_date
                                 index_col='report_date', date_parser=dateparse)

purchase_seq_test = pd.read_csv('./purchase_seq_test.csv', parse_dates=['report_date']
```

```
                                    index_col='report_date', date_parser=dateparse)


decomposition = seasonal_decompose(purchase_seq_train)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

trend = trend.fillna(0)
seasonal = seasonal.fillna(0)
residual = residual.fillna(0)

# 趋势序列模型训练
trend_model = ARIMA_Model(trend, (1, 0, 0))
trend_fit_seq = trend_model.fittedvalues
trend_predict_seq = trend_model.predict(start='2014-08-01', end='2014-08-10', dynamic=

# 残差序列模型训练
residual_model = ARIMA_Model(residual, (2, 0, 1))
residual_fit_seq = residual_model.fittedvalues
residual_predict_seq = residual_model.predict(start='2014-08-01', end='2014-08-10', dy

# 拟合训练集
fit_seq = pd.Series(seasonal['value'], index=seasonal.index)
fit_seq = fit_seq.add(trend_fit_seq, fill_value=0)
fit_seq = fit_seq.add(residual_fit_seq, fill_value=0)

plt.plot(fit_seq, color='red', label='fit_seq')
plt.plot(purchase_seq_train, color='blue', label='purchase_seq_train')
plt.legend(loc='best')
plt.show()

# 预测测试集
# 这里测试数据的周期性是根据seasonal对象打印的结果，看到里面的数据每7天一个周期，2014-08-01~20
seasonal_predict_seq = seasonal['2014-04-04':'2014-04-13']

predict_dates = pd.Series(
    ['2014-08-01', '2014-08-02', '2014-08-03', '2014-08-04', '2014-08-05', '2014-08-06
     '2014-08-09', '2014-08-10']).apply(lambda dates: pd.datetime.strptime(dates, '%Y-

seasonal_predict_seq.index = predict_dates

predict_seq = pd.Series(seasonal_predict_seq['value'], index=seasonal_predict_seq.inde
predict_seq = predict_seq.add(trend_predict_seq, fill_value=0)
predict_seq = predict_seq.add(residual_predict_seq, fill_value=0)
```

```
plt.plot(predict_seq, color='red', label='predict_seq')
plt.plot(purchase_seq_test, color='blue', label='purchase_seq_test')
plt.legend(loc='best')
plt.show()
```



模型拟合训练集结果



模型预测测试集结果

7 结语

从结果来看，模型拟合训练集的效果还是不错的；在测试集上，模型基本上预测了序列的趋势和波动。实际上，这样的数据集不适合用 ARIMA 模型来拟合（序列的（线性）自相关性不强，受随机噪声影响较大），但是这里我们采用了时间序列分解的方法，暂且预测了一个序列的趋势。