**Report 7**

**Part 1: Course exercises**

1.1: 时间序列

1.2: 时间戳

1.3: 基础频率

1.4: 重采样

1.5: Period

2.1: T

2.2: T

2.3: F

2.4: T

2.5: F

3.1: A

3.2: D

3.3: D

3.4 B

3.5: C

4.1: 时间戳表示带时区的特定的日期时间；时间差表示绝对的持续时间；时期是由时间点及其相关频率定义的时间跨度。

4.2: 如果是将高频率数据聚合到低频率，比如将每日采集的频率变成每月采集，则称为降采样；如果将低频率数据转换到高频率数据，比如将每月采集的频率变成每日采集，则称为升采样。

5:

（1）、运行产生ValueError异常，主要是因为date\_range()函数中必须传入start、end、periods、freq中至少三个参数，而代码中只传入了start和freq参数。

（2）、运行结果为5

（3）、运行出现异常，因为日期字符串的格式不能够被正确解析。

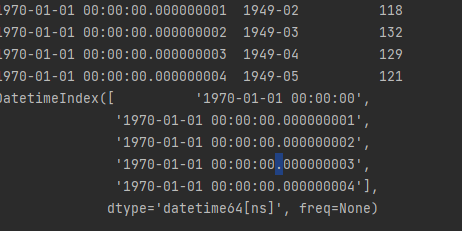
**Part 2:**

**Re-implement the codes from the stock prediction and analysis Demo in the text book page 223-228, summarize the results and screenshot the codes in the report.**

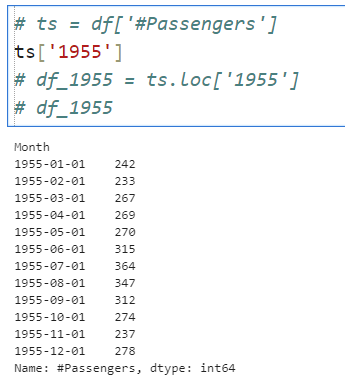
**Part 3:**

**从Moodle下载‘Report7-AirPassengers.csv’文件并完成以下操作.**

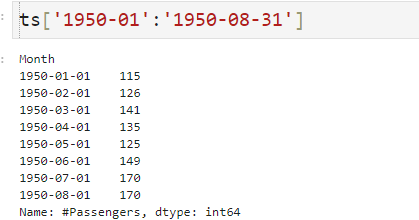
1. 读取并显示数据；

import numpy as np  
import pandas as pd  
from datetime import datetime  
df = pd.read\_csv('./Report7-AirPassengers.csv')  
df.index = pd.to\_datetime(df.index)  
ts = df  
print(ts.head())  
print(ts.head().index) 

1. 查看1955年数据；



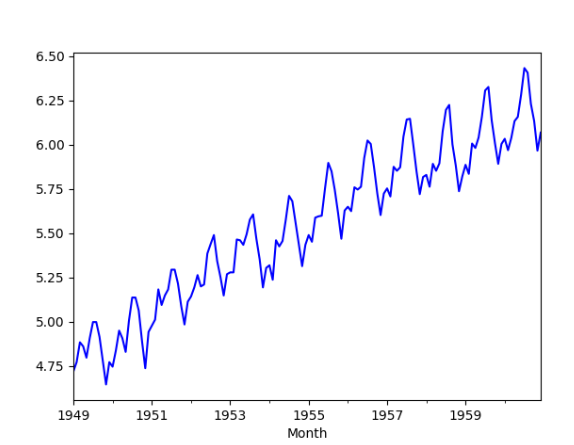
1. 对1950年1月到8月数据进行切片；



1. 利用观察法进行数据平稳性检测；

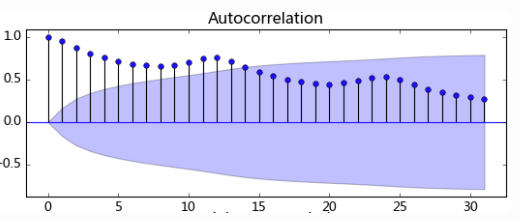
test\_stationary.py

from statsmodels.tsa.stattools import adfuller  
import pandas as pd  
import matplotlib.pyplot as plt  
import numpy as np  
from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  
  
# 移动平均图  
def draw\_trend(timeSeries, size):  
 f = plt.figure(facecolor='white')  
 # 对size个数据进行移动平均  
 rol\_mean = timeSeries.rolling(window=size).mean()  
 # 对size个数据进行加权移动平均  
 rol\_weighted\_mean = pd.ewma(timeSeries, span=size)  
  
 timeSeries.plot(color='blue', label='Original')  
 rolmean.plot(color='red', label='Rolling Mean')  
 rol\_weighted\_mean.plot(color='black', label='Weighted Rolling Mean')  
 plt.legend(loc='best')  
 plt.title('Rolling Mean')  
 plt.show()  
  
def draw\_ts(timeSeries):  
 f = plt.figure(facecolor='white')  
 timeSeries.plot(color='blue')  
 plt.show()  
def testStationarity(ts):  
 dftest = adfuller(ts)  
 # 对上述函数求得的值进行语义描述  
 dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])  
 for key,value in dftest[4].items():  
 dfoutput['Critical Value (%s)'%key] = value  
 return dfoutput  
  
# 自相关和偏相关图，默认阶数为31阶  
def draw\_acf\_pacf(ts, lags=31):  
 f = plt.figure(facecolor='white')  
 ax1 = f.add\_subplot(211)  
 plot\_acf(ts, lags=31, ax=ax1)  
 ax2 = f.add\_subplot(212)  
 plot\_pacf(ts, lags=31, ax=ax2)  
 plt.show()



1. 利用单位根检验方法对数据进行平稳性检测；

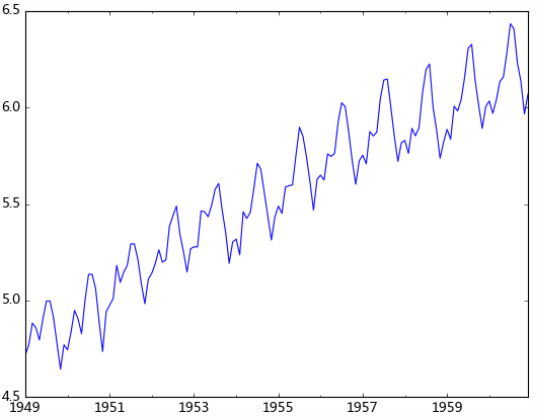
代码同上



1. 对数据进行基于对数变换的平稳性处理；

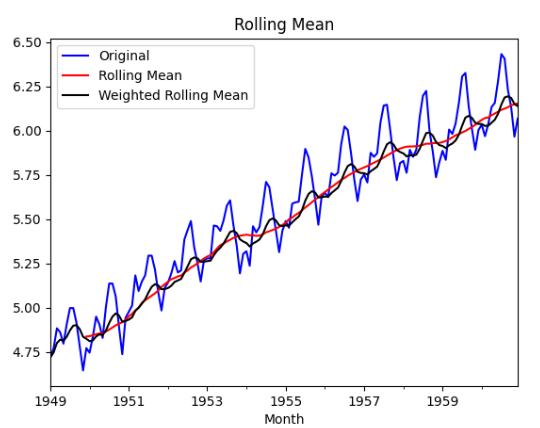
main.py

import test\_stationarity  
import numpy as np  
import pandas as pd  
  
df = pd.read\_csv(r'D:\schoolzwu\AI\machine-study\data\_analyse\3-30\Report7-AirPassengers.csv', encoding='utf-8', index\_col='Month')  
df.index = pd.to\_datetime(df.index)  
ts = df['#Passengers'] # 生成pd.Series对象  
  
ts\_log = np.log(ts)  
test\_stationarity.draw\_ts(ts\_log)



1. 对数据进行基于平滑法的平稳性处理；

test\_stationarity.draw\_trend(ts\_log, 12)



1. 对数据进行基于差分的平稳性处理；

diff\_12 = ts\_log.diff(12)

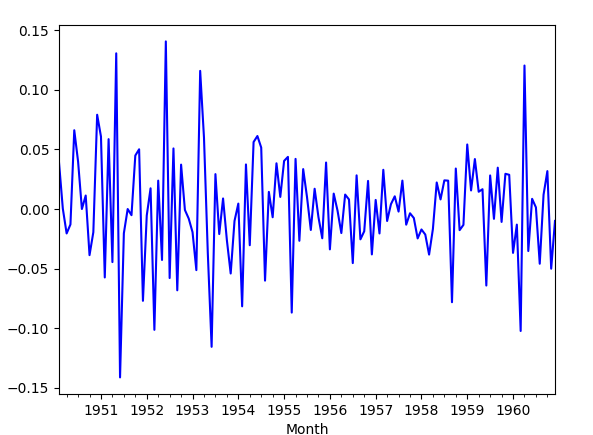
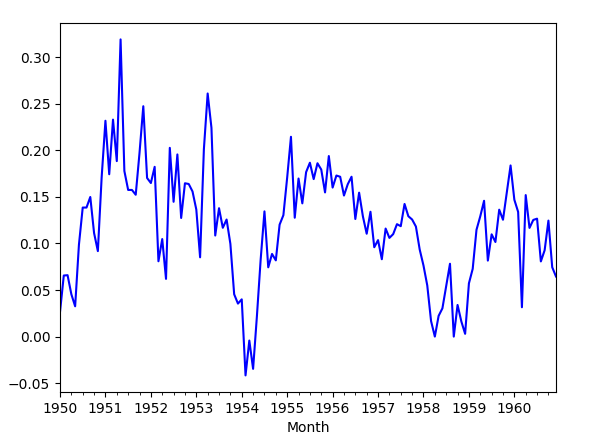
diff\_12.dropna(*inplace*=True)

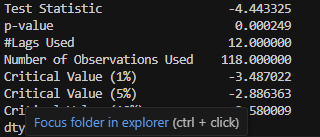
diff\_12\_1 = diff\_12.diff(1)

diff\_12\_1.dropna(*inplace*=True)

test\_stationarity.draw\_ts(diff\_12)

print(test\_stationarity.testStationarity(diff\_12\_1))





1. 对数据进行基于分解的平稳性处理；

*import* test\_stationarity

*import* numpy *as* np

*import* pandas *as* pd

*from* statsmodels.tsa.arima\_model *import* ARMA

df = pd.read\_csv(r'D:\schoolzwu\AI\machine-study\data\_analyse\3-30\Report7-AirPassengers.csv', *encoding*='utf-8', *index\_col*='Month')

df.index = pd.to\_datetime(df.index)

ts = df['#Passengers']  *# 生成pd.Series对象*

ts\_log = np.log(ts)

*# 差分*

*# diff\_12 = ts\_log.diff(12)*

*# diff\_12.dropna(inplace=True)*

*# diff\_12\_1 = diff\_12.diff(1)*

*# diff\_12\_1.dropna(inplace=True)*

*# test\_stationarity.draw\_ts(diff\_12)*

*# print(test\_stationarity.testStationarity(diff\_12\_1))*

*# 分解*

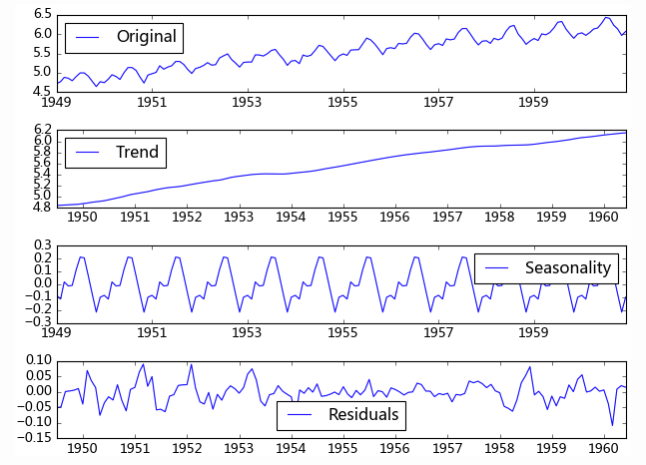
*from* statsmodels.tsa.seasonal *import* seasonal\_decompose

decomposition = seasonal\_decompose(ts\_log, *model*="additive")

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid



1. 对数据进行基于1阶差分的模型识别；

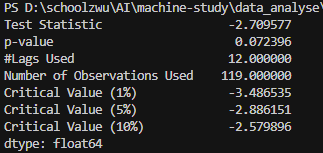
rol\_mean = ts\_log.rolling(*window*=12).mean()

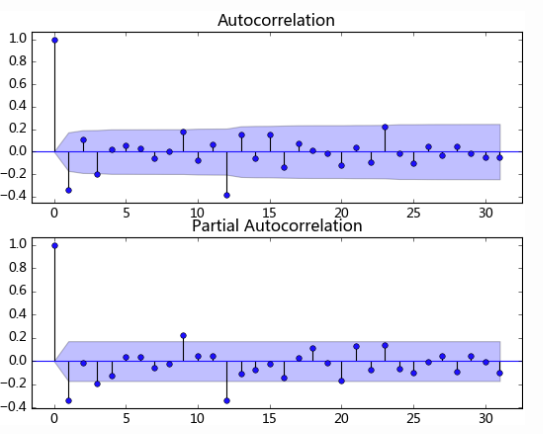
rol\_mean.dropna(*inplace*=True)

ts\_diff\_1 = rol\_mean.diff(1)

ts\_diff\_1.dropna(*inplace*=True)

print(test\_stationarity.testStationarity(ts\_diff\_1))





1. 对数据进行逆变换还原后进行基于均方根误差的样本拟合；

ts = ts[log\_recover.index]  *# 过滤没有预测的记录*

plt.figure(*facecolor*='white')

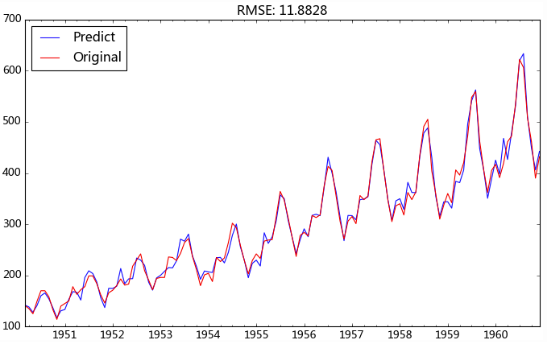
log\_recover.plot(*color*='blue', *label*='Predict')

ts.plot(*color*='red', *label*='Original')

plt.legend(*loc*='best')

plt.title('RMSE: %.4f'% np.sqrt(sum((log\_recover-ts)\*\*2)/ts.size))

plt.show()



1. 分离差分，完善ARIMA模型；

model = ARIMA(ts\_diff\_2, *order*=(1, 1, 0))

*# order=(p,d,q) 其中 p=1 是 AR 阶数，d=1 是差分阶数，q=1 是 MA 阶数*

result\_arma = model.fit()

predict\_ts = result\_arma.predict()

*# 一阶差分还原*

diff\_shift\_ts = ts\_diff\_1.shift(1)

diff\_recover\_1 = predict\_ts.add(diff\_shift\_ts)

*# 再次一阶差分还原*

rol\_shift\_ts = rol\_mean.shift(1)

diff\_recover = diff\_recover\_1.add(rol\_shift\_ts)

*# 移动平均还原*

rol\_sum = ts\_log.rolling(*window*=11).sum()

rol\_recover = diff\_recover\*12 - rol\_sum.shift(1)

*# 对数还原*

log\_recover = np.exp(rol\_recover)

log\_recover.dropna(*inplace*=True)

ts = ts[log\_recover.index]  *# 过滤没有预测的记录*

plt.figure(*facecolor*='white')

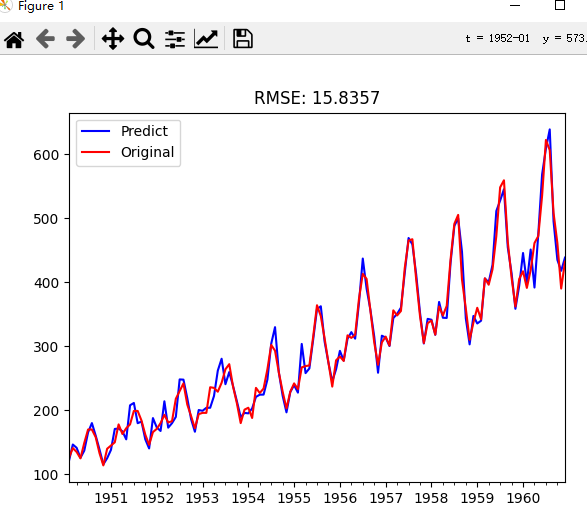
log\_recover.plot(*color*='blue', *label*='Predict')

ts.plot(*color*='red', *label*='Original')

plt.legend(*loc*='best')

plt.title('RMSE: %.4f'% np.sqrt(sum((log\_recover-ts)\*\*2)/ts.size))

plt.show()



1. 以周为单位对数据进行滚动预测；

*from* dateutil.relativedelta *import* relativedelta

def **\_add\_new\_data**(*ts*, *dat*, *type*='day'):

*if* *type* == 'day':

        new\_index = *ts*.index[-1] + relativedelta(*days*=1)

*elif* *type* == 'month':

        new\_index = *ts*.index[-1] + relativedelta(*months*=1)

*ts*[new\_index] = *dat*

def **add\_today\_data**(*model*, *ts*,  *data*, *d*, *type*='day'):

    \_add\_new\_data(*ts*, *data*, *type*)  *# 为原始序列添加数据*

*# 为滞后序列添加新值*

    d\_ts = diff\_ts(*ts*, *d*)

*model*.add\_today\_data(d\_ts[-1], *type*)

def **forecast\_next\_day\_data**(*model*, *type*='day'):

*if* *model* == None:

*raise* ValueError('No model fit before')

    fc = *model*.forecast\_next\_day\_value(*type*)

*return* predict\_diff\_recover(fc, [12, 1])

ts\_train = ts\_log[:'1956-12']

ts\_test = ts\_log['1957-1':]

diffed\_ts = diff\_ts(ts\_train, [12, 1])

forecast\_list = []

*for* i, dta *in* enumerate(ts\_test):

*if* i%7 == 0:

        model = arima\_model(diffed\_ts)

        model.certain\_model(1, 1)

    forecast\_data = forecast\_next\_day\_data(model, *type*='month')

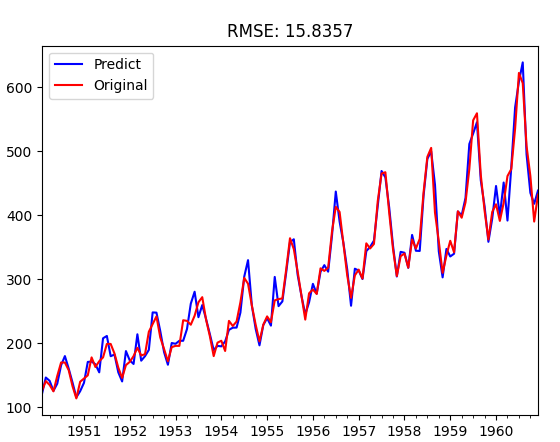
    forecast\_list.append(forecast\_data)

    add\_today\_data(model, ts\_train, dta, [12, 1], *type*='month')

predict\_ts = pd.Series(*data*=forecast\_list, *index*=ts['1957-1':].index)

log\_recover = np.exp(predict\_ts)

original\_ts = ts['1957-1':]



Ref：<https://www.cnblogs.com/foley/p/5582358.html>（任务实现请参考该网址）