**VAR模型的引入和Python实践**

向量自回归 (VAR) 是一种随机过程模型，用于捕获多个时间序列之间的线性相互依赖性。 VAR 模型通过允许多个进化变量来概括单变量自回归模型（AR 模型）。

VAR 中的所有变量都以相同的方式进入模型：每个变量都有一个方程式，根据其自身的滞后值、其他模型变量的滞后值和一个误差项来解释其演变。

**第一章：简介**

首先，什么是向量自回归（VAR）以及何时使用它？

向量自回归 （VAR） 是一种多变量预测算法，当两个或多个时间序列相互影响时使用。

这意味着，使用 VAR 的基本要求是：

1. 至少需要两个时间序列（变量）；
2. 时间序列应相互影响。

好。那么为什么它被称为“自回归”呢？

它被认为是自回归模型，因为每个变量（时间序列）都建模为过去值的函数，即预测变量只不过是序列的滞后（时间延迟值）。

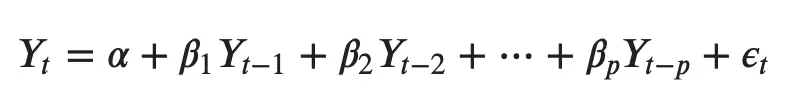
好的，那么VAR与其他自回归模型（如AR，ARMA或ARIMA）有何不同？

主要区别在于这些模型是单向的，其中预测变量影响 Y，反之亦然。而矢量自动回归 （VAR） 是双向的。也就是说，变量相互影响。

**第二章：VAR模型公式背后的直觉**

在自回归模型中，时间序列被建模为其自身滞后的线性组合。也就是说，序列的过去值用于预测当前和未来。

典型的 AR（p） 模型方程如下所示：



其中α是截距，一个常数，β1，β2 直到 βp 是 Y 到 p 阶的滞后的系数。阶“p”表示，最多使用 Y 的 p 滞后，它们是方程中的预测因子。ε\_{t}是错误，被视为白噪声。

好。那么，VAR模型的公式是什么样的呢？

在VAR模型中，每个变量都被建模为自身过去值和系统中其他变量过去值的线性组合。由于我们有多个相互影响的时间序列，因此将其建模为方程组，每个变量（时间序列）一个方程。

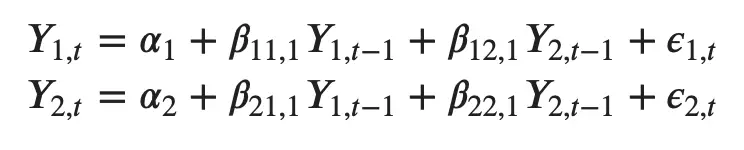
也就是说，如果有5个相互影响的时间序列，我们将拥有一个由5个方程组成的系统。

那么，方程是如何精确构建的？

假设我们有两个变量（时间序列）Y1 和 Y2，需要在时间（t）预测这些变量的值。

为了计算Y1（t），VAR将使用Y1和Y2的过去值。同样，要计算 Y2（t），请使用 Y1 和 Y2 的过去值。

例如，具有两个时间序列（变量“Y1”和“Y1”）的 VAR（2） 模型的方程组如下：

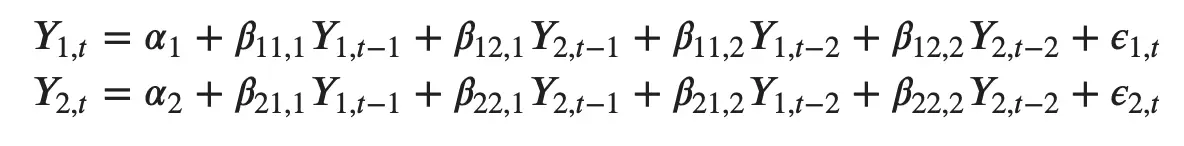


其中，Y{1，t-1} 和 Y{2，t-1} 分别是时间序列 Y1 和 Y2 的第一个滞后。

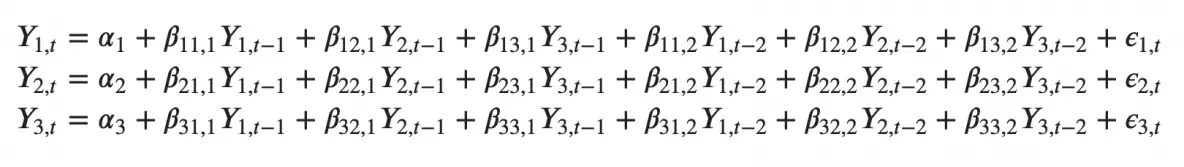
上述方程称为 VAR（1） 模型，因为每个方程的阶数为 1，也就是说，它最多包含一个预测变量（Y1 和 Y2）的滞后。

由于方程中的 Y 项是相互关联的，因此 Y 被视为内生变量，而不是外生预测因子。

类似地，两个变量的二阶 VAR（2） 模型将为每个变量（Y1 和 Y2）包含最多两个滞后。



能想象一个有三个变量（Y2、Y1和Y2）的二阶VAR（3）模型会是：



随着模型中时间序列（变量）数量的增加，方程组会变大。

**第三章：在Python中构建VAR模型**

构建 VAR 模型的过程涉及以下步骤：

1. 分析时间序列特征
2. 检验时间序列之间的因果关系
3. 平稳性测试
4. 如果需要，转换序列以使其静止
5. 找到最佳顺序 （p）
6. 准备训练和测试数据集
7. 训练模型
8. 回滚转换（如果有）
9. 使用测试集评估模型
10. 对未来的预测

**3.1 导入需要的包**

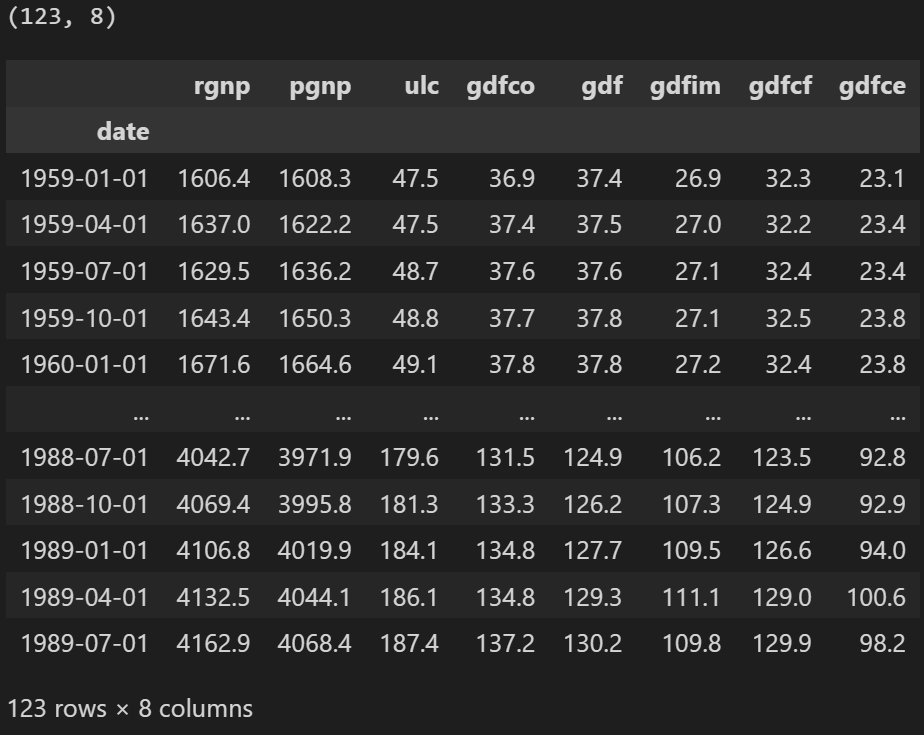
1. import pandas as pd
2. import numpy as np
3. import matplotlib.pyplot as plt
4. %matplotlib inline
5. # Import Statsmodels
6. from statsmodels.tsa.api import VAR
7. from statsmodels.tsa.stattools import adfuller
8. from statsmodels.tools.eval\_measures import rmse, aic

**3.2 导入数据集**

使用Yash P Mehra在1994年的文章“工资增长和通货膨胀过程：一种实证方法”中使用的时间序列。

此数据集具有以下 8 个季度时间序列：

1. rgnp : Real GNP.（实际国民生产总值）
2. pgnp : Potential real GNP.（潜在实际国民生产总值）
3. ulc : Unit labor cost.（单位人工成本）
4. gdfco : Fixed weight deflator for personal consumption expenditure excluding food and energy.（不包括食品和能源的个人消费支出固定权重平减指数）
5. gdf : Fixed weight GNP deflator.（固定权重国民生产总值平减指数）
6. gdfim : Fixed weight import deflator.（固定重量进口平减指数）
7. gdfcf : Fixed weight deflator for food in personal consumption expenditure.（个人消费支出中食品固定权重平减指数）
8. gdfce : Fixed weight deflator for energy in personal consumption expenditure.（个人消费支出能源固定权重平减指数）
9. filepath = 'https://raw.githubusercontent.com/selva86/datasets/master/Raotbl6.csv'
10. df = pd.read\_csv(filepath, parse\_dates = ['date'], index\_col = 'date')
11. print(df.shape)
12. df



**3.3 可视化时间序列**

fig, axes = plt.subplots(nrows=4, ncols=2, dpi=120, figsize=(10,6))

for i, ax in enumerate(axes.flatten()):

    data = df[df.columns[i]]

    ax.plot(data, color='red', linewidth=1)

    # Decorations

    ax.set\_title(df.columns[i])

    ax.xaxis.set\_ticks\_position('none')

    ax.yaxis.set\_ticks\_position('none')

    ax.spines["top"].set\_alpha(0)

    ax.tick\_params(labelsize=6)

plt.tight\_layout()



每个系列多年来都有相当相似的趋势模式，除了 和 ，从 1980 年开始注意到不同的模式。

好的，分析的下一步是检查这些序列之间的因果关系。格兰杰因果关系检验和协整检验可以帮助我们解决这个问题。

**3.4 使用格兰杰因果关系检验检验因果关系**

矢量自动回归背后的基础是系统中的每个时间序列相互影响。也就是说，可以使用自身的过去值以及系统中的其他序列来预测序列。

使用格兰杰因果关系检验，甚至可以在构建模型之前测试这种关系。

格兰杰的因果关系到底检验的是什么呢？

格兰杰因果关系检验回归方程中过去值的系数为零的原假设。

简单来说，时间序列 （X） 的过去值不会导致其他序列 （Y）。因此，如果从检验获得的 p 值小于显著性水平 0.05，则可以安全地否定原假设。

下面的代码对给定数据帧中时间序列的所有可能组合实现格兰杰因果关系检验，并将每个组合的 p 值存储在输出矩阵中。

from statsmodels.tsa.stattools import grangercausalitytests

maxlag = 12

test = 'ssr\_chi2test'

def grangers\_causation\_matrix(data, variables, test='ssr\_chi2test', verbose=False):

    df = pd.DataFrame(np.zeros((len(variables), len(variables))), columns=variables, index=variables)

    for c in df.columns:

        for r in df.index:

            test\_result = grangercausalitytests(data[[r, c]], maxlag=maxlag, verbose=False)

            p\_values = [round(test\_result[i+1][0][test][1],4) for i in range(maxlag)]

            if verbose: print(f'Y = {r}, X = {c}, P Values = {p\_values}')

            min\_p\_value = np.min(p\_values)

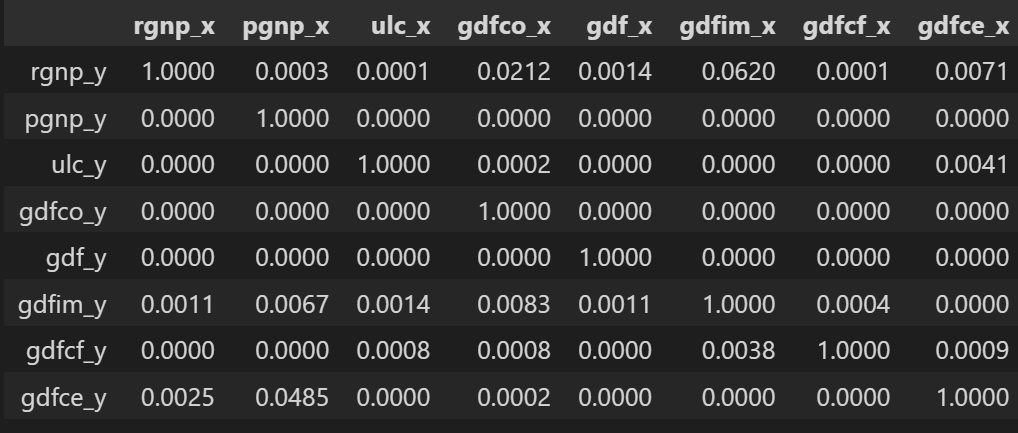
            df.loc[r, c] = min\_p\_value

    df.columns = [var + '\_x' for var in variables]

    df.index = [var + '\_y' for var in variables]

    return df

grangers\_causation\_matrix(df, variables = df.columns)



检查时间序列所有可能组合的格兰杰因果关系。行是响应变量，列是预测变量。

表中的值是 P 值。 P 值小于显着性水平 (0.05)，意味着零假设对应的过去值的系数是零，即 X 不导致 Y 可以被拒绝。

Data: 包含时间序列变量的df数据框

variables : list containing names of the time series variables.

那么如何阅读上面的输出呢？行是响应 （Y），列是预测变量序列 （X）。

例如，如果在（第 0 行，第 0003 列）中取值 1.2，则它是指导致 的 p 值。而 0.000 in （第 2 行，第 1 列） 是指导致 的 p 值。pgnp\_x rgnp\_y rgnp\_y pgnp\_x

那么，如何解释p值呢？

如果给定的 p 值<显著性水平 （0.05），则相应的 X 系列（列）会导致 Y（行）。

例如，P 值 0.0003 在（第 1 行，第 2 列）表示原因的格兰杰因果关系检验的 p 值，该值小于显著性水平 0.05。pgnp\_x rgnp\_y

因此，我们可以拒绝原假设并得出结论原因。pgnp\_x rgnp\_y

查看上表中的 P 值，几乎可以观察到系统中的所有变量（时间序列）都是相互交替的。

这使得这种多时间序列系统成为使用 VAR 模型进行预测的良好候选者。

接下来，让我们进行协整测试。

**3.5 协整测试**

协整检验有助于确定两个或多个时间序列之间存在统计显著性联系。

但是，协整是什么意思？

要理解这一点，首先需要知道什么是“积分顺序”（d）。

积分阶数（d）只不过是使非平稳时间序列静止所需的差分数。

现在，当我们有两个或多个时间序列，并且存在它们的线性组合，其积分阶数 （d） 小于单个序列的阶数时，则序列集合称为共整。

还行？

当两个或多个时间序列共整时，这意味着它们具有长期的统计显著关系。

这是向量自回归（VAR）模型所基于的基本前提。因此，在开始构建VAR模型之前实施协整测试是相当普遍的。

好的，那么如何做这个测试呢？

参考论文：

https://www.jstor.org/stable/2938278?seq=1#page\_scan\_tab\_contents



在Python中实现是相当简单的，如下所示：

from statsmodels.tsa.vector\_ar.vecm import coint\_johansen

def cointegration\_test(df, alpha=0.05):

    out = coint\_johansen(df,-1,5)

    d = {'0.90':0, '0.95':1, '0.99':2}

    traces = out.lr1

    cvts = out.cvt[:, d[str(1 - alpha)]]

    def adjust(val, length = 6): return str(val).ljust(length)

    # Summary

    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'\*20)

    for col, trace, cvt in zip(df.columns, traces, cvts):

        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", adjust(cvt, 8), ' =>  ' , trace > cvt)

cointegration\_test(df)



**3.6 将序列拆分为训练数据和测试数据**

将数据集拆分为训练数据和测试数据。

VAR模型将被拟合，然后用于预测接下来的4个观测值。这些预测将与测试数据中的实际值进行比较。

为了进行比较，我们将使用多个预测准确性指标，如本文后面所示。

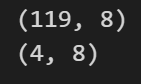
nobs = 4

df\_train, df\_test = df[0:-nobs], df[-nobs:]

# Check size

print(df\_train.shape)  # (119, 8)

print(df\_test.shape)  # (4, 8)



**3.7 检查平稳性并使时间序列保持平稳**

由于 VAR 模型要求要预测的时间序列是平稳的，因此习惯上检查系统中的所有时间序列的平稳性。

刷新一下，平稳时间序列是其特征（如均值和方差）不随时间变化的时间序列。

那么，如何测试平稳性呢？

有一套称为单位根测试的测试。流行的是：

1. 增强迪基-富勒测试（ADF 测试）
2. KPSS测试
3. 菲利普-佩伦测试

让我们使用 ADF 测试来实现我们的目的。

顺便说一下，如果发现某个序列是非平稳的，则通过对序列进行一次差分并再次重复测试直到它变得平稳来使其平稳。

由于差分会将序列的长度减少 1，并且由于所有时间序列的长度必须相同，因此如果选择差分，则需要差分系统中的所有序列。

似乎明白了点

让我们实现 ADF 测试。

首先，我们实现一个不错的函数 （），它写出任何给定时间序列的 ADF 测试结果，并在每个序列上逐个实现此函数。adfuller\_test()

def adfuller\_test(series, signif=0.05, name='', verbose=False):

    """Perform ADFuller to test for Stationarity of given series and print report"""

    r = adfuller(series, autolag='AIC')

    output = {'test\_statistic':round(r[0], 4), 'pvalue':round(r[1], 4), 'n\_lags':round(r[2], 4), 'n\_obs':r[3]}

    p\_value = output['pvalue']

    def adjust(val, length= 6): return str(val).ljust(length)

    # Print Summary

    print(f'    Augmented Dickey-Fuller Test on "{name}"', "\n   ", '-'\*47)

    print(f' Null Hypothesis: Data has unit root. Non-Stationary.')

    print(f' Significance Level    = {signif}')

    print(f' Test Statistic        = {output["test\_statistic"]}')

    print(f' No. Lags Chosen       = {output["n\_lags"]}')

    for key,val in r[4].items():

        print(f' Critical value {adjust(key)} = {round(val, 3)}')

    if p\_value <= signif:

        print(f" => P-Value = {p\_value}. Rejecting Null Hypothesis.")

        print(f" => Series is Stationary.")

    else:

        print(f" => P-Value = {p\_value}. Weak evidence to reject the Null Hypothesis.")

        print(f" => Series is Non-Stationary.")

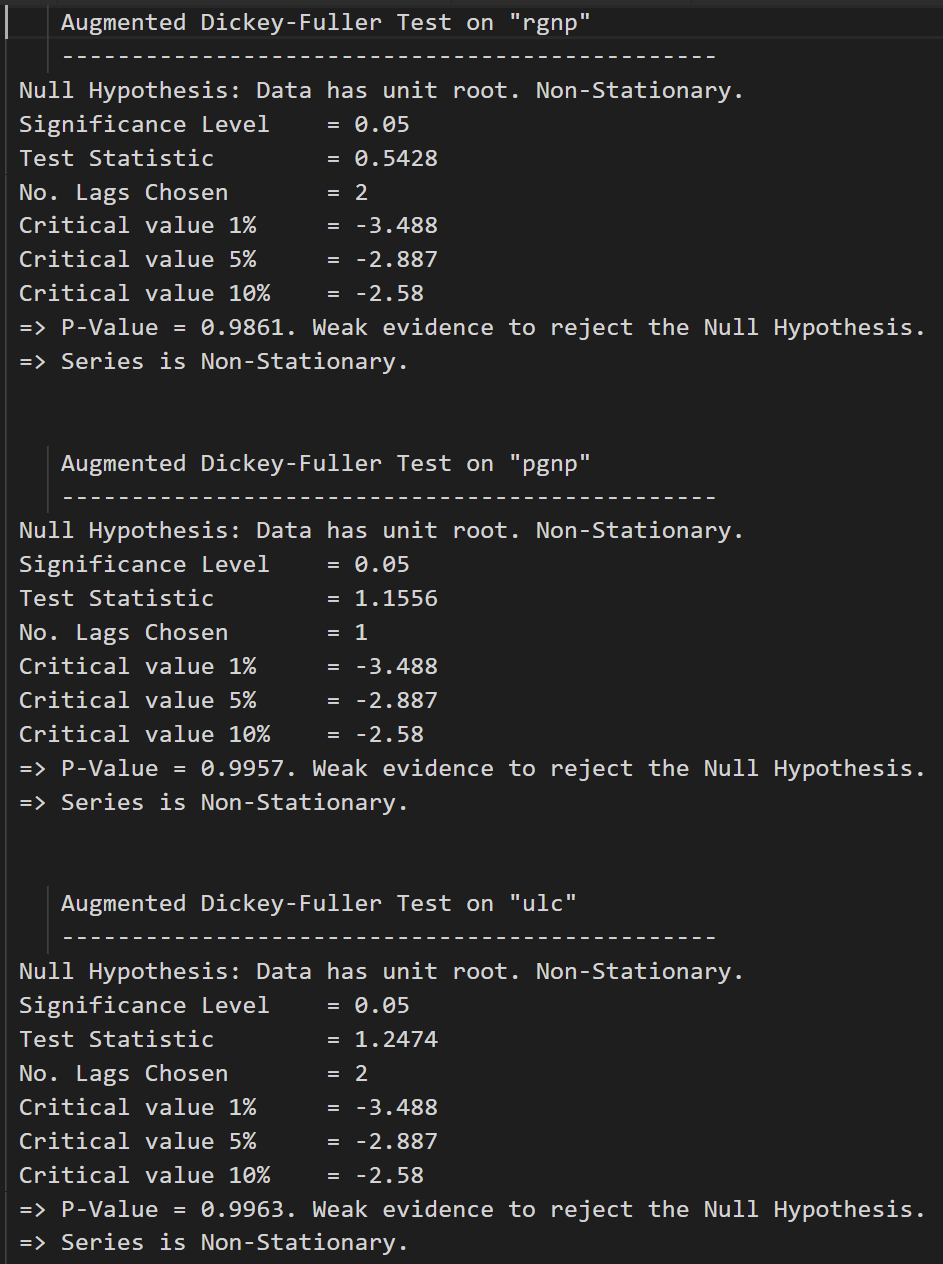
在每个系列上调用：

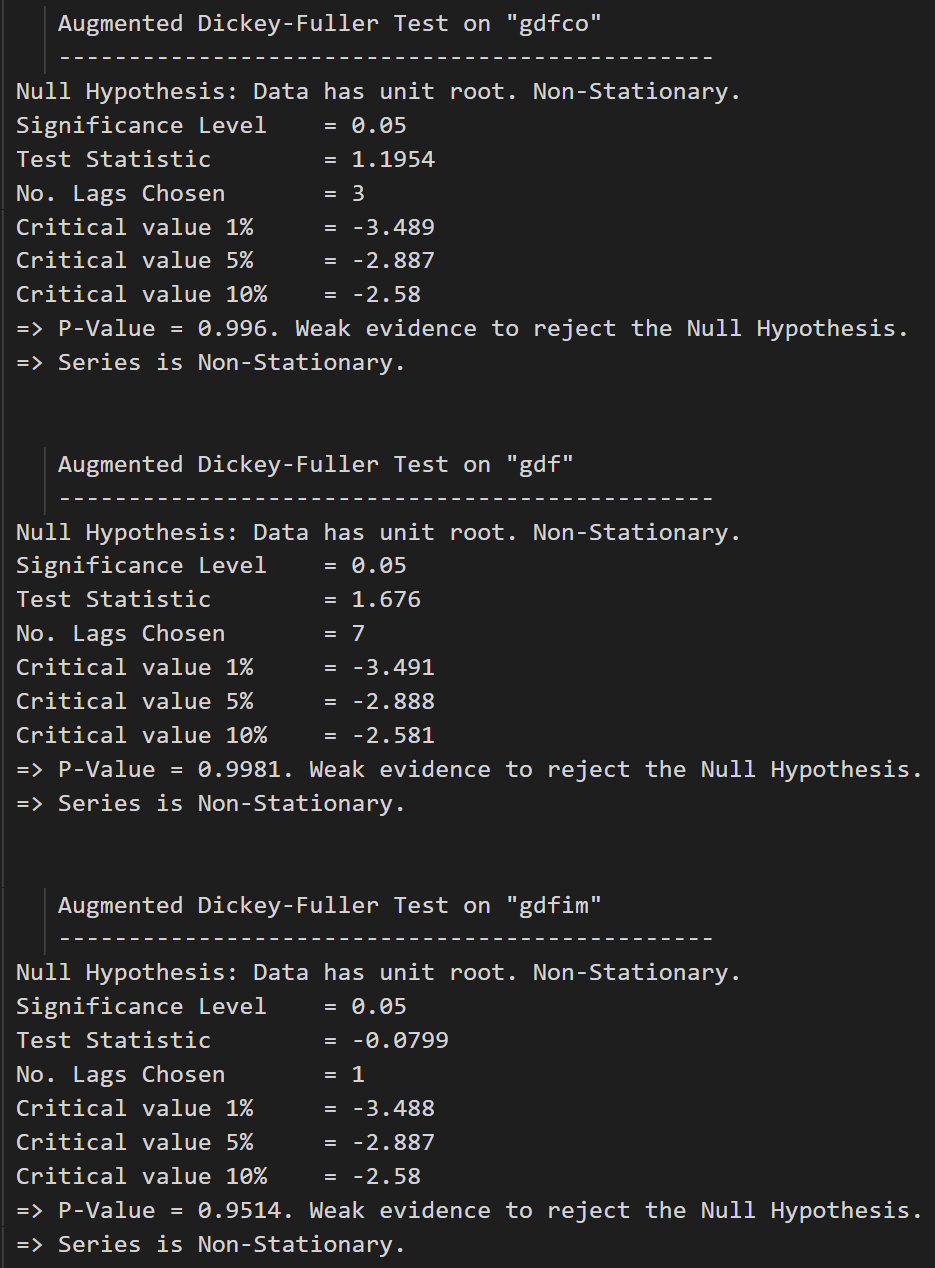
# ADF Test on each column

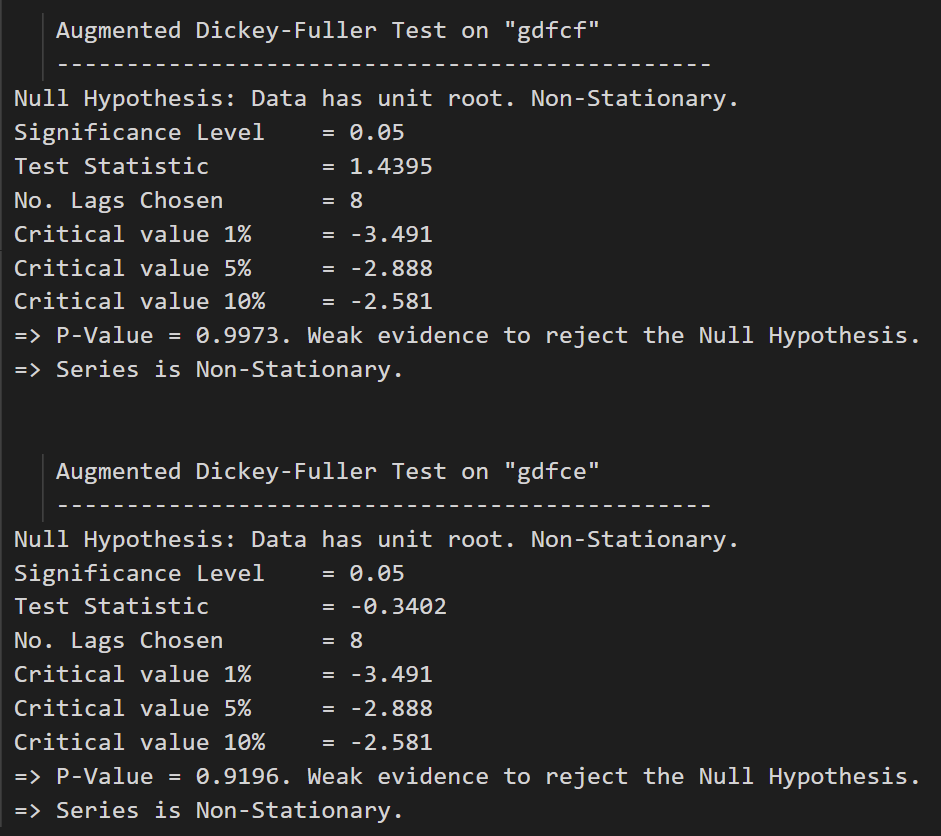
for name, column in df\_train.iteritems():

    adfuller\_test(column, name=column.name)

    print('\n')







ADF 测试确认没有一个时间序列是静止的。让我们将它们全部差分一次，然后再次检查。

# 1st difference

df\_differenced = df\_train.diff().dropna()

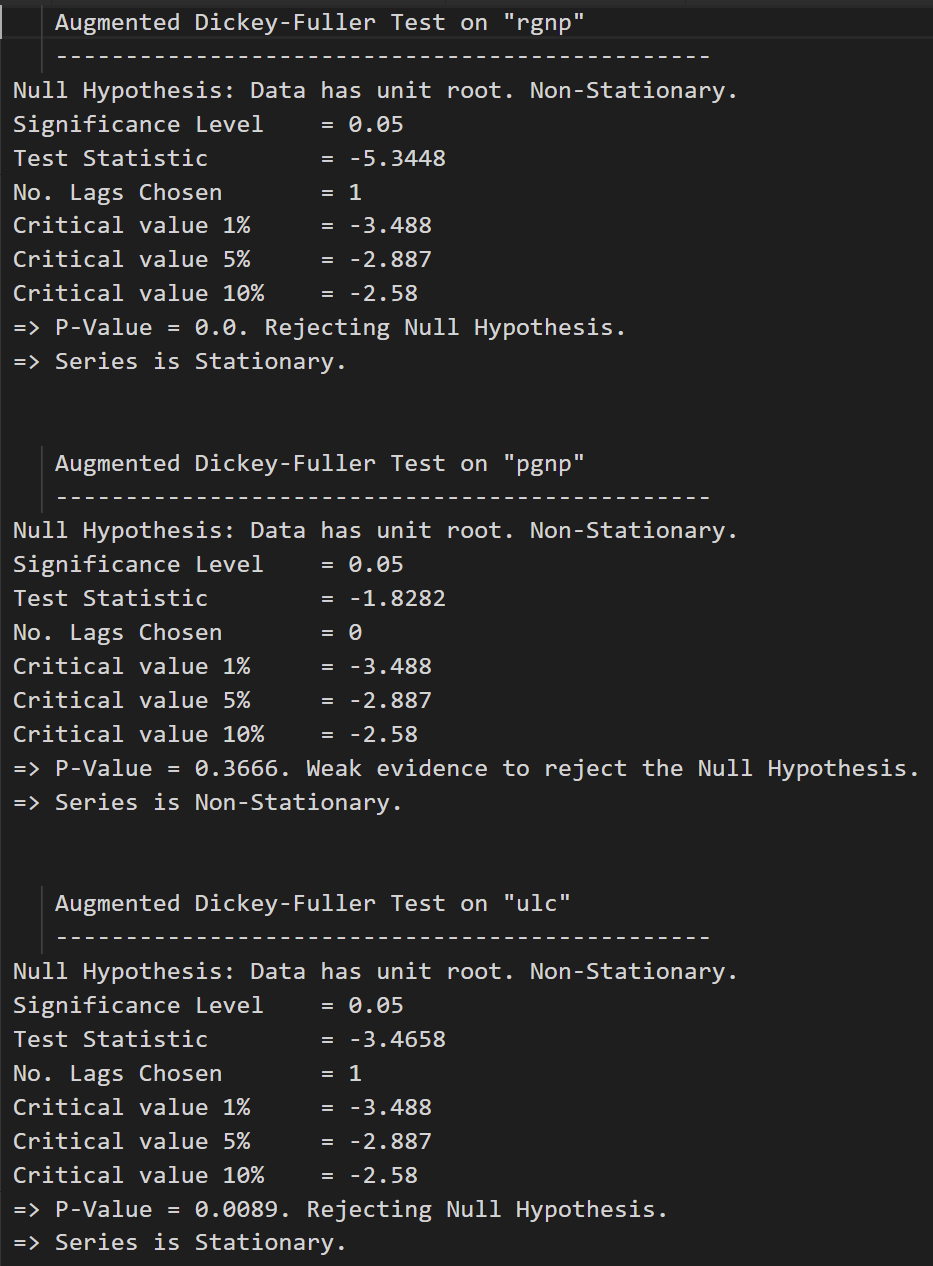
对每个差分序列重新运行 ADF 测试。

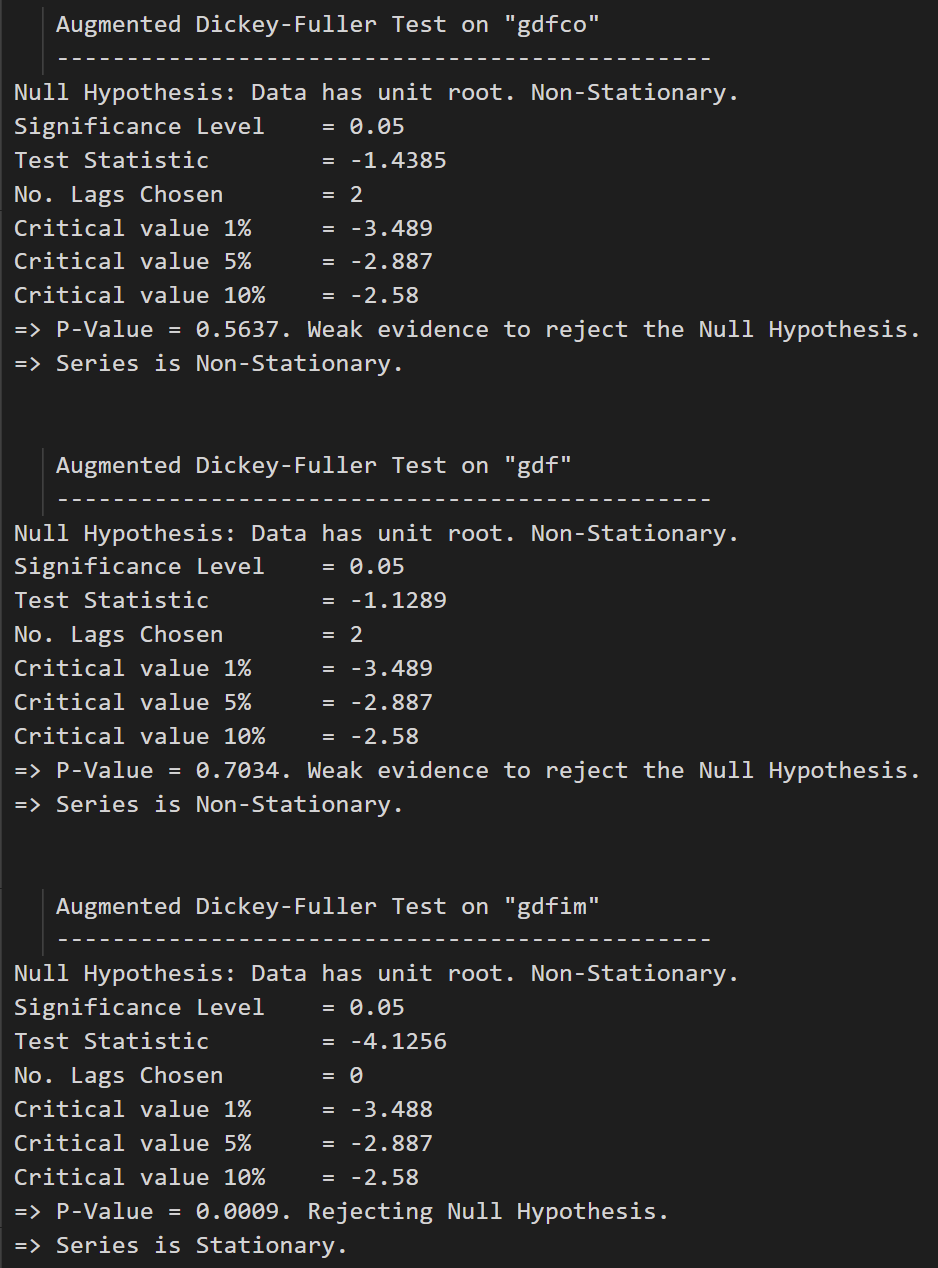
# ADF Test on each column of 1st Differences Dataframe

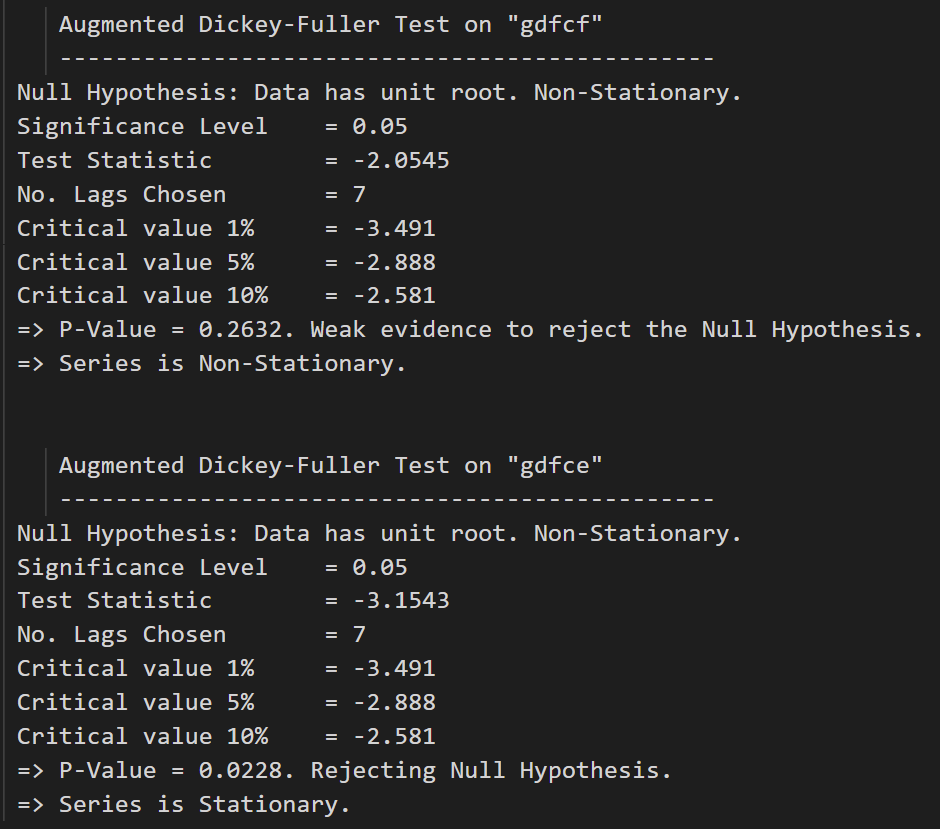
for name, column in df\_differenced.iteritems():

    adfuller\_test(column, name=column.name)

    print('\n')







在第一个差分之后，实际工资（制造业）仍然不是静止的。它的临界值介于 5% 到 10% 的显著性水平之间。

VAR 模型中的所有序列都应具有相同数量的观测值。

因此，我们只剩下两个选择之一。

也就是说，要么继续第一个差分序列，要么再差一次所有系列。

# Second Differencing

df\_differenced = df\_differenced.diff().dropna()

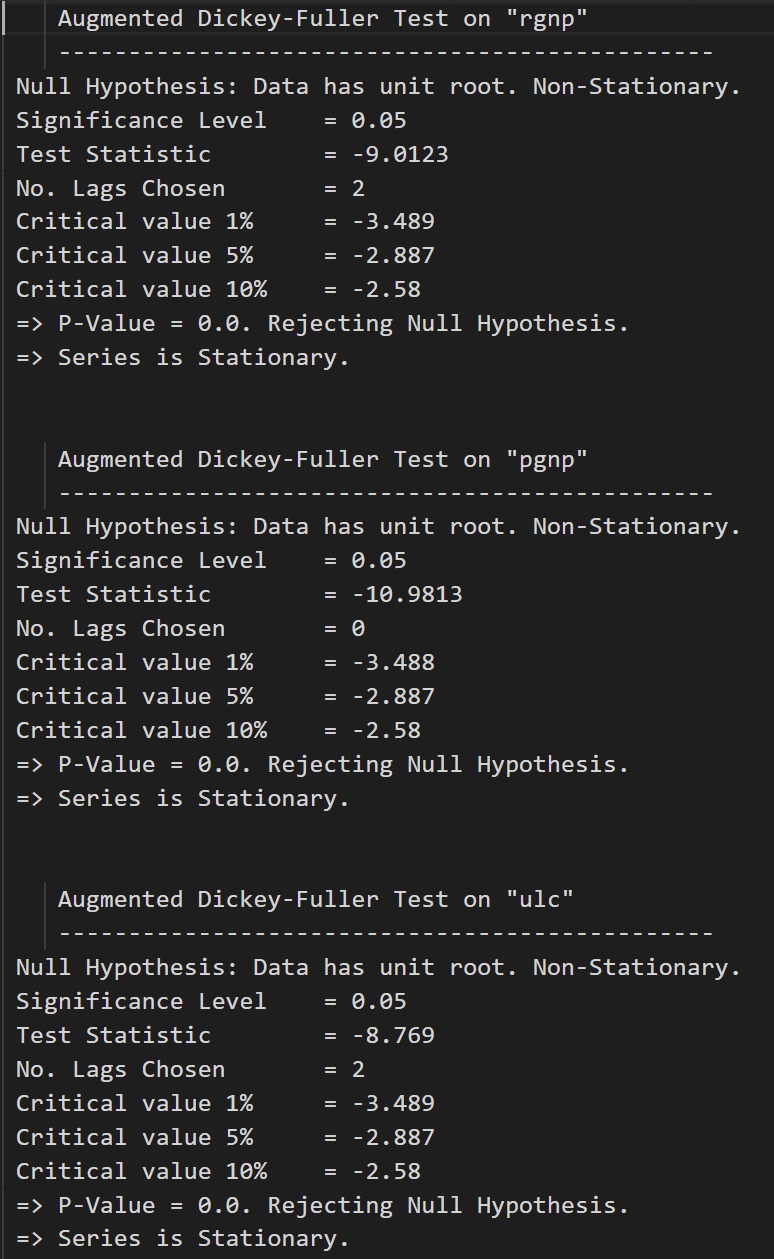
再次运行 ADF 测试。

# ADF Test on each column of 2nd Differences Dataframe

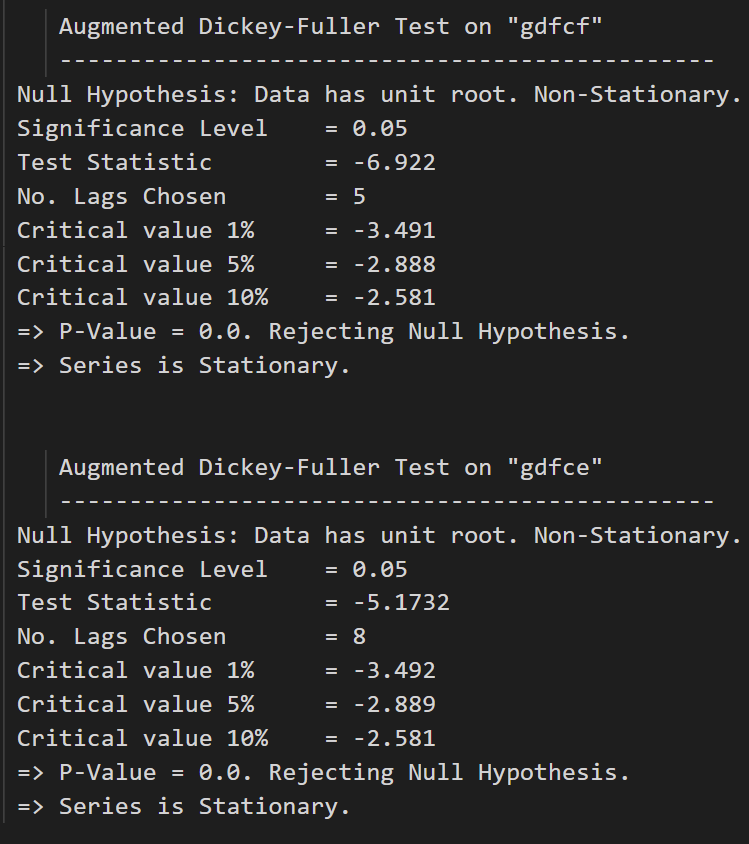
for name, column in df\_differenced.iteritems():

    adfuller\_test(column, name=column.name)

    print('\n')







所有系列现在都是静止的。

让我们准备训练和测试数据集。

**3.8 如何选择VAR模型的阶数（P）**

为了选择 VAR 模型的正确顺序，我们迭代拟合 VAR 模型的递增阶数，并选择给出 AIC 最少模型的顺序。

虽然通常的做法是查看 AIC，但也可以检查BIC和FPE的其他最佳拟合比较估计值。

model = VAR(df\_differenced)

for i in [1,2,3,4,5,6,7,8,9]:

    result = model.fit(i)

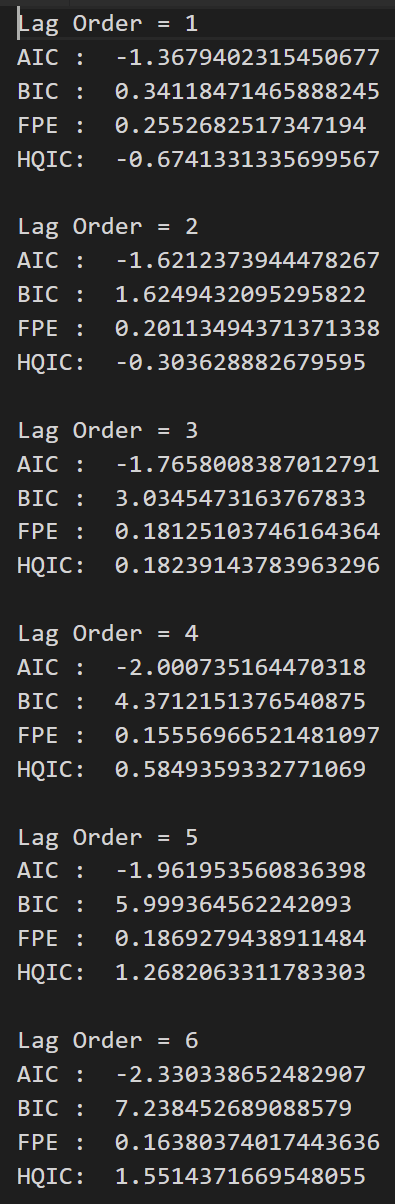
    print('Lag Order =', i)

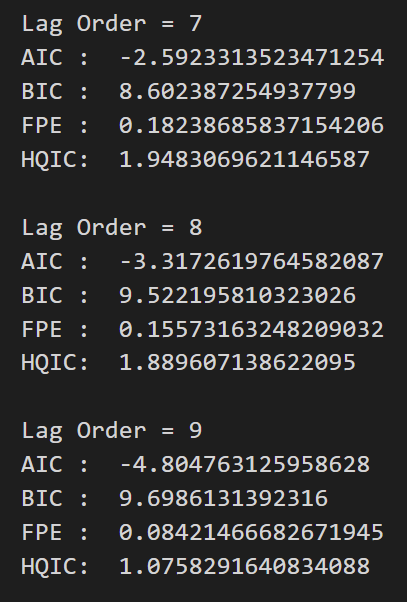
    print('AIC : ', result.aic)

    print('BIC : ', result.bic)

    print('FPE : ', result.fpe)

    print('HQIC: ', result.hqic, '\n')





在上述输出中，AIC 在滞后 4 时降至最低，然后在滞后 5 处增加，然后持续进一步下降。

让我们使用滞后 4 模型。

选择 VAR 模型阶数 （p） 的另一种方法是使用该方法。model.select\_order(maxlags)

所选顺序 （p） 是给出最低“AIC”、“BIC”、“FPE”和“HQIC”分数的顺序。

x = model.select\_order(maxlags=12)

x.summary()



根据 FPE 和 HQIC，在 3 的滞后量级观察到最佳滞后。

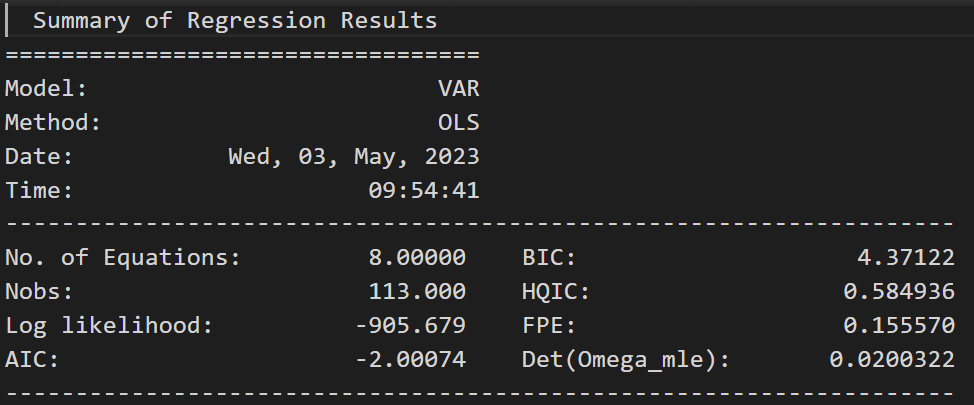
但是，我没有解释为什么观察到的 AIC 和 BIC 值在使用时与在使用时看到的不同。

由于显式计算的 AIC 在滞后 4 时最低，因此我选择所选顺序为 4。

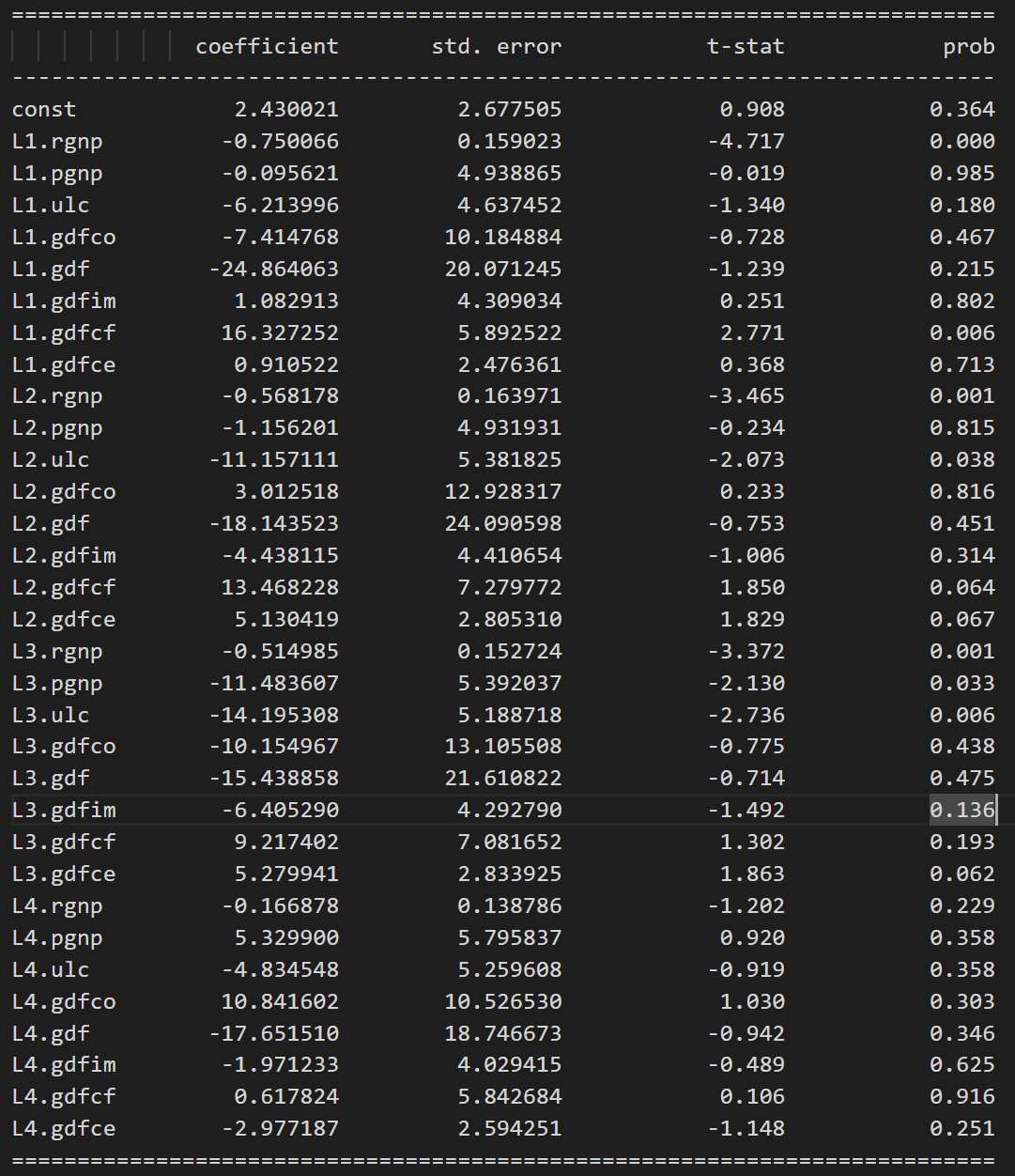
**3.9 训练选定订单的VAR模型（p）**

model\_fitted = model.fit(4)

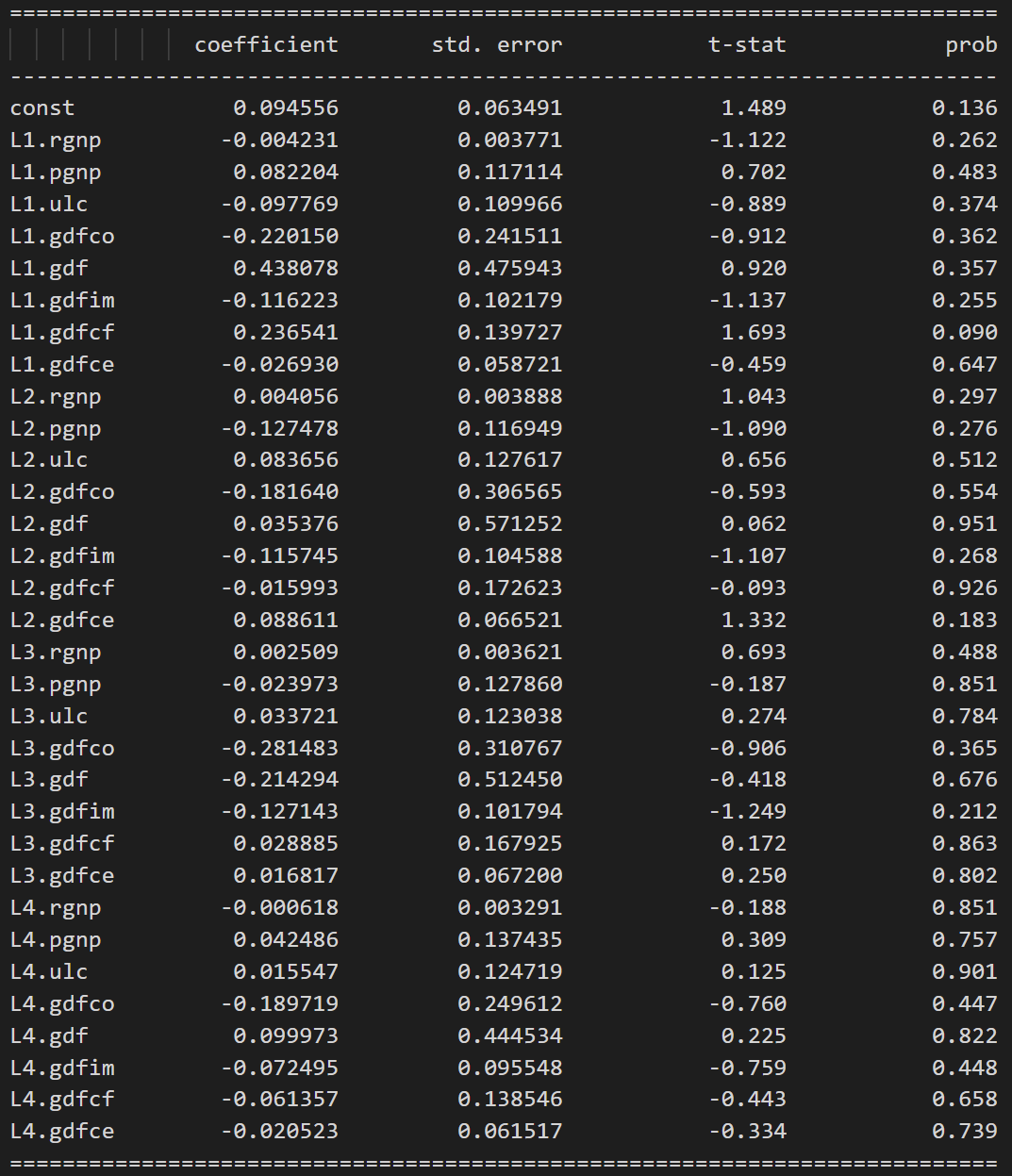
model\_fitted.summary()

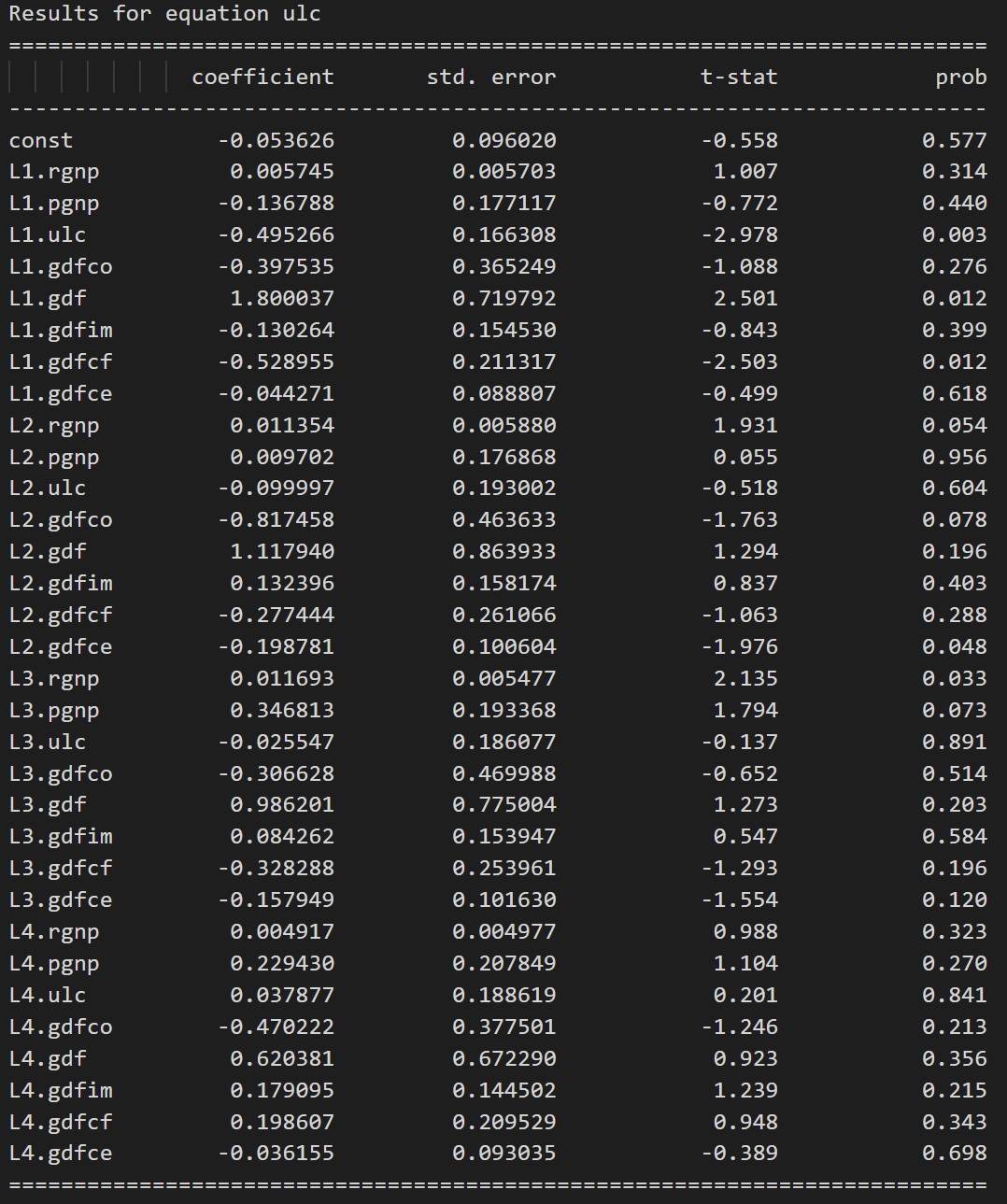


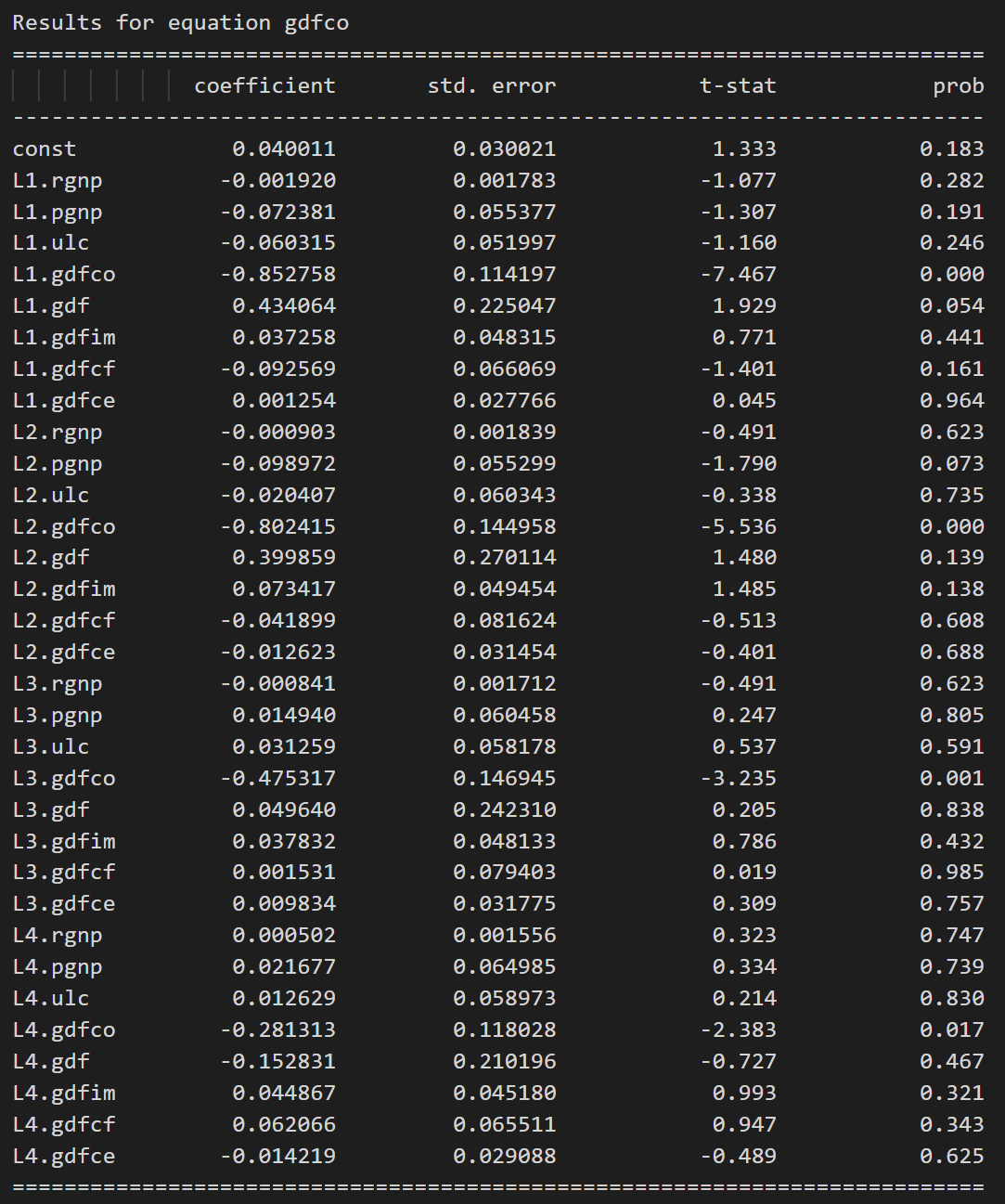
Results for equation rgnp：

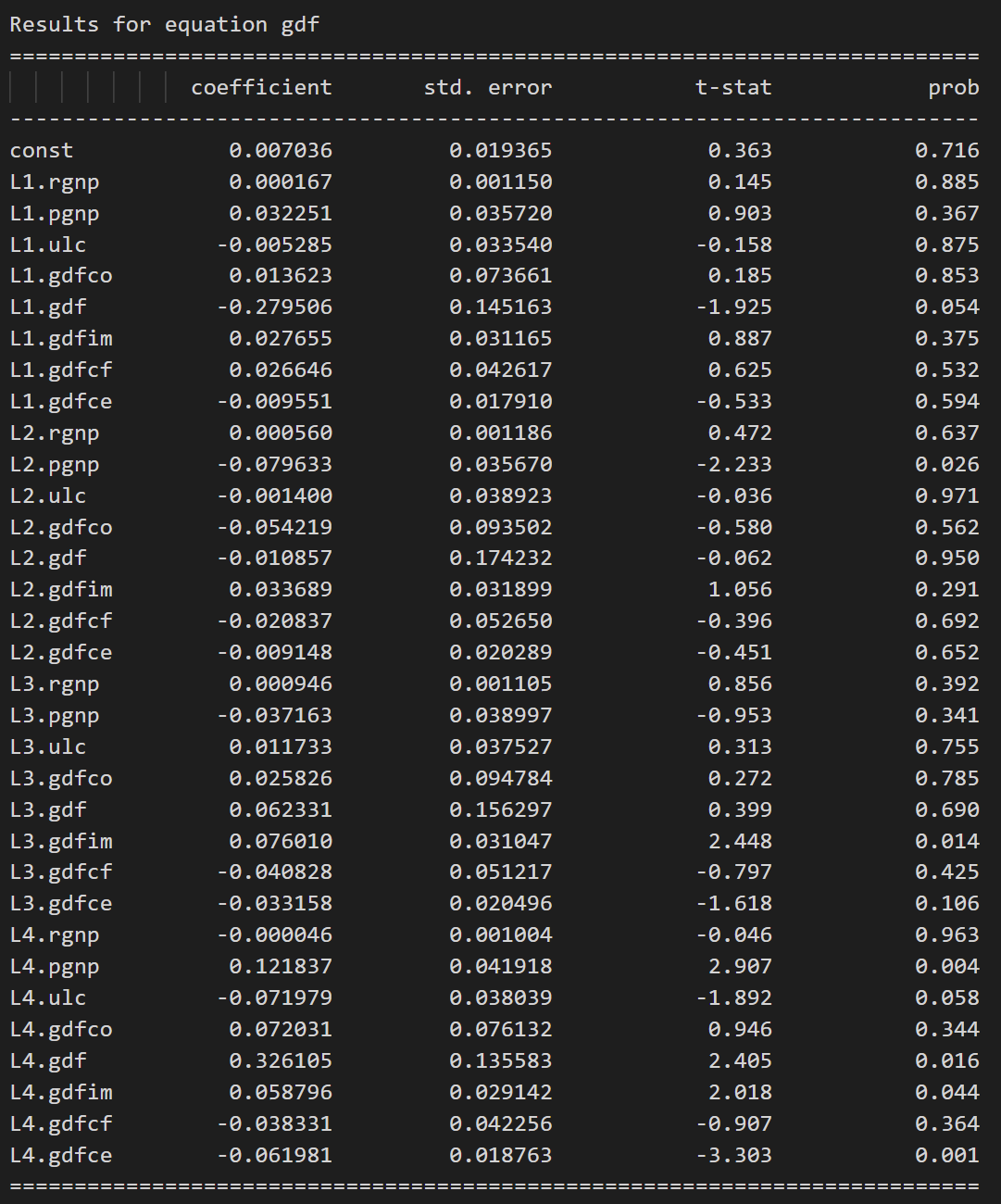


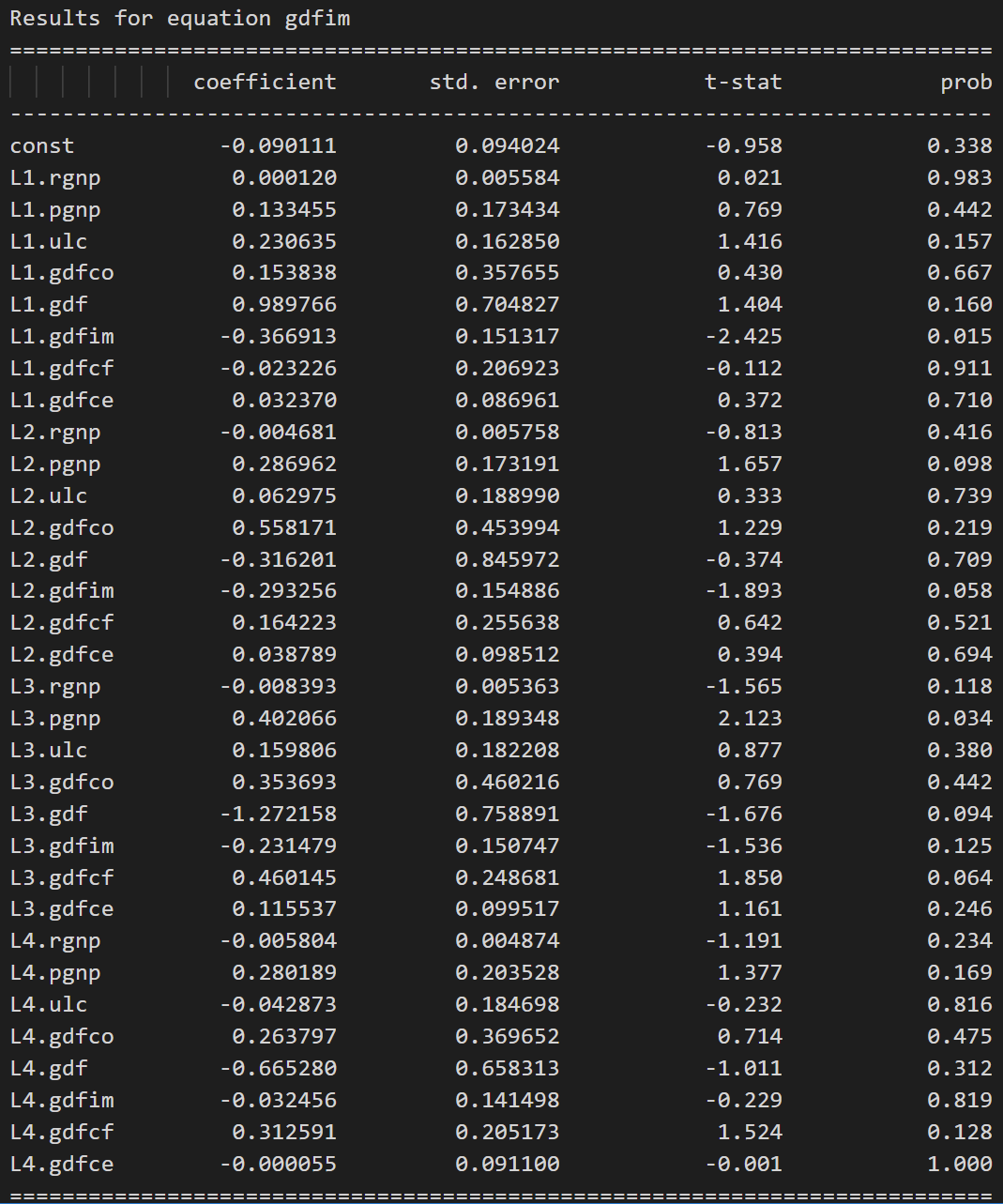
Results for equation pgnp：

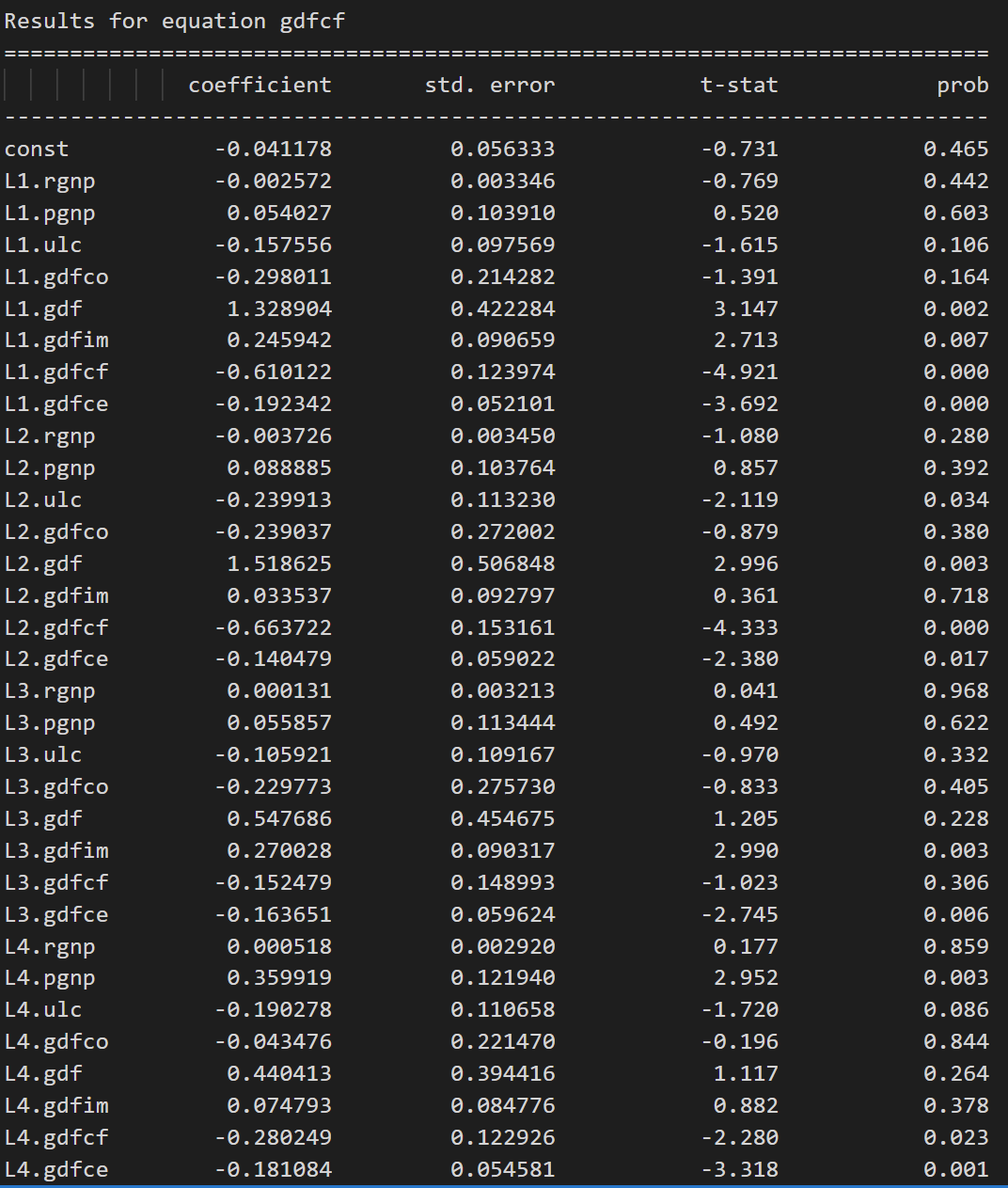


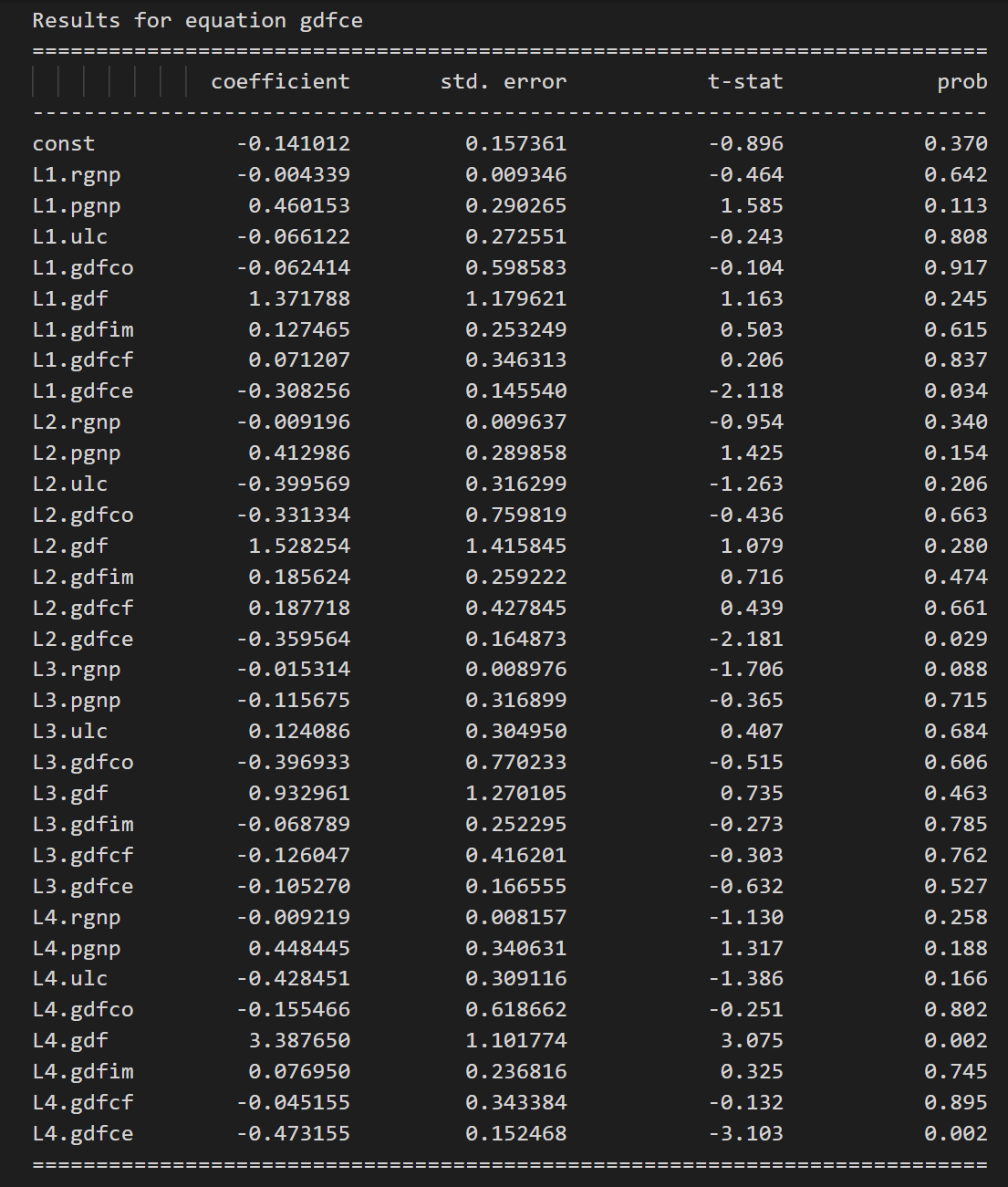


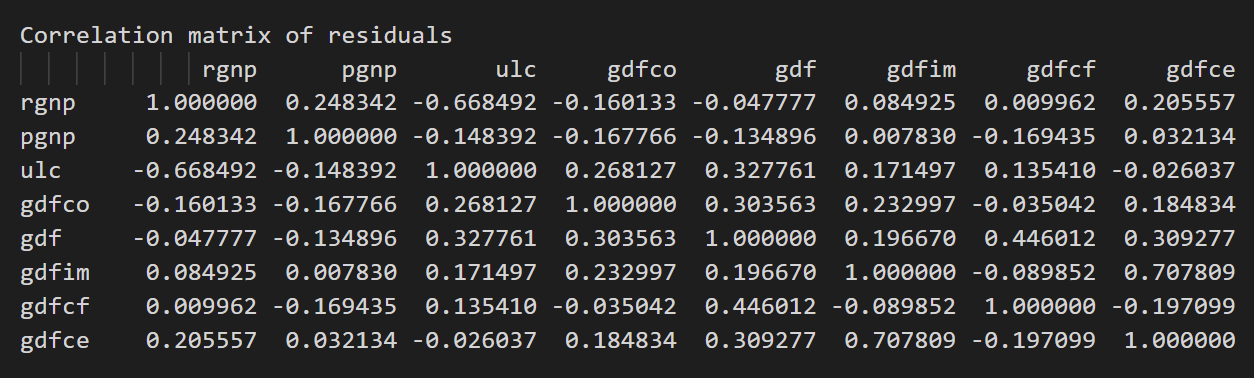












**3.10 使用德宾沃森统计量检查残差（误差）的序列相关性**

残差的序列相关性用于检查残差中是否存在任何剩余模式（误差）。

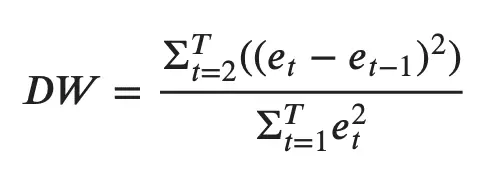
这对我们意味着什么？

如果残差中还存在任何相关性，那么时间序列中仍有一些模式有待模型解释。在这种情况下，典型的行动过程是增加模型的顺序或将更多预测因子引入系统，或者寻找不同的算法来对时间序列进行建模。

因此，检查序列相关性是为了确保模型能够充分解释时间序列中的方差和模式。

好了，回到正题。

检查误差序列相关性的常用方法是使用德宾沃森统计量进行测量。



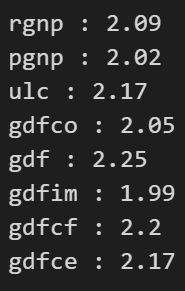
此统计数据的值可以在 0 到 4 之间变化。它越接近值 2，则没有显著的序列相关性。越接近 0，序列相关为正，越接近 4 表示序列相关为负。

from statsmodels.stats.stattools import durbin\_watson

out = durbin\_watson(model\_fitted.resid)

for col, val in zip(df.columns, out):

    print((col), ':', round(val, 2))



序列相关性似乎还不错。让我们继续进行预测。

**3.11 如何使用统计模型预测VAR模型**

为了进行预测，VAR 模型期望过去数据的观测值达到滞后顺序数。

这是因为，VAR 模型中的项本质上是数据集中各种时间序列的滞后，因此您需要为其提供模型使用的滞后顺序所指示的尽可能多的先前值。

# Get the lag order

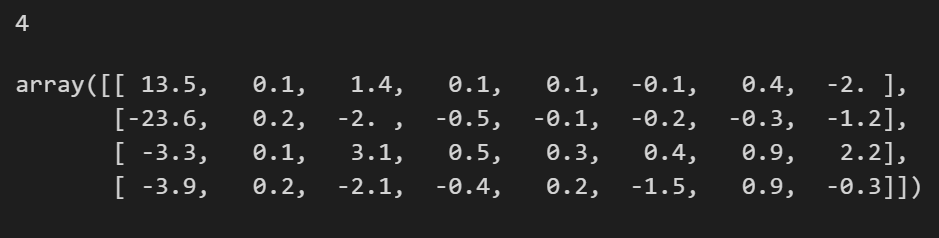
lag\_order = model\_fitted.k\_ar

print(lag\_order)  #> 4

# Input data for forecasting

forecast\_input = df\_differenced.values[-lag\_order:]

forecast\_input



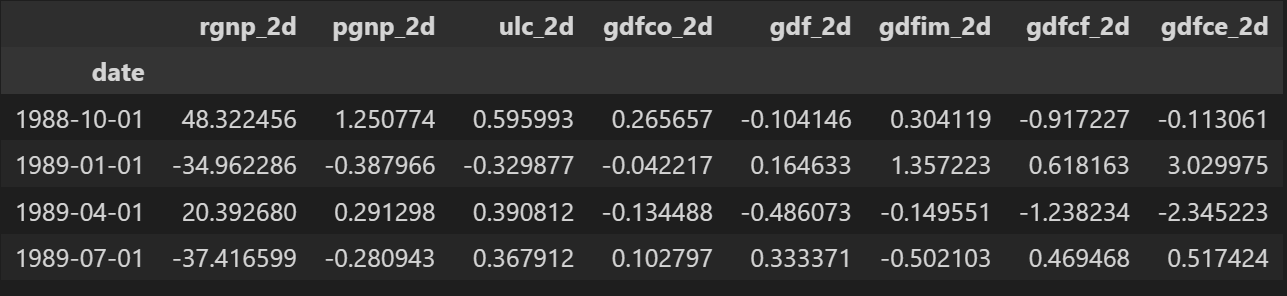
让我们预测一下。

# Forecast

fc = model\_fitted.forecast(y=forecast\_input, steps=nobs)

df\_forecast = pd.DataFrame(fc, index=df.index[-nobs:], columns=df.columns + '\_2d')

df\_forecast



预测是生成的，但它是模型使用的训练数据的规模。因此，要使其恢复到原始比例，您需要将其去差分，就像对原始输入数据进行差分的次数一样。

在这种情况下，它是两次。

**3.12 反转变换以获得真实预测**

def invert\_transformation(df\_train, df\_forecast, second\_diff=False):

    df\_fc = df\_forecast.copy()

    columns = df\_train.columns

    for col in columns:

        # Roll back 2nd Diff

        if second\_diff:

            df\_fc[str(col)+'\_1d'] = (df\_train[col].iloc[-1]-df\_train[col].iloc[-2]) + df\_fc[str(col)+'\_2d'].cumsum()

        # Roll back 1st Diff

        df\_fc[str(col)+'\_forecast'] = df\_train[col].iloc[-1] + df\_fc[str(col)+'\_1d'].cumsum()

    return df\_fc

df\_results = invert\_transformation(df\_train, df\_forecast, second\_diff=True)

df\_results.loc[:, ['rgnp\_forecast', 'pgnp\_forecast', 'ulc\_forecast', 'gdfco\_forecast',

                   'gdf\_forecast', 'gdfim\_forecast', 'gdfcf\_forecast', 'gdfce\_forecast']]



预测恢复到原来的规模。让我们根据测试数据的实际值绘制预测值。

**3.13 预测与实际图**

fig, axes = plt.subplots(nrows=int(len(df.columns)/2), ncols=2, dpi=150, figsize=(10,10))

for i, (col,ax) in enumerate(zip(df.columns, axes.flatten())):

    df\_results[col+'\_forecast'].plot(legend=True, ax=ax).autoscale(axis='x',tight=True)

    df\_test[col][-nobs:].plot(legend=True, ax=ax);

    ax.set\_title(col + ": Forecast vs Actuals")

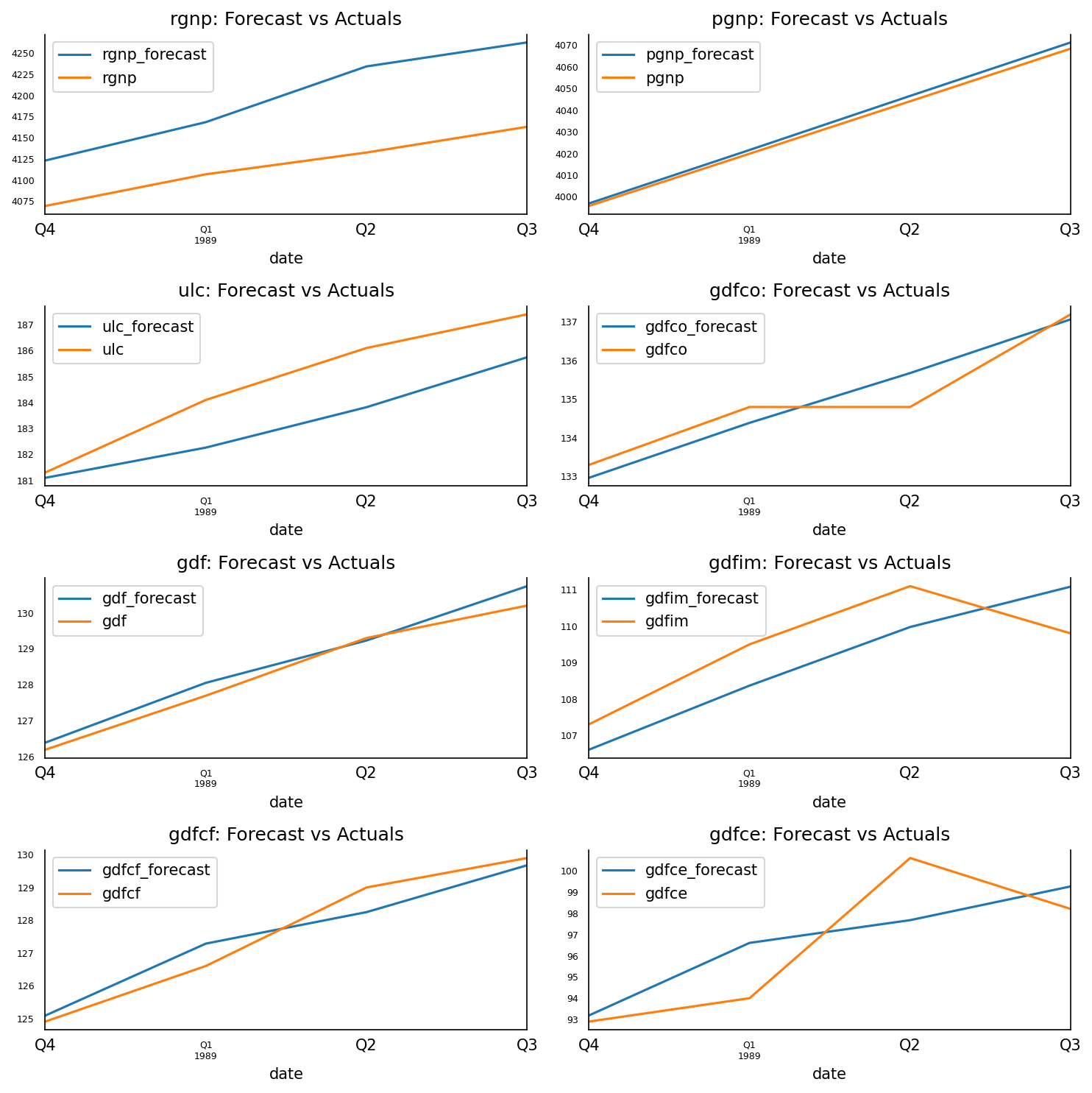
    ax.xaxis.set\_ticks\_position('none')

    ax.yaxis.set\_ticks\_position('none')

    ax.spines["top"].set\_alpha(0)

    ax.tick\_params(labelsize=6)

plt.tight\_layout()



**3.14 评估预测**

为了评估预测，让我们计算一组全面的指标，即MAPE，ME，MAE，MPE，RMSE，corr和minmax。

from statsmodels.tsa.stattools import acf

def forecast\_accuracy(forecast, actual):

    mape = np.mean(np.abs(forecast - actual)/np.abs(actual))  # MAPE

    me = np.mean(forecast - actual)             # ME

    mae = np.mean(np.abs(forecast - actual))    # MAE

    mpe = np.mean((forecast - actual)/actual)   # MPE

    rmse = np.mean((forecast - actual)\*\*2)\*\*.5  # RMSE

    corr = np.corrcoef(forecast, actual)[0,1]   # corr

    mins = np.amin(np.hstack([forecast[:,None],

                              actual[:,None]]), axis=1)

    maxs = np.amax(np.hstack([forecast[:,None],

                              actual[:,None]]), axis=1)

    minmax = 1 - np.mean(mins/maxs)             # minmax

    return({'mape':mape, 'me':me, 'mae': mae,

            'mpe': mpe, 'rmse':rmse, 'corr':corr, 'minmax':minmax})

print('Forecast Accuracy of: rgnp')

accuracy\_prod = forecast\_accuracy(df\_results['rgnp\_forecast'].values, df\_test['rgnp'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: pgnp')

accuracy\_prod = forecast\_accuracy(df\_results['pgnp\_forecast'].values, df\_test['pgnp'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: ulc')

accuracy\_prod = forecast\_accuracy(df\_results['ulc\_forecast'].values, df\_test['ulc'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: gdfco')

accuracy\_prod = forecast\_accuracy(df\_results['gdfco\_forecast'].values, df\_test['gdfco'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: gdf')

accuracy\_prod = forecast\_accuracy(df\_results['gdf\_forecast'].values, df\_test['gdf'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: gdfim')

accuracy\_prod = forecast\_accuracy(df\_results['gdfim\_forecast'].values, df\_test['gdfim'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: gdfcf')

accuracy\_prod = forecast\_accuracy(df\_results['gdfcf\_forecast'].values, df\_test['gdfcf'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

print('\nForecast Accuracy of: gdfce')

accuracy\_prod = forecast\_accuracy(df\_results['gdfce\_forecast'].values, df\_test['gdfce'])

for k, v in accuracy\_prod.items():

    print(k, ': ', round(v,4))

