# Классификация и регрессия многомерных временных рядов

Классификация и регрессия многомерных временных рядов с использование специальных методов машинного обучения. Особенности представления многомерных временных рядов в sktime. Изучение метода WEASEL. Изучение методов векторной авторегрессии библиотеки statsmodels.

## Импорт и даные

```
!pip install -U sktime
```

```
import matplotlib.pyplot as plt
import numpy as np

import pandas as pd

%matplotlib inline
```

### Набор данных

Набор данных BasicMotions. Набор данных BasicMotions состоит из четырех классов: ходьбы, отдыха, бега и бадминтона. Данные собранны с помощью умных часов с 3D-акселерометром и 3D-гироскопом. От участников эксперимента по сбору данных требовалось записывать движение в общей сложности пять раз. При это данные отбирался каждые десятые доли секунды в течение десяти секунд.

```
from sklearn.model_selection import train_test_split
from sktime.datasets import load_basic_motions

X, y = load_basic_motions(return_X_y=True)
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)
```

```
(60, 6) (60,) (20, 6) (20,)
```

```
#  multivariate input data
X_train.head()
```

```
# multi-class target variable
np.unique(y_train)
```

```
array(['badminton', 'running', 'standing', 'walking'], dtype=object)
```

### Методы классификации в Sktime

Sktime предлагает три основных способа решения задач многомерной классификации временных рядов:

- Конкатенация столбцов временных рядов в один столбец длинных временных рядов с помощью ColumnConcatenator и применение одномерных классификаторов к объединенным данным.

- Объединение данных по столбцам с помощью ColumnEnsembleClassifier, в котором один и тот же классификатор проходит для каждого столбца временных рядов, как для одномерных данных, и их прогнозы, в конце, агрегированы.
- Специальные методы обработки многомерных временных рядов, например поиск объединяющих функций или общих представлений данных, таких как шейплеты или словари в многомерных пространствах.

**Конкатенация временных рядов**

Способ конкатенации - использовать метод ColumnConcatenator. Метод может быть объединен в конвейер с любым одномерным классификатором.

```
from sktime.transformations.panel.compose import ColumnConcatenator
from sktime.classification.interval_based import
TimeSeriesForestClassifier
from sklearn.pipeline import Pipeline

steps = [
    ("concatenate", ColumnConcatenator()),
    ("classify", TimeSeriesForestClassifier(n_estimators=100)),
]
clf = Pipeline(steps)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)

1.0
```

**Объединение данных по столбцам**

Мы также можем установить один классификатор для каждого столбца временных рядов как для одномерного, а затем агрегировать их прогнозы. Для этого можо использовать класс методов ColumnEnsembleClassifier. Интерфейс класса похож на знакомый ColumnTransformer от sklearn. Однако, ColumnEnsembleClassifier позволяет использовать разные методы оценки для разных столбцов или подмножеств столбцов входных данных. Все они будут обработаны отдельно, а признаки, генерируемые каждым преобразователем, будут объединены для формирования единого выхода.

```
from sktime.classification.compose import ColumnEnsembleClassifier
from sktime.classification.interval_based import
RandomIntervalSpectralForest
from sktime.classification.dictionary_based import ContractableBOSS
from sktime.classification.shapelet_based import ROCKETClassifier
from sktime.classification.interval_based import
TimeSeriesForestClassifier

clf = ColumnEnsembleClassifier(
    estimators=[
        ("TSF0", TimeSeriesForestClassifier(n_estimators=50), [0]),
#column 0
        ("cBOSS1", ContractableBOSS(), [1]), #column 1
        ("cBOSS2", ContractableBOSS(), [2]), #column 2
        ("RISF", RandomIntervalSpectralForest(n_estimators=50), [3]),
```

```
    #column 3
        ]
)
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
1.0
```

**Специальные методы обработки многомерных временных рядов**

**WEASEL (Извлечение слов для классификации временных рядов) + MUSE (Многовариантное символьное расширение)**

Здесь мы будем использовать метод WEASEL, преобразуя временные ряды в векторы признаков. Для этого будет использоваться подход скользящего окна. Полученные признки затем анализируются с помощью классификатора машинного обучения. Метод WEASEL основан на методе BOSS и его модификациях. В расширении WEASEL MUSE для многомерных данных применяется особая техника многомерного символьного расширения.

```
from sktime.classification.dictionary_based import MUSE
from sktime.classification.dictionary_based import WEASEL

muse = MUSE()
muse.fit(X_train, y_train)
muse.score(X_test, y_test)
```

```
1.0
```

**Mr-SEQL**

Другой специальный метод - это **Mr-SEQL**, который извлекает признаки из каждого измерения данных независимо. Для этого используя несколько символьных представлений временных рядов (SAX, SFA). Затем к извлеченным признакам применяется логистическая регрессия.

```
from sktime.classification.shapelet_based import MrSEQLClassifier

clf = MrSEQLClassifier()
clf.fit(X_train, y_train)
clf.score(X_test, y_test)
```

```
1.0
```

### Упражнение 1

1. Попробуйте использовать стандартные классификаторы из sklearn для описанного набора данных. Сравните результаты.

### Упражнение 2

1. Загрузите набор данных *japanese_vowels* с многомерными данными (*load_japanese_vowels*) попробуйте применить изученные классификаторы к этому набору данных и сравните результаты.

## Предсказание многомерных данных с помощью векторной авторегрессии.

### Набор данных.

VAR (векторная авторегрессия) - это частный случай методов на основе AR для многомерных данных (см. работу про ARMA).

Здесь мы воспользуемся набором данных эмпирического исследования инфляции, взятого на слух.

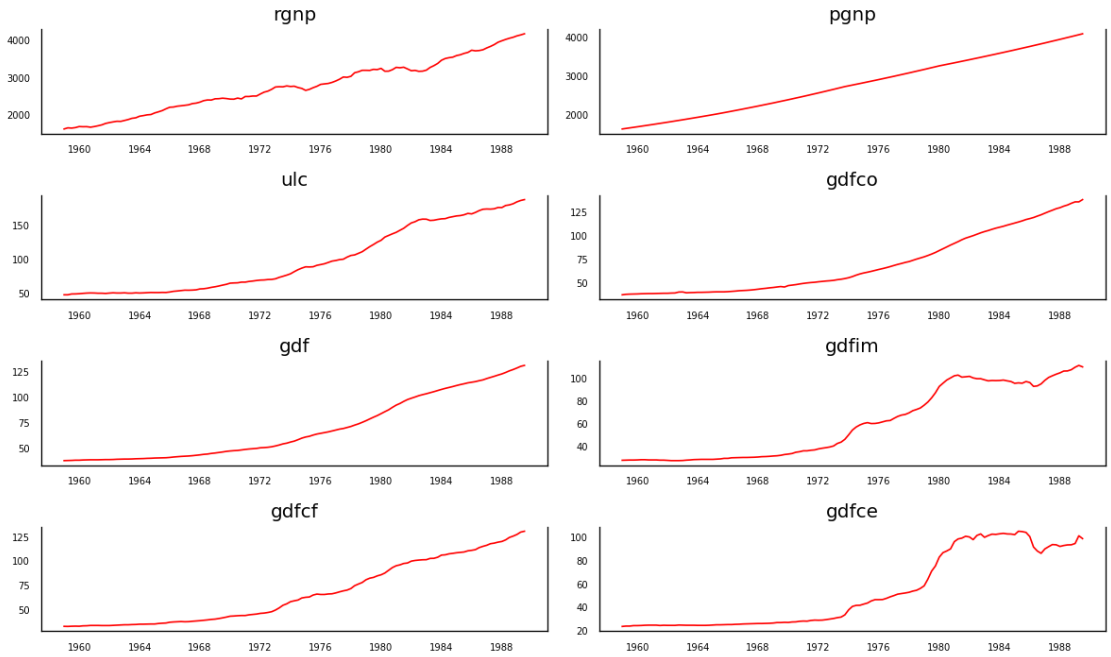
```
filepath =
'https://raw.githubusercontent.com/selva86/datasets/master/Raotbl6.csv'
df = pd.read_csv(filepath, parse_dates=['date'], index_col='date')
print(df.shape)  # (123, 8)
df.tail()
```

```
(123, 8)
```

| date | rgnp | pgnp | ulc | gdfco | gdf | gdfim | gdfcf | gdfce |
|---|---|---|---|---|---|---|---|---|
| 1988-07-01 | 4042.7 | 3971.9 | 179.6 | 131.5 | 124.9 | 106.2 | 123.5 | 92.8 |
| 1988-10-01 | 4069.4 | 3995.8 | 181.3 | 133.3 | 126.2 | 107.3 | 124.9 | 92.9 |
| 1989-01-01 | 4106.8 | 4019.9 | 184.1 | 134.8 | 127.7 | 109.5 | 126.6 | 94.0 |
| 1989-04-01 | 4132.5 | 4044.1 | 186.1 | 134.8 | 129.3 | 111.1 | 129.0 | 100.6 |
| 1989-07-01 | 4162.9 | 4068.4 | 187.4 | 137.2 | 130.2 | 109.8 | 129.9 | 98.2 |

Давайте исследуем и визуализируем его.

```
# Plot
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();
```




Кроме того, мы можем проверить причинно-следственную связь (казуальность) между этими рядами с помощью теста причинности Грейнджера и теста коинтеграции.

Наличие казуальности означает влияние одного ряда на другой (например, поведение одного ряда является причиной запаздывающего поведения другого ряда). В случае причинно-следственной связи для любого ряда вы можете предсказать его значения с учетом прошлых значений самого себя вместе с результатами для других рядов в системе. В идеале в многомерном ряду должна быть казуальность.

```
from statsmodels.tsa.stattools import grangercausalitytests

maxlag=12
test = 'ssr_chi2test'

def grangers_causation_matrix(data, variables, test='ssr_chi2test',
verbose=False):
    """
    Check Granger Causality of all possible combinations of the Time
series.
        The rows are the response variable, columns are predictors. The
values in the table
        are the P-Values.
        P-Values lesser than the significance level (0.05), i
        mplies the Null Hypothesis that the coefficients
        of the corresponding past values is zero, that is,
        the X does not cause Y can be rejected.

    Paramteres
    ------------
    data : pandas dataframe,
         containing the time series variables.
    variables : list,
```

```
        containing names of the time series variables.

    Returns
    ----------
    casual matrx: pandas dataframe,
        matrix with p-values.

    """
    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] = int(100*min_p_value)/100 #for rounding to
second sign after dot.

    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)
```

| | rgnp_x | pgnp_x | ulc_x | gdfco_x | gdf_x | gdfim_x | gdfcf_x | gdfce_x |
|---|---|---|---|---|---|---|---|---|
| rgnp_y | 1.0 | 0.00 | 0.0 | 0.02 | 0.0 | 0.06 | 0.0 | 0.0 |
| pgnp_y | 0.0 | 1.00 | 0.0 | 0.00 | 0.0 | 0.00 | 0.0 | 0.0 |
| ulc_y | 0.0 | 0.00 | 1.0 | 0.00 | 0.0 | 0.00 | 0.0 | 0.0 |
| gdfco_y | 0.0 | 0.00 | 0.0 | 1.00 | 0.0 | 0.00 | 0.0 | 0.0 |
| gdf_y | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 | 0.00 | 0.0 | 0.0 |
| gdfim_y | 0.0 | 0.00 | 0.0 | 0.00 | 0.0 | 1.00 | 0.0 | 0.0 |
| gdfcf_y | 0.0 | 0.00 | 0.0 | 0.00 | 0.0 | 0.00 | 1.0 | 0.0 |
| gdfce_y | 0.0 | 0.04 | 0.0 | 0.00 | 0.0 | 0.00 | 0.0 | 1.0 |

Итак, мы видим, что все переменные имеют причинность (p-значение меньше уровня значимости 0,05 - таким образом, мы отвергаем нулевую гипотезу об их независимости). Другими словами, все переменные (временные ряды) в системе взаимозаменяемо

вызывают друг друга. Это позволяет использовать модели VAR для прогнозирования этой системы временных рядов.

Кроме того, мы протестируем нашу систему на тесте коинтеграции. Коинтеграционный тест. помогает установить наличие статистически значимой связи между двумя или более временными рядами. Когда два или более временных ряда коинтегрируются, это означает, что они имеют долгосрочную статистически значимую взаимосвязь.

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen

def cointegration_test(df, alpha=0.05):
    """Perform Johanson's Cointegration Test and Report Summary"""
    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)
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 ----------------------------------------
rgnp   ::  248.0     > 143.6691  =>   True
pgnp   ::  183.12    > 111.7797  =>   True
ulc    ::  130.01    > 83.9383   =>   True
gdfco  ::  85.28     > 60.0627   =>   True
gdf    ::  55.05     > 40.1749   =>   True
gdfim  ::  31.59     > 24.2761   =>   True
gdfcf  ::  14.06     > 12.3212   =>   True
gdfce  ::  0.45      > 4.1296    =>   False
```

Таким образом, все, кроме последнего ряда, могут быть коингерированными. Сделаем разделение тренировочной и тест выборок.

```
TEST_SIZE = 20
df_train, df_test = df[0:-TEST_SIZE], df[-TEST_SIZE:]

# Check size
print(df_train.shape)  #
print(df_test.shape)  #
```

```
(103, 8)
(20, 8)
```

Для того, чтобы сделать VAR нам также нужно проверить наши ряды на стационарность. Очевидно, что исходный ряд не стационарен. Мы можем проверить производную (или двойную производную), чтобы привести временной ряд к стационарному.

```
df_differenced  = df_train.diff().diff().dropna()
df_differenced.shape
```

```
(101, 8)
```

```
from statsmodels.tsa.stattools import adfuller

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.")


for name, column in df_differenced.iteritems():
    adfuller_test(column, name=column.name)
    print('\n')
```

```
    Augmented Dickey-Fuller Test on "rgnp"
    -----------------------------------------------
 => P-Value = 0.0. Rejecting Null Hypothesis.
 => Series is Stationary.


    Augmented Dickey-Fuller Test on "pgnp"
    -----------------------------------------------
 => P-Value = 0.0. Rejecting Null Hypothesis.
 => Series is Stationary.
```

```
   Augmented Dickey-Fuller Test on "ulc"
   ---------------------------------------------
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "gdfco"
   ---------------------------------------------
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "gdf"
   ---------------------------------------------
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "gdfim"
   ---------------------------------------------
=> P-Value = 0.0. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "gdfcf"
   ---------------------------------------------
=> P-Value = 0.0107. Rejecting Null Hypothesis.
=> Series is Stationary.


   Augmented Dickey-Fuller Test on "gdfce"
   ---------------------------------------------
=> P-Value = 0.0013. Rejecting Null Hypothesis.
=> Series is Stationary.
```

### Исследование предсказаний методом VAR

Попытаемся найти лучший порядок модели VAR для изучаемого временного ряда.

```
from statsmodels.tsa.api import VAR
from statsmodels.tools.eval_measures import rmse, aic

model = VAR(df_differenced)

orders_grid = range(1,11)

for order in orders_grid:
    result = model.fit(maxlags=order)
    print('Lag Order =', order, end = ' \t' )
    print('AIC : ', (result.aic * 100)//100 , end = ' \t' )
```

```
    print('BIC : ', (result.bic* 100)//100 , end = ' \t' )
    print('HQIC: ', (result.hqic* 100)//100 , end = ' \n' )
```

```
Lag Order = 1     AIC :  -3.0       BIC :  -1.0       HQIC:  -2.0
Lag Order = 2     AIC :  -3.0       BIC :  0.0  HQIC:  -2.0
Lag Order = 3     AIC :  -4.0       BIC :  2.0  HQIC:  -1.0
Lag Order = 4     AIC :  -4.0       BIC :  3.0  HQIC:  -1.0
Lag Order = 5     AIC :  -4.0       BIC :  5.0  HQIC:  -1.0
Lag Order = 6     AIC :  -5.0       BIC :  5.0  HQIC:  -1.0
Lag Order = 7     AIC :  -7.0       BIC :  6.0  HQIC:  -2.0
Lag Order = 8     AIC :  -7.0       BIC :  7.0  HQIC:  -2.0
Lag Order = 9     AIC :  -11.0      BIC :  5.0  HQIC:  -4.0
Lag Order = 10    AIC :  -17.0      BIC :  1.0  HQIC:  -10.0
```

Здесь мы можем выбрать лаг номер 10 из-за минимальности критериев AIC и BIC. Альтернативный метод выбора порядка (p) моделей VAR - использовать метод model.select_order (maxlags). Выбранный порядок (p) - это порядок, который дает самые низкие оценки целевых показателей.

```
x = model.select_order(maxlags=10)
x.summary()
```

```
<class 'statsmodels.iolib.table.SimpleTable'>
```

Здесь же можно выбрать лаг 10. Попробуем модель и сделам для нее прогноз.

```
model_fitted = model.fit(10)
model_fitted.summary()
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Thu, 06, May, 2021
Time:                     13:29:06
--------------------------------------------------------------------
No. of Equations:         8.00000    BIC:                    1.55401
Nobs:                     91.0000    HQIC:                  -9.11225
Log likelihood:           357.824    FPE:                0.000406468
AIC:                     -16.3255    Det(Omega_mle):     2.49534e-06
--------------------------------------------------------------------
Results for equation rgnp
===============================================================================
               coefficient       std. error           t-stat            prob
-------------------------------------------------------------------------------
const            10.961108         8.108555            1.352           0.176
L1.rgnp          -0.289441         0.437760           -0.661           0.508
L1.pgnp           1.766595        15.395450            0.115           0.909
L1.ulc            4.968445        15.602307            0.318           0.750
L1.gdfco         59.807552        37.660066            1.588           0.112
L1.gdf           21.551681        47.962697            0.449           0.653
L1.gdfim         50.469749        40.318172            1.252           0.211
L1.gdfcf          2.001096        25.535114            0.078           0.938
```

| | | | | |
|---|---|---|---|---|
| L1.gdfce | -41.667132 | 25.400615 | -1.640 | 0.101 |
| L2.rgnp | -0.153897 | 0.525975 | -0.293 | 0.770 |
| L2.pgnp | 25.425671 | 18.849172 | 1.349 | 0.177 |
| L2.ulc | 13.893131 | 20.801834 | 0.668 | 0.504 |
| L2.gdfco | 11.294964 | 29.336411 | 0.385 | 0.700 |
| L2.gdf | -5.149716 | 63.615018 | -0.081 | 0.935 |
| L2.gdfim | -8.761459 | 35.899269 | -0.244 | 0.807 |
| L2.gdfcf | 8.276169 | 28.385694 | 0.292 | 0.771 |
| L2.gdfce | 12.505169 | 22.005904 | 0.568 | 0.570 |
| L3.rgnp | -1.032423 | 0.641259 | -1.610 | 0.107 |
| L3.pgnp | -4.363007 | 13.903035 | -0.314 | 0.754 |
| L3.ulc | -25.732302 | 19.936867 | -1.291 | 0.197 |
| L3.gdfco | -65.487618 | 34.991350 | -1.872 | 0.061 |
| L3.gdf | 50.241986 | 77.952055 | 0.645 | 0.519 |
| L3.gdfim | -72.184607 | 43.875041 | -1.645 | 0.100 |
| L3.gdfcf | 0.876908 | 31.330987 | 0.028 | 0.978 |
| L3.gdfce | 20.616932 | 19.458613 | 1.060 | 0.289 |
| L4.rgnp | -1.042113 | 0.780056 | -1.336 | 0.182 |
| L4.pgnp | -35.335017 | 17.095562 | -2.067 | 0.039 |
| L4.ulc | -8.643392 | 22.271938 | -0.388 | 0.698 |
| L4.gdfco | 10.821848 | 33.802505 | 0.320 | 0.749 |
| L4.gdf | 75.721449 | 105.281815 | 0.719 | 0.472 |
| L4.gdfim | 12.046864 | 36.167860 | 0.333 | 0.739 |
| L4.gdfcf | -55.125906 | 36.538890 | -1.509 | 0.131 |
| L4.gdfce | -19.061640 | 26.165830 | -0.728 | 0.466 |
| L5.rgnp | -1.106692 | 0.775773 | -1.427 | 0.154 |
| L5.pgnp | 11.484546 | 12.664658 | 0.907 | 0.365 |
| L5.ulc | 22.555188 | 26.003602 | 0.867 | 0.386 |
| L5.gdfco | 9.730838 | 39.722666 | 0.245 | 0.806 |
| L5.gdf | -49.599132 | 110.162051 | -0.450 | 0.653 |
| L5.gdfim | 28.590239 | 30.644514 | 0.933 | 0.351 |
| L5.gdfcf | 40.060230 | 35.277424 | 1.136 | 0.256 |
| L5.gdfce | 16.135333 | 23.160742 | 0.697 | 0.486 |
| L6.rgnp | -0.600804 | 0.755748 | -0.795 | 0.427 |
| L6.pgnp | 24.691642 | 18.414748 | 1.341 | 0.180 |
| L6.ulc | 25.659348 | 34.684448 | 0.740 | 0.459 |
| L6.gdfco | -15.825576 | 33.890239 | -0.467 | 0.641 |
| L6.gdf | -286.584339 | 165.827852 | -1.728 | 0.084 |
| L6.gdfim | -33.607115 | 25.362884 | -1.325 | 0.185 |
| L6.gdfcf | 52.558637 | 51.953177 | 1.012 | 0.312 |
| L6.gdfce | 36.517452 | 27.047688 | 1.350 | 0.177 |
| L7.rgnp | -0.783274 | 0.684260 | -1.145 | 0.252 |
| L7.pgnp | -29.321245 | 16.987606 | -1.726 | 0.084 |
| L7.ulc | -4.787894 | 31.797309 | -0.151 | 0.880 |
| L7.gdfco | 2.470476 | 30.554357 | 0.081 | 0.936 |
| L7.gdf | -110.241741 | 124.639669 | -0.884 | 0.376 |
| L7.gdfim | -10.723240 | 22.233003 | -0.482 | 0.630 |
| L7.gdfcf | -18.211105 | 26.207002 | -0.695 | 0.487 |
| L7.gdfce | 34.128883 | 26.345370 | 1.295 | 0.195 |
| L8.rgnp | -0.461890 | 0.530381 | -0.871 | 0.384 |
| L8.pgnp | -17.548020 | 14.957340 | -1.173 | 0.241 |
| L8.ulc | -8.937186 | 33.911876 | -0.264 | 0.792 |

| | | | | |
|---|---|---|---|---|
| L8.gdfco | -0.939297 | 29.343366 | -0.032 | 0.974 |
| L8.gdf | 60.598971 | 119.714229 | 0.506 | 0.613 |
| L8.gdfim | -13.968582 | 17.591071 | -0.794 | 0.427 |
| L8.gdfcf | -31.859381 | 24.570670 | -1.297 | 0.195 |
| L8.gdfce | -7.596453 | 20.259235 | -0.375 | 0.708 |
| L9.rgnp | -0.759340 | 0.475196 | -1.598 | 0.110 |
| L9.pgnp | 23.630280 | 17.367019 | 1.361 | 0.174 |
| L9.ulc | -15.809308 | 28.302857 | -0.559 | 0.576 |
| L9.gdfco | 23.472448 | 26.619442 | 0.882 | 0.378 |
| L9.gdf | -225.948688 | 121.289898 | -1.863 | 0.062 |
| L9.gdfim | -18.827809 | 24.366003 | -0.773 | 0.440 |
| L9.gdfcf | 28.226395 | 31.075405 | 0.908 | 0.364 |
| L9.gdfce | -5.107341 | 18.622411 | -0.274 | 0.784 |
| L10.rgnp | -0.315439 | 0.438678 | -0.719 | 0.472 |
| L10.pgnp | -39.801073 | 22.354642 | -1.780 | 0.075 |
| L10.ulc | 5.983507 | 21.529761 | 0.278 | 0.781 |
| L10.gdfco | 6.656294 | 23.560607 | 0.283 | 0.778 |
| L10.gdf | -187.547795 | 109.557900 | -1.712 | 0.087 |
| L10.gdfim | 26.471104 | 17.510484 | 1.512 | 0.131 |
| L10.gdfcf | 43.334368 | 37.380604 | 1.159 | 0.246 |
| L10.gdfce | 43.066363 | 28.964944 | 1.487 | 0.137 |

Results for equation pgnp

| | coefficient | std. error | t-stat | prob |
|---|---|---|---|---|
| const | 0.071770 | 0.225331 | 0.319 | 0.750 |
| L1.rgnp | -0.007481 | 0.012165 | -0.615 | 0.539 |
| L1.pgnp | 0.413218 | 0.427829 | 0.966 | 0.334 |
| L1.ulc | -0.041095 | 0.433577 | -0.095 | 0.924 |
| L1.gdfco | -0.369833 | 1.046547 | -0.353 | 0.724 |
| L1.gdf | 0.056462 | 1.332850 | 0.042 | 0.966 |
| L1.gdfim | -0.683348 | 1.120414 | -0.610 | 0.542 |
| L1.gdfcf | 0.209568 | 0.709603 | 0.295 | 0.768 |
| L1.gdfce | 0.184715 | 0.705865 | 0.262 | 0.794 |
| L2.rgnp | 0.008324 | 0.014616 | 0.570 | 0.569 |
| L2.pgnp | -0.770887 | 0.523805 | -1.472 | 0.141 |
| L2.ulc | 0.021806 | 0.578068 | 0.038 | 0.970 |
| L2.gdfco | 0.004414 | 0.815238 | 0.005 | 0.996 |
| L2.gdf | 0.561598 | 1.767817 | 0.318 | 0.751 |
| L2.gdfim | -0.036314 | 0.997616 | -0.036 | 0.971 |
| L2.gdfcf | -1.380325 | 0.788819 | -1.750 | 0.080 |
| L2.gdfce | 0.049528 | 0.611529 | 0.081 | 0.935 |
| L3.rgnp | 0.005827 | 0.017820 | 0.327 | 0.744 |
| L3.pgnp | 0.335829 | 0.386356 | 0.869 | 0.385 |
| L3.ulc | 0.489828 | 0.554032 | 0.884 | 0.377 |
| L3.gdfco | 0.155515 | 0.972385 | 0.160 | 0.873 |
| L3.gdf | -1.249281 | 2.166233 | -0.577 | 0.564 |
| L3.gdfim | -0.089946 | 1.219257 | -0.074 | 0.941 |
| L3.gdfcf | 0.573562 | 0.870666 | 0.659 | 0.510 |
| L3.gdfce | -0.122050 | 0.540741 | -0.226 | 0.821 |

```
L4.rgnp          0.019606        0.021677          0.904          0.366
L4.pgnp         -0.010203        0.475074         -0.021          0.983
L4.ulc           0.445603        0.618922          0.720          0.472
L4.gdfco         0.400878        0.939348          0.427          0.670
L4.gdf          -1.156002        2.925708         -0.395          0.693
L4.gdfim        -0.764893        1.005080         -0.761          0.447
L4.gdfcf        -0.241491        1.015390         -0.238          0.812
L4.gdfce         0.329685        0.727130          0.453          0.650
L5.rgn
```

Для полученной модели мы можем проверить некоторые оставшиеся паттерны в остатках с помощью теста durbin_watson. Для этого теста значение статистики durbin_watson может варьироваться от 0 до 4. Чем ближе к значению 2, тем больше вероятности, что значимой корреляции в остатках нет. Чем ближе к 0, тем больше вероятность, что имеется положительная корреляция в ряду, а чем ближе к 4, тем больше отрицательная корреляция во временном ряду.

```
from statsmodels.stats.stattools import durbin_watson
out = durbin_watson(model_fitted.resid)

print(out)
```

```
[2.48414886 2.12539032 1.87510872 2.70538591 1.82807508 2.67882018
 1.69115062 2.23853132]
```

Вероятно, модель вполне неплохо выбрана. Теперь сделаем прогноз. Для первого прогнозируемого (вне выборки) значения нам нужно взять последние значения lag_order наших тренировочных данных. Потому что это объем данных необходим для дальнейшего прогнозирования.

```
# Get the lag order
lag_order = model_fitted.k_ar
print(lag_order)  #7

# Input data for forecasting
forecast_input = df_differenced.values[-lag_order:]
forecast_input.shape
```

```
10

(10, 8)
```

Теперь сделаем предсказание.

```
# Forecast
fc = model_fitted.forecast(y=forecast_input, steps=TEST_SIZE)

df_forecast = pd.DataFrame(fc, index=df.index[-TEST_SIZE:],
columns=df.columns + '_2d')

df_forecast
```

| date | rgnp_2d | pgnp_2d | ulc_2d | gdfco_2d | gdf_2d | gdfim_2d |
|---|---|---|---|---|---|---|
| 1984-10-01 | -254.223407 | 0.794179 | 3.635249 | -2.090034 | -1.231496 | -3.999701 |
| 1985-01-01 | 545.492461 | -2.869124 | -15.802553 | -0.348122 | 0.028494 | 2.459419 |
| 1985-04-01 | -428.584066 | 4.186865 | 13.625643 | -1.246798 | 0.305369 | -4.685231 |
| 1985-07-01 | 140.682735 | 1.904957 | -9.044991 | 2.859512 | -1.231346 | -0.214502 |
| 1985-10-01 | 393.312114 | 1.745448 | -5.334634 | -3.801062 | 0.604795 | 2.523042 |
| 1986-01-01 | -431.430526 | 3.094269 | 9.248289 | 2.467791 | -0.068532 | -0.623337 |
| 1986-04-01 | 211.462016 | -2.870648 | -5.645376 | 1.704179 | -0.418001 | 3.685622 |
| 1986-07-01 | -319.855266 | 2.616225 | 12.554721 | -3.826439 | 0.219042 | 0.617610 |
| 1986-10-01 | 638.529366 | -4.189477 | -22.001288 | 3.701743 | 0.202896 | 6.943042 |
| 1987-01-01 | -860.693134 | 0.870008 | 30.302504 | -2.795695 | 0.067845 | -6.273698 |
| 1987-04-01 | 97.968225 | -3.481638 | -20.451789 | -0.142631 | -2.983454 | -0.986872 |
| 1987-07-01 | 1027.403961 | 2.329599 | -13.315277 | -3.335507 | 2.536531 | 0.191114 |
| 1987-10-01 | -1437.484738 | 4.418689 | 30.753789 | 1.265989 | -1.674456 | -14.270573 |
| 1988-01-01 | 1465.037828 | -7.116710 | -47.432353 | 0.630634 | -2.069900 | 11.236675 |
| 1988-04-01 | -931.343278 | 20.518855 | 43.909236 | -7.460224 | 1.975770 | -20.622528 |
| 1988-07-01 | 1027.364925 | -9.676417 | -48.480190 | 11.492351 | -1.341915 | 16.877336 |
| 1988-10-01 | -971.410795 | 8.367490 | 50.847173 | -11.245886 | 0.822961 | -1.430279 |
| 1989-01-01 | -171.311232 | 2.933600 | -21.442960 | 8.236130 | -2.848599 | -5.439209 |
| 1989-04-01 | 1875.299522 | -4.390903 | -27.394012 | -3.684045 | 3.278405 | 25.470502 |
| 1989-07-01 | -3443.879553 | 3.708056 | 82.614658 | -2.101355 | -1.304433 | -19.856987 |

| date | gdfcf_2d | gdfce_2d |
|---|---|---|
| 1984-10-01 | -3.542227 | -10.426202 |
| 1985-01-01 | 6.994432 | 10.242797 |
| 1985-04-01 | -6.051656 | -11.254760 |
| 1985-07-01 | 0.275311 | 6.465466 |
| 1985-10-01 | 7.823499 | -5.560102 |
| 1986-01-01 | -9.235718 | 7.659798 |
| 1986-04-01 | 3.883849 | 9.996336 |
| 1986-07-01 | -0.370686 | -25.651486 |
| 1986-10-01 | 3.488349 | 32.252237 |
| 1987-01-01 | -8.776201 | -20.256527 |
| 1987-04-01 | 1.221704 | -6.382542 |
| 1987-07-01 | 14.113054 | 1.903902 |
| 1987-10-01 | -24.239268 | -6.344731 |
| 1988-01-01 | 22.188391 | 15.853283 |
| 1988-04-01 | -10.434767 | -56.366669 |
| 1988-07-01 | 3.443776 | 81.781937 |
| 1988-10-01 | 1.479236 | -53.750058 |
| 1989-01-01 | -10.083819 | 2.545288 |
| 1989-04-01 | 23.904385 | 45.714004 |
| 1989-07-01 | -35.944685 | -47.065721 |

```
    def invert_transformation(df_train, df_forecast, second_diff=False):
        """Revert back the differencing to get the forecast to original
    scale."""
        df_fc = df_forecast.copy()
        columns = df_train.columns
        for col in columns:
```

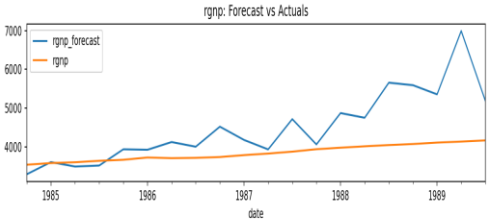
```
            # Roll back 2nd Diff
            if second_diff:
                df_fc[str(col)+'_1d'] = (df_train[col].iloc[-1]-
        df_train[col].iloc[-2]) + df_fc[str(col)+'_2d'].cumsum()
            # Roll back 1st Diff
            df_fc[str(col)+'_forecast'] = df_train[col].iloc[-1] +
        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']]
```

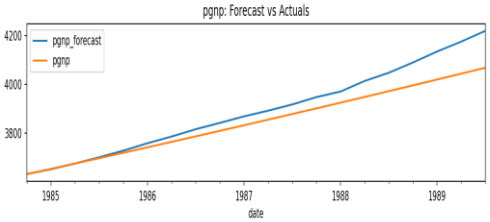
| date | rgnp_forecast | pgnp_forecast | ulc_forecast | gdfco_forecast |
|---|---|---|---|---|
| 1984-10-01 | 3288.976593 | 3630.694179 | 166.635249 | 109.409966 |
| 1985-01-01 | 3602.845646 | 3650.219233 | 156.267944 | 108.171809 |
| 1985-04-01 | 3488.130634 | 3673.931152 | 159.526282 | 105.686854 |
| 1985-07-01 | 3514.098356 | 3699.548028 | 153.739629 | 106.061412 |
| 1985-10-01 | 3933.378192 | 3726.910352 | 142.618342 | 102.634908 |
| 1986-01-01 | 3921.227502 | 3757.366944 | 140.745343 | 101.676194 |
| 1986-04-01 | 4120.538828 | 3784.952889 | 133.226968 | 102.421660 |
| 1986-07-01 | 3999.994889 | 3815.155058 | 138.263313 | 99.340687 |
| 1986-10-01 | 4517.980314 | 3841.167750 | 121.298371 | 99.961457 |
| 1987-01-01 | 4175.272606 | 3868.050449 | 134.635933 | 97.786532 |
| 1987-04-01 | 3930.533123 | 3891.451511 | 127.521705 | 95.468977 |
| 1987-07-01 | 4713.197601 | 3917.182171 | 107.092201 | 89.815914 |
| 1987-10-01 | 4058.377340 | 3947.331521 | 117.416486 | 85.428841 |
| 1988-01-01 | 4868.594908 | 3970.364160 | 80.308418 | 81.672402 |
| 1988-04-01 | 4747.469197 | 4013.915655 | 87.109586 | 70.455738 |
| 1988-07-01 | 5653.708412 | 4047.790732 | 45.430565 | 70.731426 |
| 1988-10-01 | 5588.536832 | 4090.033298 | 54.598717 | 59.761227 |
| 1989-01-01 | 5352.054019 | 4135.209465 | 42.323908 | 57.027159 |
| 1989-04-01 | 6990.870729 | 4175.994729 | 2.655088 | 50.609045 |
| 1989-07-01 | 5185.807885 | 4220.488049 | 45.600926 | 42.089575 |

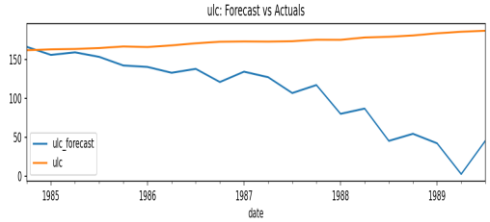
| date | gdf_forecast | gdfim_forecast | gdfcf_forecast | gdfce_forecast |
|---|---|---|---|---|
| 1984-10-01 | 108.368504 | 92.700299 | 104.157773 | 91.273798 |
| 1985-01-01 | 108.065502 | 90.460017 | 108.609978 | 90.590394 |
| 1985-04-01 | 108.067869 | 83.534504 | 107.010527 | 78.652230 |
| 1985-07-01 | 106.838890 | 76.394488 | 105.686387 | 73.179532 |
| 1985-10-01 | 106.214706 | 71.777514 | 112.185746 | 62.146732 |
| 1986-01-01 | 105.521991 | 66.537204 | 109.449386 | 58.773730 |
| 1986-04-01 | 104.411275 | 64.982516 | 110.596876 | 65.397065 |
| 1986-07-01 | 103.519600 | 64.045438 | 111.373680 | 46.368913 |
| 1986-10-01 | 102.830822 | 70.051402 | 115.638832 | 59.592999 |
| 1987-01-01 | 102.209888 | 69.783669 | 111.127784 | 52.560557 |

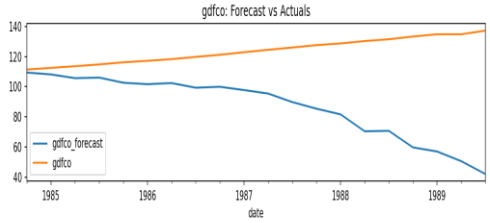
| | | | | |
|---|---|---|---|---|
| 1987-04-01 | 98.605501 | 68.529063 | 107.838440 | 39.145574 |
| 1987-07-01 | 97.537644 | 67.465571 | 118.662150 | 27.634492 |
| 1987-10-01 | 94.795332 | 52.131505 | 105.246592 | 9.778680 |
| 1988-01-01 | 89.983119 | 48.034115 | 114.019425 | 7.776150 |
| 1988-04-01 | 87.146676 | 23.314196 | 112.357491 | -50.593049 |
| 1988-07-01 | 82.968319 | 15.471613 | 114.139334 | -27.180311 |
| 1988-10-01 | 79.612923 | 6.198752 | 117.400412 | -57.517632 |
| 1989-01-01 | 73.408927 | -8.513318 | 110.577671 | -85.309664 |
| 1989-04-01 | 70.483337 | 2.245113 | 127.659314 | -67.387692 |
| 1989-07-01 | 66.253314 | -6.853442 | 108.796272 | -96.531441 |

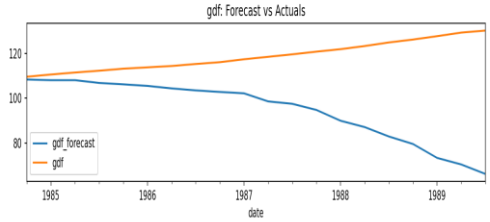
```
fig, axes = plt.subplots(nrows=len(df.columns), ncols=1, dpi=150,
figsize=(10,20))

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

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

    df_test[col][-TEST_SIZE:].plot(legend=True, ax=ax);

    ax.set_title(col + ": Forecast vs Actuals")


plt.tight_layout();
```

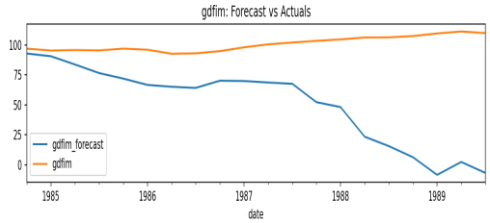

rgnp: Forecast vs Actuals
rgnp_forecast
rgnp
7000
6000
5000
4000
1985
1986
1987
1988
1989
date

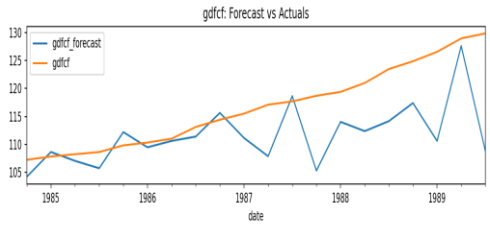

pgnp: Forecast vs Actuals
pgnp_forecast
pgnp
4200
4000
3800
1985
1986
1987
1988
1989
date

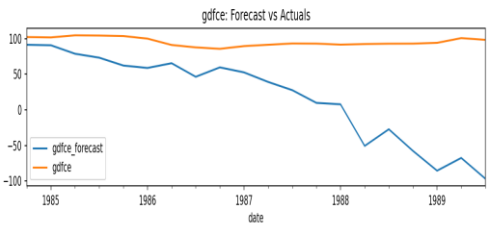

ulc: Forecast vs Actuals
ulc_forecast
ulc
150
100
50
0
1985
1986
1987
1988
1989
date


gdfco: Forecast vs Actuals
gdfco_forecast
gdfco
140
120
100
80
60
40
1985
1986
1987
1988
1989
date


gdf: Forecast vs Actuals
gdf_forecast
gdf
120
100
80
1985
1986
1987
1988
1989
date


gdfim: Forecast vs Actuals
gdfim_forecast
gdfim
100
75
50
25
0
1985
1986
1987
1988
1989
date


gdfcf: Forecast vs Actuals
gdfcf_forecast
gdfcf
130
125
120
115
110
105
1985
1986
1987
1988
1989
date


gdfce: Forecast vs Actuals
gdfce_forecast
gdfce
100
50
0
−50
−100
1985
1986
1987
1988
1989
date

В дополнение мы можем оценить результаты по некоторым метрикам

```
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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')
```

```
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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')

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, '\t : ', round(v,4), end='\t')
```

```
Forecast Accuracy of: rgnp
mape    :  0.1735  rmse    :  973.9519      corr    :  0.8749  minmax       :
0.1336
Forecast Accuracy of: pgnp
mape    :  0.0121  rmse    :  64.8452 corr    :  0.9964  minmax       :  0.0118
Forecast Accuracy of: ulc
mape    :  0.3694  rmse    :  84.6745 corr    :  -0.9424 minmax       :  0.3693
Forecast Accuracy of: gdfco
mape    :  0.286   rmse    :  46.5543 corr    :  -0.9492 minmax       :  0.286
Forecast Accuracy of: gdf
mape    :  0.194   rmse    :  30.9204 corr    :  -0.9757 minmax       :  0.194
Forecast Accuracy of: gdfim
mape    :  0.4762  rmse    :  62.3369 corr    :  -0.8948 minmax       :  0.4762
Forecast Accuracy of: gdfcf
mape    :  0.0462  rmse    :  7.9659  corr    :  0.5428  minmax       :  0.0461
Forecast Accuracy of: gdfce
mape    :  0.8072  rmse    :  96.5182 corr    :  0.1955  minmax       :  0.8072
```

### Упражнение 3

1. Как можно заметить, сложно найти лучшую модель для всего временного ряда, о собенно на относительно большом горизонте. Задача: изменить порядок VAR и н айти наилучший для тестового набора данных – для всех составляющих многоме рного ряда.

2. Попробуйте построить модель ARMA для отдельных составляющих временного ряда, где точность оказалась наименьшей.