Использование моделей АРСС

Использование моделей АРСС для предсказания и анализа временных рядов. Библиотеки sktime, statsmodels, pmdarima. Выбор параметров для модели ARIMA. Тесты на стационарность. Автоматические методы подбора параметров. Анализ остатков. Особенности выбора параметров для модели SARIMA. Использование экзогенных факторов – модель SARIMAX.

Импорт библиотек и данных.

Один из методов, доступных в Python для моделирования и прогнозирования временных рядов, известен как SARIMAX, что означает сезонное авторегрессионное интегрированное скользящие средние с экзогенными регрессорами.

Классический подход к адаптации модели ARIMA - следовать методологии Бокса-Дженкинса.

- Идентификация модели: используйте графический метод и метод сводной статистики для определения тренда и сезонности, чтобы получить представление о порядке производной (d) и порядках модели (p – порядок авторегрессии и q – порядок скользящего среднего).
- Оценка модели: оценка коэффициентов регрессионной модели.
- Диагностика модели максимального правдоподобия: используйте графический метод и статистические тесты остаточных ошибок, чтобы определить особенности данных, не охваченной моделью.

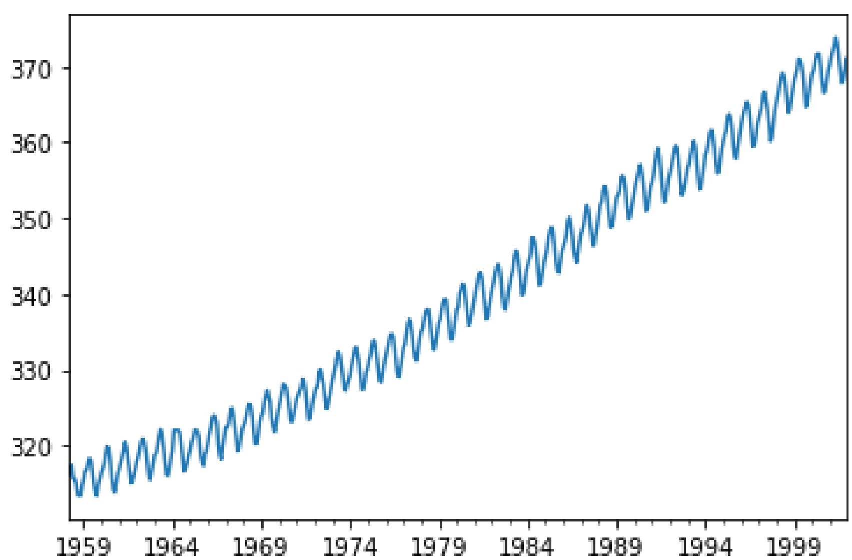
```
!pip install -U statsmodels
!pip install -U pmdarima

import warnings
import itertools
import pandas as pd
import numpy as np

import statsmodels.api as sm

from statsmodels.tsa.stattools import adfuller

import matplotlib.pyplot as plt
#

from sktime.forecasting.model_selection import temporal_train_test_split

from sktime.utils.plotting import plot_series

from sklearn.metrics import mean_squared_error

import pmdarima as pm
```

Мы будем работать с набором данных под названием «Атмосферные выбросы CO2 из непрерывных проб воздуха в обсерватории Мауна-Лоа, Гавайи, США», который собирал замеры выбросов CO2 с марта 1958 года по декабрь 2001 года..

```
dataset = sm.datasets.co2.load_pandas()
y_ = dataset.data
y_.head()

              co2
1958-03-29  316.1
1958-04-05  317.3
1958-04-12  317.6
1958-04-19  317.5
1958-04-26  316.4
```

Для начала произведем небольшую предварительную обработку данных.

```
# The 'MS' string groups the by start of the month
y = y_['co2'].resample('MS').mean()

# The term bfill means that we use the value before filling in missing val
ues
y = y.fillna(y.bfill())

y.head()

1958-03-01    316.100000
1958-04-01    317.200000
1958-05-01    317.433333
1958-06-01    315.625000
1958-07-01    315.625000
Freq: MS, Name: co2, dtype: float64

y.plot();
```



Когда мы наносим данные на график, появляются некоторые различимые закономерности. Временной ряды имеют очевидную сезонность, а также общую тенденцию к росту.

```
y.describe()
```

```
count    526.000000
mean     339.624826
std       17.110954
min      313.400000
25%      324.025000
50%      337.912500
75%      354.537500
max      373.800000
Name: co2, dtype: float64
```

Исследование модели ARIMA

Один из наиболее распространенных методов, используемых при прогнозировании временных рядов, известен как модель ARIMA, что означает AutoregRessive Integrated Moving Average. Существует три различных параметра (порядка) с целыми значениями (p, d, q), которые используются для параметризации моделей ARIMA. По этой причине модели ARIMA обозначаются обозначением ARIMA (p, d, q):

- p - авторегрессивная часть модели. Этот параметр позволяет учесть влияние прошлых значений на текущее для модели. Прошлые значения здесь называются запаздывающими наблюдениями (также известными как «запаздывание» или «лаг»). Интуитивно это похоже на утверждение, что завтра, вероятно, будет тепло, если в последние 3 дня было тепло. Другими словами, здесь мы можем сказать, что наше текущее значение температуры зависит от последних трех значений.

- d – интегрирование модели. Этот параметр включает в себя степень различия лагов (то есть количество прошлых временных точек, которые нужно вычесть из текущего значения), чтобы сделать временной ряд стационарным (чтобы исключить часть тренда). Интуитивно это было бы похоже на утверждение о том, что, вероятно, будет одно и то же повышение температуры каждый день (или одно и то же ускорение для второй производной и т.д.).

- q - скользящая средняя часть модели. Этот параметр позволяет представить остаточную часть (шум, ошибку) модели как линейную комбинацию остаточных значений, наблюдаемых в предыдущие моменты времени.

Ручной выбор параметров модели ARIMA

Для начала рассмотрим порядок дифференцирования для тостижения стационарности. Как правило, это 1-3 порядок, реже - больше.

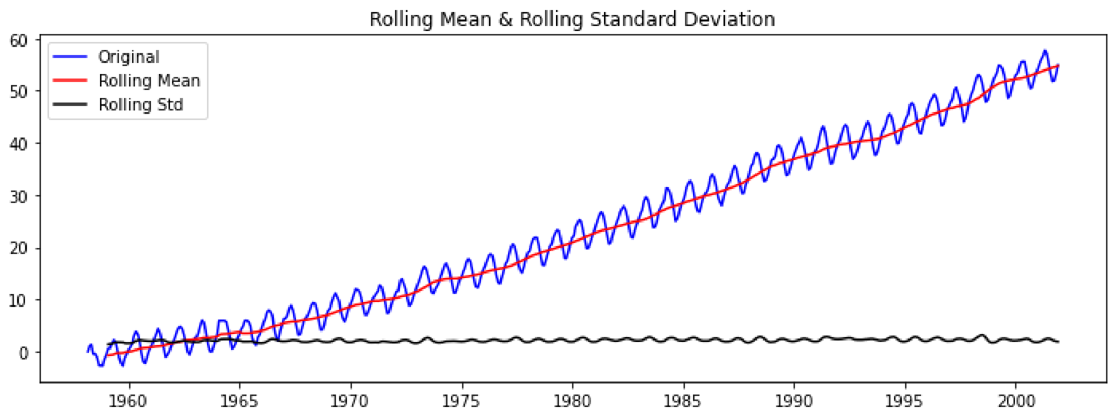
Для проверки стационарности здесь мы будем использовать два метода:

- Скользящая статистика: построение скользящего среднего и скользящего стандартного отклонения. Идея этого метода в том, что временные ряды являются стационарными, если они остаются неизменными во времени.

- Расширенный тест Дики-Фуллера: временной ряд считается стационарным, если значение p низкое (в соответствии с нулевой гипотезой), а критические значения с доверительными интервалами 1%, 5%, 10% максимально близки к статистике ADF.

Скользящая статистика визуально показывает нестационарность среднего значения. А также мы видим уменьшение дисперсии.

```
rolling_mean = y.rolling(window = 12).mean()
rolling_std  = y.rolling(window = 12).std()

plt.figure(figsize=(12,4))

plt.plot(y-y[0], color = 'blue', label = 'Original')
plt.plot(rolling_mean-y[0], color = 'red', label = 'Rolling Mean')
plt.plot(rolling_std, color = 'black', label = 'Rolling Std')

plt.legend(loc = 'best')
plt.title('Rolling Mean & Rolling Standard Deviation')
plt.show()
```



Тест ADF также показывает, что статистика ADF далека от критических значений, а значение p превышает пороговое значение (0,05). Таким образом, можно сделать вывод, что временной ряд не является стационарным.

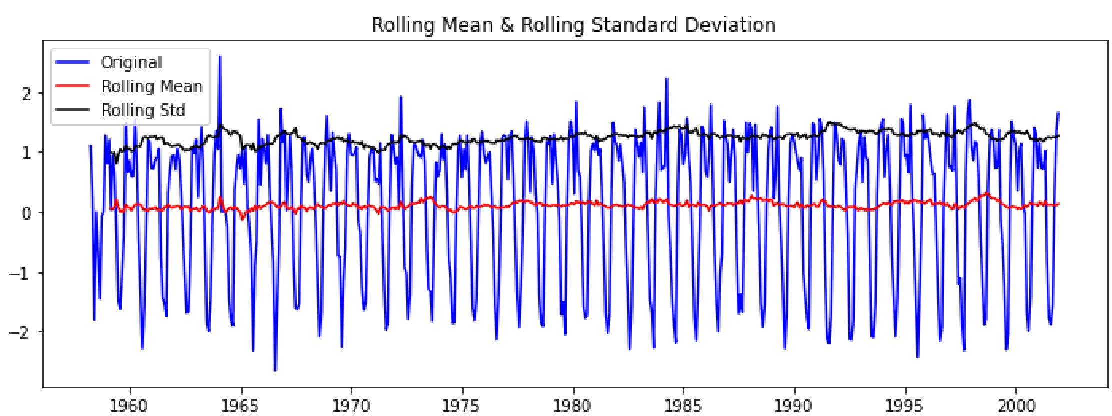
```
result = adfuller(y)

print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: 2.359809953995333
p-value: 0.9989901230798025
Critical Values:
      1%: -3.4432119442564324
```

```
5%: -2.8672126791646955
10%: -2.569791324979607
```

Теперь давайте посмотрим на 1ю производную

```
y_diff = y.diff(1)
# for fill obtained first NaN Value with next
y_diff = y_diff.dropna()

rolling_mean = y_diff.rolling(window = 12).mean()
rolling_std  = y_diff.rolling(window = 12).std()

plt.figure(figsize=(12,4))

plt.plot(y_diff, color = 'blue', label = 'Original')
plt.plot(rolling_mean, color = 'red', label = 'Rolling Mean')
plt.plot(rolling_std, color = 'black', label = 'Rolling Std')

plt.legend(loc = 'best')
plt.title('Rolling Mean & Rolling Standard Deviation')
plt.show()
```



Здесь и ниже мы видим, что наши данные теперь удовлетворяют стационарным критериям.

```
y_diff.head()

1958-04-01    1.100000
1958-05-01    0.233333
1958-06-01   -1.808333
1958-07-01    0.000000
1958-08-01   -0.675000
Freq: MS, Name: co2, dtype: float64

result = adfuller(y_diff)


print('ADF Statistic: {}'.format(result[0]))
print('p-value: {}'.format(result[1]))
print('Critical Values:')
```

```
for key, value in result[4].items():
    print('\t{}: {}'.format(key, value))
```

```
ADF Statistic: -5.063202630318491
p-value: 1.6614851317686715e-05
Critical Values:
	1%: -3.4432119442564324
	5%: -2.8672126791646955
	10%: -2.569791324979607
```

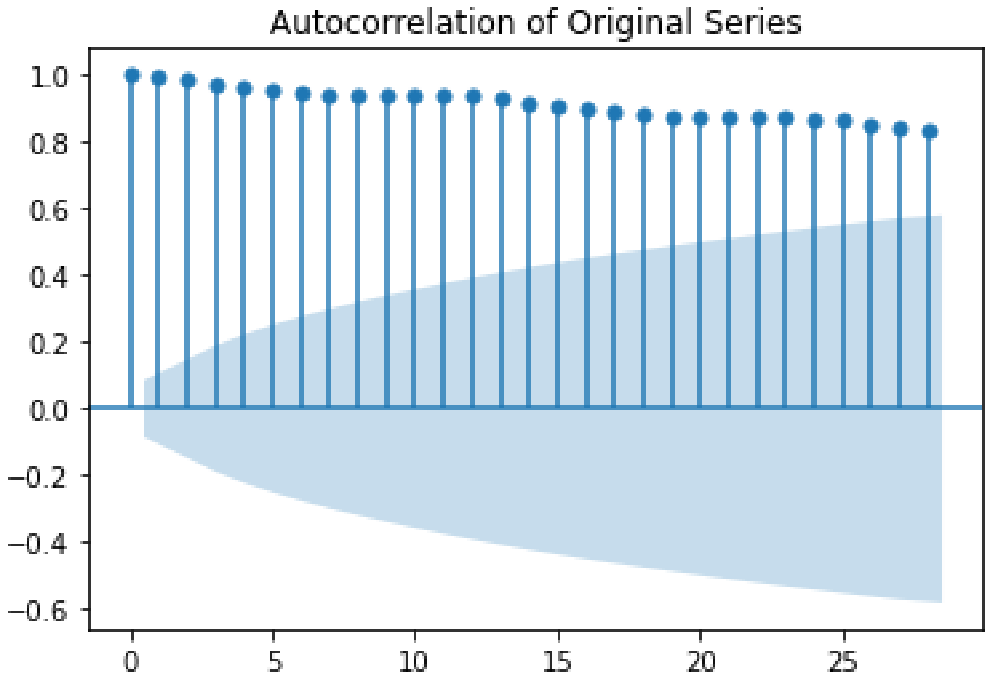
```
y_diff.head()
```

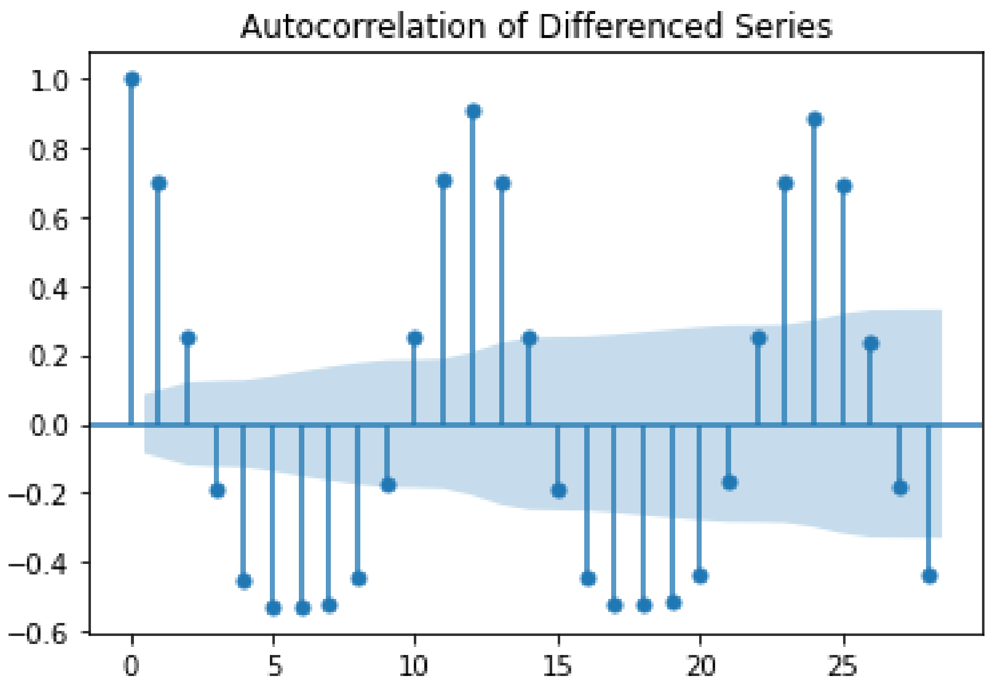
```
1958-04-01    1.100000
1958-05-01    0.233333
1958-06-01   -1.808333
1958-07-01    0.000000
1958-08-01   -0.675000
Freq: MS, Name: co2, dtype: float64
```

Примечание. Помимо ADF существует множество тестов, среди которых также полезно проверить

- Анализ ACF (АКФ) , в котором для нестационарного процесса вы увидите медленное уменьшение значений АКФ, и резкий спад значений автокорреляции для стационарного случая.
- Тест Квятковского – Филлипса – Шмидта – Шина (KPSS), который дает значения, отличающиеся от ADF в случае детерминированного тренда с точками перегиба.

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

# Original Series
plot_acf(y[:], title='Autocorrelation of Original Series');plt.show()

# Usual Differencing
plot_acf(y_diff[:], title='Autocorrelation of Differenced Series');plt.sho
w()

plt.show();
```






```
from statsmodels.tsa.stattools import kpss

def kpss_test(series, **kw):
    statistic, p_value, n_lags, critical_values = kpss(series, **kw)
    # Format Output
    print(f'KPSS Statistic: {statistic}')
    print(f'p-value: {p_value}')
    print(f'num lags: {n_lags}')
    print('Critial Values:')
    for key, value in critical_values.items():
```

```
        print(f'    {key} : {value}')
    print(f'Result: The series is {"not " if p_value < 0.05 else ""}statio
nary')

kpss_test(y_diff)
```

```
KPSS Statistic: 0.07042168811681968
p-value: 0.1
num lags: 19
Critial Values:
   10% : 0.347
   5% : 0.463
   2.5% : 0.574
   1% : 0.739
Result: The series is stationary
```

Примечание. Если ваш ряд немного недодифференцирован, то необходимо будет добавить один или несколького дополнительных слогаемых в авторегрессионную часть (повысить порядок) обычно это компенсирует. Аналогичным образом, если разница немного выше, попробуйте добавить дополнительный член к скользящиму средниму.

После выбора порядка интеграции необходимо выбрать порядки AR и MA частей. Для этого могут быть даны следующие рекомендации по этому поводу

- Чтобы оценить порядок авторегрессии (порядок AR), проанализируйте график частичной автокорреляции (PACF). Как правило, график состоит из доверительных интервалов, которые отображаются в виде конуса. По умолчанию установлен доверительный интервал 95%, что предполагает, что значения корреляции за пределами этого интервала, скорее всего, являются корреляцией, а не статистической случайностью.

- После оценки AR мы можем сделать первоначальное предположение о порядке скользящего среднего (порядок MA). Для этого нужно будем использовать график автокорреляции (ACF). Число ненулевых членов ACF сообщает, сколько членов MA необходимо для устранения любой автокорреляции в стационарном ряду.

```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

plot_pacf(y_diff);
plt.show()
```

```
C:\Users\Администратор\AppData\Roaming\Python\Python37\site-packages
\statsmodels\regression\linear_model.py:1434: RuntimeWarning: invali
d value encountered in sqrt
  return rho, np.sqrt(sigmasq)
```

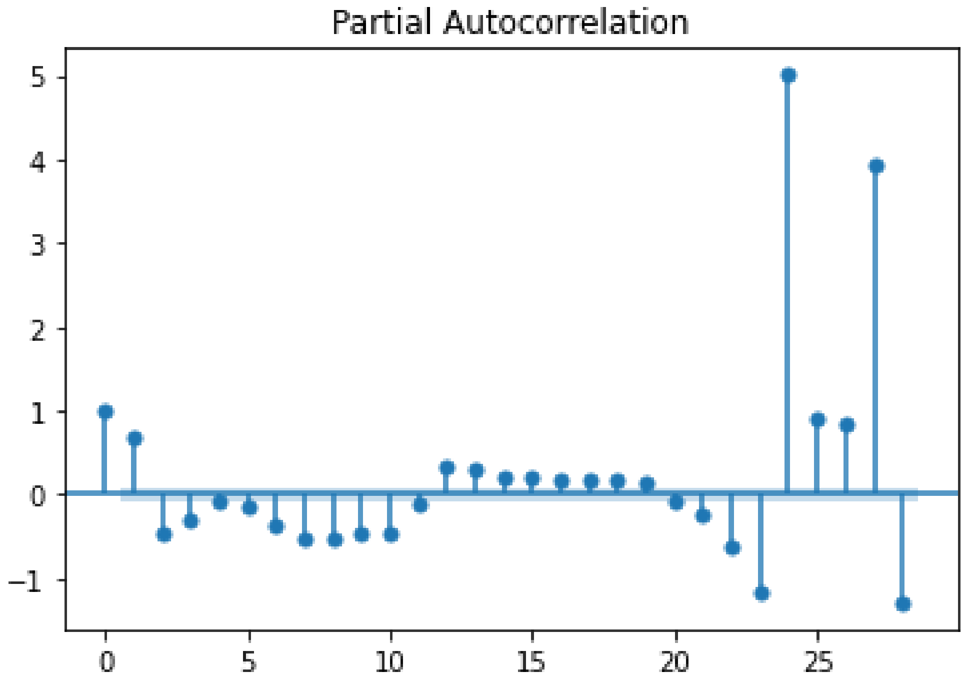


График PACF показывает, что у нас есть как минимум модель AR 1-го порядка с некоторыми дополнительными эффектами, такими как сезонность или не стационарность.

Примечание. График начинается с лага- 0, поэтому мы не можем его учитывать.

```
plot_acf(y_diff);
plt.show()
```

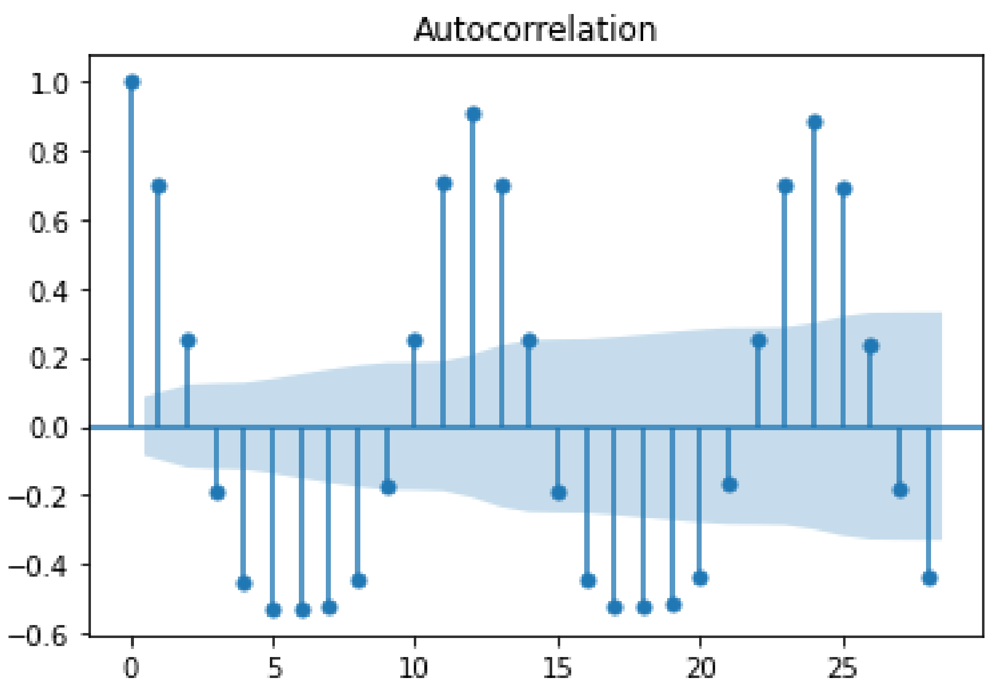


График АКФ показывает зависимость как минимум 2-го порядка, а также наличие некоторой сезонности.

Тестирование выбранной модели

Давайте протестируем выбранную модель ARMA(p=1,d=1,q=2).

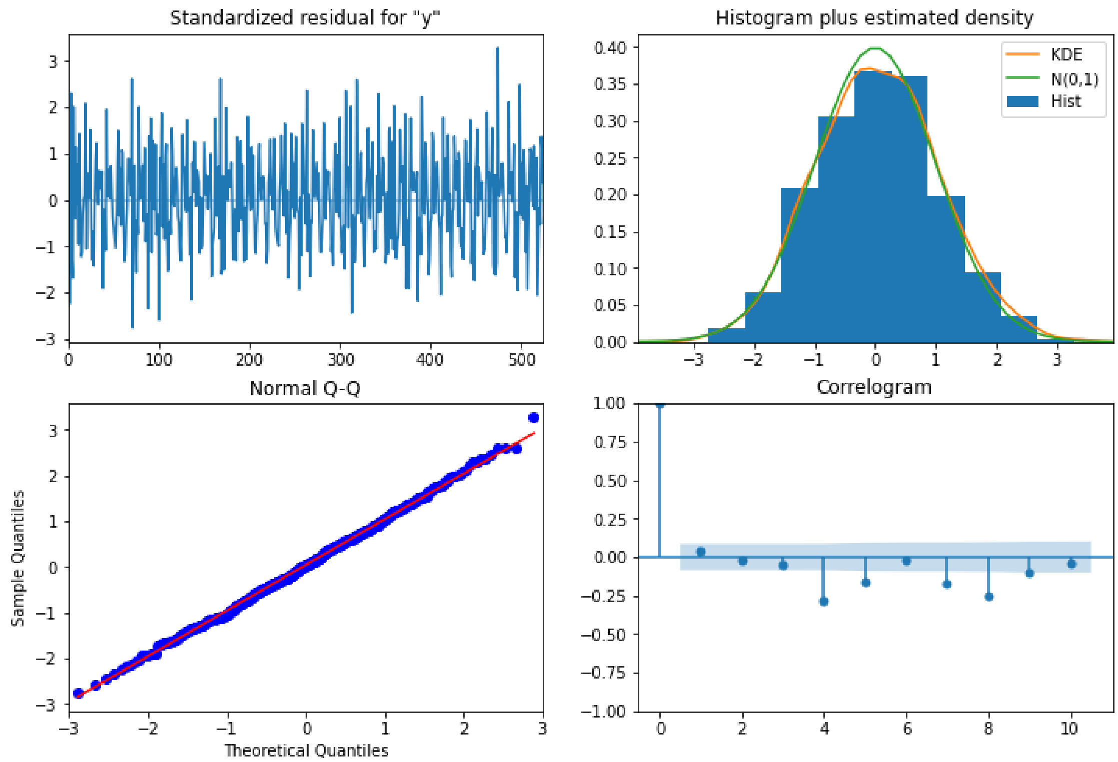
```
from statsmodels.tsa.arima.model import ARIMA

# 1,1,2 ARIMA Model
model = ARIMA(y.values, order=(1,1,2))
model_fit = model.fit()
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  526
Model:                 ARIMA(1, 1, 2)   Log Likelihood                -607.411
Date:                Mon, 03 May 2021   AIC                           1222.822
Time:                        11:42:34   BIC                           1239.876
Sample:                             0   HQIC                          1229.500
                                - 526
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4018      0.073      5.529      0.000       0.259       0.544
ma.L1          0.5221      0.072      7.241      0.000       0.381       0.663
ma.L2          0.3636      0.057      6.379      0.000       0.252       0.475
sigma2         0.5909      0.041     14.295      0.000       0.510       0.672
================================================================================
Ljung-Box (L1) (Q):                   0.74   Jarque-Bera (JB):                 1
.15
Prob(Q):                              0.39   Prob(JB):                         0
.56
Heteroskedasticity (H):               0.99   Skew:                             0
.09
Prob(H) (two-sided):                  0.96   Kurtosis:                         2
.85
================================================================================
```

Выведенное описание модели раскрывает много информации. В первой таблице представлена общая информация, включая критерии качества (AIC, BIC и HQIC). Таблица посередине - это таблица коэффициентов, где значения под «coef» - это веса соответствующих слогаемых. Значение sigma2 – это RSS ошибка модели. В последней таблице представлены результаты различных статистических тестов для полученных остатков.

Помимо табличного представления, мы можем проводить диагностику остатков графическим способом.

```
model_fit.plot_diagnostics(figsize=(12,8));
```



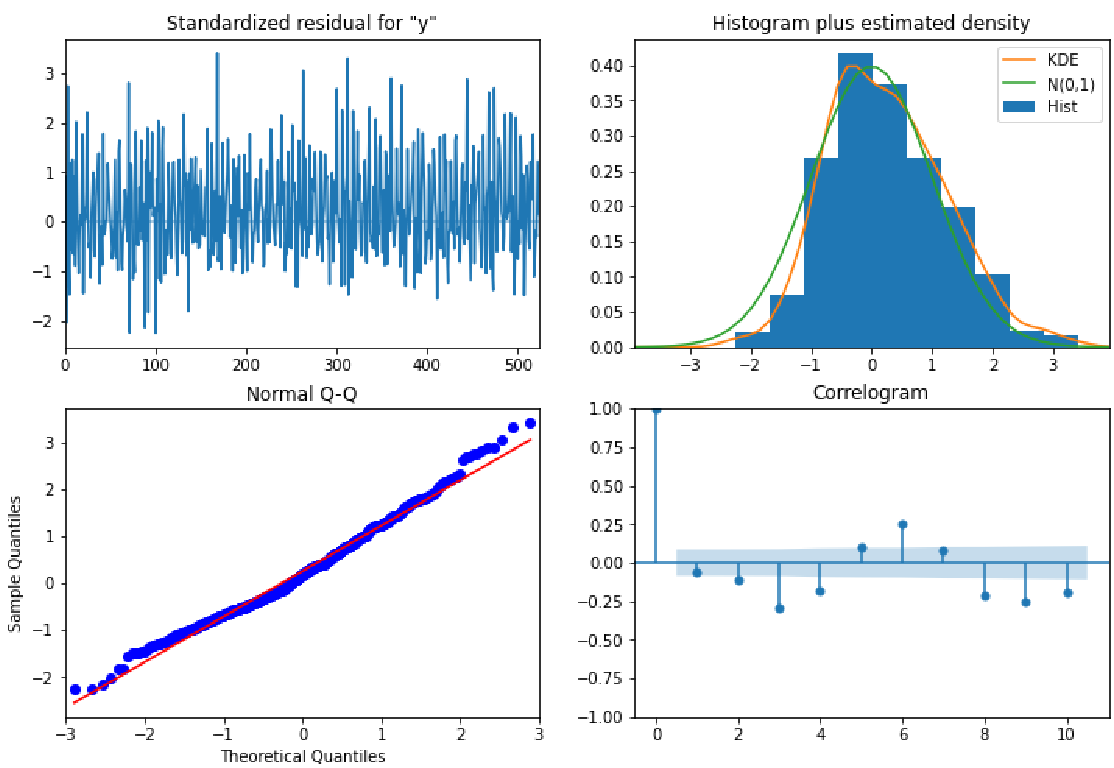
На графиках выше мы видим: остаточные ошибки колеблются около нулевого среднего и имеют равномерную дисперсию (верхний левый график). Остаток имеет почти нормальное распределение (верхний правый график). График Q-Q также показывает почти нормальное распределение (внизу слева, в идеале все точки должны точно совпадать с красной линией). Однако на графике ACF (коррелограмме) мы можем заметить некоторые выбросы, превышающие уровень доверительного интервала (внизу справа, любая автокорреляция будет означать, что существует некоторая закономерность в остаточных ошибках, которые не объясняются в модели).

Проведенный анализ показывает, что мы можем улучшить нашу модель. В качестве первого предположения мы можем попытаться увеличить AR-порядок модели.

```
from statsmodels.tsa.arima.model import ARIMA

# 1,1,2 ARIMA Model
model = ARIMA(y.values, order=(2,1,2))
model_fit = model.fit()
print(model_fit.summary())
model_fit.plot_diagnostics(figsize=(12,8));
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  526
Model:                 ARIMA(2, 1, 2)   Log Likelihood                -533.373
Date:                Mon, 03 May 2021   AIC                           1076.745
Time:                        11:42:37   BIC                           1098.062
Sample:                             0   HQIC                          1085.092
                                - 526
Covariance Type:                  opg
```

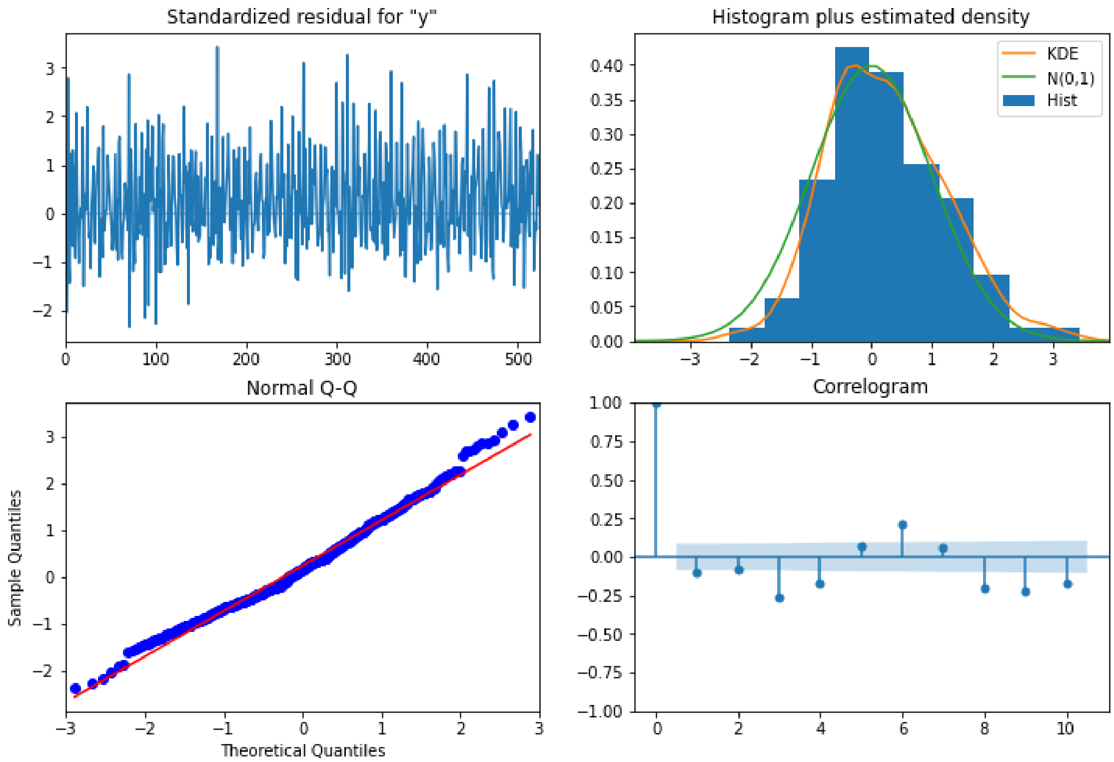
```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.5539      0.034     45.970      0.000       1.488       1.620
ar.L2         -0.8466      0.038    -22.003      0.000      -0.922      -0.771
ma.L1         -0.8716      0.059    -14.654      0.000      -0.988      -0.755
ma.L2          0.0571      0.068      0.836      0.403      -0.077       0.191
sigma2         0.4447      0.027     16.593      0.000       0.392       0.497
================================================================================
Ljung-Box (L1) (Q):                   2.10   Jarque-Bera (JB):                14
.18
Prob(Q):                              0.15   Prob(JB):                         0
.00
Heteroskedasticity (H):               0.97   Skew:                             0
.40
Prob(H) (two-sided):                  0.82   Kurtosis:                         3
.06
================================================================================
```



Как мы видим здесь, мы уменьшаем значения как критериев AIC (и BIC), так и ошибку RSS (sigma2) - это означает, что мы движемся в правильном направлении. Однако мы немного ухудшили поведение остатков . Поиск лучших параметров - сложная задача. Здесь мы также можем заметить, что у нас есть небольшое значение компоненты ma.L2, и можем попробовать его устранить.

```python
from statsmodels.tsa.arima.model import ARIMA

# 1,1,2 ARIMA Model
model = ARIMA(y.values, order=(2,1,1))
model_fit = model.fit()
print(model_fit.summary())
model_fit.plot_diagnostics(figsize=(12,8));
```

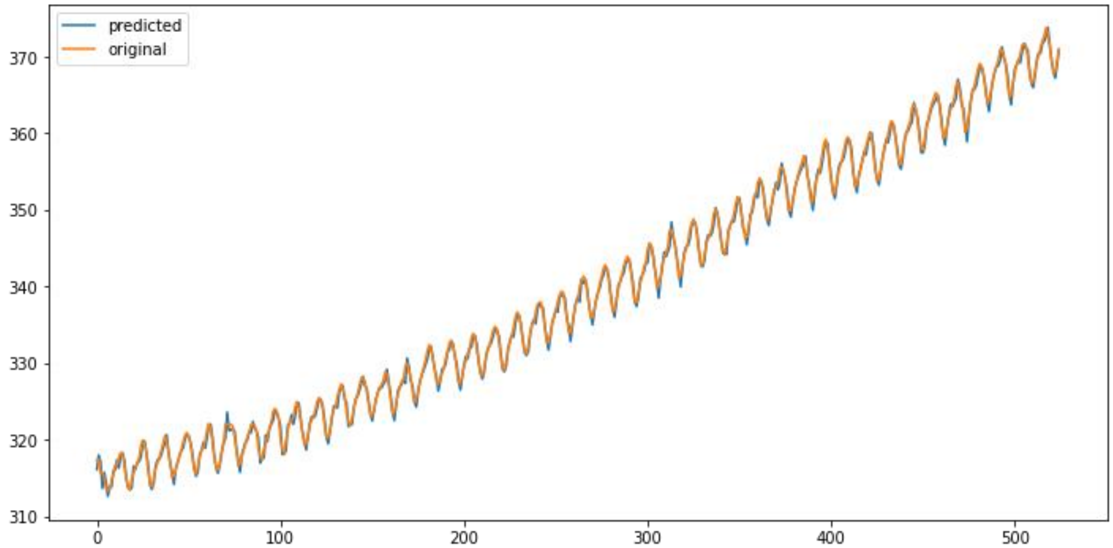
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  526
Model:                 ARIMA(2, 1, 1)   Log Likelihood                -533.989
Date:                Mon, 03 May 2021   AIC                           1075.978
Time:                        11:42:40   BIC                           1093.032
Sample:                             0   HQIC                          1082.656
                                - 526
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.5353      0.028     55.566      0.000       1.481       1.589
ar.L2         -0.8285      0.030    -27.249      0.000      -0.888      -0.769
ma.L1         -0.8117      0.037    -21.901      0.000      -0.884      -0.739
sigma2         0.4458      0.026     17.040      0.000       0.395       0.497
================================================================================
Ljung-Box (L1) (Q):                   5.44   Jarque-Bera (JB):                13
.62
Prob(Q):                              0.02   Prob(JB):                         0
.00
Heteroskedasticity (H):               0.94   Skew:                             0
.39
Prob(H) (two-sided):                  0.68   Kurtosis:                         3
.10
================================================================================
```



Теперь мы видим, что действительно второй член не влияет на точность предсказания данных.

Теперь мы можем построить график для подобранной модели. В следующем примере, когда вы устанавливаете dynamic = False, для прогнозирования используются запаздывающие значения в выборке. То есть модель обучается до предыдущего значения, чтобы сделать следующий прогноз. Это может привести к тому, что подогнанный прогноз и фактические данные будут выглядеть искусственно хорошими.

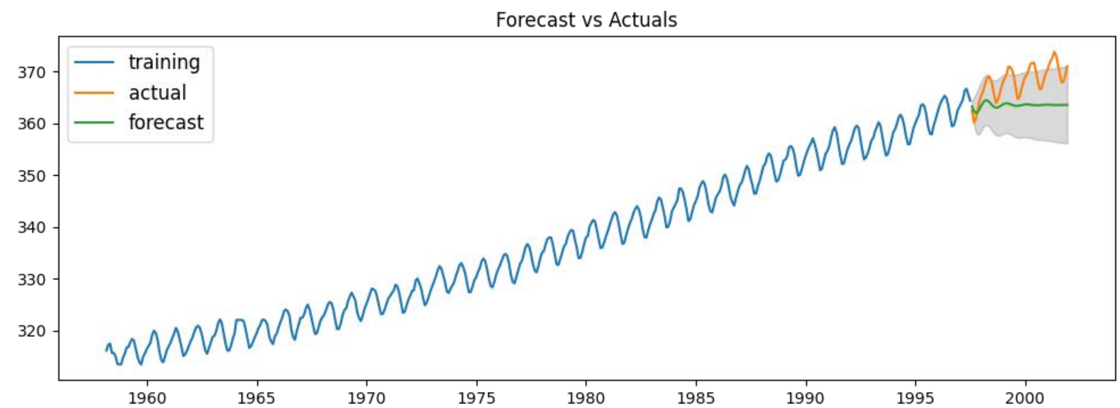
```
# Actual vs Fitted
y_hat = model_fit.predict(dynamic=False)

plt.figure(figsize=(12,6))
plt.plot(y_hat[1:], label='predicted')
plt.plot(y[1:].values, label='original')
plt.legend()
plt.show()
```



Помимо построения модели по существующим данным, мы можем проверить модель тестовых данных. Для этого мы можем разделить наши данные на две выборки - тестовая и тренировочная.

```
# Create Training and Test
train = y[:int(y.size*0.9)]
test = y[int(y.size*0.9):]

# Build Model
model = ARIMA(train, order=(2, 1, 1))
fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.size, alpha=0.05, dynamic=False)
# 95% conf

# forecast = fitted.forecast(test.size, alpha=0.05)  # 95% conf
forecast = forecast_res.predicted_mean

# Make as pandas series
fc_series = pd.Series(forecast.values, index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower co2'], index=test.
index)
upper_series = pd.Series(forecast_res.conf_int()['upper co2'], index=test.
index)


# Plot
plt.figure(figsize=(12,4), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')
```

```python
plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()
```



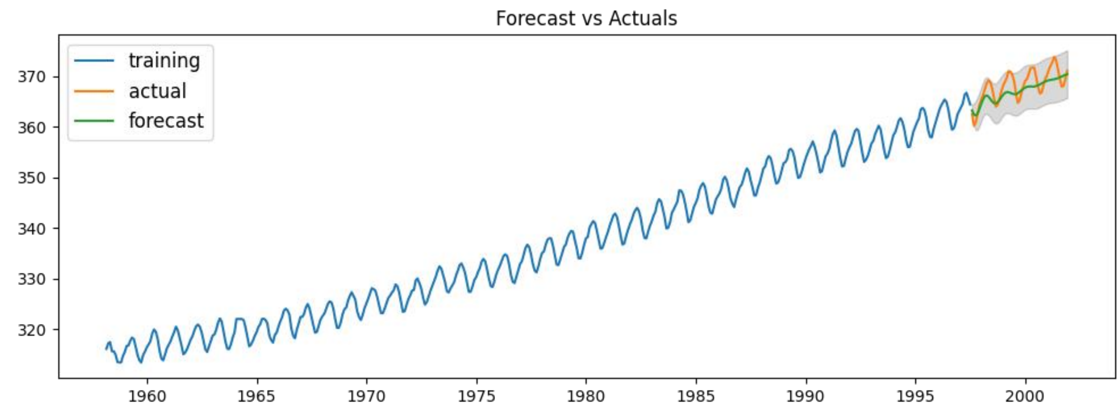
Теперь мы видим, что наша модель была переобучена. Для оценки точности нашего прогноза мы можем ввести следующие меры:

```python
# Accuracy metrics
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({'mean absoute percentage error':mape,
            'mean absoute error            ': mae,
            'mean percentage error         ': mpe,
            'root mean square              ':rmse,
            'correlation coefficient       ':corr,
            'minmax error                  ':minmax})

forecast_accuracy(fc_series.values, test.values)

{'mean absoute percentage error': 0.0037188201927048957,
 'mean absoute error            ': 1.370467806185665,
```

```
 'mean percentage error        ': -0.0036203570222564747,
 'root mean square             ': 1.448243682381261,
 'correlation coefficient      ': 0.984801545546153,
 'minmax error                 ': 0.003718716705316538}
```

Мы можем выбрать лучшие порядки модели, как показано ниже.

```
# Create Training and Test
train = y[:int(y.size*0.9)]
test = y[int(y.size*0.9):]

# Build Model
model = ARIMA(train, order=(3, 2, 2))
fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.size, alpha=0.05, dynamic=False)
# 95% conf
forecast = forecast_res.predicted_mean

# forecast = fitted.forecast(test.size, alpha=0.05)  # Alternative method

# Make as pandas series
fc_series = pd.Series(forecast.values, index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower co2'], index=test.
index)
upper_series = pd.Series(forecast_res.conf_int()['upper co2'], index=test.
index)


# Plot
plt.figure(figsize=(12,4), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.values)
```




```
{'mean absoute percentage error': 0.0050279116520248295,
 'mean absoute error            ': 1.8565906438131856,
 'mean percentage error         ': -0.0030232064973775142,
 'root mean square              ': 2.2157261786757645,
 'correlation coefficient       ': 0.7836715399134366,
 'minmax error                  ': 0.005023574563809641}
```

Сейчас мы видим, что наши метрики стали значительно лучше.

Автоматические методы выбора порядка с помощью библиотеки pmdarima

Помимо ручного выбора параметров ARIMA, мы можем использовать автоматический поиск arima с использованием библиотеки pmdarima.

```
import pmdarima as pm

model = pm.auto_arima(y,
                      start_p=1,
                      start_q=1,
                      test='adf', # use adftest to find optimal 'd'
                      max_p=10,
                      max_q=10, # maximum p and q
                      m=1,              # frequency of series
                      d=None,           # let model determine 'd'
                      seasonal=False,   # No Seasonality
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True)


model.summary()

Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=1268.116, Time=0.12 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1676.811, Time=0.01 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1328.472, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=1374.168, Time=0.07 sec
```

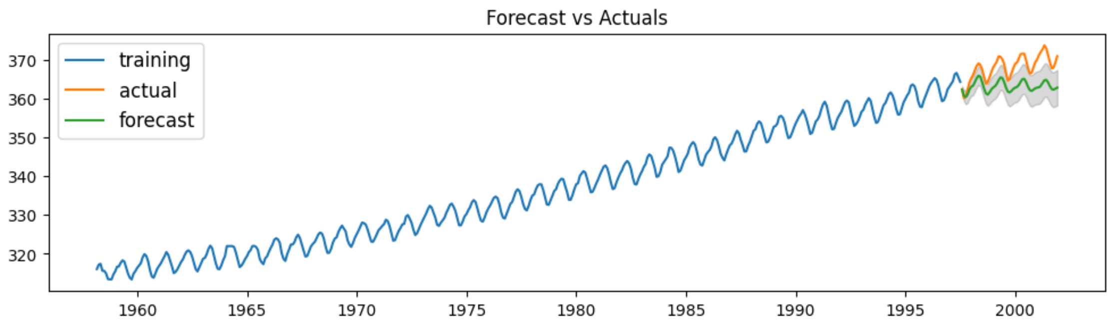
```
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1678.850, Time=0.01 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=1010.848, Time=0.26 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1205.382, Time=0.06 sec
 ARIMA(3,1,1)(0,0,0)[0] intercept   : AIC=1009.758, Time=0.53 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1150.521, Time=0.17 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=1008.632, Time=0.91 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=1149.017, Time=0.15 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=947.708, Time=1.09 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=1138.051, Time=0.18 sec
 ARIMA(6,1,1)(0,0,0)[0] intercept   : AIC=833.411, Time=1.28 sec
 ARIMA(6,1,0)(0,0,0)[0] intercept   : AIC=1060.199, Time=0.29 sec
 ARIMA(7,1,1)(0,0,0)[0] intercept   : AIC=724.613, Time=1.20 sec
 ARIMA(7,1,0)(0,0,0)[0] intercept   : AIC=905.168, Time=0.51 sec
 ARIMA(8,1,1)(0,0,0)[0] intercept   : AIC=655.728, Time=1.48 sec
 ARIMA(8,1,0)(0,0,0)[0] intercept   : AIC=inf, Time=0.88 sec
 ARIMA(9,1,1)(0,0,0)[0] intercept   : AIC=624.495, Time=1.78 sec
 ARIMA(9,1,0)(0,0,0)[0] intercept   : AIC=inf, Time=1.45 sec
 ARIMA(10,1,1)(0,0,0)[0] intercept   : AIC=590.889, Time=2.07 sec
 ARIMA(10,1,0)(0,0,0)[0] intercept   : AIC=inf, Time=1.90 sec
 ARIMA(10,1,2)(0,0,0)[0] intercept   : AIC=496.938, Time=2.33 sec
 ARIMA(9,1,2)(0,0,0)[0] intercept   : AIC=506.775, Time=1.91 sec
 ARIMA(10,1,3)(0,0,0)[0] intercept   : AIC=505.994, Time=2.32 sec
 ARIMA(9,1,3)(0,0,0)[0] intercept   : AIC=532.159, Time=2.36 sec
 ARIMA(10,1,2)(0,0,0)[0]             : AIC=693.807, Time=1.39 sec

Best model:  ARIMA(10,1,2)(0,0,0)[0] intercept
Total fit time: 26.787 seconds

<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  526
Model:              SARIMAX(10, 1, 2)   Log Likelihood                -234.469
Date:                Tue, 04 May 2021   AIC                            496.938
Time:                        14:11:30   BIC                            556.626
Sample:                             0   HQIC                           520.311
                                - 526
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.3105      0.028     11.163      0.000       0.256       0.365
ar.L1          0.7235      0.060     12.039      0.000       0.606       0.841
ar.L2         -0.8153      0.053    -15.298      0.000      -0.920      -0.711
ar.L3         -0.1545      0.060     -2.592      0.010      -0.271      -0.038
ar.L4         -0.1730      0.061     -2.850      0.004      -0.292      -0.054
ar.L5         -0.2430      0.065     -3.737      0.000      -0.371      -0.116
ar.L6         -0.2120      0.066     -3.228      0.001      -0.341      -0.083
ar.L7         -0.2672      0.063     -4.211      0.000      -0.392      -0.143
ar.L8         -0.2796      0.064     -4.349      0.000      -0.406      -0.154
ar.L9         -0.1906      0.057     -3.368      0.001      -0.301      -0.080
```

```
ar.L10        -0.2236      0.052     -4.294      0.000      -0.326      -0.122
ma.L1         -0.8896      0.047    -18.940      0.000      -0.982      -0.798
ma.L2          0.7777      0.037     21.013      0.000       0.705       0.850
sigma2         0.1406      0.008     17.405      0.000       0.125       0.156
===================================================================================
Ljung-Box (L1) (Q):                   0.95   Jarque-Bera (JB):                    7
.85
Prob(Q):                              0.33   Prob(JB):                            0
.02
Heteroskedasticity (H):               0.66   Skew:                                0
.11
Prob(H) (two-sided):                  0.01   Kurtosis:                            3
.56
===================================================================================
```

Автопоиск предлагает использовать модель ARIMA (10,1,2). Протестируем ее.

```
# Create Training and Test
train = y[:int(y.size*0.9)]
test = y[int(y.size*0.9):]

# Build Model
model = ARIMA(train, order=(10, 1, 2))
fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.size, alpha=0.05, dynamic=False)
# 95% conf

# forecast = fitted.forecast(test.size, alpha=0.05)  # 95% conf
forecast = forecast_res.predicted_mean

# Make as pandas series
fc_series = pd.Series(forecast.values, index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower co2'], index=test.
index)
upper_series = pd.Series(forecast_res.conf_int()['upper co2'], index=test.
index)


# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
```

```
                  color='k',
                  alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.values)
```

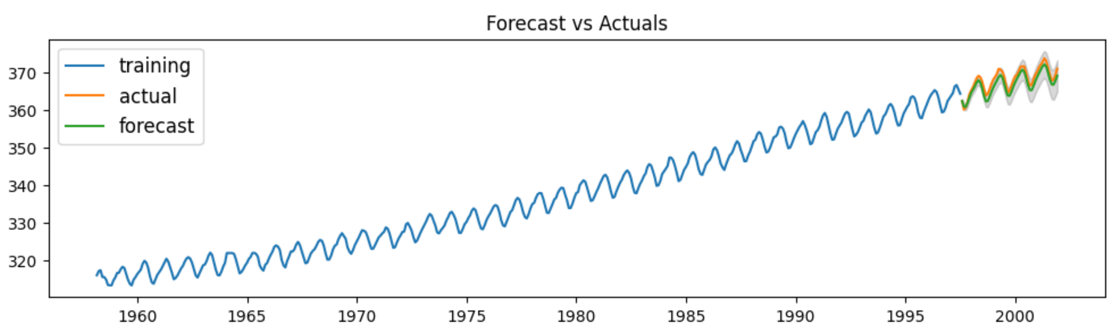



```
{'mape': 0.01330633295669916,
 'me': -4.89114134519186,
 'mae': 4.914562517264946,
 'mpe': -0.013241301241932851,
 'rmse': 5.379130185554309,
 'corr': 0.7410145431091859,
 'minmax': 0.013306277017209989}
```

Как мы видим, автопоиск не гарантирует лучших результатов в прогнозе. Это связано с отсутствием перекрестной проверки, но мы, вероятно, сможем улучшить эту модель вручную.

```
# Create Training and Test
train = y[:int(y.size*0.9)]
test = y[int(y.size*0.9):]

# Build Model
model = ARIMA(train, order=(10, 2,8))
fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.size, alpha=0.05, dynamic=False)
# 95% conf

# forecast = fitted.forecast(test.size, alpha=0.05)  # 95% conf
forecast = forecast_res.predicted_mean

# Make as pandas series
fc_series = pd.Series(forecast.values, index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower co2'], index=test.
index)
```

```
upper_series = pd.Series(forecast_res.conf_int()['upper co2'], index=test.
index)


# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.values)
```




```
{'mape': 0.0037188201927048957,
 'me': -1.3349999198951579,
 'mae': 1.370467806185665,
 'mpe': -0.0036203570222564747,
 'rmse': 1.448243682381261,
 'corr': 0.984801545546153,
 'minmax': 0.003718716705316538}
```

Упражнение 1

1. Попробуйте смоделировать процесс случайного блуждания (из работы №2) и выберите для него лучшую модель ARIMA.

2. Попробуйте смоделировать некоторый временной ряд с небольшой сезонностью, логистическим трендом, небольшим эффектом праздников и найдите для этого лучшую модель ARIMA.

3. Возьмите набор данных пассажира авиалайнера (из работы № 4) и попытайтесь найти лучшую модель для его прогнозирования.

Сезонная модель ARIMA (SARIMA)

Важность сезонной производной

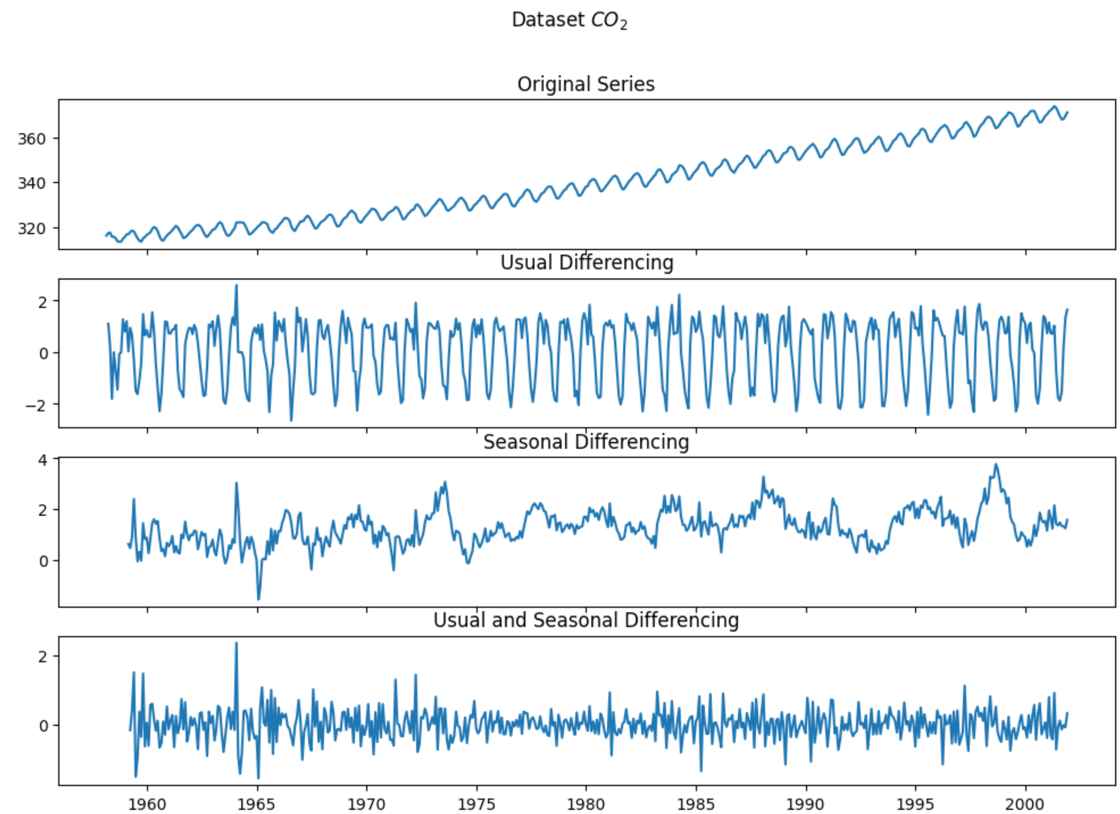
Проблема с простой моделью ARIMA в том, что она не подразумевает нестационарную сезонность. Если для временного ряда имеет место значительный эффект сезонности, тогда следует выбирать модель SARIMA. В этой модели используются сезонные производные.

Сезонная производная аналогична обычной производной, но вместо вычитания последовательных членов вы вычитаете значение из предыдущего периода сезонности.

> Примечания. При работе с сезонными эффектами мы используем сезонный ARIMA (SARIMA), который обозначается как SARIMA (p, d, q) (P, D, Q) s. Здесь (p, d, q) являются несезонными параметрами, описанными выше, а (P, D, Q) следуют тому же порядку определений, но применяются к сезонной составляющей временного ряда. Член s - это периодичность временного ряда (4 для квартальных периодов, 12 для годовых периодов и т.д.).

Для начала давайте посмотрим, как работает сезонное дифференцирование

```
SEASON = 12

# Plot
fig, axes = plt.subplots(4, 1, figsize=(12,8), dpi=100, sharex=True)

# Original Series
axes[0].plot(y[:])
axes[0].set_title('Original Series')

# Usual Differencing
axes[1].plot(y[:].diff(1))
axes[1].set_title('Usual Differencing')

# Seasinal Differencing
axes[2].plot(y[:].diff(SEASON))
axes[2].set_title('Seasonal Differencing')

# Seasinal and Usual Differencing
axes[3].plot(y[:].diff(1).diff(SEASON))
axes[3].set_title('Usual and Seasonal Differencing')

plt.suptitle('Dataset $CO_2$', fontsize=12)
plt.show()
```

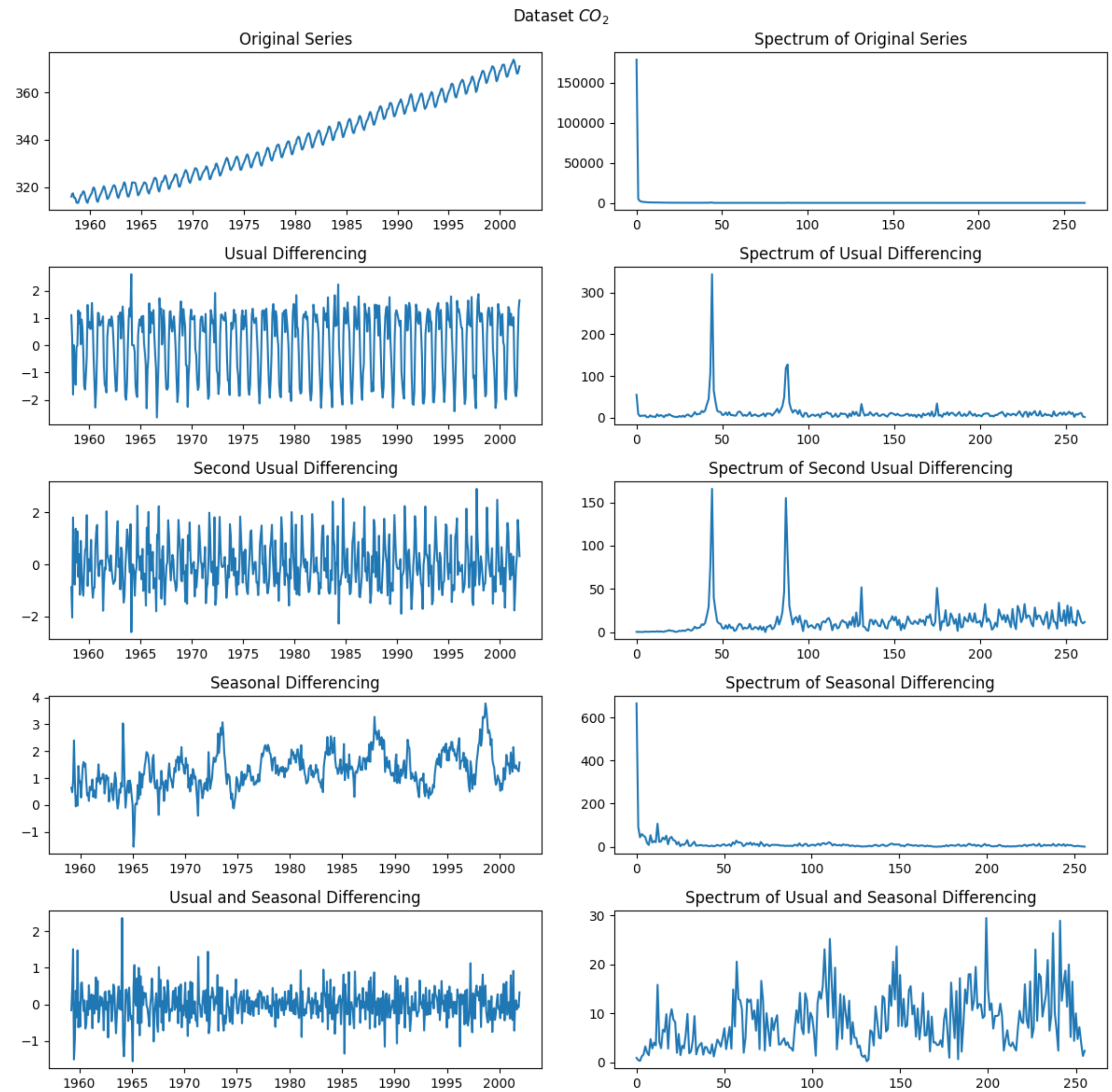


Как мы видим, сезонная разница может помочь сделать данные более стационарными.

Примечание. Как мы можем видеть на графике обычной разницы, у нас есть как минимум две сезонные составляющие с разными периодами, но, взяв одну сезонную разницу, мы исключаем почти все сезонные влияния. Посмотрим на спектр.

```
SEASON = 12

def afft(x):
    x_np  = x.dropna().values
    return np.abs(np.fft.fft(x_np))[:x_np.size//2] # the spectrum is mirrored relative to the middle point




# Plot
fig, axes = plt.subplots(5, 2, figsize=(12,12), dpi=100)

# Original Series
axes[0,0].plot(y[:])
axes[0,0].set_title('Original Series')
axes[0,1].plot(afft(y))
axes[0,1].set_title('Spectrum of Original Series')

# Usual Differencing
axes[1,0].plot(y[:].diff(1))
```

```
axes[1,0].set_title('Usual Differencing')
axes[1,1].plot(afft(y.diff(1)))
axes[1,1].set_title('Spectrum of Usual Differencing')

# Usual Differencing
axes[2,0].plot(y[:].diff(1).diff(1))
axes[2,0].set_title('Second Usual Differencing')
axes[2,1].plot(afft(y.diff(1).diff(1)))
axes[2,1].set_title('Spectrum of Second Usual Differencing')

# Seasinal Differencing
axes[3,0].plot(y[:].diff(SEASON))
axes[3,0].set_title('Seasonal Differencing')
axes[3,1].plot(afft(y.diff(SEASON)))
axes[3,1].set_title('Spectrum of Seasonal Differencing')


# Seasinal and Usual Differencing
axes[4,0].plot(y[:].diff(1).diff(SEASON))
axes[4,0].set_title('Usual and Seasonal Differencing')
axes[4,1].plot(afft(y.diff(1).diff(SEASON)))
axes[4,1].set_title('Spectrum of Usual and Seasonal Differencing')

plt.suptitle('Dataset $CO_2$', fontsize=12)

fig.tight_layout()

plt.show()
```



Как мы видим на графиках выше, спектр обычной разности содержит по крайней мере 4 компоненты (пика), но все с одним и тем же шагом (то есть один с периодом 12, следующий с периодом 24 и так далее). Благодаря этому используя разность с периодом 12 мы исключаем все сезонные составляющие на нижнем графике.

Также необходимо отметить, что тренд - это самая низкочастотная часть (см. Начало спектра). Таким образом, мы практически исключаем влияние тренда. При этом оставшуюся часть влияния тренда мы исключаем, беря вторую разницу. Это подтверждает наше предположение о необходимости дифференцирования порядка в приведенных выше примерах.

Упражнения 2

1. Проверьте значения KPSS и ADF для исходных данных; данных с обычной производной; данных со второй обычной производной; данные с сезонной производной; и данные с обоими производными.

2. Проверьте графики ACF и PACF и сделайте выводы о порядке модели для модели со второй обычной производной; для данных с сезонной производной; и для данных с обоими производными.

3. Проверьте и докажите, почему вам не нужно брать 3-ю обычную производную и вторую сезонную производную.

Выбор порядка модели SARIMA

В объяснении выше мы отметили, что сезонная разница делает данные более стационарными. Затем нам нужно выбрать наилучшее начальное предположение о порядках модели SARIMA.

Правила выбора начальных порядков

- Правильный порядок d - это порядок дифференцирования, который дает временной ряд с шумоподобным поведением, т.е. случайные колебания около четко определенного среднего значения с почти постоянным разбросом, проверьте на стационарность по критериям, указанным выше. Если временной ряд имеет положительные значения ACF с большим значением лага добавить обычную производную.

- Используйте сезонную производную D только в случае сильного влияния сезонности для модели.

- Количество слагаемых AR (порядок) определяется как последнее значение лага PACF перед быстрым уменьшением от положительных значений до нуля.

- Количество слагаемых скользящего среднего (MA) определяется как последнее значение лага ACF перед быстрым увеличением от отрицательных значений до нуля.

- Добавьте слагаемое SAR, если значения ACF периодически положительная.

- Помимо этого, порядок SAR может быть оценен из PACF. Посмотрите на количество значений лагов, которые кратны периоду сезона. Например, если период равен 24, и мы видим, что 24-е и 48-е запаздывания значительны в PACF, это означает, что начальное P должно быть 2.

- Добавьте член SMA, если значения ACF периодически отрицательный. Используйте те же правила определения количества лагов, что и для SAR.

- Если ваш временной ряд немного недодифференцирован, добавьте дополнительное слагаемое в AR.

- Если ваши ряды немного передифференцирован, добавьте дополнительные слагаемое в MA.

- Старайтесь избегать использования более одного или двух сезонных порядков (SAR + SMA) в одной модели, так как это может привести к переобучению данных и/или проблемам в точности оценок.

```python
import pmdarima as pm

# Seasonal - fit stepwise auto-ARIMA
smodel = pm.auto_arima(train,
                       start_p=10, #Search for Usual AR order
                       start_q=1,  #Search for Usual MA order
                       test='adf',
                       max_p=10,
                       max_q=10,
                       d=None, #Search for Usual Difference Order
                       m=12,   #The period for seasonal differencing
                       seasonal=True, #SARIMA ENABLE
                       start_P=0, #Search for Seasonal AR order
                       start_Q=0, #Search for Seasonal MA order
                       D=None, #Search for Seasonal Difference Order
                       trace=True,
                       error_action='ignore',
                       suppress_warnings=True,
                       stepwise=True)

smodel.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(10,1,1)(0,0,0)[12] intercept   : AIC=519.755, Time=1.98 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=1500.491, Time=0.01 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=inf, Time=0.49 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=1025.470, Time=0.29 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=1502.018, Time=0.01 sec
 ARIMA(10,1,1)(1,0,0)[12] intercept   : AIC=513.823, Time=5.88 sec
 ARIMA(10,1,1)(2,0,0)[12] intercept   : AIC=483.623, Time=20.12 sec
 ARIMA(10,1,1)(2,0,1)[12] intercept   : AIC=inf, Time=18.83 sec
 ARIMA(10,1,1)(1,0,1)[12] intercept   : AIC=inf, Time=5.53 sec
 ARIMA(9,1,1)(2,0,0)[12] intercept   : AIC=481.660, Time=15.57 sec
 ARIMA(9,1,1)(1,0,0)[12] intercept   : AIC=564.281, Time=4.26 sec
 ARIMA(9,1,1)(2,0,1)[12] intercept   : AIC=435.565, Time=16.06 sec
 ARIMA(9,1,1)(1,0,1)[12] intercept   : AIC=inf, Time=4.43 sec
 ARIMA(9,1,1)(2,0,2)[12] intercept   : AIC=inf, Time=27.89 sec
 ARIMA(9,1,1)(1,0,2)[12] intercept   : AIC=433.047, Time=10.23 sec
 ARIMA(9,1,1)(0,0,2)[12] intercept   : AIC=534.469, Time=13.22 sec
 ARIMA(9,1,1)(0,0,1)[12] intercept   : AIC=537.707, Time=2.59 sec
 ARIMA(8,1,1)(1,0,2)[12] intercept   : AIC=428.678, Time=7.85 sec
 ARIMA(8,1,1)(0,0,2)[12] intercept   : AIC=552.529, Time=5.32 sec
 ARIMA(8,1,1)(1,0,1)[12] intercept   : AIC=431.771, Time=4.36 sec
 ARIMA(8,1,1)(2,0,2)[12] intercept   : AIC=inf, Time=16.16 sec
 ARIMA(8,1,1)(0,0,1)[12] intercept   : AIC=557.384, Time=2.26 sec
 ARIMA(8,1,1)(2,0,1)[12] intercept   : AIC=440.569, Time=13.39 sec
 ARIMA(7,1,1)(1,0,2)[12] intercept   : AIC=391.199, Time=7.25 sec
 ARIMA(7,1,1)(0,0,2)[12] intercept   : AIC=601.475, Time=5.55 sec
 ARIMA(7,1,1)(1,0,1)[12] intercept   : AIC=382.841, Time=3.62 sec
 ARIMA(7,1,1)(0,0,1)[12] intercept   : AIC=611.099, Time=2.03 sec
 ARIMA(7,1,1)(1,0,0)[12] intercept   : AIC=564.036, Time=2.93 sec
 ARIMA(7,1,1)(2,0,1)[12] intercept   : AIC=385.415, Time=11.30 sec
```

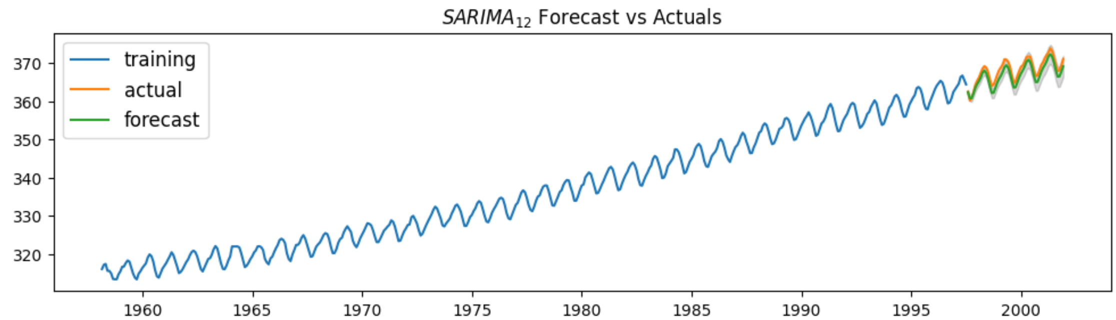
```
 ARIMA(7,1,1)(0,0,0)[12] intercept   : AIC=638.723, Time=1.16 sec
 ARIMA(7,1,1)(2,0,0)[12] intercept   : AIC=480.526, Time=9.47 sec
 ARIMA(7,1,1)(2,0,2)[12] intercept   : AIC=inf, Time=10.34 sec
 ARIMA(6,1,1)(1,0,1)[12] intercept   : AIC=382.791, Time=4.19 sec
 ARIMA(6,1,1)(0,0,1)[12] intercept   : AIC=682.219, Time=1.92 sec
 ARIMA(6,1,1)(1,0,0)[12] intercept   : AIC=559.635, Time=2.86 sec
 ARIMA(6,1,1)(2,0,1)[12] intercept   : AIC=384.434, Time=10.86 sec
 ARIMA(6,1,1)(1,0,2)[12] intercept   : AIC=389.497, Time=6.65 sec
 ARIMA(6,1,1)(0,0,0)[12] intercept   : AIC=732.258, Time=0.74 sec
 ARIMA(6,1,1)(0,0,2)[12] intercept   : AIC=663.277, Time=4.05 sec
 ARIMA(6,1,1)(2,0,0)[12] intercept   : AIC=479.604, Time=10.68 sec
 ARIMA(6,1,1)(2,0,2)[12] intercept   : AIC=inf, Time=11.57 sec
 ARIMA(5,1,1)(1,0,1)[12] intercept   : AIC=387.764, Time=2.54 sec
 ARIMA(6,1,0)(1,0,1)[12] intercept   : AIC=386.595, Time=3.07 sec
 ARIMA(6,1,2)(1,0,1)[12] intercept   : AIC=385.992, Time=3.48 sec
 ARIMA(5,1,0)(1,0,1)[12] intercept   : AIC=388.336, Time=2.22 sec
 ARIMA(5,1,2)(1,0,1)[12] intercept   : AIC=384.240, Time=2.86 sec
 ARIMA(7,1,0)(1,0,1)[12] intercept   : AIC=437.263, Time=3.05 sec
 ARIMA(7,1,2)(1,0,1)[12] intercept   : AIC=382.793, Time=3.90 sec
 ARIMA(6,1,1)(1,0,1)[12]             : AIC=365.821, Time=2.90 sec
 ARIMA(6,1,1)(0,0,1)[12]             : AIC=765.356, Time=0.67 sec
 ARIMA(6,1,1)(1,0,0)[12]             : AIC=552.143, Time=1.60 sec
 ARIMA(6,1,1)(2,0,1)[12]             : AIC=363.673, Time=11.63 sec
 ARIMA(6,1,1)(2,0,0)[12]             : AIC=477.943, Time=3.26 sec
 ARIMA(6,1,1)(2,0,2)[12]             : AIC=365.390, Time=9.48 sec
 ARIMA(6,1,1)(1,0,2)[12]             : AIC=363.652, Time=5.92 sec
 ARIMA(6,1,1)(0,0,2)[12]             : AIC=724.284, Time=1.50 sec
 ARIMA(5,1,1)(1,0,2)[12]             : AIC=inf, Time=9.47 sec
 ARIMA(6,1,0)(1,0,2)[12]             : AIC=365.332, Time=4.45 sec
 ARIMA(7,1,1)(1,0,2)[12]             : AIC=364.297, Time=6.05 sec
 ARIMA(6,1,2)(1,0,2)[12]             : AIC=inf, Time=7.69 sec
 ARIMA(5,1,0)(1,0,2)[12]             : AIC=365.808, Time=3.46 sec
 ARIMA(5,1,2)(1,0,2)[12]             : AIC=inf, Time=6.37 sec
 ARIMA(7,1,0)(1,0,2)[12]             : AIC=364.445, Time=5.01 sec
 ARIMA(7,1,2)(1,0,2)[12]             : AIC=inf, Time=7.35 sec

Best model:  ARIMA(6,1,1)(1,0,2)[12]
Total fit time: 411.861 seconds

<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
Dep. Variable:                                      y   No. Observations:
473
Model:             SARIMAX(6, 1, 1)x(1, 0, [1, 2], 12)   Log Likelihood
-170.826
Date:                                Mon, 03 May 2021   AIC
363.652
Time:                                        16:45:14   BIC
409.378
Sample:                                             0   HQIC
```

```
381.638
                                                             - 473
Covariance Type:                                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.2816      0.144      1.956      0.050      -0.001       0.564
ar.L2         -0.0111      0.033     -0.337      0.736      -0.076       0.053
ar.L3         -0.1518      0.063     -2.425      0.015      -0.274      -0.029
ar.L4          0.0018      0.036      0.050      0.960      -0.069       0.073
ar.L5          0.0127      0.045      0.282      0.778      -0.076       0.101
ar.L6         -0.0934      0.045     -2.079      0.038      -0.181      -0.005
ma.L1         -0.5693      0.136     -4.187      0.000      -0.836      -0.303
ar.S.L12       0.9995      0.001   1926.967      0.000       0.998       1.000
ma.S.L12      -0.8511      0.044    -19.237      0.000      -0.938      -0.764
ma.S.L24      -0.0268      0.042     -0.634      0.526      -0.110       0.056
sigma2         0.1088      0.006     19.283      0.000       0.098       0.120
================================================================================
===
Ljung-Box (L1) (Q):                   0.04   Jarque-Bera (JB):                99
.33
Prob(Q):                              0.83   Prob(JB):                         0
.00
Heteroskedasticity (H):               0.60   Skew:                             0
.37
Prob(H) (two-sided):                  0.00   Kurtosis:                         5
.12
================================================================================
```

Метод автопоиска дал порядок модели SARIMAX(6, 1, 1)x(1, 0, [1, 2], 12)

```
fitted, confint = smodel.predict(n_periods=test.size,
                                 return_conf_int=True)

# make series for plotting purpose
fc_series    = pd.Series(fitted, index=test.index)
lower_series = pd.Series(confint[:, 0], index=test.index)
upper_series = pd.Series(confint[:, 1], index=test.index)


# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
```

```
                    color='k',
                    alpha=0.15)

plt.title('$SARIMA_{12}$ Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.values)
```

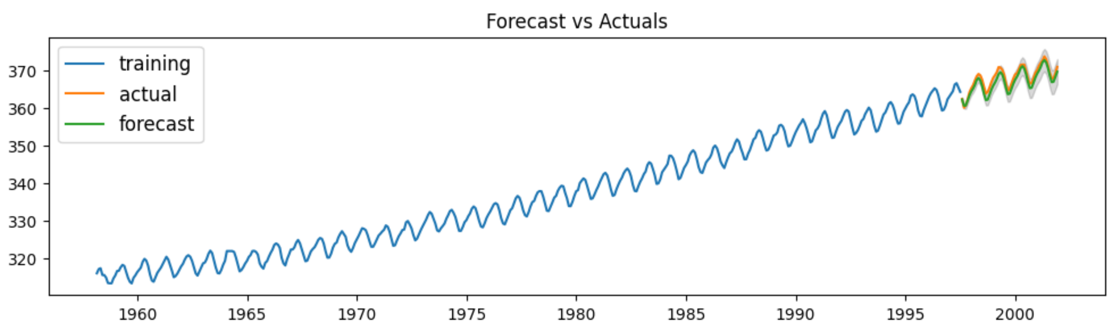



```
{'mean absoute percentage error': 0.003947772742976413,
 'mean absoute error            ': 1.454722139505576,
 'mean percentage error         ': -0.0038805878660586987,
 'root mean square              ': 1.5338558138656706,
 'correlation coefficient       ': 0.9846420347534222,
 'minmax error                  ': 0.003947716428085446}
```

Теперь попробуем с функцией из statsmodels

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = sm.tsa.statespace.SARIMAX(train,
                                  order=(10, 2, 8),
                                  seasonal_order=(1, 1, 2, 12))

fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.size, alpha=0.05, dynamic=False)
# 95% conf

# forecast = fitted.forecast(test.size, alpha=0.05)  # 95% conf
forecast = forecast_res.predicted_mean

# Make as pandas series
fc_series = pd.Series(forecast.values, index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower co2'], index=test.
index)
upper_series = pd.Series(forecast_res.conf_int()['upper co2'], index=test.
index)
```

```
# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train, label='training')
plt.plot(test,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.values)
```




```
{'mean absoute percentage error': 0.0031670916924645934,
 'mean absoute error            ': 1.166196574986573,
 'mean percentage error         ': -0.0031093048607534345,
 'root mean square              ': 1.2596568798934529,
 'correlation coefficient       ': 0.9850223690012644,
 'minmax error                  ': 0.0031670475141546417}
```

Упражнение 3

1. Сравните результаты, полученные с помощью автопоиска, с нашим исходным предположением о первой обычной производной и первой сезонной производной. Сравните предыдущие результаты с теми, что получены по второй сезонной производной. Выберите порядок модели из двух предыдущих проведите для него основные тесты.

Модель SARIMAX (SARIMA с экзогенными регрессорами)

В некоторых случаях мы можем повысить точность прогноза модели, введя экзогенную переменную. Основным требованием для использования экзогенной переменной является то, что вам необходимо знать значение переменной в течение тренировочной выборки и в период прогноза. В целом экзогенная переменная может быть какой угодно, независимо от обучающих данных.

Для примера в наших примерах мы можем извлечь компонент сезона из тестовых данных и рассматривать его как экзогенную переменную для обучающих данных. Мы сделаем это с помощью процедуры Season_decompose.

```
#ReSample to DataFrame

y = y_['co2'].resample('MS').mean()

# The term bfill means that we use the value before filling in missing val
ues
y = y.fillna(y.bfill())

ydf = pd.DataFrame(y)
ydf.columns = ['endog']
ydf.head()

                 endog
1958-03-01  316.100000
1958-04-01  317.200000
1958-05-01  317.433333
1958-06-01  315.625000
1958-07-01  315.625000
```

Создадим новую выборку co2 как тренировочную (endog) и создадим экзогенную выборку (exog)

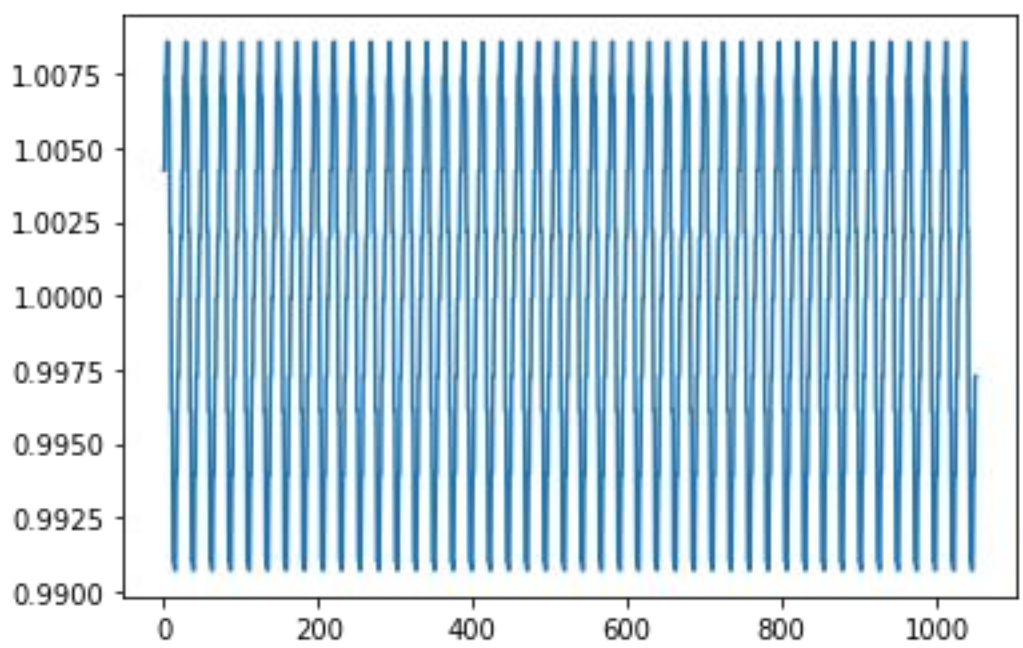
```
# Compute Seasonal Index
from statsmodels.tsa.seasonal import seasonal_decompose

y = y_['co2'].resample('MS').mean()

# The term bfill means that we use the value before filling in missing val
ues
y = y.fillna(y.bfill())

ydf = pd.DataFrame(y)
ydf.columns = ['endog']
ydf.head()

# multiplicative seasonal component
result_mul = seasonal_decompose(ydf.endog,   # 3 years
                                model='multiplicative')

seasonal_index = result_mul.seasonal[-24:].to_frame()
```

```python
seasonal_index['month'] = pd.to_datetime(seasonal_index.index).month

# merge with the base data
ydf['month'] = ydf.index.month

ydf = pd.merge(ydf, seasonal_index, how='left', on='month')
ydf.columns = ['endog', 'month', 'exog']
```

```python
ydf.head(12)
```

```
         endog  month      exog
0   316.100000      3  1.004239
1   316.100000      3  1.004239
2   317.200000      4  1.007380
3   317.200000      4  1.007380
4   317.433333      5  1.008577
5   317.433333      5  1.008577
6   315.625000      6  1.006689
7   315.625000      6  1.006689
8   315.625000      7  1.002177
9   315.625000      7  1.002177
10  314.950000      8  0.996102
11  314.950000      8  0.996102
```

```python
ydf.exog.plot();
```



Разделим выбоки

```python
train , test = pm.model_selection.train_test_split(ydf,test_size=0.1)
```

```python
# Seasonal - fit stepwise auto-ARIMA
sxmodel = pm.auto_arima(train.endog,
                        exogenous = train.exog.values.reshape(-1,1),
                        start_p=10, #Search for Usual AR order
                        start_q=1,  #Search for Usual MA order
                        test='adf',
                        max_p=10,
                        max_q=10,
                        d=None, #Search for Usual Difference Order
                        m=12,   #The period for seasonal differencing
                        seasonal=True, #SARIMA ENABLE
                        start_P=0, #Search for Seasonal AR order
                        start_Q=0, #Search for Seasonal MA order
                        D=None, #Search for Seasonal Difference Order
                        trace=True,
                        error_action='ignore',
                        suppress_warnings=True,
                        stepwise=True)

sxmodel.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(10,1,1)(0,0,0)[12] intercept   : AIC=296.674, Time=2.31 sec
 ARIMA(0,1,0)(0,0,0)[12] intercept   : AIC=332.551, Time=0.08 sec
 ARIMA(1,1,0)(1,0,0)[12] intercept   : AIC=316.706, Time=0.30 sec
 ARIMA(0,1,1)(0,0,1)[12] intercept   : AIC=311.755, Time=0.35 sec
 ARIMA(0,1,0)(0,0,0)[12]             : AIC=372.107, Time=0.08 sec
 ARIMA(10,1,1)(1,0,0)[12] intercept   : AIC=296.680, Time=4.98 sec
 ARIMA(10,1,1)(0,0,1)[12] intercept   : AIC=296.619, Time=2.98 sec
 ARIMA(10,1,1)(1,0,1)[12] intercept   : AIC=299.271, Time=5.72 sec
 ARIMA(10,1,1)(0,0,2)[12] intercept   : AIC=298.551, Time=9.68 sec
 ARIMA(10,1,1)(1,0,2)[12] intercept   : AIC=300.372, Time=8.92 sec
 ARIMA(9,1,1)(0,0,1)[12] intercept   : AIC=295.174, Time=2.97 sec
 ARIMA(9,1,1)(0,0,0)[12] intercept   : AIC=295.872, Time=3.38 sec
 ARIMA(9,1,1)(1,0,1)[12] intercept   : AIC=298.494, Time=6.89 sec
 ARIMA(9,1,1)(0,0,2)[12] intercept   : AIC=297.171, Time=18.97 sec
 ARIMA(9,1,1)(1,0,0)[12] intercept   : AIC=295.241, Time=5.43 sec
 ARIMA(9,1,1)(1,0,2)[12] intercept   : AIC=299.335, Time=15.26 sec
 ARIMA(8,1,1)(0,0,1)[12] intercept   : AIC=295.533, Time=2.22 sec
 ARIMA(9,1,0)(0,0,1)[12] intercept   : AIC=295.512, Time=1.05 sec
 ARIMA(9,1,2)(0,0,1)[12] intercept   : AIC=297.423, Time=2.92 sec
 ARIMA(8,1,0)(0,0,1)[12] intercept   : AIC=293.675, Time=0.96 sec
 ARIMA(8,1,0)(0,0,0)[12] intercept   : AIC=295.455, Time=0.55 sec
 ARIMA(8,1,0)(1,0,1)[12] intercept   : AIC=295.873, Time=1.87 sec
 ARIMA(8,1,0)(0,0,2)[12] intercept   : AIC=295.664, Time=2.39 sec
 ARIMA(8,1,0)(1,0,0)[12] intercept   : AIC=293.714, Time=1.43 sec
 ARIMA(8,1,0)(1,0,2)[12] intercept   : AIC=296.049, Time=6.62 sec
 ARIMA(7,1,0)(0,0,1)[12] intercept   : AIC=296.960, Time=0.66 sec
 ARIMA(7,1,1)(0,0,1)[12] intercept   : AIC=296.689, Time=1.33 sec
 ARIMA(8,1,0)(0,0,1)[12]             : AIC=360.138, Time=1.20 sec
```

```
Best model:  ARIMA(8,1,0)(0,0,1)[12] intercept
Total fit time: 111.542 seconds
```

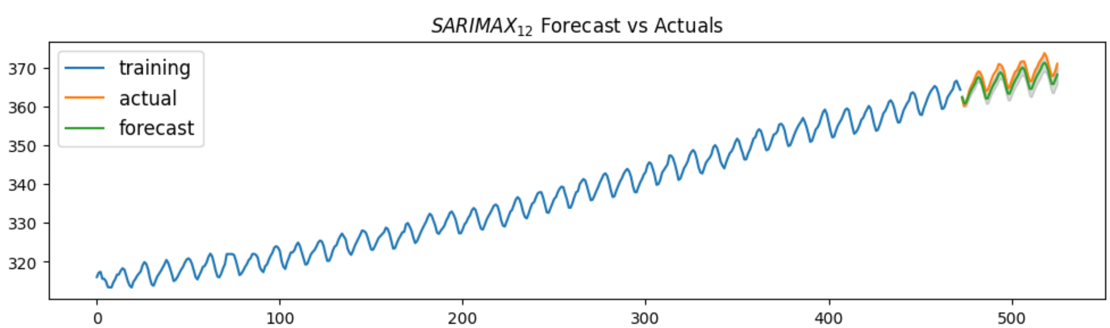
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                      y   No. Observations:
473
Model:             SARIMAX(8, 1, 0)x(0, 0, [1], 12)   Log Likelihood
-134.837
Date:                                Mon, 03 May 2021   AIC
293.675
Time:                                        22:38:33   BIC
343.558
Sample:                                             0   HQIC
313.297
                                                - 473
Covariance Type:                                  opg
========================================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
----------------------------------------------------------------------------------------
intercept      0.2309      0.030      7.823      0.000       0.173       0.289
x1           336.3614      4.631     72.635      0.000     327.285     345.438
ar.L1         -0.2627      0.037     -7.096      0.000      -0.335      -0.190
ar.L2         -0.1614      0.039     -4.099      0.000      -0.239      -0.084
ar.L3         -0.2515      0.047     -5.318      0.000      -0.344      -0.159
ar.L4         -0.1415      0.052     -2.707      0.007      -0.244      -0.039
ar.L5         -0.0650      0.055     -1.173      0.241      -0.174       0.044
ar.L6         -0.1158      0.053     -2.173      0.030      -0.220      -0.011
ar.L7         -0.1117      0.048     -2.312      0.021      -0.206      -0.017
ar.L8         -0.1129      0.048     -2.358      0.018      -0.207      -0.019
ma.S.L12       0.0983      0.048      2.044      0.041       0.004       0.193
sigma2         0.1036      0.005     20.190      0.000       0.094       0.114
==========================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                      157
.98
Prob(Q):                              0.96   Prob(JB):                                0
.00
Heteroskedasticity (H):               0.60   Skew:                                    0
.38
Prob(H) (two-sided):                  0.00   Kurtosis:                                5
.73
==========================================================================================

"""
```

Попробуем предсказание.

```
ex4test = pd.DataFrame(test.exog)
ex4test.head()
```

```
          exog
20752  0.996102
20753  0.996102
20754  0.996102
20755  0.996102
20756  0.996102
```

```
fitted, confint = sxmodel.predict(n_periods=np.shape(test.exog.values)[0],
                                exogenous=ex4test,
                                return_conf_int=True)

# make series for plotting purpose
fc_series    = pd.Series(fitted, index=test.index)
lower_series = pd.Series(confint[:, 0], index=test.index)
upper_series = pd.Series(confint[:, 1], index=test.index)


# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train.endog, label='training')
plt.plot(test.endog,  label='actual')
plt.plot(fc_series, label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('$SARIMAX_{12}$ Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()

forecast_accuracy(fc_series.values, test.endog)
```



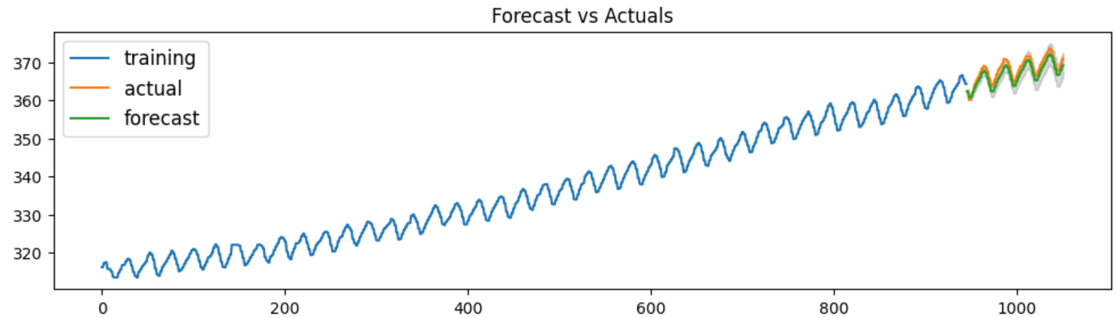

```
{'mean absoute percentage error': 0.005060117981539687,
 'mean absoute error            ': 1.8659586807355004,
 'mean percentage error         ': -0.0049627858435570835,
```

```
 'root mean square              ': 1.9576150319647736,
 'correlation coefficient       ': 0.9820501688597131,
 'minmax error                  ': 0.0050600257762539735}
```

Теперь попробуем модель SARIMAX из statsmodels.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX

model = sm.tsa.statespace.SARIMAX(endog=train.endog,
                                  exog =train.exog,
                                  order=(8, 2, 8),
                                  seasonal_order=(0, 0, 1, 12))

fitted = model.fit()

# Forecast
forecast_res = fitted.get_forecast(test.endog.size,
                                   exog    = test.exog,
                                   alpha   = 0.05,
                                   dynamic = False)  # 95% conf

forecast = forecast_res.predicted_mean

# Make as pandas series
fc_series = pd.Series(forecast.values,
                      index=test.index)

lower_series = pd.Series(forecast_res.conf_int()['lower endog'], index=tes
t.index)
upper_series = pd.Series(forecast_res.conf_int()['upper endog'], index=tes
t.index)


# Plot
plt.figure(figsize=(12,3), dpi=100)

plt.plot(train.endog, label='training')
plt.plot(test.endog,  label='actual')
plt.plot(fc_series,   label='forecast')

plt.fill_between(lower_series.index,
                 lower_series,
                 upper_series,
                 color='k',
                 alpha=0.15)

plt.title('Forecast vs Actuals')
plt.legend(loc='upper left', fontsize=12)

plt.show()
```

```
forecast_accuracy(fc_series.values, test.endog.values)
```




```
{'mean absoute percentage error': 0.0036812992027410066,
 'mean absoute error           ': 1.356824961133212,
 'mean percentage error        ': -0.0035763191213044553,
 'root mean square             ': 1.4331867044616442,
 'correlation coefficient      ': 0.9855394369808252,
 'minmax error                 ': 0.0036812000258082955}
```

Для нашего игрушечного случая мы не получили большого прироста точности, вероятно это связано с тем, что экзогенные данные выбраны из той же выборки, что и тренировочные, и не вносят новой информации.

Упражнение 4

1. Попробуйте использовать тренд тестовых данных в качестве экзогенных факторов вместо сезонности.