Введение в statsmodels. Непараметрический анализ временных рядов.

Знакомство с библиотекой статистического анализа временных рядов statsmodels.tsa. Разложение временных рядов. Методы непараметрического предсказания временных рядов. Методы скользящего среднего.

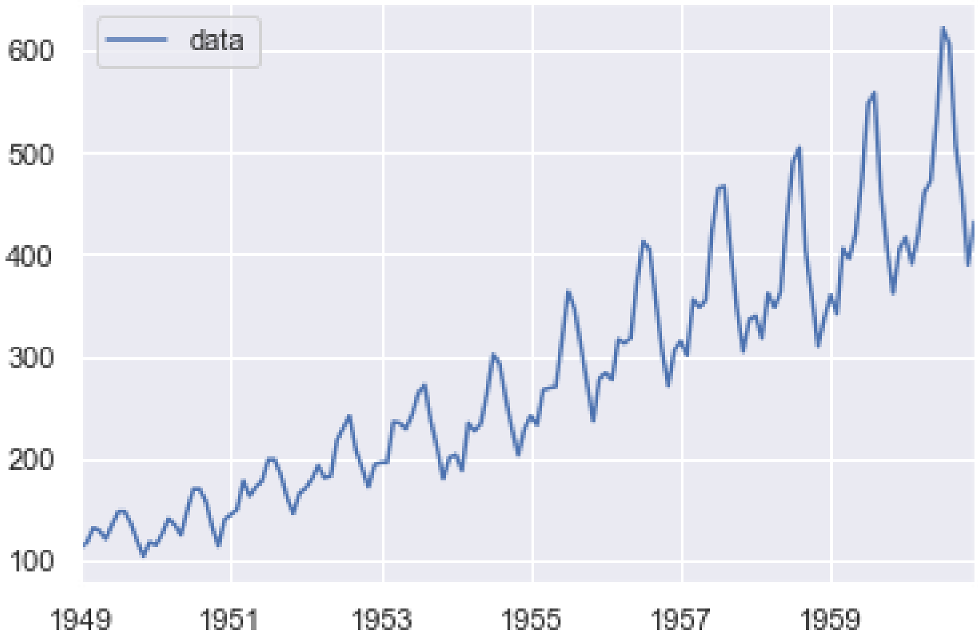
Импорт данных.

Statsmodels фреймворка python - один из самых популярных инструментов исследователей для решения многих задач статистического анализа. Одним из наиболее интересных модулей этой библиотеки является statsmodels.tsa, описание которого вы можете найти здесь: https://www.statsmodels.org/stable/tsa.html

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import matplotlib.dates as mdates
# Use seaborn style defaults and set the default figure size
sns.set(rc={'font.size': 15})
%matplotlib inline

try:
    import statsmodels.api as sm
except:
    !pip install statsmodels
finally:
    import statsmodels.api as sm
    import statsmodels
```

В этой работе мы будем работать со следующим набором данных

```
airpass = sm.datasets.get_rdataset("AirPassengers", "datasets")

airpass = pd.DataFrame(airpass.data["value"])

airpass.index = pd.date_range(start = "1949-01", periods =
len(airpass.index), freq = "M").to_period()

airpass.index = airpass.index.to_timestamp()

airpass=airpass.rename(columns={"value": "data"}, inplace = False)

airpass.plot();
```



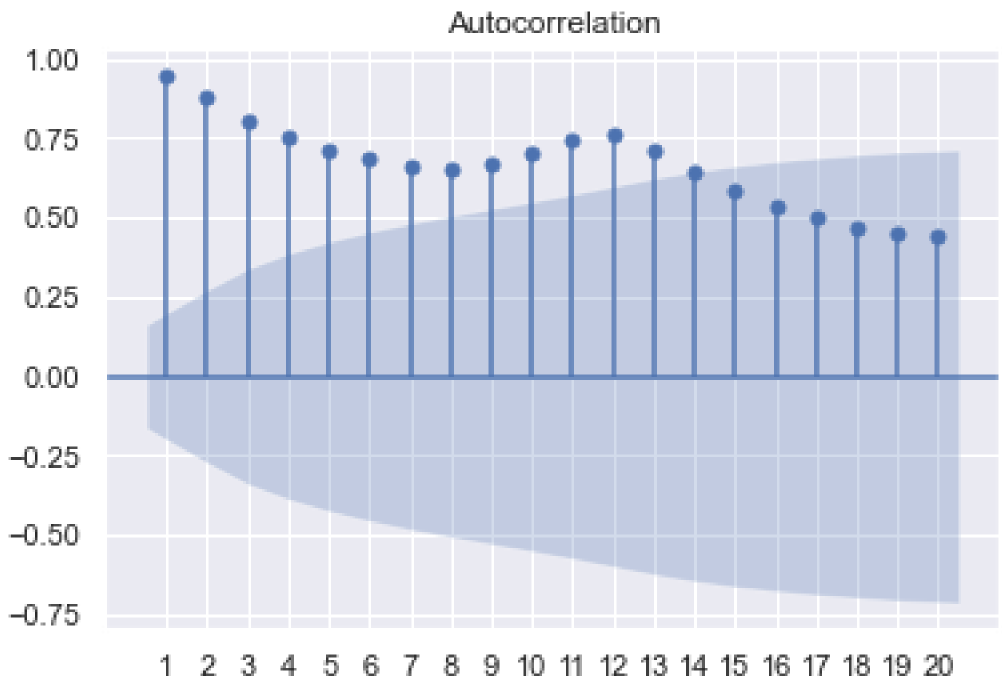
Введение в StatsModels

Для визуализации временных рядов с помощью StatsModels воспользуемся модулем sm.graphics. Для начала проанализируем функцию автокорреляции (АКФ, ACF). АКФ - это степень зависимости текущих значений временного ряда по отношению к его отстающей версии самого себя

$$cor(k) = \frac{1}{N}\frac{\sum_{i=0}^{N-1}(y_k - ev)(y_{i-k} - ev)}{var(y)}$$

Ниже взяты первые 20 лагов (начиная от 1-го ($k = 1$))

```
#Plot the ACF:
lags = 20
sm.graphics.tsa.plot_acf(airpass,
                         lags = lags,
                         zero = False)
plt.xticks(np.arange(1,  lags + 1, 1.0));
plt.show()
```



Помимо полной корреляции временного ряда пакет StatsModels позволяет оценить его частичную корреляционную функцию (PACF). «Частичная» корреляция между двумя переменными - это степень корреляции между ними, которая не объясняется их взаимной корреляцией с заданным набором других переменных. Например, если мы регрессируем переменную Y по другим переменным X1, X2 и X3, частичная корреляция между Y и X3 - это степень корреляции между Y и X3, которая не объясняется их общими корреляциями с X1 и X2. Здесь X1 X2 и X3 могут быть отстающими версиями Y.

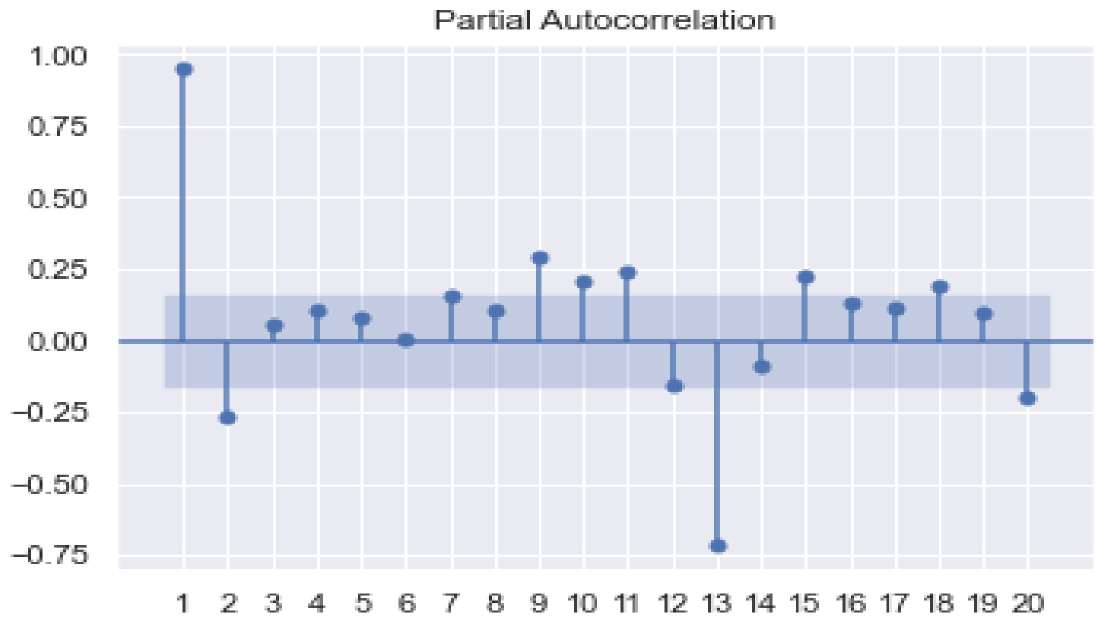
$$PACF(k,p) = \frac{\sum_{i=0}^{N-1}\left(y_k - \hat{y}_{k|k-p+1}\right)\cdot\left(y_{i-k} - \hat{y}_{i-k|i-k-p+1}\right)}{std\left(y_k - \hat{y}_{k|k-p+1}\right)\cdot std\left(y_{i-k} - \hat{y}_{i-k|i-k-p+1}\right)} = \frac{PACVF(k,p)}{std\left(y_k - \hat{y}_{k|k-p+1}\right)\cdot std\left(y_{i-k} - \hat{y}_{i-k|i-k-p+1}\right)};$$

где

- $\hat{y}_{k|k-p+1}$ - это $\hat{y}_k$ предсказанное по $y_{k-1}, \dots, y_{k-p+1}$;
- $\hat{y}_{i-k|i-k-p+1}$ - это $\hat{y}_{i-k}$ предсказанное по $y_{i-k-1}, \dots, y_{i-k-p+1}$;
- $PACVF(k,p)$ - это частичная ковариация,

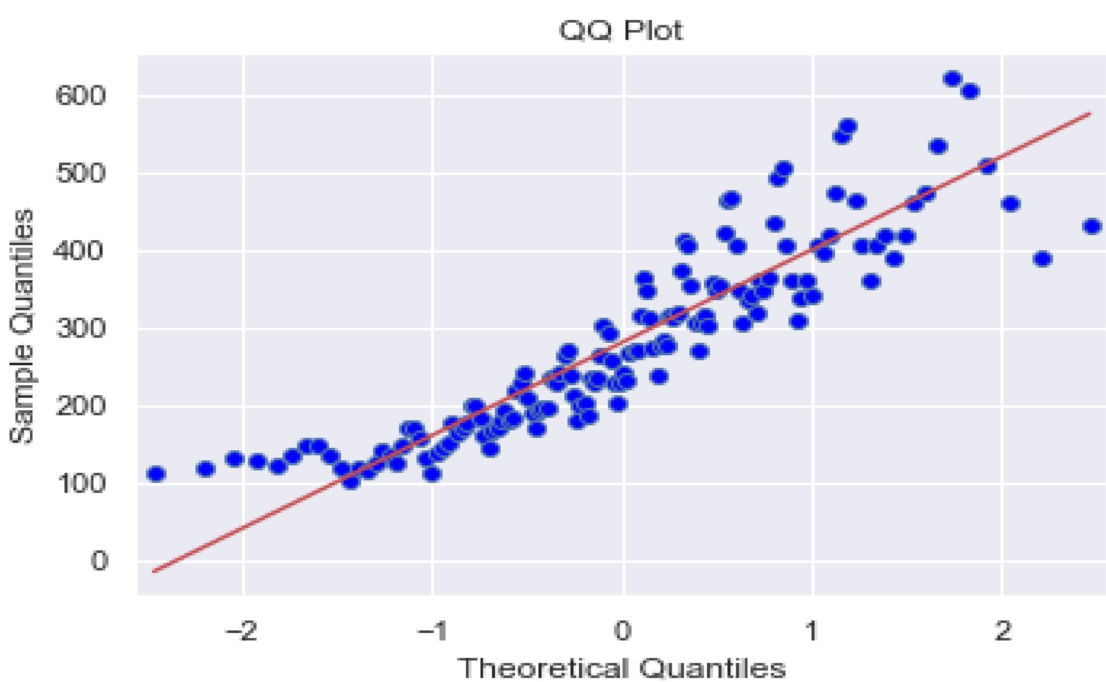
$$PACVF(k,p) = \sum_{i=0}^{N-1}\left(y_k - \hat{y}_{k|k-p+1}\right)\cdot\left(y_{i-k} - \hat{y}_{i-k|i-k-p+1}\right)/N$$

```
lags = 20
sm.graphics.tsa.plot_pacf(airpass,
                          lags = lags,
                          zero = False)
plt.xticks(np.arange(1,  lags + 1, 1.0))
plt.show()
```

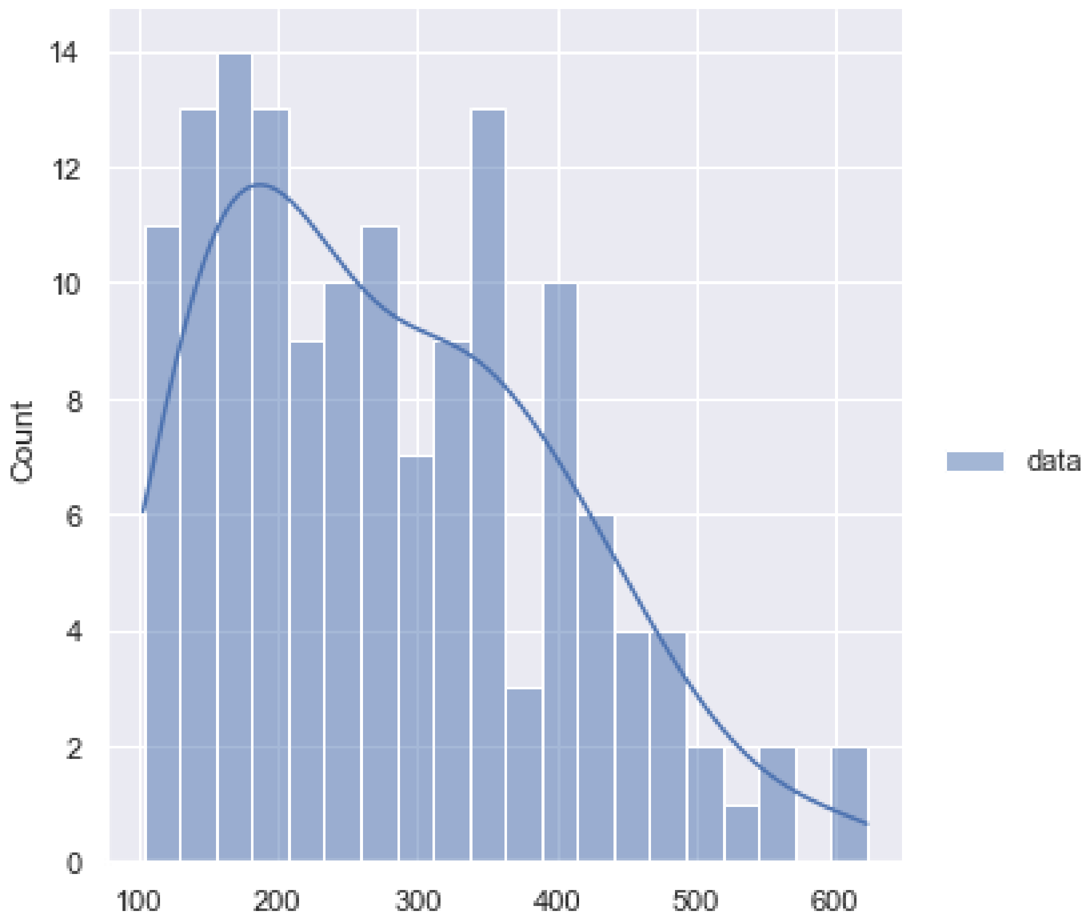


Для графической проверки распределения в наборе данных также можно построить график Q – Q (квантиль-квантиль) и график гистограммы.

```
#Plot the QQ plot of the data:
sm.qqplot(airpass,
          line='s')
plt.title("QQ Plot")
plt.show()
```



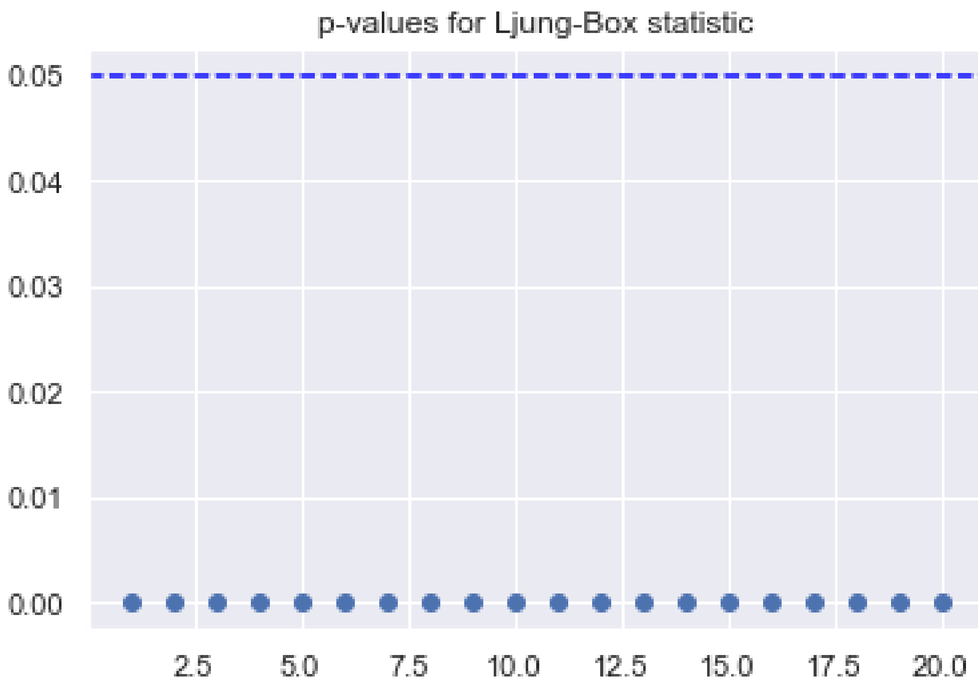

```
sns.displot(airpass,bins=20,kde=True);
```



Также проверим данные по тесту Люнга-Бокса (гипотеза нормального распределения).

```
lags = 20

#The Ljung-Box test results for the first k lags:
tmp_acor = sm.stats.diagnostic.acorr_ljungbox(airpass, lags = lags,
boxpierce = True, return_df = False)

# get the p-values
p_vals = pd.Series(tmp_acor[1])

#Start the index from 1 instead of 0 (because Ljung-Box test is for
lag values from 1 to k)
p_vals.index += 1
fig = plt.figure()

#Plot the p-values:
p_vals.plot(linestyle='',
            marker='o',
            title = "p-values for Ljung-Box statistic",
            legend = False)
```

```
#Add the horizontal 0.05 critical value line
plt.axhline(y = 0.05, color = 'blue', linestyle='--')
plt.show()
```



Мы также можем проверить стационарность данных с помощью расширенного теста Дики – Фуллера (ADF).

```
statmodels.tsa.stattools.adfuller.

dftest = sm.tsa.stattools.adfuller(airpass.data, autolag='AIC')

print("Test statistic = {:.3f}".format(dftest[0]))
print("P-value = {:.3f}".format(dftest[1]))
print("Critical values :")

for k, v in dftest[4].items():
    print("\t{}: {} - The data is {} stationary with {}%
confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))


Test statistic = 0.815
P-value = 0.992
Critical values :
	1%: -3.4816817173418295 - The data is not stationary with 99%
confidence
	5%: -2.8840418343195267 - The data is not stationary with 95%
confidence
	10%: -2.578770059171598 - The data is not stationary with 90%
confidence
```

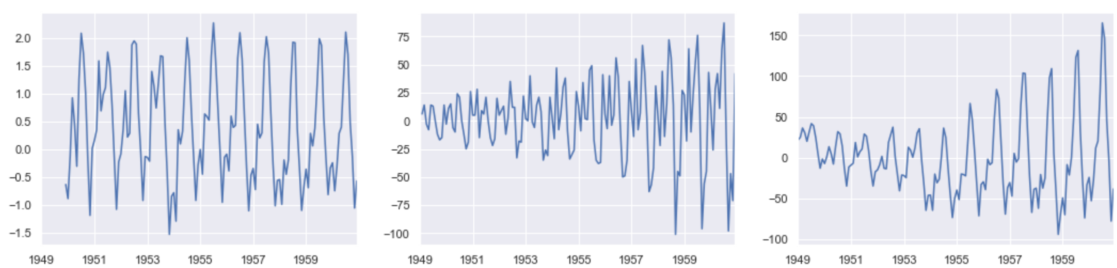
Упражнение 1

1. Промоделируйте нормальное распределение проведите для него весь представленный выше анализ.

Разложение тренд-сезон-остаток с помощью StatsModels

Мы также можем попробовать убрать тренд из данных, для этого можно использовать встроенную процедуру statsmodels.tsa.stattools.detrend. На практике мы воспользуемся тремя разными методами и сравним их результаты.

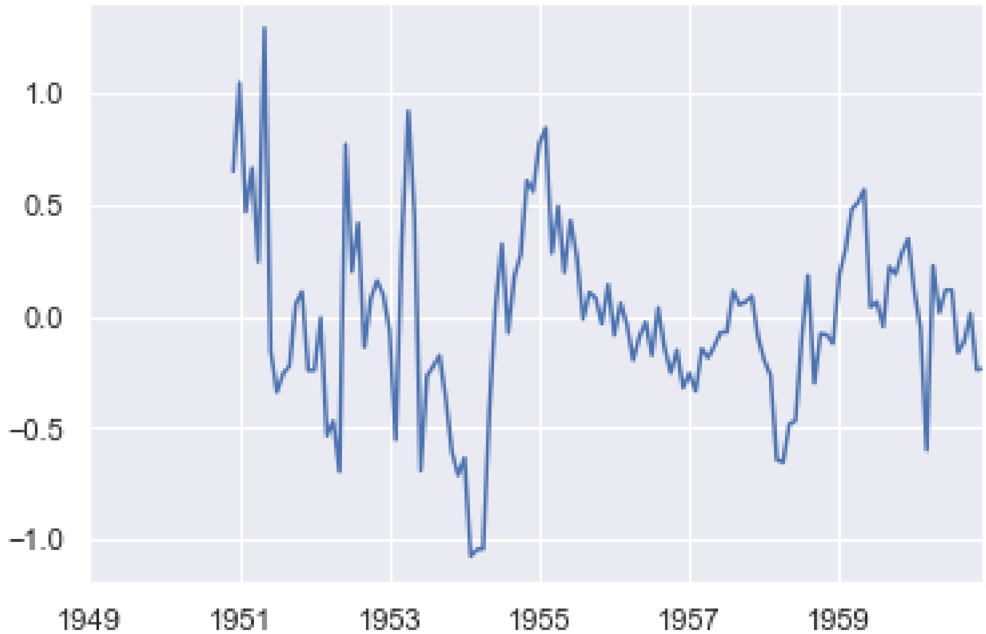
```
airpass[['de_trend_1']] = (airpass[['data']] -
airpass[['data']].rolling(window=12).mean()) /
airpass[['data']].rolling(window=12).std()

airpass[['de_trend_2']] = airpass[['data']].diff(1)

airpass[['de_trend_3']] =sm.tsa.tsatools.detrend(airpass[['data']],
order=1)

plt.figure(figsize = (18,4))

plt.subplot(131)
airpass.de_trend_1.plot();
plt.subplot(132)
airpass.de_trend_2.plot();
plt.subplot(133)
airpass.de_trend_3.plot();
```



Обратите внимание, что для того, чтобы сделать данные стационарными, мы также можем удалить сезонную часть. Самый простой способ - сделать сезонную производную для рядов с исключенным трендом.

```
airpass[['de_season']] = airpass.de_trend_1.diff(12)
airpass.de_season.plot();
airpass.head(3)
```

	data	de_trend_1	de_trend_2	de_trend_3	de_season
1949-01-01	112	NaN	NaN	21.690038	NaN
1949-02-01	118	NaN	6.0	25.032854	NaN
1949-03-01	132	NaN	14.0	36.375670	NaN




```
print('airpass.de_trend_1')

dftest = sm.tsa.stattools.adfuller(airpass.de_trend_1.dropna(),
autolag='AIC')

print("Test statistic = {:.3f}".format(dftest[0]))
print("P-value = {:.3f}".format(dftest[1]))
print("Critical values :")

for k, v in dftest[4].items():
    print("\t{}: {} - The data is {} stationary with {}%
confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))



print('airpass.de_season')

dftest = sm.tsa.stattools.adfuller(airpass.de_season.dropna(),
autolag='AIC')

print("Test statistic = {:.3f}".format(dftest[0]))
print("P-value = {:.3f}".format(dftest[1]))
print("Critical values :")

for k, v in dftest[4].items():
    print("\t{}: {} - The data is {} stationary with {}%
confidence".format(k, v, "not" if v<dftest[0] else "", 100-int(k[:-1])))
```

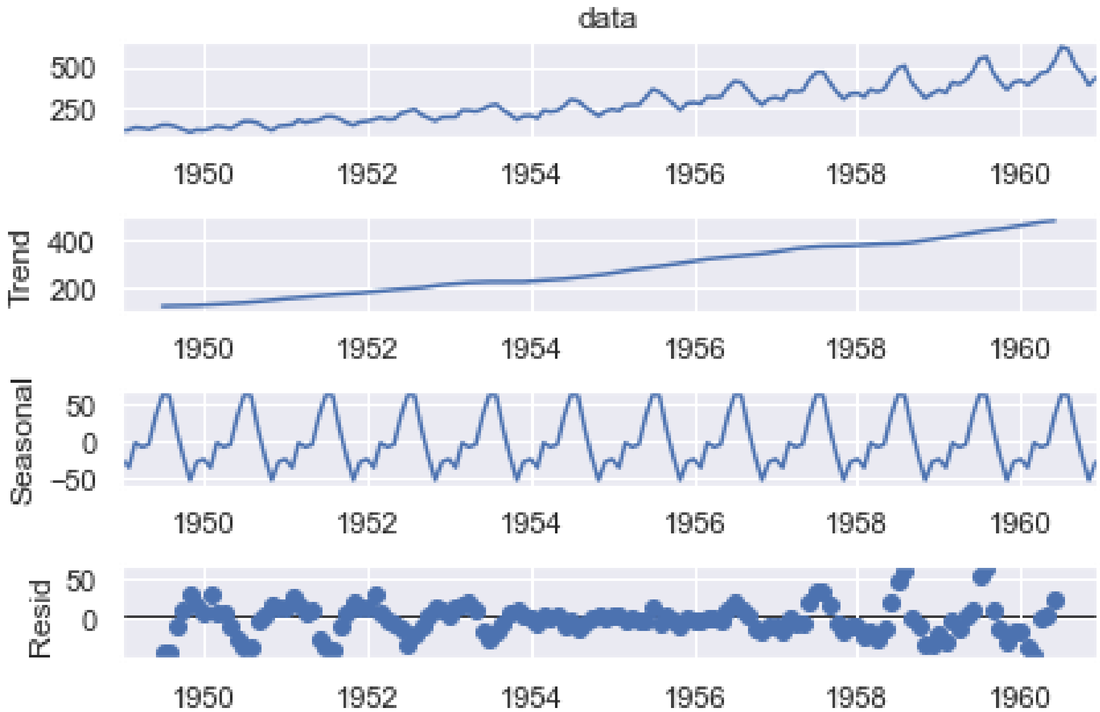
```
airpass.de_trend_1
Test statistic = -2.481
P-value = 0.120
Critical values :
	1%: -3.4865346059036564 - The data is not stationary with 99% confidence
	5%: -2.8861509858476264 - The data is not stationary with 95% confidence
	10%: -2.579896092790057 - The data is not stationary with 90% confidence
airpass.de_season
Test statistic = -3.181
P-value = 0.021
Critical values :
	1%: -3.4924012594942333 - The data is not stationary with 99% confidence
	5%: -2.8886968193364835 - The data is  stationary with 95% confidence
	10%: -2.5812552709190673 - The data is  stationary with 90% confidence
```

Упражнение 2

1. Проверьте изучаемый ряд на стационарность , а также ряды de_trend_2 и de_trend_3 с и без сезонной части.
2. Проведите графический анализ de_season части.
3. Визуализируйте части изучаемого временного ряда (тренд, сезонность, остаток) с использованием изученного метода (используйте $trend = y(t) - seasonal - residual$ и для сезонности также).

Существует множество методов декомпозиции временных рядов. Мы начнем со statsmodels.tsa.seasonal_decompose, чтобы автоматически разложить ряд. Метод реализует одностороннюю или двухстороннюю декомпозицию тренда простым скользящим средним, а затем пытается найти такую составляющую, что $\tilde{S}_t = \tilde{S}_{t+period}$, где $\tilde{S}$ – это временной ряд без тренда.

```
result = sm.tsa.seasonal_decompose(airpass.data, model='additive', period = 12, two_sided = True)

result.plot()

plt.show()
```



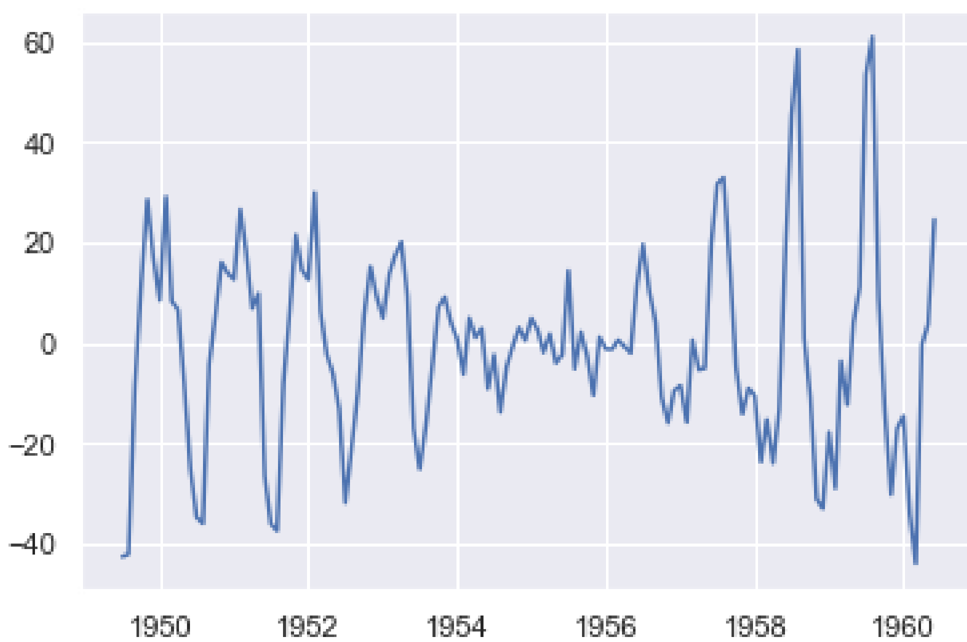
Мы также можем построить результаты разложения вместе.

```
plt.figure(figsize=(18,6))

plt.plot(airpass.data, label="airpass")

result.trend.plot(label="trend")

result.seasonal.plot(label="seasonal")

result.resid.plot(label="resid")

plt.legend(fontsize ='x-large')
plt.show()
```

Давайте проанализируем остатки нашей декомпозиции

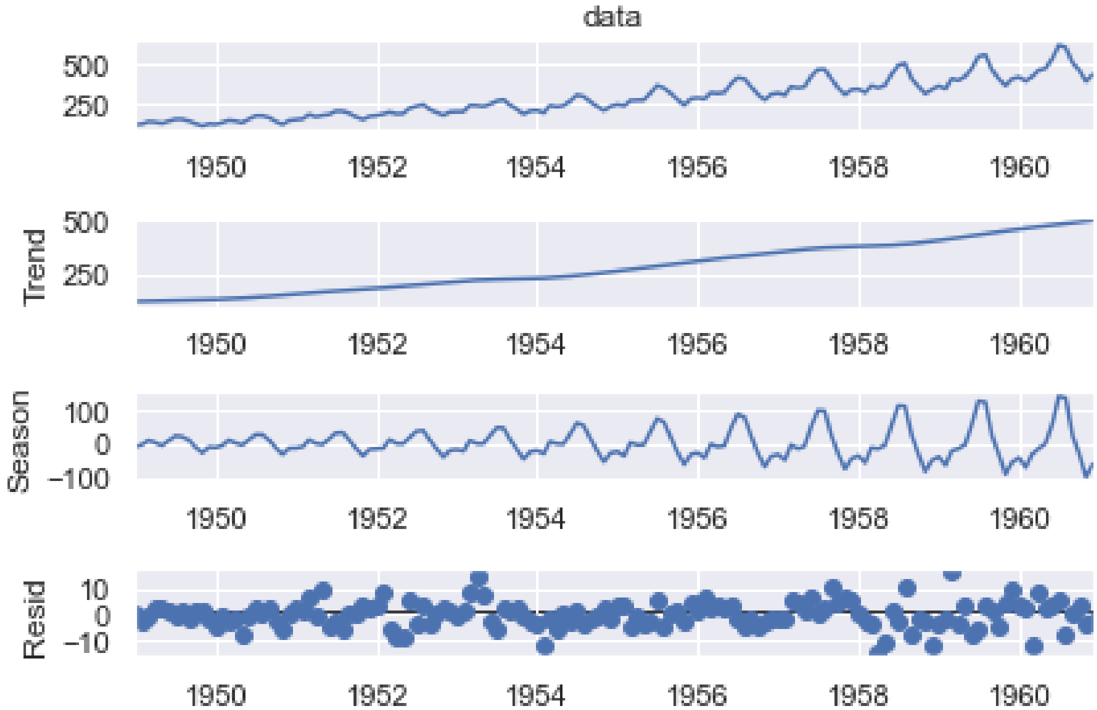
```
residuals =result.resid

rmse = np.sqrt(np.sum(np.power(residuals,2)))

print('Test RMSE: %.3f' % rmse)

residuals = pd.DataFrame(residuals)

print(residuals.describe())

plt.plot(residuals)

plt.show()
```

```
Test RMSE: 221.531
            resid
count  132.000000
mean    -0.751263
std     19.340535
min    -43.967172
25%    -11.248422
50%     -0.452020
75%      9.527146
max     61.051768
```



Другой метод разложения тренд-сезонной составляющих основан на локальной полиномиальной регрессии, которая также известна как LOESS (сглаживание локально оцененной диаграммы разброса). Этот метод называется разложением по сезонным трендам с использованием LOESS (**Seasonal-Trend decomposition using LOESS, STL**).

```
result_stl =sm.tsa.STL(airpass.data, period = 12,).fit()

fig = result_stl.plot()
```



Упражнение 3

1. Проанализируйте с помощью ранее показанного графического анализа и ADF остатки разложения `statsmodels.tsa.seasonal_decompose`.

2. Проанализируйте с помощью ранее показанного графического анализа и ADF остатки разложения `statsmodels.tsa.STL`.

Предсказание временных рядов

Наивное предсказание

Мы можем сделать простой (наивный) одношаговый прогноз. Давайте превратим наш набор данных в задачу обучения с учителем. Мы можем добиться этого, создав ряд с задержкой. Теперь, в преобразованном наборе данных значения в (t) являются предикторами (X), а значения в (t + 1) являются целевой переменной (Y).

```
airpass[['label']] = airpass.data.shift(1)
airpass.head()
```

	data	de_trend_1	de_trend_2	de_trend_3	de_season	label
1949-01-01	112	NaN	NaN	21.690038	NaN	NaN
1949-02-01	118	NaN	6.0	25.032854	NaN	112.0
1949-03-01	132	NaN	14.0	36.375670	NaN	118.0
1949-04-01	129	NaN	-3.0	30.718487	NaN	132.0
1949-05-01	121	NaN	-8.0	20.061303	NaN	129.0

Затем мы можем разделить набор данных на обучающую и тестовую подмножества, как показано ниже: 70% серии - обучающие, а 30% - тестовые данные.

```
x = airpass.data.values
y = airpass.label.values

train_size = int(len(x) * 0.7)

x_train, x_test = x[:train_size], x[train_size:]
y_train, y_test = y[:train_size], y[train_size:]
```

Наивное предсказание эквивалентно $y(t+1) = y(t)$

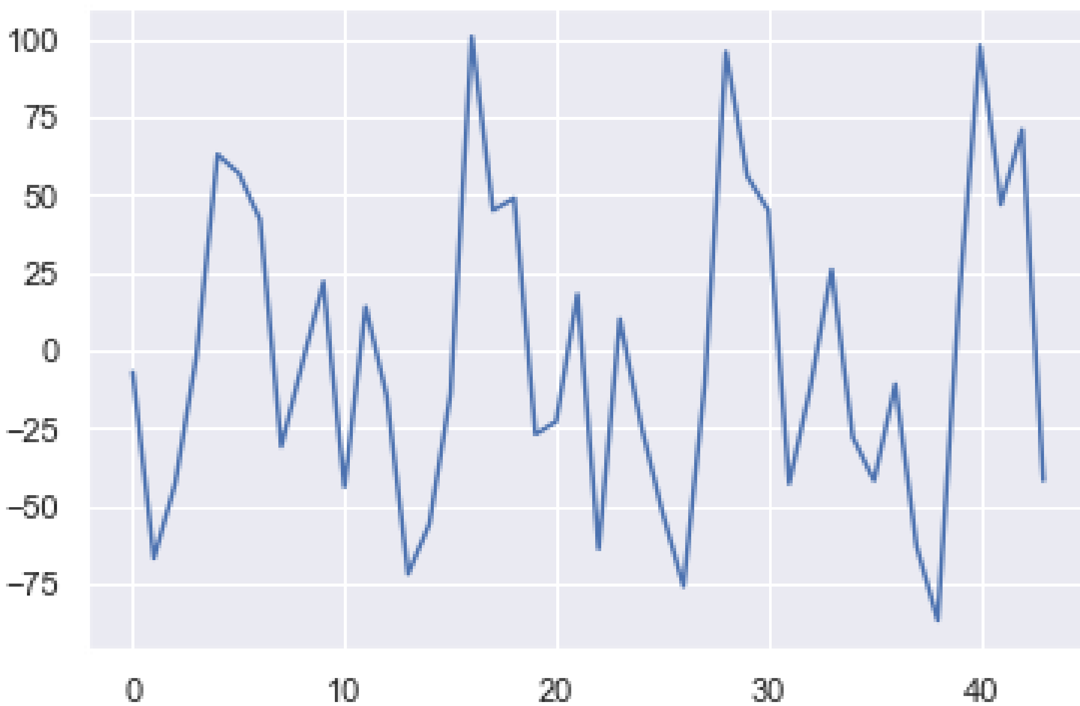
```
def naive_forecast(x_ts, n_predict):
    forecast = [x_ts[-1]]*n_predict
    return forecast

def forecast_residual(x_pred, x_ground_truth):
    residual = x_ground_truth-x_pred
    return residual
```

Мы можем оценить нашу базовую модель на тестовом наборе данных. Мы шаг за шагом будем просматривать набор тестовых данных (второй столбец) и получать прогнозы.

```
n_preidctions = x_test.size

predicted = naive_forecast(x_train,n_preidctions)

residuals = forecast_residual(x_test,y_test)

rmse = np.sqrt(np.sum(np.power(residuals,2)))
```

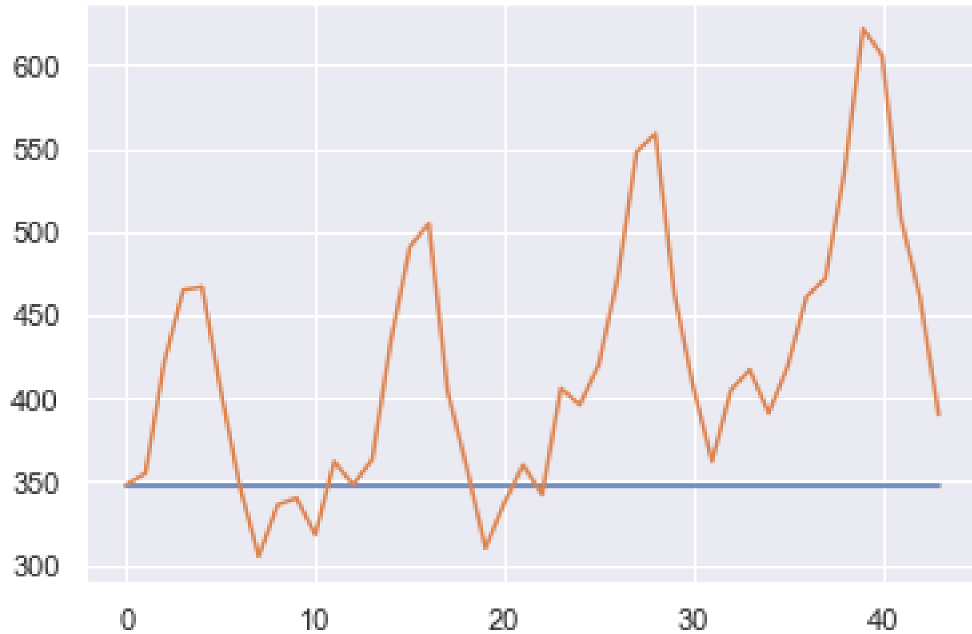
```python
print('Test RMSE: %.3f' % rmse)

residuals = pd.DataFrame(residuals)

print(residuals.describe())

plt.plot(residuals)

plt.show()
```

```
Test RMSE: 327.744
                0
count   44.000000
mean    -1.909091
std     49.943139
min    -87.000000
25%    -42.250000
50%    -11.000000
75%     42.750000
max    101.000000
```



Также будет полезно визуализировать данные.

```
plt.plot(predicted)
plt.plot(y_test)
```



Как мы видим выше, остатки нашего прогноза достаточно отличаются от нормального распределения. Также сводная статистика позволяет предположить смещение в модели.

Помимо простого наивного прогноза, его можно сделать сезонно-наивным,

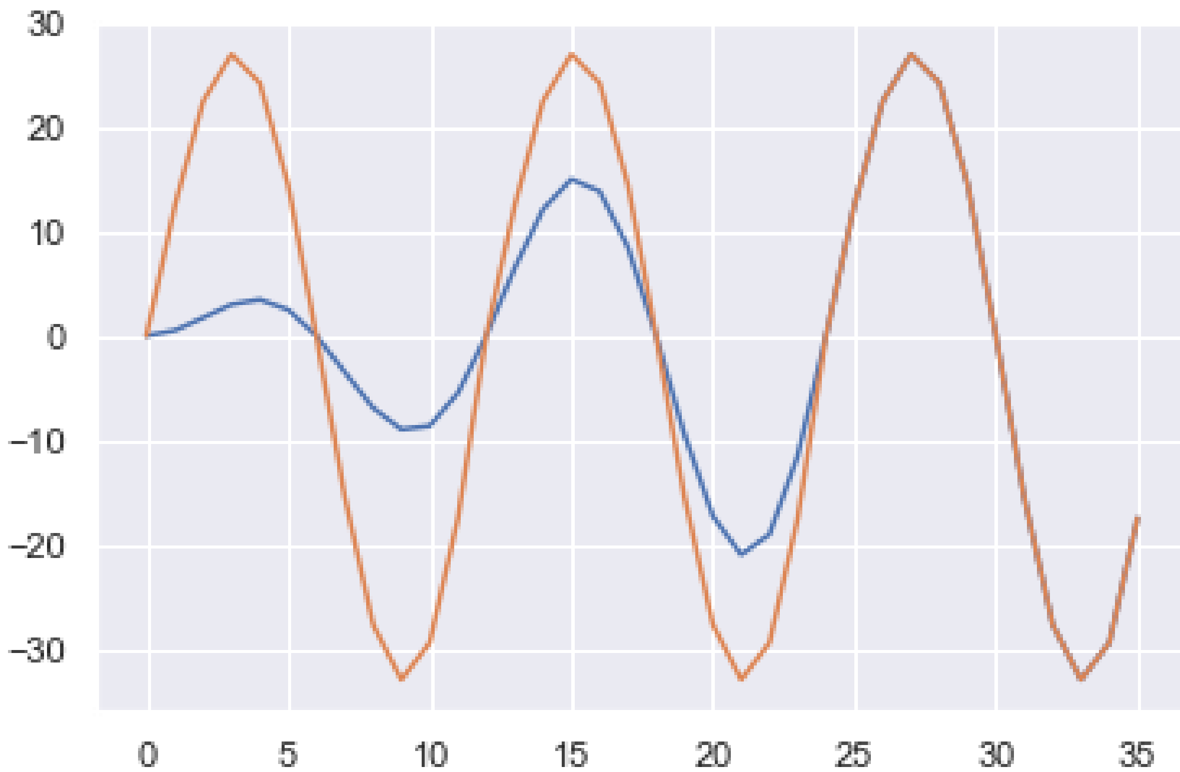
$$y(t) = y(t - t_s),$$

где t_s s лаг (задержанное значение) временного ряда, где s – это период.

```
def snaive_forecast(x_ts, season_period, n_predict):
    forecast = np.zeros(n_predict)

    for i in range (min(n_predict,season_period)):
        forecast[i] = x_ts[-season_period+i]

    if n_predict>season_period:
        for i in range (n_predict-season_period):
            forecast[i+season_period] = forecast[i]

    return forecast
```

Давайте протестируем нашу функцию

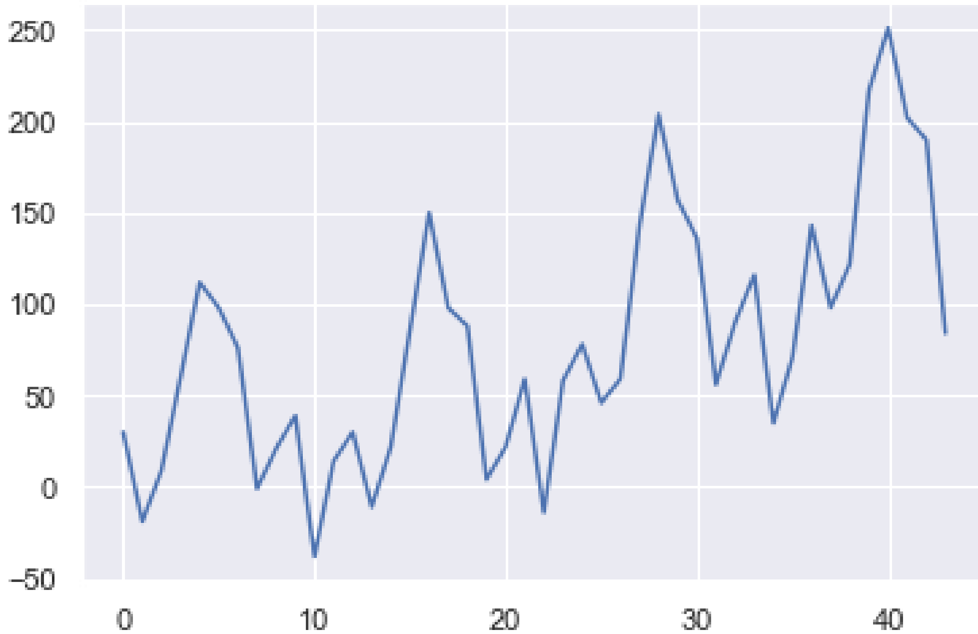
```
x =np.arange(36)* np.sin(2*np.pi*np.arange(36)/12)
plt.plot(x)
x_also = snaive_forecast(x, season_period=12, n_predict=36)
plt.plot(x_also)
```



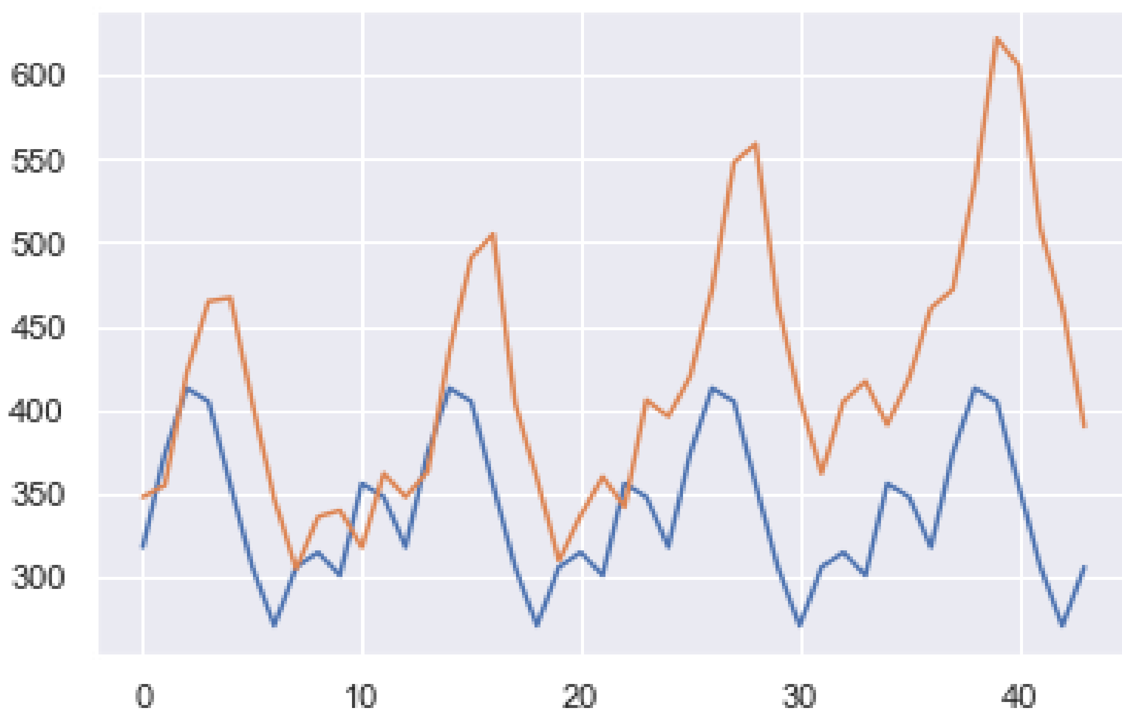

```
n_preidctions = x_test.size

season_period = 12

predicted = snaive_forecast(x_train,season_period,n_preidctions)

residuals = forecast_residual(predicted,y_test)

rmse = np.sqrt(np.sum(np.power(residuals,2)))

print('Test RMSE: %.3f' % rmse)

residuals = pd.DataFrame(residuals)

print(residuals.describe())

plt.plot(residuals)

plt.show()
```

```
Test RMSE: 692.528
                0
count   44.000000
mean    79.340909
std     68.643990
min    -38.000000
25%     28.000000
```

```
50%       73.500000
75%      117.500000
max      251.000000
```




```
plt.plot(predicted)
plt.plot(y_test);
```



Как мы видим предсказание несколько лучше.

Упражнение 4

1. Постройте графики результатов прогноза и проведите их вместе.
2. Проверьте остатки наивного прогноза с ранее показанным графическим анализом (ACF, PACF, Q-Q, Hist, p-значения Ljung-Box).
3. Реализуйте прогноз на основескользящего среднего, используя $y(t) = \frac{1}{M}\sum_{i=0}^{M-1} y\,(t-i)$ и сделай сравнение с наивным прогнозом.

Exercise 5

1. Сделайте прогноз для одно из полученных выше результатов разложения на сезон и тренд (можно наивный прогноз).
2. Проверьте остатки прогноза при помощи изученных тестов (ACF,PACF, Q-Q, Hist, Ljung-Box p-values, ADF).

Сглаживающие предсказания

Помимо простого скользящего среднего, может выполняться экспоненциальное сглаживание.

```
from statsmodels.tsa.holtwinters import (SimpleExpSmoothing, # SEMA
                                         Holt,# DEMA
                                         ExponentialSmoothing) # TEMA
```

Single Exponential Smoothing (экспоненциальное скользящее среднее, **SEMA**):

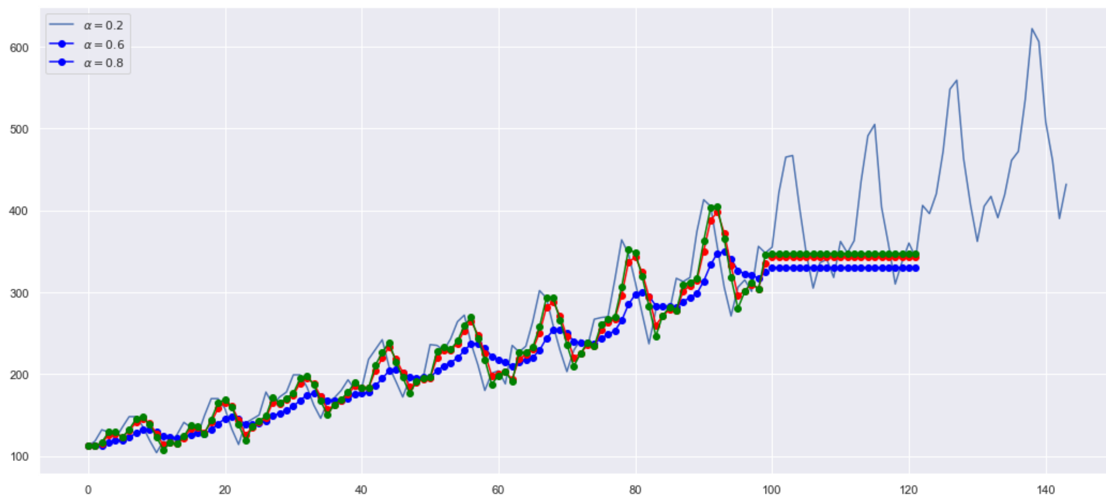
$$\hat{y}_0 = y_0;$$

$$\hat{y}_n = \alpha y_n + (1-\alpha)\hat{y}_{n-1},$$

где α параметр сглаживания; $\hat{y}$ предсказанное значение.

```
n_predict = x_test.size//2

x_train = pd.DataFrame(x_train)
plt.figure(figsize=(18,8))

plt.plot(airpass.data.values, label='groud')

# Simple Exponential Smoothing
fit1 =
SimpleExpSmoothing(x_train).fit(smoothing_level=0.2,optimized=False)

fcast1 = fit1.forecast(n_predict).rename(r'$\alpha=0.2$')
# plot
fcast1.plot(marker='o', color='blue', legend=True)
fit1.fittedvalues.plot(marker='o',  color='blue')



fit2 =
SimpleExpSmoothing(x_train).fit(smoothing_level=0.6,optimized=False)
```

```
fcast2 = fit2.forecast(n_predict).rename(r'$\alpha=0.6$')
# plot
fcast2.plot(marker='o', color='red', legend=True)
fit2.fittedvalues.plot(marker='o', color='red')


fit3 = 
SimpleExpSmoothing(x_train).fit(smoothing_level=0.8,optimized=False)
fcast3 = 
fit3.forecast(n_predict).rename(r'$\alpha=%s$'%fit3.model.params['smoothin
g_level'])
# plot
fcast3.plot(marker='o', color='green', legend=True)
fit3.fittedvalues.plot(marker='o', color='green')

plt.show()
```



Double Exponential Smoothing (двойное экспоненциальное сглаживание, **DEMA, Holt smoothing, Сглаживание Холта**):

$$b_0 = y_1 - y_0; \rightarrow \ trend$$

$$l_0 = y_0; \rightarrow \ level$$

$$l_n = \alpha \, y_n + (1-\alpha)(l_{n-1} + b_{n-1});$$

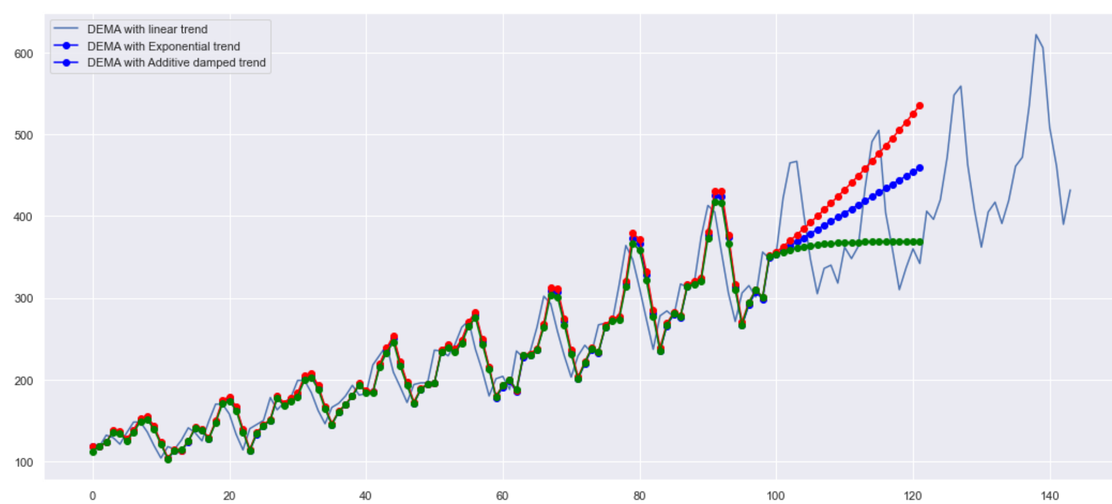
$$b_n = \beta(l_n - l_{n-1}) + (1-\beta)b_{n-1};$$

$$\hat{y}_{n+1} = l_n + b_n$$

где β дополнительный параметр сглаживания.

```
n_predict = x_test.size//2

x_train = pd.DataFrame(x_train)
plt.figure(figsize=(18,8))
```

```python
plt.plot(airpass.data.values, label='groud')


fit1 = Holt(x_train).fit(smoothing_level=0.8, smoothing_trend=0.2,
optimized=False)
fcast1 = fit1.forecast(n_predict).rename("DEMA with linear trend")

fit2 = Holt(x_train, exponential=True).fit(smoothing_level=0.8,
smoothing_trend=0.2, optimized=False)
fcast2 = fit2.forecast(n_predict).rename("DEMA with Exponential trend")

fit3 = Holt(x_train, damped_trend=True).fit(smoothing_level=0.8,
smoothing_trend=0.2)
fcast3 = fit3.forecast(n_predict).rename("DEMA with Additive damped
trend")


fit1.fittedvalues.plot(marker="o", color='blue')
fcast1.plot(color='blue', marker="o", legend=True)

fit2.fittedvalues.plot(marker="o", color='red')
fcast2.plot(color='red', marker="o", legend=True)

fit3.fittedvalues.plot(marker="o", color='green')
fcast3.plot(color='green', marker="o", legend=True)

plt.show()
```



Triple Exponential Smoothing (ог тройное экспоненциальное сглаживание, **TEMA**, **Holt-Winters, Холт-Винтер сглаживание, HW**):

$$b_0 = y_1 - y_0; \rightarrow trend$$

$$l_0 = y_0; \rightarrow level$$

$$s_0 = \sum_{i=0}^{L-1} (y_{L+i} - y_i)/L^2 \,; \to seasonality$$

$$l\_n \ = \ \alpha(y\_n - s_{n-L} \ + \ (1-\alpha)(l_{n-1} + b_{n-1});$$

$$b_n = \beta \ (l_n - l_{n-1}) + \ (1-\beta)b_{n-1};$$

$$s_n = \gamma(y_n - l_n) + (1-\gamma)s_{n-L};$$

$$\hat{y}_{n+m} \ = l_n + mb_n + s_{n-L+1+(m-1)modL}$$

где

- γ параметр тройного сглаживания;
- L период сезонности;
- m число точек для предсказания.

Индекс $n - L + 1 + (m-1)modL$ в уравнении прогноза для TEMA - это смещение сезонных компонентов от последнего полного сезона из наблюдаемых данных (т.е. если мы прогнозируем 3-ю точку в сезоне 45 в будущем, мы не можем использовать сезонные компоненты из 44-го сезона поскольку этот сезон также является прогнозируемым - мы можем использовать только точки из наблюдаемых данных).

Для TEMA можно добавить дополнительные уравнения для оценки значений отклонения.

$$\hat{y}_{max_x} = l_{n-1} + b_{n-1} + s_{n-L} + md_{k-L},$$

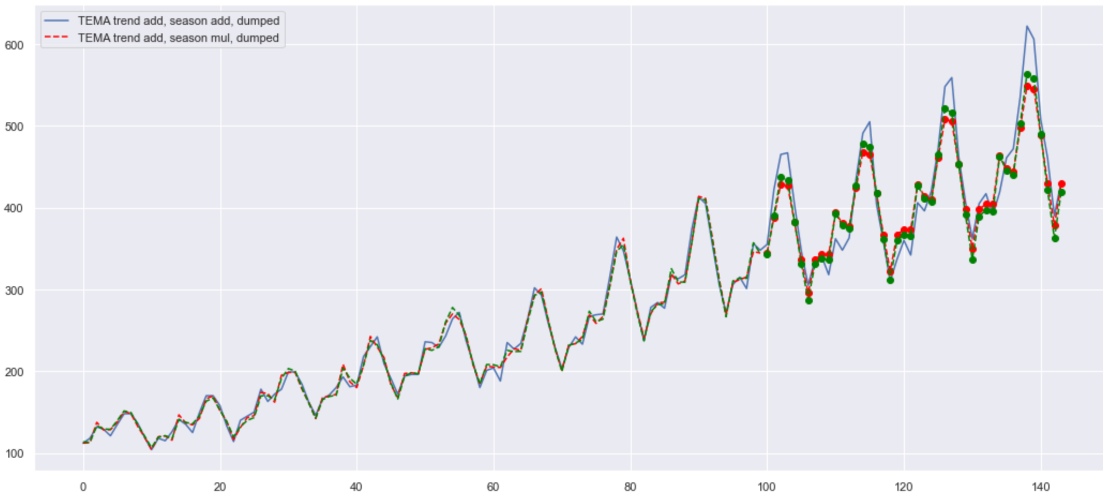
$$\hat{y}_{min_x} = l_{n-1} + b_{n-1} + s_{n-L} - md_{k-L},$$

$$d_k = \gamma \mid y_k - \widehat{y_k} \mid + (1-\gamma)d_{k-L},$$

где d ожидаемое отклонение.

```
n_predict = x_test.size

x_train = pd.DataFrame(x_train)


fit1 = ExponentialSmoothing(x_train, seasonal_periods=12, trend='add', seasonal='add').fit(use_boxcox=True)
fit2 = ExponentialSmoothing(x_train, seasonal_periods=12, trend='add', seasonal='mul').fit(use_boxcox=True)

fit3 = ExponentialSmoothing(x_train, seasonal_periods=12, trend='add', seasonal='add', damped_trend=True).fit(use_boxcox=True)
fit4 = ExponentialSmoothing(x_train, seasonal_periods=12, trend='add', seasonal='mul', damped_trend=True).fit(use_boxcox=True)

plt.figure(figsize=(18,8))

plt.plot(airpass.data.values, label='groud')
```

```
fit1.fittedvalues.plot(style='--', color='red' )
fit2.fittedvalues.plot(style='--', color='green', label='trend add, season
mul')

fit1.forecast(n_predict).rename("TEMA trend add, season
add").plot(style='--', marker='o', color='red', legend=True)
fit2.forecast(n_predict).rename("TEMA trend add, season
mul").plot(style='--', marker='o', color='green', legend=True)

plt.show()

plt.figure(figsize=(18,8))

plt.plot(airpass.data.values, label='groud')

fit3.fittedvalues.plot(style='--', color='red')
fit4.fittedvalues.plot(style='--', color='green')

fit3.forecast(n_predict).rename("TEMA trend add, season add,
dumped").plot(style='--', marker='o', color='red', legend=True)
fit4.forecast(n_predict).rename("TEMA trend add, season mul,
dumped").plot(style='--', marker='o', color='green', legend=True)

plt.show()
```



Упражнение 6

1. Проанализируйте остатки прогнозов SEMA, DEMA и TEMA, как в предыдущем упражнении.