In [311…
```
import pandas as pd; import numpy as np; import matplotlib.pyplot as plt;
from statsmodels.tsa.arima.model import ARIMA;
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.stattools import kpss;
from statsmodels.graphics.tsaplots import plot_acf;
from statsmodels.graphics.tsaplots import plot_pacf;
import warnings as yapping; yapping.simplefilter('ignore')
from scipy.interpolate import CubicSpline
import scipy.stats as st
from scipy.stats import t
import os

data_dir = './data'
```

Вариант 3

Задание 1

Известны статистические данные о поквартальной динамике оборота розничной торговли региона, приведенные в файле **region_oborot**.

(а) Проверить наличие сезонных колебаний в исходном временном ряду при помощи автокорреляционного анализа. (0,5 балла)

(б) Построить аддитивную (для четных вариантов) или мультипликативную (для нечетных вариантов) модели временного ряда. (1 балл)

(в) По построенной модели выполнить точечный и интервальный прогнозы на 2024 год. (0,5 балла)

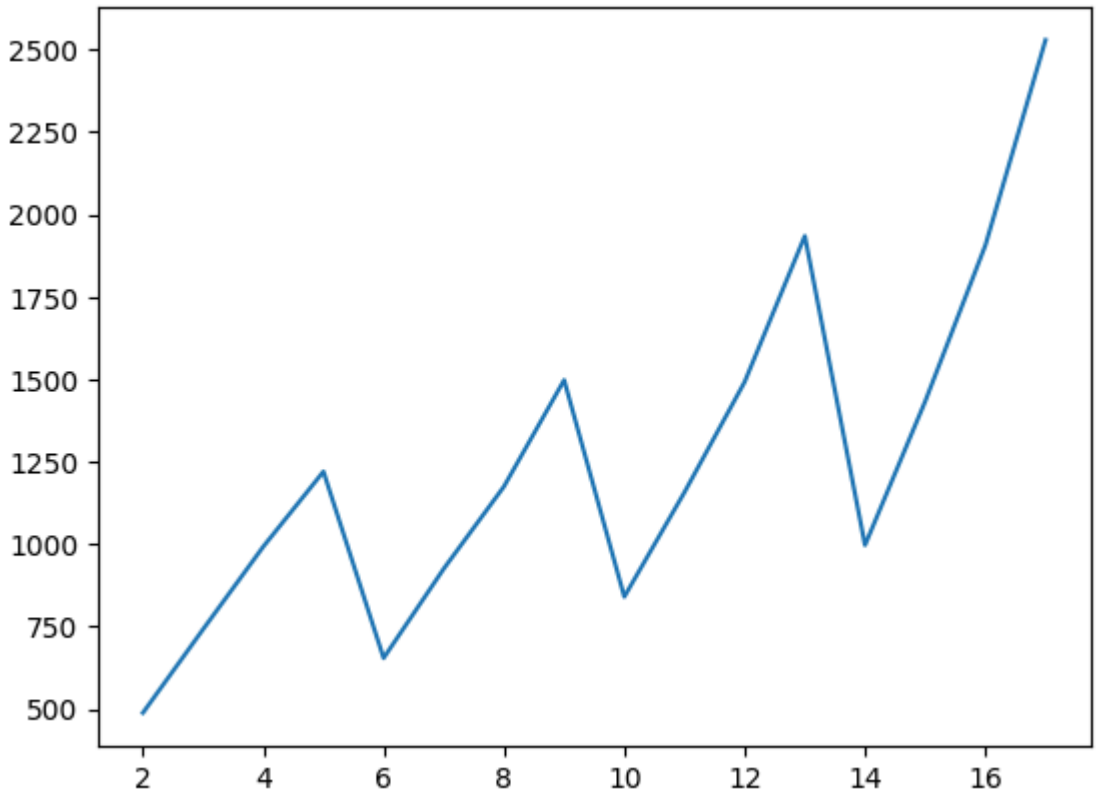
In [312…
```
data = pd.read_excel(os.path.join(data_dir, 'region_oborot.xlsx'))
data.drop([0,1],axis=0,inplace=True)
data.drop([f'Unnamed: {i}' for i in [0,3,5,6,7]],axis=1,inplace=True)
data
```

Out[312…

	Unnamed: 1	Unnamed: 2	Unnamed: 4
2	2020	I	488.4
3	NaN	II	741.73
4	NaN	III	992.31
5	NaN	IV	1219.9
6	2021	I	654.5
7	NaN	II	925.54
8	NaN	III	1174.69
9	NaN	IV	1497.32
10	2022	I	840.29
11	NaN	II	1157.31
12	NaN	III	1491.93
13	NaN	IV	1934.24
14	2023	I	996.82
15	NaN	II	1432.75
16	NaN	III	1905.75
17	NaN	IV	2528.46

In [313…
```
data.iloc[:,2].plot()
```

Out[313… <Axes: >



На графике видно сезонность

In [314…

```
vals = data.iloc[:,2].values.reshape(4,4)
origin_data = pd.DataFrame(vals, columns=['I','II','III','IV'], index=[2020+i for i in range(4)])
origin_data
```

Out[314...

	I	II	III	IV
2020	488.4	741.73	992.31	1219.9
2021	654.5	925.54	1174.69	1497.32
2022	840.29	1157.31	1491.93	1934.24
2023	996.82	1432.75	1905.75	2528.46

In [315...

```
n_new = 1
y = origin_data.values.ravel()

n_new = origin_data.shape[1] * n_new

data = pd.DataFrame(origin_data.values.ravel().reshape(-1,1),index=pd.RangeIndex(1,origin_data.values.size+1,name='t'),columns

data
```

Out[315…

t	y_t
1	488.4
2	741.73
3	992.31
4	1219.9
5	654.5
6	925.54
7	1174.69
8	1497.32
9	840.29
10	1157.31
11	1491.93
12	1934.24
13	996.82
14	1432.75
15	1905.75
16	2528.46

Для выявления структуры имеющегося временного ряда проведем автокорреляционный анализ. Пользуясь данными итоговой строки таблицы, рассчитаем коэффициенты автокорреляции 1-го, 2-го, 3-го и 4-го порядков.

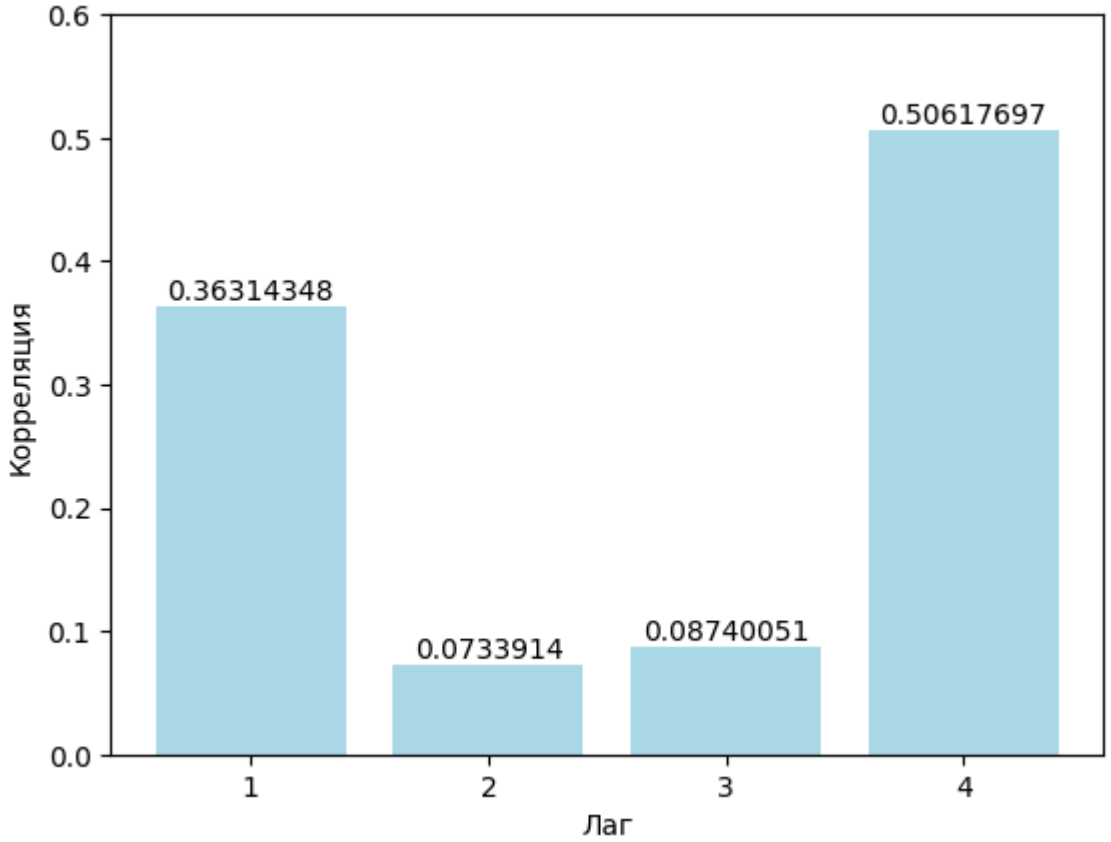
In [316…

```
data['(y_t - _y)**2'] = (data['y_t'] - data['y_t'].mean())**2
r = dict()
for i in range(1,n_new+1):
```

```
    data[f'y_t-{i}'] =data['y_t'].shift(i)
    data[f'{i}'] = (data['y_t'] - data['y_t'].mean())*(data[f'y_t-{i}'] - data['y_t'].mean())
    r[f'{i}'] = data[f'{i}'].sum()/data['(y_t - _y)**2'].sum()
```

In [317…

```
def bar_plot_dict(r:dict):
        # Данные для коррелограммы
        lags = r.keys()
        correlations = r.values()

        # Создание графика
        fig, ax = plt.subplots()
        bars = ax.bar(lags, correlations, color='lightblue')

        # Добавление значений на вершины столбцов
        for bar in bars:
            yval = bar.get_height()
            ax.text(bar.get_x() + bar.get_width()/2, yval, round(yval, 8), ha='center', va='bottom')

        # Настройка осей и заголовков
        ax.set_xlabel('Лаг')
        ax.set_ylabel('Корреляция')
        ax.set_ylim(0, 0.6)

        # Отображение графика
        plt.show()

bar_plot_dict(r)
```

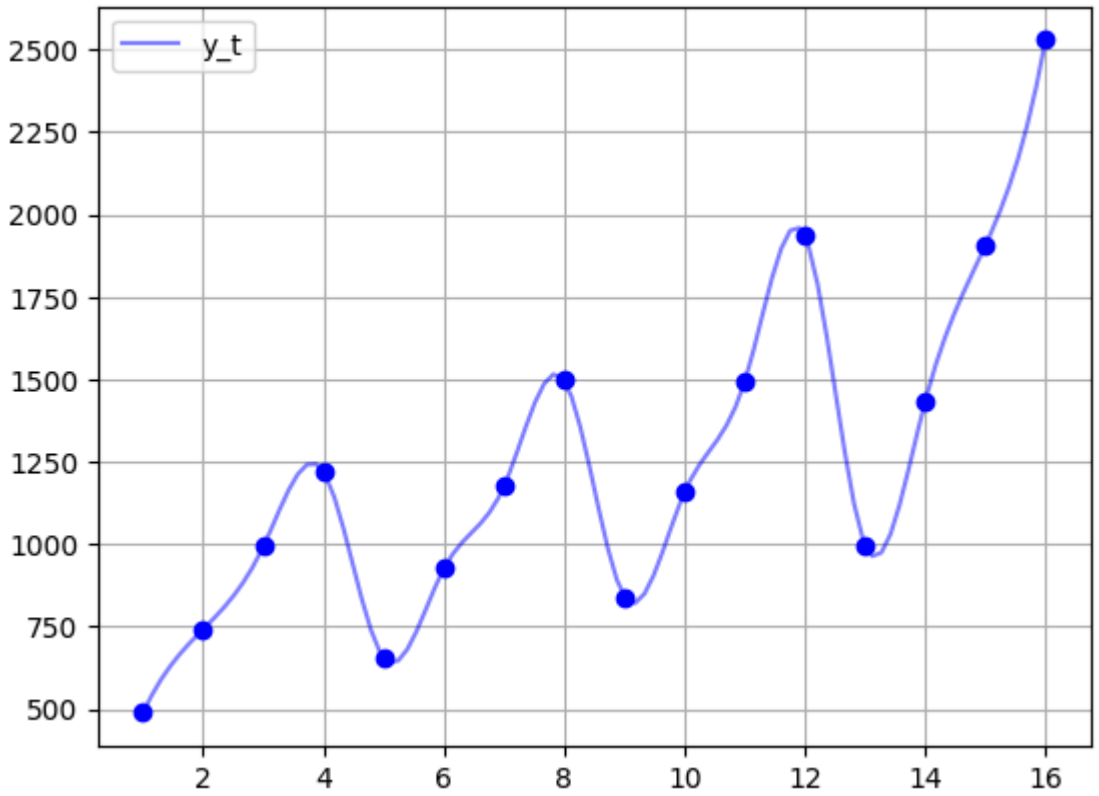


Из графика коррелограммы и значений коэффициентов автокорреляции видно, что наиболее тесная связь наблюдается при временном лаге 4 при умеренном коэффициенте автокорреляции 1-го порядка. Из этого следует, что во временном ряду наряду с тенденцией T присутствуют сезонные колебания S с периодичностью в 4 квартала, то есть характер динамики ежегодно повторяется. Данные предположения также подтверждаются графиком динамики наблюдаемых значений исследуемого показателя

In [318…

```
def smooth_plot(datas = [], indices = [], xfroms = [], markers = [],last = True,dropna = True):

        if xfroms == []:
            xfroms = [1]*len(indices)

        lines = []
```

```
        for i in range(len(indices)):
            # Данные
            if dropna:
                y = datas[i][indices[i]].dropna().values
            else:
                y = datas[i][indices[i]].values
            x = np.arange(xfroms[i], y.size+xfroms[i], dtype = float)
            # Создание сглаживающей функции
            cs = CubicSpline(x, y)

            # Генерация новых точек для плавного графика
            x_smooth = np.linspace(x.min(), x.max(), 100)
            y_smooth = cs(x_smooth)

            if len(markers):
                plt.plot(x, y, markers[i]+'o')  # Оригинальные точки
                lines.append(plt.plot(x_smooth, y_smooth, markers[i]+'-', alpha = 0.5, label = indices[i]))  # Сглаженный граф


        if last:
            plt.legend()
            plt.grid()
            plt.show()

smooth_plot([data],['y_t'],[1],['b'])
```



Из графика поквартальной динамики видно, что амплитуда колебаний постепенно увеличивается, следовательно, для моделирования такого временного ряда, целесообразнее использовать мультипликативную модель.

Произведем выравнивание исходного временного ряда с помощью скользящей средней

In [319…

```
data2 = pd.DataFrame(y.reshape(-1,1), columns=['y_t'])

k4 =[]
yc_t = []
for i in range(y.size - (n_new-1)):
    k4.append(sum([y[i+j] for j in range(n_new)]))
    yc_t.append(sum([y[i+j] for j in range(n_new)])/n_new)
```

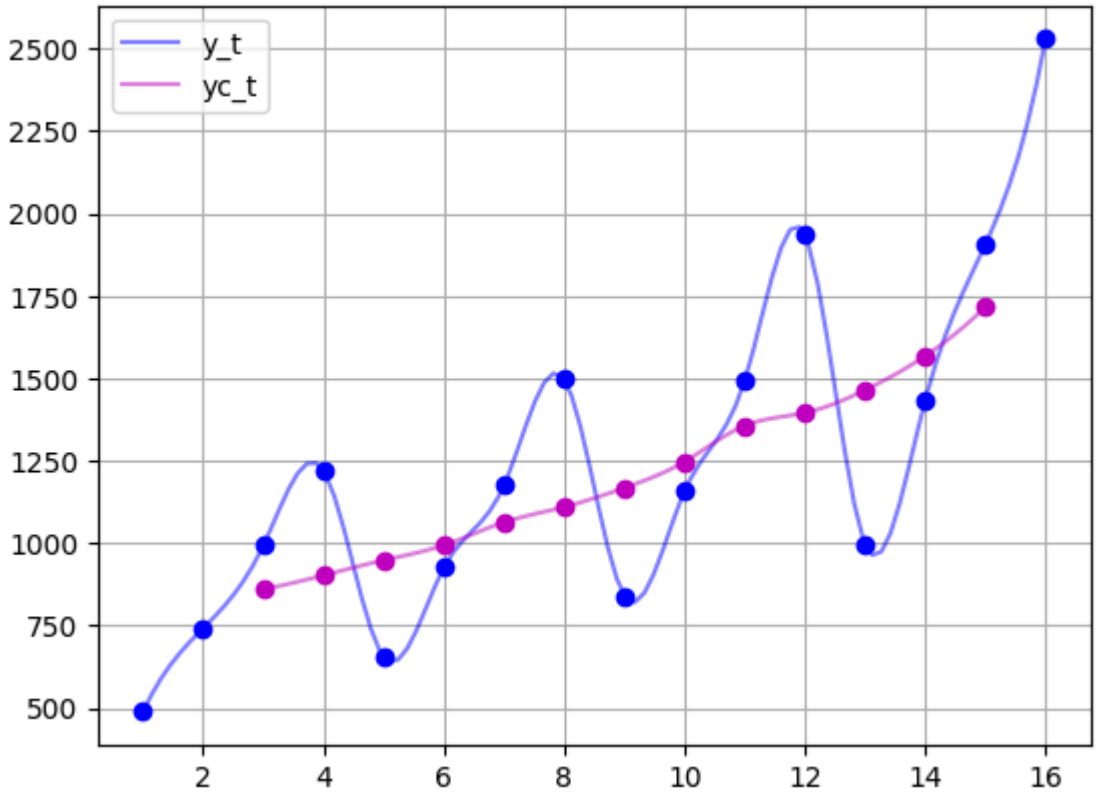
```
data2['4k'] = [pd.NA]*(n_new//2) + k4 + [pd.NA]*(n_new - n_new//2 - 1)
data2['yc_t'] = [pd.NA]*(n_new//2) + yc_t + [pd.NA]*(n_new - n_new//2 - 1)
data2['_ycc_t'] = [pd.NA]*(n_new//2) + [(data2['yc_t'][i]+data2['yc_t'][i+1])/2 for i in range(n_new//2,y.size-n_new//2 - n_ne
data2['mark'] = (data2['y_t']/data2['_ycc_t'])


data2
```

Out[319...

	y_t	4k	yc_t	_ycc_t	mark
0	488.4	<NA>	<NA>	<NA>	<NA>
1	741.73	<NA>	<NA>	<NA>	<NA>
2	992.31	3442.34	860.585	881.3475	1.125901
3	1219.9	3608.44	902.11	925.08625	1.318688
4	654.5	3792.25	948.0625	970.86	0.674145
5	925.54	3974.63	993.6575	1028.335	0.900037
6	1174.69	4252.05	1063.0125	1086.23625	1.081431
7	1497.32	4437.84	1109.46	1138.43125	1.315249
8	840.29	4669.61	1167.4025	1207.0575	0.696147
9	1157.31	4986.85	1246.7125	1301.3275	0.88933
10	1491.93	5423.77	1355.9425	1375.50875	1.084639
11	1934.24	5580.3	1395.075	1429.505	1.353084
12	996.82	5855.74	1463.935	1515.6625	0.657679
13	1432.75	6269.56	1567.39	1641.6675	0.872741
14	1905.75	6863.78	1715.945	<NA>	<NA>
15	2528.46	<NA>	<NA>	<NA>	<NA>

Произведя подобные преобразования, мы сгладили имевшиеся в исходном временном ряду t y сезонные колебания, что хорошо заметно на рисунке:

In [320…
```
smooth_plot([data,data2],['y_t','yc_t'], [1,3],['b','m'])
```



Найдем оценки сезонной компоненты как частное от деления фактических уровней ряда t y на центрированные скользящие средние сц t y. Далее рассчитаем средние значения сезонных колебаний для каждого квартала i S. Для этого перенесем имеющиеся оценки в таблицу

In [321…
```
marks = data2['mark'].fillna(0).values.reshape(*origin_data.shape).astype(float)
marks
```

Out[321…
```
array([[0.        , 0.        , 1.12590096, 1.31868785],
       [0.67414457, 0.90003744, 1.08143141, 1.31524851],
       [0.69614745, 0.88933032, 1.08463868, 1.35308376],
       [0.6576794 , 0.87274067, 0.        , 0.        ]])
```

In [322…
```
_S_i = marks.sum(axis=0)/np.count_nonzero(marks,0)
_S_i
```

Out[322…
```
array([0.67599047, 0.88736948, 1.09732369, 1.32900671])
```

Взаимопогашаемость сезонных воздействий в мультипликативной модели выражается в том, что сумма значений i S сезонной компоненты S должна быть равна числу периодов в цикле, то есть для данного примера – четырем. Суммируя i S , получаем:

In [323…
```
_S = (_S_i).sum()
_S
```

Out[323…
```
3.9896903422872896
```

Рассчитаем корректирующий коэффициент:

In [324…
```
k = origin_data.shape[0]/_S
k
```

Out[324…
```
1.0025840746594885
```

Определим скорректированные значения сезонной компоненты, умножив ее средние оценки i S на корректирующий коэффициент k:

In [325…
```
d_S_i = _S_i*k
d_S_i
```

Out[325…
```
array([0.67773728, 0.88966251, 1.10015925, 1.33244096])
```

Проверим выполнение свойства взаимопогашаемости сезонных колебаний в мультипликативной модели:

In [326…
```
d_S_i.sum()
```

Out[326…
```
4.000000000000001
```

Для выполнения последующих расчетов перенесем полученные значения S в таблицу в соответствии с каждым кварталом.

In [327…
```
data3 = pd.DataFrame(data.loc[:,'y_t'])
data3['S'] = list(d_S_i)*origin_data.shape[0]
data3
```

Out[327…

t	y_t	S
1	488.4	0.677737
2	741.73	0.889663
3	992.31	1.100159
4	1219.9	1.332441
5	654.5	0.677737
6	925.54	0.889663
7	1174.69	1.100159
8	1497.32	1.332441
9	840.29	0.677737
10	1157.31	0.889663
11	1491.93	1.100159
12	1934.24	1.332441
13	996.82	0.677737
14	1432.75	0.889663
15	1905.75	1.100159
16	2528.46	1.332441

Исключим из исходного временного ряда сезонные колебания, разделив каждый уровень t y на соответствующее значение i S. В итоге получим временной ряд, содержащий только тенденцию T и случайную компоненту E.

In [328…
```
data3['TE'] = data3['y_t']/data3['S']
data3
```

Out[328…

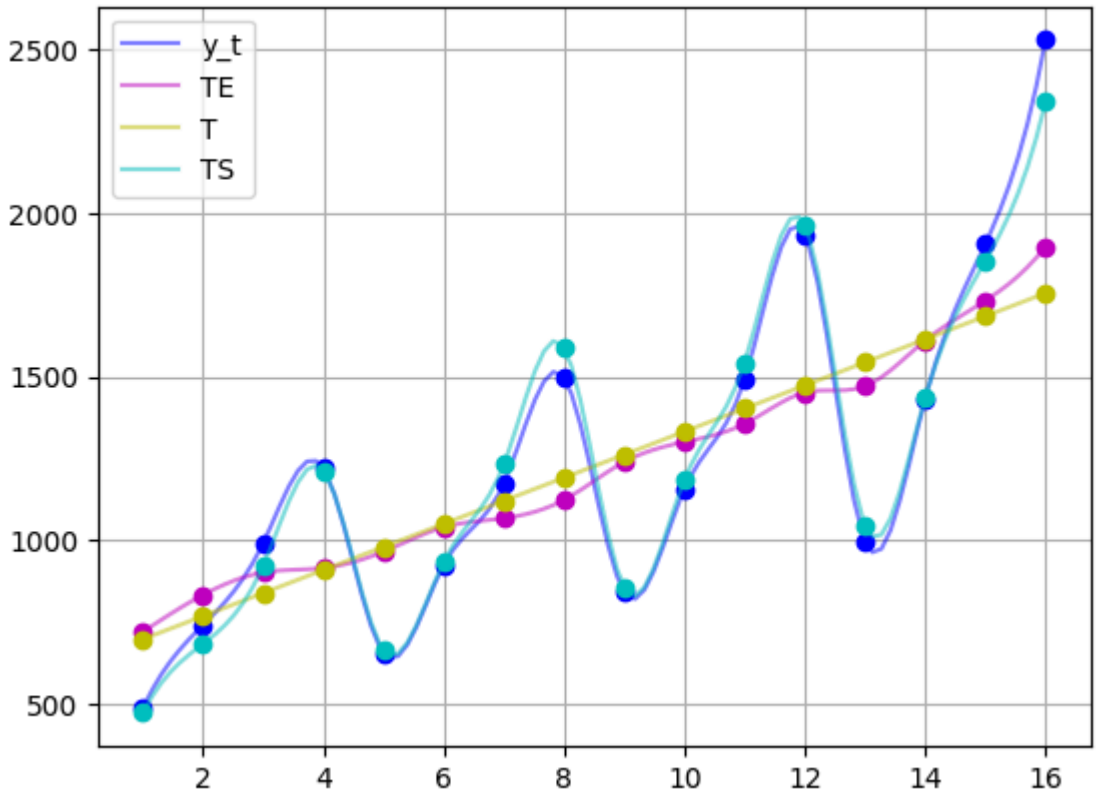
t	y_t	S	TE
1	488.4	0.677737	720.633219
2	741.73	0.889663	833.720647
3	992.31	1.100159	901.969417
4	1219.9	1.332441	915.537752
5	654.5	0.677737	965.713435
6	925.54	0.889663	1040.327083
7	1174.69	1.100159	1067.745417
8	1497.32	1.332441	1123.7421
9	840.29	0.677737	1239.846207
10	1157.31	0.889663	1300.841602
11	1491.93	1.100159	1356.10367
12	1934.24	1.332441	1451.651563
13	996.82	0.677737	1470.805908
14	1432.75	0.889663	1610.442151
15	1905.75	1.100159	1732.249214
16	2528.46	1.332441	1897.615038

Решим систему уравнений для получения коэффициентов модели

In [329…
```
from sympy import symbols, Eq,solve
b0, b1 = symbols('b0 b1')

# Запись уравнений системы
eq1 = Eq(data3.index.size * b0 + pd.Series(data3.index).sum() * b1, data3['TE'].sum())
eq2 = Eq(pd.Series(data3.index).sum() * b0 + (pd.Series(data3.index)**2).sum() * b1, (data3.index*data3['TE']).sum())

# Решение системы уравнений
solution = solve((eq1, eq2), (b0, b1))

# Вывод решения
print("b0 =", solution[b0])
print("b1 =", solution[b1])
b0 = float(solution[b0])
b1 = float(solution[b1])
```

```
b0 = 627.783343770040
b1 = 70.4736097361688
```

In [330…
```
data3['T'] = pd.Series(b0 + np.arange(0,data.shape[0] +1)*b1)
data3['TS'] = data3['T']*data3['S']
data['e_t'] =  data['y_t']-data3['TS']
smooth_plot([data3]*4,['y_t','TE','T','TS'],markers=['b','m','y','c'])
```



Таким образом, мультипликативную модель изучаемого временного ряда можно представить следующим образом:

In [331…
```
from IPython.display import Math

Math(f'y = ({b0} + {b1}' + r'\times t)\times \hat{S_i} \times e_t')
```

Out[331…
$\displaystyle y = (627.78334377004 + 70.47360973616883\times t)\times \hat{S_i} \times e_t$

Для расчета показателей качества модели значения отн t е относительной ошибки E не могут быть использованы. Поэтому для оценки качества модели необходимо рассчитать значения абсолютной ошибки по формуле Eабс=Yt - (T·S), то есть) Значения остатков t е представлены в графе 1.

In [332…

```
data4 = pd.DataFrame(data.loc[:,'e_t'])
data4['(dyt - _y_t)**2'] = (data3['TS'] - data['y_t'].mean())**2
data4['(y_t - dyt)**2'] = (data3['TS'] - data['y_t'])**2
data4['|y_t - dyt|/y_t'] = np.abs(data3['TS'] - data['y_t'])/data['y_t'] * 100
data4
```

Out[332…

t	e_t	(dyt - _y_t)**2	(y_t - dyt)**2	|y_t - dyt|/y_t
1	15.165229	6.016119e+05	229.984182	3.105084
2	57.819241	3.191804e+05	3343.064582	7.795187
3	69.051764	1.060238e+05	4768.146167	6.958689
4	7.808063	1.352718e+03	60.965855	0.640058
5	-9.785142	3.417409e+05	95.749	1.495056
6	-9.161672	9.870252e+04	83.936241	0.989873
7	-58.697011	2.397617e+02	3445.339079	4.996809
8	-90.379633	1.148047e+05	8168.478041	6.036093
9	-15.045513	1.548704e+05	226.367462	1.790514
10	-28.182585	4.016855e+03	794.258121	2.43518
11	-51.585786	8.681540e+04	2661.093321	3.457655
12	-29.067329	5.104189e+05	844.90963	1.502778
13	-49.565884	4.100032e+04	2456.776876	4.972401
14	-3.533498	3.512335e+04	12.485611	0.246624
15	52.105439	3.657508e+05	2714.976747	2.734117
16	189.544974	1.188195e+06	35927.297317	7.496459

In [333…
```
Math(r'\hat{S_i} = ' + f'{list(d_S_i)}')
```

Out[333…
$\displaystyle \hat{S_i} = [0.6777372831706586, 0.8896625060995372, 1.1001592524505148, 1.332440958279 29]$

Предсказанные значения:

In [334…
```
def hat_y(t):
        return (b0 + b1*t)* d_S_i[t%len(d_S_i)-1]

h_y = np.vectorize(hat_y)

predicted = h_y(range(data.shape[0] +1 ,data.shape[0] +1 + n_new))
predicted
```

Out[334…
```
array([1237.43625539, 1687.07441144, 2163.77333647, 2714.52272197])
```

Рассчитаем стандартную ошибку регрессии:

In [335…
```
S_e = np.sqrt(data4['(y_t - dyt)**2'].sum()/(data.shape[0] - 1 - 1))
S_e
```

Out[335…
```
68.57416643561601
```

Стандартные ошибки прогноза будут следующими:

In [336…
```
def s_e(t):
    return S_e * np.sqrt(1 + 1/data.shape[0] + (t - np.mean(data.index))**2/np.sum((data.index - np.mean(data.index))**2))

std = np.vectorize(s_e)


errors = std(range(data.shape[0] +1 ,data.shape[0] +1 + n_new))
errors
```

Out[336…
```
array([77.43113576, 79.02235277, 80.75360649, 82.61609381])
```

Нахождение критического значения t

In [337…
```
alpha = 0.95
degrees_of_freedom = data.shape[0] - 2
t_critical = t.ppf(1 - (1 - alpha) / 2, degrees_of_freedom)

lower_boarder = (h_y(range(data.shape[0] +1 ,data.shape[0] +1 + n_new)) - std(range(data.shape[0] +1 ,data.shape[0] +1 + n_new
upper_boarder = (h_y(range(data.shape[0] +1 ,data.shape[0] +1 + n_new)) + std(range(data.shape[0] +1 ,data.shape[0] +1 + n_new

print(f"Критическое значение t_{{0.95;{degrees_of_freedom}}} = {t_critical:.2f}")
```

```
Критическое значение t_{0.95;14} = 2.14
```

Границы предсказаний

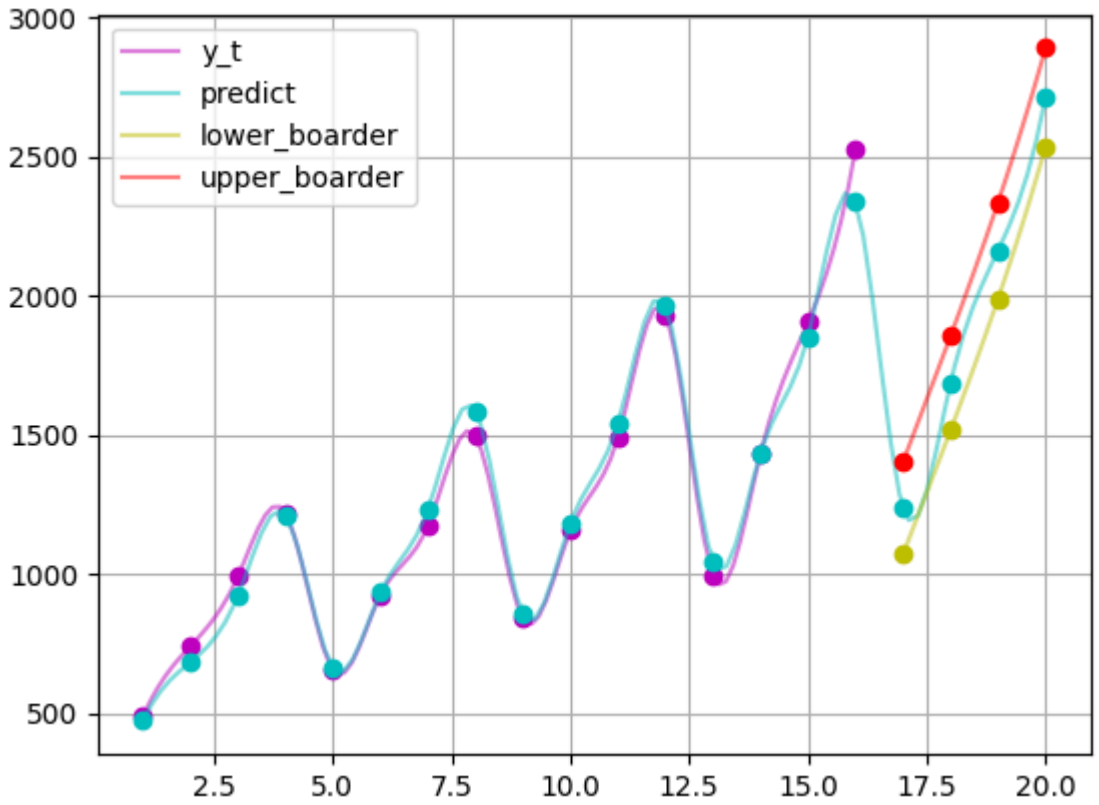
In [338…
```
data5 = pd.DataFrame(np.hstack((data3.iloc[:,4].values,predicted)),columns =['predict'])



data6 = pd.DataFrame(np.vstack((lower_boarder, upper_boarder)).T, columns=['lower_boarder','upper_boarder'])
data6
```

Out[338…

	lower_boarder	upper_boarder
0	1071.362986	1403.509525
1	1517.588321	1856.560502
2	1990.574076	2336.972597
3	2537.328824	2891.716620

Итоговый график модели

In [339…
```
smooth_plot([data3,data5,data6,data6],['y_t','predict','lower_boarder','upper_boarder'],[1,1,data.shape[0]+1,data.shape[0]+1],
```



Коэффициент детерминации

In [340…
```
R2 = data4['(dyt - _y_t)**2'].sum()/(data4['(dyt - _y_t)**2'].sum() + data4['(y_t - dyt)**2'].sum())
R2
```

Out[340…
```
0.983687062386617
```

Средняя ошибка аппроксимации

In [341…
```
A_ = data4['|y_t - dyt|/y_t'].mean()
A_
```

Out[341…
```
3.5407860092523546
```

Задание 2

На основе представленных в файле **rus_income** данных о динамике индекса реальных денежных доходов населения России:
(а) подберите ARIMA-модель, удачно описывающую динамику индекса реального дохода; (2,5 балла)
(б) на основе полученной модели постройте прогноз индекса реального дохода на полгода (2 квартала) вперед (по сравнению с имеющимися в файле данными); (2 балла)
(в) сравните прогнозные значения со значениями, наблюдавшимися на практике в рассматриваемые месяцы. (Примечание: фактическое значение индекса реальных денежных доходов населения в первом квартале 2008 г., согласно данным Росстата, составило 165,6.) (0,5 балла)

(а) подберите ARIMA-модель, удачно описывающую динамику индекса реального дохода;

In [342…

```
data = pd.read_excel(os.path.join(data_dir, 'rus_income.xlsx'), names = ['period','I'])
data.drop(list(range(3)), axis=0,inplace = True)
periods = data['period']

data.drop('period', axis=1,inplace=True)
data.reset_index(inplace=True)

data.drop('index', axis=1,inplace=True)
data
```

Out[342...

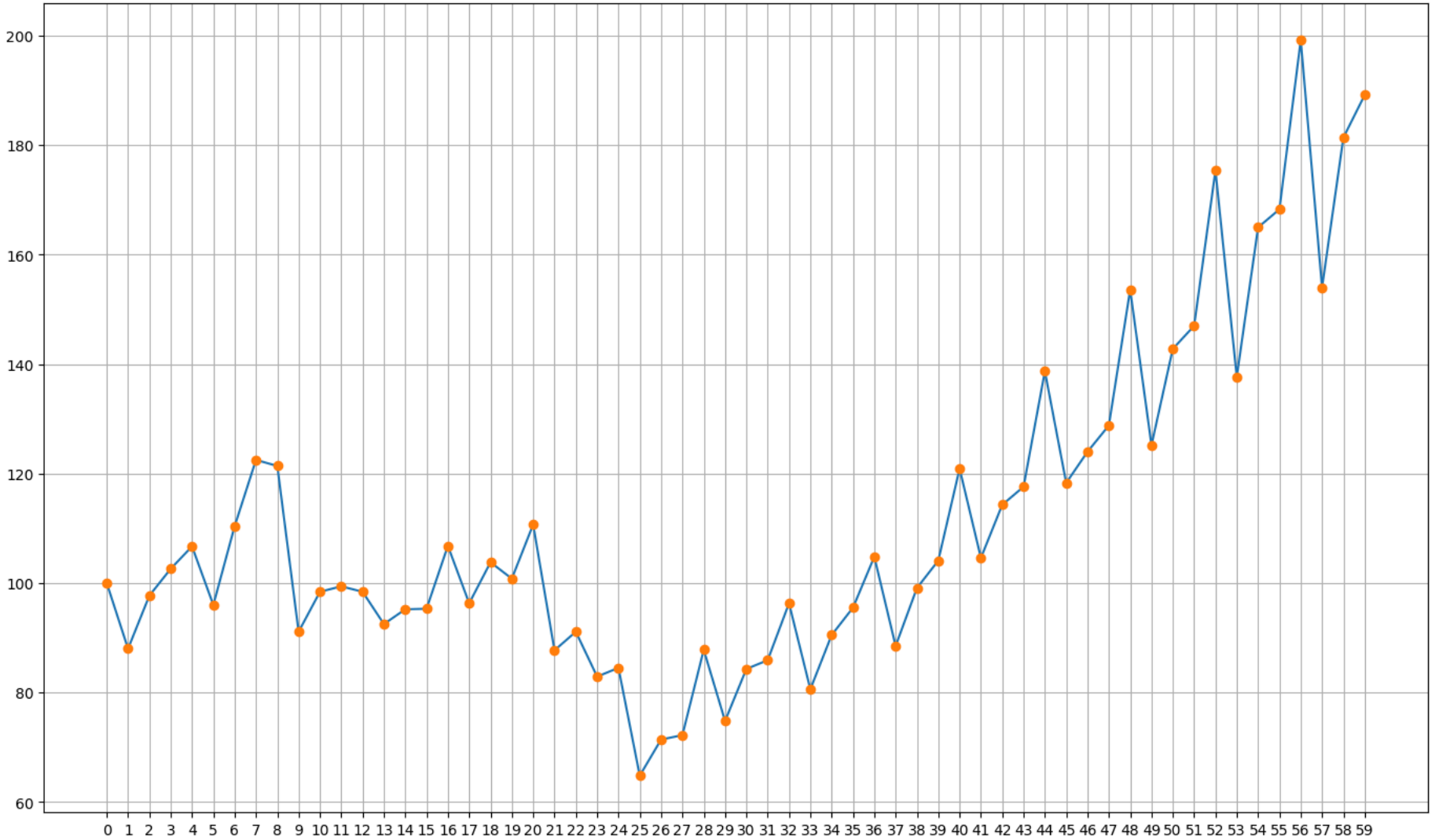
	I
0	100
1	88
2	97.7
3	102.6
4	106.7
5	96
6	110.4
7	122.5
8	121.4
9	91.1
10	98.4
11	99.4
12	98.4
13	92.5
14	95.2
15	95.3
16	106.8
17	96.3
18	103.8
19	100.8
20	110.7
21	87.7

	I
22	91.1
23	82.9
24	84.5
25	64.8
26	71.4
27	72.2
28	87.8
29	74.8
30	84.3
31	85.9
32	96.3
33	80.6
34	90.5
35	95.5
36	104.8
37	88.5
38	99.1
39	104.1
40	120.9
41	104.6
42	114.3
43	117.6

	I
44	138.7
45	118.3
46	124
47	128.8
48	153.6
49	125.2
50	142.8
51	147
52	175.4
53	137.7
54	165
55	168.3
56	199.2
57	154
58	181.4
59	189.2

In [343…

```
plt.figure(dpi = 100, figsize=(17, 10))
plt.plot(data)
plt.plot(data,'o')
plt.xticks(data.index)
plt.grid()
plt.show()
```

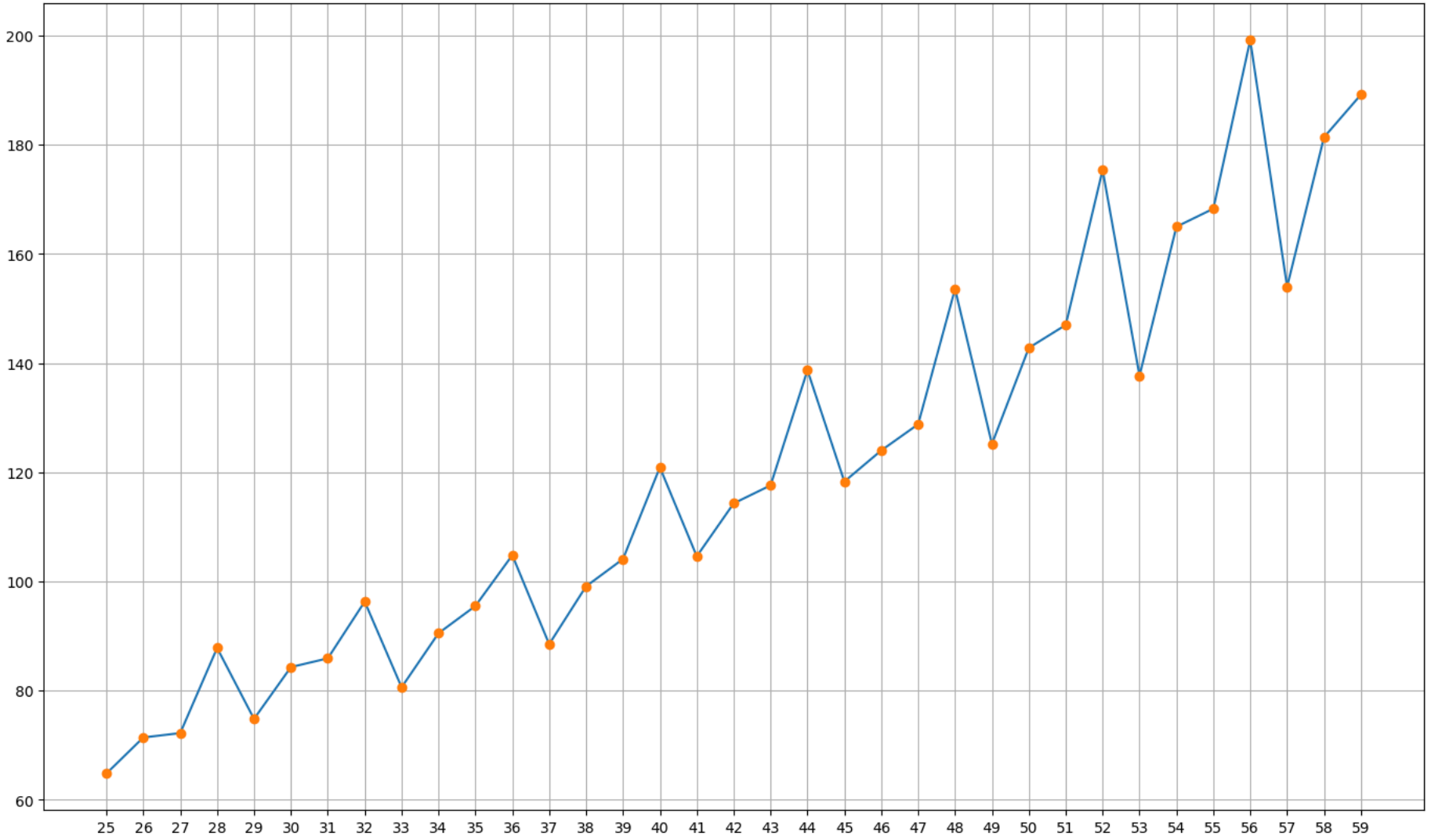


Наблюдаем неструктуризированные данные до 25го измерения. Будем строить модель с 25го измерения

In [344…

```
data.drop(list(range(25)), axis=0,inplace = True)

plt.figure(dpi = 100, figsize=(17, 10))
plt.plot(data)
plt.plot(data,'o')
```

```
plt.xticks(data.index)
plt.grid()
plt.show()
```



In [345…

```
datacopy = data.copy()

d = 0
```

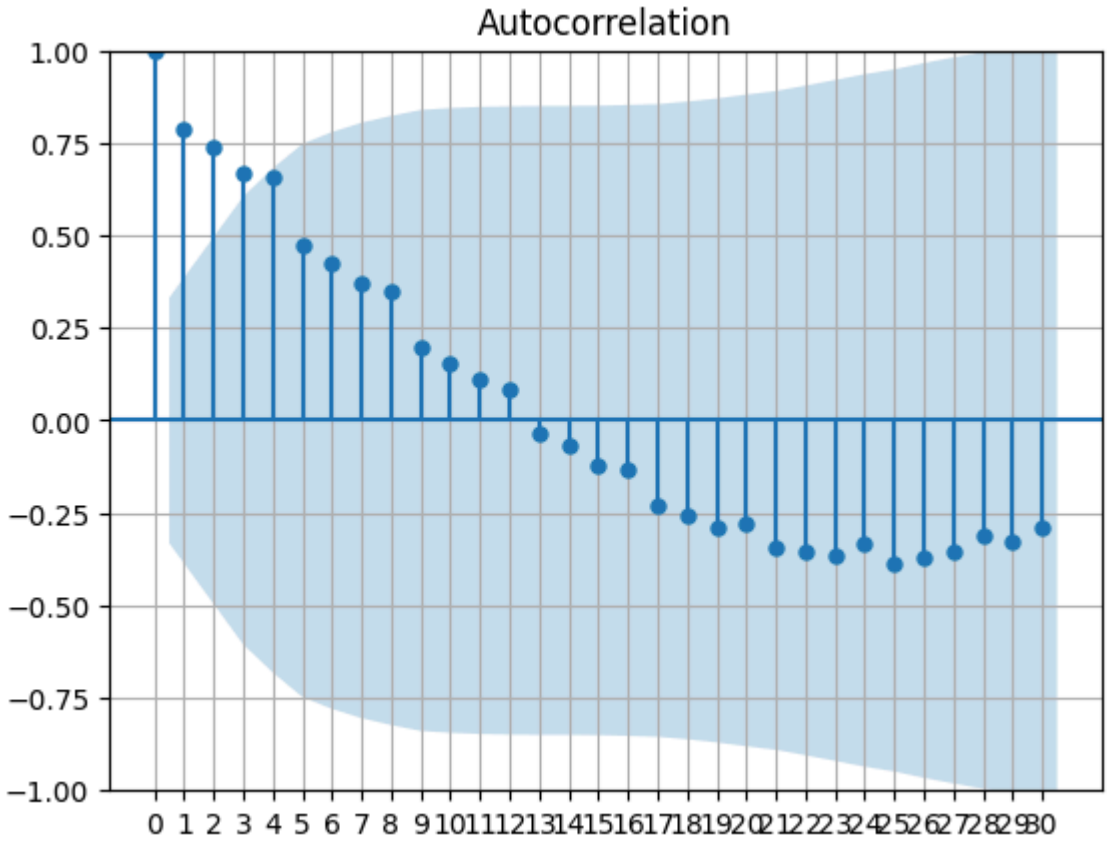
```
result = adfuller(datacopy.diff().dropna())
print(f'p-value = {result[1]}\np-value < 0.05: {result[1] < 0.05}\n')

if result[1] < 0.05:
    d = 1
else:
    d = 1
    while not result[1] < 0.05:
        datacopy = datacopy.diff()
        result = adfuller(datacopy.diff().dropna())
        print(f'p-value = {result[1]}\np-value < 0.05: {result[1] < 0.05}\n')
        d+=1

print(f'd = {d}')
```

```
p-value = 0.4011960695579423
p-value < 0.05: False

p-value = 0.01486007794604307
p-value < 0.05: True

d = 2
```

In [346…

```
plt.figure(dpi = 100, figsize=(17, 10))
plot_acf(data, lags=30, alpha=0.05)
plt.xticks(range(31))
plt.grid()
plt.show()
```

```
<Figure size 1700x1000 with 0 Axes>
```

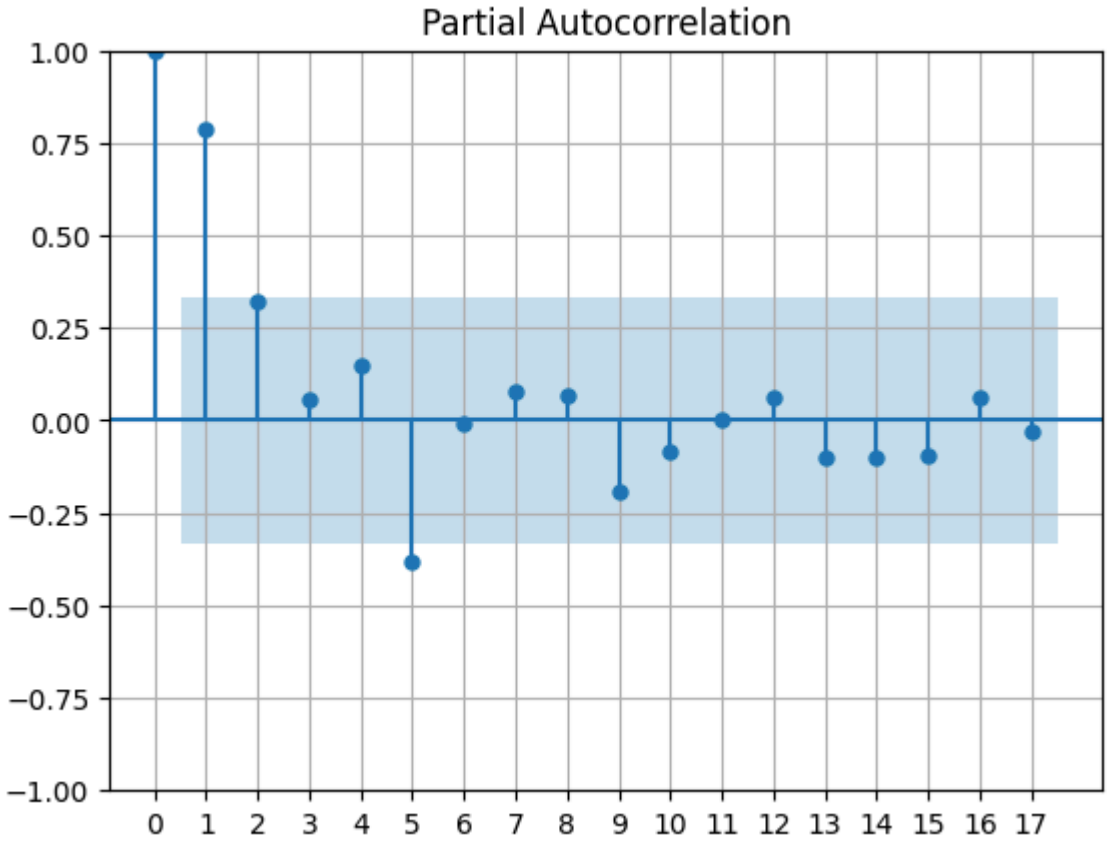


Наблюдаем значимую автокорреляцию на 3 лаге, после которой на других лагах автокорреляция незначимая и падает к 0

In [347…
```
q = 3
```

Определим p

In [348…
```
plot_pacf(data, lags=17, alpha=0.05)  # PACF (для выбора p)
plt.xticks(range(18))
plt.grid()
plt.show()
```



Значение p — это номер первого значимого лага, после которого PACF становится очень близка к нулю

In [349…
```
p = 5
```

In [350…
```
data['I'] = data['I'].astype(np.float64)
```

In [351…
```
order = (p,d,q)

arima = ARIMA(data['I'],order=order).fit()
arima.summary()
```

Out[351...

SARIMAX Results

Dep. Variable:	I	**No. Observations:**	35
Model:	ARIMA(5, 2, 3)	**Log Likelihood**	-96.990
Date:	Пт, 28 мар 2025	**AIC**	211.980
Time:	09:55:47	**BIC**	225.449
Sample:	0	**HQIC**	216.512
	- 35		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.5404	0.352	1.534	0.125	-0.150	1.231
ar.L2	-0.2778	0.204	-1.364	0.173	-0.677	0.122
ar.L3	-0.2848	0.207	-1.376	0.169	-0.690	0.121
ar.L4	0.6518	0.180	3.624	0.000	0.299	1.004
ar.L5	-0.8613	0.362	-2.379	0.017	-1.571	-0.152
ma.L1	-2.2672	0.558	-4.060	0.000	-3.362	-1.173
ma.L2	2.0856	1.029	2.028	0.043	0.070	4.101
ma.L3	-0.7225	0.446	-1.621	0.105	-1.596	0.151
sigma2	13.5448	7.129	1.900	0.057	-0.428	27.517

Ljung-Box (L1) (Q):	0.27	**Jarque-Bera (JB):**	0.85
Prob(Q):	0.60	**Prob(JB):**	0.65
Heteroskedasticity (H):	3.74	**Skew:**	0.05
Prob(H) (two-sided):	0.04	**Kurtosis:**	2.22

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

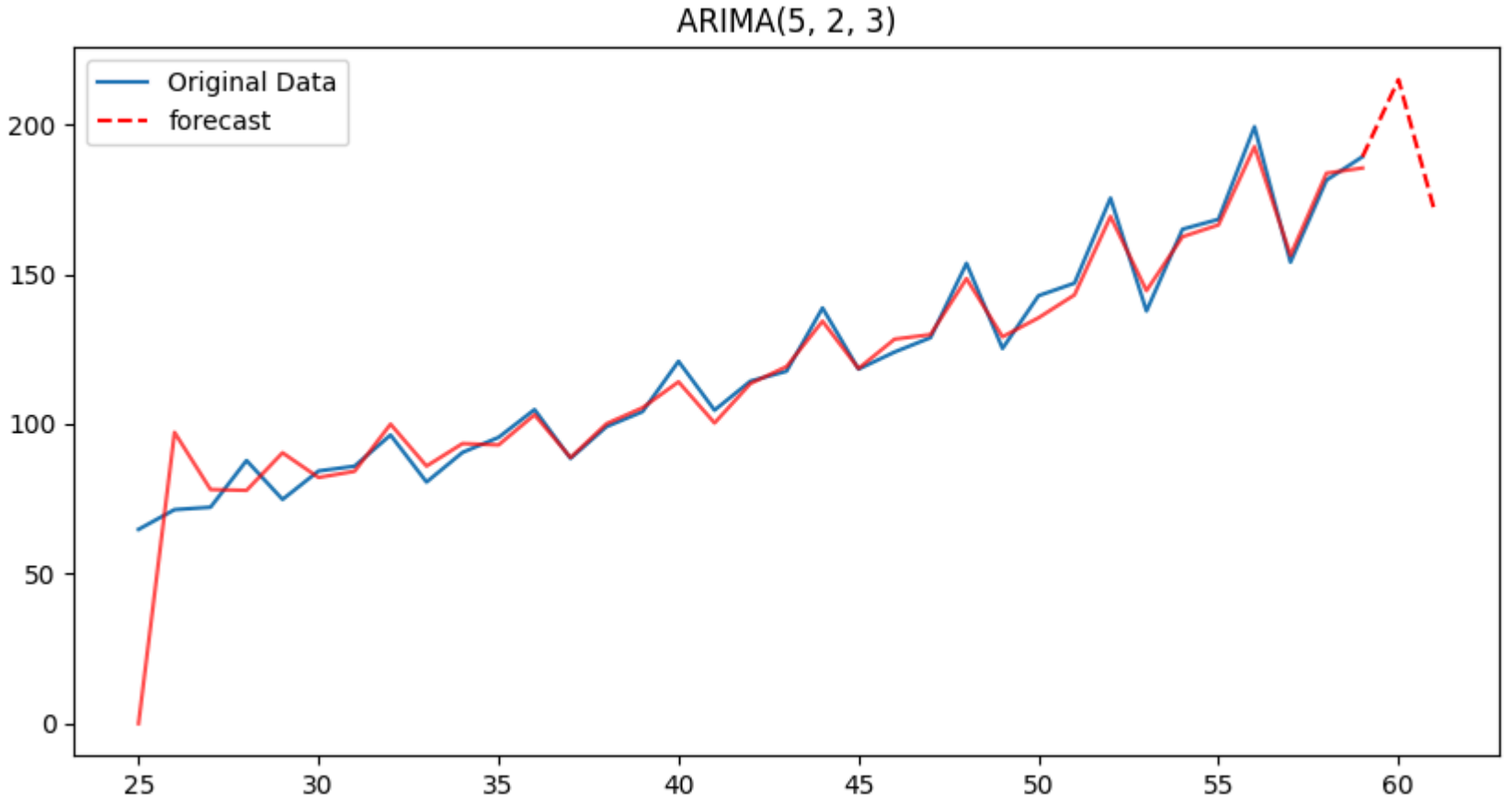
(б) на основе полученной модели постройте прогноз индекса реального дохода на полгода (2 квартала) вперед (по сравнению с имеющимися в файле данными);

прогноз на два шага

In [ ]:
```
forecast_steps = 2
forecast = arima.forecast(steps=forecast_steps)
delta = 25
```

Визуализация

In [ ]:
```
plt.figure(figsize=(10,5))
plt.plot(data, label="Original Data")
plt.plot(data.index, arima.predict(), color='red', alpha=0.7)
plt.plot(range(len(data) - 1 + delta, len(data) + forecast_steps+ delta), np.hstack([data.iloc[-1], forecast]), label="forecas
plt.legend()
plt.title(f"ARIMA{order}")
plt.show()
```



In [353…
```
forecast
```

Out[353…
```
60    214.929454
61    171.186313
Name: predicted_mean, dtype: float64
```

(в) сравните прогнозные значения со значениями, наблюдавшимися на практике в рассматриваемые месяцы. (Примечание: фактическое значение индекса реальных денежных доходов населения в первом квартале 2008 г., согласно данным Росстата, составило 165,6.)

In [360…
```
(forecast.iloc[-1] - 165.6)/165.6
```

Out[360…
```
0.03373377392870983
```

Отличия прогноза и фактических данных незначительны - всего 3 процента