

**Московский государственный технический
университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Разработка интернет приложений»

Отчет по лабораторной работе №6
«Анализ и прогнозирование временного ряда»

Выполнил:

студент группы ИУ5-62Б
Заузолков Денис

Проверил:

преподаватель каф. ИУ5
Гапанюк Ю.Е.

Москва, 2022 г.

Описание задания:

1. Выберите набор данных (датасет) для решения задачи прогнозирования временного ряда.
2. Визуализируйте временной ряд и его основные характеристики.
3. Разделите временной ряд на обучающую и тестовую выборку.
4. Произведите прогнозирование временного ряда с использованием как минимум двух методов.
5. Визуализируйте тестовую выборку и каждый из прогнозов.
6. Оцените качество прогноза в каждом случае с помощью метрик.

Лабораторная работа №6: "Анализ и прогнозирование временного ряда".

```
In [2]: import numpy as np
import pandas as pd
from matplotlib import pyplot
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
from statsmodels.tsa.arima.model import ARIMA
from sklearn.model_selection import GridSearchCV
from gplearn.genetic import SymbolicRegressor
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
```

Использован датасет, содержащий данные об изменении численности населения: <https://www.kaggle.com/datasets/census/population-time-series-data?datasetId=51748&sortBy=voteCount>

```
In [4]: data = pd.read_csv('POP.csv')
data.head()
```

```
Out[4]:
```

	realtime_start	value	date	realtime_end
0	2019-12-06	156309.0	1952-01-01	2019-12-06
1	2019-12-06	156527.0	1952-02-01	2019-12-06
2	2019-12-06	156731.0	1952-03-01	2019-12-06
3	2019-12-06	156943.0	1952-04-01	2019-12-06
4	2019-12-06	157140.0	1952-05-01	2019-12-06

Проигнорируем данные о реальном времени, поскольку мы концентрируемся только на диапазоне дат, в котором меняется население.

```
In [5]: data = data.drop(['realtime_start', 'realtime_end'], axis=1)
```

```
In [6]: """Преобразование столбца даты в объект datetime и установка его в качестве индекса"""
data['date'] = pd.to_datetime(data['date'])
data.set_index('date', inplace=True)
data.head()
```

```
Out[6]:
```

	value
date	
1952-01-01	156309.0
1952-02-01	156527.0
1952-03-01	156731.0
1952-04-01	156943.0
1952-05-01	157140.0

```
In [7]: data.describe()
```

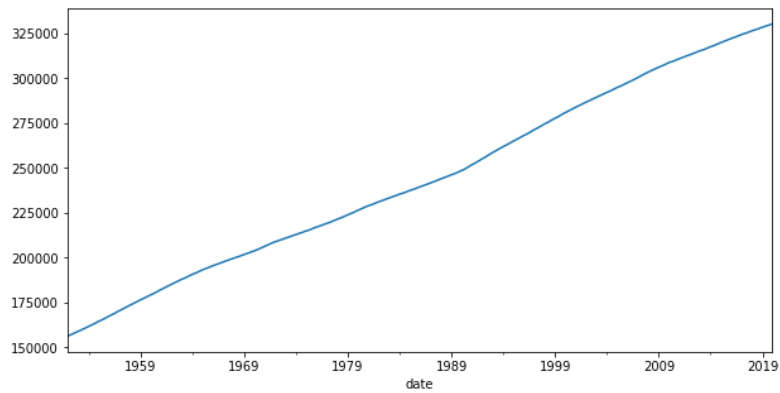
```
Out[7]:
```

	value
count	816.000000
mean	243847.767826
std	50519.140567
min	156309.000000
25%	201725.250000
50%	239557.500000
75%	289364.250000
max	330309.946000

Визуализация временного ряда

```
In [8]: fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Временной ряд в виде графика')
data.plot(ax=ax, legend=False)
pyplot.show()
```

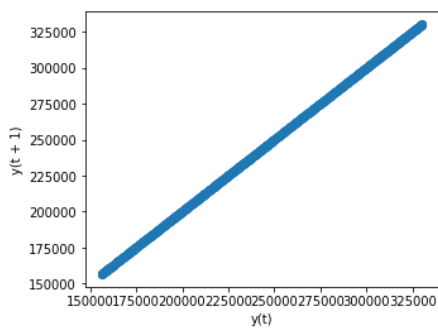
Временной ряд в виде графика



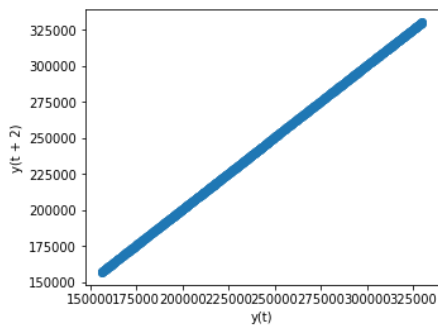
In [9]:

```
for i in range(1, 5):
    fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(5,4))
    fig.suptitle(f'Лег порядка {i}')
    pd.plotting.lag_plot(data, lag=i, ax=ax)
    pyplot.show()
```

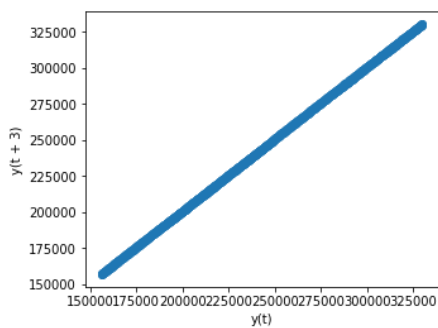
Лег порядка 1

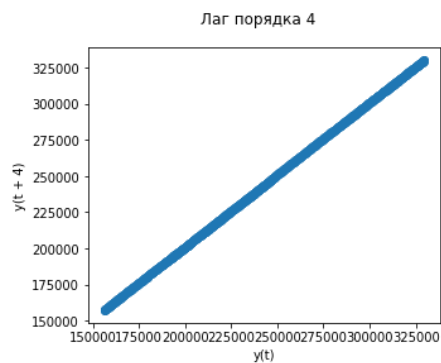


Лег порядка 2

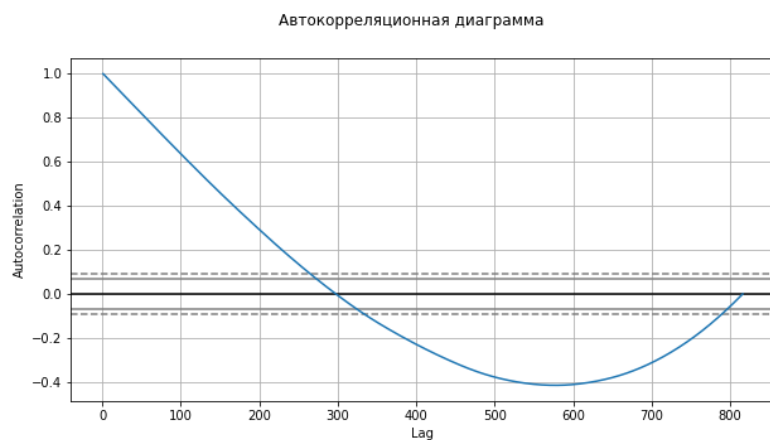


Лег порядка 3



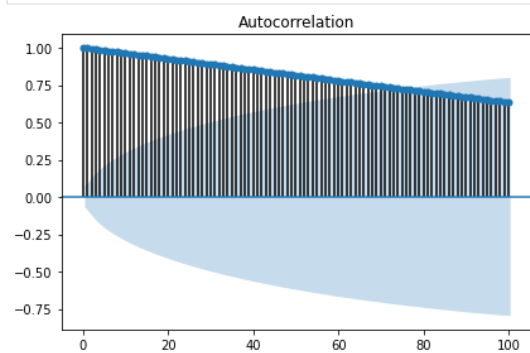


```
In [10]: fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Автокорреляционная диаграмма')
pd.plotting.autocorrelation_plot(data, ax=ax)
pyplot.show()
```



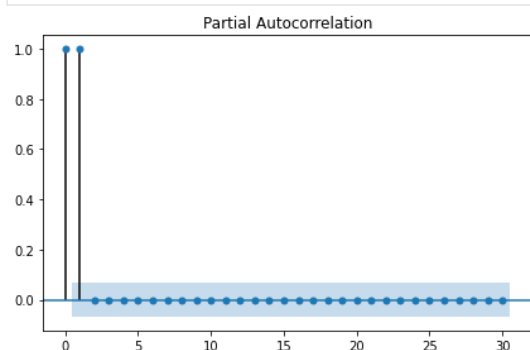
Автокорреляционная функция

```
In [42]: plot_acf(data, lags=100)
plt.tight_layout()
```



Частичная автокорреляционная функция

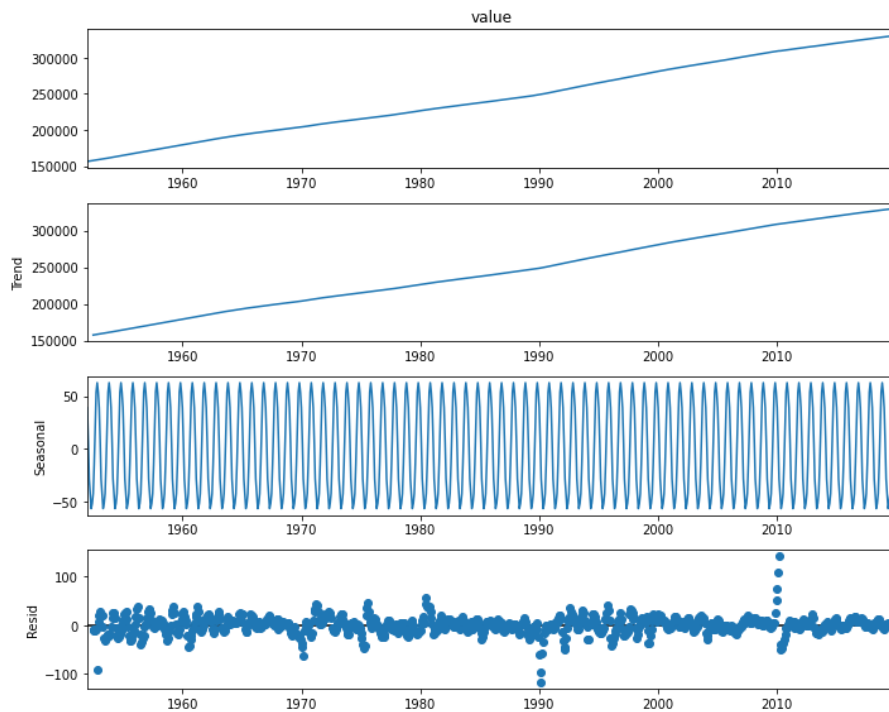
```
In [40]: plot_pacf(data, lags=30)
plt.tight_layout()
```



Декомпозиция временного ряда

```
In [14]: decomposed = seasonal_decompose(data['value'], model = 'add')
fig = decomposed.plot()
```

```
fig.set_size_inches((10, 8))
fig.tight_layout()
plt.show()
```



Наблюдается положительная динамика с 1952 по 2019 год.

Разделение временного ряда на обучающую и тестовую выборку

```
In [15]: data_2 = data.copy()
```

```
In [16]: # Целочисленная метка шкалы времени
xnum = list(range(data_2.shape[0]))
# Разделение выборки на обучающую и тестовую
Y = data_2['value'].values
train_size = int(len(Y) * 0.7)
xnum_train, xnum_test = xnum[0:train_size], xnum[train_size:]
train, test = Y[0:train_size], Y[train_size:]
history_arima = [x for x in train]
```

Прогнозирование временного ряда авторегрессионным методом (ARIMA)

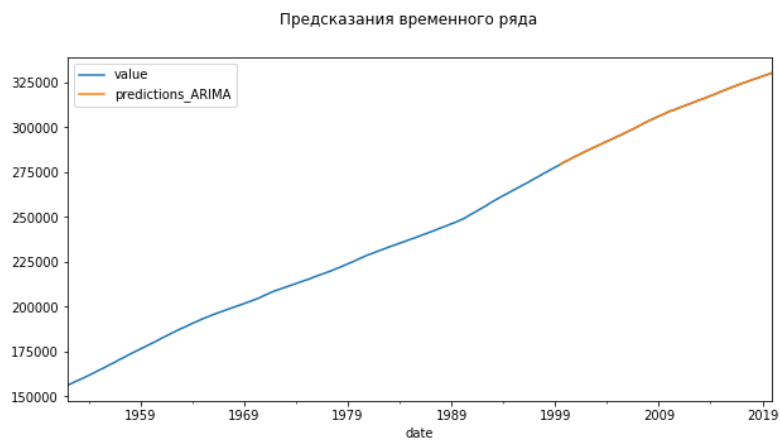
```
In [17]: # Параметры модели (p,d,q)
arima_order = (2,1,0)
# Формирование предсказаний
predictions_arima = list()
for t in range(len(test)):
    model_arima = ARIMA(history_arima, order=arima_order)
    model_arima_fit = model_arima.fit()
    yhat_arima = model_arima_fit.forecast()[0]
    predictions_arima.append(yhat_arima)
    history_arima.append(test[t])
# Вычисление метрики RMSE
error_arima = mean_squared_error(test, predictions_arima, squared=False)
```

```
In [18]: # Ошибка прогноза
np.mean(Y), error_arima
```

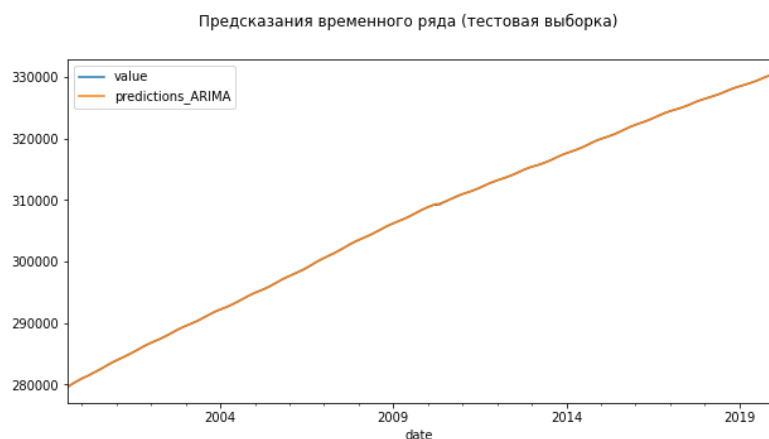
```
Out[18]: (243847.7678259804, 24.173499535797916)
```

```
In [19]: # Записываем предсказания в DataFrame
data_2['predictions_ARIMA'] = (train_size * [np.NaN]) + list(predictions_arima)
```

```
In [20]: fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда')
data_2.plot(ax=ax, legend=True)
pyplot.show()
```



```
In [21]: fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда (тестовая выборка)')
data_2[train_size:].plot(ax=ax, legend=True)
pyplot.show()
```



Прогнозирование временного ряда методом символьной регрессии

```
In [22]: function_set = ['add', 'sub', 'mul', 'div', 'sin']
SR = SymbolicRegressor(population_size=500, metric='mse',
                       generations=70, stopping_criteria=0.01,
                       init_depth=(4, 10), verbose=1, function_set=function_set,
                       const_range=(-100, 100), random_state=0)
```

```
In [23]: SR.fit(np.array(xnum_train).reshape(-1, 1), train.reshape(-1, 1))
```

/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().

y = column_or_1d(y, warn=True)

Population Average			Best Individual			
Gen	Length	Fitness	Length	Fitness	OOB Fitness	Time Left
0	263.65	2.43463e+63	23	7.14077e+09	N/A	2.77n
1	130.98	5.77055e+16	43	6.06688e+09	N/A	1.13n
2	53.10	4.58992e+15	34	3.54847e+09	N/A	39.70s
3	34.28	1.99853e+19	13	1.42699e+09	N/A	32.05s
4	35.05	2.10424e+16	38	1.04052e+09	N/A	31.71s
5	30.47	2.56729e+16	36	4.29436e+08	N/A	29.49s
6	31.30	3.00498e+16	50	6.39791e+07	N/A	30.52s
7	38.37	8.59782e+15	35	1.51165e+07	N/A	30.47s
8	43.37	5.29474e+15	47	4.76034e+06	N/A	30.80s
9	37.70	8.42452e+15	35	4.14545e+06	N/A	27.96s
10	40.68	5.69103e+15	32	3.65059e+06	N/A	30.63s
11	45.38	5.71108e+15	29	3.65015e+06	N/A	29.93s
12	41.36	5.72894e+15	29	3.65015e+06	N/A	29.92s
13	35.07	3.58233e+15	29	3.65015e+06	N/A	25.72s
14	33.33	8.46569e+15	35	3.53261e+06	N/A	25.03s
15	31.43	3.14997e+19	35	3.53261e+06	N/A	24.63s
16	30.19	1.42657e+16	35	3.53261e+06	N/A	22.80s
17	30.81	2.81228e+15	35	3.53261e+06	N/A	24.42s
18	33.31	5.72757e+15	35	3.53261e+06	N/A	23.06s
19	33.71	1.26632e+16	35	3.50395e+06	N/A	22.17s
20	34.95	1.70198e+16	35	3.50395e+06	N/A	22.53s
21	42.21	6.70957e+15	35	3.50395e+06	N/A	24.04s
22	54.68	6.78469e+15	35	3.50395e+06	N/A	24.09s
23	50.99	6.47928e+18	102	3.50387e+06	N/A	23.79s
24	42.69	8.57551e+15	71	3.50376e+06	N/A	20.67s
25	59.07	6.73374e+21	85	3.49756e+06	N/A	23.04s
26	89.07	1.51918e+25	85	3.49756e+06	N/A	26.59s
27	100.70	2.98833e+18	91	3.48956e+06	N/A	28.13s
28	120.58	7.92131e+23	91	3.48956e+06	N/A	31.64s
29	142.26	1.91023e+18	127	3.48498e+06	N/A	33.27s
30	116.37	6.9315e+21	54	3.46676e+06	N/A	35.69s

```
Out[23]: SymbolicRegressor(const_range=(-100, 100),
                             function_set=['add', 'sub', 'mul', 'div', 'sin'],
                             generations=70, init_depth=(4, 10), metric='mse',
                             population_size=500, random_state=0, stopping_criteria=0.01,
                             verbose=1)
```

[illegible]

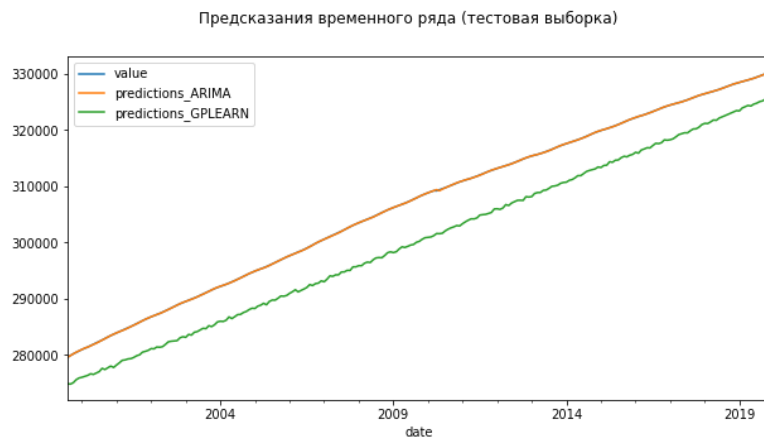

```
4), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0)), add(X0, X0))))), add(add(X0, X0), X0), sin(mul(sub(-36.019, -77.644), add(add(add(X0, X0), X0), sin(X0))))), 51.302, add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(sub(-36.019, -77.644), add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0)), add(X0, X0))), add(X0, X0))), sin(mul(sub(-36.019, -77.644), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0)), add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(add(add(X0, sub(mul(sub(-36.019, -77.644), add(X0, X0)), sub(add(X0, X0), mul(55.353, mul(sub(-36.019, -77.644), add(add(add(X0, add(add(add(X0, X0), X0), X0)), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0)), sin(X0))))), 51.302), add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(sin(mul(sub(-36.019, -77.644), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0)), add(X0, X0))), sin(mul(sub(-36.019, -77.644), add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0))), sin(mul(sub(-36.019, -77.644), add(add(55.353, 51.302), add(X0, X0))))), 51.302, add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(sub(-36.019, -77.644), add(X0, X0))), add(X0, X0))))), sin(mul(sub(-36.019, -77.644), add(add(add(add(X0, 51.302), add(add(X0, X0), X0)), add(X0, X0))), sin(mul(sub(-36.019, -77.644), add(-36.019, add(X0, X0))))), sub(add(add(X0, X0), X0), mul(55.353, 65.255)))
```

```
In [25]: # Предсказания
y_sr = SR.predict(np.array(xnum_test).reshape(-1, 1))
y_sr[:10]
```

```
Out[25]: array([274891.75793852, 274817.36307349, 275068.42349884, 275594.91553188,
275909.57465535, 276033.9204471 , 276192.3291504 , 276368.95663777,
276651.56236873, 276542.01774132])
```

```
In [26]: # Записываем предсказания в DataFrame
data_2['predictions_GPLEARN'] = (train_size * [np.NaN]) + list(y_sr)
```

```
In [27]: fig, ax = pyplot.subplots(1, 1, sharex='col', sharey='row', figsize=(10,5))
fig.suptitle('Предсказания временного ряда (тестовая выборка)')
data_2[train_size:].plot(ax=ax, legend=True)
pyplot.show()
```



```
In [28]: error_SR = mean_squared_error(test, y_sr, squared=False)
```

```
In [29]: # Ошибка прогноза
np.mean(Y), error_SR
```

```
Out[29]: (243847.7678259804, 6510.330169456957)
```

Качество прогноза моделей

```
In [30]: def print_metrics(y_test, y_pred):
print(f"R^2: {r2_score(y_test, y_pred)}")
print(f"MSE: {mean_squared_error(y_test, y_pred, squared=False)}")
print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
```

```
In [31]: print("ARIMA")
print_metrics(test, predictions_arima)

print("\nGPLEARN")
print_metrics(test, y_sr)
```

```
ARIMA
R^2: 0.9999973075905872
MSE: 24.173499535797916
MAE: 16.034435631401305
```

```
GPLEARN
R^2: 0.8047153645391025
MSE: 6510.330169456957
MAE: 6443.710113418146
```

Вывод: Обе модели, ARIMA и GPLEARN, показали хороший результат. Лучшей по всем используемым метрикам оказалась модель ARIMA.