Гридасов Егор

Для решения задачи я пробовал несколько подходов: LSTM, RandomForestRegressor, линейную регрессию. В итоге я пришел к выводу, что для такой задачи, где нужно предсказать до 360 следующих значений, лучше всего подходит линейная регрессия.

Для начала я обработал данные и привел их к удобному для работы виду.

```
data = pd.read_csv('DS_Sber.csv', sep=';')
data.columns = data.columns.str.lower()
data['reportdate'] = data['reportdate'].str.replace('.', '/')
data["reportdate"] = pd.to_datetime(data["reportdate"], infer_datetime_format=True)

data[250:].head(10)
```

Рисунок 1 — Преобразование данных

	reportdate	value
0	2013-12-30	3457625638
1	2013-12-31	3417092149
2	2014-01-01	3417092149
3	2014-01-02	3417092149
4	2014-01-03	3417092149
5	2014-01-04	3417092149
6	2014-01-05	3417092149
7	2014-01-06	3320846785
8	2014-01-07	3320846785
9	2014-01-08	3630283744

Рисунок 2 — Преобразованный датасет

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2111 entries, 0 to 2110
Data columns (total 2 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   reportdate  2111 non-null   datetime64[ns]
 1   value       2111 non-null   int64
dtypes: datetime64[ns](1), int64(1)
memory usage: 33.1 KB
```

Рисунок 3 — Преобразованный датасет

После я решил поближе познакомиться с данными и выяснить, есть ли выбросы. Пришел к выводу, что данные не имеют выбросок.

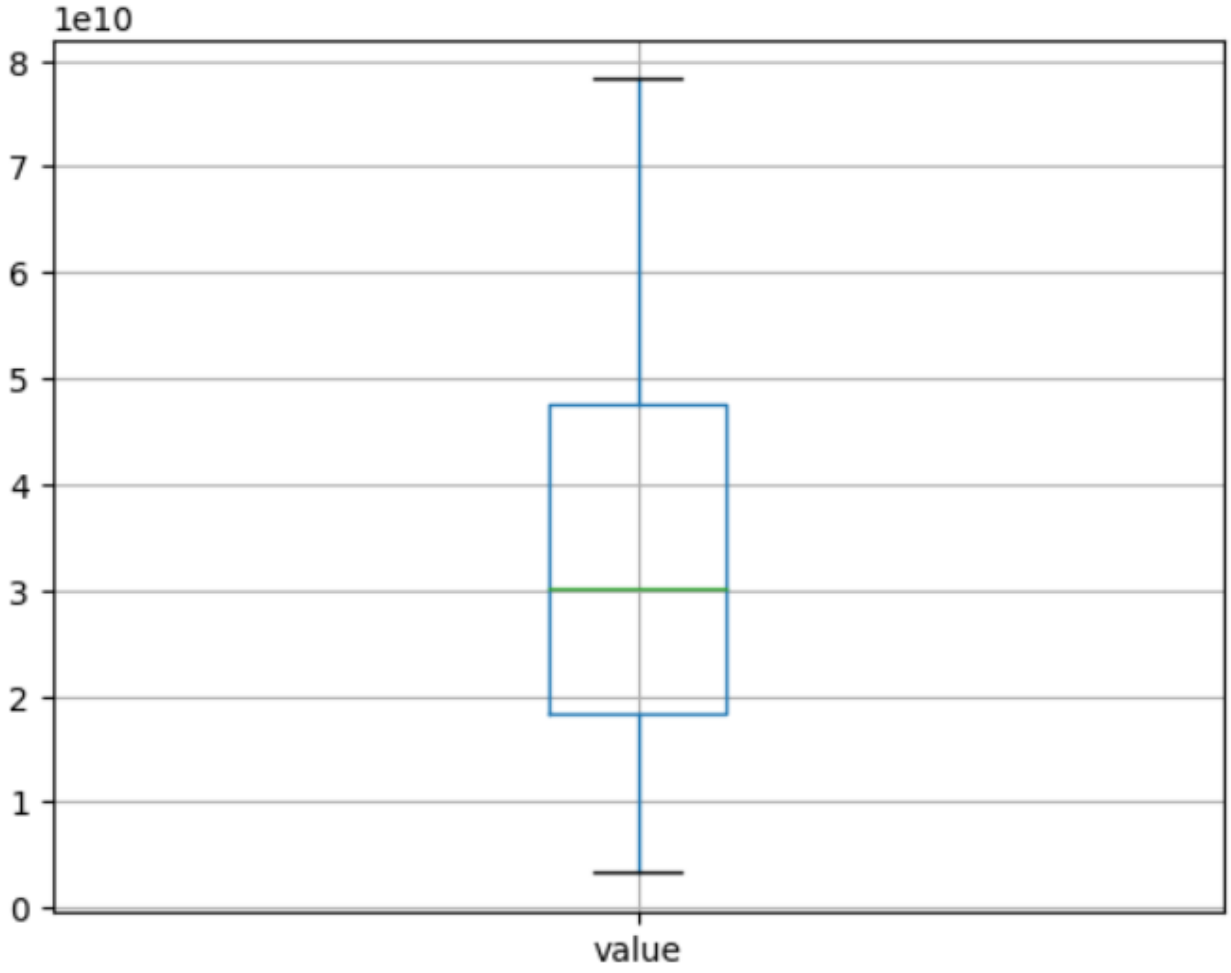

Рисунок 4 — boxplot по значениям

	value
count	2.111000e+03
mean	3.389668e+10
std	1.875474e+10
min	3.282810e+09
25%	1.824160e+10
50%	3.021352e+10
75%	4.758339e+10
max	7.822860e+10

Рисунок 5 — Сводка по данным value из датасета

Также я построил матрицу корреляций, сместив данные для предсказания на максимально возможный промежуток в рамках задачи: 360 значений. В итоге обнаружилась сильная корреляция, что очень хорошо (0.94).

	value	predict
value	1.000000	0.943179
predict	0.943179	1.000000

Рисунок 6 — Матрица корреляций

После я построил график изначальных значений, чтобы понять, как ведут себя данные с течением времени.

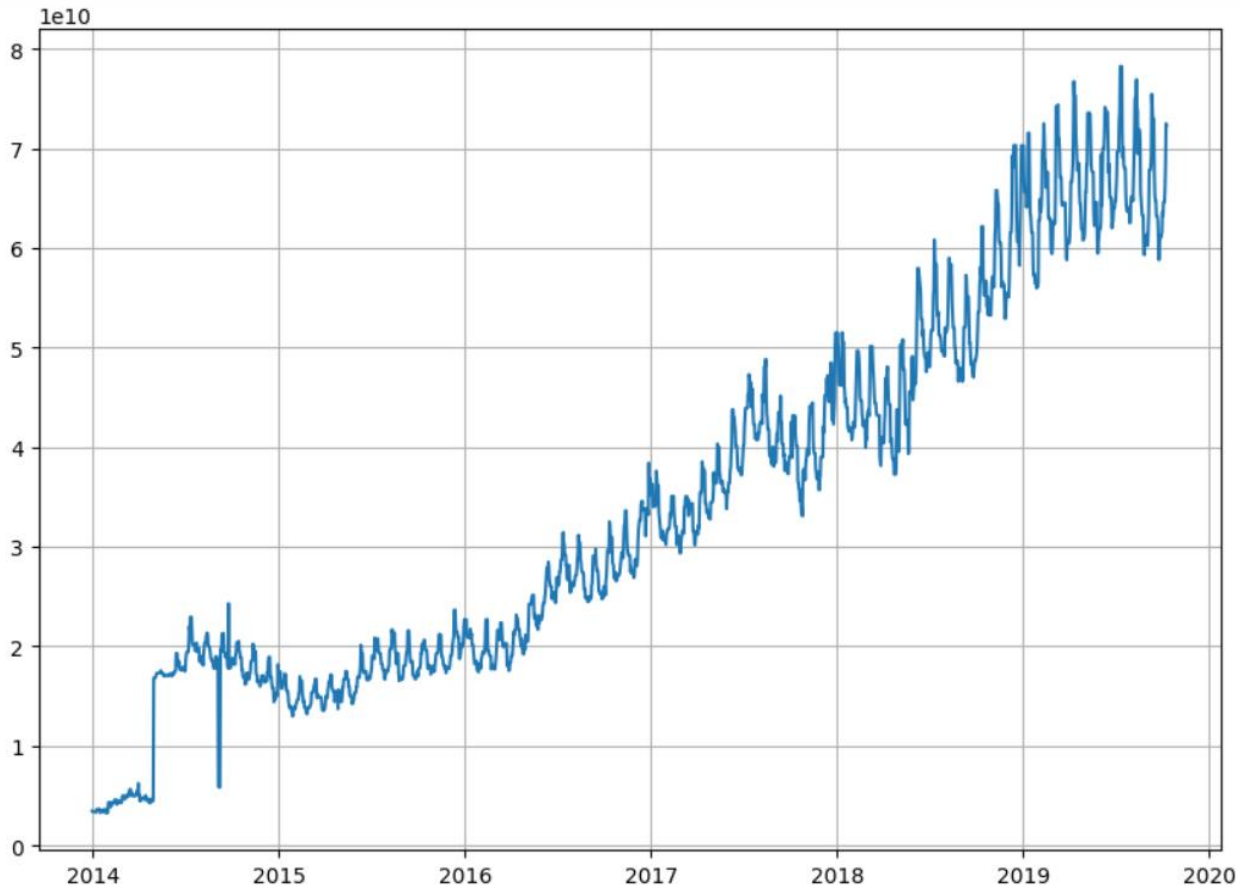

Рисунок 7 — График значений датасета

Первые 250 значений не видно закономерности, но постепенно данные обретают некоторую траекторию развития.

Для начала я попробовал построить LSTM-модель, для проверки разделив датасет и обработав данные, сместив их.

```
from sklearn.preprocessing import MinMaxScaler
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout
```

```
df = data['value'].values.reshape((-1, 1))[252:]
df
```

```
array([[19175854331],
       [19739611315],
       [19433070371],
       ...,
       [68424049766],
       [72492897583],
       [72307860851]], dtype=int64)
```

```
dataset_train = np.array(df[:int(df.shape[0]*0.8)])
dataset_test = np.array(df[int(df.shape[0]*0.8):])
```

```
scaler = MinMaxScaler(feature_range=(0, 1))
dataset_train = scaler.fit_transform(dataset_train)
dataset_test = scaler.transform(dataset_test)

dataset_train
```

```
array([[0.12949958],
       [0.14128199],
       [0.13487535],
       ...,
       [0.75412709],
       [0.76059506],
       [0.77529796]])
```

Рисунок 8 — Подготовка данных для обучения

```python
def create_dataset(df, window):
    x = []
    y = []
    for i in range(window, df.shape[0]):
        x.append(df[i-window:i, 0])
        y.append(df[i, 0])
    x = np.array(x)
    y = np.array(y)
    return (x, y)
```

Рисунок 9 — Функция для подготовки данных

```python
x_train = np.reshape(x_train, (x_train.shape[0], x_train.shape[1], 1))
x_test = np.reshape(x_test, (x_test.shape[0], x_test.shape[1], 1))

x_train.shape, y_train.shape
```

```
((1127, 360, 1), (1127,))
```

Рисунок 10 — Итоговая размерность данных

После того, как данные были подготовлены, я решил, что настало время для создания и обучения модели.

```python
def create_model():
    model = Sequential(name='Sber')
    model.add(LSTM(units=128, activation='relu', return_sequences=True, input_shape=(x_train.shape[1], 1)))
    model.add(Dropout(0.2))
    model.add(LSTM(units=128, return_sequences=True, activation='relu'))
    model.add(Dropout(0.2))
    model.add(LSTM(units=1, activation='relu'))
    model.add(Dense(units=1))

    model.compile(loss='mean_squared_error', optimizer='adam')

    return model
```

```python
model = create_model()

model.summary()
```

```
Model: "Sber"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 lstm_3 (LSTM)               (None, 360, 128)          66560

 dropout_2 (Dropout)         (None, 360, 128)          0

 lstm_4 (LSTM)               (None, 360, 128)          131584

 dropout_3 (Dropout)         (None, 360, 128)          0

 lstm_5 (LSTM)               (None, 1)                 520

 dense_1 (Dense)             (None, 1)                 2

=================================================================
Total params: 198,666
Trainable params: 198,666
Non-trainable params: 0
```

Рисунок 11 — Созданная модель

```
model.fit(x_train, y_train, batch_size=16, epochs=50)
```

```
Epoch 1/50
71/71 [==============================] - 34s 424ms/step - loss: 0.0258
Epoch 2/50
71/71 [==============================] - 30s 426ms/step - loss: 0.0050
Epoch 3/50
71/71 [==============================] - 31s 435ms/step - loss: 0.0044
Epoch 4/50
71/71 [==============================] - 23s 325ms/step - loss: 0.0042
Epoch 5/50
71/71 [==============================] - 22s 315ms/step - loss: 0.0042
Epoch 6/50
71/71 [==============================] - 22s 311ms/step - loss: 0.0037
Epoch 7/50
71/71 [==============================] - 25s 354ms/step - loss: 0.0032
Epoch 8/50
71/71 [==============================] - 22s 309ms/step - loss: 0.0030
Epoch 9/50
71/71 [==============================] - 26s 369ms/step - loss: 0.0026
Epoch 10/50
71/71 [==============================] - 23s 329ms/step - loss: 0.0026
Epoch 11/50
71/71 [==============================] - 22s 304ms/step - loss: 0.0024
Epoch 12/50
71/71 [==============================] - 22s 311ms/step - loss: 0.0021
Epoch 13/50
71/71 [==============================] - 22s 311ms/step - loss: 0.0022
Epoch 14/50
71/71 [==============================] - 26s 367ms/step - loss: 0.0024
Epoch 15/50
71/71 [==============================] - 22s 307ms/step - loss: 0.0019
Epoch 16/50
71/71 [==============================] - 24s 338ms/step - loss: 0.0020
Epoch 17/50
71/71 [==============================] - 22s 305ms/step - loss: 0.0019
Epoch 18/50
```

Рисунок 12 — Процесс обучения модели

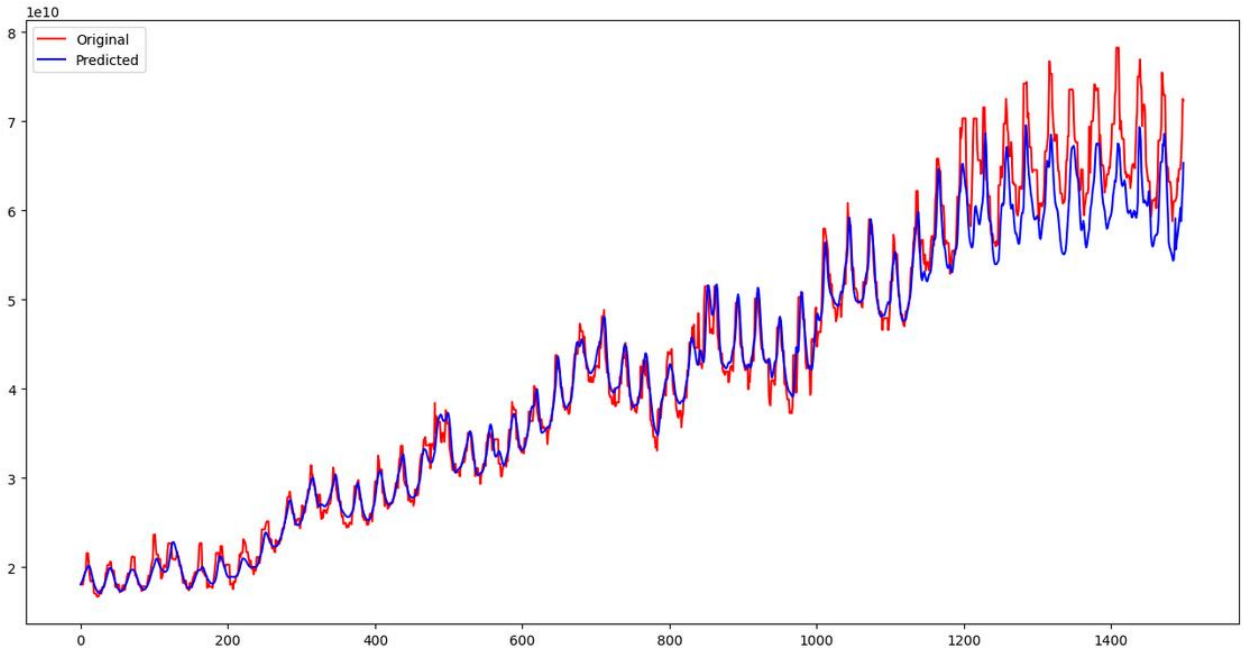

Рисунок 13 — Тестирование модели

Несмотря на то, что модель хорошо показывает себя на тренировочных данных, на тестовых она ведет себя уже не так хорошо, что мне показалось плохим знаком. К сожалению, мои опасения подтвердились, когда я попробовал предсказать следующие 360 значений.

Для предсказаний я брал последние 360 значений для создания одного нового. Добавлял это значение в изначальный массив и таким образом двигался вперед.

```
nswr = df.copy()

for _ in range(window):
    x_data = nswr[-window:].reshape(1, window)

    prediction = model.predict(x_data)

    nswr = np.append(nswr, prediction)

nswr[-window:]
```

Рисунок 14 — Процесс предсказания

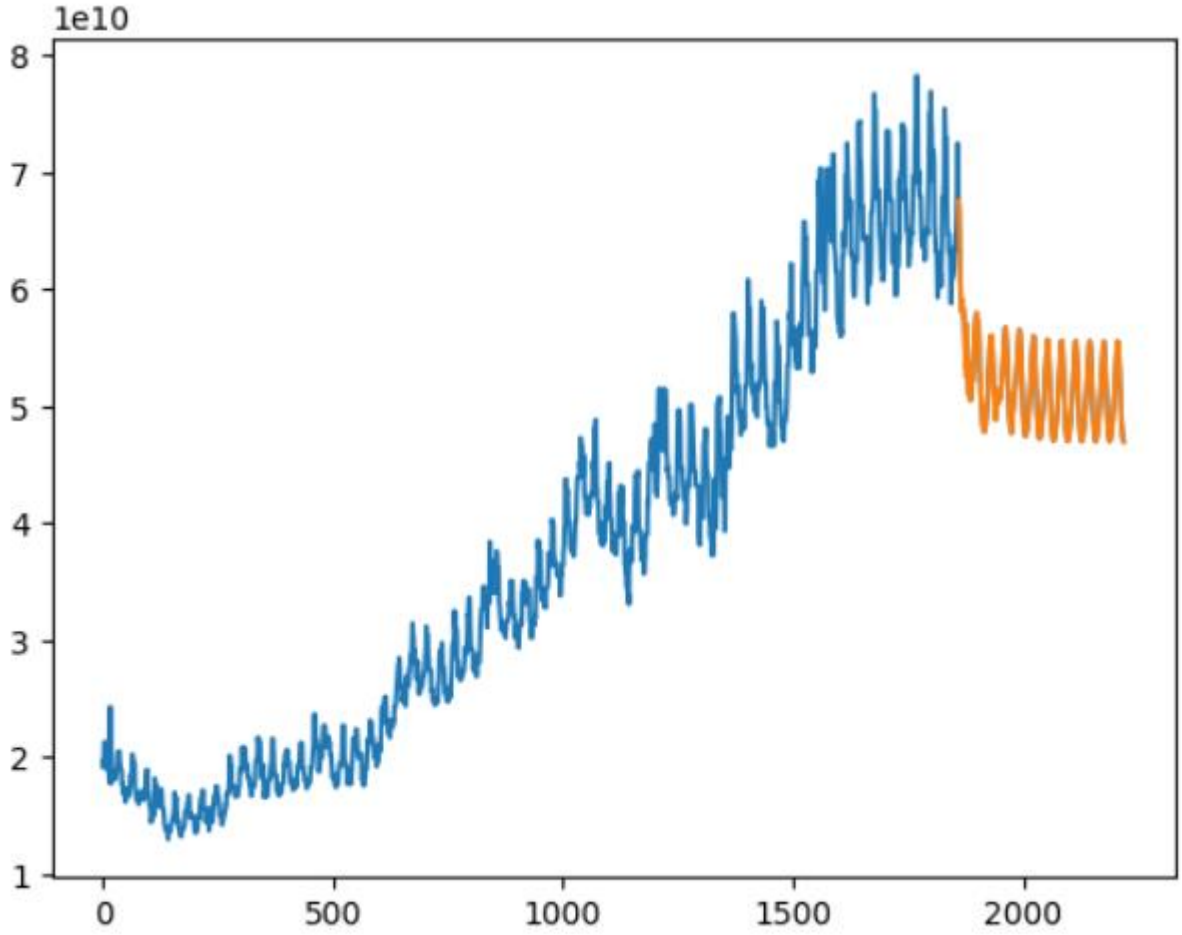

Рисунок 15 — Тестирование модели

Как мы видим, модель не только занижает значения сразу после первых предсказаний, но и подстраивается под паттерн последних значений, что приводит к бесконечной синусоиде. Я пытался настроить модель и тренировать ее в разных пропорциях и с разными слоями, но результат был одинаковый, если не хуже.

Следующим я попробовал RandomForestRegressor.

```
from sklearn.ensemble import RandomForestRegressor
```

```
df = data['value'].values.reshape((-1, 1))
dataset_train = np.array(df)

window = 30 * 12

x_train, y_train = create_dataset(dataset_train, window)

model = RandomForestRegressor()

model.fit(x_train, y_train)
accuracy = model.score(x_train, y_train)

accuracy
```

```
0.9993714368681053
```

Рисунок 16 — Создание, обучение и проверка модели

Эту модель я пытался обучить на всей генеральной совокупности, но это привело к похожему результату, что и LSTM – модель предсказывает значения, подстраивается под новые и в итоге идет по своему паттерну. В данном случае модель ушла в затухающую синусоиду.

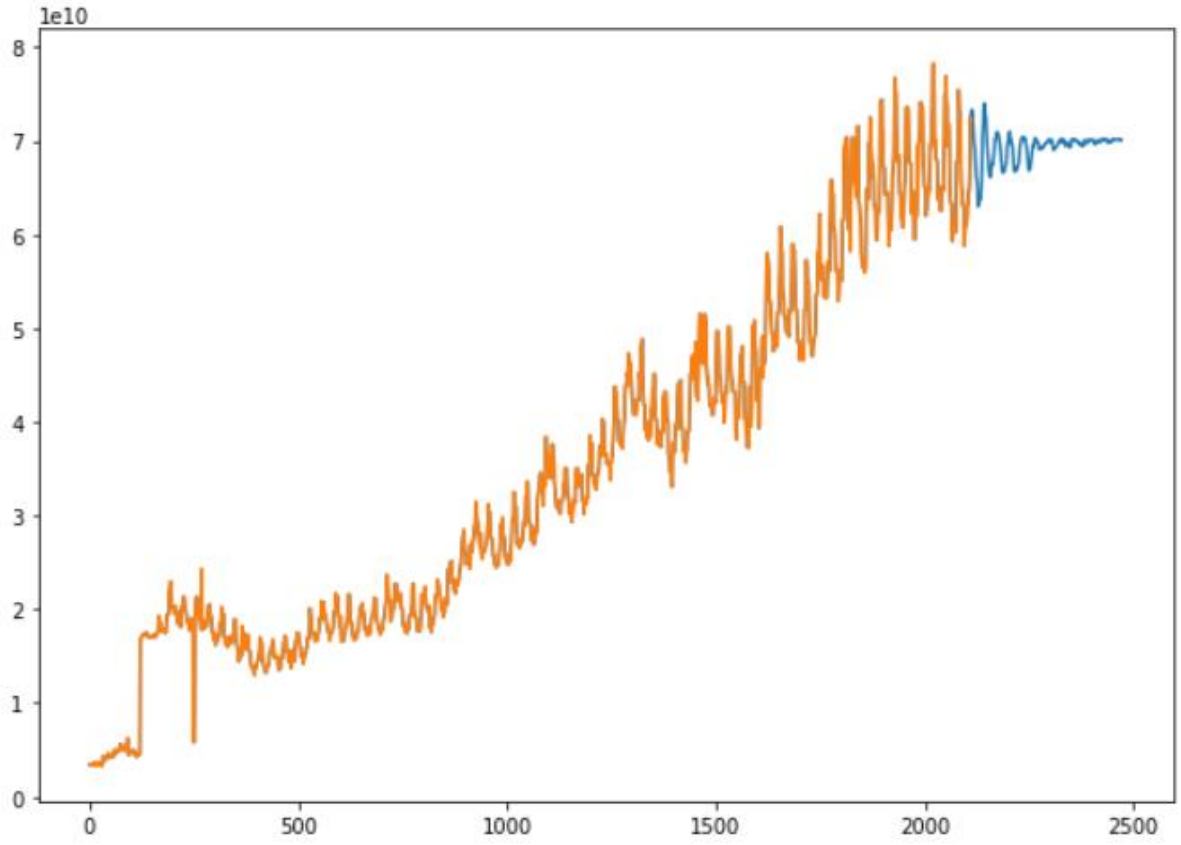

Рисунок 17 — Построение графика с изначальным датасетом и предсказанием

Последним вариантом я попробовал линейную регрессию. Линейная регрессия имеет недостатки, но отсутствие выбросов немного улучшает ситуацию.

```
model = LinearRegression()
```

```
model.fit(x_train, y_train)
accuracy = clf.score(x_train, y_train)

accuracy
```

```
0.9972750343159179
```

Рисунок 17 — Построение модели и обучение на тренировочных данных

В результате предсказания на тренировочных данных получились в достаточной мере верными, чтобы выбрать эту модель для выполнения задания.

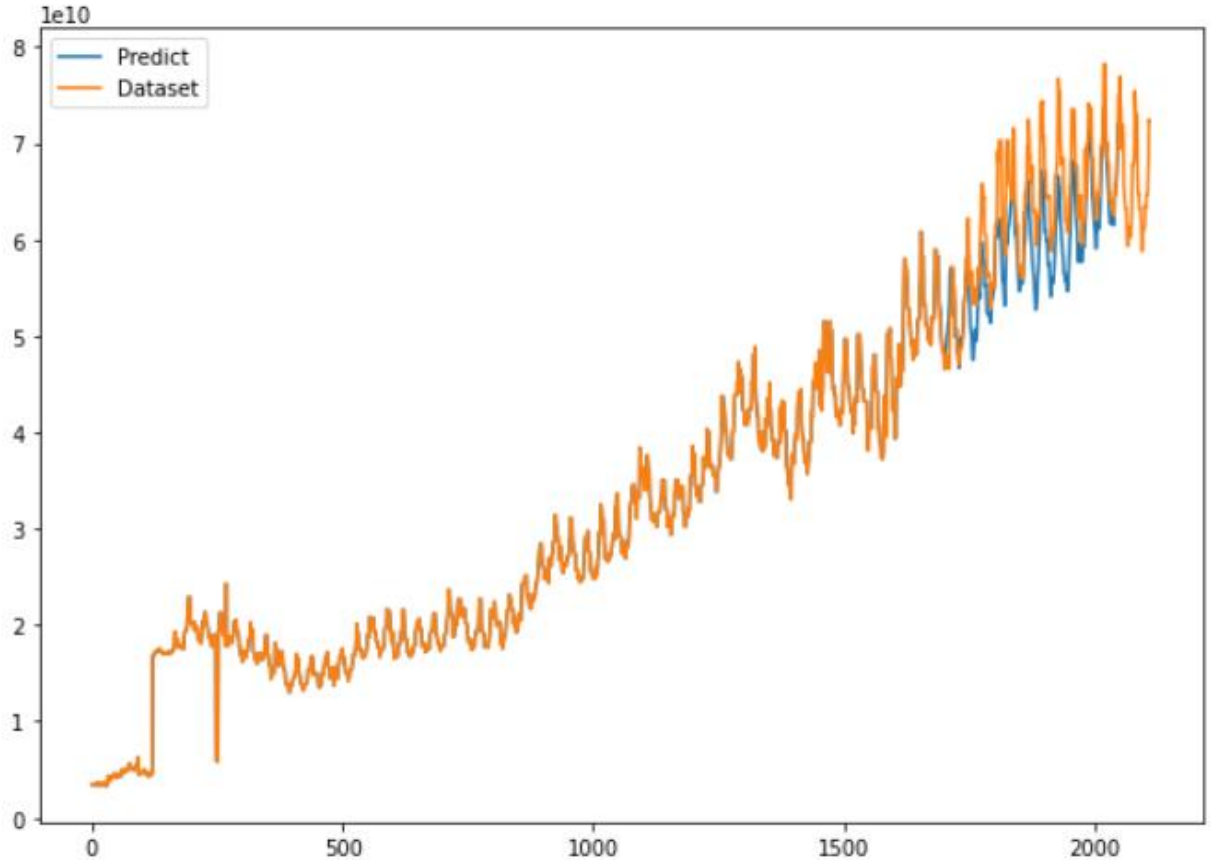

Рисунок 18 — Результат работы модели на тренировочных данных

Дальше я создал два файла: один для создания модели, а второй — main — для выполнения основной программы. По итогу обучения пользователь может сохранить модель, если ее надо будет использовать в дальнейшем.

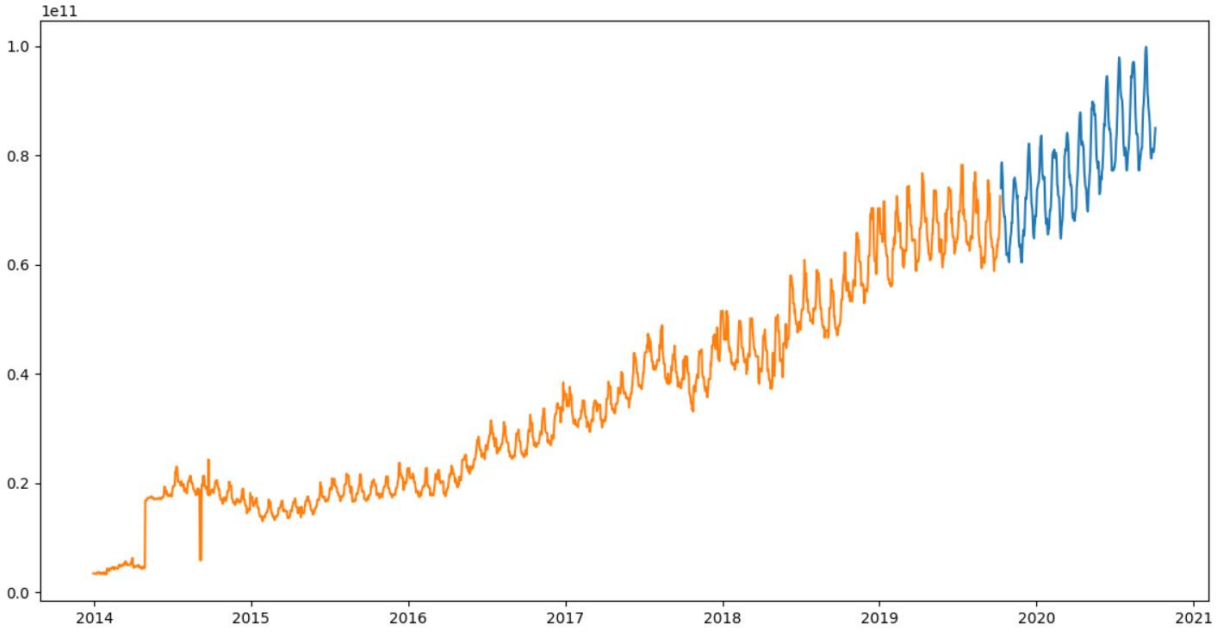

Рисунок 18 — Результат работы программы на всей генеральной совокупности с окном в 360 значений

В результате работы была построена модель, предсказывающая 30 * количество_месяцев значений. Результат работы программы заключается в сохранении модели и предсказаний в виде графика.