



ОТЧЕТ ПО ЗАДАНИЮ №5

Прогнозирование потока событий

Автоматический поиск аномалий во временных рядах с использованием
алгоритма ARIMA

*Работу выполнила Акулова Екатерина
для ГК Innostage
(задача для стажеров DS, BD)*

27 декабря 2022

Содержание

ВВЕДЕНИЕ	3
1. Подготовка окружения.....	4
2. Алгоритм ARIMA.....	5
3. Блок - схема	6
4. Реализация	7
4.1 Анализ временного ряда	8
4.2 Подготовка данных	17
4.3 Обучение модели ARIMA и прогнозирование.....	19
4.4 Поиск аномалий.....	22
4.5 Результат.....	24
ЗАКЛЮЧЕНИЕ	29

ВВЕДЕНИЕ

Аномалии - это образцы данных, которые не соответствуют четко определенному понятию нормального поведения. Важность выявления аномалий обусловлена тем фактом, что аномалии в данных приводят к значительной и действенной информации в самых разных областях применения. Например, аномальная схема трафика в компьютерной сети может означать, что компьютер взломан и отправляет конфиденциальные данные неавторизованному адресату. Аномалии в данных транзакций по кредитной карте, которые могут указывать на кредитную карту или кражу личных данных. Обнаружение аномалий было изучено несколькими исследовательскими сообществами для решения проблем в разных областях применения. Во многих областях, таких как безопасность, обнаружение мошенничества, медицинское обслуживание и т.д.

Временной ряд - это собранный в разные моменты времени статистический материал о значении каких-либо параметров (в простейшем случае одного) исследуемого процесса. Другими словами, упорядоченная во времени последовательность значений какого-либо датчика. Такие данные легко собрать, и они имеют высокую значимость в определении состояния системы, так как могут быть получены в реальном времени. Таким образом остро встает задача анализа данного типа данных.

Задача направлена на прогнозирование потока событий и обнаружение аномалий с использованием языка python и любых библиотек. В результате должна получиться таблица интервалов и значений, на которых произошли отклонения от прогнозных данных. Количество значений не должно превышать 1% от всего объема данных.

В данной работе рассматривается алгоритм ARIMA (семейство алгоритмов ARMA/ARIMA/ARMA/SARMA). ARIMA - авторегрессионная интегрированная модель скользящего среднего, представляет собой алгоритм прогнозирования, основанный на идее, что информация о прошлых значениях временного ряда может использоваться для прогнозирования будущих значений.

1. Подготовка окружения

Вся работа выполнялась на macOS Monterey 12.6

Для выполнения задания необходимо установить:

1. Jupyter Notebook. *Использовался для анализа данных и визуализации*
2. Pycharm *Использовался для написания программного кода*

Библиотеки для работы:

- statsmodels
- pmdarima
- sklearn
- pandas
- numpy
- plotly
- itertools

Данные были приложены к заданию.

2. Алгоритм ARIMA

ARIMA - (A)авто- (R)егрессионная- (I)интегрированная модель (M)скользящего(A)среднего. Вот разбивка того, что означает каждый из этих терминов:

Авторегрессионная: это, по сути, означает, что ARIMA просматривает исторические данные, чтобы помочь предсказать следующую точку данных (насколько далеко назад мы ищем, чтобы помочь предсказать следующую точку данных). В ARIMA термин «AR» также обозначается как «*p*» и может быть установлен вручную, просмотрев графики PACF.

Интегрированная: этот термин в ARIMA относится к разности значений в наборе данных временного ряда. Это помогает преобразовать ряд в стационарный ряд. Когда временной ряд является стационарным, среднее значение остается примерно одинаковым. Нет тренда вверх или вниз с течением времени. В ARIMA «*i*» также упоминается как «*d*». Тест ADF может помочь определить, является ли ряд стационарным.

Скользящее среднее: компонент скользящего среднего в ARIMA по сути является окном. Этот компонент просматривает предыдущие значения, чтобы помочь оценить тенденцию ряда. В ARIMA этот термин обозначается как «*q*», и его можно установить вручную, посмотрев на графики ACF.

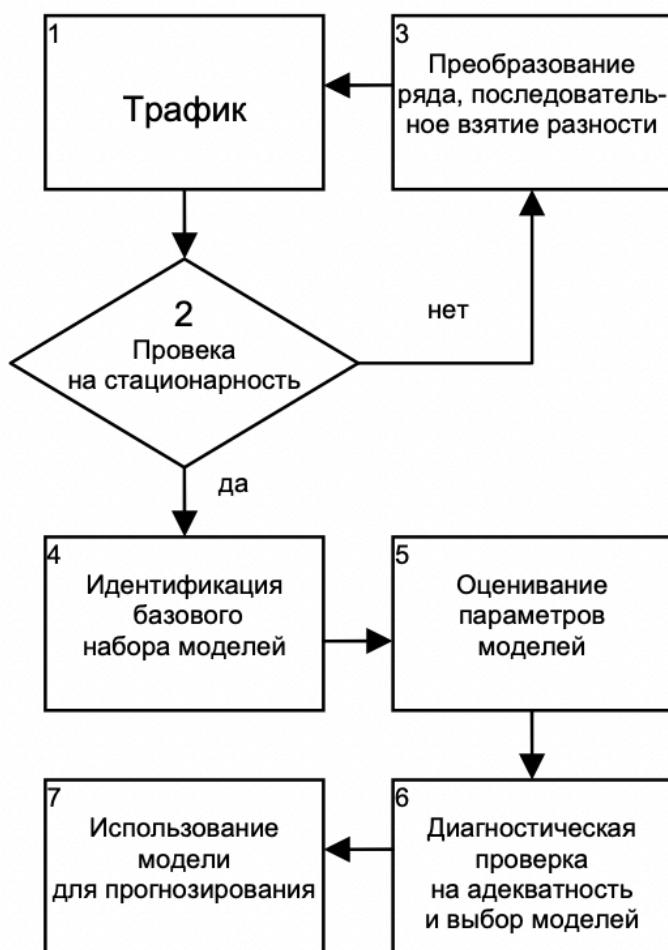
Было бы удобно, если бы была функция, которая автоматически пыталась оценить эти параметры для нас, чтобы мы могли просто, быстро и легко получить прогноз? Такая функция есть! Аналогичная `auto.arima` для R функция есть и в Python благодаря библиотеке `pmdarima`. Мы также будем использовать эту библиотеку и функцию `auto_arima`, чтобы упростить задачу.

3. Блок - схема

Блок-схема программы



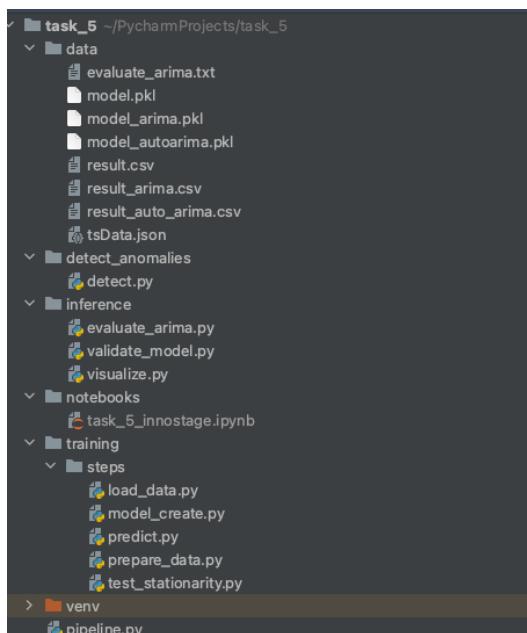
Блок-схема алгоритма



4. Реализация

Программа разделена следующим образом:

1. Директория `data` содержит файлы - исходные данные, `evaluate_arima` с оценкой разных параметров для модели `arima`, сохраненные модели, таблица-результат выявленных аномалий для `auto arima` и `arima`
2. Директория `notebooks` содержит ноутбук с первичным анализом данных и графиками.
3. Директория `detect_anomalies`:
 - `detect.py` для обнаружения аномалий.
4. `Inference`
 - `evaluate_arima.py` для оценки параметров модели.
 - `validate_model.py` для валидации модели.
 - `visualize.py` для построения итоговых графиков.
5. Директория `training`
 - `load_data.py` для загрузки исходных данных.
 - `prepare_data.py` для подготовки данных.
 - `test_stationarity.py` содержит тест на стационарность данных.
 - `model_create.py` для создания моделей.
 - `predict.py` для получения предсказаний.
6. `pipeline.py` - содержит три функции пайплайна, которые используют данные с разными преобразованиями и параметрами модели ARIMA.



4.1 Анализ временного ряда

Для начала загрузим нужные библиотеки.

```
import pandas as pd
import matplotlib.pyplot as plt
from statsmodels.graphics.tsaplots import plot_predict
import numpy as np
from pyspark.sql import SparkSession
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from sklearn.preprocessing import StandardScaler
import statsmodels
import statsmodels.api as sm
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
import plotly
import plotly.graph_objects as go
from itertools import cycle
from pmdarima.arima import ADFTest
from pmdarima.arima import auto_arima
import seaborn as sns

import warnings
warnings.filterwarnings("ignore")
```

Загрузим данные, переведем индекс датасета в формат datetime, посмотрим на данные и установим интервал временного ряда в 5 минут, так как данных много и это повлияет на скорость обучения модели.

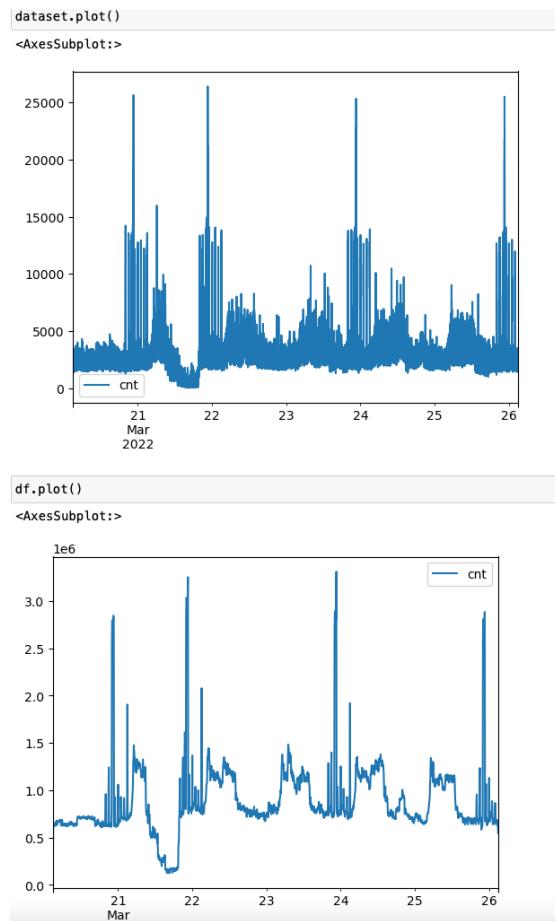
```
dataset.shape
(518401, 1)
: path = "tsData.json"
dataset = pd.read_json(path)

: dataset.index = pd.to_datetime(dataset.index)

: df = pd.DataFrame(dataset.cnt.resample("5MIN", base=1).sum())

: print(df.info())
print(df.shape)
print(df.describe())
print(df.head())
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 1729 entries, 2022-03-20 02:56:00 to 2022-03-26 02:56:00
Freq: 5T
Data columns (total 1 columns):
 #   Column Non-Null Count Dtype  
--- 
 0   cnt      1729 non-null   int64  
dtypes: int64(1)
memory usage: 27.0 KB
None
(1729, 1)
          cnt
count  1.729000e+03
mean   8.836529e+05
std    3.800296e+05
min    1.244250e+05
25%   6.957280e+05
50%   7.812090e+05
75%   1.105356e+06
max    3.308481e+06
          cnt
2022-03-20 02:56:00 126077
2022-03-20 03:01:00 646378
2022-03-20 03:06:00 616458
2022-03-20 03:11:00 653960
2022-03-20 03:16:00 623906
```

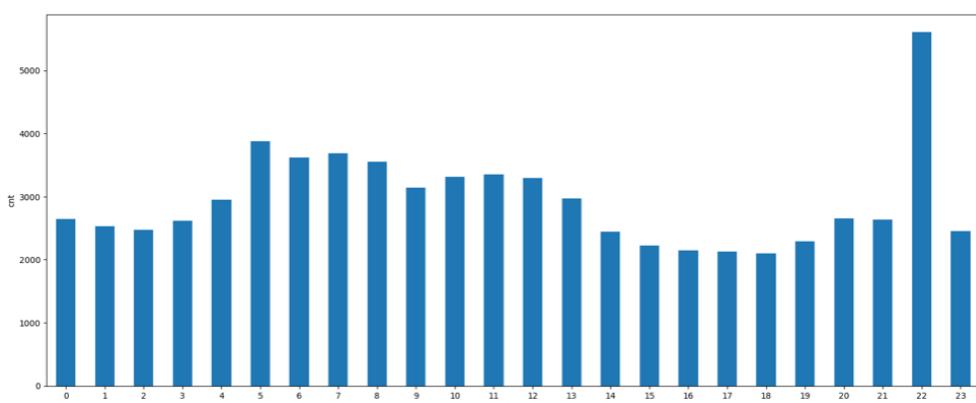
Посмотрим на графики с интервалами в 1 секунду и в 5 минут.



Тренд и сезонность в данных с интервалом в 5 минут сохранились. Значит такой интервал подходит для повторной выборки.

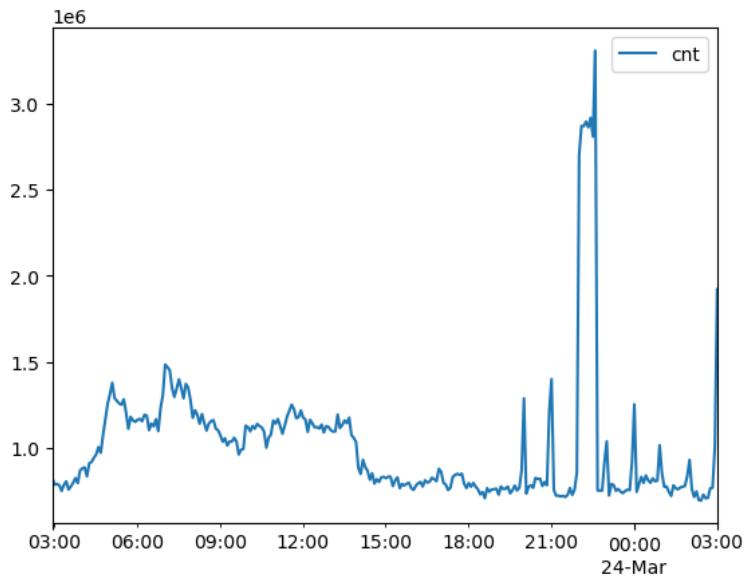
Построим график среднего значения cnt для каждого часа в сутках.

```
fig, axs = plt.subplots(figsize=(20, 8))  
dataset.groupby(  
    dataset.index.hour)[["cnt"]].mean().plot(kind='bar',  
    rot=0,  
    ax=axs)  
  
plt.xlabel("Day")  
plt.ylabel("cnt")  
  
Text(0, 0.5, 'cnt')
```



Выберем любой день и построим график.

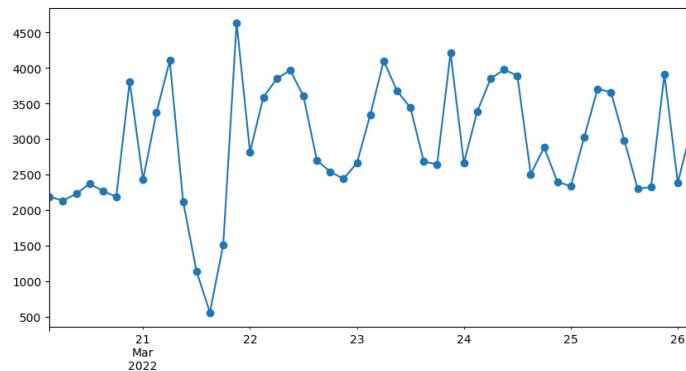
```
df['2022-03-23 02:56:00':'2022-03-24 03:01:00'].plot()  
<AxesSubplot:>
```



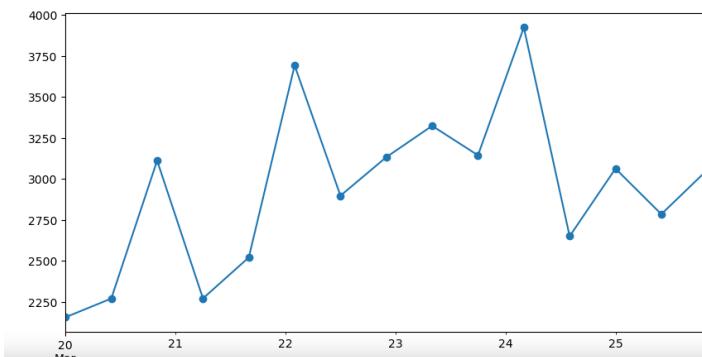
Действительно, можно отметить что самые высокие значения наблюдаются в 22 часа и значения выше среднего в ~ 5 - 12 часов.

Построим еще пару графиков с другими временными интервалами и пойдем дальше.

```
: dataset.cnt.resample("3H").mean().plot(style="-o", figsize=(10, 5))  
: <AxesSubplot:>
```



```
: dataset.cnt.resample("10H").mean().plot(style="-o", figsize=(10, 5))  
: <AxesSubplot:>
```

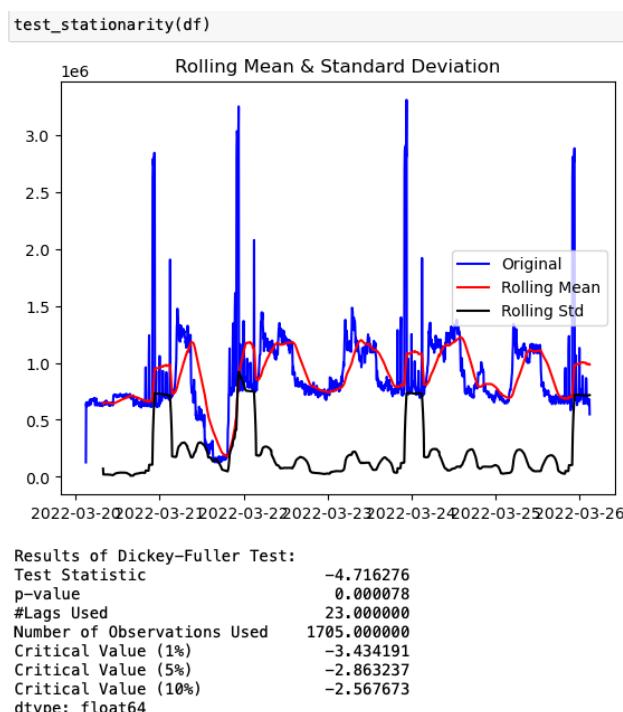


Перейдем к тесту на стационарность временного ряда. Данные для обучения модели ARIMA должны быть стационарны, то есть статистические свойства процесса, генерирующего временные ряды, не должны изменяться со временем. Если временной ряд имеет определенное поведение на временном интервале, то существует высокая вероятность того, что на другом интервале оно будет вести себя так же, при условии, что временной ряд является стационарным. Это помогает точно прогнозировать.

```
def test_stationarity(timeseries):
    #Determining rolling statistics
    rolmean = timeseries.rolling(window=60,center=False).mean()
    rolstd = timeseries.rolling(window=60,center=False).std()
    #Plot rolling statistics:
    orig = plt.plot(timeseries, color='blue',label='Original')
    mean = plt.plot(rolmean, color='red', label='Rolling Mean')
    std = plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation')
    plt.show(block=False)
    #Perform Dickey-Fuller test:
    print('Results of Dickey-Fuller Test:')
    dfoutput = adfuller(timeseries, autolag='AIC')
    dfoutput = pd.Series(dfoutput[0:4], index=['Test Statistic','p-value','#Lags Used','Number of Observations Used'])
    for key,value in dfoutput[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

В данной функции мы определяем окно для скользящего среднего и скользящего стандартного отклонения и рисуем график. Далее используем тест Дики-Фуллера для проверки на стационарность.

Посмотрим на результат выполнения функции для данных с интервалом в 5 минут.

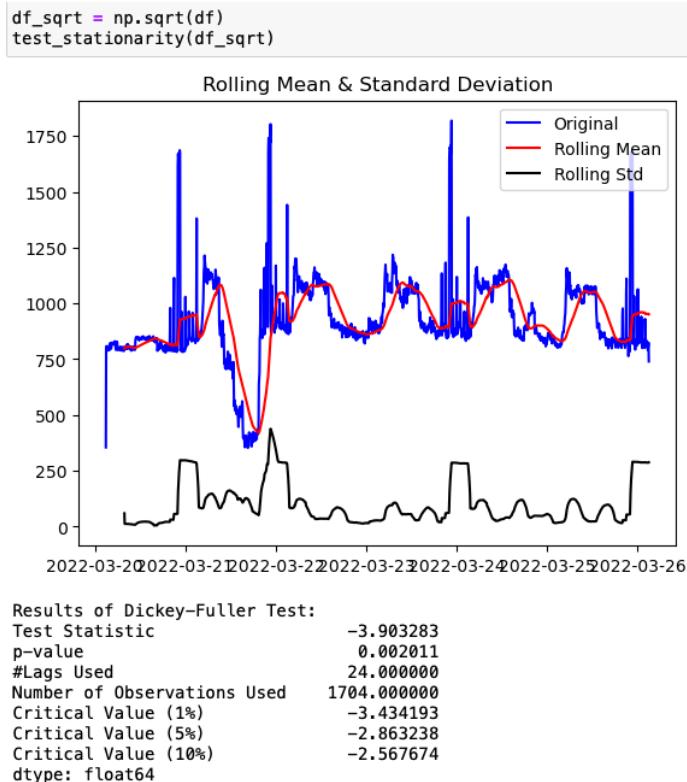


Теперь разделим данные наполовину и посмотрим, насколько различается медиана и дисперсия двух массивов.

```
X = df.values
split = int(len(X) / 2)
X1, X2 = X[0:split], X[split: -1]
mean1, mean2 = X1.mean(), X2.mean()
var1, var2 = X1.var(), X2.var()
print('mean1=%f, mean2=%f' % (mean1, mean2))
print('variance1=%f, variance2=%f' % (var1, var2))

mean1=814332.800926, mean2=953362.144676
variance1=164627243280.585388, variance2=114422375715.126053
```

Нулевая гипотеза о нестационарности временного ряда отвергается. Но тем не менее полученный график выглядит не идеально, можно попробовать преобразовать данные.



Набор данных мы преобразовали в новый набор данных, где вместо предыдущих значений - их положительный квадратный корень. Результат теста на стационарность положителен.

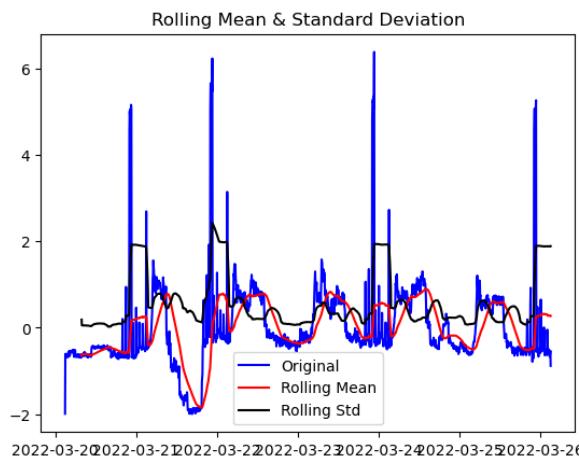
Проведем стандартизацию данных и посмотрим на результат.

```

scaler = StandardScaler()
scaler.fit(df['cnt'].values.reshape(-1, 1))
df['stand_value'] = scaler.transform(df['cnt'].values.reshape(-1, 1))
df = df.drop(columns=['cnt'], axis=1)

```

```
test_stationarity(df)
```



```

Results of Dickey-Fuller Test:
Test Statistic      -4.716276
p-value            0.000078
#Lags Used        23.000000
Number of Observations Used   1705.000000
Critical Value (1%)    -3.434191
Critical Value (5%)     -2.863237
Critical Value (10%)    -2.567673
dtype: float64

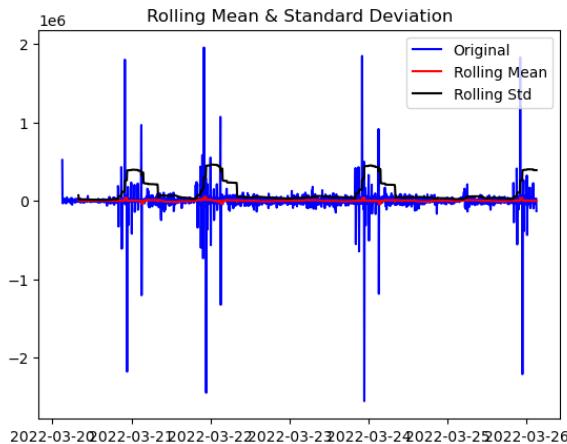
```

Данные стационарны и их можно использовать для обучения модели. А что если вычислить дискретную разность и сдвинуть данные?

```

dfDiffShifting = df - df.shift()
dfDiffShifting.dropna(inplace=True)
test_stationarity(dfDiffShifting)

```



```

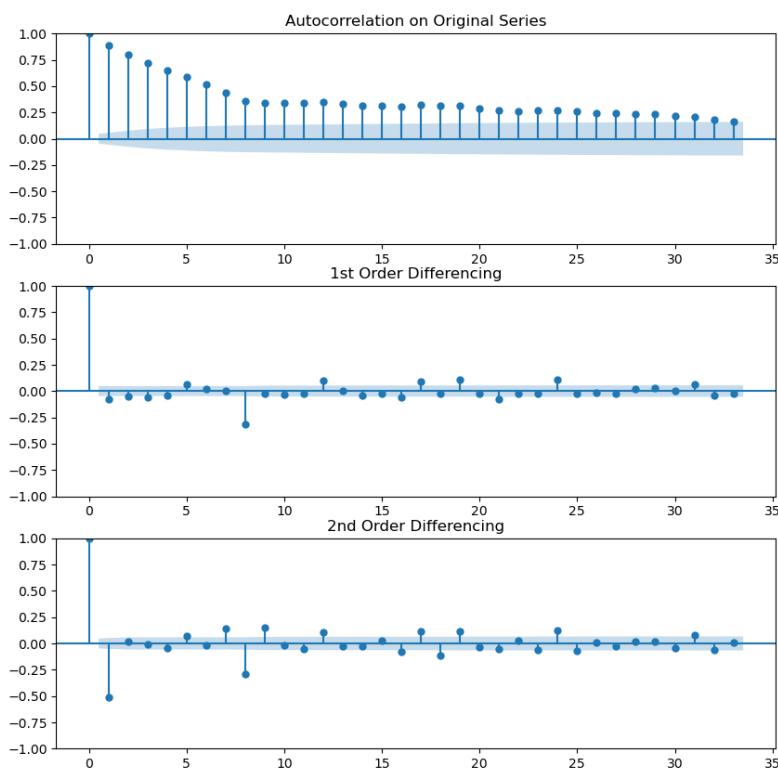
Results of Dickey-Fuller Test:
Test Statistic      -1.281066e+01
p-value            6.457139e-24
#Lags Used        2.200000e+01
Number of Observations Used   1.705000e+03
Critical Value (1%)    -3.434191e+00
Critical Value (5%)     -2.863237e+00
Critical Value (10%)    -2.567673e+00
dtype: float64

```

Тут мы получили ровную линию скользящего среднего и крайне низкое значение p-value путем сдвига данных. А еще мы получили значение d = 1. Но можно проверить это другим способом.

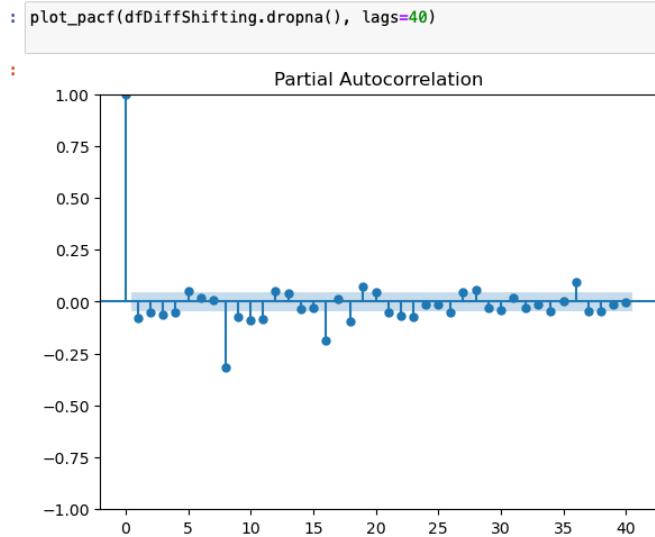
Итак, нам нужно определить значение параметров для ARIMA - p , d и q (p - порядок компоненты AR, d - порядок интегрированного ряда, q - порядок компоненты MA). Это можно сделать через построения графиков автокорреляции и частичной автокорреляции. Нам нужно проверить, при каком значении по оси x линия графика падает до 0 по оси y в первый раз.

```
fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(311)
fig = plot_acf(df, ax=ax1,
                title="Autocorrelation on Original Series")
ax2 = fig.add_subplot(312)
fig = plot_acf(df.diff().dropna(), ax=ax2,
                title="1st Order Differencing")
ax3 = fig.add_subplot(313)
fig = plot_acf(df.diff().diff().dropna(), ax=ax3,
                title="2nd Order Differencing")
fig.show()
#значение d будет 1
```



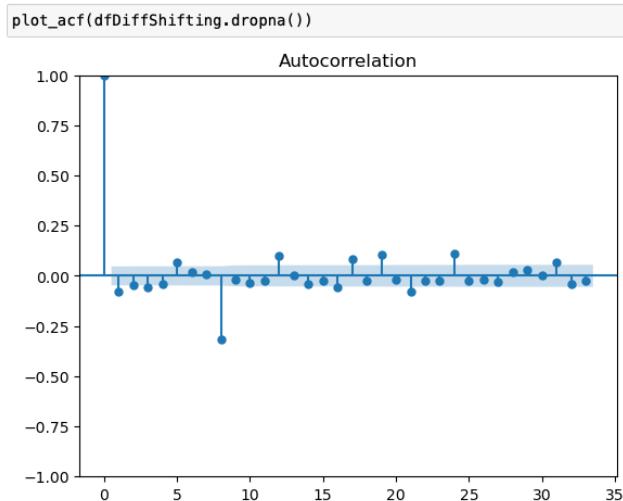
Как видим по графикам, $d = 1$, так как только первый лаг гораздо выше уровня значимости.

Зная, что мы должны вычислить 1-ю дискретную разность, мы продолжаем выяснять порядок AR - p . AR относится к прошлым значениям, используемым для прогнозирования следующего значения. Мы получаем p путем подсчета количества лагов выше уровня значимости в функции частичной автокорреляции:



Выше только один лаг, таким образом, $p = 1$.

МА (скользящее среднее) — используется для определения количества прошлых ошибок прогноза, используемых для прогнозирования будущих значений. Функция автокорреляции поможет нам определять порядок членов МА - q , т. к. по ее коррелограмме можно определить количество автокорреляционных коэффициентов сильно отличных от 0 в модели МА.



Можно заметить больше всего отличен один лаг. Но возможно, p , d , q могут принимать другие значения. Об этом будет написано далее.

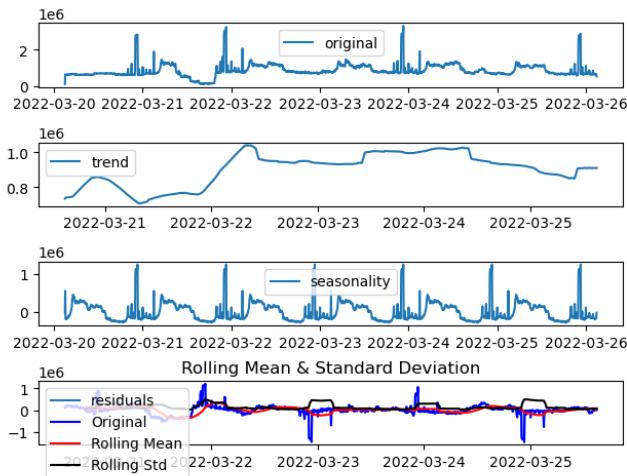
Напоследок посмотрим на компоненты временного ряда.

```

decomposition = seasonal_decompose(df, period = 288)
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid
plt.subplot(411)
plt.plot(df, label='original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal, label='seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='residuals')
plt.legend(loc='best')
plt.tight_layout()

decomposedData = residual
decomposedData.dropna(inplace=True)
test_stationarity(decomposedData)

```



```

Results of Dickey-Fuller Test:
Test Statistic      -4.676116
p-value            0.000093
#Lags Used        24.000000
Number of Observations Used 1416.000000
Critical Value (1%) -3.434977
Critical Value (5%) -2.863583
Critical Value (10%) -2.567858
dtype: float64

```

Последний график это неравномерности которые есть во временном ряду, они не имеют порядок, размер и не могут помочь в прогнозировании. Судя по тесту, этот шум стационарен.

Теперь мы закончили с анализом временного ряда. Можно переходить к следующему пункту.

4.2 Подготовка данных

Сначала нам нужно ввести данные, создать повторную выборку с интервалом в 5 минут, разделить данные на тренировочную и вестовую выборки, и преобразовать данные. В одной из трех функций данные не преобразовываются, начнем с нее.

```
def pipeline_arima(use_new_model=False, input_model_path='', output_model_path='data/model.pkl'):
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    train, test = data_split(data)
    if use_new_model:
        model = model_arima(train, 1, 1, 1)
        save_model(model, output_model_path)
    else:
        model_pickle = open(input_model_path, "rb")
        model = pickle.load(model_pickle)
    prediction_test = get_predictions_arima(model, train, test)
    res, pred_test = detect(prediction_test, test)
    res.to_csv('data/result.csv')
    pred_test = standardising_res(train, pred_test)
    rmse, mse, mae = validation(pred_test)
    percent_anomaly = len(res) / len(test) * 100
    print("Percent anomaly = ", percent_anomaly)
    print("RMSE = ", rmse, "\nMSE = ", mse, "\nMAE = ", mae)
```

В данном участке кода подготовка данных происходит в первых четырех строчках. Заглянем в каждую из функций.

```
def load_data(path):
    dataset = pd.read_json(path)
    return dataset

def data_resample(data):
    data = pd.DataFrame(data.cnt.resample("5MIN", base=1).sum())
    return data

def data_split(data):
    train_set, test_set = np.split(data, [int(.67 * len(data))])
    return train_set, test_set
```

Во втором случае добавлена функция для стандартизации данных и функция для вычисления медианы и стандартного отклонения. Стандартизация изменяет форму распределения данных (приводится к нормальному распределению). Медиану и стандартное отклонение нужно вычислить, чтобы затем привести данные к их изначальному виду.

```

def pipeline_arima_stand(use_new_model=False, input_model_path='', output_model_path='data/model_arima.pkl'):
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    train, test = data_split(data)
    std, mean = std_mean(train, 'cnt')
    train, test = standardising_data(train, test)
    if use_new_model:
        model = model_arima(train, 1, 1, 1)
        save_model(model, output_model_path)
    else:
        model_pickle = open(input_model_path, "rb")
        model = pickle.load(model_pickle)
    res, pred_test = detect(model, train, test, std, mean)
    res.to_csv('data/result_arima.csv')
    rmse, mse, mae = validation(pred_test)
    percent_anomaly = len(res) / len(test) * 100
    print("Percent anomaly = ", percent_anomaly)
    print("RMSE = ", rmse, "\nMSE = ", mse, "\nMAE = ", mae)

```

```

def standardising_data(train, test):
    scaler = StandardScaler()
    scaler.fit(train['cnt'].values.reshape(-1, 1))
    train['stand_value'] = scaler.transform(train['cnt'].values.reshape(-1, 1))
    test['stand_value'] = scaler.transform(test['cnt'].values.reshape(-1, 1))
    train = train.drop(columns=['cnt'], axis=1)
    test = test.drop(columns=['cnt'], axis=1)
    train = train.rename(columns={'stand_value': 'cnt'})
    test = test.rename(columns={'stand_value': 'cnt'})
    return train, test

```

```

def std_mean(data, col):
    std = data[col].std()
    mean = data[col].mean()
    return std, mean

```

В третьем случае была добавлена строчка с преобразованием тренировочных и тестовых данных через нахождение квадратного корня их значений.

```

def pipeline_auto_arima(use_new_model=False, input_model_path='', output_model_path='data/model_autoarima.pkl'):
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    train, test = data_split(data)
    train_sqrt, test_sqrt = np.sqrt(train), np.sqrt(test)
    if use_new_model:
        model = model_auto_arima(train_sqrt)
        save_model(model, output_model_path)
    else:
        model_pickle = open(input_model_path, "rb")
        model = pickle.load(model_pickle)
    res, pred_test = detect_data_sqrt(model, train_sqrt, test, test_sqrt)
    res.to_csv('data/result_auto_arima.csv')
    pred_test = standardising_res(train, pred_test)
    rmse, mse, mae = validation(pred_test)
    print(rmse, mse, mae)
    percent_anomaly = len(res) / len(test) * 100
    print("Percent anomaly = ", percent_anomaly)
    print("RMSE = ", rmse, "\nMSE = ", mse, "\nMAE = ", mae)

```

4.3 Обучение модели ARIMA и прогнозирование

Перед тем как создать и обучить модель, нужно проверить данные на стационарность.

```
def isStationary(data, maxval=0.05):
    test = adfuller(data, autolag='AIC')
    p = test[1]
    if p <= maxval:
        return 1
    else:
        return 0
```

Если выбран параметр «AIC» или «BIC», то количество лагов выбирается таким образом, чтобы минимизировать соответствующий информационный критерий.

Теперь перейдем к обучению модели.

```
def model_auto_arima(train_set):
    if isStationary(train_set):
        model = auto_arima(train_set, start_p=1, start_q=1, max_p=3, max_q=3, m=7,
                            start_P=0, seasonal=True, d=1, D=1, trace=True,
                            error_action='ignore', suppress_warnings=True,
                            stepwise=True)
        return model
    else:
        sys.exit("Data is not stationary.")

def model_arima(train, p, d, q):
    if isStationary(train):
        model = ARIMA(train.values, order=(p, d, q))
        model_fit = model.fit()
        return model_fit
    else:
        sys.exit("Data is not stationary.")
```

В данном участке кода создаются две модели ARIMA с автоматическим подбором параметров и без него.

Рассмотрим функцию, которая поможет оценить, насколько подобранные параметры подходят для модели без автоматического подбора через вычисление средней квадратичной ошибки моделей с различными параметрами путем их перебора. Ниже описан способ ее реализации.

```
def arima_params_evaluate():
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    best = get_eval(data)
    print(best)
```

```

def get_eval(df):
    p_values = [0, 1, 2, 4]
    d_values = range(0, 3)
    q_values = range(0, 3)
    best = evaluate_models(df, p_values, d_values, q_values)
    return best

```

Здесь мы определяем диапазон значений для каждого из трех параметров. Таким образом, функция будет перебирать каждую возможную комбинацию.

Посмотрим, как работает подбор параметров. Все параметры модели и полученные ошибки записываются в файл, чтобы можно было запустить данный код отдельно и затем оценить параметры перед обучением, прогнозом и выявлением аномалий.

```

def evaluate_arima_model(X, arima_order):
    train, test = data_split(X)
    train, test = standardising_data(train, test)
    train, test = train.values, test.values
    history = [x for x in train]
    predictions = list()
    for t in range(len(test)):
        model = ARIMA(history, order=arima_order)
        model_fit = model.fit()
        yhat = model_fit.forecast()[0]
        predictions.append(yhat)
        history.append(test[t])
    error = mean_squared_error(test, predictions)
    return error

def evaluate_models(dataset, p_values, d_values, q_values):
    check_list = []
    best_score, best_cfg = float("inf"), None
    for p in p_values:
        for d in d_values:
            for q in q_values:
                order = (p, d, q)
                try:
                    mse = evaluate_arima_model(dataset, order)
                    if mse < best_score:
                        best_score, best_cfg = mse, order
                    check_list.append(f'ARIMA{order} : {mse}')
                    print(f'ARIMA{order} MSE={mse}')
                except:
                    continue
    with open('data/evaluate_arima.txt', 'w') as f:
        f.writelines(f'{row}\n' for row in check_list)
    return best_score, best_cfg

```

Здесь создаются различные комбинации из переданных значений и каждая из них оценивается. Раскроем алгоритм:

1. Разделить набор данных на обучающий и тестовый наборы
2. Пройти временные шаги в тестовом наборе данных
 - a. Обучить модель ARIMA
 - б. Сделать одношаговый прогноз

- в. Сохранить прогноз; получить и сохранить фактическое наблюдение.
3. Вычислить ошибку для прогнозов по сравнению с ожидаемыми значениями.

Наименьшее полученное значение MSE — лучший набор параметров.

Вывод будет показан в разделе с результатами.

Теперь перейдем к получению предсказаний. Напишем две функции для двух моделей, одна из которых (1) будет использовать `model.predict()`, а другая (2) работать по похожему принципу, но используя написанную от руки функцию предсказания. В нее мы будем передавать коэффициенты AR и MA и таблицу разностей каждого элемента в тестовой выборке с его предыдущим элементом. В результате обучения мы должны получить новый столбец в датафрейме тестовых данных, который будет содержать предсказанное значение для каждой строки.

```
def get_prediction_auto_arima(model, train_sqrt, test_sqrt, test):
    history = [x for x in train_sqrt.values]
    predictions = list()
    predict_sqrt = list()
    for t in range(len(test)):
        model.fit(history)
        output = model.predict(n_periods=1)
        predict_sqrt.append(output[0])
        yhat = output[0] ** 2
        predictions.append(yhat)
        obs = test_sqrt.values[t]
        history.append(obs)
    prediction = pd.DataFrame(predictions)
    prediction.index = test.index
    test['predict'] = prediction
    return test
```

```
def get_predictions_arima(model, train_set, test_set):
    predictions = list()
    train, test = train_set.values, test_set.values
    history = [x for x in train]
    for t in range(len(test)):
        ar_coef, ma_coef = model.arparams, model.maparams
        resid = model.resid
        diff = difference(history)
        yhat = history[-1] + predict(ar_coef, diff) + predict(ma_coef, resid)
        obs = test_set.values[t]
        predictions.append(yhat)
        history.append(obs)
    prediction = pd.DataFrame(predictions)
    prediction.index = test_set.index
    test_set['predict'] = prediction
    return test_set
```

```

def predict(coef, history):
    yhat = 0.0
    for i in range(1, len(coef)+1):
        yhat += coef[i-1] * history[-i]
    return yhat

def difference(dataset):
    diff = list()
    for i in range(1, len(dataset)):
        value = dataset[i] - dataset[i - 1]
        diff.append(value)
    return np.array(diff)

```

В первом случае мы имели преобразованные значения в виде квадратного корня значений из оригинальной выборки. Мы возводим их во вторую степень. Во втором случае данные не меняются.

4.4 Поиск аномалий

Рассмотрим функцию для обнаружения аномалий на основе ошибок прогнозирования и определенного порогового значения.

```

def calculate_prediction_errors(input_data):
    return (abs(input_data['cnt'] - input_data['predict'])).to_numpy()

def detect_anomalies(pred_error_threshold, df):
    test_reconstruction_errors = calculate_prediction_errors(df)
    predicted_anomalies = list(
        map(lambda v: 1 if v > pred_error_threshold else 0, test_reconstruction_errors))
    df['anomaly_predicted'] = predicted_anomalies
    indexes = [i for i, x in enumerate(predicted_anomalies) if x == 1]
    return indexes

```

Сначала мы вычисляем ошибку. Делаем мы это путем вычисления абсолютного значения числа, которое представляет из себя разность значений cnt с предсказанными значениями. Затем мы обозначаем предсказанные значения как аномалии либо не аномалии в соответствии с пороговым значением. Пороговое значение представляет из себя число стандартных отклонений ошибки прогнозирования, которое отличается от средней ошибки прогнозирования, чтобы классифицировать ее как аномалию. Чем больше значение параметра конфигурации тем меньше чувствительность обнаружения аномалий. Функция возвращает индексы выявленных аномалий.

Теперь переходим к моменту, где мы должны получить итоговые графики и таблицу с выявленными аномалиями.

```
def detect_std(prediction_test, test_set, std, mean):
    anomaly_config = 3
    test_pred_errors = calculate_prediction_errors(prediction_test)
    pred_error_threshold = np.mean(test_pred_errors) + anomaly_config * np.std(test_pred_errors)
    test_anomalies_idxs = detect_anomalies(pred_error_threshold, test_set)

    pred_anomaly_on_test_data(prediction_test, test_anomalies_idxs)

    detected_anomalies = test_set['cnt'].iloc[test_anomalies_idxs]
    orig_det_anom = (detected_anomalies * std) + mean
    return orig_det_anom, prediction_test

def detect(prediction_test, test_set):
    anomaly_config = 3
    test_pred_errors = calculate_prediction_errors(prediction_test)
    pred_error_threshold = np.mean(test_pred_errors) + anomaly_config * np.std(test_pred_errors)
    test_anomalies_idxs = detect_anomalies(pred_error_threshold, test_set)

    pred_anomaly_on_test_data(prediction_test, test_anomalies_idxs)

    detected_anomalies = test_set['cnt'].iloc[test_anomalies_idxs]
    return detected_anomalies, prediction_test
```

Здесь две разных функции так как на вход они принимают таблицы со стандартизованными данными, которые нужно обратно преобразовать, чтобы функция вернула таблицу с выявленными аномалиями, как положено, либо с данными, которые не нужно обратно преобразовывать. Все шаги были описаны выше, поэтому посмотрим только на часть с визуализацией.

```
def pred_anomaly_on_test_data(test_set, test_anomalies_idxs):
    ano_ind = np.where(test_set["anomaly_predicted"] == 1)
    layout = dict(xaxis=dict(title='Time'), yaxis=dict(title='Count'))

    fig = go.Figure(layout=layout)

    fig.add_trace(go.Scatter(x=test_set.index, y=test_set['cnt'],
                            mode='markers',
                            marker=dict(color='blue')))

    fig.add_trace(go.Scatter(x=test_set.index, y=test_set['predict'],
                            mode='markers',
                            marker=dict(color='orange')))

    fig.add_trace(go.Scatter(x=test_set.iloc[ano_ind].index, y=test_set['cnt'].iloc[test_anomalies_idxs],
                            mode='markers',
                            marker=dict(color='red')))

    nam = cycle(['Actual', 'Prediction', 'Detected Anomaly'])
    fig.for_each_trace(lambda t: t.update(name=next(nam)))
    fig.show()
```

В итоге должен получиться scatter plot, с точками изначальных данных синего цвета, предсказанных - желтого, а красными будут обозначены аномалии. Давайте посмотрим на полученные результаты.

4.5 Результат

Первое, на что стоит обратить внимание - параметры модели ARIMA и ее ошибки.

```
ARIMA% : (0, 0, 0), MSE=%.3f: 0.6028039949610886
ARIMA% : (0, 0, 1), MSE=%.3f: 0.2479060983104408
ARIMA% : (0, 0, 2), MSE=%.3f: 0.1745763468583507
ARIMA% : (0, 1, 0), MSE=%.3f: 0.10990735133540817
ARIMA% : (0, 1, 1), MSE=%.3f: 0.11130263472696954
ARIMA% : (0, 1, 2), MSE=%.3f: 0.11135153601624889
ARIMA% : (0, 2, 0), MSE=%.3f: 0.21794584827171562
ARIMA% : (0, 2, 1), MSE=%.3f: 0.10997637016711095
ARIMA% : (0, 2, 2), MSE=%.3f: 0.11135667361122241
ARIMA% : (1, 0, 0), MSE=%.3f: 0.10607548401872834
ARIMA% : (1, 0, 1), MSE=%.3f: 0.10679889582437226
ARIMA% : (1, 0, 2), MSE=%.3f: 0.10692358854667786
ARIMA% : (1, 1, 0), MSE=%.3f: 0.1104445796839076
ARIMA% : (1, 1, 1), MSE=%.3f: 0.1064624286788015
ARIMA% : (1, 1, 2), MSE=%.3f: 0.10680437439525324
ARIMA% : (1, 2, 0), MSE=%.3f: 0.17065581321275666
ARIMA% : (1, 2, 1), MSE=%.3f: 0.11109996108836644
ARIMA% : (1, 2, 2), MSE=%.3f: 0.10808828201002871
ARIMA% : (2, 0, 0), MSE=%.3f: 0.10676888747520398
ARIMA% : (2, 0, 1), MSE=%.3f: 0.1069614791793353
ARIMA% : (2, 0, 2), MSE=%.3f: 0.10714964602458303
ARIMA% : (2, 1, 0), MSE=%.3f: 0.11107778914972083
```

Запуская код, получаем файл примерно такого содержания. Весь файл можно посмотреть в приложении. Посмотрим, модель с какими параметрами имеет наименьшую MSE.

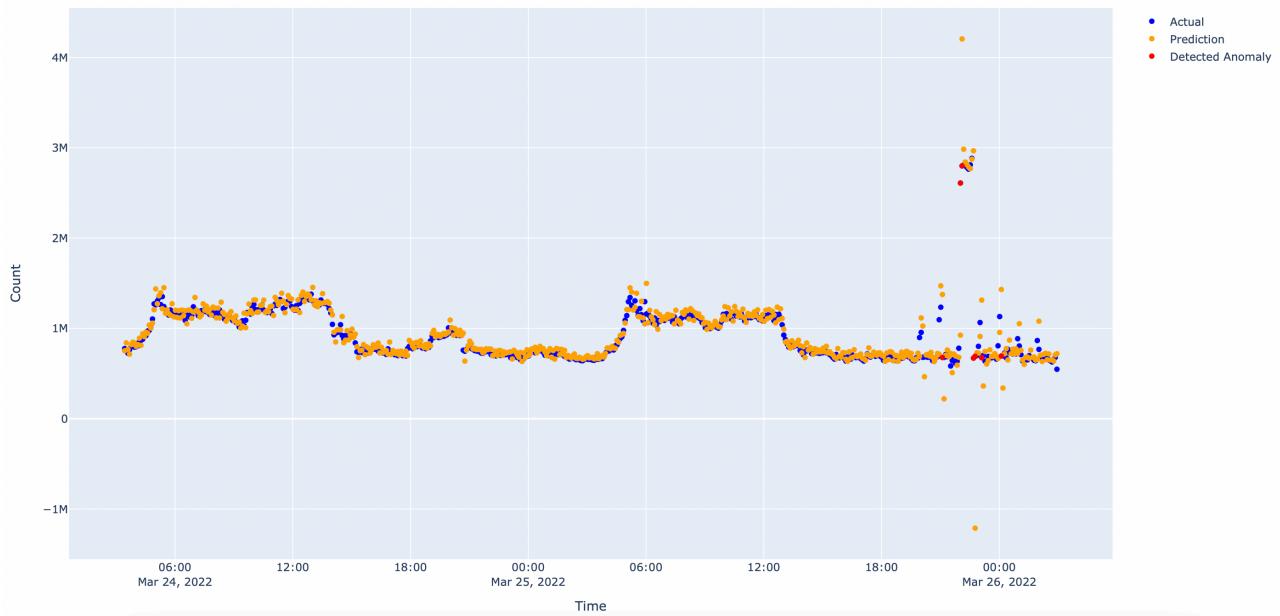
```
(0.10410682954828836, (4, 1, 2))
```

Теперь мы сможем сориентироваться, какие параметры можно использовать для получения хорошего результата. Начнем с обозначенных в начале 1, 1, 1.

```
def pipeline_arima(use_new_model=False, input_model_path='', output_model_path='data/model.pkl'):
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    train, test = data_split(data)
    if use_new_model:
        model = model_arima(train, 1, 1, 1)
        save_model(model, output_model_path)
    else:
        model_pickle = open(input_model_path, "rb")
        model = pickle.load(model_pickle)
    prediction_test = get_predictions_arima(model, train, test)
    res, pred_test = detect(prediction_test, test)
    res.to_csv('data/result.csv')
    pred_test = standardising_res(train, pred_test)
    rmse, mse, mae = validation(pred_test)
    percent_anomaly = len(res) / len(test) * 100
    print("Percent anomaly = ", percent_anomaly)
    print("RMSE = ", rmse, "\nMSE = ", mse, "\nMAE = ", mae)
```

```
pipeline_arima(use_new_model=True)
```

Полученный график:



Вывод программы:

```
Percent anomaly = 1.2259194395796849
RMSE = 0.438818510971249
MSE = 0.1925616855710242
MAE = 0.16366192082928024
```

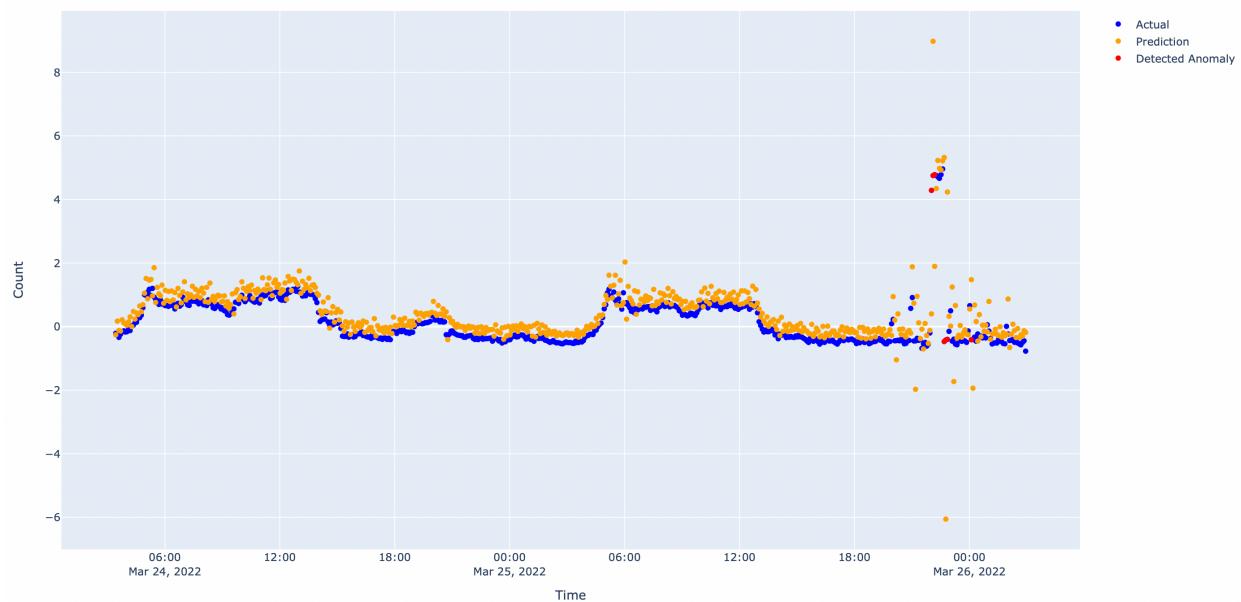
Файл csv с таблицей аномальных значений:

```
,cnt
2022-03-25 21:06:00,677870
2022-03-25 22:01:00,2609252
2022-03-25 22:06:00,2798999
2022-03-25 22:41:00,670747
2022-03-25 22:46:00,690784
2022-03-25 23:06:00,675602
2022-03-26 00:06:00,693855
```

В тестовых данных было найдено 7 аномальных значений, что равняется $\sim 1.2\%$ общего объема тестовой выборки.

Теперь укажем лучшие параметры по результату перебора. Так же используем нормализацию данных.

Полученный график:



Вывод программы:

```
Percent anomaly = 1.2259194395796849
RMSE = 0.6059035767279567
MSE = 0.3671191442917309
MAE = 0.3533360686136413
```

Файл csv:

```
,cnt
2022-03-25 22:01:00,2610006
2022-03-25 22:06:00,2799835
2022-03-25 22:11:00,2811283
2022-03-25 22:46:00,690709
2022-03-25 22:51:00,698357
2022-03-26 00:06:00,693782
```

Отметим, процент аномалий не поменялся, но некоторые из строк оказались другими, а так же возросли ошибки.

Заглянув в файл с параметрами моделей и их ошибками, видим что модель с параметрами 2, 1, 2 имеет так же относительно маленькую ошибку. Давайте используем эти параметры.

```
Percent anomaly = 1.0507880910683012
RMSE = 0.46302537727680415
MSE = 0.21439250000232682
MAE = 0.2221083240837752
```

Процент уменьшился как и ошибки.

```
,cnt
2022-03-25 21:06:00,677790
2022-03-25 22:01:00,2610006
2022-03-25 22:11:00,2811283
2022-03-25 22:41:00,670664
2022-03-25 22:51:00,698357
2022-03-26 00:06:00,693782
```

В результате получаем 6 аномальных значений. Уже лучше.

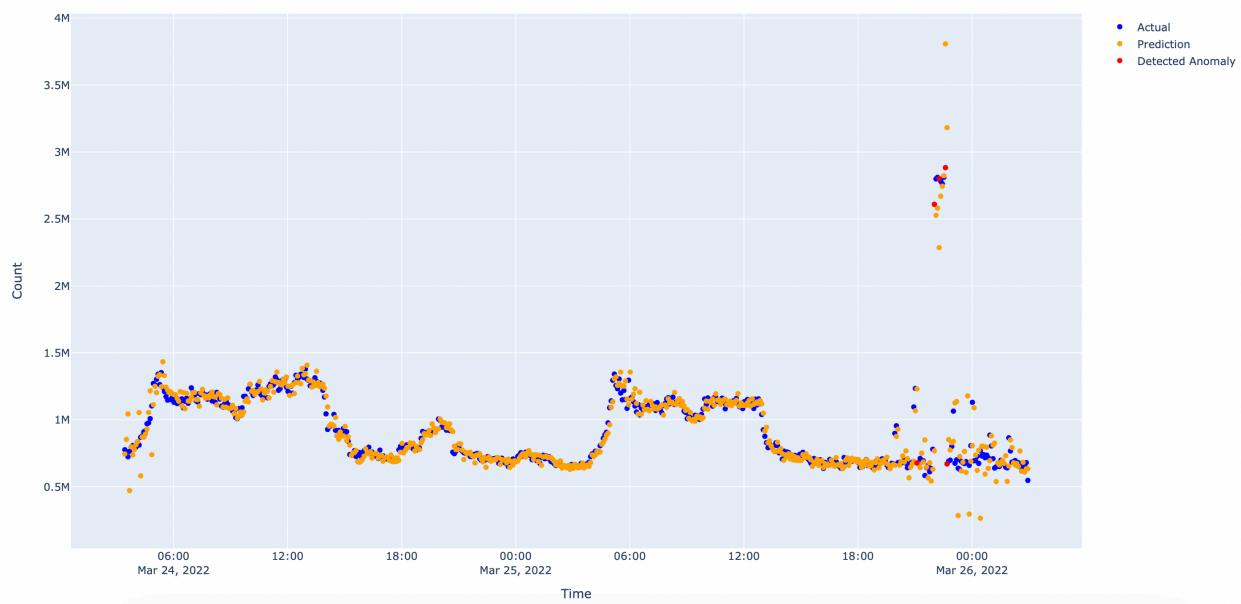
Что выдаст модель с автоматическим подбором параметров? Запустим код.

```
def pipeline_auto_arima(use_new_model=False, input_model_path='', output_model_path='data/model_autoarima.pkl'):
    file = "data/tsData.json"
    data = load_data(file)
    data = data_resample(data)
    train, test = data_split(data)
    train_sqrt, test_sqrt = np.sqrt(train), np.sqrt(test)
    if use_new_model:
        model = model_auto_arima(train_sqrt)
        save_model(model, output_model_path)
    else:
        model_pickle = open(input_model_path, "rb")
        model = pickle.load(model_pickle)
    prediction_test = get_prediction_auto_arima(model, train_sqrt, test_sqrt, test)
    res, pred_test = detect(prediction_test, test)
    res.to_csv('data/result_auto_arima.csv')
    pred_test = standardising_res(train, pred_test)
    rmse, mse, mae = validation(pred_test)
    print(rmse, mse, mae)
    percent_anomaly = len(res) / len(test) * 100
    print("Percent anomaly = ", percent_anomaly)
    print("RMSE = ", rmse, "\nMSE = ", mse, "\nMAE = ", mae)

pipeline_auto_arima(use_new_model=True)
```

```
,cnt  
2022-03-25 21:06:00,677870  
2022-03-25 22:01:00,2609252  
2022-03-25 22:16:00,2800719  
2022-03-25 22:36:00,2883255  
2022-03-25 22:41:00,670747
```

```
Percent anomaly = 0.8756567425569177  
RMSE = 0.39868751573414984  
MSE = 0.158951735202268  
MAE = 0.14481073894468383
```



Мы нашли лучший результат.

ЗАКЛЮЧЕНИЕ

В работе был использован классический метод подгонки модели ARIMA - метод Бокса - Дженкинса.

Это процесс, который использует анализ временных рядов и диагностику для определения подходящих параметров модели ARIMA.

Таким образом, этапы следующие:

1. Идентификация. Использовались графики и сводная статистика для определения трендовых, сезонных и авторегрессионных элементов, чтобы понять величину отклонения и размер требуемой задержки.
2. Оценка параметров. Использовалась программа подбора, чтобы найти коэффициенты регрессионной модели.
3. Проверка модели.

Этот процесс описан в классическом учебнике 1970 года с темой анализ временных рядов Джорджа Бокс и Гвилима Дженкинса.

В результате был достигнут целевой показатель менее 1%. Это означает, что количество аномальных значений не превышает 1%. Значит, можно сделать вывод об эффективности алгоритма.

Удалось реализовать различные подходы к решению задачи, что заняло больше времени, но помогло разобраться в алгоритме прогнозирования и выявления аномалий. При выполнении задачи был получен уникальный для меня опыт.