Построение моделей временного ряда для описания нагрузочных характеристик серверов в кластере

Докладчик:
Накорнеева Юлия, МФТИ ФПМИ Б05-253

Научный руководитель:
Саенко Владимир Иванович, доц. каф. ТиПИ

▶▶▶▶▶

Май, 2025

План доклада

1. Постановка задачи
2. Эмуляция нагрузки на кластер
3. Анализ данных
4. Итоги и перспективы

Постановка задачи

- ## Контекст

Имеется система серверов, объединенных в кластер. Для кластера существенно, чтобы его мощность была достаточной для обработки запросов, но при этом не слишком превосходила эту нагрузку для экономии ресурсов. Это создает необходимость в прогнозировании нагрузки для эффективного управления ресурсами и предотвращения перегрузок.

- ## Цель

Построить модели временного ряда для предсказания изменения нагрузки на сервер. Модель необходима для принятия решения об изменении конфигурации сервера. .

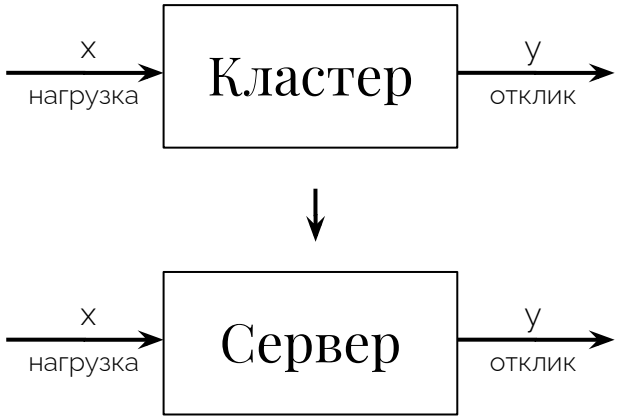

Рисунок 1 – Обобщенная схема приема данных для кластера

Постановка задачи

- Этапы работы:

1. Прием/генерация пакетов
2. Формирование нагрузочной характеристики
3. Предварительная обработка входных данных
4. Анализ данных
5. Формирование рекомендаций и построение моделей.
6. Создание программной системы

Рисунок 2 – Схема полного цикла подготовки данных

Мотивация и актуальность

В современной IT-инфраструктуре компании активно арендуют серверные мощности, и для них крайне важно находить оптимальный баланс между затратами на аренду серверов и скоростью ответа на пользовательские запросы.

Как показывает практика, в условиях высокой конкуренции даже кратковременные задержки в работе системы могут привести к потере клиентов. При этом избыточное выделение ресурсов значительно увеличивает эксплуатационные или арендуемые расходы. В связи с этим любое решение по оперативному управлению вычислительной мощностью серверов оказывается актуальным.

Существующие исследования и их ограничения

Исследования в области применения временных рядов имеют хорошо разработанную научную базу, однако в контексте нашей задачи общего решения нет.

В имеющихся исследованиях обычно решается более общая задача из-за чего блок прогнозирования ограничивается одним методом решения.

1. Прогнозирование нагрузки в облачных системах
Tirado, J. M., Higuero, D., Isaila, F., & Carretero, J. (2011). *Predictive Data Grouping and Placement for Cloud-based Elastic Server Infrastructures.* IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid). DOI: 10.1109/CCGrid.2011.49

2. Прогностическая балансировка кластеров
Sheikhi, S., & Babamir, S. M. (2016). *A Predictive Framework for Load Balancing Clustered Web Servers.* The Journal of Supercomputing, 72, 588–611. DOI: 10.1007/s11227-015-1584-8

Эмуляция нагрузки

Нагрузку можно классифицировать

По частоте:
- низкая,
- средняя,
- высокая,
- пиковая.

По типу:
- равномерная,
- импульсивная,
- сезонная.

По частоте дискретизации и нагрузочной характеристики

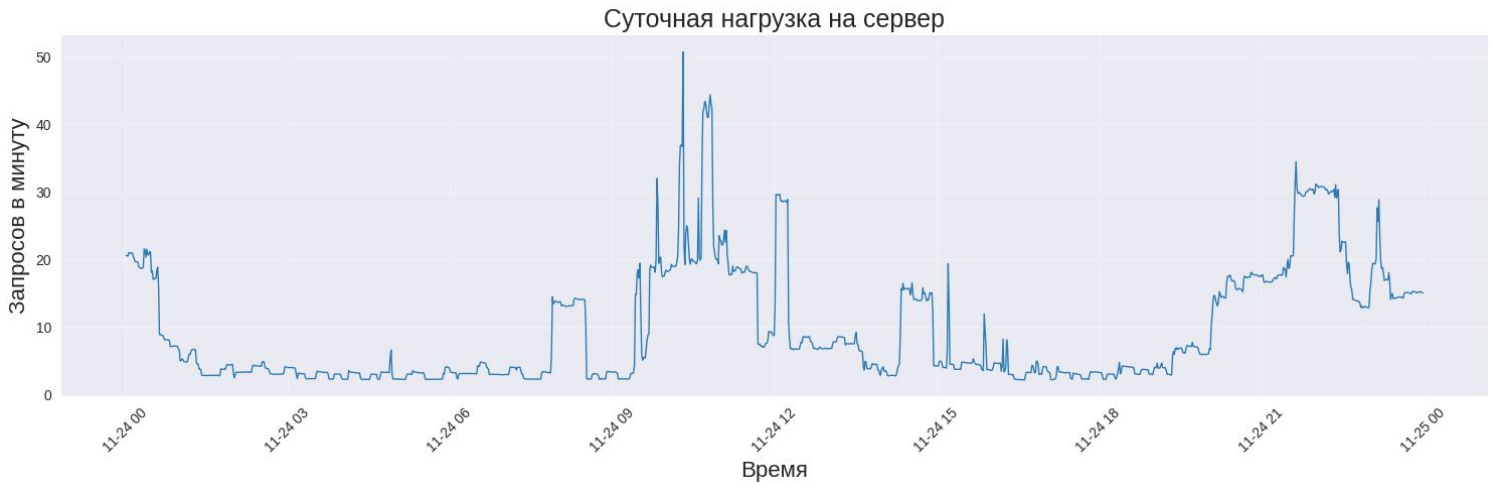

Рисунок 3 – пример нагрузки на сервер за сутки

Формирование нагрузочной характеристики

Временной ряд — это последовательность числовых данных, упорядоченных во времени, в которой каждое значение соответствует определённому моменту или интервалу времени.

В контексте нашей задачи в качестве числовых данных было выбрано количество запросов на сервер в момент времени. Интервалы снятия данных от 30 секунд до 1 минуты.

Таким образом мы свяжем задачу прогнозирования временных рядов с реальной задачей.

Эмуляция нагрузки на кластер

Два подхода в эмуляции

1. Синтетические данные
2. Реальные данные

Синтетические данные

Цель

- Контролируемая проверка моделей
- Воспроизведение редких сценариев

Библиотеки

- Numpy
- Pandas
- Scipy.interpolate

Реализация

Функция-генератор синтетических данных нагрузки на сервер.
Входные параметры:

- Заданные контрольные точки (час, значение) как основу паттерна
- Сплайн-интерполяцию указанной степени для сглаживания
- Заданное временное разрешение для дискретизации данных
- Велечину колебаний

Синтетические данные

Пример сгенерированных данных с шагом дискретизации 3 минуты, степенью 2 и следующими ключевыми точками:

```
    (0, 80),     # Ночной спад
    (3, 75),     # Ночной спад
    (6, 50),     # Ночной минимум
    (9, 150),    # Утренний подъем
    (12, 180),   # Обеденный пик
    (14, 100),   # Послеобеденный спад
    (18, 200),   # Вечерний максимум
    (22, 80)     # Вечерний спад
```

Рисунок 4 – пример сгенерированных синтетических данных

Синтетические данные

Функции генерации на языке python:

```
def generate_daily_pattern(peaks, resolution=1, degree=2):
    """
    Генерирует суточный паттерн нагрузки на основе заданных пиков

    Параметры:
    peaks - список кортежей (час, значение нагрузки) для основных точек паттерна
    resolution - как часто снимаются данные в минутах

    Возвращает:
    Массив значений нагрузки за сутки с заданным разрешением
    """
    hours, values = zip(*peaks)

    # Добавляем первую точку в конец для замыкания цикла
    hours = list(hours) + [24 + hours[0]]
    values = list(values) + [values[0]]

    # Создаем временную ось с заданным разрешением
    day_minutes = 24 * 60
    minute_points = np.linspace(0, day_minutes, day_minutes // resolution + 1)
    hour_points = np.array(hours) * 60

    # Используем сплайн-интерполяцию для гладкого паттерна
    spline = make_interp_spline(hour_points, values, k=degree, bc_type='periodic')
    pattern = spline(minute_points)

    # Заполняем возможные NaN значения (если есть)
    pattern = np.nan_to_num(pattern, nan=np.mean(values))

    return pattern
```

Синтетические данные

Функции генерации на языке python:

```
def generate_server_load_data(
    days=1,
    resolution=1,
    degree=2,
    daily_peaks=None,
    multiplicative_noise=0.3,
    additive_noise_range=10,
):
    """
    Генерирует данные нагрузки на сервер с двумя типами шума:
    - Мультипликативный (процентные колебания)
    - Аддитивный (фиксированные отклонения ±X)

    Параметры:
    - days: количество дней для генерации
    - resolution: частота данных в минутах (по умолчанию 1)
    - daily_peaks: список кортежей (час, значение) для суточного паттерна
    - multiplicative_noise: уровень относительного шума (0-1)
    - additive_noise_range: абсолютный разброс аддитивного шума (±X)

    Возвращает:
    - DataFrame с колонками: timestamp, load
    """
    # Генерируем базовый суточный паттерн
    daily_pattern = generate_daily_pattern(daily_peaks, resolution, degree)

    # Количество точек в сутки
    points_per_day = len(daily_pattern)
```

Синтетические данные

Функции генерации на языке python:

```
# Создаем временную шкалу
    start_date = datetime.now().replace(hour=0, minute=0, second=0, microsecond=0)
    timestamps = [start_date + timedelta(minutes=resolution*i)
                  for i in range(points_per_day * days)]

    # Повторяем паттерн для каждого дня
    load_data = np.tile(daily_pattern, days)

    # 1. Мультипликативный шум (умножаем на случайный коэффициент)
    if multiplicative_noise > 0:
        noise = np.random.normal(1, multiplicative_noise/3, size=len(load_data))
        noise = np.clip(noise, 1 - multiplicative_noise, 1 + multiplicative_noise)
        load_data = load_data * noise

    # 2. Аддитивный шум (добавляем случайное значение ±additive_noise_range)
    if additive_noise_range > 0:
        additive_noise = np.random.uniform(
            -additive_noise_range,
            additive_noise_range,
            size=len(load_data)
        )
        load_data = load_data + additive_noise

    # Создаем DataFrame и убираем отрицательные значения
    df = pd.DataFrame({
        'timestamp': timestamps,
        'load': np.clip(load_data, 0, None)  # load_data.clip(0)
    })

    return df
```

Реальные данные

Цель

- Реальное поведение
- Проверка системы

Библиотеки

- Открытые датасеты
- Pandas

Реализация

Используется открытый датасет с `https://archive.ics.uci.edu`

Он обрабатывается на наличие пропусков значений и необходимую дискретизацию

Реальные данные

График реальных данных со степенью дискретизации 1 минута

Рисунок 5 – пример сгенерированных реальных данных

Реальные данные

Функции генерации на языке python:

```
def plot_power_consumption(resolution=1, days=1):
    """
    Функция реальных данных.

    Параметры:
    resolution minutes : шаг дискретизации в минутах (по умолчанию 1)
    days : количество дней для отображения (по умолчанию 1)
    """
    # Параметры
    minutes_per_day = 60 * 24  # 1440 минут в сутках
    total_points = int(minutes_per_day * days / resolution)

    # Загрузка данных
    url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00235/household_power_consumption.zip"
    df = pd.read_csv(url, compression='zip', delimiter=';',
                     parse_dates={'datetime': ['Date', 'Time']}

    # Обработка данных
    df = df.dropna(subset=['datetime', 'Global_active_power'])
    df['Global_active_power'] = pd.to_numeric(df['Global_active_power']) * 10
    df = df.sort_values('datetime').tail(total_points)

    # Создание временной шкалы
    start_date = datetime.now().replace(hour=0, minute=0, second=0, microsecond=0)
    timestamps = [start_date + timedelta(minutes=resolution_minutes*i)
                  for i in range(total_points)]

    # Создание DataFrame
    real_data = pd.DataFrame({
        'timestamp': timestamps,
        'load': np.clip(df['Global_active_power'].values, 0, None)
    })

    return real_data
```

Анализ данных:

Состоит из двух направлений исследования:

- Краткосрочный прогноз на 20 минут вперед
- Построение паттерна суточной нагрузки

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Ручная настройка параметров

Выбор теста на стационарность в модели ARIMA: расширенный тест Дики-Фуллер (ADF)

При проверке стационарности временного ряда перед построением модели ARIMA часто используется расширенный тест Дики-Фуллер (ADF). Этот тест помогает определить наличие единичного корня, что свидетельствует о нестационарности ряда. У ADF-теста есть три варианта реализации, выбор которых зависит от структуры данных:

1. Тест для ряда с нулевым средним
   Проверяет гипотезу о единичном корне против альтернативы стационарности ряда с нулевым средним.
   Когда использовать? Если предполагается, что ряд имеет нулевое среднее значение (например, после центрирования).
2. Тест с константой (общий случай)
   Проверяет гипотезу о единичном корне против стационарности ряда с ненулевым средним, но без тренда.
   Когда использовать? Если ряд колеблется вокруг некоторого постоянного уровня, но не имеет выраженного тренда. Использование теста только с константой (без тренда) повышает его мощность.
3. Тест с линейным трендом
   Проверяет гипотезу о единичном корне против стационарности ряда с линейным трендом.
   Когда использовать? Если ряд содержит детерминированный долгосрочный тренд (например, монотонный рост или убывание).

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Ручная настройка параметров

Исходя из графика нашего временного ряда, наиболее целесообразно использовать общий случай ADF

Фунция общего случая ADF

```
[ ]  from statsmodels.tsa.stattools import adfuller

     # Проверка стационарности
     def adf_test(series):
         result = adfuller(series, autolag='AIC')
         print('ADF Statistic:', result[0])
         print('p-value:', result[1])
         print('Critical Values:', result[4])
         if result[1] <= 0.05:
             print("Ряд стационарен (отвергаем H0)")
         else:
             print("Ряд нестационарен (не отвергаем H0)")
```

Рисунок 5 – функция проверки стационарности ряда

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Ручная настройка параметров

1) Проверяем исходный ряд на стационарность

```
[ ] adf_test(data)
```

```
ADF Statistic: -1.2490705588866586
p-value: 0.6521389045759629
Critical Values: {'1%': -3.4489583388155194, '5%': -2.869739378430086, '10%': -2.5711381780459}
Ряд нестационарен (не отвергаем H0)
```

2) Проверяем ряд после дифференцирования

```
[ ] # Первое дифференцирование
    diff_1 = data['Requests'].diff().dropna()
    adf_test(diff_1)
```

```
ADF Statistic: -14.698725508333125
p-value: 2.969870894669899e-27
Critical Values: {'1%': -3.4489583388155194, '5%': -2.869739378430086, '10%': -2.5711381780459}
Ряд стационарен (отвергаем H0)
```

Значит d = 1

Рисунок 6 – определение параметра d

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Ручная настройка параметров

Подбор параметров p (AR) и q (MA)

Для их определения нам надо изучить автокорреляционную (ACF) и частично автокорреляционную(PACF) функции для ряда первых разностей.

ACF поможет нам определить q, т. к. по ее коррелограмме можно определить количество автокорреляционных коэффициентов сильно отличных от 0 в модели MA

PACF поможет нам определить p, т. к. по ее коррелограмме можно определить максимальный номер коэффициента сильно отличный от 0 в модели AR.

```
[ ]  from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
     import matplotlib.pyplot as plt

     # Графики для дифференцированного ряда
     plot_acf(diff_1, lags=20)
     plot_pacf(diff_1, lags=20)
     plt.show()
```

Рисунок 7 – код ACF и PACF

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Ручная настройка параметров

Подбор параметров p (AR) и q (MA)

Можно сделать вывод, что

* p = 1 или 2 или 3
* q = 1 или 2

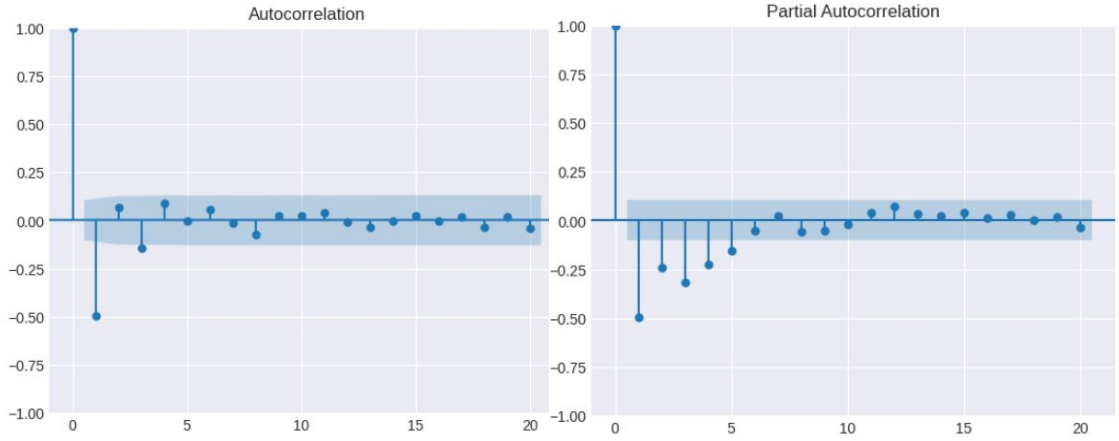

Рисунок 8 – графики ACF и PACF

Анализ данных:

ARIMA

Определение параметров модели (p, d, q) - Автоматическая настройка параметров

Автоматическая настройка параметров дала наилучший результат модели с параметрами (3, 1, 1), что соответствует диапазону параметров подобранных вручную

```
[ ]  from pmdarima import auto_arima

     model = auto_arima(
         data['Requests'],
         seasonal=False,  # у нас нет сезонности
         stepwise=True,   # ускоренный подбор
         trace=True       # вывод подбора
     )
     print(model.summary())
```

Рисунок 9 – код автоматического подбора параметров

Анализ данных:

ARIMA

В ходе применения модели был получен результат прогнозирования временного ряда на 20 минут

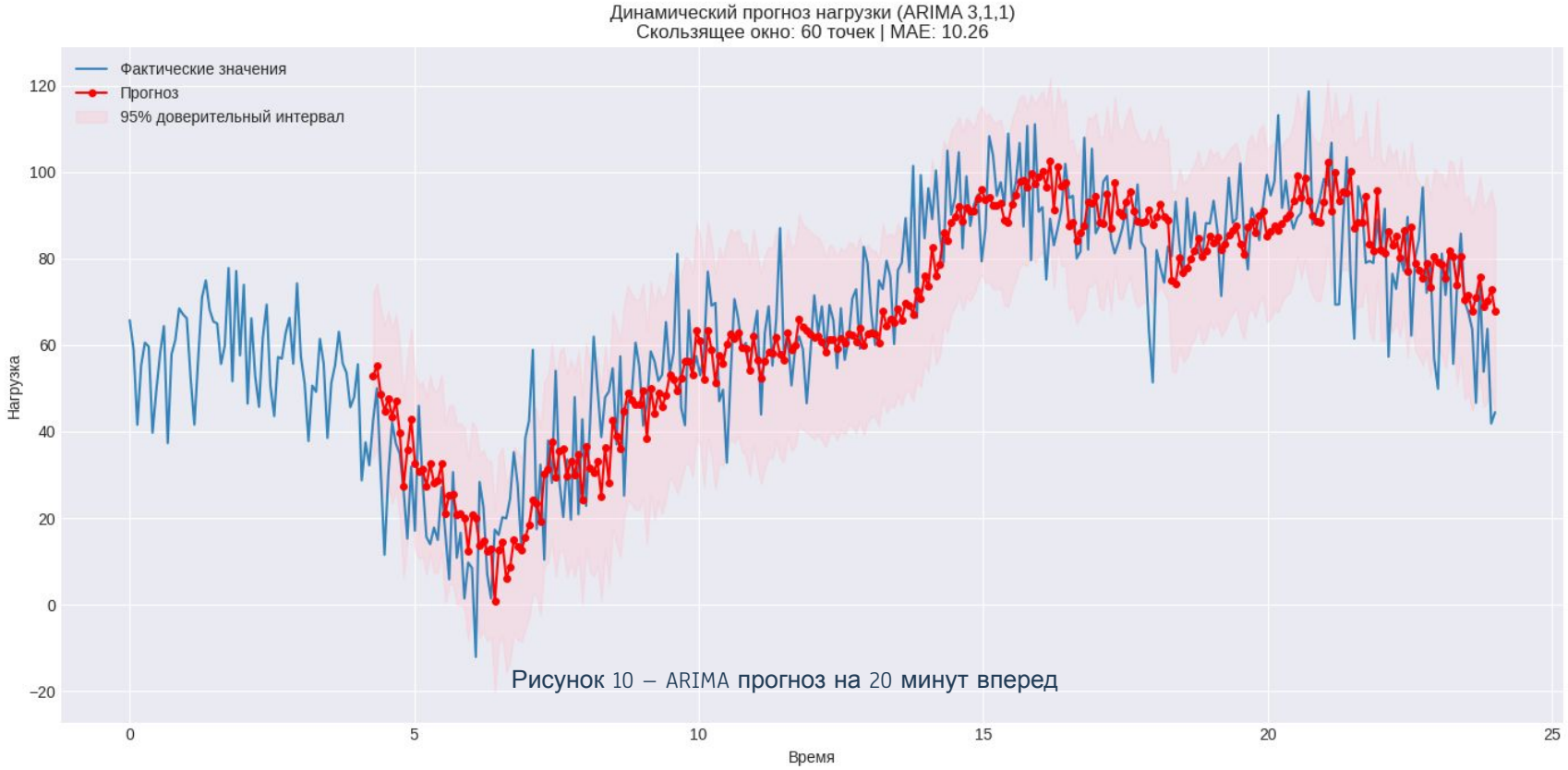

Рисунок 10 – ARIMA прогноз на 20 минут вперед

Анализ данных:

ES

В ходе применения модели был получен результат прогнозирования временного ряда на 20 минут

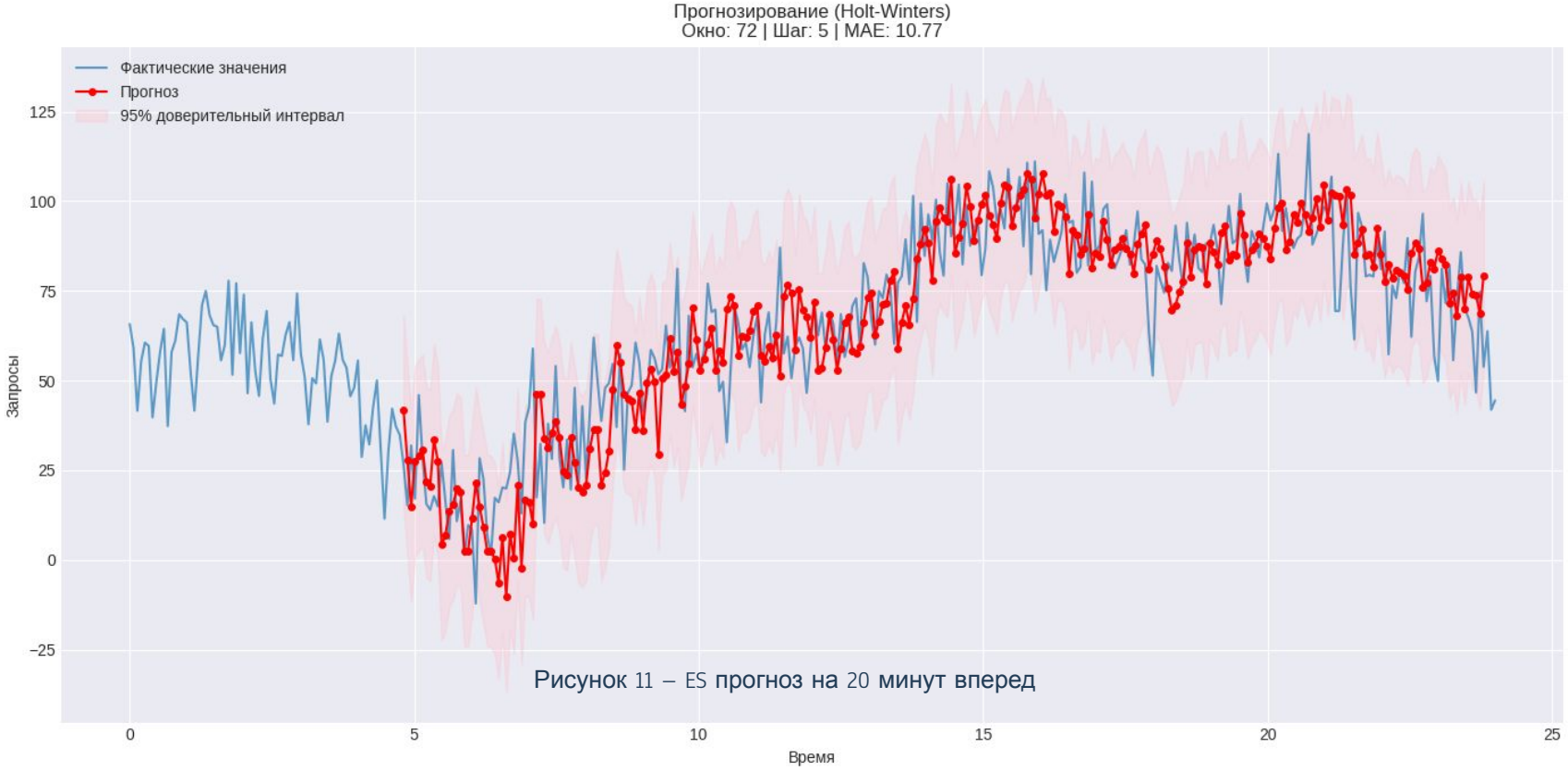

Рисунок 11 – ES прогноз на 20 минут вперед

Анализ данных:

ARIMA

Пример прогнозирования на реальных данных на 20 минут вперед

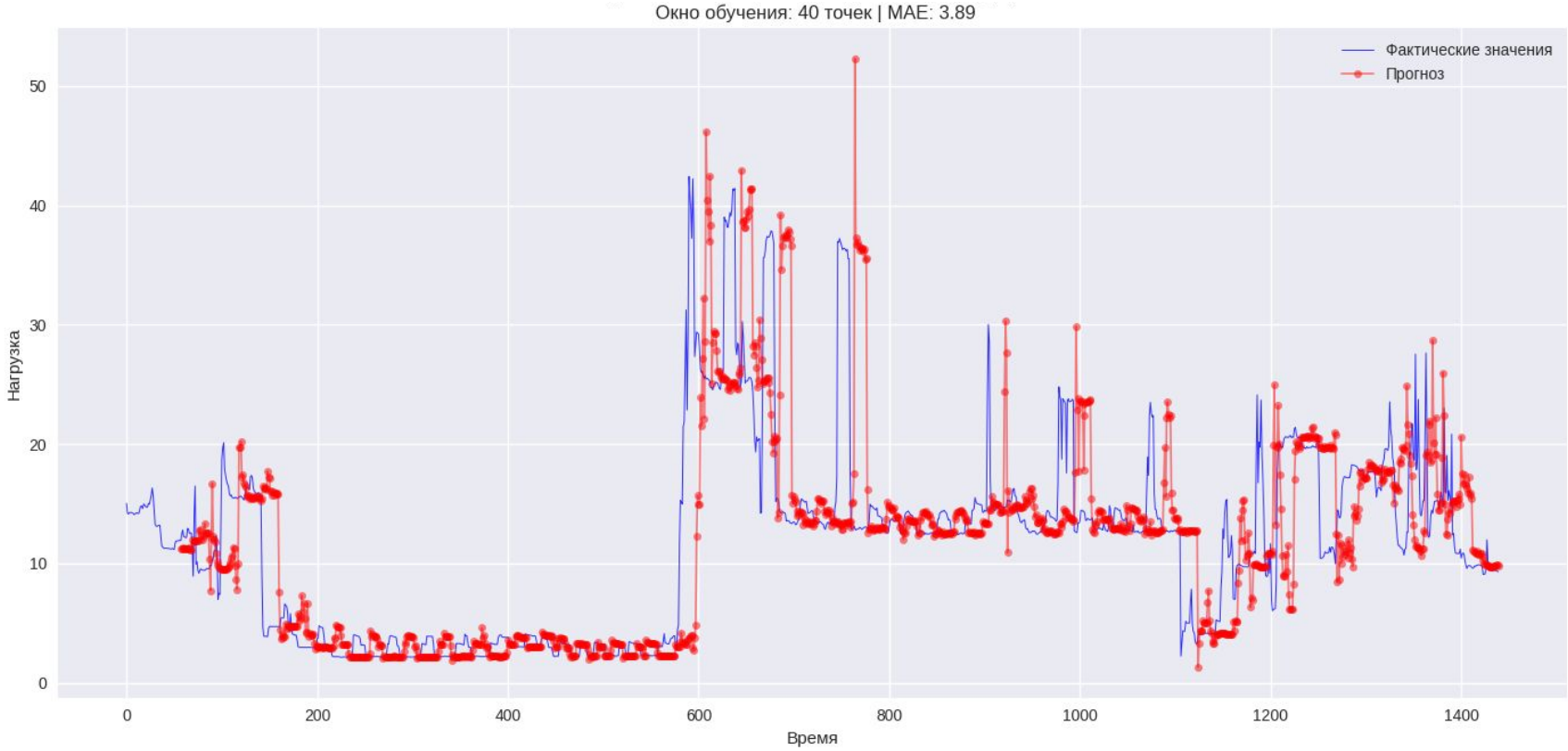

Рисунок 12 – ES прогноз на 20 минут вперед на реальных данных

Анализ данных:

ES

Пример прогнозирования на реальных данных на 20 минут вперед

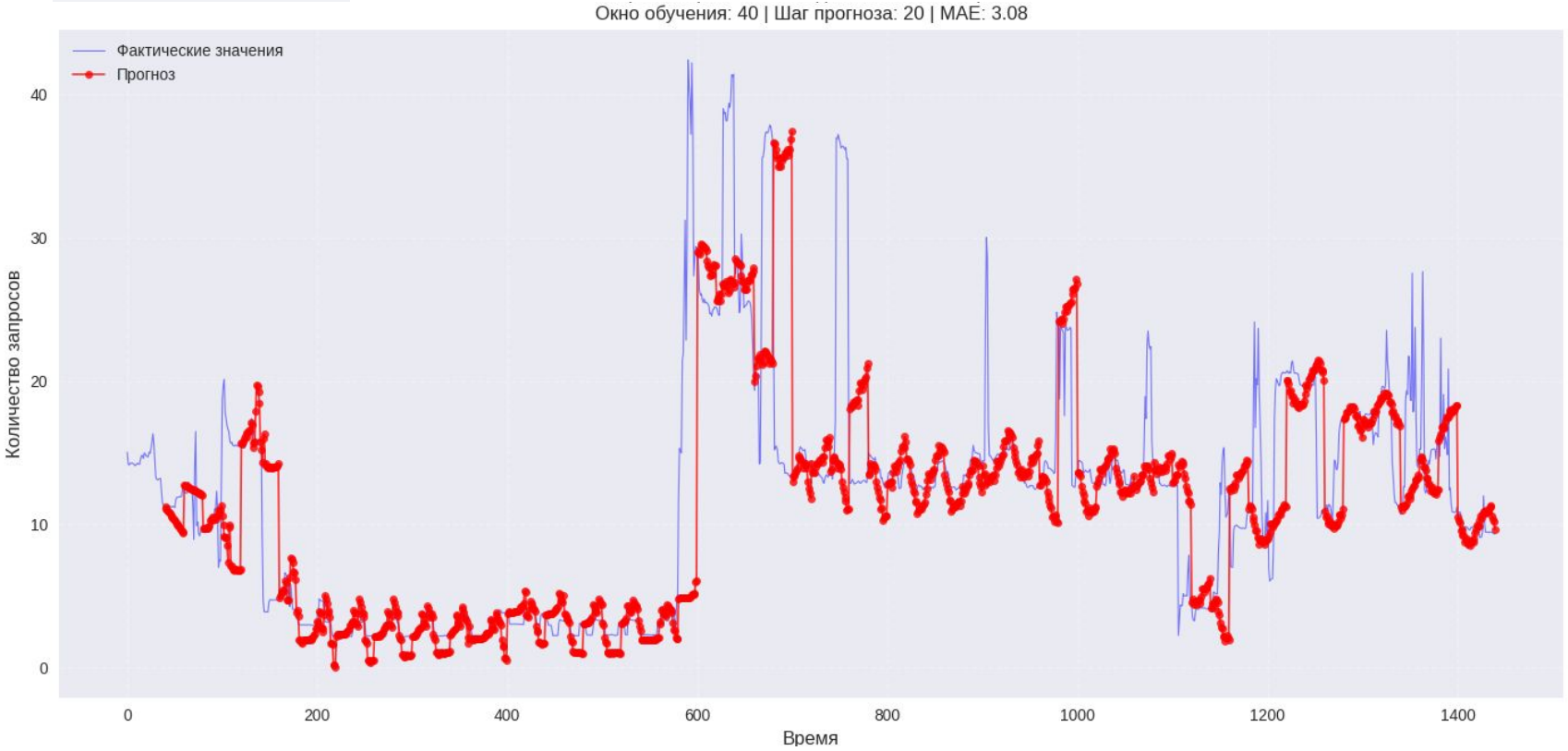

Рисунок 12 – ES прогноз на 20 минут вперед на реальных данных

Анализ данных:

Spline

Состоит из двух этапов:

1. Нахождение опорных точек
2. Построение сплайна

Анализ данных:

Spline

Нахождение опорных точек

Код функции:

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.cluster import KMeans

def find_upper_points(data, value_col='load', n_clusters=20,
random_state=42):
    """
    Находит верхние точки временного ряда методом K-средних.
    Для каждого кластера возвращает точку с максимальным значением.

    Параметры:
    ----------
    data : pd.DataFrame
        Исходные данные с колонками 'timestamp' и value_col
    value_col : str, optional (default='load')
        Название колонки со значениями временного ряда
    n_clusters : int, optional (default=20)
        Количество кластеров (контрольных точек)
    random_state : int, optional (default=42)
        Для воспроизводимости результатов

    Возвращает:
    -----------
    upper_points : np.array
        Массив верхних точек в формате [[timestamp, value], ...]
    """
```

Анализ данных:

Spline

Нахождение опорных точек

Код функции:

```
    # Подготовка данных для кластеризации
    X = np.column_stack((
        np.arange(len(data)),  # Индексы как временная ось
        data[value_col].values # Значения временного ряда
    ))

    # Кластеризация K-means
    kmeans = KMeans(n_clusters=n_clusters, random_state=random_state)
    kmeans.fit(X)

    # Получаем метки кластеров для каждой точки
    labels = kmeans.labels_

    # Находим верхние точки (с максимальными значениями) для каждого
кластера
    upper_points = []
    for i in range(n_clusters):
        cluster_points = X[labels == i]
        if len(cluster_points) > 0:
            # Находим индекс точки с максимальным значением (второй
столбец)
            max_idx = np.argmax(cluster_points[:, 1])
            upper_points.append(cluster_points[max_idx])

    upper_points = np.array(upper_points)

    # Сортируем по времени (первый столбец)
    upper_points = upper_points[upper_points[:, 0].argsort()]

    return upper_points
```

Анализ данных:

Spline

Нахождение опорных точек

Визуализация на синтетических данных:

Рисунок 13 – Опорный точки на синтетических данных

Анализ данных:

Spline

Нахождение опорных точек

Визуализация на реальных данных:

Рисунок 14 – Опорный точки на реальных данных

Анализ данных:

Spline

Построение сплайна

Код функции:

```
from scipy.interpolate import make_interp_spline

def plot bspline from_control_points(control_points, data,
value col='load'):
    x = control points[:, 0].astype(int)
    y = control_points[:, 1]

    # Создаем B-сплайн
    bspline = make_interp_spline(x, y, 2)

    # Генерируем точки для гладкого сплайна
    x new = np.linspace(x.min(), x.max(), 1000)
    y_new = bspline(x_new)

    # Визуализация (аналогично предыдущему примеру)
    plt.figure(figsize=(15, 5))
    plt.plot(data['timestamp'], data[value_col], color='#2c7bb6',
alpha=0.7, label='Исходные данные')
    plt.plot(data['timestamp'].iloc[x new.astype(int)], y_new,
color='#fdae61', linewidth=2, label='B-сплайн')
    plt.scatter(data['timestamp'].iloc[x], y, color='#d7191c', s=50,
label='Контрольные точки')
    plt.title('Аппроксимация B-сплайном')
    plt.legend()
    plt.grid(True, alpha=0.3)
    plt.xticks(rotation=45)
    plt.tight layout()
    plt.show()
```

Анализ данных:

Spline

Построение сплайна

Визуализация на синтетических данных:

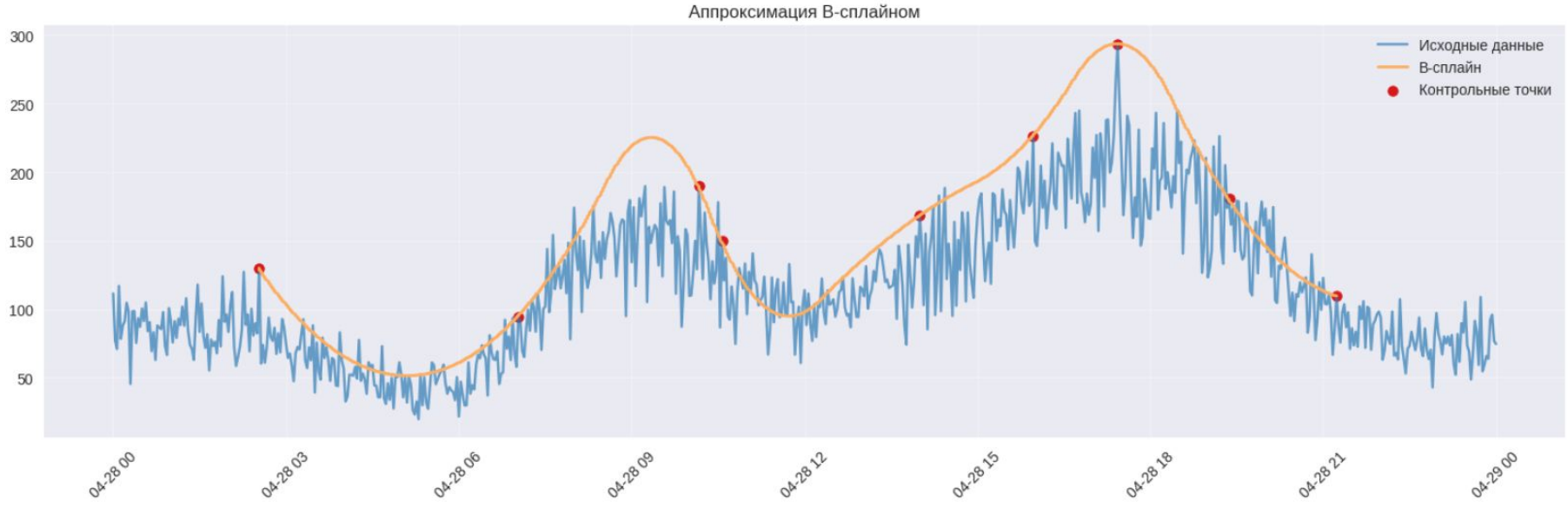

Рисунок 15 – Сплайн на синтетических данных

Анализ данных:

Spline

Построение сплайна

Визуализация на реальных данных:

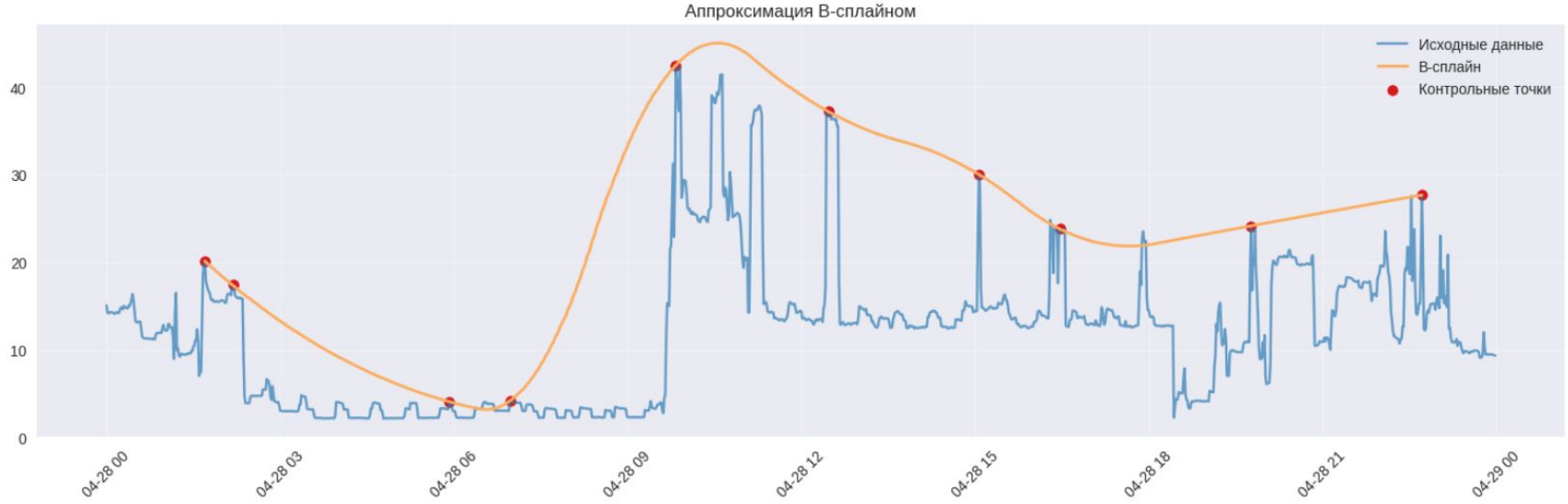

Рисунок 15 – Сплайн на реальных данных

Результаты

- Реализованы блоки:
    - формирования (эмуляции) нагрузки
    - формирования нагрузочных характеристик
    - предварительной обработки данных
    - анализа данных
    - формирование рекомендаций и построение моделей
- Перспективы: использование в управлении данных

Литература

1. Броквелл П.Дж., Дэвис Р.А. Введение в анализ временных рядов и прогнозирование. Пер. с англ./ Под ред. Д.С. Шалымова. Москва: Техносфера, 2017. - 464 с..

2. Ширяев, А. Н. Вероятностно-статистические методы анализа временных рядов / А. Н. Ширяев. — Москва : МГУ имени Ломоносова, 2010. — 288 с.

3. Андрианова Е. Г., Головин С. А., Зыков С. В., Лесько С. А., Чукалина Е. Р. Обзор современных моделей и методов анализа временных рядов динамики процессов в социальных, экономических и социотехнических системах // Russian Technological Journal. – 2020. – Т. 8, № 4. – С. 7–45

Источники

1. Документация KMeans
   https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html
2. Документация B-Spline
   https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.make_interp_spline.html
3. Документация ARIMA
   https://www.statsmodels.org/stable/generated/statsmodels.tsa.arima.model.ARIMA.html
4. Докуиентация ES
   https://www.statsmodels.org/stable/generated/statsmodels.tsa.holtwinters.ExponentialSmoothing.html

Спасибо за внимание!

Накорнеева Юлия

nakorneeva.ya@phystech.edu

Tg: @yulianakorneeva