

Приложение на Flask по прогнозированию цен: интеграция машинного обучения и интерактивной визуализации данных

Лапушкина Елена Павловна

Приамурский государственный университет им. Шолом-Алейхема

Студент

Аннотация

В данной статье описывается процесс создания веб-приложения на Flask, которое позволяет пользователям делать прогнозы цены на основе обученных моделей машинного обучения. Используемые модели, такие как, Linear Regression, XGBoost, Random Forest были предварительно обучены на данных о цене ноутбуков. Статья включает в себя пошаговое руководство по созданию веб-приложения и интеграции моделей.

Ключевые слова: Модель обучения, прогнозирование, Flask.

Price forecasting app on Flask: integration of machine learning and interactive data visualization

Lapushkina Elena Pavlovna

Sholom-Aleichem Priamursky State University

Student

Abstract

This article describes the process of creating a web application on Flask that allows users to make price forecasts based on trained machine learning models. The models used, such as Linear Regression, XGBoost, and Random Forest, were pre-trained on laptop price data. The article includes a step-by-step guide on how to create an application and integrate models.

Keywords: Learning Model, Forecasting, Flask.

1 Введение

1.1 Актуальность

Актуальность визуализации данных в веб-приложениях Flask обусловлена ростом объема и скорости поступления данных, необходимостью оперативно превращать их в понятные инсайты для пользователей. Современные приложения требуют интерактивности, гибкости и масштабируемости: визуализация помогает быстро выявлять тренды, аномалии и зависимости, поддерживает принятие обоснованных решений, улучшает пользовательский опыт, упрощает анализ сложных наборов данных и повышает конкурентоспособность продукта.

1.2 Обзор исследований

А. Оразгулыев, С.А. Гараджаева в своем исследовании рассмотрели важность визуализации данных как неотъемлемой части анализа и представления результатов исследований [1]. А.С. Ремнев, А.Р. Муртазина в своем исследовании рассматривали влияние Python на анализ данных, подчеркнув его адаптируемость, интуитивность синтаксиса и обширную экосистему модулей, а также роль открытой поддержки сообщества [2]. Т.С.Волокитина исследовала преимущества о возможности Google Colab, для изучения технологий машинного обучения и нейронных сетей. Предоставила теоретическую справку и анализ возможностей Google Colab [3]. В.Н.Клячкин, В.Н. Ковальногов, В.В. Шеркунов, Х. Мохамед в своей научной работе посвятили исследованию анализа влияния различных факторов на эффективность работы горелочных устройств, оцениваемую по температуре дымовых газов, с применением методов машинного обучения [4]. А.В. Дагаев рассмотрел, как анализ развития языков программирования позволяет упростить и ускорить процесс анализа данных. [5]. М.А. Полячкова, А.Н.Еримова исследовали популярные веб-фреймворки для Python, такие как Django и Flask, чтобы выявить их сильные и слабые стороны, что поможет программистам и работодателям выбирать подходящие инструменты для разработки [6].

1.3 Цель исследования

Целью данного исследования является разработка и внедрение веб-приложения на основе Flask, которое предоставляет пользователям возможность осуществлять прогнозирование цен на ноутбуки, используя предобученные модели машинного обучения.

2 Материалы и методы

Для создания веб-приложения был выбран Flask — легковесный и гибкий фреймворк для разработки веб-приложений на Python.

Ссылка на датасет:
<https://docs.google.com/spreadsheets/d/1hr87pheAovKvz9pUnWwT2f-bXrDMgf-T/edit?usp=sharing&ouid=110343077311761014179&rtpof=true&sd=true>

Сервис Google Colab. Ссылка на ноутбук:
<https://colab.research.google.com/drive/1KD6nXf9m4hrBiFWxGxkFX5y8mOZeU1L1?usp=sharing>.

3 Результаты и дискуссии

Первым шагом будет загрузка датасета в ноутбук Google Colab, для проведения этапов исследования (рис. 1).

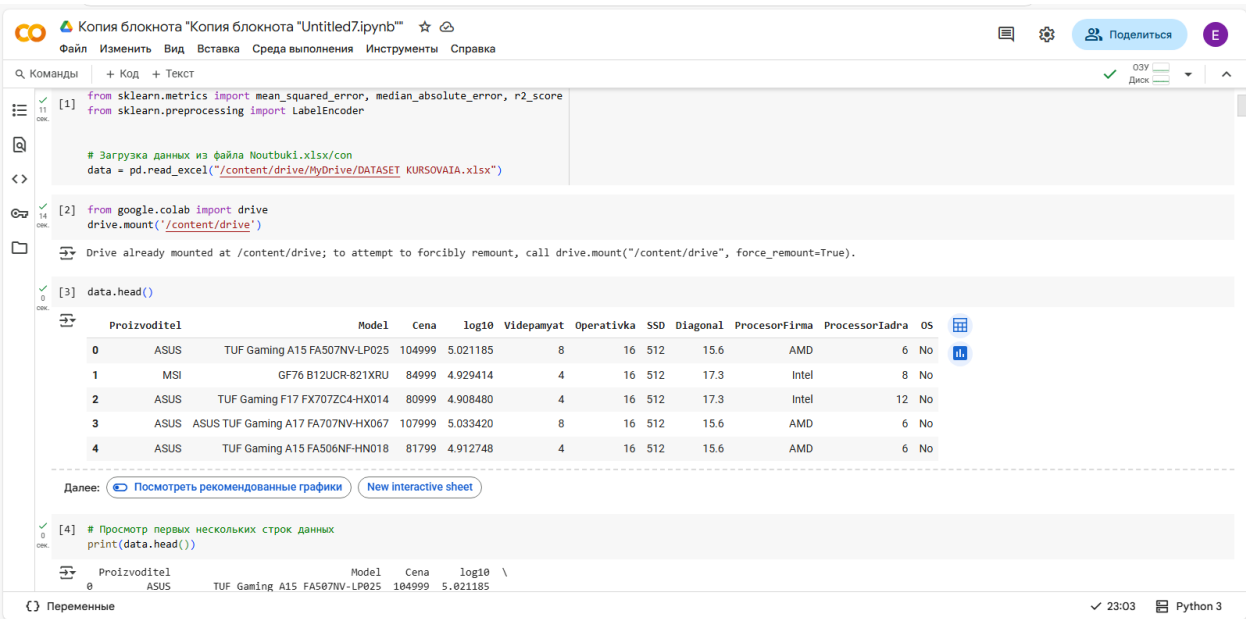


Рисунок 1- Загрузка данных

Далее с помощью описания кода провести анализ данных (рис. 2).

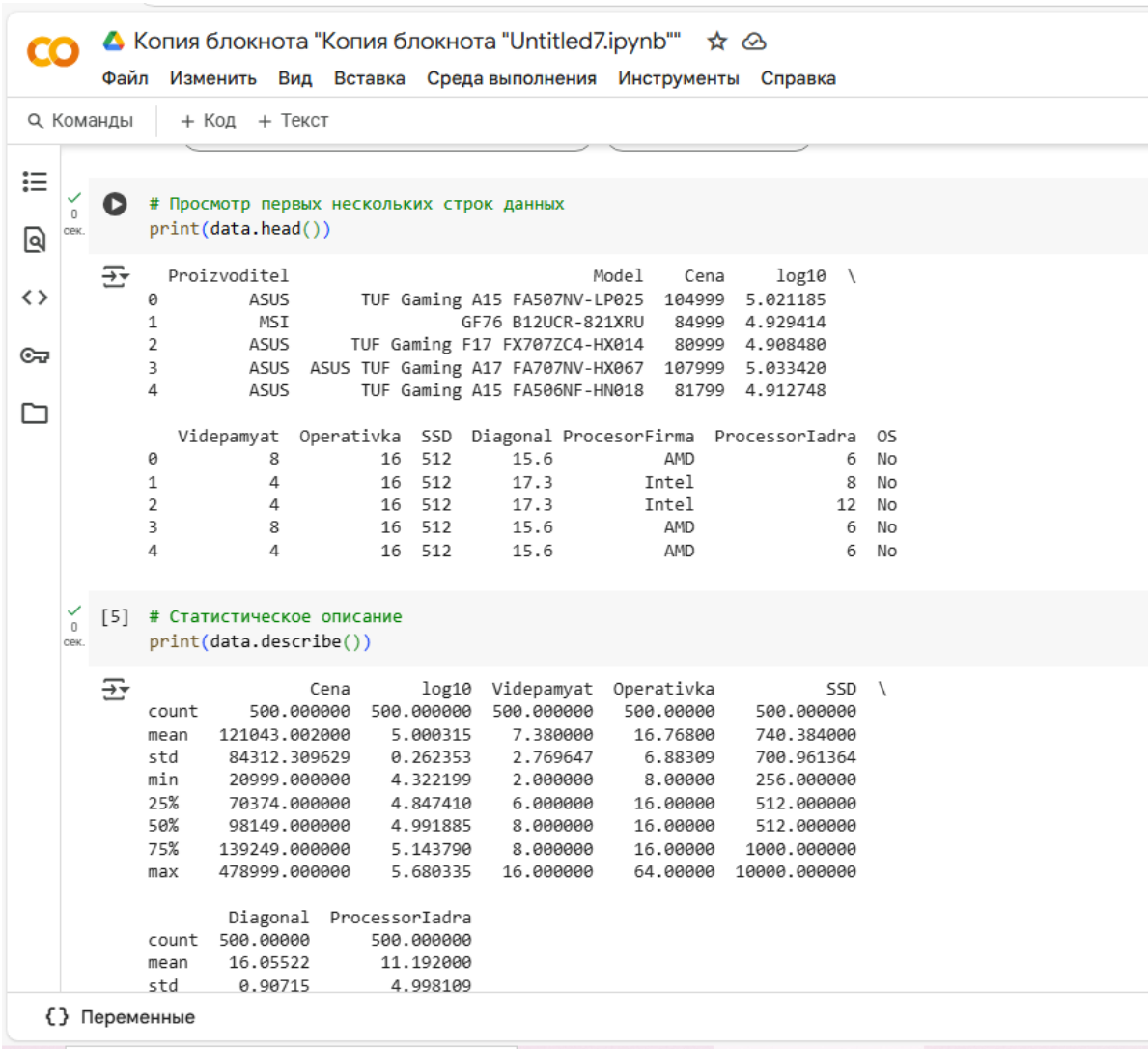


Рисунок 2- Анализ данных

Чтобы, выявить взаимосвязи между различными переменными набора данных, необходимо построить матрицу корреляции. Это поможет определить, какие факторы могут оказывать влияние друг на друга, что в свою очередь может быть полезно для дальнейшего анализа, прогнозирования или построения моделей (рис.3).

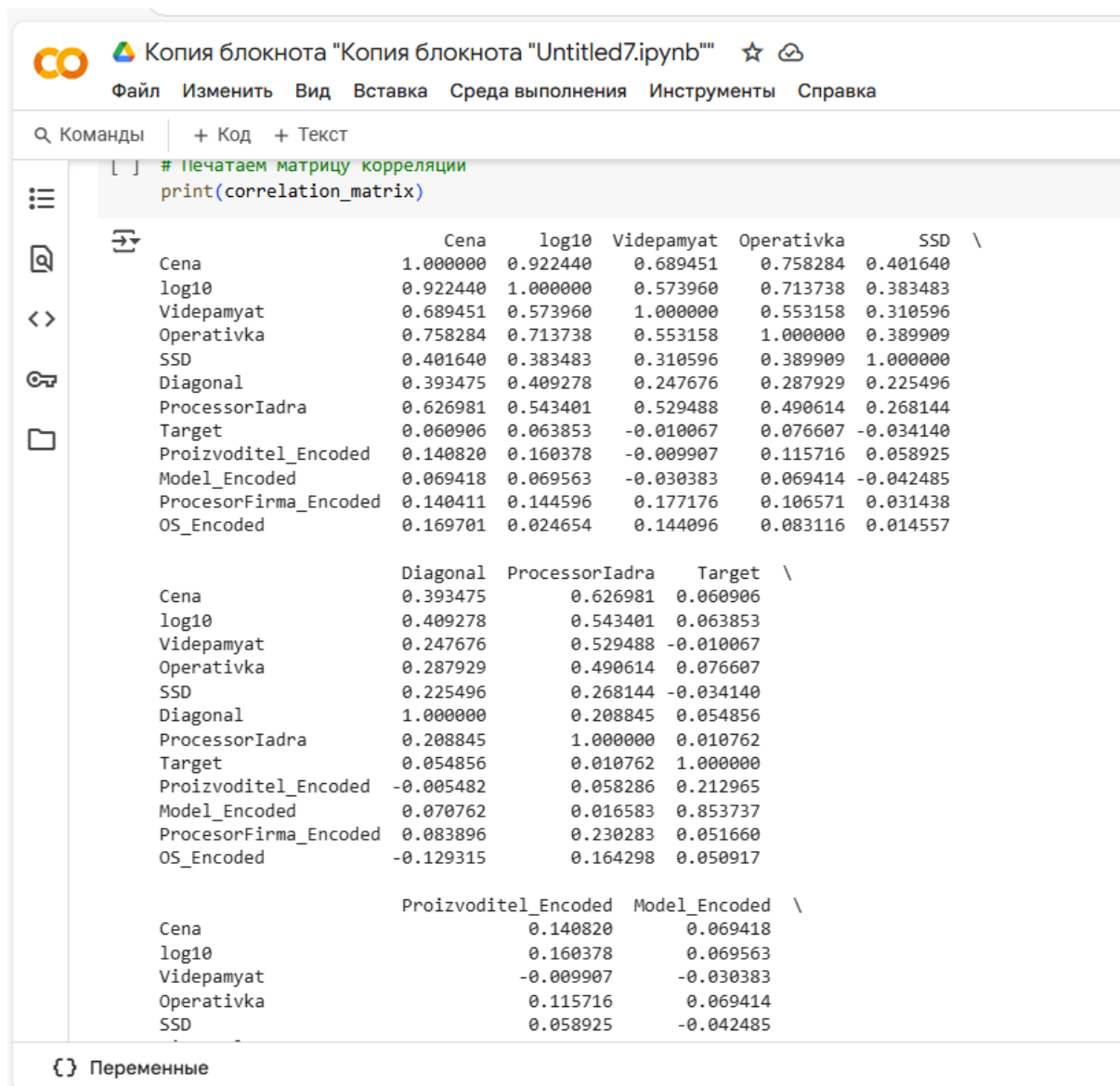


Рисунок 3- Матрица корреляции

В данном датасет, некоторые колонки имеют категориальное значение, применено Target Encoding, для их кодирования в числовые значения (рис. 4).

```

# Применяем Target Encoding для 'Proizvoditel'
mean_proizvoditel = df.groupby('Proizvoditel')['PopularityCoefficient'].mean().to_dict()
df['Proizvoditel_Encoded'] = df['Proizvoditel'].map(mean_proizvoditel).fillna(0)

# Применяем Target Encoding для 'Model'
mean_model = df.groupby('Model')['PopularityCoefficient'].mean().to_dict()
df['Model_Encoded'] = df['Model'].map(mean_model).fillna(0)

# Применяем Target Encoding для 'ProcesorFirma'
mean_procesorfirma = df.groupby('ProcesorFirma')['PopularityCoefficient'].mean().to_dict()
df['ProcesorFirma_Encoded'] = df['ProcesorFirma'].map(mean_procesorfirma).fillna(0)

# Применяем Target Encoding для 'OS'
mean_os = df.groupby('OS')['PopularityCoefficient'].mean().to_dict()
df['OS_Encoded'] = df['OS'].map(mean_os).fillna(0)

# Создаем новую колонку 'efficiency', используя закодированные значения
df['efficiency'] = df[['Proizvoditel_Encoded', 'Model_Encoded', 'ProcesorFirma_Encoded', 'OS_Encoded']].mean(axis=1)

# Проверка наличия колонки 'efficiency' в DataFrame
print(df[['Proizvoditel_Encoded', 'Model_Encoded', 'ProcesorFirma_Encoded', 'OS_Encoded', 'efficiency']].head())

# Вычисление корреляционной матрицы
correlation_matrix = df[['Proizvoditel_Encoded', 'Model_Encoded', 'ProcesorFirma_Encoded', 'OS_Encoded', 'efficiency',
                        'PopularityCoefficient', 'Cena', 'log10', 'Videpamyat', 'Operativka', 'SSD', 'Diagonal',
                        'ProcessorIadra']].corr()

# Построение матрицы корреляции
plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True)
plt.title('Матрица корреляции закодированных переменных и других характеристик', fontsize=16)

```

Рисунок 4- Кодирование категориальных данных методом Target Encoding

Выполнить обучение и оценку моделей, а также вычисление и вывод метрик (рис. 5).

```

[23] results = {}
for model_name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    mse = mean_squared_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)
    results[model_name] = {
        'MSE': mse,
        'R2': r2
    }
results_df = pd.DataFrame(results).T
print("Results:")
print(results_df)

```

	MSE	R2
Linear Regression	2.089726e+09	0.741139
Random Forest	1.412010e+09	0.825090
Decision Tree	1.815617e+09	0.775093
Support Vector Regressor	9.167508e+09	-0.135610
K-Nearest Neighbors	1.851159e+09	0.770691
XGBoost	1.433654e+09	0.822408
AdaBoost	1.600117e+09	0.801788

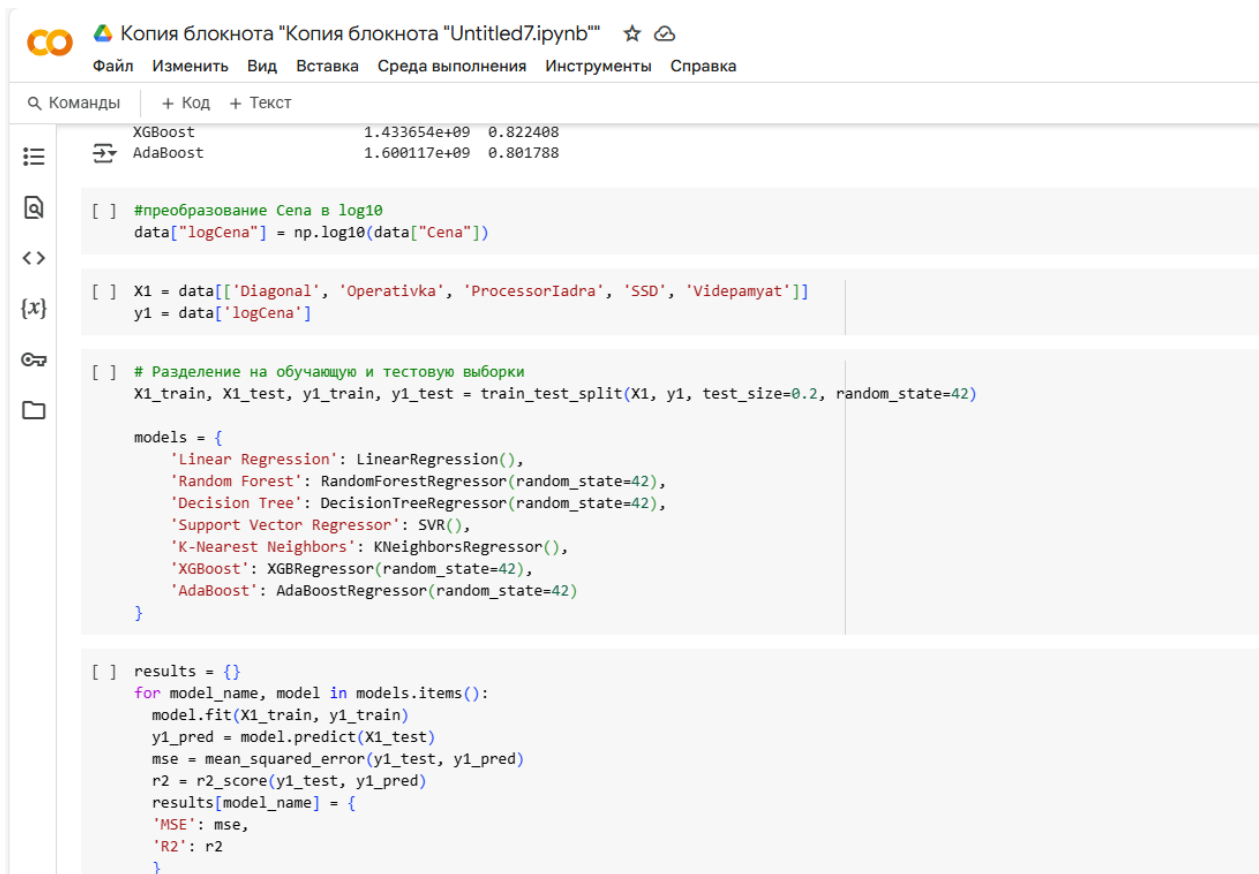
```

[31] #преобразование Cena в log10
data["logCena"] = np.log10(data["Cena"])

```

Рисунок 5- Обзор метрик MSE, R2

Из полученных данных следует, что модель Random Forest показала наиболее точный результат по метрике R2. Но для того, чтобы точно убедиться в надежности данной модели, необходимо провести еще одно исследование с выводом метрик, преобразовав перед этим категорию «Cena» с использованием $\log_{10}$. Провести разделение на обучающую и тестовую выборки (Рис.6).



```

XGBoost      1.433654e+09  0.822408
AdaBoost      1.600117e+09  0.801788

[ ] #преобразование Cena в log10
    data["logCena"] = np.log10(data["Cena"])

[ ] X1 = data[['Diagonal', 'Operativka', 'ProcessorIadra', 'SSD', 'Videpamyat']]
    y1 = data['logCena']

[ ] # Разделение на обучающую и тестовую выборки
    X1_train, X1_test, y1_train, y1_test = train_test_split(X1, y1, test_size=0.2, random_state=42)

models = {
    'Linear Regression': LinearRegression(),
    'Random Forest': RandomForestRegressor(random_state=42),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Support Vector Regressor': SVR(),
    'K-Nearest Neighbors': KNeighborsRegressor(),
    'XGBoost': XGBRegressor(random_state=42),
    'AdaBoost': AdaBoostRegressor(random_state=42)
}

[ ] results = {}
    for model_name, model in models.items():
        model.fit(X1_train, y1_train)
        y1_pred = model.predict(X1_test)
        mse = mean_squared_error(y1_test, y1_pred)
        r2 = r2_score(y1_test, y1_pred)
        results[model_name] = {
            'MSE': mse,
            'R2': r2
        }

```

Рисунок 6 – Преобразование в $\log_{10}$ Cena

Далее выполнить вывод метрик с новой переменной (рис.7).



```
[ ] 'Random Forest': RandomForestRegressor(random_state=42),
    'Decision Tree': DecisionTreeRegressor(random_state=42),
    'Support Vector Regressor': SVR(),
    'K-Nearest Neighbors': KNeighborsRegressor(),
    'XGBoost': XGBRegressor(random_state=42),
    'AdaBoost': AdaBoostRegressor(random_state=42)
}

[ ] results = {}
for model_name, model in models.items():
    model.fit(X1_train, y1_train)
    y1_pred = model.predict(X1_test)
    mse = mean_squared_error(y1_test, y1_pred)
    r2 = r2_score(y1_test, y1_pred)
    results[model_name] = {
        'MSE': mse,
        'R2': r2
    }
results_df = pd.DataFrame(results).T
print("Results:")
print(results_df)
```

Results:

	MSE	R2
Linear Regression	0.030312	0.605096
Random Forest	0.019317	0.748332
Decision Tree	0.021804	0.715933
Support Vector Regressor	0.032002	0.583077
K-Nearest Neighbors	0.024242	0.684181
XGBoost	0.019923	0.740438
AdaBoost	0.023461	0.694350

```
[ ] # Генерация случайных данных с выбросами
np.random.seed(0) # Для воспроизводимости
```

Рисунок 7 – Вывод метрик

На основании данных вывести итоговую таблицу (Таблица 1).

Таблица 1. Результаты моделирования

Метод	Степень разброса Получившихся моделей (%) Обычная цена	Степень разброса Получившихся моделей (%) Цена лог10
Linear Regression	74,2%	60,5%
Random Forest	82,6%	74,9%
Decision Tree	77,6%	71,6%
Support Vector Regressor	-13,6%	58,4%
K- Nearest Neighbors	77,1%	68,5%
XGBoost	82,3%	74,1%
AdaBoost	80,2%	69,5%

Следующим шагом будет построение модели Flask, для запуска системы на веб-приложения понадобится программа Open Server Panel, PyCharm, NotePad++ (или любой другой текстовый редактор, например блокнот). Сначала следует подготовить программу Open Server Panel для запуска веб-приложения, зайти в настройки (рис. 8).

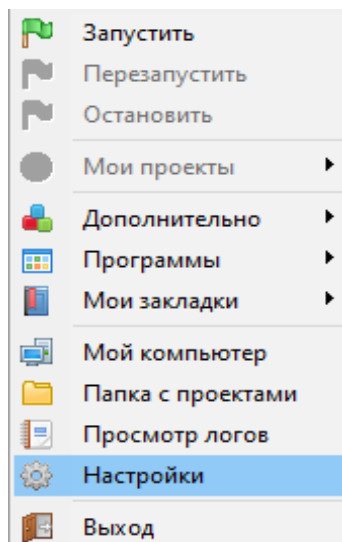


Рисунок 8 Вкладка настройки приложения Open Server Panel

Во вкладке «Модули» необходимо поставить базовые настройки для двух строк (рис. 9).

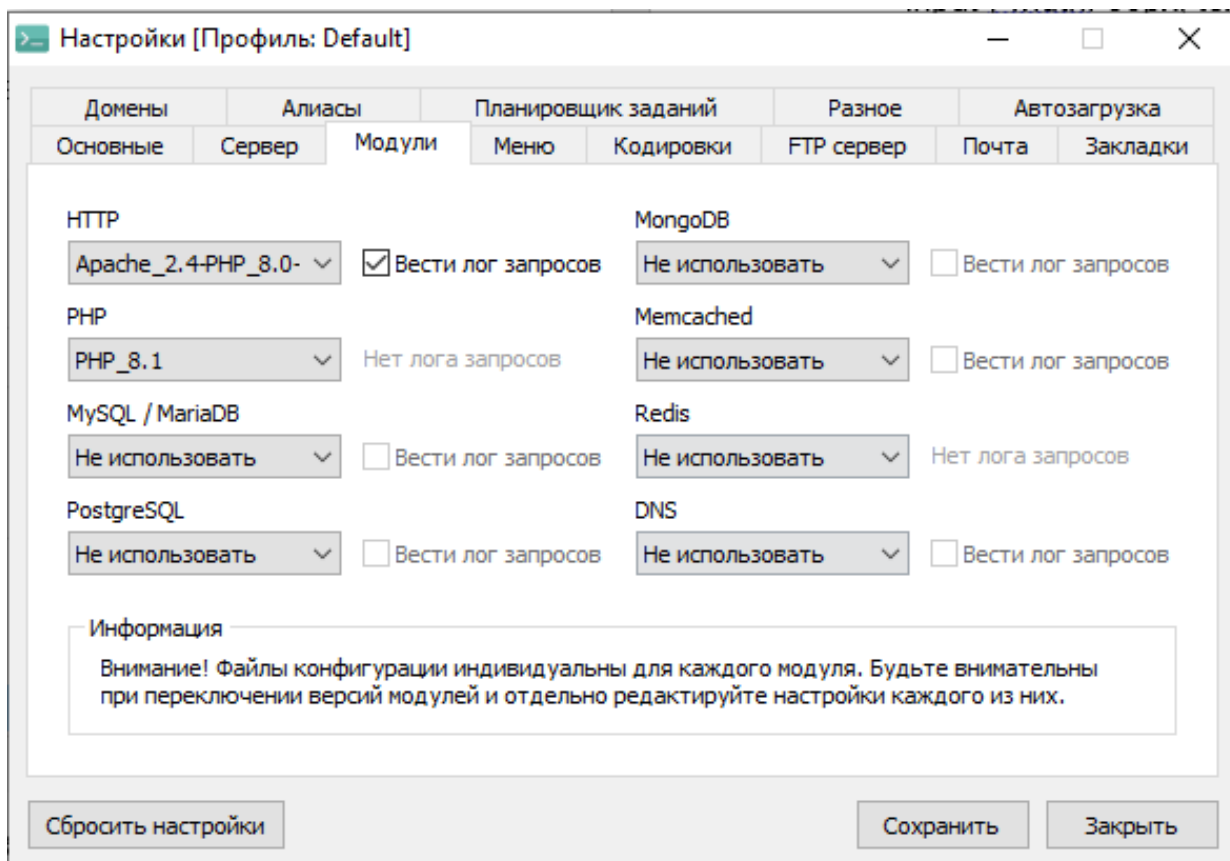


Рисунок 9 – Настройки для вкладки «Модули»

После настройки программы, необходимо перейти в расположение localhost, для этого необходимо нажать ПКМ по значку Open Server Panel на рабочем столе, после «Расположение файла» и перейти в папку localhost (рис. 10).

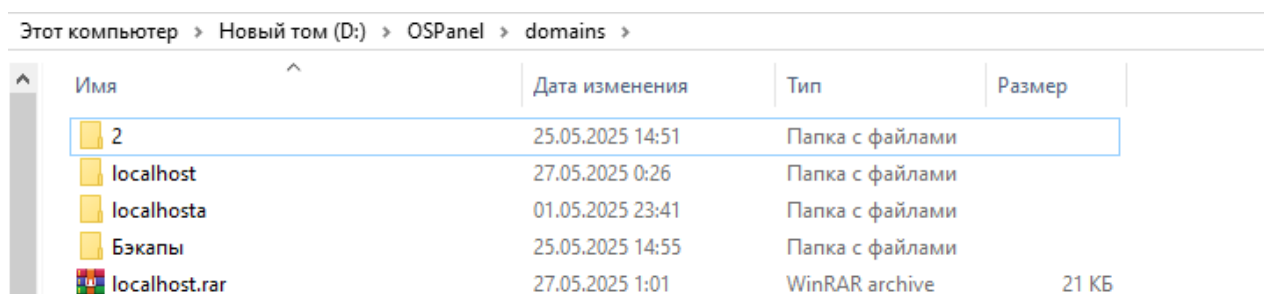


Рисунок 10 - Переход в директорию папки localhost

В данную папку необходимо выгрузить обученную модель, скаченную из Google Colab с расширением pb, после этого создать папку model и поместить в нее папки assets, variables, а также файл с моделью (рис. 11).

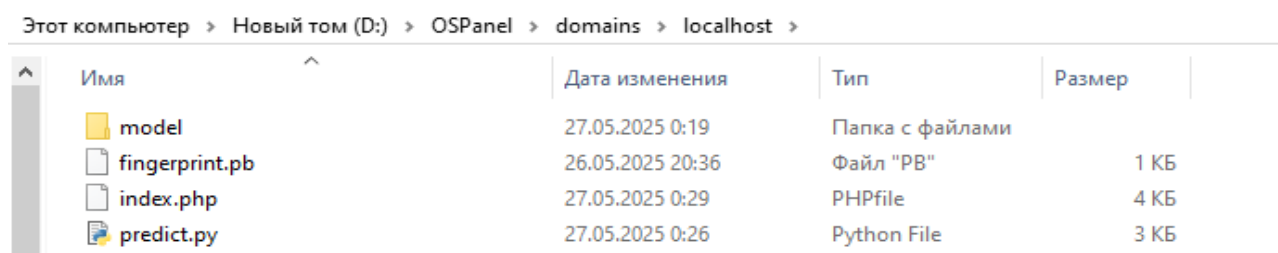


Рисунок 11- Структура папок

Далее нужно создать файл index.php для веб-страницы в корне папки localhost. Файл открыть через NotePad++ или Блокнот. Нужно написать код, который указывает, что использоваться будет HTML5-документ, с указанием языка страницы – русский, установки кодировки и заголовком страницы:

```
<!DOCTYPE html>
<html lang="ru">
<head>
<meta charset="UTF-8" />
<title>Ввод данных и предсказание</title>
```

Следующим шагом следует написать стили таблицы, в которые вносятся данные:

```
<style>
table { border-collapse: collapse; margin-bottom: 10px; }
td, th { border: 1px solid #ccc; padding: 8px; text-align: center; }
input { width: 60px; text-align: center; }
#resultNumber { font-weight: bold; font-size: 24px; margin-top: 10px; }
#resultPlot img { max-width: 100%; height: auto; margin-top: 10px; }
#interactivePlot { width: 600px; height: 400px; margin-top: 20px; }
</style>
```

Для того чтобы подключить построения графиков, необходимо подключить скрипт JavaScript.

```
<script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
```

Далее необходимо прописать заголовок страницы и вставить таблицу с 10 колонками, в которых необходимо помещать значения с id (начиная с 0 до 9) и добавление кнопки «Отправить».

```
<h2>Введите данные (10 значений):</h2>
```

```
<table>
```

```
<tr>
```

```
<?php for ($i=0; $i<10; $i++): ?>
```

```
<td><input type="number" step="any" id="col<?= $i ?>" /></td>
```

```
<?php endfor; ?>
```

```
</tr>
```

```
</table>
```

```
<button onclick="sendData()">Отправить</button>
```

Чтобы выводить значения, необходимо обозначить, что нужно выводить, для этого необходимо использовать 3 значения.

```
<div id="resultNumber"></div>
```

```
<div id="resultPlot"></div>
```

```
<div id="interactivePlot"></div>
```

Далее нужно написать функцию предназначенную для построения графика рассеивания, а также отправку POST-запросов на сервер Flask в формате JSON. После, в response.json(), для того чтобы преобразовывать ответ сервера в JSON, а также настройку формата вывода предсказания в числовой. Следующим этапом необходимо закодировать график в формате Base64. С помощью trace задать описание данных для графика, а layout создаст заголовки осей.

```
<script>
```

```
function sendData() {
```

```
  const inputs = [];
```

```
  for(let i=0; i<10; i++) {
```

```
    const val = document.getElementById('col'+i).value;
```

```
    if(val === "") {
```

```
      alert('Пожалуйста, заполните все 10 полей.');
```

```
      return;
```

```
    }
```

```
    inputs.push(parseFloat(val));
```

```
  }
```

```
  fetch('http://127.0.0.1:5000/predict', {
```

```
    method: 'POST',
```

```
    headers: {
```

```
      'Content-Type': 'application/json'
```

```
    },
```

```
    body: JSON.stringify({ input: inputs })
```

```
    })
    .then(response => response.json())
    .then(data => {
      if(data.error) {
        alert('Ошибка: ' + data.error);
        return;
      }
      if(data.prediction !== null) {
        document.getElementById('resultNumber').innerText = 'Предсказание: '
+ data.prediction.toFixed(4);
      } else {
        document.getElementById('resultNumber').innerText = "";
      }
      if(data.plot) {
        document.getElementById('resultPlot').innerHTML = '';
      } else {
        document.getElementById('resultPlot').innerHTML = "";
      }
      buildScatterPlot(inputs);
    })
    .catch(err => {
      alert('Ошибка при запросе: ' + err.message);
    });
  }
  function buildScatterPlot(data) {
    const trace = {
      x: Array.from(data.keys()),
      y: data,
      mode: 'markers+lines',
      type: 'scatter',
      marker: { color: 'blue', size: 8 },
      line: { shape: 'spline' },
      name: 'Введенные данные'
    };

    const layout = {
      title: 'Диаграмма рассеивания введенных данных',
      xaxis: { title: 'Индекс' },
      yaxis: { title: 'Значение' },
      margin: { t: 40, b: 40 }
    };

    Plotly.newPlot('interactivePlot', [trace], layout, {responsive: true});
  }
```

</script>

На данном этапе заканчивается написание клиентской части (рис.12), (рис. 13).

```

1 <DOCTYPE html>
2 <html lang="ru">
3 <head>
4 <meta charset="UTF-8" />
5 <title>Ввод данных и предсказание</title>
6 <style>
7 table { border-collapse: collapse; margin-bottom: 10px; }
8 td, th { border: 1px solid #ccc; padding: 5px; text-align: center; }
9 input { width: 60px; text-align: center; }
10 #resultNumber { font-weight: bold; font-size: 24px; margin-top: 10px; }
11 #resultPlot { max-width: 100%; height: auto; margin-top: 10px; }
12 #interactivePlot { width: 600px; height: 400px; margin-top: 20px; }
13 </style>
14 <!-- Подключаем Plotly.js -->
15 <script src="https://cdn.plot.ly/plotly-latest.min.js"></script>
16 </head>
17 <body>
18 <div>Введите данные (10 значений):</div>
19 <table>
20 <tr>
21 <td><input type="text" value="" /></td></tr>
22 <td><input type="text" value="" /></td></tr>
23 <td><input type="text" value="" /></td></tr>
24 <td><input type="text" value="" /></td></tr>
25 <td><input type="text" value="" /></td></tr>
26 <td><input type="text" value="" /></td></tr>
27 <td><input type="text" value="" /></td></tr>
28 <td><input type="text" value="" /></td></tr>
29 <td><input type="text" value="" /></td></tr>
30 <td><input type="text" value="" /></td></tr>
31 </table>
32 <div id="resultNumber"></div>
33 <div id="resultPlot"></div>
34 <div id="interactivePlot"></div>
35 <script>
36 function sendData() {
37   const inputs = [];
38   for(let i=0; i<10; i++) {
39     const val = document.getElementById('col'+i).value;
40     if(val === '') {
41       alert('Пожалуйста, заполните все 10 полей.');

```

Рисунок 12 – Код для файла index.php

```

54   .then(response => response.json())
55   .then(data => {
56     if(data.error) {
57       alert('Ошибка: ' + data.error);
58       return;
59     }
60     if(data.prediction !== null) {
61       document.getElementById('resultNumber').innerText = 'Предсказание: ' + data.prediction.toFixed(4);
62     } else {
63       document.getElementById('resultNumber').innerText = '';
64     }
65     if(data.plot) {
66       document.getElementById('resultPlot').innerHTML = '';
67     } else {
68       document.getElementById('resultPlot').innerHTML = '';
69     }
70     buildScatterPlot(inputs);
71   })
72   .catch(err => {
73     alert('Ошибка при запросе: ' + err.message);
74   });
75 }
76
77 function buildScatterPlot(data) {
78   const trace = {
79     x: Array.from(data.keys()),
80     y: data,
81     mode: 'markers+lines',
82     type: 'scatter',
83     marker: { color: 'blue', size: 8 },
84     line: { shape: 'spline' },
85     name: 'Введенные данные'
86   };
87
88   const layout = {
89     title: 'Диаграмма рассеивания введенных данных',
90     xaxis: { title: 'Индекс' },
91     yaxis: { title: 'Значение' },
92     margin: { t: 40, b: 40 }
93   };
94
95   Plotly.newPlot('interactivePlot', [trace], layout, {responsive: true});
96 }
97 </script>
98 </body>
99 </html>
100

```

Рисунок 13 - Продолжение кода для index.php

Следующим этапом будет написание файла, который будет отвечать за серверную часть. Для начала установить PyCharm, установить интерпретатор версии 3.10.9. Для этого нужно перейти во вкладку File – Settings в открывшемся окне перейти во вкладку Python Interpreter, после чего нажать на «Add Interpreter» и выбрать 3.10 версию (рис. 14).

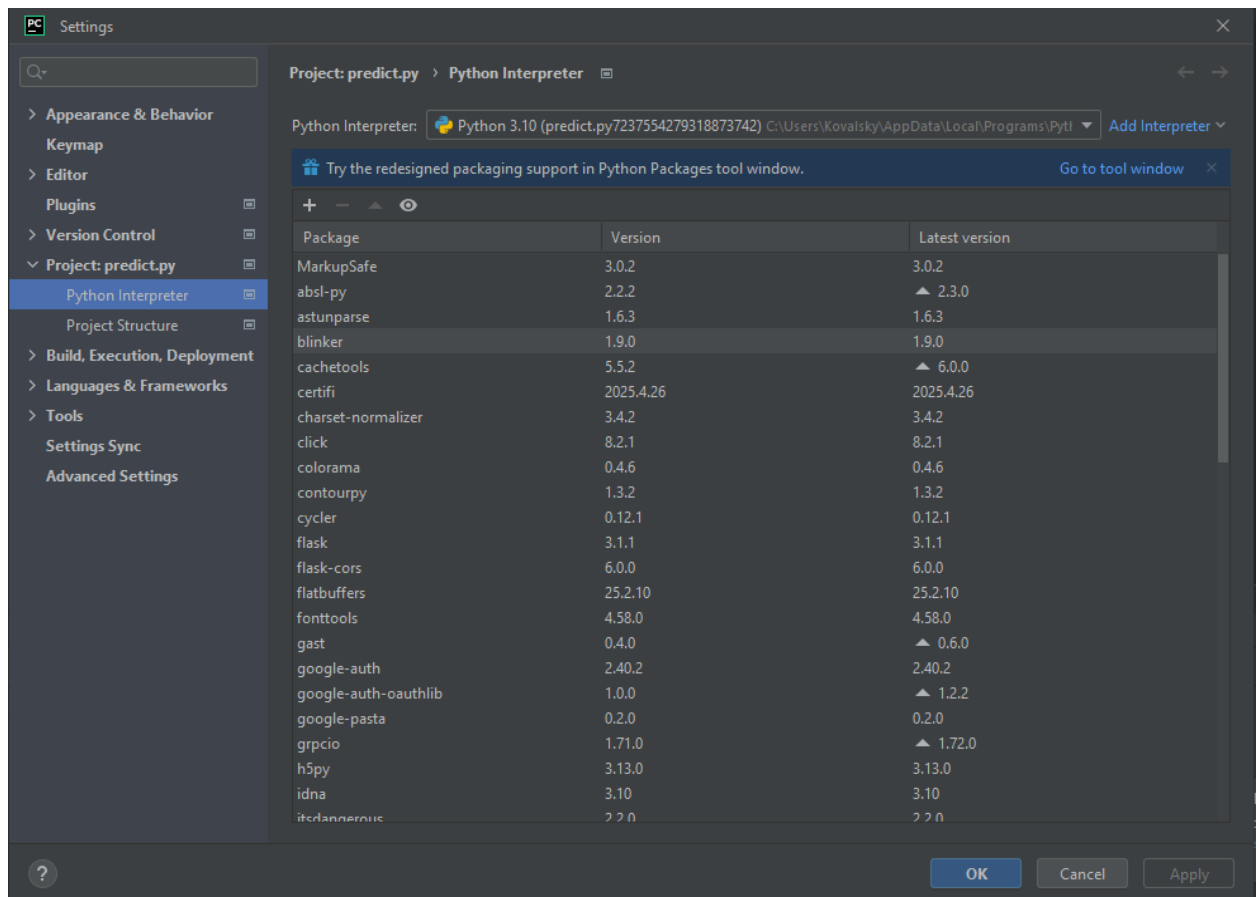


Рисунок 14- Выбор интерпретатора в PyCharm

Сначала нужно создать файл `predict.py` и открыть его в IDE, далее, добавив импорт библиотек в проект, которые будут использоваться:

```
from flask import Flask, request, jsonify
from flask_cors import CORS
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import io
import base64
import os
```

Примечание: библиотеку `keras` и `tensorflow` нужно устанавливать версии 2.12.0, так как более ранние версии уже не поддерживают работу с данным типом файла, более поздние могут не поддерживать и вовсе работу с данным файлом модели.

Далее необходимо написать код для импортирования файла и считывание модели в нем:

```
app = Flask(__name__)
CORS(app)
model_path = os.path.join(os.path.dirname(__file__), "model")
loaded = tf.saved_model.load(model_path)
infer = loaded.signatures["serving_default"]
```

Следующим шагом следует прослушать отклик серверной части при вводе чисел в модель, поскольку каждая модель по-разному рассчитана на количество входных значений, но в большинстве случаев это 10 значений, поэтому нужно прописать это в коде:

```
input_key = list(infer.structured_input_signature[1].keys())[0]
expected_input_shape =
list(infer.structured_input_signature[1].values())[0].shape
expected_feature_count = expected_input_shape[1]
print("Ожидаемый вход:")
print(infer.structured_input_signature)
```

Следующая часть кода должна получать JSON данные запроса, после преобразовать входные данные в тип переменной float32 и преобразовать массив данных в двумерный массив:

```
@app.route('/predict', methods=['POST'])
def predict():
    try:
        data = request.get_json(force=True)
        input_data = np.array(data['input'], dtype=np.float32).reshape(1, -1)
```

Для того, чтобы проверить количество входных значений, необходимо создать ошибку, которая будет появляться на сервере, если количество входных данных больше или меньше 10:

```
if input_data.shape[1] != expected_feature_count:
    return jsonify({
        'error': f'Ожидается {expected_feature_count} входных значений, а
получено {input_data.shape[1]}'
    }), 400
```

Для работы с моделью необходимо преобразовать значения в TensorFlow, так как выполнение предсказания осуществляется с его помощью. После данной операции извлекается результат из модели и преобразовывается обратно в массив NumPy:

```
input_tensor = tf.convert_to_tensor(input_data)
result = infer(**{input_key: input_tensor})
output_tensor = list(result.values())[0].numpy()
```

Следующим этапом необходимо сделать обработку результатов если модель возвращает скалярное значение (например, [1, 1]), оно извлекается и преобразуется в число с плавающей точкой:

```
scalar_value = float(output_tensor[0][0]) if output_tensor.shape == (1, 1) else
None
```

Чтобы построить график, необходимо его сохранять в PNG формате, чтобы закодировать изображение формате Base64 для передачи через HTTP:

```
plot_image_base64 = None
if output_tensor.shape[1] > 1:
    plt.figure()
    plt.plot(output_tensor[0])
```

```
plt.title("График предсказания")
plt.grid(True)
```

```
buf = io.BytesIO()
plt.savefig(buf, format='png')
buf.seek(0)
plot_image_base64 = base64.b64encode(buf.read()).decode('utf-8')
buf.close()
```

Чтобы сформировать ответ, его необходимо сформировать в JSON ответ с предсказанным числом или графиком:

```
return jsonify({
    'prediction': scalar_value,
    'plot': plot_image_base64
})
```

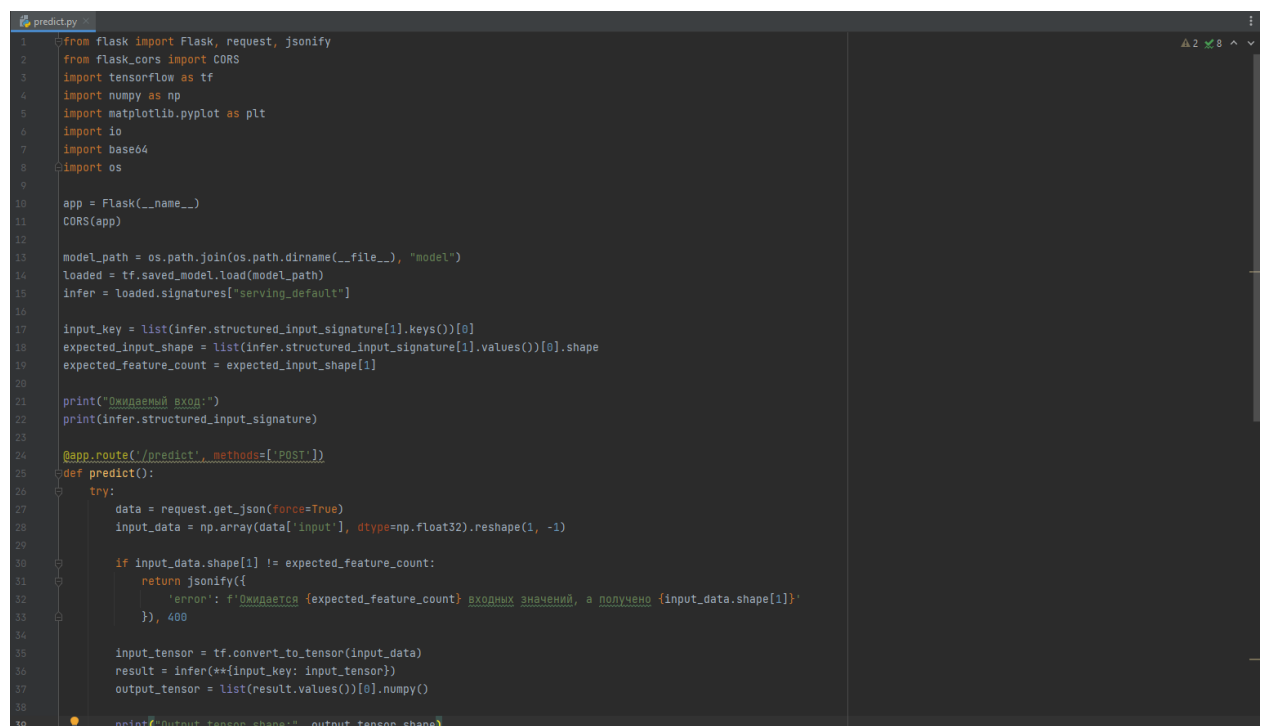
Необходимо создать отладку обработчика ошибок при формировании JSON-ответа:

```
except Exception as e:
    return jsonify({'error': str(e)}), 500
```

Чтобы запустить сервер Flask на локальном хосте, необходимо прописать ему хост и порт:

```
if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000)
```

Код для серверного файла predict.py готов (рис 15) (рис. 16).



```
1 from flask import Flask, request, jsonify
2 from flask_cors import CORS
3 import tensorflow as tf
4 import numpy as np
5 import matplotlib.pyplot as plt
6 import io
7 import base64
8 import os
9
10 app = Flask(__name__)
11 CORS(app)
12
13 model_path = os.path.join(os.path.dirname(__file__), "model")
14 loaded = tf.saved_model.load(model_path)
15 infer = loaded.signatures["serving_default"]
16
17 input_key = list(infer.structured_input_signature[1].keys())[0]
18 expected_input_shape = list(infer.structured_input_signature[1].values())[0].shape
19 expected_feature_count = expected_input_shape[1]
20
21 print("Ожидаемый вход:")
22 print(infer.structured_input_signature)
23
24 @app.route('/predict', methods=['POST'])
25 def predict():
26     try:
27         data = request.get_json(force=True)
28         input_data = np.array(data['input'], dtype=np.float32).reshape(1, -1)
29
30         if input_data.shape[1] != expected_feature_count:
31             return jsonify({
32                 'error': f'Ожидается {expected_feature_count} входных значений, а получено {input_data.shape[1]}'
33             }), 400
34
35         input_tensor = tf.convert_to_tensor(input_data)
36         result = infer(**{input_key: input_tensor})
37         output_tensor = list(result.values())[0].numpy()
38
39         print("Output tensor shape:", output_tensor.shape)
```

Рисунок 15 – Структура кода файла predict.py

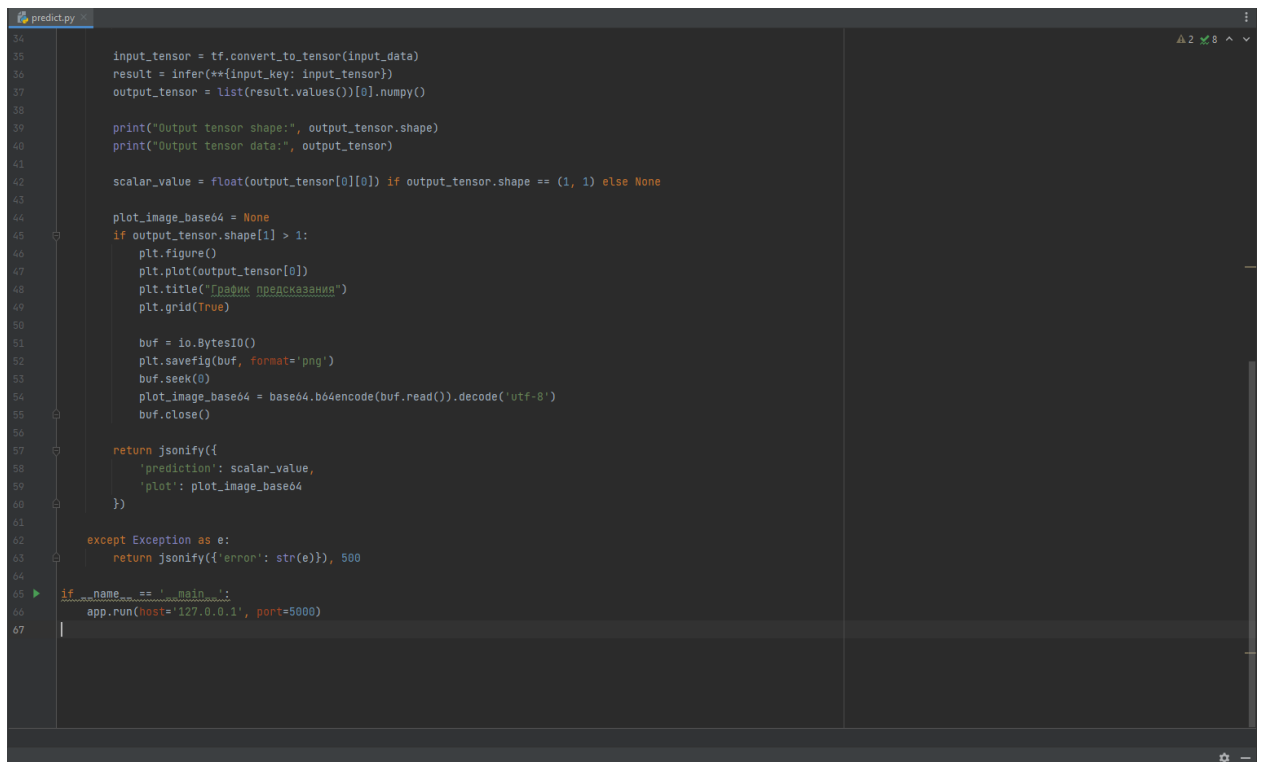


Рисунок 16 – Продолжение структуры кода файла predict.py

Для запуска приложения необходимо запустить Open Server Panel, чтобы в тее загорелся зеленый флажок. После необходимо зайти в командную строку и перейти в директорию проекта:

cd полный_путь_к_директории_localhost

Далее необходимо запустить серверный файл:

python predict.py

После некоторого времени появляется окно, что открыт доступ по хосту 127.0.0.1 и порту 5000 (рис. 17).

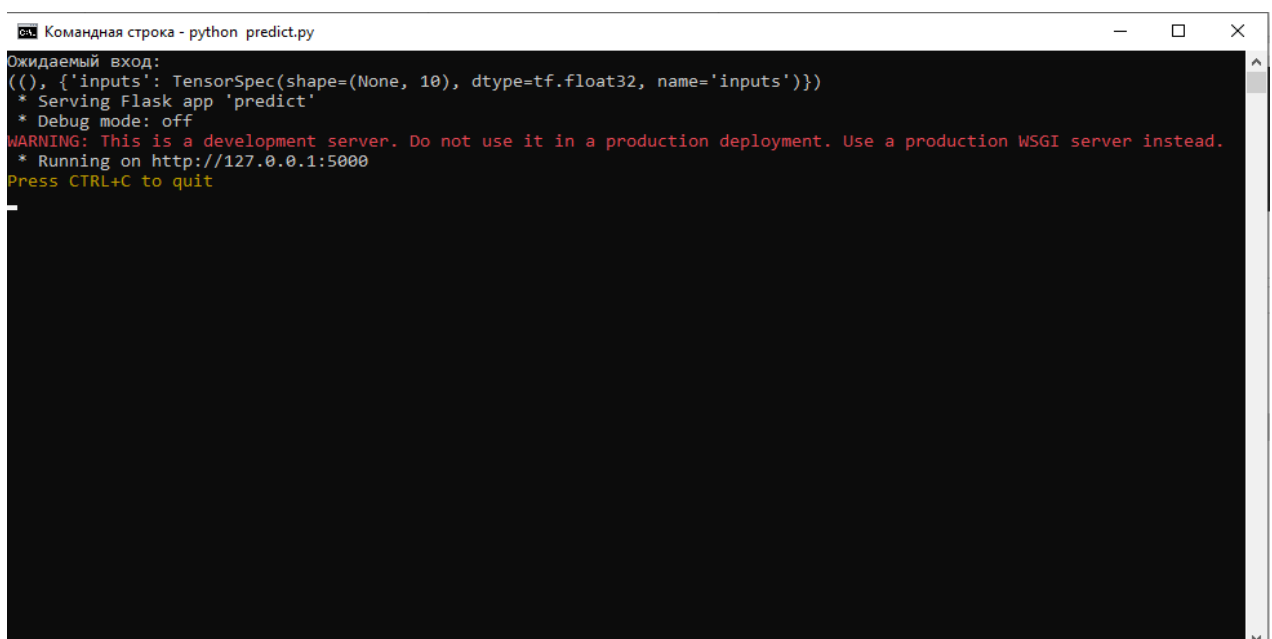


Рисунок 17 – Запуск файла predict.py через cmd

Если обратить внимание, то сначала появилось сообщение «Ожидаемый вход», после которого показано количество входных значений = (None, 10). После запуска сервера необходимо перейти в любой браузер и в url указать «localhost», после появится страница с вводом значений (рис. 18).

Введите данные (10 значений):

--	--	--	--	--	--	--	--	--	--

Рисунок 18 – Главная страница localhost

Предсказание: 1.3653- это вывод модели, которая преобразовала введённые данные в одно значение предсказания. Далее идёт график: диаграмма рассеивания введённых данных, с осями индекс «х» и значение «у». График показывает ряд точек, где жизнь линии образуют характерную волну между индексами 0 и 9 (рис. 19).

Введите данные (10 значений):

1	2	1	3	4	1	3	2.5	1.5	1.1
---	---	---	---	---	---	---	-----	-----	-----

Предсказание: 1.3653

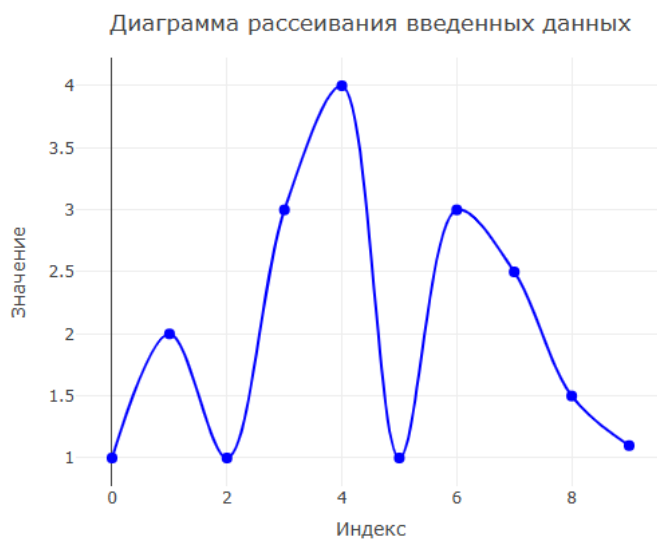
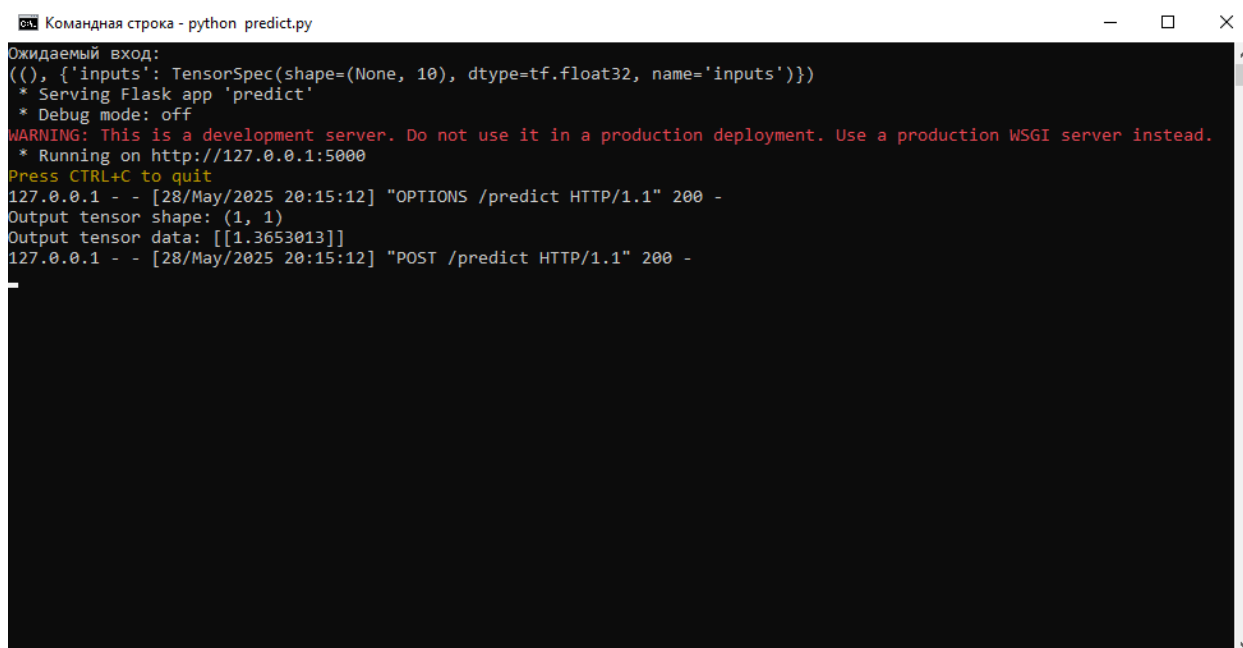


Рисунок 19 – Результат предсказания модели

Если перейти в cmd после отправки запроса, то можно увидеть, что оба запроса были получены и имеют код 200, а также значения модели (рис. 20).



```
Командная строка - python predict.py
Ожидаемый вход:
(((), {'inputs': TensorSpec(shape=(None, 10), dtype=tf.float32, name='inputs'))})
* Serving Flask app 'predict'
* Debug mode: off
WARNING: This is a development server. Do not use it in a production deployment. Use a production WSGI server instead.
* Running on http://127.0.0.1:5000
Press CTRL+C to quit
127.0.0.1 - - [28/May/2025 20:15:12] "OPTIONS /predict HTTP/1.1" 200 -
Output tensor shape: (1, 1)
Output tensor data: [[1.3653013]]
127.0.0.1 - - [28/May/2025 20:15:12] "POST /predict HTTP/1.1" 200 -
```

Рисунок 20 – Мониторинг получения запросов через cmd

4 Выводы

В результате проведенного анализа было разработано функциональное веб-приложение на базе Flask, позволяющее прогнозировать цены на ноутбуки с использованием различных моделей машинного обучения, таких как Linear Regression, XGBoost, Random Forest. Приложение успешно интегрировало предобученные модели. На данных модели Random Forest, предоставило пользователям интуитивно понятный интерфейс с визуализацией результатов. Разработанное приложение представляет собой полезный инструмент для анализа и прогнозирования цен, который может быть полезен аналитикам. Такое сочетание функциональности и удобного интерфейса предоставляет пользователям возможность более детально анализировать и понимать работу модели, что является важным шагом в оптимизации процессов принятия решений на основе данных.

Библиографический список

1. Оразгулыев А., Гараджаева С. А. Визуализация данных. Представление результатов исследования // Наука и мировоззрение. 2025. №40. С. 42-48. URL: <https://cyberleninka.ru/article/n/vizualizatsiya-dannyh-predstavlenie-rezultatov-issledovaniya>
2. Ремнев А. С., Муртазина А. Р. Использование языка PYTHON для анализа и визуализации данных // Редакционная коллегия. 2025. С. 125. URL: <https://rguk.ru/upload/medialibrary/d50/1d37nnw13nusen4je2iyrrlpqlbkgcm7d/ИНТЕКС-2025%20Часть%205.pdf#page=125>
3. Волокитина Т.С. Анализ возможностей Google Colab // Современные научные исследования и инновации. 2020. №12. С. 1. URL: <https://www.elibrary.ru/item.asp?id=44477003>

4. Клячкин В.Н., Ковальногов В.Н., Шеркунов В.В., Хуссейн Мохамед Сравнительный анализ методов построения математических моделей функционирования объекта с применением машинного обучения // Программные продукты и системы. 2023. №2. С. 189-195. URL: <https://www.elibrary.ru/item.asp?id=54262529>
5. Дагаев А.В. Методы анализа данных в машинном обучении // Экономический потенциал и корпоративная социальная ответственность. 2024. С. 31-39. URL: <https://www.elibrary.ru/item.asp?id=68621474>
6. Полячкова М.А., Ефремова А.Н. Flask как фреймворк для изучения основ веб программирования // Труды братского государственного университета. Серия: естественные и инженерные науки. 2021. Т. 1. С. 52-55. URL: <https://elibrary.ru/item.asp?id=46596870>