Министерство науки и высшего образования РФ

федеральное государственное автономное  
образовательное учреждение высшего образования

«Омский государственный технический университет»

Факультет информационных технологий и компьютерных систем

Кафедра «Прикладная математика и фундаментальная информатика»

|  |
| --- |
| **Расчетно-графическая работа** |
| по дисциплине **Машинное обучение и большие данные**  **Тема: Разработка Web-приложения (дашборда)**  **для инференса моделей ML и анализа данных** |

|  |  |  |  |
| --- | --- | --- | --- |
| Студента | Чепурко Артёма Ивановича | | |
| Курс | 2 | Группа | ФИТ-222 |
| Направление | 02.03.02 Фундаментальная информатика и информационные технологии | | |
| Руководитель | доцент | | |
|  | Моисеева Наталья Александровна | | |
| Выполнил | 09.01.2024 | | |
| Проверил | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  дата, подпись руководителя | | |

Омск 2023

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 3](#_Toc116905603)-4

[1 Разработка решения 5-](#_Toc116905604)8

1.1 Описание выбранных моделей …………………………………………... 5-7

1.2 Процесс сериализации ……………………………………………………… 8

[2 Процесс разработки дашборда 7-11](#_Toc116905605)

[2.1 Обзор разработки](#_Toc116905606) 9

[2.2 Фрагменты исходного кода 9-](#_Toc116905607)11

2.3 Визуал работы ………………………………………………………… 11-16

2.4 Ссылка на GitHub ………………………………………………………….. 17

[ЗАКЛЮЧЕНИЕ 18-](#_Toc116905614)19

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ](#_Toc116905615) 20

[ПРИЛОЖЕНИЕ А 21-](#_Toc116905616)25

ПРИЛОЖЕНИЕ Б …………………………………………………………... 26-32

**ВВЕДЕНИЕ**

В современном мире, где информация становится основным ресурсом, а её обработка и толкование служат двигателем для новаторства и развития, значимость темы курсовой работы по созданию веб-приложения очевидна. Важность сочетания веб-технологий с методами машинного обучения заключается в возможности обрабатывать обширные массивы данных и делать этот процесс понятным для многих пользователей. Эта работа фокусируется на применении передовых технологий для создания веб-приложения (панели управления), которое будет использоваться для вывода результатов моделей машинного обучения и анализа данных. Основные технологии включают:

1. *Streamlit* – это библиотека на *Python*, позволяющая легко создавать и развертывать интерактивные веб-приложения для анализа данных и машинного обучения. Она предоставляет простой и интуитивно понятный способ визуализации данных и взаимодействия с ними, не требуя глубоких знаний в веб-разработке.
2. *Sklearn* (*Scikit-learn*) – это мощная и широко используемая библиотека машинного обучения для *Python*, предоставляющая широкий спектр алгоритмов для классификации, регрессии, кластеризации и уменьшения размерности данных. Она отличается простотой использования, хорошей документацией и поддержкой различных методов предобработки данных и моделей машинного обучения.
3. *TensorFlow* – мощная система для создания и тренировки нейронных сетей, которая позволяет эффективно решать задачи связанные с большими данными и искусственным интеллектом. Эти инструменты играют ключевую роль в решении задач машинного обучения, обеспечивая необходимую гибкость, мощь и удобство в работе с данными и моделями. Разработка веб-приложения в данной РГР демонстрирует практическое применение этих технологий.
4. *Matplotlib* - это библиотека для создания статических, анимированных и интерактивных визуализаций в *Python*. Она позволяет легко рисовать графики, гистограммы, диаграммы рассеяния и множество других типов графиков, обеспечивая большую гибкость и контроль над внешним видом визуализаций. *Matplotlib* часто используется в научных и аналитических приложениях для представления данных в наглядной форме.
5. *Pandas* - это библиотека для обработки и анализа данных в *Python*, предоставляющая специальные структуры данных и функции для манипулирования табличными данными. Она особенно удобна для работы с временными рядами, статистическими данными, и любыми другими данными, которые можно представить в форме таблиц. *Pandas* облегчает такие задачи, как фильтрация данных, агрегация, преобразование, а также их визуализация с помощью интеграции с *Matplotlib*.

**РАЗРАБОТКА РЕШЕНИЯ**

* 1. **Описание выбранных моделей**

В данной работе использовались следующие модели машинного обучения:

- ***KNeighborsClassifier (KNN, метод k-ближайших соседей)***: это алгоритм машинного обучения для классификации и регрессии, который определяет класс или значение объекта на основе k наиболее похожих на него объектов в обучающем наборе данных. Прост в реализации, но менее эффективен при большом объеме данных или высокой размерности признаков.

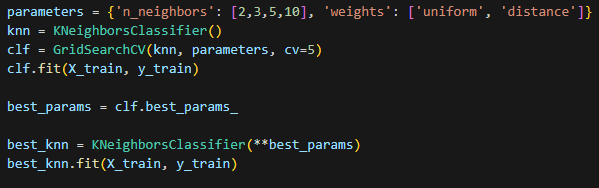


Рисунок 1 – код алгоритма *KNN* с подбором оптимальных гиперпараметров*.*

**- *KMeans* кластеризация:** группирует данные на основе их сходства, разделяя их на заданное количество кластеров *k* (в данном случае 5). Каждый кластер характеризуется своим центроидом. Алгоритм итеративно минимизирует сумму расстояний от точек данных до центроидов их кластеров, эффективно разделяя данные на гомогенные группы.



Рисунок 2 – код алгоритма *KMeans*.

- ***CatBoost****:* это высокопроизводительная библиотека для градиентного бустинга на решающих деревьях. Она оптимизирована для работы с категориальными данными и может автоматически обрабатывать их без предварительного преобразования.

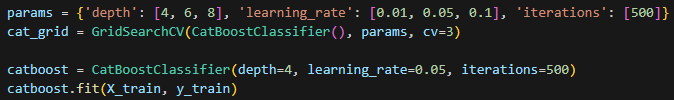


Рисунок 3 – код алгоритма *CatBoost* с подбором оптимальных гиперпараметров*.*

- ***StackingClassifier***: это метод ансамблевого машинного обучения, который объединяет предсказания от нескольких моделей обучения (в данном случае: *LogisticRegression*, *KNeighborsClassifier*, *DecisionTreeClassifier*) для повышения точности классификации. В *StackingClassifier*, базовые модели обучаются на полном обучающем наборе данных, а затем их предсказания используются как входные данные для мета-классификатора, который обучается на этих предсказаниях для выработки окончательного вердикта. Этот подход позволяет улучшить производительность модели за счет объединения сильных сторон различных алгоритмов.

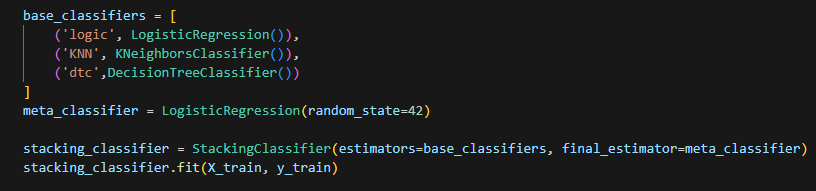


Рисунок 4 – код алгоритма *StackingClassifier*.

- ***RandomForestClassifier***: алгоритм машинного обучения, основанный на ансамбле решающих деревьев. Он обучается на разных подмножествах данных и использует усреднение для улучшения точности предсказаний и контроля переобучения. Эффективен для широкого спектра задач классификации.



Рисунок 5 – код алгоритма *RandomForestClassifier* с подбором оптимальных гиперпараметров.

- **Нейронная Сеть**: это модель глубокого обучения, имитирующая работу человеческого мозга для решения задач. Она состоит из слоёв искусственных нейронов, которые обучаются на больших объёмах данных, выявляя сложные закономерности и зависимости.

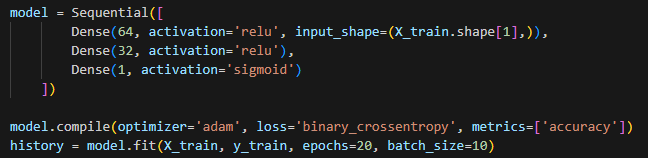


Рисунок 6 – нейронная сеть.

**1.2 Процесс сериализации.**

Сохранение и последующее применение моделей в производственной среде требует их сериализации. В данной работе используются два основных метода для этой цели:

- ***Pickle***: этот метод представляет собой простой и эффективный способ сериализации моделей на *Python*. С его помощью обученные модели могут быть сохранены в файл, что упрощает их последующую загрузку и использование.

- **H5 Формат (для нейронных сетей):** для моделей, созданных с использованием *Keras*, применяется формат h5. Он позволяет не только сохранить веса и архитектуру нейронной сети, но и конфигурацию её обучения, что является ключевым для воспроизведения результатов модели.

После сериализации, эти модели могут быть интегрированы в веб-приложения, например, разработанные на *Streamlit*. Это позволяет демонстрировать их работу в реальном времени, что является важным аспектом для практического применения моделей машинного обучения.

**ПРОЦЕСС РАЗРАБОТКИ ДАШБОРДА**

**2.1 Обзор разработки.**

Используя *Streamlit*, был разработан дашборд для анализа данных и визуализации работы моделей машинного обучения, обеспечивая интерактивный пользовательский интерфейс. Приоритетами в разработке были удобство в использовании, эффективная визуализация данных и возможность показать работу моделей в живом режиме.

**2.2 Фрагменты исходного кода.**

Импорт необходимых библиотек:

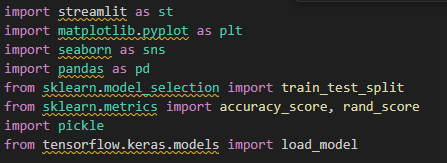


Рисунок 7 – импорт необходимых библиотек.

Подготовка данных:

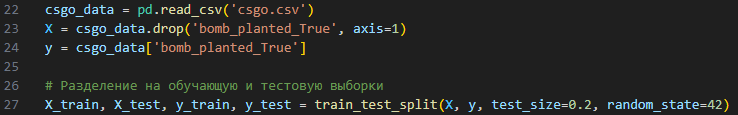


Рисунок 8 – подготовка данных

Загрузка моделей: Функция *deserialisation* загружает сериализованные модели машинного обучения для их последующего использования в дашборде.

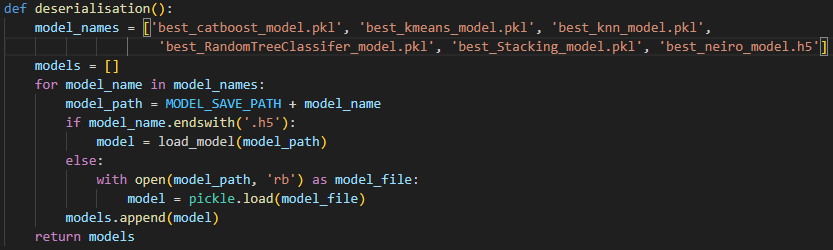


Рисунок 9 – загрузка моделей.

Конфигурация интерфейса *Streamlit*: настройка стилей и элементов навигации для дашборда.

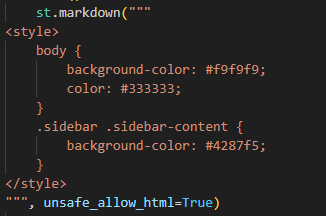


Рисунок 10 – конфигурация интерфейса.

Функции страниц дашборда: в коде имеются 4 функции, в которых содержатся страницы дашборда *Streamlit*.

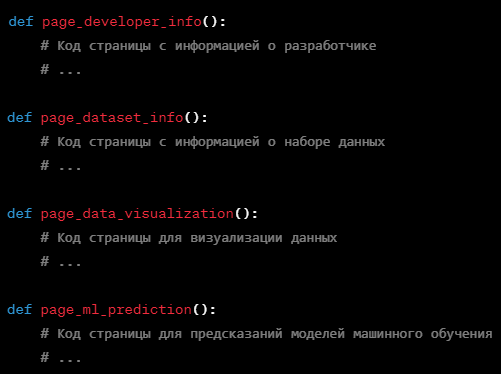


Рисунок 11 – функции страниц дашборда.

Выбор страницы в сайдбаре: пользователь может выбирать разные страницы для навигации в дашборде.

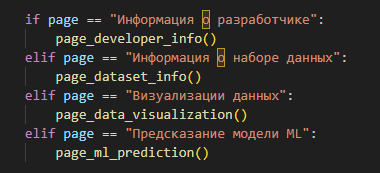


Рисунок 12 – выбор страницы.

**2.3 Визуал работы.**

Первая страница. Информация о разработчике.

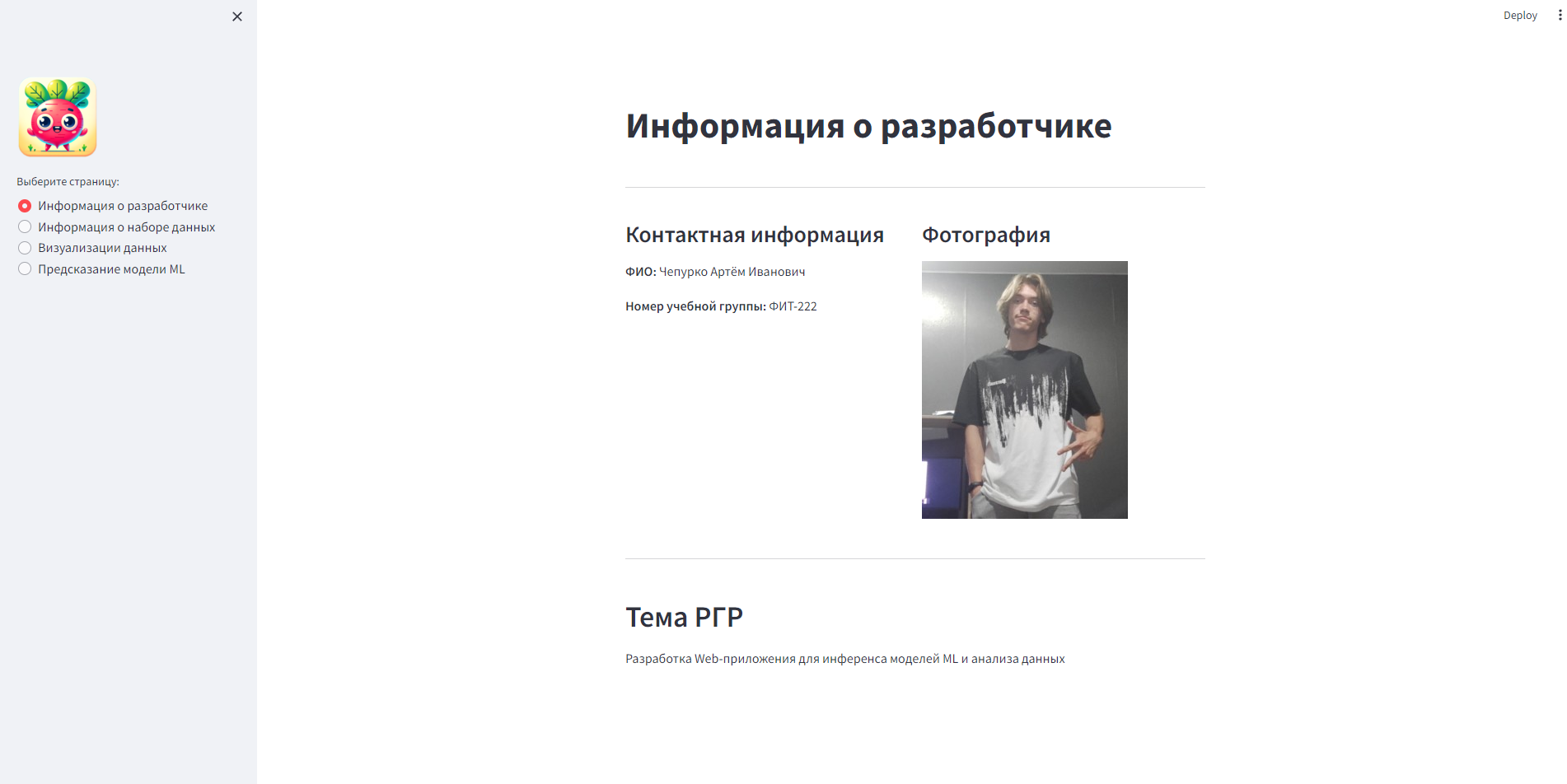


Рисунок 13 – информация о разработчике.

Вторая страница. Информация о наборе данных.

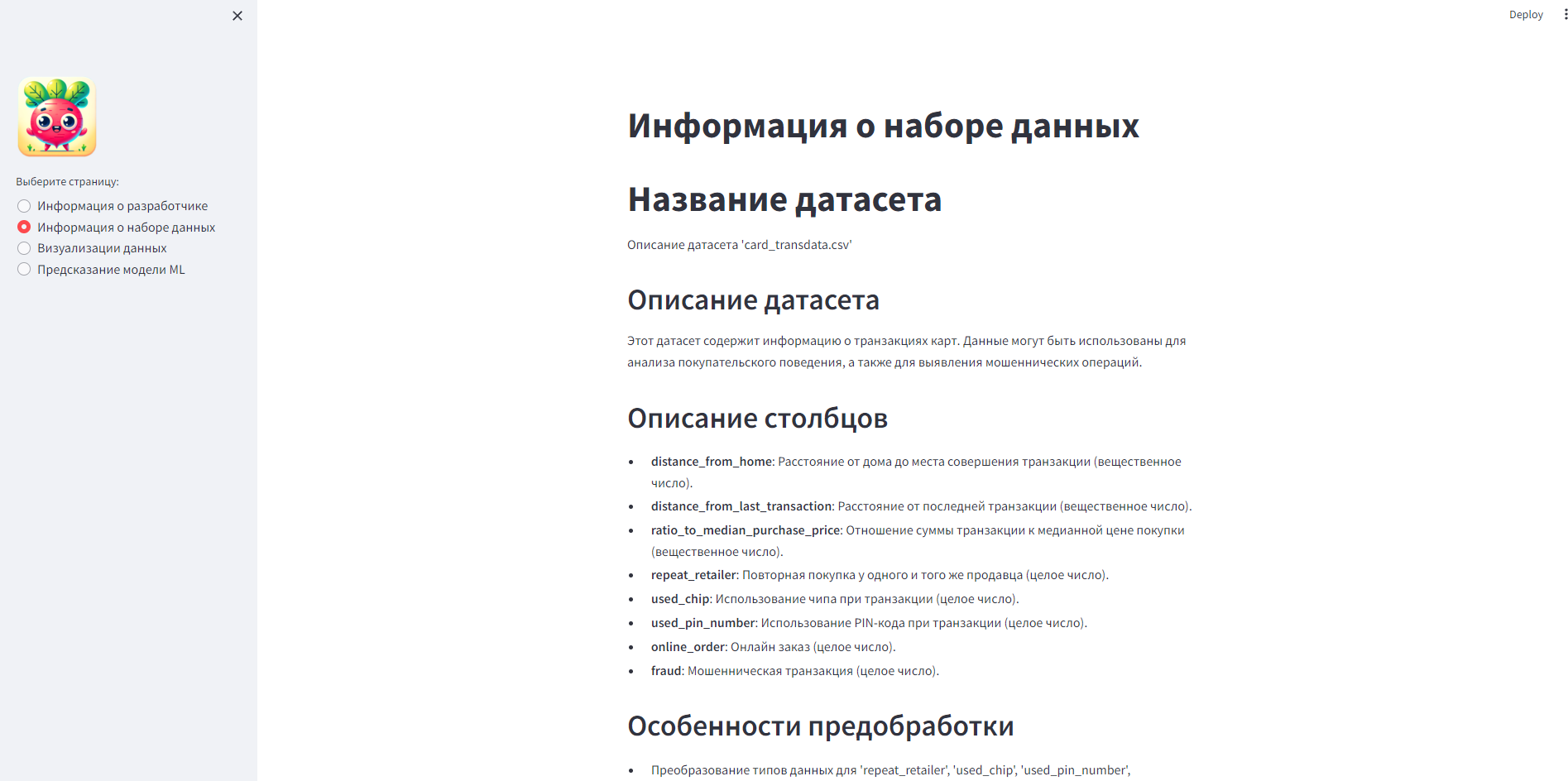


Рисунок 14 – информация о наборе данных.

Третья страница. Визуализации данных.

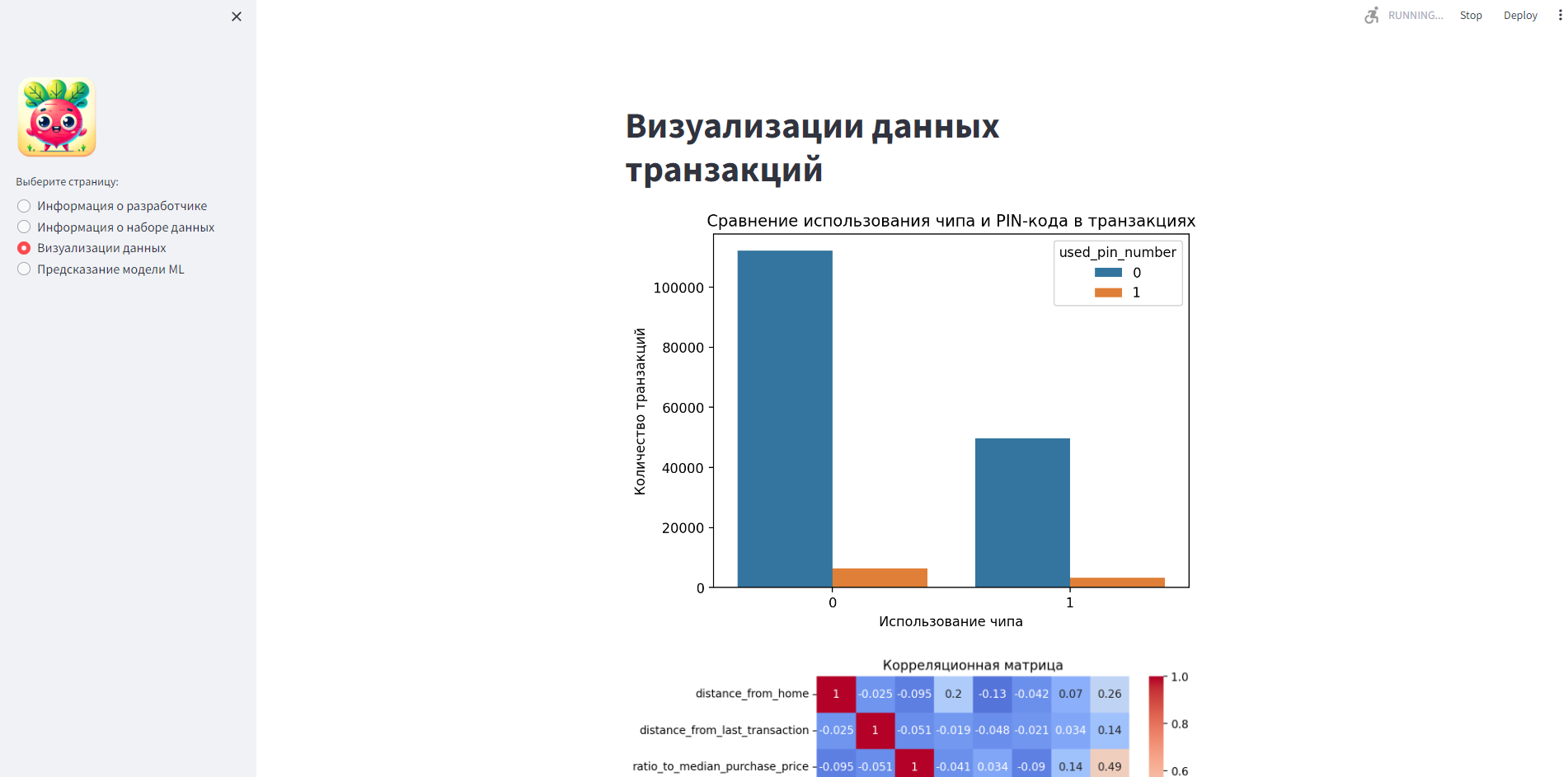


Рисунок 15 – визуализации данных.

Четвертая страница. Предсказание модели *ML*.

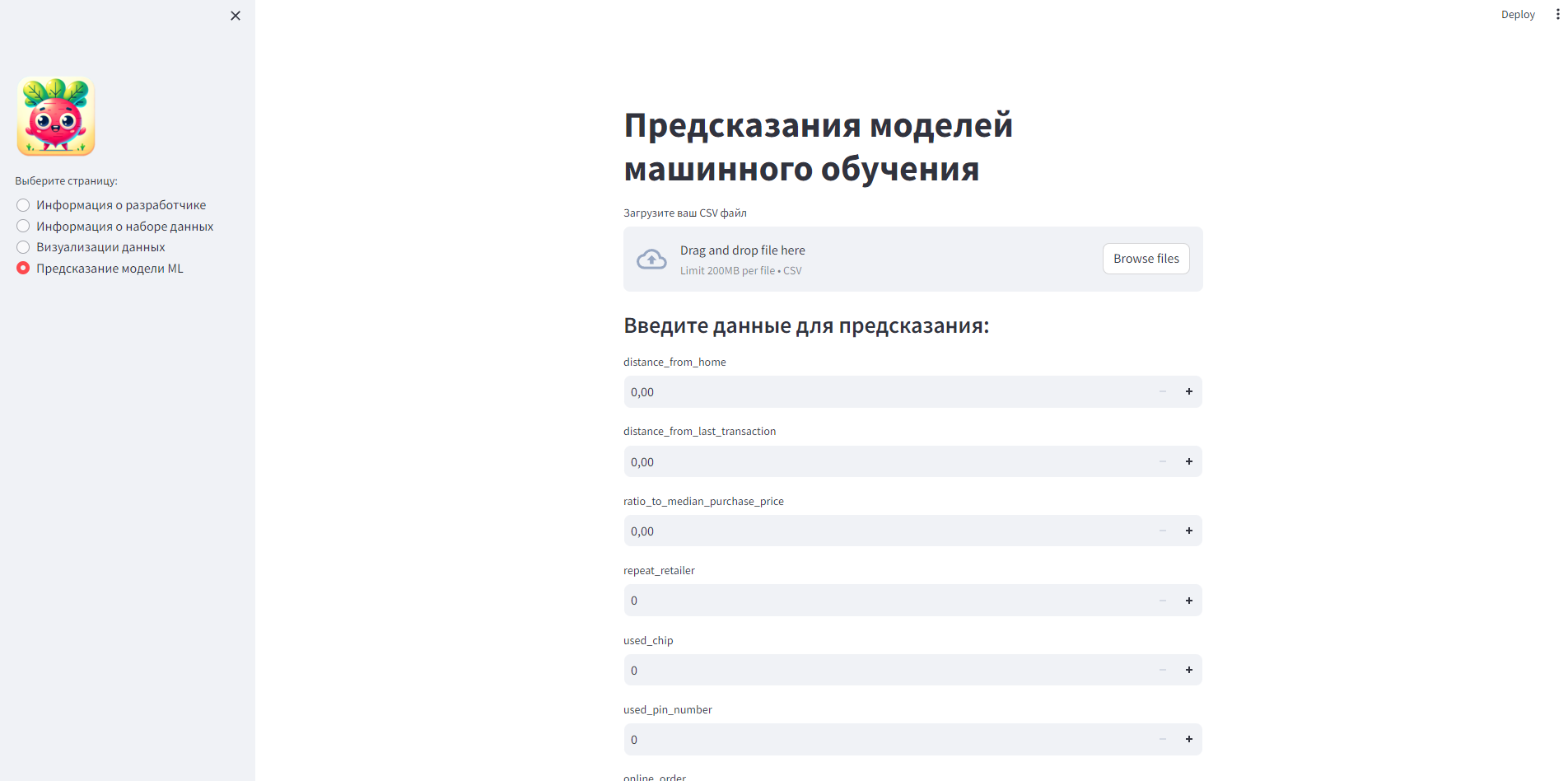


Рисунок 16 – ввод данных на странице предсказаний модели *ML*.

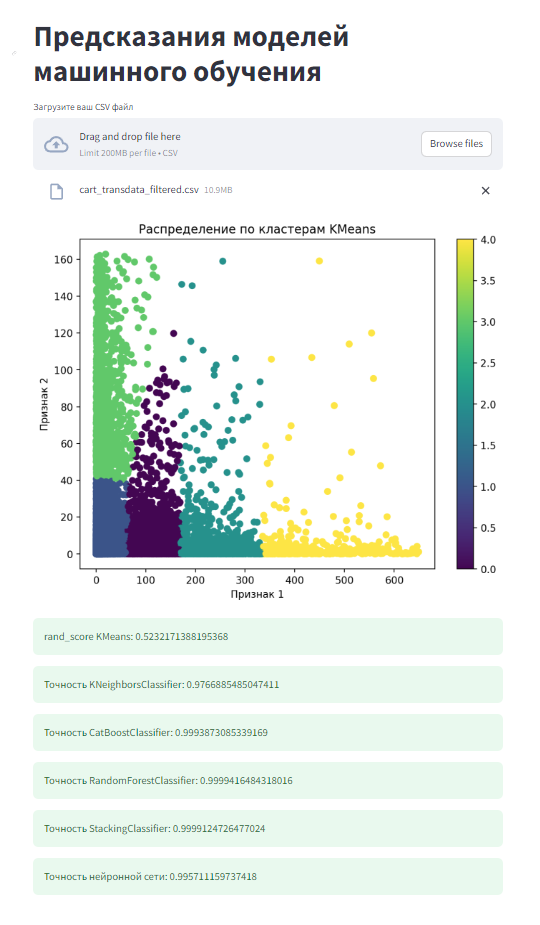


Рисунок 17 – результат загрузки датасета на странице предсказаний модели *ML*.

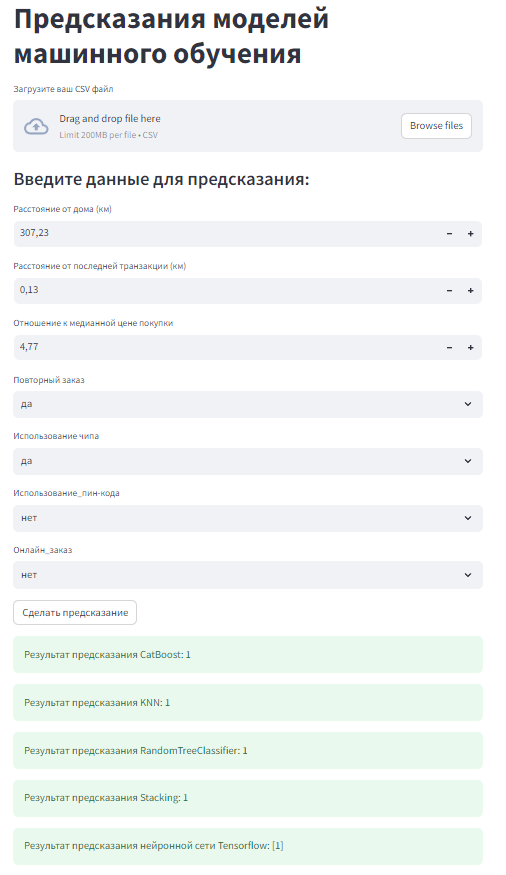


Рисунок 18 – предсказание моделей для данных, с которыми была совершена мошенническая транзакция странице предсказаний модели *ML*.

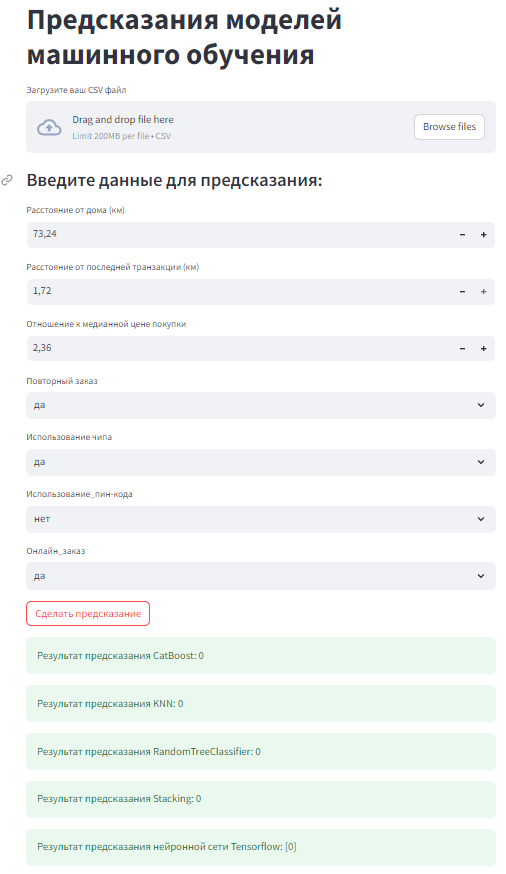


Рисунок 19 – предсказание моделей для данных, с которыми не была совершена мошенническая транзакция странице предсказаний модели *ML*.

**2.4 Ссылка на *GitHub*.**

Весь исходный код дашборда, включая файлы с данными и моделями, был размещен на *GitHub* для удобства доступа и возможности внесения изменений. Код доступен по следующей ссылке:

<https://github.com/OverCart345/RGR_for_MO>

Содержание *README.md* файла:

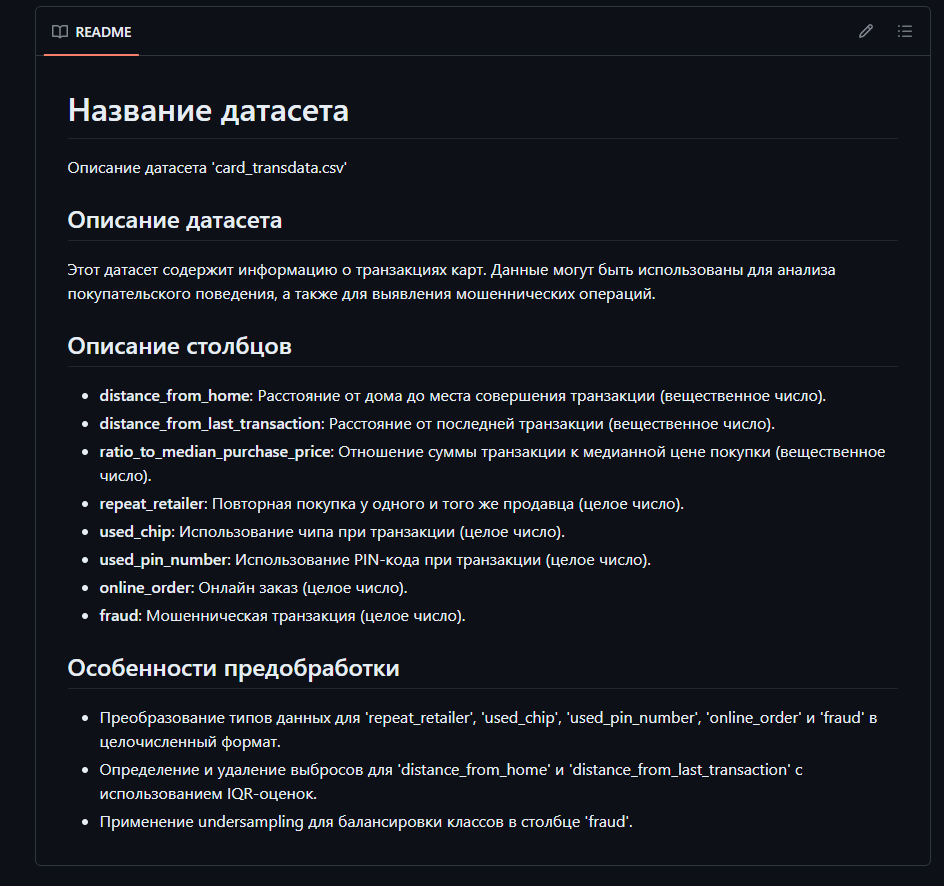


Рисунок 20 – *Readme* на *GitHub*.

**ЗАКЛЮЧЕНИЕ**

В рамках данной расчетно-графической работы (РГР) было выполнено создание веб-приложения на платформе Streamlit, направленного на анализ и визуализацию данных о транзакциях с использованием датасета 'card\_transdata.csv'. Основное внимание уделено анализу и предсказанию мошеннических транзакций на основе предоставленных данных. Основные достижения этой работы включают:

1. Разработка Веб-приложения: было успешно создано веб-приложение, предоставляющее пользовательский интерфейс для взаимодействия с данными о транзакциях. Пользователи могут удобно и интуитивно проводить анализ и визуализацию данных, а также предсказывать мошеннические транзакции в реальном времени.
2. Предобработка и Анализ Данных: произведена тщательная предобработка данных, включая преобразование типов, обработку пропущенных значений и устранение выбросов. Эти этапы подготовили данные для более точного анализа и моделирования.
3. Реализация Моделей Машинного Обучения: в рамках проекта было применено и обучено несколько моделей машинного обучения, включая древовидные модели, ансамблевые методы и нейронные сети. Модели были направлены на предсказание мошеннических транзакций на основе имеющихся данных.
4. Визуализация Данных: веб-приложение предоставляет богатые возможности для визуализации данных, включая графики и диаграммы, которые позволяют пользователям наглядно изучать распределения и тренды в данных.

Разработанное веб-приложение исследует потенциал современных технологий машинного обучения и анализа данных в контексте анализа и предсказания мошеннических транзакций с использованием данных о карточных транзакциях. Проект подчеркивает важность применения таких технологий для обработки и анализа больших объемов информации в реальном времени.

Таким образом, данная работа успешно достигла своих целей и продемонстрировала ключевые принципы работы с данными и машинным обучением в контексте разработки современных веб-приложений для анализа финансовых операций.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Документация по языку программирования *Python.* URL: https://www.python.org/doc/ (дата обращения 18.11.23).
2. Документация по библиотеке *StreamLit*. URL: https://docs.streamlit.io/ (дата обращения 18.11.23)

**Приложение А**

Первая страница. Информация о разработчике.

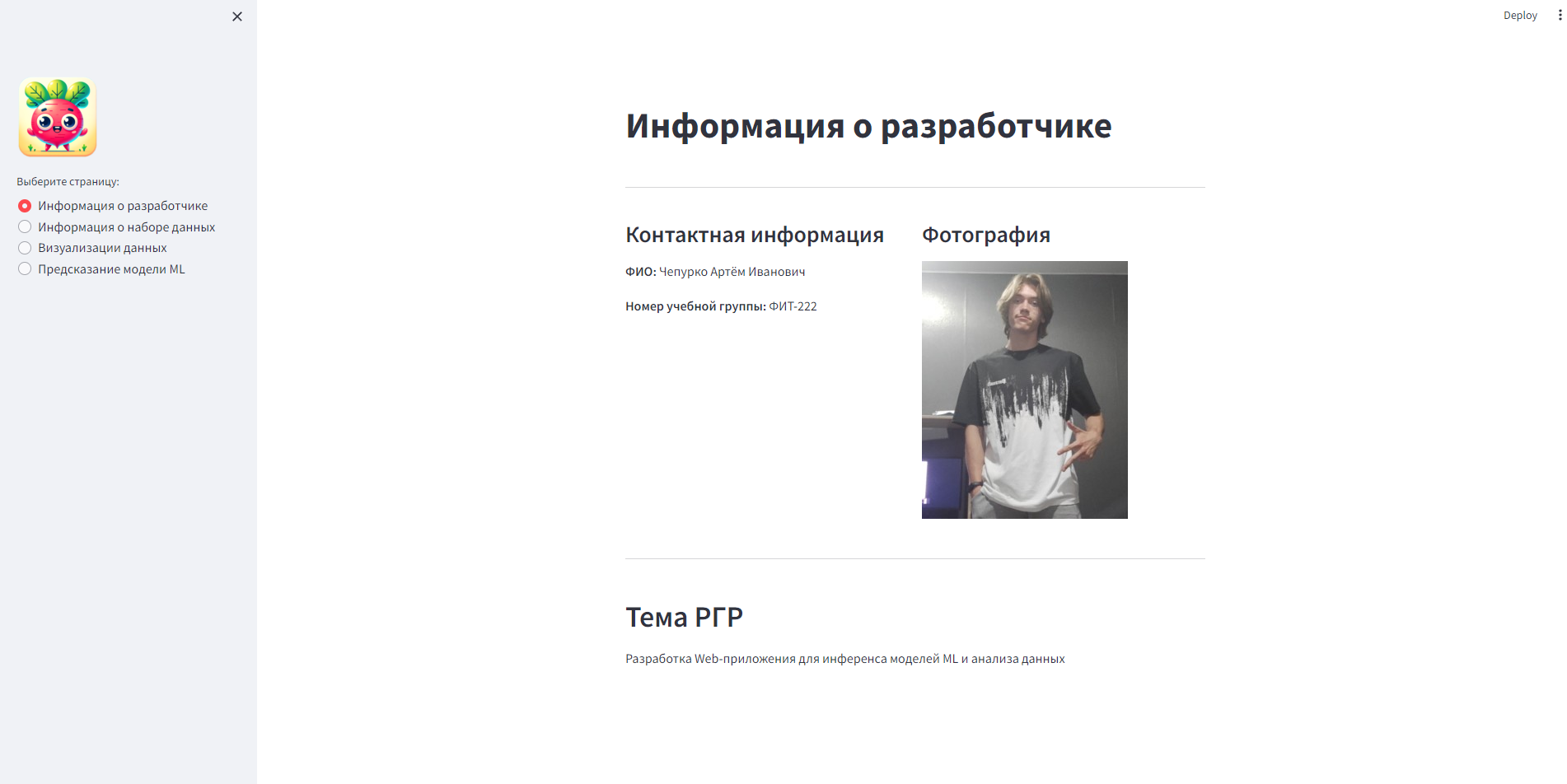


Рисунок 21 – информация о разработчике.

Вторая страница. Информация о наборе данных.

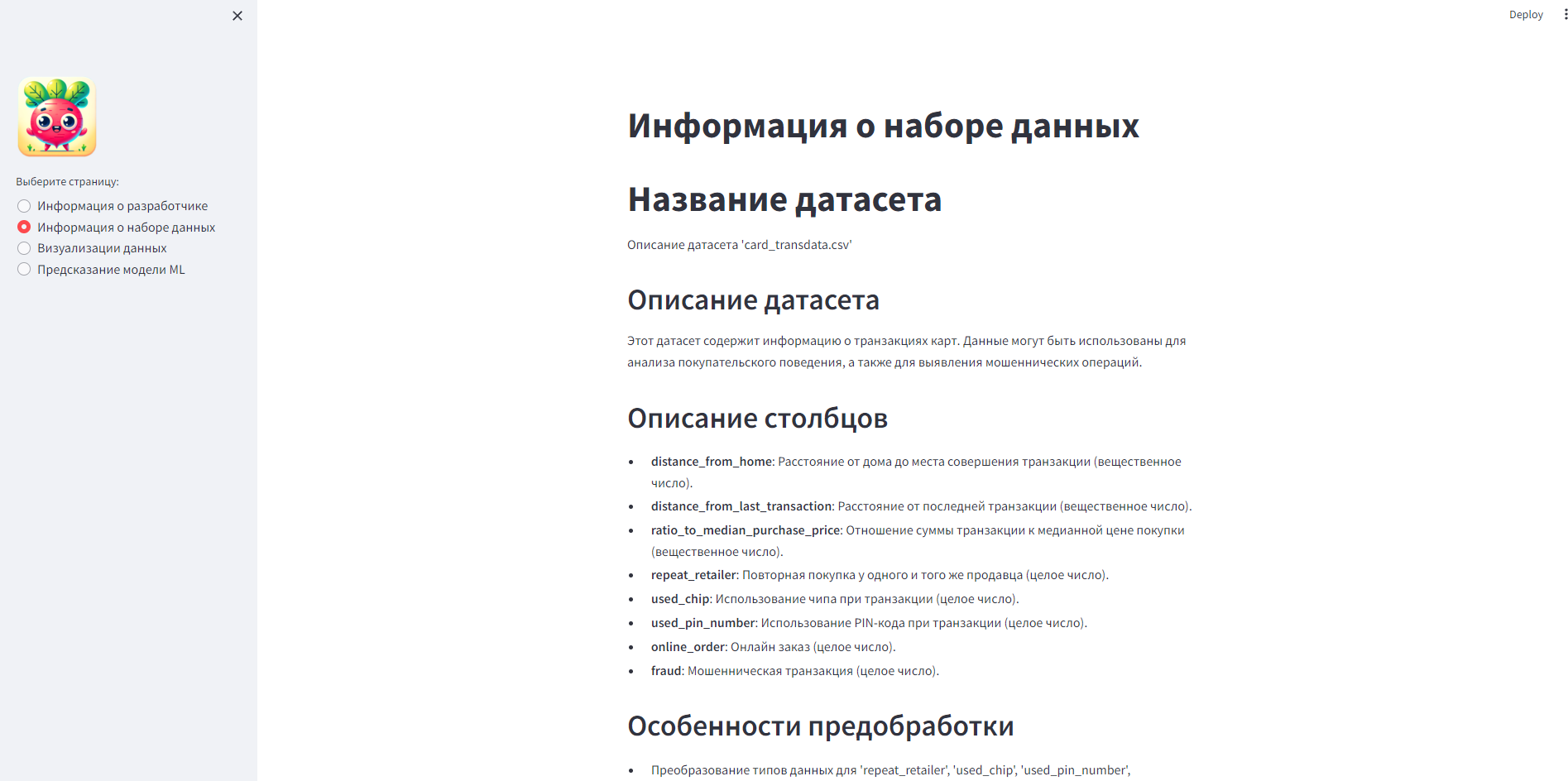


Рисунок 22 – информация о наборе данных.

Третья страница. Визуализации данных.

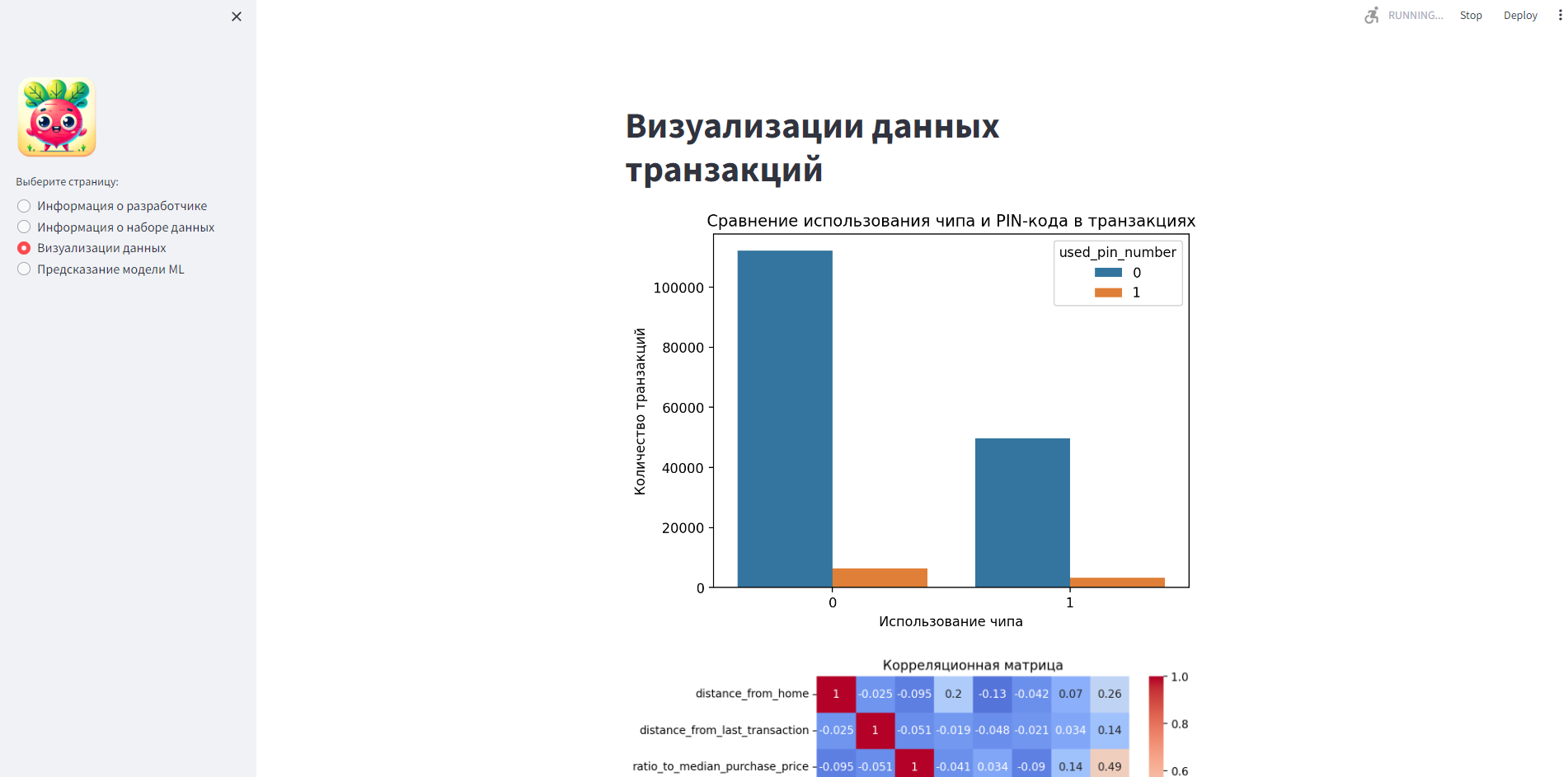


Рисунок 23 – визуализации данных.

Четвертая страница. Предсказание модели *ML*.

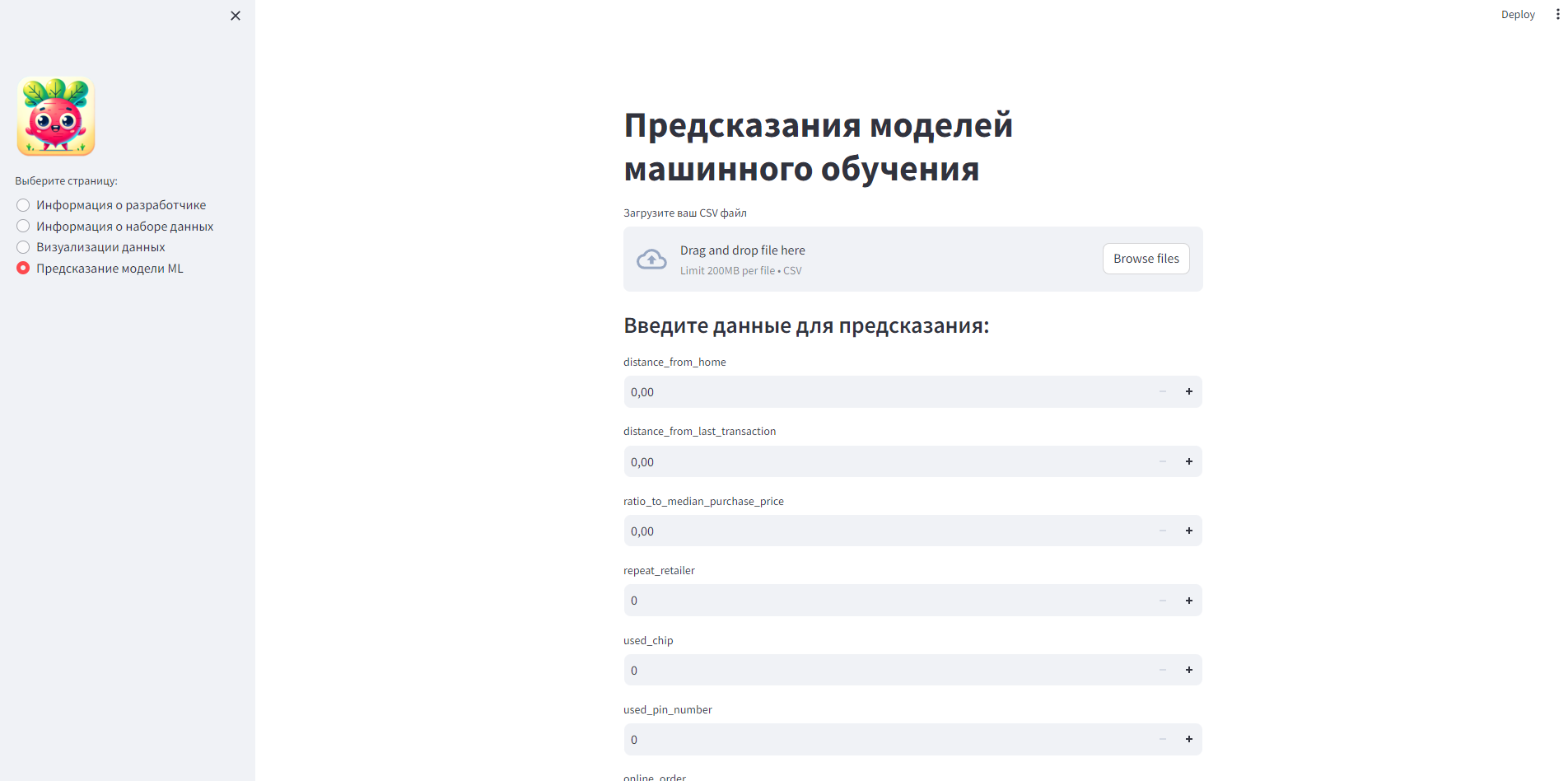


Рисунок 24 – предсказание модели *ML*.

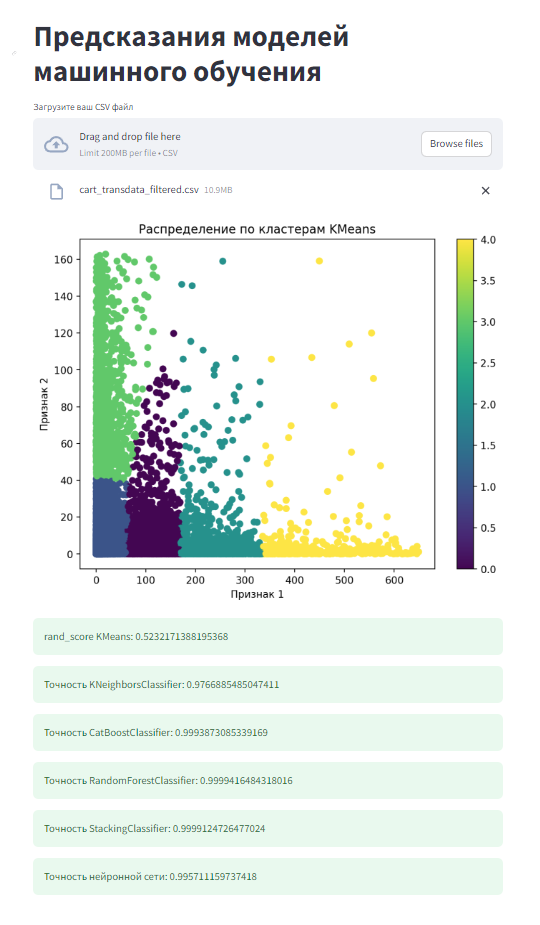


Рисунок 25 – результат загрузки датасета на странице предсказаний модели *ML*.

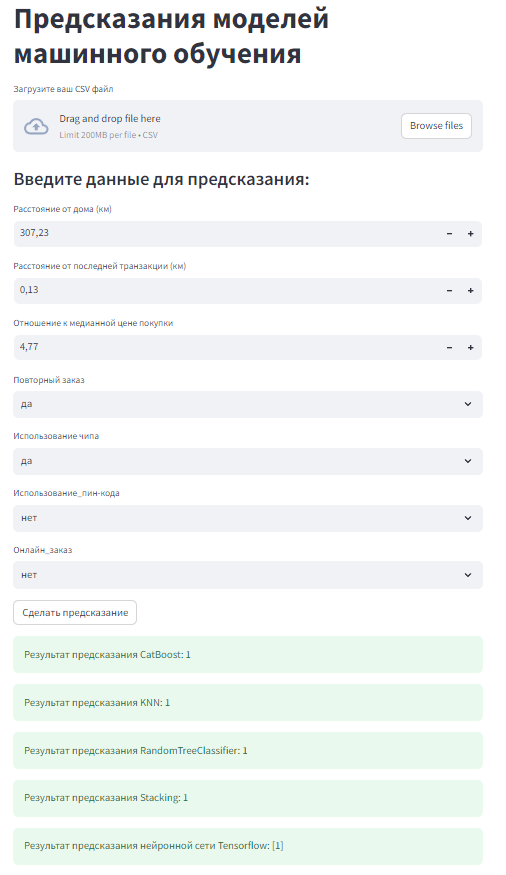


Рисунок 26 – предсказание моделей для данных, с которыми была совершена мошенническая транзакция странице предсказаний модели *ML*.

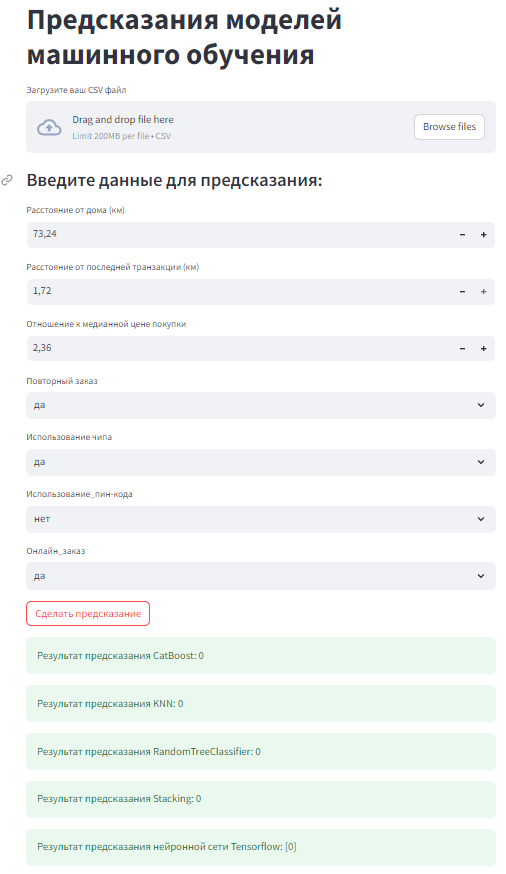


Рисунок 27 – предсказание моделей для данных, с которыми не была совершена мошенническая транзакция странице предсказаний модели *ML*

**ПРИЛОЖЕНИЕ Б**

***modelsSerialisation.py*:**

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import StackingClassifier

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

from sklearn.cluster import KMeans

import pickle

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import  GridSearchCV

from sklearn.cluster  import KMeans

from catboost import CatBoostClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.tree import DecisionTreeClassifier

cart\_data = pd.read\_csv('cart\_transdata\_filtered.csv')

X = cart\_data.drop('fraud', axis=1)

y = cart\_data['fraud']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

parameters = {'n\_neighbors': [2,3,5,10], 'weights': ['uniform', 'distance']}

knn = KNeighborsClassifier()

clf = GridSearchCV(knn, parameters, cv=5)

clf.fit(X\_train, y\_train)

best\_params = clf.best\_params\_

best\_knn = KNeighborsClassifier(\*\*best\_params)

best\_knn.fit(X\_train, y\_train)

with open('best\_knn\_model.pkl', 'wb') as file:

    pickle.dump(best\_knn, file)

best\_gmm\_model = KMeans(n\_clusters=5)

best\_gmm\_model.fit(X\_train)

with open('best\_kmeans5\_model.pkl', 'wb') as file:

    pickle.dump(best\_gmm\_model, file)

params = {'depth': [4, 6, 8], 'learning\_rate': [0.01, 0.05, 0.1], 'iterations': [500]}

cat\_grid = GridSearchCV(CatBoostClassifier(), params, cv=3)

catboost = CatBoostClassifier(depth=4, learning\_rate=0.05, iterations=500)

catboost.fit(X\_train, y\_train)

with open('best\_catboost\_model.pkl', 'wb') as file:

    pickle.dump(catboost, file)

param\_grid = {

    'n\_estimators': [100, 200, 300],

    'max\_features': ['auto', 'sqrt', 'log2'],

    'max\_depth': [4, 5, 6, 7, 8],

}

rfc = RandomForestClassifier(random\_state=42)

CV\_rfc = GridSearchCV(estimator=rfc, param\_grid=param\_grid, cv=5, verbose=10)

CV\_rfc.fit(X\_train, y\_train)

best\_rfc = RandomForestClassifier(\*\*CV\_rfc.best\_params\_)

best\_rfc.fit(X\_train, y\_train)

with open('best\_RandomTreeClassifer\_model.pkl', 'wb') as file:

    pickle.dump(best\_rfc, file)

base\_classifiers = [

    ('logic', LogisticRegression()),

    ('KNN', KNeighborsClassifier()),

    ('dtc',DecisionTreeClassifier())

]

meta\_classifier = LogisticRegression(random\_state=42)

stacking\_classifier = StackingClassifier(estimators=base\_classifiers, final\_estimator=meta\_classifier)

stacking\_classifier.fit(X\_train, y\_train)

with open('best\_Stacking\_model.pkl', 'wb') as file:

    pickle.dump(stacking\_classifier, file)

model = Sequential([

        Dense(64, activation='relu', input\_shape=(X\_train.shape[1],)),

        Dense(32, activation='relu'),

        Dense(1, activation='sigmoid')

    ])

model.compile(optimizer='adam', loss='binary\_crossentropy', metrics=['accuracy'])

model.fit(X\_train, y\_train, epochs=20, batch\_size=10)

model.save("D:/Cpython/streamlit-rgr-main/bestmodels/" + 'sdf.h5')

***mystreamlit.py:***

import streamlit as st

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score, rand\_score

import pickle

from tensorflow.keras.models import load\_model

# Константы

MODEL\_SAVE\_PATH = 'bestmodels/'

DATA\_FILE\_PATH = 'cart\_transdata\_filtered.csv'

DEVELOPER\_PHOTO\_PATH = 'full.jpg'

SIDEBAR\_IMAGE\_PATH = 'circle.png'

# Загрузка и подготовка данных

data = pd.read\_csv(DATA\_FILE\_PATH)

X = data.drop('fraud', axis=1)

y = data['fraud']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Функции визуализации

def plot\_comparison\_of\_chip\_and\_pin(data):

    fig, ax = plt.subplots()

    sns.countplot(data=data, x='used\_chip', hue='used\_pin\_number', ax=ax)

    ax.set\_title('Сравнение использования чипа и PIN-кода в транзакциях')

    ax.set\_xlabel('Использование чипа')

    ax.set\_ylabel('Количество транзакций')

    st.pyplot(fig)

def plot\_correlation\_matrix(data):

    fig, ax = plt.subplots()

    sns.heatmap(data.corr(), annot=True, cmap='coolwarm', ax=ax)

    ax.set\_title('Корреляционная матрица')

    st.pyplot(fig)

def plot\_correlation\_matrix2(data):

    fig, ax = plt.subplots()

    sns.scatterplot(data=data, x='distance\_from\_home', y='distance\_from\_last\_transaction', hue='fraud', ax=ax)

    ax.set\_title('Взаимосвязь между расстоянием от дома и последней транзакцией')

    ax.set\_xlabel('Расстояние от дома')

    ax.set\_ylabel('Расстояние от последней транзакции')

    st.pyplot(fig)

def plot\_correlation\_matrix3(data):

    fig, ax = plt.subplots()

    sns.countplot(x='fraud', data=data, ax=ax)

    ax.set\_title('Сравнение количества мошеннических и немошеннических транзакций')

    st.pyplot(fig)

def plot\_correlation\_matrix4(data):

    fig, ax = plt.subplots()

    sns.boxplot(x='fraud', y='distance\_from\_home', data=data, ax=ax)

    ax.set\_title('Сравнение расстояний от дома в мошеннических и немошеннических транзакциях')

    st.pyplot(fig)

# Функции страниц

def page\_developer\_info():

    st.title("Информация о разработчике")

    st.markdown("---")

    col1, col2 = st.columns(2)

    with col1:

        st.subheader("Контактная информация")

        st.write("\*\*ФИО:\*\* Чепурко Артём Иванович")

        st.write("\*\*Номер учебной группы:\*\* ФИТ-222")

    with col2:

        st.subheader("Фотография")

        st.image(DEVELOPER\_PHOTO\_PATH, width=250)

    st.markdown("---")

    st.header("Тема РГР")

    st.write("Разработка Web-приложения для инференса моделей ML и анализа данных")

def page\_dataset\_info():

    st.title("Информация о наборе данных")

    st.markdown("""

# Название датасета

Описание датасета 'card\_transdata.csv'

## Описание датасета

Этот датасет содержит информацию о транзакциях карт. Данные могут быть использованы для анализа покупательского поведения, а также для выявления мошеннических операций.

## Описание столбцов

- \*\*distance\_from\_home\*\*: Расстояние от дома до места совершения транзакции (вещественное число).

- \*\*distance\_from\_last\_transaction\*\*: Расстояние от последней транзакции (вещественное число).

- \*\*ratio\_to\_median\_purchase\_price\*\*: Отношение суммы транзакции к медианной цене покупки (вещественное число).

- \*\*repeat\_retailer\*\*: Повторная покупка у одного и того же продавца (целое число).

- \*\*used\_chip\*\*: Использование чипа при транзакции (целое число).

- \*\*used\_pin\_number\*\*: Использование PIN-кода при транзакции (целое число).

- \*\*online\_order\*\*: Онлайн заказ (целое число).

- \*\*fraud\*\*: Мошенническая транзакция (целое число).

## Особенности предобработки

- Преобразование типов данных для 'repeat\_retailer', 'used\_chip', 'used\_pin\_number', 'online\_order' и 'fraud' в целочисленный формат.

- Определение и удаление выбросов для 'distance\_from\_home' и 'distance\_from\_last\_transaction' с использованием IQR-оценок.

- Применение undersampling для балансировки классов в столбце 'fraud'.""")

def page\_data\_visualization():

    st.title("Визуализации данных транзакций")

    plot\_comparison\_of\_chip\_and\_pin(data)

    plot\_correlation\_matrix(data)

    plot\_correlation\_matrix2(data)

    plot\_correlation\_matrix3(data)

    plot\_correlation\_matrix4(data)

def page\_ml\_prediction():

    st.title("Предсказания моделей машинного обучения")

    uploaded\_file = st.file\_uploader("Загрузите ваш CSV файл", type="csv")

    if uploaded\_file is None:

        st.subheader("Введите данные для предсказания:")

        input\_data = {}

        feature\_names\_float = ['distance\_from\_home','distance\_from\_last\_transaction','ratio\_to\_median\_purchase\_price']

        feature\_names\_int = ['repeat\_retailer','used\_chip','used\_pin\_number','online\_order']

        for feature in feature\_names\_float:

            input\_data[feature] = st.number\_input(f"{feature}", min\_value=0.0, max\_value=100000.0, value=0.0)

        for feature in feature\_names\_int:

            input\_data[feature] = st.number\_input(f"{feature}", min\_value=0, max\_value=1, value=0)

        if st.button('Сделать предсказание'):

            model\_catboost, model\_kmeans, model\_knn, model\_random\_tree, model\_stacking, model\_neiro = deserialisation()

            prediction\_catboost = model\_catboost.predict(X\_test)

            prediction\_knn = model\_knn.predict(X\_test)

            prediction\_random\_tree = model\_random\_tree.predict(X\_test)

            prediction\_stacking = model\_stacking.predict(X\_test)

            prediction\_neiro = (model\_neiro.predict(X\_test) > 0.5).astype(int)

            st.success(f"Результат предсказания CatBoost: {prediction\_catboost[0]}")

            st.success(f"Результат предсказания KNN: {prediction\_knn[0]}")

            st.success(f"Результат предсказания RandomTreeClassifier: {prediction\_random\_tree[0]}")

            st.success(f"Результат предсказания Stacking: {prediction\_stacking[0]}")

            st.success(f"Результат предсказания нейронной сети Tensorflow: {prediction\_neiro[0]}")

    else:

        try:

            model\_catboost, model\_kmeans, model\_knn, model\_random\_tree, model\_stacking, model\_neiro = deserialisation()

            prediction\_catboost = model\_catboost.predict(X\_test)

            prediction\_kmeans = model\_kmeans.predict(X\_test)

            prediction\_knn = model\_knn.predict(X\_test)

            prediction\_random\_tree = model\_random\_tree.predict(X\_test)

            prediction\_stacking = model\_stacking.predict(X\_test)

            prediction\_neiro = model\_neiro.predict(X\_test).round()

            rand\_score\_ml2 = rand\_score(y\_test, prediction\_kmeans)

            accuracy\_Knn = accuracy\_score(y\_test, prediction\_knn)

            accuracy\_catboost = accuracy\_score(y\_test, prediction\_catboost)

            accuracy\_random\_tree = accuracy\_score(y\_test, prediction\_random\_tree)

            accuracy\_stacking = accuracy\_score(y\_test, prediction\_stacking)

            accuracy\_neiro = accuracy\_score(y\_test, prediction\_neiro)

            plt.figure(figsize=(8, 6))

            plt.scatter(X\_test.iloc[:, 0], X\_test.iloc[:, 1], c=prediction\_kmeans, cmap='viridis', marker='o')

            plt.title('Распределение по кластерам KMeans')

            plt.xlabel('Признак 1')

            plt.ylabel('Признак 2')

            plt.colorbar()

            st.pyplot(plt)

            st.success(f"rand\_score KMeans: {rand\_score\_ml2}")

            st.success(f"Точность KNeighborsClassifier: {accuracy\_Knn}")

            st.success(f"Точность CatBoostClassifier: {accuracy\_catboost}")

            st.success(f"Точность RandomForestClassifier: {accuracy\_random\_tree}")

            st.success(f"Точность StackingClassifier: {accuracy\_stacking}")

            st.success(f"Точность нейронной сети: {accuracy\_neiro}")

        except Exception as e:

            st.error(f"Произошла ошибка при обработке файла: {e}")

# Функция для загрузки моделей

def deserialisation():

    model\_names = ['best\_catboost\_model.pkl', 'best\_kmeans\_model.pkl', 'best\_knn\_model.pkl',

                    'best\_RandomTreeClassifer\_model.pkl', 'best\_Stacking\_model.pkl', 'best\_neiro\_model.h5']

    models = []

    for model\_name in model\_names:

        model\_path = MODEL\_SAVE\_PATH + model\_name

        if model\_name.endswith('.h5'):

            model = load\_model(model\_path)

        else:

            with open(model\_path, 'rb') as model\_file:

                model = pickle.load(model\_file)

        models.append(model)

    return models

# Основной код Streamlit

def main():

    st.markdown("""

<style>

    body {

        background-color: #f9f9f9;

        color: #333333;

    }

    .sidebar .sidebar-content {

        background-color: #4287f5;

    }

</style>

""", unsafe\_allow\_html=True)

    st.sidebar.image(SIDEBAR\_IMAGE\_PATH, width=100)

    page = st.sidebar.radio("Выберите страницу:", ["Информация о разработчике", "Информация о наборе данных", "Визуализации данных", "Предсказание модели ML"])

    if page == "Информация о разработчике":

        page\_developer\_info()

    elif page == "Информация о наборе данных":

        page\_dataset\_info()

    elif page == "Визуализации данных":

        page\_data\_visualization()

    elif page == "Предсказание модели ML":

        page\_ml\_prediction()

if \_\_name\_\_ == "\_\_main\_\_":

    main()