

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет

вычислительной техники

Кафедра

«Вычислительная техника»

Направление подготовки: 09.03.01 «Информатика и вычислительная техника»

Профиль подготовки: «Вычислительные машины, комплексы, системы и сети»

БАКАЛАВРСКАЯ РАБОТА

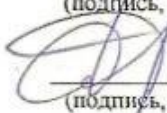
на тему

ПРОГРАММА ПЛАНИРОВАНИЯ
МАРШРУТОВ ГОРОДСКОГО ТАКСИ

Студент

 13.05.2025 Сивушин А.С.
(подпись, дата)

Руководитель

 16.06.2025 Федонин Р.Н.
(подпись, дата)

Нормоконтролёр

 16.06.2025 Бычков А.С.
(подпись, дата)

Работа допущена к защите (протокол зас. кафедры от 28.05.2025 г. № 016)

Заведующий кафедрой


(подпись)

М.А. Митрохин

(фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от

20 июня 2025 г. № 42)

Секретарь ГЭК


(подпись)

Е.И. Гурин

(фамилия, инициалы)

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Факультет
вычислительной техники

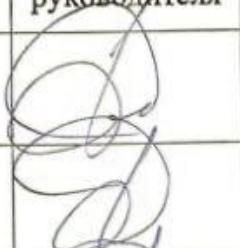
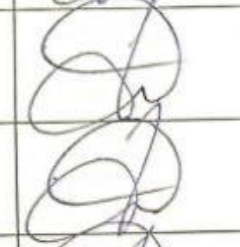
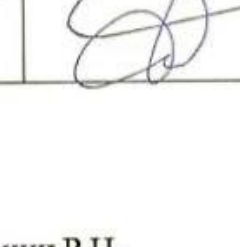
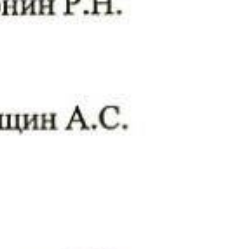
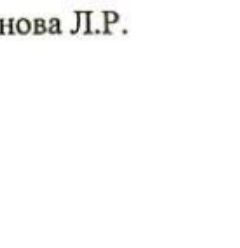
Кафедра
«Вычислительная техника»

«Утверждаю»
Заведующий кафедрой
М.А. Митрохин
«06» 05 2025г.

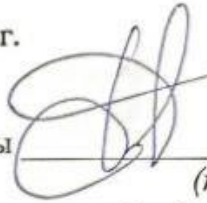
**ЗАДАНИЕ
НА БАКАЛАВРСКУЮ РАБОТУ**

1. Студент Сивущин Артём Сергеевич гр.21ВВП2 факультета вычислительной техники
2. Тема работы «Програма планирования маршрутов городского такси»
Тема утверждена приказом ПГУ № 601/о от "06" мая 2025г.
3. Руководитель бакалаврской работы ПГУ Федюнин Р.Н.
4. Задание на работу (назначение разработки, исходные данные и т.п.)
Необходимо спроектировать и разработать систему предсказания времени поездки на такси на основе машинного обучения. Данная система должна анализировать исторические данные поездок внутри города и строить модель, способную предсказывать длительность поездки на основе координат начальной и конечной точки. Система должна быть удобной для пользователей, предоставляя API для отправки координат поездки и получения предсказания времени поездки. Также должна быть реализована возможность обновления модели на новых данных через API.
5. Перечень подлежащих разработке вопросов:
 - проведение анализа предметной области: изучение процессов прогнозирования времени поездки на такси на основе исторических данных;
 - исследование методов машинного обучения для решения задачи регрессии, в частности алгоритма XGBoost;
 - анализ источников данных, используемых в модели;
 - исследование методов предобработки данных;
 - разработка и обучение модели предсказания времени поездки с последующей оптимизацией её параметров;
 - разработка веб-сервера на Flask для обработки запросов на предсказание;
 - интеграция модели с картографическим сервисом Leaflet для визуализации маршрутов;
 - разработка механизма дообучения модели на новых данных без необходимости полной переобучаемости;
 - разработка интерфейса API для взаимодействия с моделью и возможностью отправки запросов.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	Подпись руководителя
1.	Изучение предметной области	22	06.05.2025-08.05.2025	
2.	Анализ требований к программе	32	09.05.2025-12.05.2025	
3.	Выбор технологий разработки	20	13.05.2025-15.05.2025	
4.	Отбор и проектирование признаков	26	16.05.2025-17.05.2025	
5.	Разработка программы	90	18.05.2025-25.05.2025	
6.	Тестирование программы	20	26.05.2025-27.05.2025	
7.	Оформление пояснительной записки	20	28.05.2025-30.05.2025	

Дата выдачи задания "6" мая 2025 г.

Руководитель бакалаврской работы  06.05.2025 Федюнин Р.Н.
(подпись, дата)

Задание к исполнению принял студент  06.05.2025 Сивущин А.С.
(подпись, дата)

Работу к защите допустить
Декан факультета  19.06.2025 Фионова Л.Р.
(подпись, дата)

Реферат

Бакалаврская работа содержит 109 листов, 61 рисунок, 6 источников, 3 приложения.

ПРЕДСКАЗАНИЕ ВРЕМЕНИ ПОЕЗДКИ, АНАЛИЗ ДАННЫХ, ОБРАБОТКА МАРШРУТОВ, ПРОГНОЗИРОВАНИЕ, МОДЕЛИРОВАНИЕ, PYTHON, FLASK, XGBOOST, SKLEARN, LEAFLET, API

Цель работы – разработать инструмент, позволяющий автоматизировать
Цель работы – разработка инструмента, позволяющего прогнозировать время поездки на такси в городских условиях, используя алгоритмы машинного обучения. Данная система предназначена для анализа данных о поездках, построения модели предсказания времени в пути и предоставления удобного интерфейса взаимодействия.

Одним из путей достижения поставленной задачи является использование модели XGBoost, обученной на исторических данных поездок. Модель учитывает координаты начальной и конечной точки маршрута, количество пассажиров, расстояние и повороты, погодные условия (температура, осадки), день недели, а также кластеризацию географических зон. Данные маршрута загружаются через OSRM API, а погодные условия – через OpenWeather API.

					ПГУ 1.090301.26.001 ПЗ				
Изм.	Лист	№ докум.	Подпись	Дата	Разработка программы планирования маршрутов городского такси Пояснительная записка	Лит.	Лист	Листов	
Разраб.		Сивушин А.С.		15.06.24					
Провер.		Федюнин Р.Н.		16.06.24					
П. Контр		Бычков А.С.		16.06.24			2	109	
Утверд.		Митрохин М.А.		16.06.24		ФВТ, каф. ВТ гр. 21ВВП2			

Содержание

Перечень сокращений и обозначений.....	5
Введение.....	6
1 Разработка системы предсказания времени поездки.	8
1.1 Анализ предметной области	8
1.2 Постановка задачи.....	9
1.3 Решение поставленной задачи.....	10
1.4 Классификация систем предсказания времени поездки	12
1.5 Структура системы предсказания времени поездки	14
1.6 Выбор инструментов разработки	19
1.7 Выбранные технологии	21
2 Разработка алгоритма и программной реализации.....	22
2.1 Знакомство с данными, базовый анализ, расширение данных	22
2.2 Разведывательный анализ данных (EDA)	32
2.3 Отбор и преобразование признаков	35
2.4 Задача регрессии: линейная регрессия и деревья решений.....	40
2.5 Решение задачи регрессии: ансамблевые методы	45
2.6 Инициализация и запуск сервиса	51
2.7 Описание доступных функций	53
2.8 Взаимодействие пользователя с системой	54
2.9 Запуск и демонстрация работы.....	55
3 Безопасность жизнедеятельности.....	62
3.1 Правила безопасности при работе с компьютером	62

3.2	Безопасность работы с серверным оборудованием.....	63
3.3	Кибербезопасность при использовании веб-приложений.....	63
	Заключение	65
	Список использованных источников	67
	Приложение А – UML-диаграммы	68
	Приложение Б – Листинг программы	75
	Приложение В – Презентация.....	104

Перечень сокращений и обозначений

ID (Identifier) – идентификатор объекта

UML (Unified Modeling Language) – унифицированный язык моделирования

API (Application Programming Interface) – интерфейс для взаимодействия программных компонентов

OSRM (Open Source Routing Machine) – сервис маршрутизации с открытым исходным кодом

XGBoost (Extreme Gradient Boosting) – библиотека машинного обучения для градиентного бустинга

RMSE (Root Mean Square Error) – корень из средней квадратичной ошибки, метрика оценки моделей

RMLSE (Root Mean Square Logarithmic Error) – логарифмированный корень из средней квадратичной ошибки, метрика оценки моделей

EDA (Exploratory Data Analysis) – процесс изучения набора данных

OLS (Ordinary least squares) – метод линейной регрессии, который используется для оценки параметров модели

MeAE (Mean absolute error) – медианная абсолютная ошибка

Flask – легковесный веб-фреймворк на Python для разработки API

Leaflet – библиотека для работы с интерактивными картами

DMatrix – специальный формат данных для обучения XGBoost

Введение

Использование методов машинного обучения для анализа и прогнозирования данных становится все более популярны. Одной из задач в этих областях является предсказание времени поездки, что позволяет улучшать планирование маршрутов, оптимизировать работу такси и повышать удобство пользователей.

В последние годы крупные транспортные компании и агрегаторы такси активно внедряют системы анализа данных для повышения точности различных прогнозов. Использование моделей машинного обучения позволяет анализировать большое количество данных о поездках, выявлять закономерности и делать точные предсказания на основе различных факторов: расстояния, дорожных условий, погоды и других параметров.

Одним из инновационных решений является построение системы, способной на основе исторических данных о поездках прогнозировать время в пути с учетом географических координат начала и конца маршрута, погодных условий и других факторов.

Цели выпускной квалификационной работы:

- освоение современных методов обработки данных и машинного обучения;
- изучение нюансов разработки предсказательных моделей и их интеграции в веб-сервис;
- реализация системы предсказания времени поездки с использованием Flask API и машинного обучения.

Задачи, решаемые в ходе выполнения работы:

- проведение анализа данных о поездках, используемых для предсказания;
- определение значимых факторов, влияющих на длительность поездки;
- реализация модели машинного обучения с использованием XGBoost;

- разработка веб-сервиса на Flask, предоставляющего API для получения предсказаний;
- тестирование системы и оценка качества предсказаний.

					ПГУ 1.090301.44.001 ПЗ	Лист
						7
Изм.	Лист	№ докум.	Подпись	Дата		

1 Разработка системы предсказания времени поездки.

1.1 Анализ предметной области

Для повышения точности прогнозирования времени поездки на такси всё чаще применяются методы машинного обучения, позволяющие учитывать множество факторов: дорожные условия, погоду, время суток, расстояние и другие параметры [1]. Современные решения, такие как использование алгоритма XGBoost, показали высокую эффективность в задачах регрессии, особенно при анализе больших объёмов данных из транспортных систем [2].

На сегодняшний день транспортные компании и агрегаторы такси активно используют различные методы предсказания времени в пути. Основными подходами являются:

- использование традиционных алгоритмов, основанных на статистических данных, без учета динамических факторов (например, расстояния, времени суток, загруженности дорог);
- применение гибридных моделей, совмещающих исторические данные с информацией о трафике в реальном времени;
- использование глубокого обучения и градиентного бустинга, позволяющих выявлять сложные зависимости между входными параметрами и временем поездки.

На данный момент существует несколько основных источников информации, используемых для предсказания времени в пути:

- навигационные сервисы – такие как Google Maps, Яндекс.Карты и OSRM, предоставляющие данные о маршрутах, расстояниях и предполагаемом времени в пути;
- исторические данные о поездках – агрегированные данные о прошлых поездках, содержащие координаты начала и конца маршрута, длительность, время суток, количество пассажиров и погодные условия;

– данные о дорожных условиях и погоде – динамические параметры, которые могут существенно влиять на время в пути, такие как осадки, температура и дорожные события.

В настоящее время большинство решений по предсказанию времени поездки являются закрытыми. Однако использование открытых данных, таких как OSRM API и OpenWeather API, позволяет разработать предсказательную модель, доступную для внедрения в различные приложения.

На данный момент не существует универсального решения, которое бы позволяло с высокой точностью предсказывать время поездки, учитывая широкий набор факторов. Поэтому в данной работе разрабатывается система, объединяющая машинное обучение и данные о погодных условиях, что позволит повысить точность прогнозов.

1.2 Постановка задачи

Необходимо спроектировать и разработать систему предсказания времени поездки на такси с использованием машинного обучения. Целью разработки является создание модели, способной прогнозировать длительность поездки на основе различных факторов, включая координаты начальной и конечной точки маршрута, количество пассажиров, дорожные условия и погодные факторы.

Система должна предоставлять следующие возможности со стороны пользователя:

- ввод координат начальной и конечной точки поездки;
- указание количества пассажиров;
- автоматический расчет расстояния и количества поворотов на маршруте;
- учет погодных условий для более точного прогноза;
- получение предсказанного времени поездки;
- визуализация маршрута с использованием Leaflet.js.

Система должна предоставлять следующие возможности со стороны сервера:

- обработка запросов от пользователей через веб-сервис Flask API;
- запрос данных о маршруте через OSRM API (расстояние, количество шагов, время в пути);
- запрос метеорологических данных через OpenWeather API;
- предобработка входных данных и передача их в модель машинного обучения;
- прогнозирование времени поездки с использованием обученной модели XGBoost;
- возвращение результата пользователю в виде JSON-ответа.

Система должна предоставлять следующие возможности со стороны разработчика:

- реализация механизма дообучения модели на новых данных;
- поддержка API для интеграции с другими сервисами;
- оптимизация работы модели для повышения точности предсказаний;
- разработка удобного интерфейса взаимодействия с системой.

1.3 Решение поставленной задачи

Система предсказания времени поездки на такси – это программный продукт, использующий машинное обучение для анализа маршрута и предоставления прогноза времени в пути. Она позволяет предсказывать длительность поездки на основе различных факторов, включая расстояние, количество пассажиров, погодные условия и количество поворотов на маршруте.

Интеграция внешних API, таких как OSRM (Open Source Routing Machine) для получения информации о маршрутах, и OpenWeatherMap API для учёта погодных условий, позволяет значительно повысить точность прогнозов [3]. Также важным аспектом является использование фреймворков вроде Flask для

реализации RESTful API, что обеспечивает удобное взаимодействие между клиентской и серверной частями системы [4].

Подобные системы применяются в различных сферах:

- Google Maps, Яндекс.Карты – сервисы, предоставляющие прогнозы времени поездки на основе данных о пробках и исторической информации;
- Uber, Яндекс.Такси – используют машинное обучение для определения времени ожидания такси и предсказания продолжительности поездки;
- приложения городского транспорта – такие как Moovit, анализируют маршруты общественного транспорта и прогнозируют время в пути;
- навигационные системы автомобилей – встроенные решения, учитывающие дорожную ситуацию и предоставляющие рекомендации по маршрутам.

Преимущества системы предсказания времени поездки:

- использование XGBoost и предобработки данных позволяет учитывать широкий спектр факторов;
- данные о маршрутах запрашиваются через OSRM API, а погодные условия – через OpenWeather API, что делает предсказания более точными;
- модель может обучаться на новых данных, повышая свою точность с течением времени;
- система может быть интегрирована в мобильные приложения, диспетчерские службы такси и другие транспортные платформы;
- автоматическое предсказание времени поездки снижает нагрузку на операторов такси и диспетчеров;
- система предоставляет прогнозы в любое время суток без задержек;
- алгоритм можно адаптировать для предсказания времени поездок в различных городах.

1.4 Классификация систем предсказания времени поездки

Системы предсказания времени поездки различаются по используемым алгоритмам, методам обработки данных и источникам информации. В данном разделе рассмотрены основные классификации таких систем.

Классификация по алгоритму прогнозирования:

– системы, основанные на статистических данных;

1) используют простые модели, например, рассчитывающие среднее время поездки для заданного расстояния;

2) не учитывают динамические факторы, такие как пробки, погода или время суток;

3) подходят для приблизительных оценок, но обладают низкой точностью;

– системы, использующие машинное обучение;

1) анализируют исторические данные и выявляют закономерности;

2) учитывают такие параметры, как расстояние, количество пассажиров, дорожные условия, погодные условия и географическое положение;

3) позволяют строить точные предсказания за счет использования алгоритмов, таких как XGBoost, Random Forest, нейронные сети;

– гибридные системы;

1) комбинируют статистические модели с методами машинного обучения;

2) используют предсказания модели, но могут корректировать их на основе текущих данных о трафике;

3) позволяют балансировать между простотой модели и высокой точностью прогнозов.

Классификация по источникам данных:

– системы, использующие только исторические данные;

1) модель обучается на данных прошлых поездок и делает предсказания, основываясь на схожих маршрутах;

2) ограничена тем, что не учитывает текущие дорожные условия;

– системы, использующие реальные данные о маршруте;

1) интегрируются с сервисами, такими как OSRM API, для получения информации о расстоянии, количестве поворотов и сложности маршрута;

2) позволяют учитывать особенности дорог и городской инфраструктуры;

– системы, использующие данные о погоде и внешних факторах;

1) получают информацию из OpenWeather API о температуре, осадках, скорости ветра, что позволяет учитывать влияние погодных условий на время поездки;

2) могут быть полезны для городов, где климатические условия значительно влияют на транспорт;

Классификация по способу взаимодействия с пользователем:

– автономные предсказательные системы. Работают без участия пользователя, например, встраиваются в навигационные системы и автоматически рассчитывают время поездки;

– системы с API-интерфейсом;

1) пользователь отправляет запрос (например, координаты начала и конца поездки), а система возвращает предсказанное время;

2) используются в сервисах такси, мобильных приложениях и транспортных платформах;

– системы с визуализацией маршрутов;

1) позволяют пользователю выбрать начальную и конечную точки на карте, после чего отображают предсказанное время поездки;

2) реализуются с использованием Leaflet.js и картографических сервисов;

Классификация по функциональности:

– системы для индивидуального пользования. Предоставляют пользователю информацию о времени поездки для планирования маршрутов;

– системы для транспортных компаний и агрегаторов такси. Используются для оптимизации работы таксопарков, прогнозирования загруженности и расчета стоимости поездок;

– системы для городского планирования. Анализируют транспортные потоки и помогают оптимизировать городскую инфраструктуру;

В настоящий момент наибольшую популярность имеют системы, использующие машинное обучение и интеграцию с внешними API. Разрабатываемая в данной работе система использует XGBoost для построения модели, OSRM API для получения данных о маршруте и OpenWeather API для учета погодных условий. Это позволяет достигнуть высокой точности предсказаний.

Системы такого типа находят применение не только в такси, но и в логистике, городском транспорте и других областях, где требуется быстрое и точное планирование маршрутов [5]. Более того, возможность дообучения модели на новых данных делает такую систему адаптивной к изменению дорожной обстановки и климатических условий со временем [6].

1.5 Структура системы предсказания времени поездки

Система предсказания времени поездки представляет собой программу, анализирующую входные данные и прогнозирующую длительность маршрута с использованием машинного обучения.

Архитектура системы состоит из трех основных компонентов:

- клиентская часть;
- серверная часть;
- модель машинного обучения.

Клиентская часть представляет собой интерфейс, через который пользователь взаимодействует с системой. В данном случае в качестве

клиентского интерфейса используется веб-приложение с картографическим сервисом Leaflet.js.

Пользователь выбирает начальную и конечную точку маршрута, количество пассажиров в поездке, после чего отправляет запрос на сервер. Клиентская часть также отображает результат предсказания в виде рассчитанного времени поездки и построенного маршрута.

Серверная часть системы реализована на Flask и отвечает за обработку запросов, выполняя следующие задачи:

- получение входных данных (координаты маршрута, количество пассажиров);
- запрос к API OSRM для получения данных о маршруте (расстояние, количество поворотов, время в пути);
- запрос к OpenWeather API для получения погодных условий в месте поездки;
- предобработка данных перед подачей в модель машинного обучения;
- вызов обученной модели XGBoost для предсказания времени поездки;
- отправка результата клиенту в виде JSON-ответа.

Диаграмма развёртывания на рисунке 1 демонстрирует структуру клиент-серверной архитектуры.

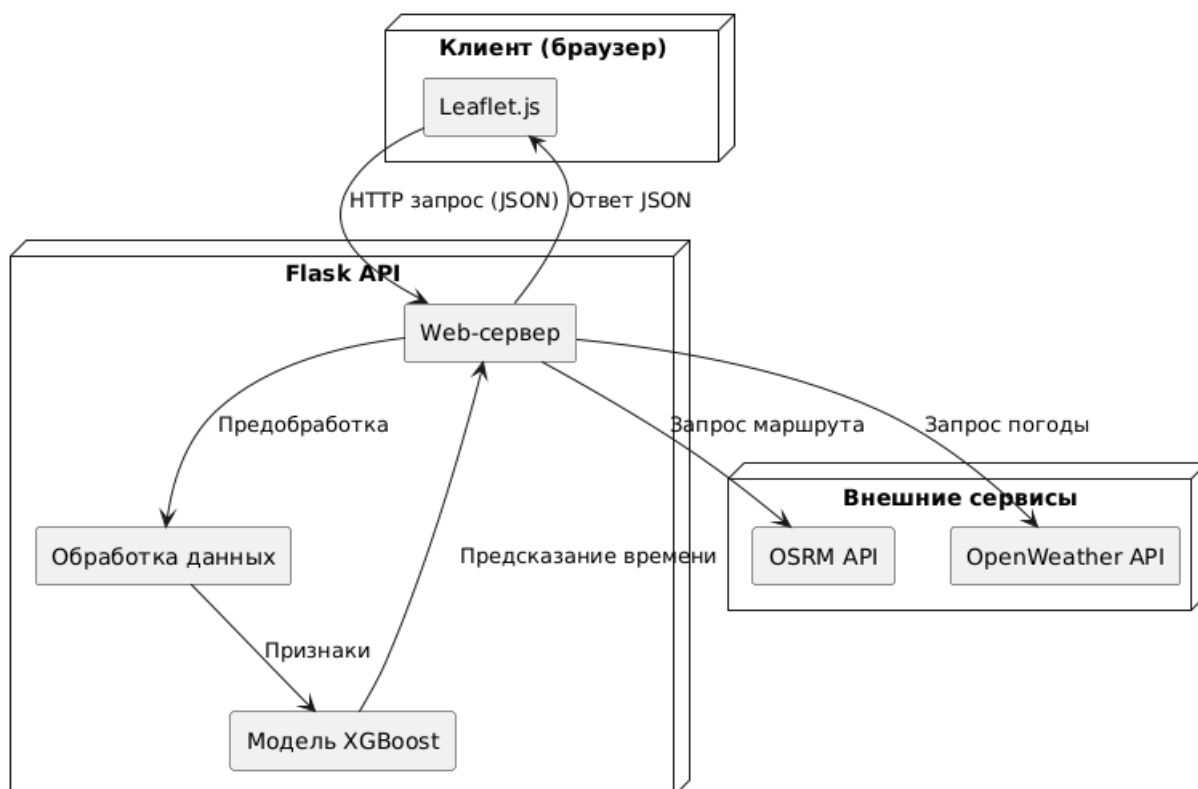


Рисунок 1 – Диаграмма развертывания

Диаграмма последовательностей на рисунке 2 демонстрирует взаимодействие пользователя с программой.

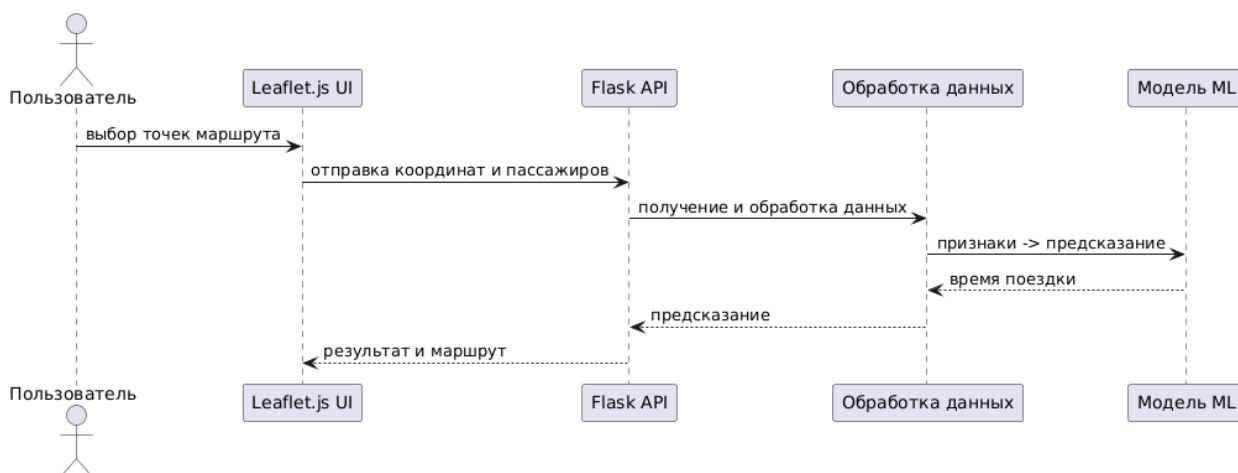


Рисунок 2 – Диаграмма последовательностей

Диаграмма прецедентов на рисунке 3 демонстрирует взаимодействие с программой со стороны пользователей с различными уровнями доступа к функционалу.



Рисунок 3 – Диаграмма прецедентов

Диаграмма вариантов использования на рисунке 4 демонстрирует как различные пользователи взаимодействуют непосредственно с моделью, предсказывающую время поездки.

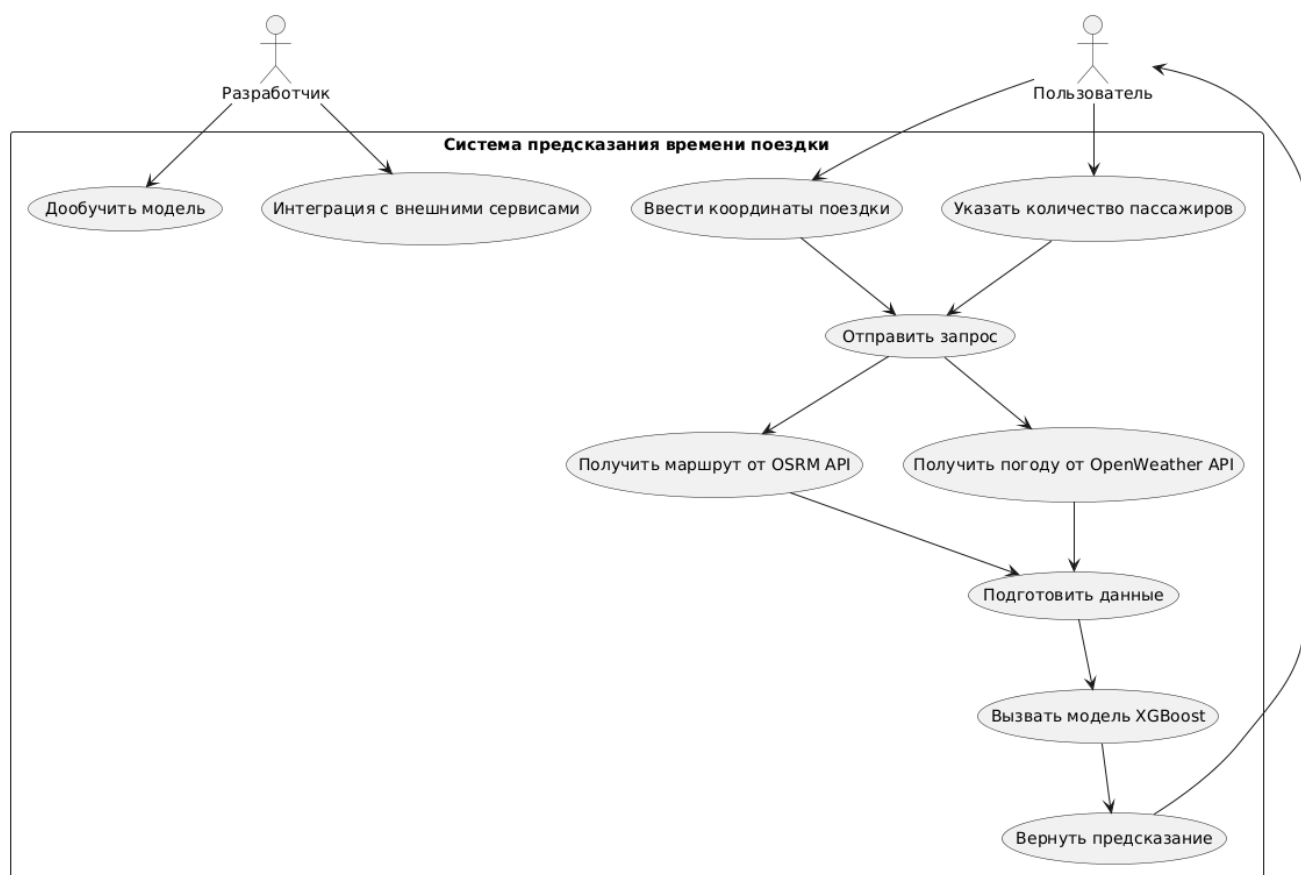


Рисунок 4 – UML-диаграмма вариантов использования

Модель машинного обучения.

В основе системы лежит алгоритм XGBoost, который обучен на большом массиве данных о поездках. Он принимает в качестве входных параметров:

- координаты начала и конца поездки;
- количество пассажиров;
- расстояние маршрута;
- количество поворотов;
- температуру и погодные условия;

- день недели и праздничные дни;
- кластеризацию географических районов;
- выходным значением является предсказанное время поездки.

Модель была обучена на 1 450 000 записей, что позволяет ей эффективно анализировать входные параметры и выдавать точные предсказания.

Обмен данными в системе.

- клиентская часть отправляет запрос с координатами маршрута;
- серверная часть обрабатывает запрос, получает дополнительные данные из OSRM API и OpenWeather API;
- модель машинного обучения выполняет предсказание;
- серверная часть возвращает результат клиенту;
- клиентская часть отображает предсказанное время поездки и маршрут на карте.

Таким образом, система предсказания времени поездки представляет собой объединение веб-технологий, API-интеграций и машинного обучения, что обеспечивает высокую точность прогнозирования.

1.6 Выбор инструментов разработки

1.6.1. Выбор архитектуры и API для маршрутизации

При разработке системы предсказания времени поездки важно выбрать инструмент для получения информации о маршрутах. Основные альтернативы:

- Google Maps API – предоставляет подробные данные о маршрутах, но требует платной подписки;
- Яндекс.Карты API – аналогичен Google Maps, но также имеет ограничения по бесплатному использованию;
- OSRM (Open Source Routing Machine) – бесплатное и мощное решение, которое позволяет рассчитывать маршруты и получать информацию о расстоянии, времени поездки и количестве поворотов.

1.6.2. Выбор языка программирования

Разработка системы предсказания времени поездки требует использования языка программирования, который поддерживает обработку больших данных, интеграцию с API и машинное обучение. Рассмотрены следующие варианты:

– Java:

- 1) Поддерживает многопоточность и высокую производительность;
- 2) Требуется большого объема кода и сложной настройки;
- 3) Не является стандартным выбором для машинного обучения;

– C#:

- 1) Хорошо интегрируется с Windows-приложениями;
- 2) Требуется .NET-экосистемы, что ограничивает кроссплатформенность;
- 3) Подходит для корпоративных решений, но не для работы с Python-библиотеками машинного обучения;

– Python:

- 1) Высокоуровневый язык с мощными библиотеками для обработки данных (pandas, NumPy, scikit-learn, XGBoost);
- 2) Простая интеграция с API и базами данных;
- 3) Является стандартом в области машинного обучения.

1.6.3. Выбор фреймворка для веб-сервера

Система должна предоставлять API для взаимодействия с пользователями. Рассмотрены следующие фреймворки:

- Django – мощный, но требует больше ресурсов и подходит для крупных веб-приложений;
- Flask – легковесный, простой в настройке и оптимален для API;
- FastAPI – более быстрый, чем Flask, но менее популярен.

1.6.4. Выбор модели машинного обучения

Для предсказания времени поездки рассмотрены несколько алгоритмов:

- линейная регрессия – проста, но не учитывает сложные зависимости;

- Random Forest – более точный, но медленный;
- XGBoost – высокая точность, поддержка работы с большими данными.

1.7 Выбранные технологии

В ходе проведенных анализов, было приняты следующие решения:

- в данной системе выбрана OSRM API как наиболее доступная и гибкая технология для получения данных о маршрутах;
- выбор сделан в пользу Python, так как он предоставляет удобные инструменты для работы с данными, предобработки и обучения моделей;
- выбран Flask, так как он обеспечивает удобную разработку REST API и хорошо совместим с Python;
- выбран XGBoost, так как он демонстрирует лучшую точность при предсказании времени поездки, так же он показал более точные результаты на тестовой выборке.

					ПГУ 1.090301.44.001 ПЗ	Лист
						21
Изм.	Лист	№ докум.	Подпись	Дата		

2 Разработка алгоритма и программной реализации

2.1 Знакомство с данными, базовый анализ, расширение данных

Основные цели проекта:

- сформировать набор данных на основе нескольких источников информации;
- спроектировать новые признаки с помощью Feature Engineering и выявить наиболее значимые при построении модели;
- исследовать предоставленные данные и выявить закономерности;
- построить несколько моделей и выбрать из них наилучшую по заданной метрике;
- спроектировать процесс предсказания времени длительности поездки для новых данных.

Для начала мы проведем базовый анализ того, насколько данные готовы к дальнейшей предобработке и анализу.

					ПГУ 1.090301.44.001 ПЗ	Лист
						22
Изм.	Лист	№ докум.	Подпись	Дата		

Посмотрим на временные рамки на рисунке 5, в которых мы работаем с данными. Переведем признак pickup_datetime в тип данных datetime с форматом «год-месяц-день час:минута:секунда». Дополнительно посмотрим на пропуски в данных.

```
taxi_data['pickup_datetime'] = pd.to_datetime(taxi_data['pickup_datetime'], format='%Y-%m-%d %H:%M:%S')
print(f"{taxi_data['pickup_datetime'].dt.date.min()} to {taxi_data['pickup_datetime'].dt.date.max()}")
✓ 0.9s

2016-01-01 to 2016-06-30

taxi_data.isnull().sum()
✓ 0.0s

id                0
vendor_id         0
pickup_datetime   0
dropoff_datetime  0
passenger_count   0
pickup_longitude  0
pickup_latitude   0
dropoff_longitude 0
dropoff_latitude  0
store_and_fwd_flag 0
trip_duration      0
dtype: int64
```

Рисунок 5 – проверка периодов времени и наличие пропусков

Пропусков в данных нет.

Посмотрим на статистические характеристики некоторых признаков на рисунке 6.

```
round(taxi_data.describe())
✓ 0.3s
```

	vendor_id	pickup_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration
count	1458644.0	1458644	1458644.0	1458644.0	1458644.0	1458644.0	1458644.0	1458644.0
mean	2.0	2016-04-01 10:10:24.940037120	2.0	-74.0	41.0	-74.0	41.0	959.0
min	1.0	2016-01-01 00:00:17	0.0	-122.0	34.0	-122.0	32.0	1.0
25%	1.0	2016-02-17 16:46:04.249999872	1.0	-74.0	41.0	-74.0	41.0	397.0
50%	2.0	2016-04-01 17:19:40	1.0	-74.0	41.0	-74.0	41.0	662.0
75%	2.0	2016-05-15 03:56:08.750000128	2.0	-74.0	41.0	-74.0	41.0	1075.0
max	2.0	2016-06-30 23:59:39	9.0	-61.0	52.0	-61.0	44.0	3526282.0
std	0.0	NaN	1.0	0.0	0.0	0.0	0.0	5237.0

Рисунок 6 – статистические характеристики

В данных присутствуют поездки, продолжительность которых составляет одну секунду, а также поездки, продолжительность которых больше 40 дней. Это выбросы.

Чтобы однозначно обозначить такие записи в таблице как выбросы, необходимо произвести более глубокий анализ данных с подключением дополнительной информации.

На основе полученной статистической информации о целевом признаке `trip_duration` мы также делаем вывод, что использование среднего в качестве меры распределения длительности поездок будет ошибкой. Лучше всего подойдут меры, не зависящие от выбросов. При проведении исследований мы будем использовать медиану, а наша метрика качества моделей будет завязана на логарифмическом среднем.

Займемся расширением исходного набора данных как с помощью внешних источников, так и с помощью манипуляций над имеющимися в данных признаками.

Так как все манипуляции над данными мы будем применять как с тренировочным, так и с тестовым набором данных, во избежание дублирования кода мы будем производить предобработку с помощью функций.

Реализуем функцию `add_datetime_features()`, показанную на рисунке 7, которая принимает на вход таблицу с данными о поездках (`DataFrame`) и возвращает ту же таблицу с добавленными в нее 3 столбцами:

- `pickup_date` - дата включения счетчика - начало поездки (без времени);
- `pickup_hour` - час дня включения счетчика;
- `pickup_day_of_week` - наименование дня недели, в который был включен счетчик.

```
def add_datetime_features(df):
    df['pickup_date'] = df['pickup_datetime'].dt.date
    df['pickup_hour'] = df['pickup_datetime'].dt.hour
    df['pickup_day_of_week'] = df['pickup_datetime'].dt.dayofweek
    return df

taxi_data = add_datetime_features(taxi_data)
print("Среднее количество поездок в день:", round(taxi_data.shape[0] / taxi_data['pickup_date'].nunique()))
✓ 0.2s

Среднее количество поездок в день: 8015
```

Рисунок 7 – функция add_datetime_features()

От того, является ли день праздничным, часто зависит ситуация на дорогах. Большое количество автомобилей создаёт пробки и потенциально увеличивает длительность совершаемой на такси поездки. Добавим в наш датасет информацию о том, является ли день праздничным. Для этого у нас есть датасет holiday_data.csv.

Реализуем функцию add_holiday_features(), которая принимает на вход две таблицы:

- таблицу с данными о поездках;
- таблицу с данными о праздничных днях.

Функция add_holiday_features() на рисунке 8 возвращает обновленную таблицу с данными о поездках с добавленным в нее столбцом pickup_holiday - бинарным признаком того, начата ли поездка в праздничный день или нет (1 - да, 0 - нет).

```
def add_holiday_features(df1, df2):
    df2['date'] = pd.to_datetime(df2['date']).dt.date
    lst = df2['date'].tolist()
    df1['pickup_holiday'] = df1['pickup_date'].apply(lambda x: 1 if x in lst else 0)
    return df1

taxi_data = add_holiday_features(taxi_data, holiday_data)
print(f"Медианная длительность поездки в праздники: {round(taxi_data[taxi_data['pickup_holiday'] == 1]['trip_duration'].median())} секунд")
✓ 0.4s

Медианная длительность поездки в праздники: 585 секунд
```

Рисунок 8 – функция add_holiday_features()

Теперь, когда мы разобрались с временными характеристиками, переходим к географическим. В данных есть столбцы, содержащие

информацию о широте и долготе. Из них можно извлечь очень много полезной информации.

Реализуем функцию `add_osrm_features()` (см. рисунок 9), которая принимает на вход две таблицы:

- таблицу с данными о поездках;
- таблицу с данными из OSRM.

Функция возвращает обновленную таблицу с данными о поездках с добавленными в нее 3 столбцами:

- `total_distance`;
- `total_travel_time`;
- `number_of_steps`.

```
def add_osrm_features(df1, df2):  
    df1 = df1.merge(df2[['id', 'total_distance', 'total_travel_time', 'number_of_steps']], left_on='id', right_on='id', how='left')  
    return df1
```

Рисунок 9 – функция `add_holiday_features()`

С помощью географических характеристик можно извлечь ещё немного информации. Если рассматривать Землю как сферу, то, зная широту и долготу начальной и конечной точек маршрута, можно вычислить географическое расстояние между точками на сфере, а также узнать направление движения автомобиля.

Воспользуемся следующими функциями:

- `get_haversine_distance` — функция для вычисления расстояния по формуле гаверсинуса (в километрах);
- `get_angle_direction` — функция для вычисления угла направления движения (в градусах).

Реализация функция показана на рисунке 10.

Каждая из функций имеет четыре аргумента:

- `lat1` и `lng1` — векторы-столбцы с широтой и долготой первой точки;

– lat2 и lng2 — векторы-столбцы с широтой и долготой второй точки.

```
def get_haversine_distance(lat1, lng1, lat2, lng2):
    # переводим углы в радианы
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    # радиус земли в километрах
    EARTH_RADIUS = 6371
    # считаем кратчайшее расстояние h по формуле Хаверсина
    lat_delta = lat2 - lat1
    lng_delta = lng2 - lng1
    d = np.sin(lat_delta * 0.5) ** 2 + np.cos(lat1) * np.cos(lat2) * np.sin(lng_delta * 0.5) ** 2
    h = 2 * EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

def get_angle_direction(lat1, lng1, lat2, lng2):
    # переводим углы в радианы
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2, lng2))
    # считаем угол направления движения alpha по формуле угла пеленга
    lng_delta_rad = lng2 - lng1
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) * np.cos(lat2) * np.cos(lng_delta_rad)
    alpha = np.degrees(np.arctan2(y, x))
    return alpha
```

Рисунок 10 – функции get_haversine_distance() и get_angle_direction()

Реализуем функцию add_geographical_features() (см. рисунок 11), которая принимает на вход таблицу с данными о поездках и возвращает обновленную таблицу с добавленными в нее 2 столбцами:

- haversine_distance - расстояние Хаверсина между точкой, в которой был включен счетчик, и точкой, в которой счетчик был выключен;
- direction - направление движения из точки, в которой был включен счетчик, в точку, в которой счетчик был выключен.

```
def add_geographical_features(df):
    df['havers_distance'] = get_haversine_distance(df['pickup_latitude'], df['pickup_longitude'], df['dropoff_latitude'], df['dropoff_longitude'])
    df['direction'] = get_angle_direction(df['pickup_latitude'], df['pickup_longitude'], df['dropoff_latitude'], df['dropoff_longitude'])
    return df
```

Рисунок 11 – функция add_geographical_features()

Следующая методика манипуляций с географическими данными — это применение кластеризации по географическим координатам. Идея заключается в том, чтобы сгруппировать все поездки в зависимости от координат начала и конца поездки с помощью методов кластеризации, тем самым добавив информацию о районах, в которых находятся точки старта и конца поездки.

Наша гипотеза состоит в том, что длительность поездки будет зависеть от того, к какому географическому кластеру будет принадлежать поездка.

В качестве метода кластеризации воспользуемся алгоритмом K-Means. Возьмём десять кластеров и обучим его на данных, сформированных из широты и долготы всех точек начала и конца поездок (см. рисунок 12).

```
coords = np.hstack((taxi_data[['pickup_latitude', 'pickup_longitude']],
                    taxi_data[['dropoff_latitude', 'dropoff_longitude']]))
kmeans = cluster.KMeans(n_clusters=10, n_init=10, random_state=42)
kmeans.fit(coords)
```

Рисунок 12 – кластеризации по географическим координатам

Реализуем функцию `add_cluster_features()` (см. рисунок 13), которая принимает на вход таблицу с данными о поездках и обученный алгоритм кластеризации, и возвращает обновленную таблицу с добавленным в нее столбцом `geo_cluster` - географический кластер, к которому относится поездка.

```
def add_cluster_features(df, kmeans):
    df['geo_cluster'] = kmeans.predict(coords)
    return df
```

Рисунок 13 – функция `add_cluster_features()`

Другим важным фактором, потенциально влияющим на дорожное движение, является погода. Так, например, логично предположить, что снег и дождь могут затруднять движение и увеличивать время поездки.

Реализуем функцию `add_weather_features()` (см. рисунок 14), которая принимает на вход две таблицы:

- таблицу с данными о поездках;
- таблицу с данными о погодных условиях на каждый час.

Функция возвращает обновленную таблицу с данными о поездках с добавленными в нее 5 столбцами:

- temperature - температура;
- visibility - видимость;
- wind speed - средняя скорость ветра;
- precip - количество осадков;
- events - погодные явления.

```
def add_weather_features(df1, df2):
    df2['date'] = pd.to_datetime(df2['time']).dt.date
    df2['hour'] = pd.to_datetime(df2['time']).dt.hour
    df1 = df1.merge(df2, how='left', left_on=['pickup_date', 'pickup_hour'], right_on=['date', 'hour'])
    df1.drop(columns=['time', 'windchill', 'heat index', 'humidity', 'pressure', 'dew Point', 'wind dir', 'gust speed', 'conditions', 'date', 'hour'], axis=1, inplace=True)
    return df1
```

Рисунок 14 – функция add_weather_features()

В результате объединения у нас получились записи пропуски в столбцах temperature, visibility, wind speed, precip и events. Это связано с тем, что в таблице с данными о погодных условиях отсутствуют измерения для некоторых моментов времени, в которых включался счетчик поездки.

Теперь, когда мы расширили датасет новыми признаками, нам необходимо немного его подчистить. В результате добавления новой информации у нас появились пропущенные значения в данных. Благодаря новым добавленным признакам мы сможем более детально изучить поездки с аномальной длительностью.

Реализуем функцию fill_null_weather_data() (см. рисунок 15), которая принимает на вход таблицу с данными о поездках. Функция должна заполнять пропущенные значения в столбцах.

Пропуски в столбцах с погодными условиями - temperature, visibility, wind speed, precip заполним медианным значением температуры, влажности, скорости ветра и видимости в зависимости от даты начала поездки.

Пропуски в столбце events заполним строкой 'None' - символом отсутствия погодных явлений (снега/дождя/тумана).

Пропуски в столбцах с информацией из OSRM API - total_distance, total_travel_time и number_of_steps заполним медианным значением по столбцам.

```
def fill_null_weather_data(df):
    col1 = ['temperature', 'visibility', 'wind speed', 'precip']
    df[col1] = df[col1].fillna(df.groupby('pickup_date')[col1].transform('median'))

    df['events'] = df['events'].fillna('None')

    col2 = ['total_distance', 'total_travel_time', 'number_of_steps']
    df[col2] = df[col2].fillna(df[col2].median())
    return df
```

Рисунок 15 – функция fill_null_weather_data()

В завершение первой части найдем очевидные выбросы в целевой переменной - длительности поездки.

Проще всего найти слишком продолжительные поездки. Выбросами будут считаться поездки, длительность которых превышает 24 часа.

Чуть сложнее с анализом поездок, длительность которых слишком мала. Потому что к ним относятся действительно реальные поездки на короткие расстояния, поездки, которые были отменены через секунду после того, как включился счетчик, а также перемещения на большие расстояния за считанные секунды.

Считать выбросами будем только последнюю группу. Воспользуемся информацией о кратчайшем расстоянии, которое проезжает такси. Вычислить среднюю скорость автомобиля на кратчайшем пути следующим образом:

$$v_{\text{cp}} = \frac{S}{1000 * t} * 3600,$$

где:

v_{cp} – средняя скорость;

S – расстояние;

T – длительность поездки.

На рисунке 16 показана диаграмма рассеяния средней скорости движения автомобилей:

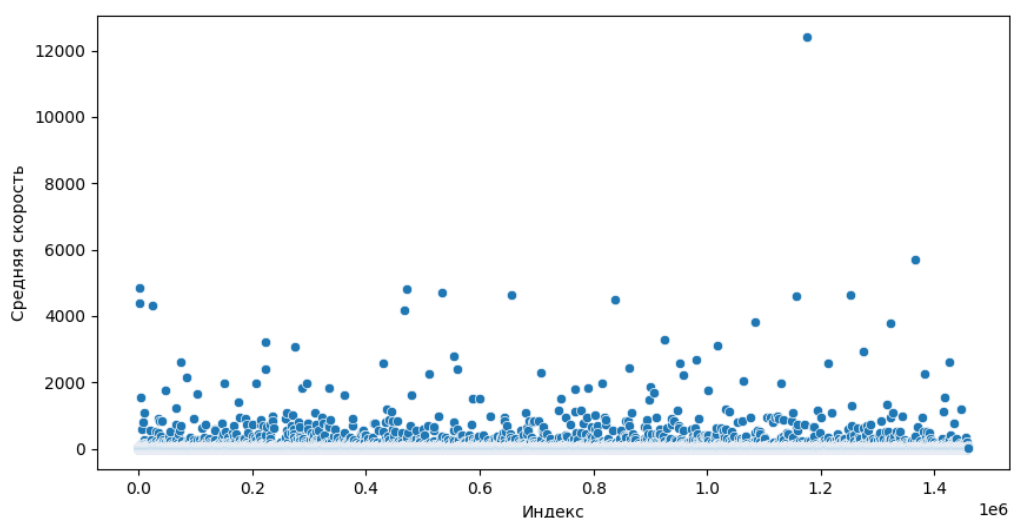


Рисунок 16 – диаграмма рассеяния средней скорости

Найдем поездки, длительность которых превышает 24 часа, и удалим их из набора данных. Так же найдем поездки, средняя скорость которых по кратчайшему пути превышает 300 км/ч и удалим их из данных (см. рисунок 17).

```
taxi_data = taxi_data[(taxi_data['trip_duration'] <= 24*60*60)]
✓ 0.1s

avg_speed = taxi_data['total_distance'] / taxi_data['trip_duration'] * 3.6
taxi_data = taxi_data[avg_speed <= 300]
✓ 0.1s
```

Рисунок 17 – удаление выбросов по скорости движения

2.2 Разведывательный анализ данных (EDA)

Начинаем с целевого признака. Основной метрикой качества решения, поставленной задачи, будет RMSLE - Root Mean Squared Log Error, которая вычисляется на основе целевой переменной в логарифмическом масштабе. В таком случае целесообразно сразу логарифмировать признак длительности поездки и рассматривать при анализе логарифм в качестве целевого признака:

$$trip_duration_log = \log(trip_duration + 1),$$

где:

$trip_duration$ – длительность поездки;

$trip_duration$ – длительность поездки в логарифмическом масштабе;

$\log$ - натуральный логарифм.

На рисунке 18 показаны построенная гистограмма и коробчатая диаграмма длительности поездок в логарифмическом масштабе ($trip_duration_log$).

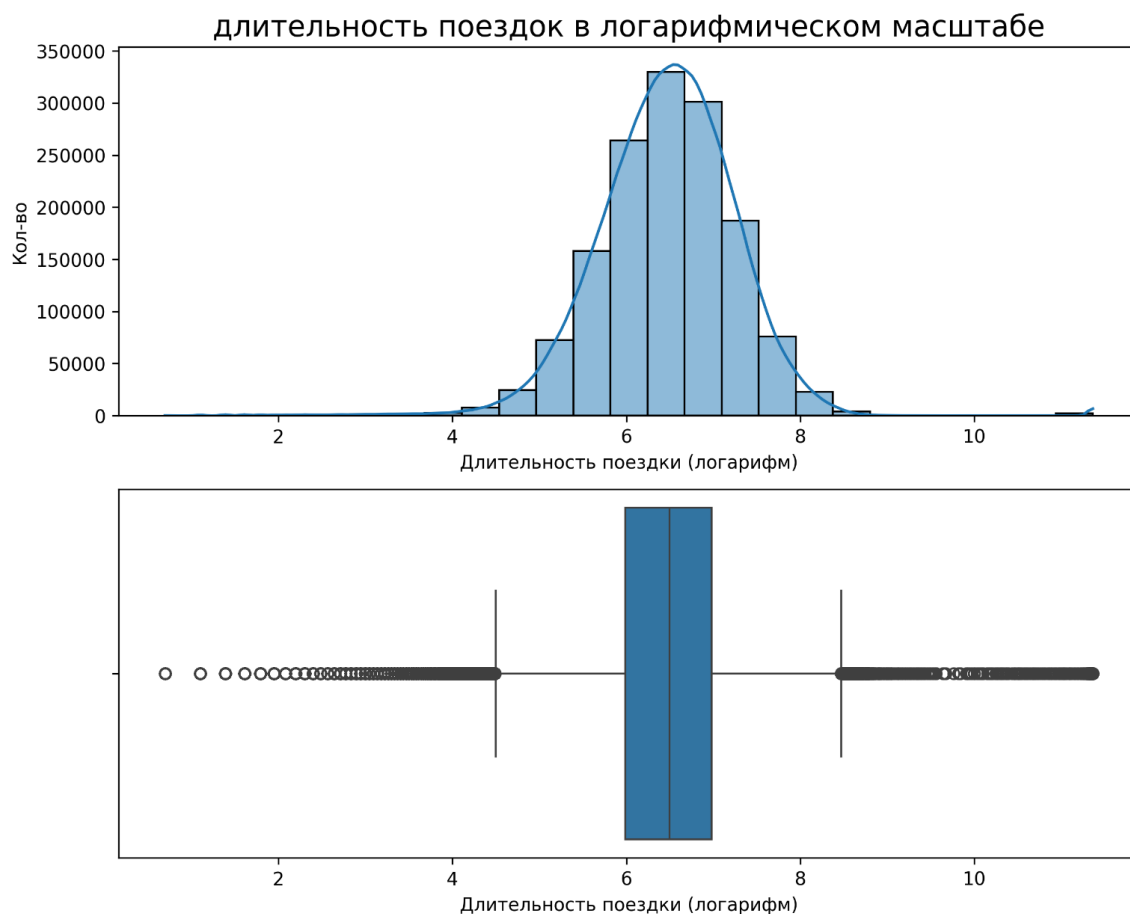


Рисунок 18 – гистограмма и коробчатая диаграмма длительности поездок в логарифмическом масштабе

Построим квантиль-квантиль график (Q-Q Plot), который визуализирует, насколько распределение выборки близко к нормальному.

На рисунке 19 показано, что если все точки лежат вдоль диагонали, то мы считаем, что данные в выборке распределены нормально.

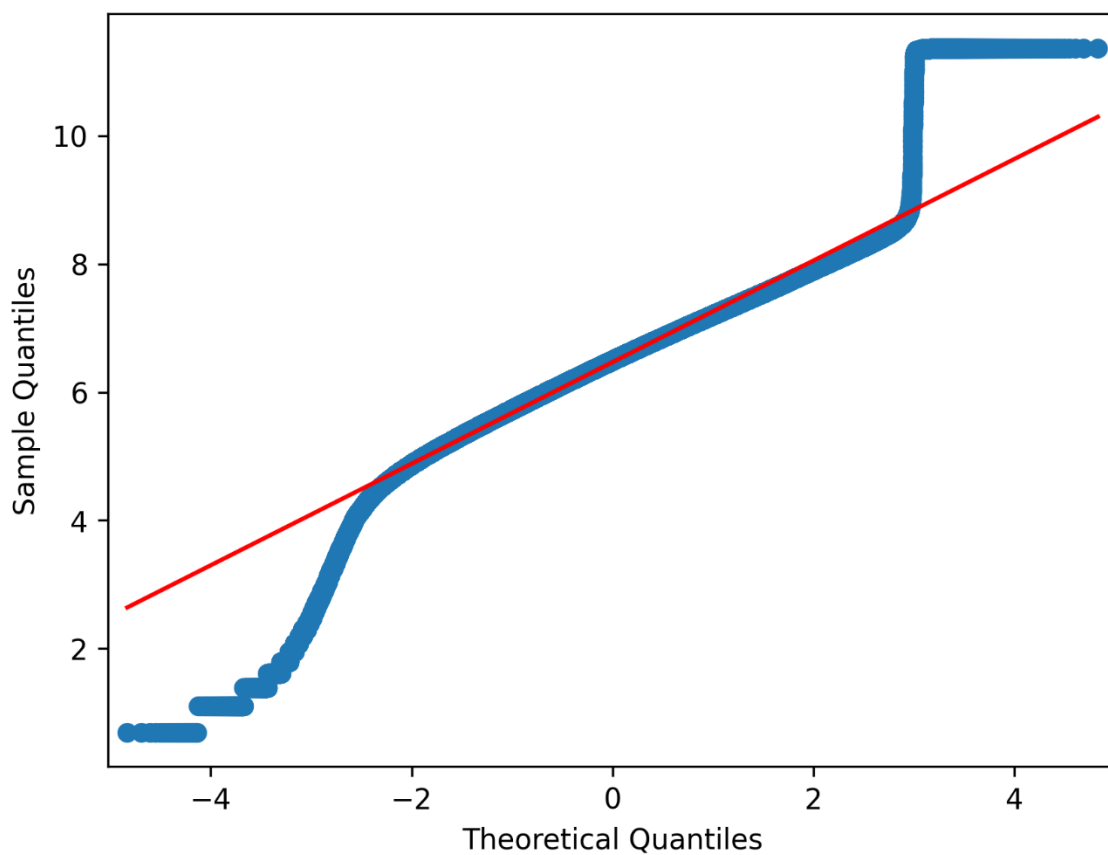


Рисунок 19 – квантиль-квантиль график

На основе данных графика можно сделать вывод, что значения могут быть и не подчинены нормальному закону распределения.

На рисунке 20 показаны две построенные визуализации:

- распределение количества поездок в зависимости от часа дня;
- зависимость медианной длительности поездки от часа дня.

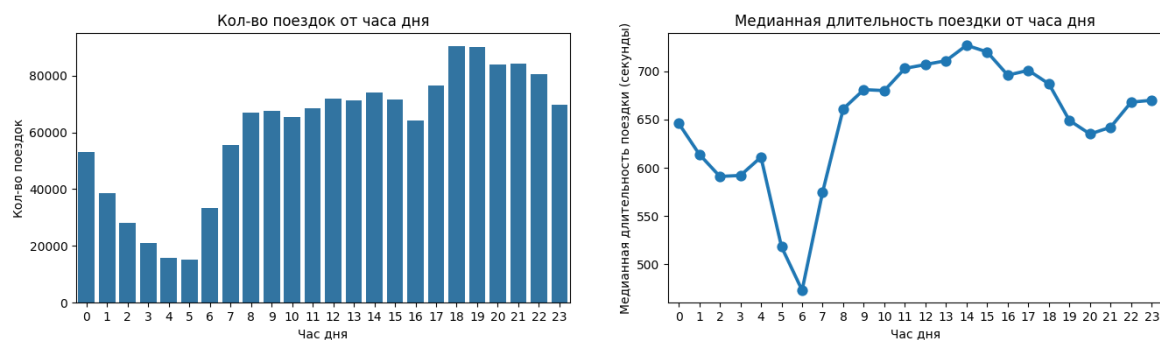


Рисунок 20 – количество поездок в зависимости от часа дня и медианная длительности поездки от часа дня

На рисунке 21 показаны две другие построенные визуализации:

- распределение количества поездок в зависимости от дня недели;
- зависимость медианной длительности поездки от дня недели.

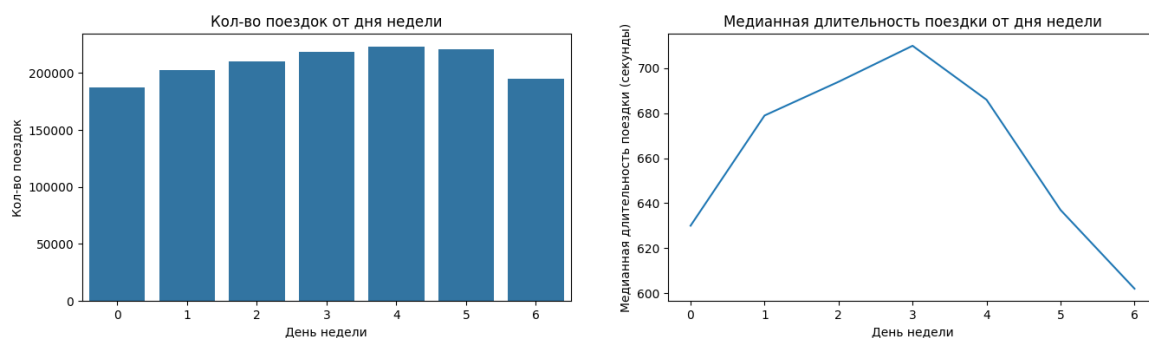


Рисунок 21 – количество поездок в зависимости от дня недели и зависимость медианной длительности поездки от дня недели

2.3 Отбор и преобразование признаков

Перед тем как перейти к построению модели, осталось сделать ещё несколько шагов:

- многие алгоритмы машинного обучения не могут обрабатывать категориальные признаки в их обычном виде. Поэтому нам необходимо их закодировать;
- надо отобрать признаки, которые мы будем использовать для обучения модели;
- необходимо масштабировать и трансформировать некоторые признаки для того, чтобы улучшить сходимость моделей, в основе которых лежат численные методы.

Ранее мы извлекли всю необходимую для нас информацию из даты начала поездки, теперь мы можем избавиться от этих признаков, так как они нам больше не понадобятся (см. рисунок 22).

```
drop_columns = ['pickup_datetime', 'pickup_date', 'dropoff_datetime', 'id']
train_data = train_data.drop(drop_columns, axis=1)
```

Рисунок 22 – удаление ненужных признаков

Теперь займёмся кодированием категориальных признаков. В наших данных к категориальным можно отнести следующие признаки:

- pickup_day_of_week;
- geo_cluster;
- events.

Так как признаки pickup_day_of_week, geo_cluster, events являются номинальными, для них отлично подойдёт однократное кодирование (см. рисунок 23).

```
from sklearn.preprocessing import OneHotEncoder

columns_to_change = ['pickup_day_of_week', 'geo_cluster', 'events']
one_hot_encoder = OneHotEncoder(drop='first')
data_onehot = one_hot_encoder.fit_transform(train_data[columns_to_change])

column_names = one_hot_encoder.get_feature_names_out()

data_onehot = pd.DataFrame(data_onehot.toarray(), columns=column_names)
```

Рисунок 23 – однократное кодирование признаков

Добавим полученную таблицу с закодированными признаками (см. рисунок 24).

```
train_data = pd.concat(
    [train_data.reset_index(drop=True).drop(columns=columns_to_change, axis=1), data_onehot],
    axis=1
)
```

Рисунок 24 – присоединение таблицы с закодированными признаками

Теперь, когда категориальные признаки предобработаны, сформируем матрицу наблюдений X, вектор целевой переменной y и его логарифм y_log. В матрицу наблюдений войдут все столбцы из таблицы с поездками за исключением целевого признака trip_duration и его логарифмированной версии trip_duration_log (см. рисунок 25).

```
x = train_data.drop(['trip_duration', 'trip_duration_log'], axis=1)
y_log = train_data['trip_duration_log']
```

Рисунок 25 – формируем матрицу наблюдений X и вектор целевой переменной y

Все наши модели мы будем обучать на логарифмированной версии `y_log`.

Далее нам необходимо определить тип валидации — схему, по которой мы будем оценивать качество построенных моделей. Так как наш набор данных довольно большой (почти 1.5 миллиона наблюдений), то мы свободно можем использовать hold-out-валидацию. Использование k-fold-валидации и тем более leave-one-out-валидации при таком объёме данных будет избыточным и времязатратным.

Разобьем выборку на обучающую и валидационную в соотношении 67/33 (см. рисунок 26).

```
x_train, x_valid, y_train_log, y_valid_log = model_selection.train_test_split(
    x, y_log,
    test_size=0.33,
    random_state=42
)
```

Рисунок 26 – разбиение выборки

На данный момент у нас достаточно много признаков: скорее всего, не все из них будут важны. Оставим лишь те, которые сильнее всего связаны с целевой переменной и точно будут вносить вклад в повышение качества модели.

С помощью SelectKBest отберем 25 признаков, наилучшим образом подходящих для предсказания целевой переменной в логарифмическом масштабе. В тренировочной и валидационной выборках оставим только полученные 25 признаков, откинув ненужные (см. рисунок 27). Далее мы будем работать и строить модели на отобранных признаках.

```
from sklearn.feature_selection import SelectKBest, f_regression

selector = SelectKBest(score_func=f_regression, k=25)
selector.fit(X_train, y_train_log)

best_features = selector.get_feature_names_out().tolist()

X_train = X_train[best_features]
X_valid = X_valid[best_features]
```

Рисунок 27 – отбор признаков

Так как мы будем использовать различные модели, в том числе внутри которых заложены численные методы оптимизации, то заранее позаботимся о масштабировании факторов.

Нормализуем значения в обучающей и валидационной выборках с помощью MinMaxScaler из библиотеки sklearn (см. рисунок 28).

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
scaler.fit(X_train)
X_train_scaled = pd.DataFrame(scaler.transform(X_train), columns=X_train.columns)
X_valid_scaled = pd.DataFrame(scaler.transform(X_valid), columns=X_valid.columns)
```

Рисунок 28 – нормализуем предикторы

2.4 Задача регрессии: линейная регрессия и деревья решений

Мы будем использовать метрику RMSLE (Root Mean Squared Log Error) для измерения качества наших моделей:

$$RMSLE = \sqrt{\frac{1}{n} \sum_{i=1}^n (\log(y_i + 1) - \log(\hat{y}_i + 1))^2},$$

где:

- y_i - истинная длительность i -ой поездки на такси (trip_duration);
- $\hat{y}_i$ - предсказанная моделью длительность i -ой поездки на такси.

Заметим, что логарифмирование целевого признака мы уже провели заранее, поэтому нам будет достаточно вычислить метрику RMSE для модели, обученной прогнозировать длительность поездки такси в логарифмическом масштабе:

$$z_i = \log(y_i + 1),$$
$$RMSLE = \sqrt{\frac{1}{n} \sum_{i=1}^n (z_i - \hat{z}_i)^2} = \sqrt{MSE(z_i, \hat{z}_i)},$$

где:

- z_i - истинная длительность i -ой поездки на такси в логарифмическом масштабе (trip_duration);
- $\hat{z}_i$ - предсказанная моделью длительность i -ой поездки на такси в логарифмическом масштабе.

Для начала воспользуемся самой простой моделью регрессии — классическая линейная регрессия по методу OLS(см. рисунок 29).

```
from sklearn.linear_model import LinearRegression

lin_reg = LinearRegression()
lin_reg.fit(X_train_scaled, y_train_log)
y_train_pred = lin_reg.predict(X_train_scaled)
y_valid_pred = lin_reg.predict(X_valid_scaled)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(2))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(2))
```

✓ 0.5s

Train RMSLE: 0.53
Valid RMSLE: 0.54

Рисунок 29 – построение линейной регрессии

Теперь попробуем уменьшить ошибку прогноза, воспользовавшись полиномиальной регрессией. Сгенерируем полиномиальные признаки 2-ой степени с помощью PolynomialFeatures из библиотеки sklearn (см. рисунок 30).

```
from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include_bias=False)
poly.fit(X_train_scaled)
X_train_poly = poly.transform(X_train_scaled)
X_valid_poly = poly.transform(X_valid_scaled)
```

Рисунок 30 – построение полиномиальных признаков

Воспользуемся моделью регрессии на полиномиальных признаках 2-ой степени (см. рисунок 31).

```
lin_reg = LinearRegression()
lin_reg.fit(X_train_poly, y_train_log)
y_train_pred = lin_reg.predict(X_train_poly)
y_valid_pred = lin_reg.predict(X_valid_poly)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(2))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(2))
✓ 30.0s

Train RMSLE: 0.47
Valid RMSLE: 0.61
```

Рисунок 31 – построение полиномиальной регрессии

Наблюдаются признаки переобучения в нашей модели. Чтобы исправить проблему переобучения, попробуем воспользоваться регуляризацией.

Построим модель полиномиальной регрессии 2-ой степени с L2-регуляризацией (регуляризация по Тихонову) (см. рисунок 32).

```
from sklearn.linear_model import Ridge

ridge = Ridge(alpha=1)
ridge.fit(X_train_poly, y_train_log)
y_train_pred = ridge.predict(X_train_poly)
y_valid_pred = ridge.predict(X_valid_poly)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(2))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(2))

Train RMSLE: 0.48
Valid RMSLE: 0.48
```

Рисунок 32 – построение модели полиномиальной регрессии 2-ой степени с L2-регуляризацией

Модель полиномиальной регрессии с L2-регуляризацией показывает лучший результат, чем классическая модель линейной регрессии по методу OLS.

Еще одной из простейших моделей, которой мы попробуем воспользоваться, будет дерево решений. Обучение дерева решений без

ограничения его глубины практически всегда приводит к переобучению.

Подберем оптимальную глубину дерева. (см. рисунок 33).

```
max_depths = range(7, 20)
train_scores = []
valid_scores = []
for depth in max_depths:
    dtr = DecisionTreeRegressor(max_depth=depth, random_state=42)
    dtr.fit(X_train_scaled, y_train_log)
    y_train_pred = dtr.predict(X_train_scaled)
    y_valid_pred = dtr.predict(X_valid_scaled)
    train_scores.append(np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)))
    valid_scores.append(np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)))
```

Рисунок 33 – подбор оптимальной глубины

Построим линейные графики изменения метрики RMSLE на тренировочной и валидационной выборках в зависимости от значения параметра глубины дерева решений(см. рисунок 34).

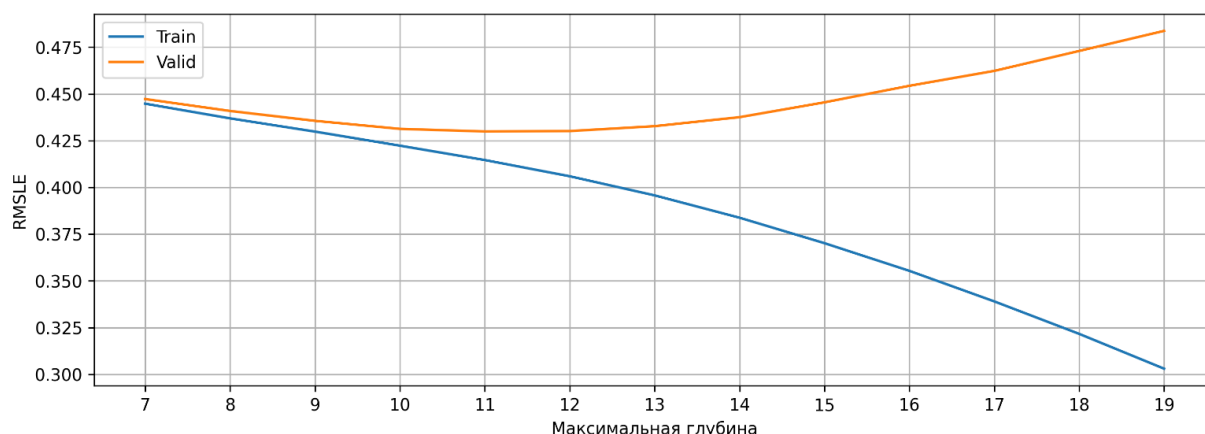


Рисунок 34 – RMLSE на тренировочной и валидационной выборках

Оптимальное значение максимальной глубины дерева, для которой будет наблюдаться минимальное значение RMSLE на обучающей выборке, но при этом ещё не будет наблюдаться переобучение (валидационная кривая ещё не начинает возрастать), соответствует $\text{max_depth} = 12$.

Обучим модель с оптимальным значением максимальной глубины дерева (см. рисунок 35).

```

dtr = DecisionTreeRegressor(max_depth=12, random_state=42)
dtr.fit(X_train_scaled, y_train_log)
y_train_pred = dtr.predict(X_train_scaled)
y_valid_pred = dtr.predict(X_valid_scaled)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(2))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(2))

```

```

Train RMSLE: 0.41
Valid RMSLE: 0.43

```

Рисунок 35 – построение модели дерева решений

Модель дерева решений с оптимальным значением глубины показало наилучший результат метрики RMSLE.

					ПГУ 1.090301.44.001 ПЗ	Лист
						44
Изм.	Лист	№ докум.	Подпись	Дата		

2.5 Решение задачи регрессии: ансамблевые методы

Начинаем со случайного леса (см. рисунок 36).

```
from sklearn.ensemble import RandomForestRegressor

rfr = RandomForestRegressor(
    n_estimators=200,
    max_depth=12,
    criterion='squared_error',
    min_samples_split=20,
    random_state=42,
    n_jobs=-1,
    verbose=True
)
rfr.fit(X_train_scaled, y_train_log)
y_train_pred = rfr.predict(X_train_scaled)
y_valid_pred = rfr.predict(X_valid_scaled)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(4))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(4))
```

```
[Parallel(n_jobs=-1)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=-1)]: Done 26 tasks      | elapsed: 32.0s
[Parallel(n_jobs=-1)]: Done 176 tasks    | elapsed: 2.8min
[Parallel(n_jobs=-1)]: Done 200 out of 200 | elapsed: 3.1min finished
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=12)]: Done 26 tasks     | elapsed: 0.2s
[Parallel(n_jobs=12)]: Done 176 tasks    | elapsed: 1.6s
[Parallel(n_jobs=12)]: Done 200 out of 200 | elapsed: 1.8s finished
[Parallel(n_jobs=12)]: Using backend ThreadingBackend with 12 concurrent workers.
[Parallel(n_jobs=12)]: Done 26 tasks     | elapsed: 0.1s
Train RMSLE: 0.3992
Valid RMSLE: 0.414
```

Рисунок 36 – обучение модели случайного леса

Благодаря параллельному обучению нескольких деревьев, модель случайного леса дала нам небольшой прирост качества по сравнению с обычным деревом решений.

Посмотрим, что будет при последовательном ансамблировании(см. рисунок 37).

```
from sklearn.ensemble import GradientBoostingRegressor

gbr = GradientBoostingRegressor(
    learning_rate=0.5,
    n_estimators=100,
    max_depth=6,
    min_samples_split=30,
    random_state=42,
    verbose=True
)
gbr.fit(X_train_scaled, y_train_log)
y_train_pred = gbr.predict(X_train_scaled)
y_valid_pred = gbr.predict(X_valid_scaled)

print("Train RMSLE:", np.sqrt(metrics.mean_squared_error(y_train_log, y_train_pred)).round(4))
print("Valid RMSLE:", np.sqrt(metrics.mean_squared_error(y_valid_log, y_valid_pred)).round(4))
```

Рисунок 37 – обучение модели последовательного ансамблирования

Посмотрим на точность предсказаний построенной модели на рисунке 38.

```
Train RMSLE: 0.3715
Valid RMSLE: 0.393
```

Рисунок 38 – RMSLE модели последовательного ансамблирования

					ПГУ 1.090301.44.001 ПЗ	Лист
						46
Изм.	Лист	№ докум.	Подпись	Дата		

Теперь посмотрим на важность факторов для наилучшей ансамблевой модели. Построим столбчатую диаграмму коэффициентов значимости каждого из факторов (см. рисунок 39).

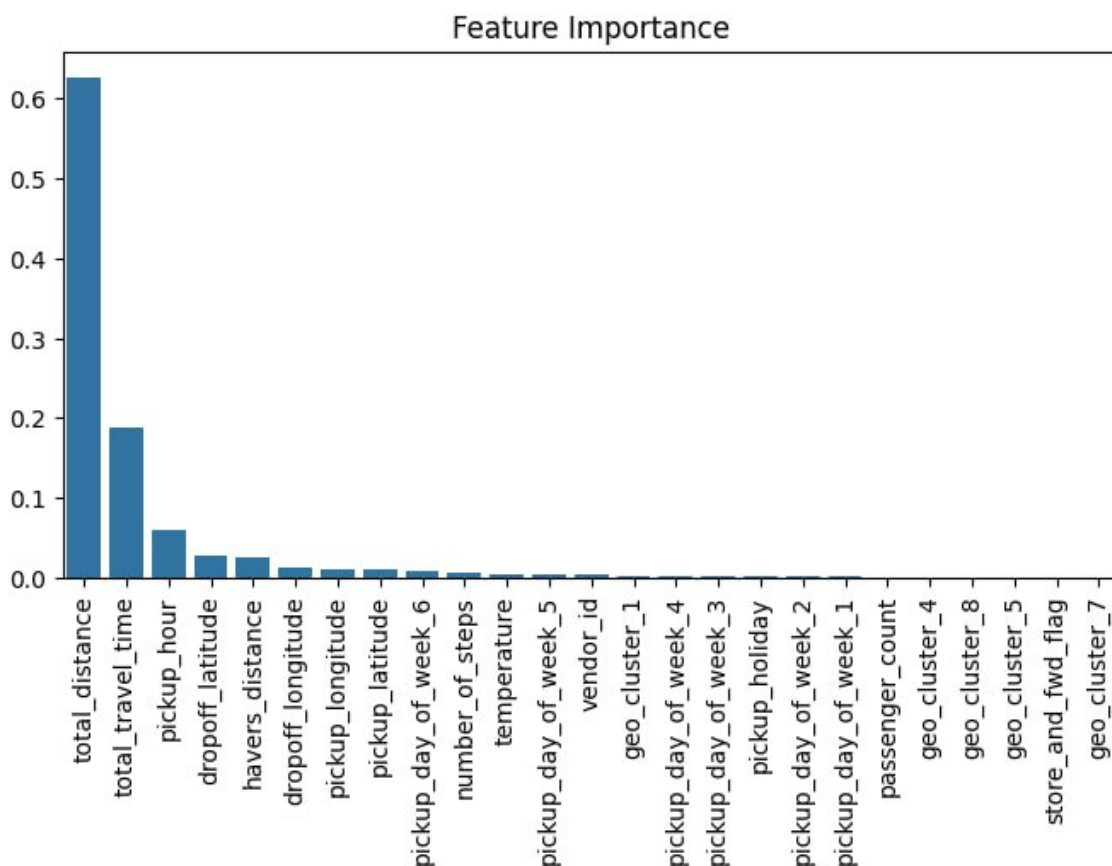


Рисунок 39 – столбчатая диаграмма коэффициентов значимости факторов

Топ-3 факторов, наиболее значимых для предсказания целевого признака (длительности поездки) в логарифмическом масштабе:

- total_distance;
- total_travel_time;
- pickup_hour.

Несмотря на все преимущества, у метрики RMSLE есть один недостаток — она не является интерпретируемой. Нам хотелось бы понять, насколько наша итоговая модель хорошо предсказывает целевой признак с точки зрения бизнес-

процесса. Точнее, мы бы хотели получить более понятное представление об ошибке прогноза длительности поездки.

Для лучшей из построенных моделей рассчитаем медианную абсолютную ошибку (MeAE - в sklearn функция `median_absolute_error`) предсказания длительности поездки такси на валидационной выборке (см. рисунок 40).

$$MeAE = median(|y_i - \hat{y}_i|)$$

```
# переводим значения из логарифмического масштаба в истинный
y_valid_exp = np.exp(y_valid_log) - 1
y_valid_pred_exp = np.exp(y_valid_pred) - 1

print(f"Valid MeAE: {(metrics.median_absolute_error(y_valid_exp, y_valid_pred_exp)/60).round(1)} mins")
```

Valid MeAE: 1.8 mins

Рисунок 40 – рассчитываем медианную абсолютную ошибку

Попробуем улучшить свое предсказание, воспользовавшись моделью экстремального градиентного бустинга (XGBoost) из библиотеки `xgboost`.

XGBoost - современная модель машинного обучения, которая является продолжением идеи градиентного бустинга Фридмана. У нее есть несколько преимуществ по сравнению с классической моделью градиентного бустинга из библиотеки `sklearn`:

- повышенная производительность путем параллелизации процесса обучения;
- повышенное качество решения за счет усовершенствования алгоритма бустинга;
- меньшая склонность к переобучению и широкий функционал возможности управления параметрами модели.

Перед обучением модели необходимо перевести наборы данных в тип данных `xgboost.DMatrix` (см. рисунок 41).

```
dtrain = xgb.DMatrix(X_train_scaled, label=y_train_log, feature_names=best_features)
dvalid = xgb.DMatrix(X_valid_scaled, label=y_valid_log, feature_names=best_features)
```

Рисунок 41 – меняем тип данных

Обучение модели XGBoost происходит с помощью метода `train`, в который необходимо передать параметры модели, набор данных, количество базовых моделей в ансамбле, а также дополнительные параметры (см. рисунок 42).

```
# Гиперпараметры модели
xgb_pars = {'min_child_weight': 20,
            'eta': 0.1,
            'colsample_bytree': 0.9,
            'max_depth': 6,
            'subsample': 0.9,
            'lambda': 1,
            'nthread': -1,
            'booster': 'gbtree',
            'eval_metric': 'rmse',
            'objective': 'reg:squarederror'
            }

# Тренировочная и валидационная выборки
watchlist = [(dtrain, 'train'), (dvalid, 'valid')]

# Обучаем модель XGBoost
model = xgb.train(
    params=xgb_pars, #гиперпараметры модели
    dtrain=dtrain, #обучающая выборка
    num_boost_round=300, #количество моделей в ансамбле
    evals=watchlist, #выборки, на которых считается матрица
    early_stopping_rounds=20, #ранняя остановка
    maximize=False, #смена поиска максимума на минимум
    verbose_eval=30 #шаг, через который происходит отображение метрик
)
```

Рисунок 42 – Обучение модели XGBoost

Посмотрим на RMSE на обученной модели на рисунке 43.

```
[299]  train-rmse:0.38366  valid-rmse:0.39415
```

Рисунок 43 – RMSE модели XGBoost

Наилучший результат (наименьшее значение RMSLE на валидационной выборке) среди всех моделей показал градиентный бустинг над деревьями решений.

Также как и все модели, основанные на использовании деревьев решений в качестве базовых моделей, XGBoost имеет возможность определения коэффициентов важности факторов. Посмотрим на них на рисунке 44.

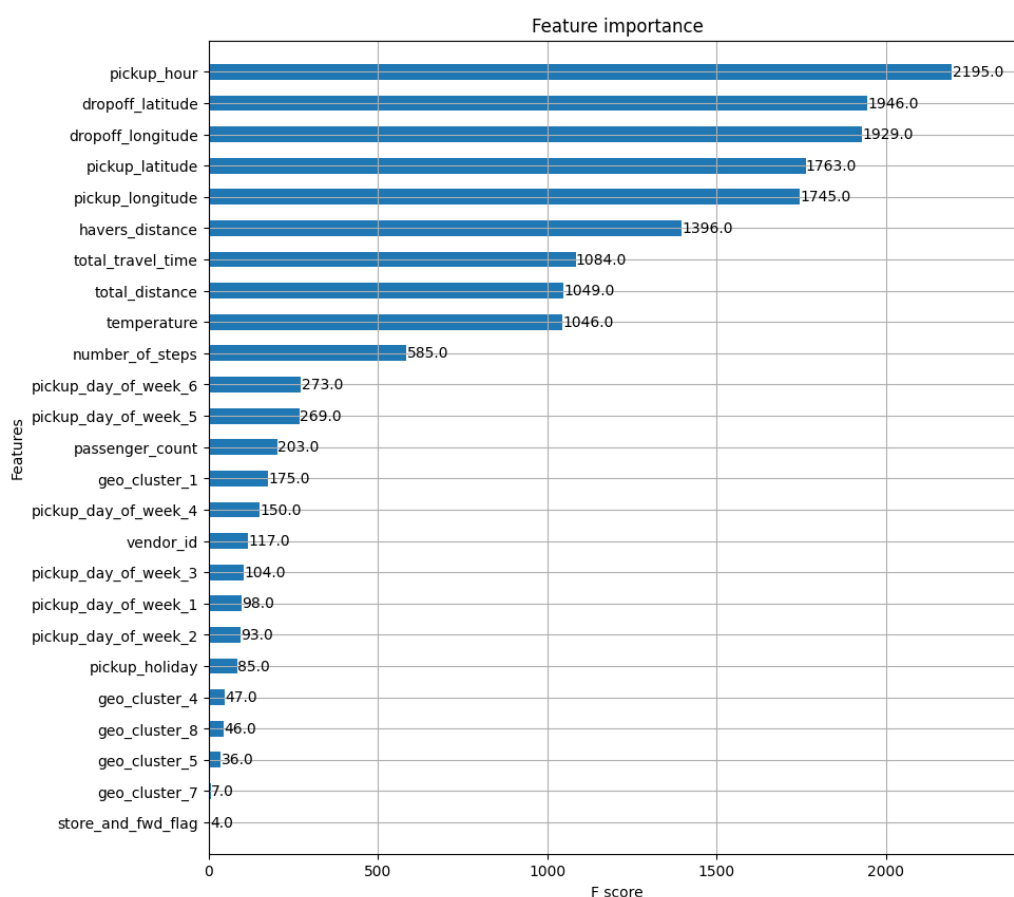


Рисунок 44 – значимость факторов модели XGBoost

Финальный шаг - предсказание для отложенного тестового набора данных.

Только после выполнения всех этих шагов можно сделать предсказание длительности поездки для тестовой выборки (см. рисунок 45). Не забудем перевести предсказания из логарифмического масштаба в истинный, используя формулу:

$$y_i = \exp(z_i) - 1.$$

```
# Делаем предсказание на тестовом наборе данных
dtest = xgb.DMatrix(X_test_scaled, feature_names=best_features)
y_test_predict = np.exp(model.predict(dtest)) - 1
print('Modeling RMSLE %.5f' % model.best_score)
```



```
Modeling RMSLE 0.39415
```

Рисунок 45 – RMSLE модели XGBoost

2.6 Инициализация и запуск сервиса

Перед тем как приступить к разработке системы предсказания времени поездки, необходимо подготовить окружение и настроить основные компоненты.

2.6.1. Настройка серверной части

Установка зависимостей:

```
pip install flask xgboost pandas numpy requests scikit-learn
joblib
```

Запуск Flask API, который будет обрабатывать запросы пользователей.

2.6.2 Подключение к OSRM API

OSRM (Open Source Routing Machine) используется для получения данных о маршрутах, включая расстояние, время поездки и количество шагов;

Запрос к API осуществляется через HTTP:

```
url = f"http://router.project-
osrm.org/route/v1/driving/{pickup_longitude},{pickup_latitude};{dr
opoff_longitude},{dropoff_latitude}?overview=full&steps=true&geome
tries=geojson"

response = requests.get(url).json()
```

2.6.3 Подключение к OpenWeather API. Получение текущих погодных условий в заданной точке:

```
weather_url =
f"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon={l
on}&appid={API_KEY}&units=metric"

weather_data = requests.get(weather_url).json()
```

2.6.4 Запуск модели машинного обучения. Загрузка предварительно обученной модели XGBoost:

```
model = xgb.Booster()
model.load_model("xgb_model.json")
```

Посмотрим на диаграмму классов на рисунке 46, демонстрирующую принцип работы сервиса.

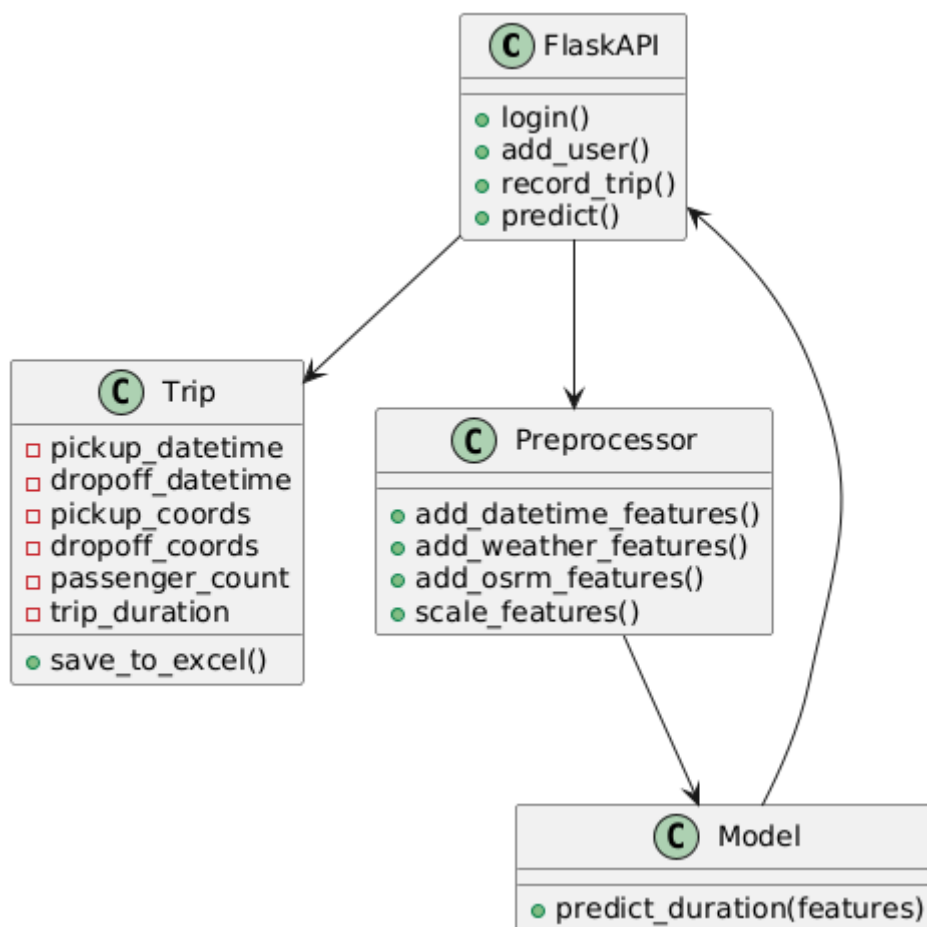


Рисунок 46 – Диаграмма классов

2.7 Описание доступных функций

Приложение реализует следующие функции:

– обработка входных данных;

1) принимает координаты начала и конца маршрута, количество пассажиров;

2) получает данные о маршруте через OSRM API;

3) запрашивает погодные условия из OpenWeather API;

– предобработка данных;

1) вычисляет расстояние и количество поворотов на маршруте;

2) кодирует категориальные признаки (день недели, погодные условия);

- 3) масштабирует числовые данные для подачи в модель;
 - прогнозирование времени поездки;
- 1) использует модель XGBoost для предсказания длительности поездки;
- 2) преобразует логарифмированное значение времени в исходную величину;
 - визуализация маршрута;
- 1) клиентская часть на Leaflet.js позволяет пользователю выбрать точки маршрута;
- 2) отображает на карте построенный маршрут и рассчитанное время поездки;
 - дообучение модели;
- 1) поддерживает загрузку новых данных для обновления модели;
- 2) позволяет сохранять улучшенную модель для дальнейшего использования.

2.8 Взаимодействие пользователя с системой

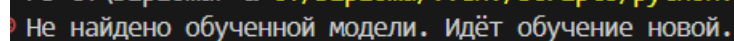
Последовательность работы программы:

- запуск веб-приложения;
- 1) пользователь открывает карту и выбирает начальную и конечную точку;
- 2) указывает количество пассажиров;
- 3) отправляет запрос на сервер;
 - обработка запроса сервером;
- 1) Flask API принимает входные данные;
- 2) запрашивает информацию о маршруте и погоде;
- 3) применяет модель машинного обучения;
 - получение результата;
- 1) сервер отправляет предсказанное время поездки;

2) веб-приложение отображает результат и маршрут на карте.

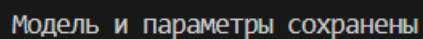
2.9 Запуск и демонстрация работы

Выводы в терминал сервера о состоянии программы, при самом первом запуске, показаны на рисунках 47-49.



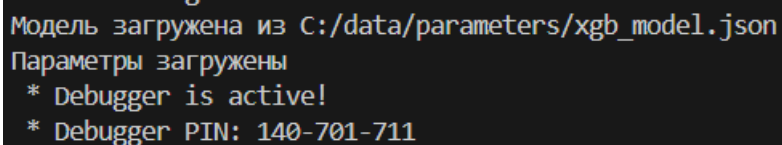
```
Не найдено обученной модели. Идёт обучение новой.
```

Рисунок 47 – Вывод в консоль при запуске программы при отсутствии ранее натренированной модели



```
Модель и параметры сохранены
```

Рисунок 48 – Вывод в консоль при запуске программы после обучения или дообучения модели



```
Модель загружена из C:/data/parameters/xgb_model.json
Параметры загружены
* Debugger is active!
* Debugger PIN: 140-701-711
```

Рисунок 49 – Вывод в консоль об успешном запуске программы

Запускаем Leaflet.html. На рисунке 50 показано, что в браузере открывается окно входа в систему.

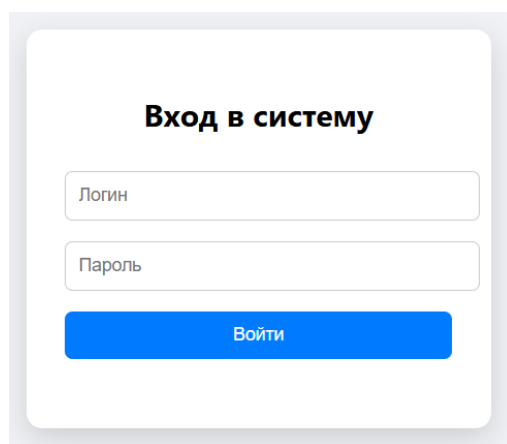
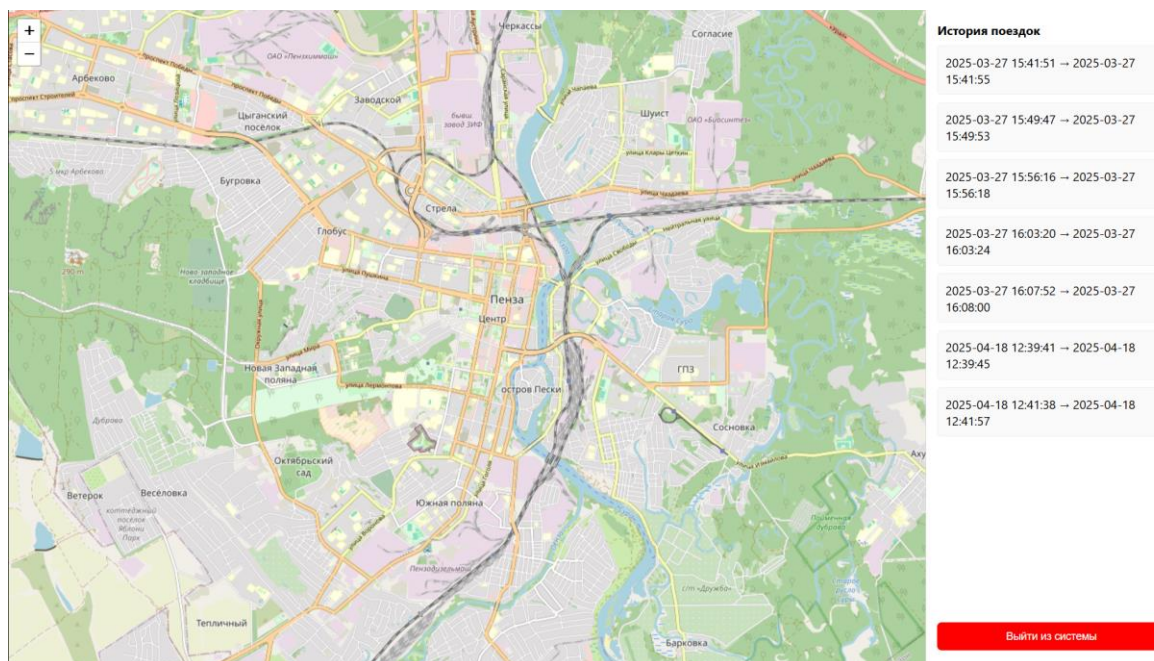


Рисунок 50 – открывается окно входа в систему

Входим с аккаунта обычного пользователя (см. рисунок 51). В истории поездок в окне справа мы можем найти нашу недавнюю поездку, посмотреть точки и маршрут на карте и увидеть округлённое время в пути.



История поездок				
2025-03-27 15:41:51 → 2025-03-27 15:41:55				
2025-03-27 15:49:47 → 2025-03-27 15:49:53				
2025-03-27 15:56:16 → 2025-03-27 15:56:18				
2025-03-27 16:03:20 → 2025-03-27 16:03:24				
2025-03-27 16:07:52 → 2025-03-27 16:08:00				
2025-04-18 12:39:41 → 2025-04-18 12:39:45				
2025-04-18 12:41:38 → 2025-04-18 12:41:57				

Рисунок 51 – Окно программы для обычного пользователя.

Ставим на карте 2 точки начала и конца маршрута. После указываем количество пассажиров (см. рисунок 52).

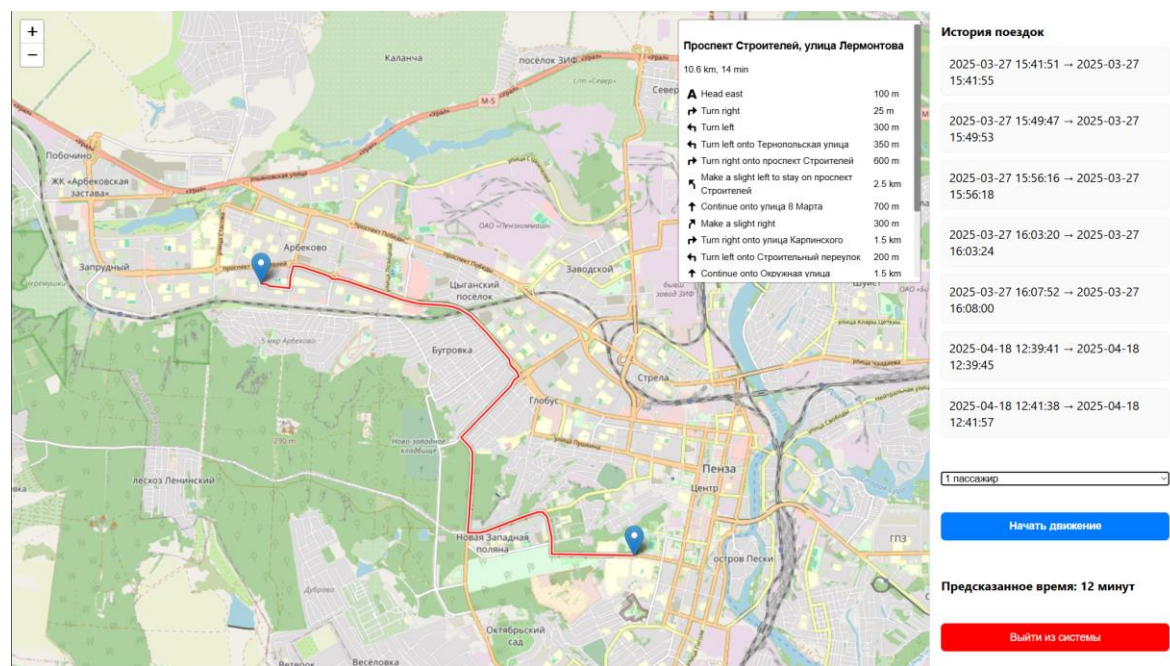


Рисунок 52 – поставили 2 метки на карте и количество пассажиров

Как можно заметить предсказанное время поездки не точно. Программа сейчас обучена на малом количестве поездок. Далее мы это исправим. Нажимаем на кнопку «Начать движение» (см. рисунок 53).

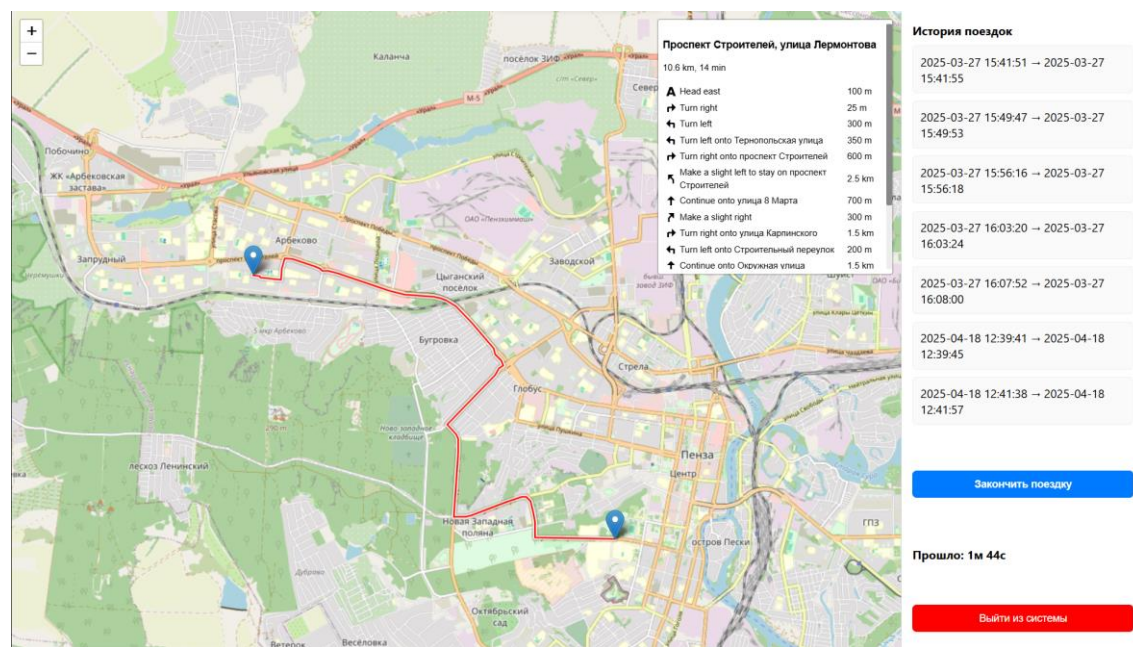


Рисунок 53 – нажали на кнопку «Начать движение»

Во время поездки считывается реальное время в пути. По завершении поездки вся информация о поездке записывается в excel таблицу вместе с реальным временем (см. рисунок 54).

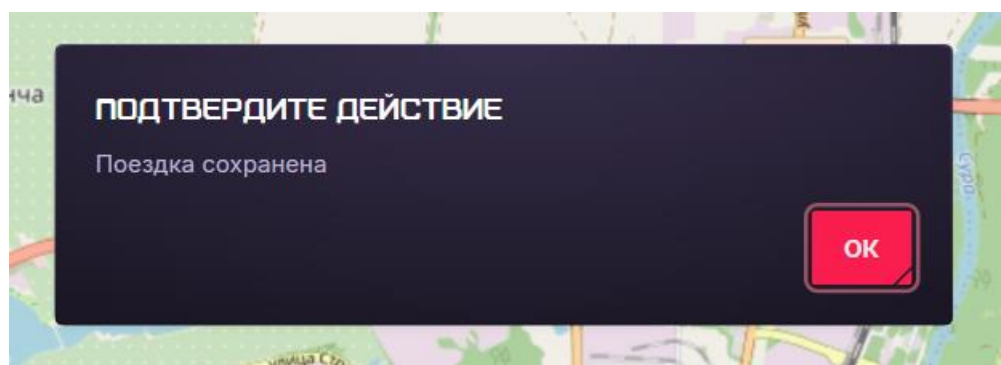


Рисунок 54 – уведомление о завершении поездки

При заходе в приложение от имени администратора мы можем добавлять и удалять обычных пользователей, но не таких же админов как и мы. Так же мы

можем дообучить модель на большем количестве данных, которые можно скопить, найти или приобрести. При нажатии на кнопку “Дообучить модель” программа не останавливает свою работу, а просто дообучает модель из файла, который указывается в коде (см. рисунок 55).

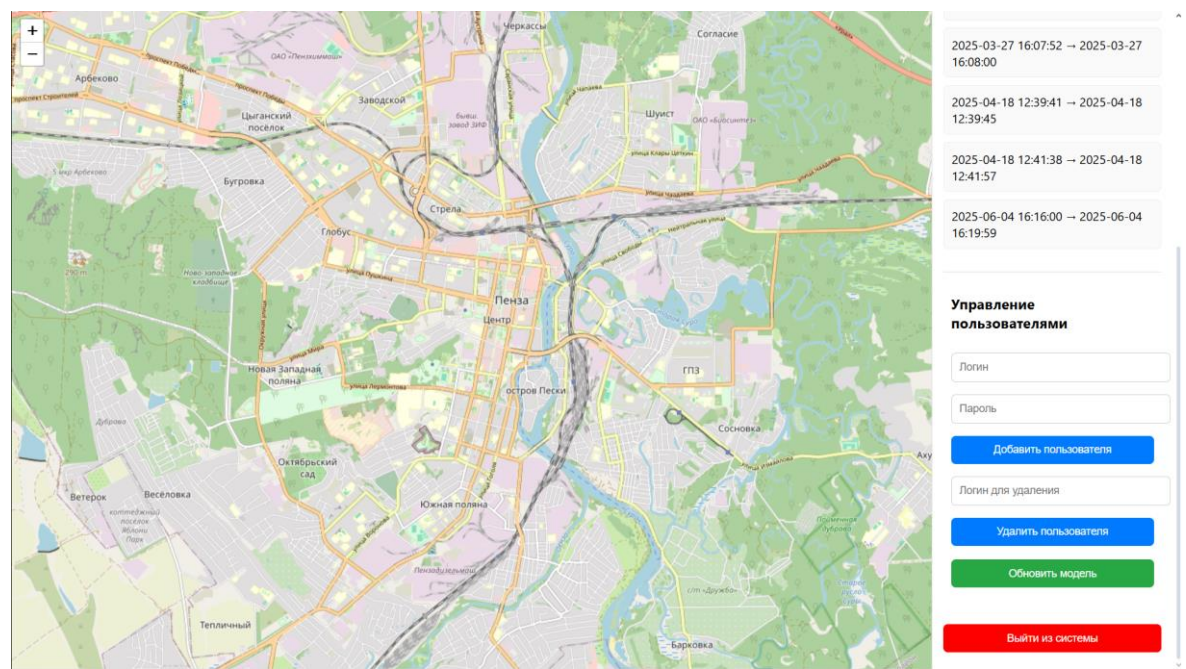


Рисунок 55 – окно администратора

Нажмем на кнопку «Дообучить модель». Результат работы показан на рисунке 56.

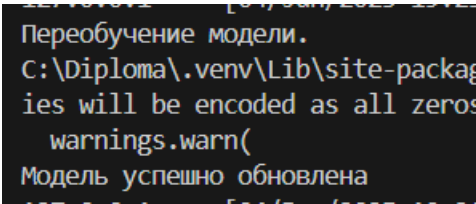


Рисунок 56 – вывод в консоли после дообучения модели

Дообученная модель как видно имеет более точные предсказания о времени в пути (см. рисунок 57).

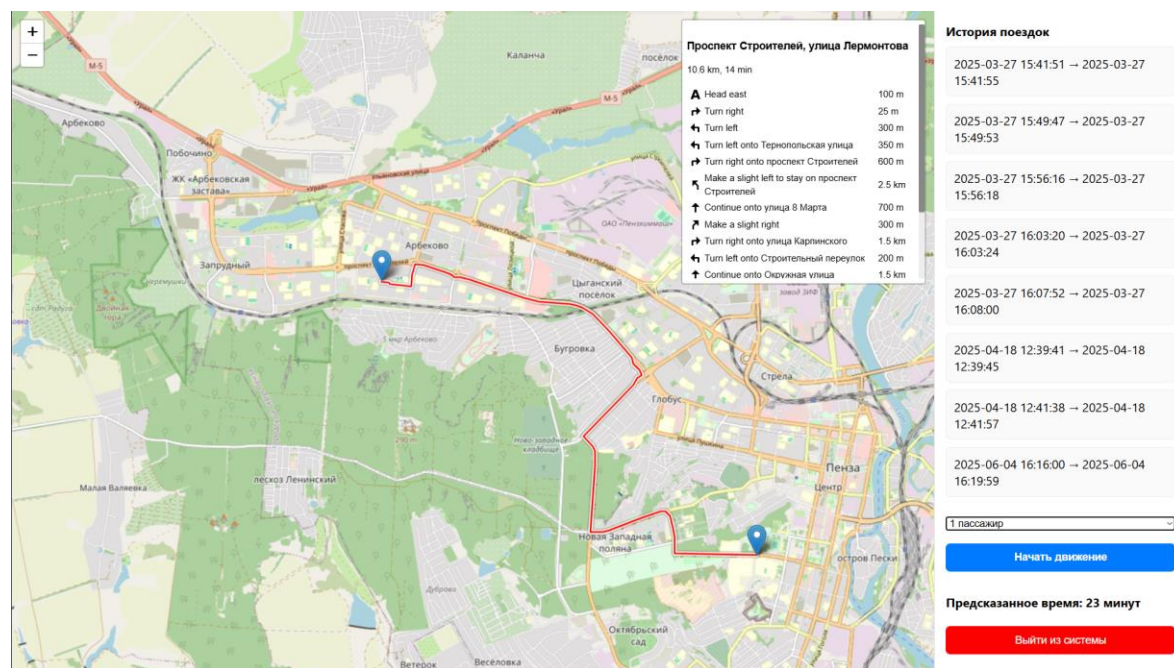


Рисунок 57 – Предсказанный маршрут и более точное время поездки

Созданные администраторам аккаунты пользователей добавляются в json файл и могут сразу начинать пользоваться приложением (см. рисунки 58-60).

Управление пользователями

new_user

.....|

Добавить пользователя

Рисунок 58 – добавление нового пользователя в окне администратора

```

{} users.json > ...
1  {
2      "admin": {
3          "password": "admin",
4          "role": "admin"
5      },
6      "pope": {
7          "password": "pope",
8          "role": "user"
9      }
10 }

```

Рисунок 59 – файл с логинами до добавления пользователя

```

{}
1  {
2      "admin": {
3          "password": "admin",
4          "role": "admin"
5      },
6      "pope": {
7          "password": "pope",
8          "role": "user"
9      },
10     "new_user": {
11         "password": "new_user",
12         "role": "user"
13     }
14 }

```

Рисунок 60 – файл с логинами после добавления пользователя

Информация о всех прошедших поездках сохраняется в файл, который можно использовать для улучшения прогнозов моделей (см. рисунок 61).

	A	B	C	D	E	F	G	H	I	J
	id	pickup_datetime	dropoff_datetime	passenger_count	pickup_longitude	pickup_latitude	dropoff_longitude	dropoff_latitude	trip_duration	username
1	id6710550	2025-03-27 15:17:	2025-03-27 15:17:	1	45,0202183	53,19329104	45,00511711	53,20244083	5	pepe
2	id7336538	2025-03-27 15:35:	2025-03-27 15:35:	1	44,89725788	53,22172232	44,97800418	53,18392208	1	pepe
3	id6864089	2025-03-27 15:41:	2025-03-27 15:41:	1	45,016443	53,19246849	45,00511711	53,19925405	3	pepe
4	id8358930	2025-03-27 15:49:	2025-03-27 15:49:	1	45,01712942	53,19246849	45,01009364	53,20079607	6	pepe
5	id5930834	2025-03-27 15:56:	2025-03-27 15:56:	1	45,0146234	53,19703775	45,00771632	53,20166389	1	pepe
6	id8983590	2025-03-27 16:03:	2025-03-27 16:03:	2	45,01936028	53,19298258	45,00975043	53,20007646	4	pepe
7	id5389674	2025-03-27 16:07:	2025-03-27 16:08:	2	45,01592819	53,19174875	45,00511711	53,19976806	7	pepe
8	id7888246	2025-04-18 12:39:	2025-04-18 12:39:	2	45,02090472	53,1915431	44,99241839	53,20151566	4	pepe
9	id5963010	2025-04-18 12:41:	2025-04-18 12:41:	2	45,0144513	53,1930023	44,99656154	53,20138118	19	pepe
10	id6325672	2025-06-04 16:16:	2025-06-04 16:19:	1	44,91564687	53,22030857	44,99980096	53,18353592	107	pepe

Рисунок 61 – файл с сохраненными поездками

3 Безопасность жизнедеятельности

3.1 Правила безопасности при работе с компьютером

Работа с компьютерными системами, в том числе с разработкой программного обеспечения, требует соблюдения основных правил безопасности, предусмотренных ГОСТ 12.0.003-2015. К основным рискам относятся:

- повышенная температура процессора и других компонентов компьютера;
- длительное пребывание в статичной позе, вызывающее физическую усталость;
- высокие зрительные нагрузки при работе с монитором;
- воздействие статического электричества;
- недостаточная освещенность рабочего пространства;
- воздействие электромагнитного излучения от мониторов и других устройств.

Для снижения рисков необходимо соблюдать следующие рекомендации:

- регулярно делать перерывы в работе, каждые 45-60 минут выполнять разминку;
- использовать эргономичную мебель и корректно настраивать высоту стола и кресла;
- соблюдать расстояние не менее 50 см от глаз до монитора;
- поддерживать уровень освещения в помещении, чтобы избежать усталости глаз;
- использовать заземление для предотвращения накопления статического электричества.

3.2 Безопасность работы с серверным оборудованием

При эксплуатации серверов и облачных систем важно учитывать следующие факторы:

- высокая нагрузка на серверное оборудование, требующая системы охлаждения;
- электромагнитные помехи, возникающие при одновременной работе нескольких устройств;
- возможные сбои электропитания, которые могут привести к потере данных;

Рекомендации по защите серверной инфраструктуры:

- использовать бесперебойные источники питания (ИБП) для предотвращения аварийных отключений;
- размещать серверы в хорошо вентилируемых помещениях для защиты от перегрева;

Регулярно обновлять программное обеспечение для обеспечения кибербезопасности.

3.3 Кибербезопасность при использовании веб-приложений

Разработка системы предсказания времени поездки требует защиты пользовательских данных и устойчивости к кибератакам. Основные угрозы:

- перехват и подмена запросов при использовании Flask API;
- возможные SQL-инъекции при хранении данных;
- угрозы DDoS-атак, перегружающих серверную инфраструктуру.

Рекомендации по обеспечению безопасности:

- использование HTTPS для шифрования передаваемых данных;
- ограничение доступа к API через токены аутентификации;
- регулярные проверки уязвимостей и мониторинг активности на сервере.

Заключение

Результатом выполнения данной выпускной квалификационной работы является успешно реализованная система предсказания времени поездки на такси на основе машинного обучения.

В ходе работы был проведен анализ и выбор наилучшего из методов прогнозирования, подключены новые источники данных и созданы новые признаки из уже имеющихся. Проведен выбор оптимальных инструментов для реализации системы, включая язык программирования Python, фреймворк Flask и Leaflet.js для визуализации маршрутов.

С использованием приведенных выше технологий была разработана система, обладающую следующим функционалом:

- получение координат маршрута, количества пассажиров;
- расчет расстояния, времени поездки и количества «шагов» на маршруте;
- учет погодных условий при прогнозировании;
- кодирование признаков, масштабирование числовых данных;
- использование обученной модели XGBoost;
- отображение на карте в Leaflet.js;
- возможность интеграции системы с внешними сервисами;
- обновление данных и возможность улучшения точности прогнозов.

Перспективы развития.

В дальнейшем система может быть расширена следующими возможностями:

- добавление анализа пробок для более точного предсказания;
- учтывание типа транспортного средства (такси, автобус, спецтехника);
- внедрение автоматического сбора новых данных для периодического обновления модели;
- интеграция с мобильными приложениями.

					ПГУ 1.090301.44.001 ПЗ	Лист
						65
Изм.	Лист	№ докум.	Подпись	Дата		

Разработанная система предсказания времени поездки готова к эксплуатации и может быть использована в сервисах такси, городских транспортных системах и логистике.

					ПГУ 1.090301.44.001 ПЗ	Лист
						66
Изм.	Лист	№ докум.	Подпись	Дата		

Список использованных источников

1. Zhang, Y. Travel Time Prediction Using Machine Learning: A Review / Y. Zhang, S. Wang // IEEE Access. – 2020. – Т. 8. – С. 194754–194766.

2. Chen, T. XGBoost: A Scalable Tree Boosting System / T. Chen, C. Guestrin // Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. – New York : Association for Computing Machinery, 2020. – С. 785–794.

3. Использование машинного обучения для предсказания времени прибытия в сервисах каршеринга [Электронный ресурс] / Uber Engineering. – URL: <https://eng.uber.com/eta-prediction-machine-learning/> (дата обращения: 05.05.2025).

4. Flask – легковесный веб-фреймворк на Python для разработки API [Электронный ресурс] – URL: <https://flask.palletsprojects.com/en/3.0.x/> (дата обращения: 05.05.2025).

5. Moovit Inc. Приложение с прогнозированием времени в городском транспорте [Электронный ресурс] – URL: <https://www.tranzmate.com/moovit-alternative> (дата обращения: 05.05.2025).

6. MIT CSAIL. Улучшение эффективности каршеринга с помощью машинного обучения [Электронный ресурс] / Massachusetts Institute of Technology – URL: <https://news.mit.edu/2022/ride-sharing-machine-learning-0720> (дата обращения: 05.05.2025).

Приложение А

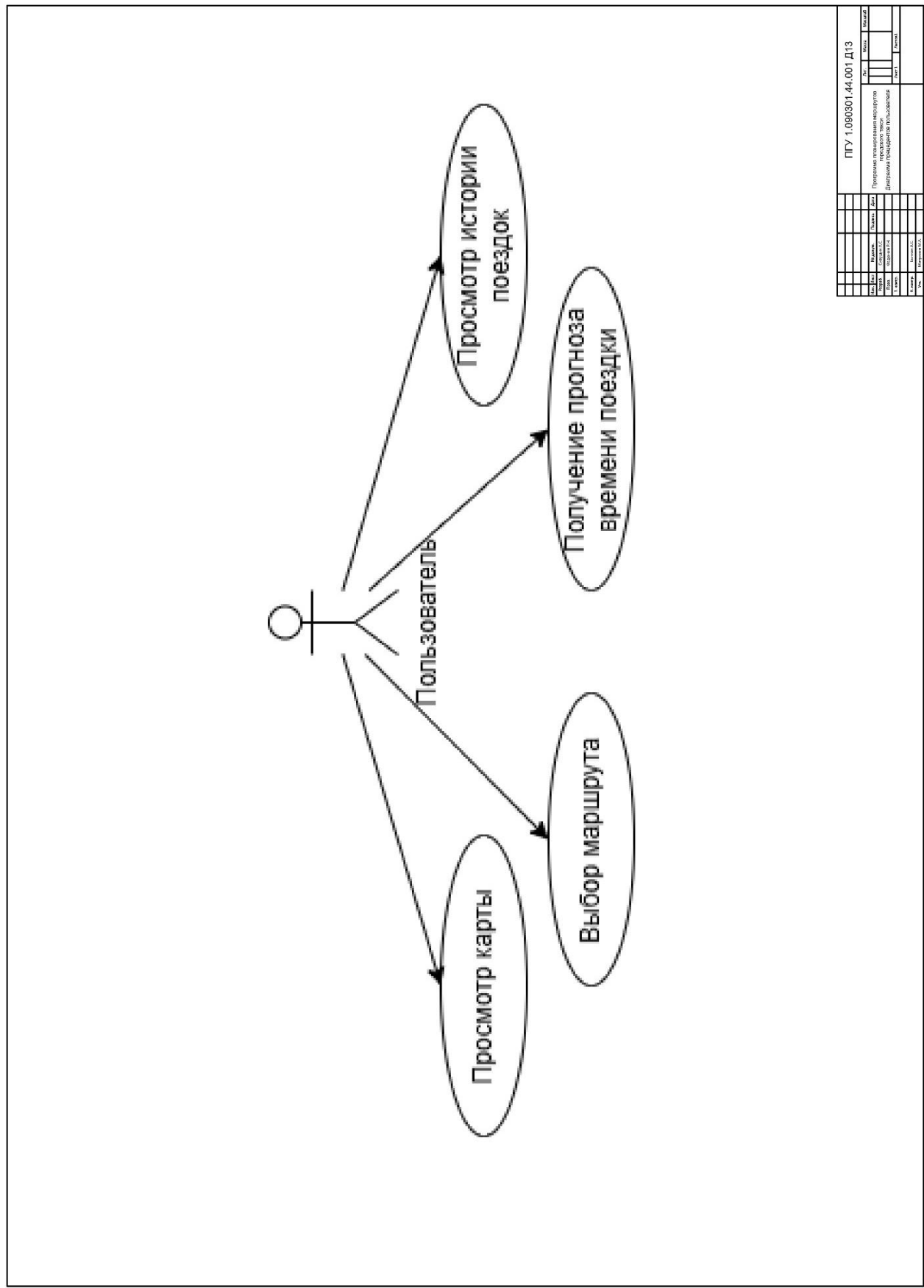
(Обязательное)

Программа планирования маршрутов городского такси

UML-диаграммы

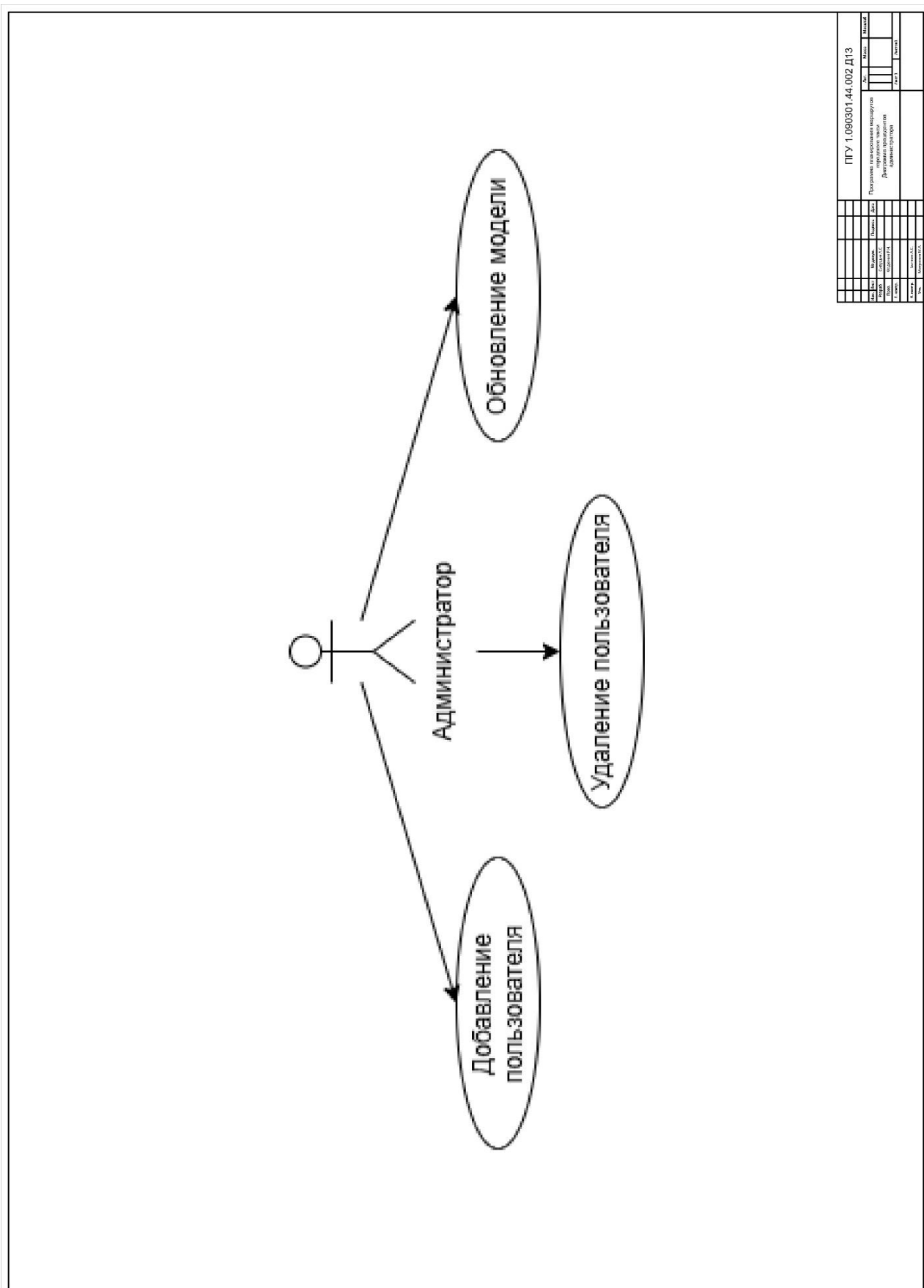
					ПГУ 1.090301.44.001 ПЗ	Лист
						68
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение А.1 - Диаграмма прецедентов пользователя



ПГУ 1.090301.44.001 Д13									
№	Имя	Дата	Стр.	Всего	Получено, просмотрено, изменено, удалено, возвращено				
1	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
2	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
3	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
4	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
5	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
6	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
7	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
8	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
9	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				
10	Создан	2020.01.14	1	1	Диаграмма прецедентов пользователя				

Приложение А.2 - Диаграмма прецедентов администратора



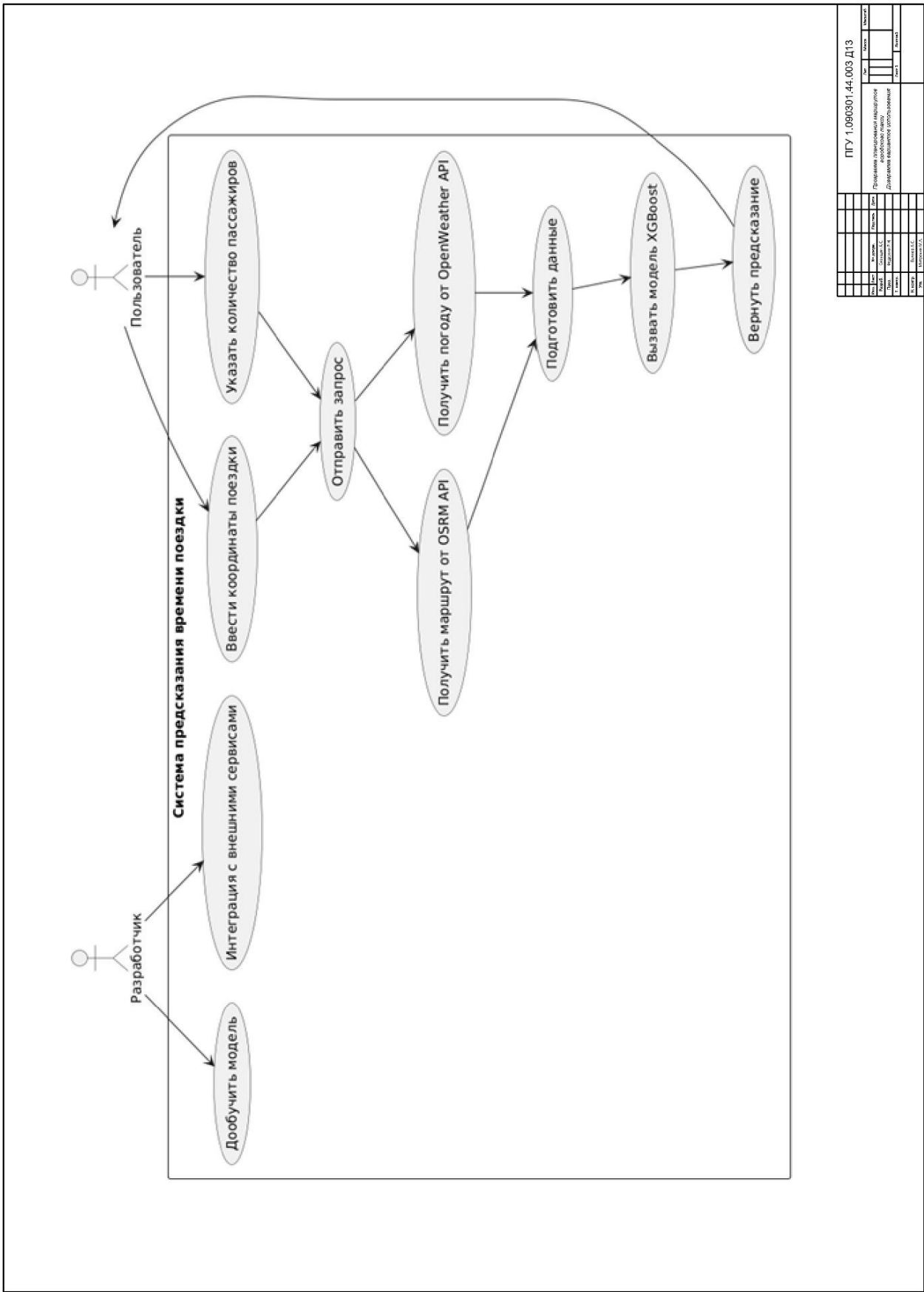
Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

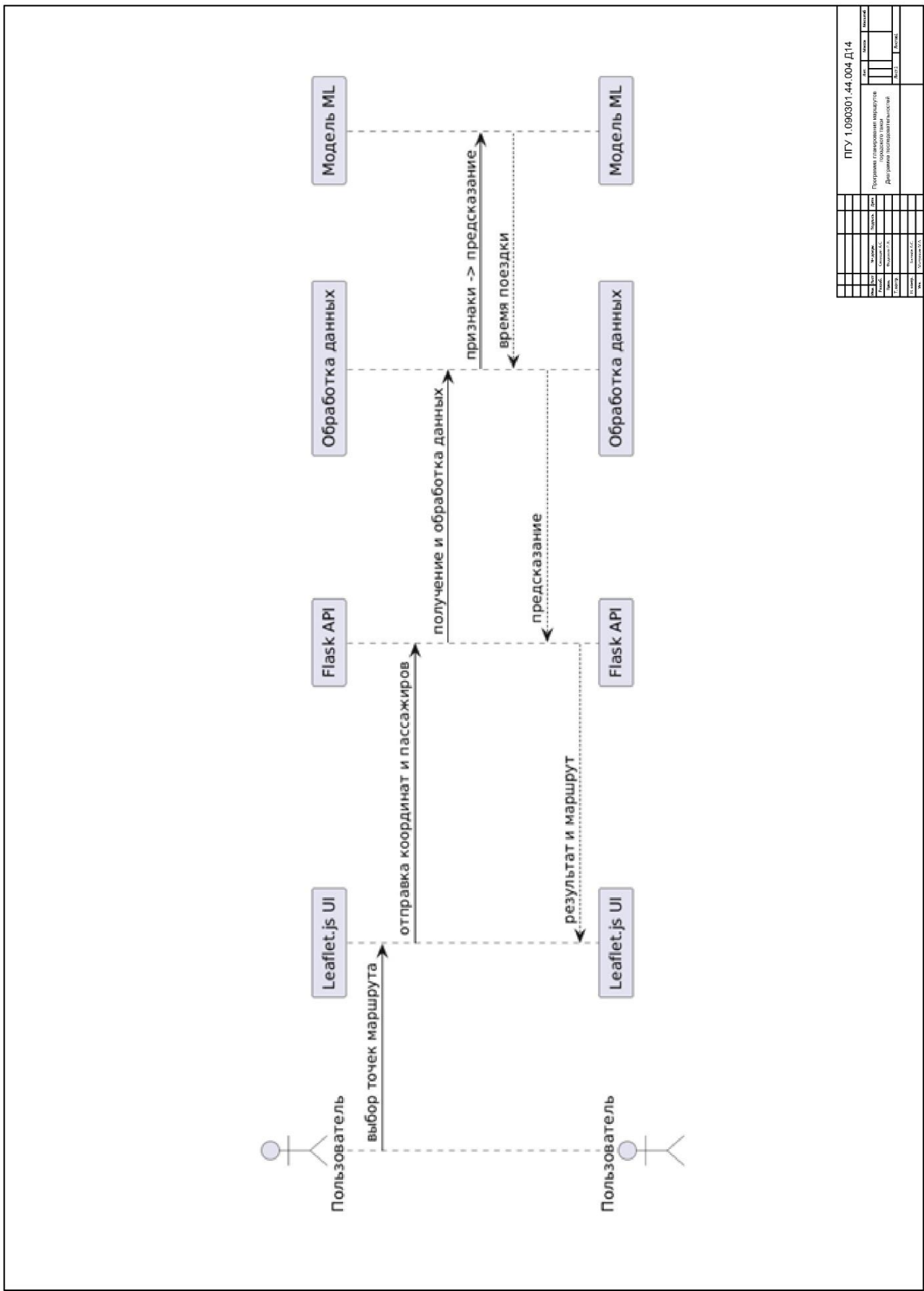
Лист

70

Приложение А.3 - Диаграмма вариантов использования



Приложение А.4 - Диаграмма последовательностей

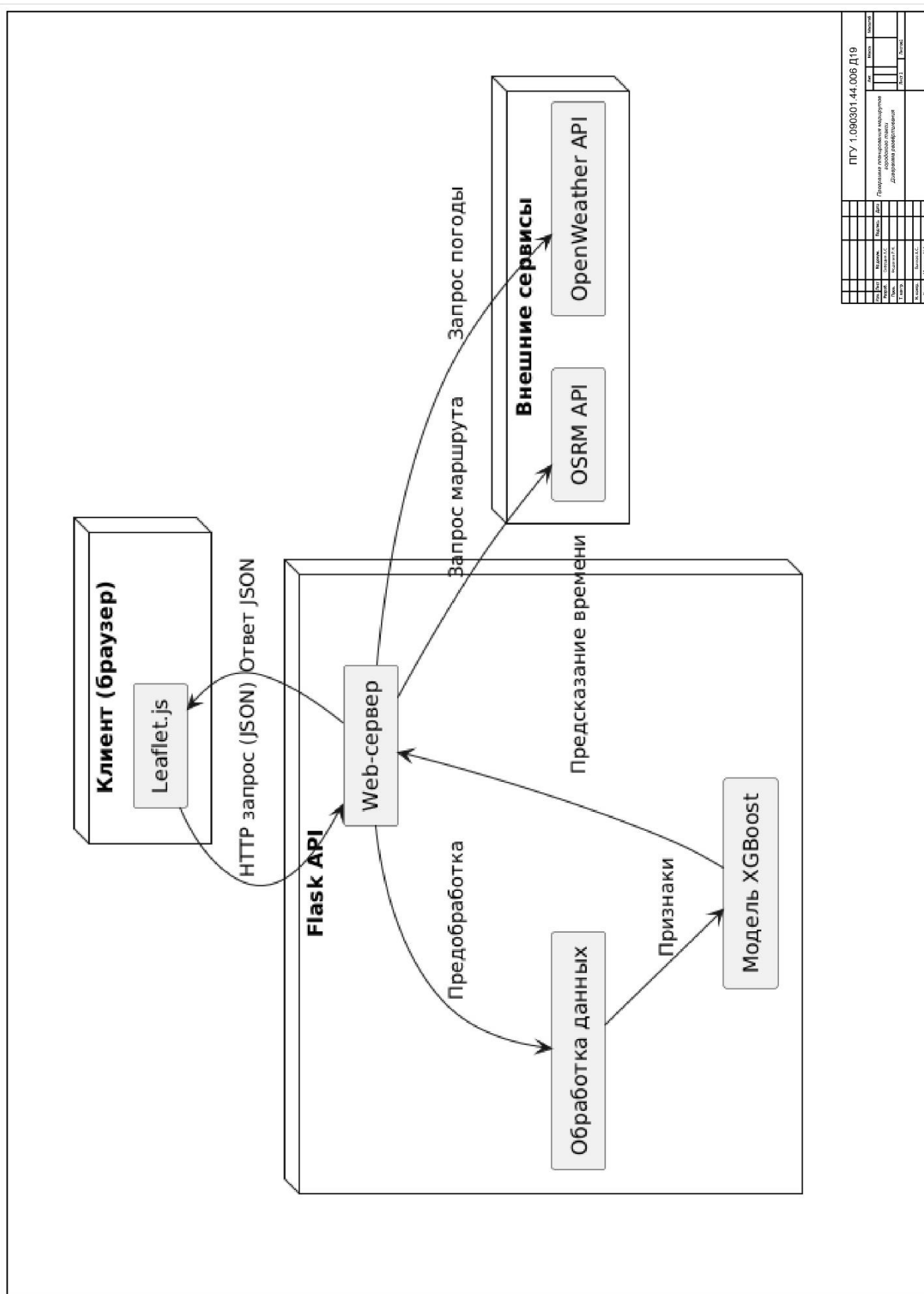


ПГУ 1.090301.44.004 Д14									
№	№	№	№	№	№	№	№	№	№
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10
1	2	3	4	5	6	7	8	9	10

Приложение А.5 - Диаграмма классов

[illegible]

Приложение А.6 - Диаграмма развёртывания



Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

74

Приложение Б

(обязательное)

Разработка системы предсказания времени поездки на такси на основе
машинного обучения

Листинг программы

Файл «app.py»

```
from flask import Flask, request, jsonify
from flask_cors import CORS, cross_origin
import pandas as pd
import numpy as np
import xgboost as xgb
from sklearn.preprocessing import MinMaxScaler, OneHotEncoder
from sklearn.feature_selection import SelectKBest, f_regression
from sklearn import cluster
import os
import joblib
import json
import warnings
import requests
import gc
import chardet
import random
from datetime import datetime
import openpyxl
from openpyxl import Workbook

# Подавление предупреждений от joblib
warnings.filterwarnings("ignore", category=UserWarning,
module="joblib.externals.loky.backend.context")

gc.collect()

# Инициализация Flask приложения
app = Flask(__name__)
CORS(app, resources={r"/predict": {"origins": "*"}})

# Глобальные переменные для инициализации
model = None
scaler = None
best_features = None
one_hot_encoder = None
columns_to_change = ['pickup_day_of_week', 'geo_cluster',
'events']
kmeans = None
```

```

USERS_FILE = 'users.json'
TRIP_FILE = 'C:/data/trips.xlsx'

def load_users():
    if not os.path.exists(USERS_FILE):
        with open(USERS_FILE, 'w', encoding='utf-8') as f:
            json.dump({
                "admin": {"password": "admin", "role": "admin"}
            }, f, indent=4)
    with open(USERS_FILE, 'r', encoding='utf-8') as f:
        return json.load(f)

def save_users(users):
    with open(USERS_FILE, 'w', encoding='utf-8') as f:
        json.dump(users, f, indent=4)

users = load_users()

# === AUTH ===
@app.route('/login', methods=['POST'])
@cross_origin()
def login():
    data = request.json
    username = data.get('username')
    password = data.get('password')

    with open('users.json', 'r', encoding='utf-8') as file:
        users = json.load(file)

    if username in users and users[username]["password"] == password:
        return jsonify({"status": "ok", "role": users[username]["role"]})

    return jsonify({"message": "Неверный логин или пароль"}), 401

@app.route('/add_user', methods=['POST'])
@cross_origin()
def add_user():
    try:
        data = request.json
        username = data.get('username')
        password = data.get('password')

        if not username or not password:
            return jsonify({'message': 'Логин и пароль обязательны!'}), 400

```

```

        with open('users.json', 'r+', encoding='utf-8') as file:
            users = json.load(file)
            if username in users:
                return jsonify({'message': 'Пользователь уже
существует!'}), 400
            users[username] = {"password": password, "role":
"user"}

            file.seek(0)
            json.dump(users, file, indent=4)
            file.truncate()

        print(f"Пользователь {username} добавлен!") # Лог в
консоль

        return jsonify({'message': 'Пользователь успешно
добавлен!'})

    except Exception as e:
        print(f"Ошибка при добавлении пользователя: {e}")
        return jsonify({'message': f'Ошибка: {str(e)}'}), 500

@app.route('/delete_user', methods=['POST'])
@cross_origin()
def delete_user():
    data = request.get_json()
    username = data['username']
    if username == 'admin':
        return jsonify({'status': 'error', 'message': 'Нельзя
удалить администратора'}), 400
    if username not in users:
        return jsonify({'status': 'error', 'message':
'Пользователь не найден'}), 404
    users.pop(username)
    save_users(users)
    return jsonify({'status': 'ok', 'message': f'Пользователь
{username} удалён'})

def save_trip_to_excel(data):
    # Если файл не существует – создаем с заголовками
    if not os.path.exists(TRIP_FILE):
        wb = Workbook()
        ws = wb.active
        ws.title = "Trips"
        headers = [
            'id', 'pickup_datetime', 'dropoff_datetime',
'passenger_count',
            'pickup_longitude', 'pickup_latitude',
'dropoff_longitude',
            'dropoff_latitude', 'trip_duration', 'username'
        ]

```

```

        ws.append(headers)
        wb.save(TRIP_FILE)

    # Загружаем существующую таблицу
    df = pd.read_excel(TRIP_FILE)

    #-----ЛОГИРОВАНИЕ-----
    print("Загружено строк из Excel:", len(df))
    print("Входные даты:", data['pickup_datetime'],
          data['dropoff_datetime'])
    try:
        pickup_time = datetime.strptime(data['pickup_datetime'],
                                         "%Y-%m-%d %H:%M:%S")
        dropoff_time =
datetime.strptime(data['dropoff_datetime'], "%Y-%m-%d %H:%M:%S")
    except ValueError as e:
        print("Ошибка в формате даты:", e)
        raise

    #-----

    # Генерируем id
    trip_id = f"id{random.randint(5000001, 9999999)}"
    pickup_time = datetime.strptime(data['pickup_datetime'],
                                     "%Y-%m-%d %H:%M:%S")
    dropoff_time = datetime.strptime(data['dropoff_datetime'],
                                     "%Y-%m-%d %H:%M:%S")
    trip_duration = int(data.get('actual_duration',
                                (dropoff_time - pickup_time).total_seconds()))

    row = {
        'id': trip_id,
        'pickup_datetime': data['pickup_datetime'],
        'dropoff_datetime': data['dropoff_datetime'],
        'passenger_count': data['passenger_count'],
        'pickup_longitude': data['pickup_longitude'],
        'pickup_latitude': data['pickup_latitude'],
        'dropoff_longitude': data['dropoff_longitude'],
        'dropoff_latitude': data['dropoff_latitude'],
        'trip_duration': trip_duration,
        'username': data['username']
    }

    # Добавляем строку и сохраняем
    df = pd.concat([df, pd.DataFrame([row])], ignore_index=True)
    df.to_excel(TRIP_FILE, index=False)

    return trip_id

@app.route('/record_trip', methods=['POST'])
@cross_origin()

```

```

def record_trip():
    try:
        data = request.get_json()
        print(" ♦ Данные для записи поездки:", data) # ЛОГ
        trip_id = save_trip_to_excel(data)
        print(f"Поездка сохранена с id: {trip_id}") # ЛОГ
        return jsonify({'status': 'ok', 'message': 'Поездка
записана', 'id': trip_id})
    except Exception as e:
        print("Ошибка при записи в Excel:", e)
        return jsonify({'status': 'error', 'message': str(e)}),
500

@app.route('/user_trips', methods=['POST'])
@cross_origin()
def get_user_trips():
    try:
        data = request.get_json()
        username = data['username']
        role = data.get('role', 'user')

        if not os.path.exists(TRIP_FILE):
            return jsonify({'trips': []})

        df = pd.read_excel(TRIP_FILE)
        if role == 'admin':
            trips = df.to_dict(orient='records')
        else:
            trips = df[df['username'] ==
username].to_dict(orient='records')

        return jsonify({'trips': trips})
    except Exception as e:
        print("Ошибка при получении истории поездок:", e)
        return jsonify({'status': 'error', 'message': str(e)}),
500

def detect_encoding(file_path):
    with open(file_path, 'rb') as f:
        raw_data = f.read(100000) # Читаем первые 100КБ
        result = chardet.detect(raw_data)
        return result['encoding']

def get_osrm_data(pickup_latitude, pickup_longitude,
dropoff_latitude, dropoff_longitude):
    """
    Получение данных из OSRM API.
    :param pickup_latitude: Широта точки подачи
    :param pickup_longitude: Долгота точки подачи
    :param dropoff_latitude: Широта точки назначения

```

```

        :param dropoff_longitude: Долгота точки назначения
        :return: Словарь с данными total_distance,
total_travel_time, number_of_steps
        """
        # URL OSRM API с параметром overview=full для получения
шагов
        url = f"http://router.project-
osrm.org/route/v1/driving/{pickup_longitude},{pickup_latitude};{
dropoff_longitude},{dropoff_latitude}?overview=full&steps=true&g
eometries=geojson&annotations=true"
        print(f"Получение OSRM даты из URL: {url}") # Логирование
URL для диагностики

    try:
        # Отправка GET-запроса к OSRM API
        response = requests.get(url)
        response.raise_for_status() # Проверка статуса ответа
        data = response.json()
        print("OSRM Ответ:", data) # Логирование ответа для
диагностики

        # Извлечение данных из ответа
        routes = data.get('routes', [])
        if not routes:
            raise ValueError("Пути не были найдены в OSRM
ответе")

        route = routes[0]
        total_distance = float(route.get('distance', 0)) #
Расстояние в метрах
        total_travel_time = float(route.get('duration', 0)) #
Время в секундах

        # Получение количества шагов
        legs = route.get('legs', [])
        print("OSRM Legs:", legs)
        if legs:
            steps = legs[0].get('steps', [])
            number_of_steps = len(steps)
        else:
            number_of_steps = 0

        return {
            'total_distance': total_distance,
            'total_travel_time': total_travel_time,
            'number_of_steps': number_of_steps
        }

    except Exception as e:
        print(f"Ошибка при получении OSRM информации: {e}")

```

```

        return {
            'total_distance': 0.0,
            'total_travel_time': 0.0,
            'number_of_steps': 0.0
        }

# Функция для добавления временных признаков
def add_datetime_features(df):
    df['pickup_date'] = df['pickup_datetime'].dt.date
    df['pickup_hour'] = df['pickup_datetime'].dt.hour
    df['pickup_day_of_week'] =
df['pickup_datetime'].dt.dayofweek
    return df

# Функция для добавления праздничных дней
def add_holiday_features(df1, df2):
    df2['date'] = pd.to_datetime(df2['date']).dt.date
    lst = df2['date'].tolist()
    df1['pickup_holiday'] = df1['pickup_date'].apply(lambda x: 1
if x in lst else 0)
    return df1

# Функция для добавления OSRM-признаков
def add_osrm_features(df1, df2):
    df1 = df1.merge(df2[['id', 'total_distance',
'total_travel_time', 'number_of_steps']], left_on='id',
right_on='id', how='left')
    return df1

# Функция для расчета расстояния Хаверсина
def get_haversine_distance(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2,
lng2))
    EARTH_RADIUS = 6371
    lat_delta = lat2 - lat1
    lng_delta = lng2 - lng1
    d = np.sin(lat_delta * 0.5) ** 2 + np.cos(lat1) *
np.cos(lat2) * np.sin(lng_delta * 0.5) ** 2
    h = 2 * EARTH_RADIUS * np.arcsin(np.sqrt(d))
    return h

# Функция для расчета угла направления
def get_angle_direction(lat1, lng1, lat2, lng2):
    lat1, lng1, lat2, lng2 = map(np.radians, (lat1, lng1, lat2,
lng2))
    lng_delta_rad = lng2 - lng1
    y = np.sin(lng_delta_rad) * np.cos(lat2)
    x = np.cos(lat1) * np.sin(lat2) - np.sin(lat1) *
np.cos(lat2) * np.cos(lng_delta_rad)
    alpha = np.degrees(np.arctan2(y, x))

```

```

    return alpha

# Функция для добавления географических признаков
def add_geographical_features(df):
    df['havers_distance'] =
get_haversine_distance(df['pickup_latitude'],
df['pickup_longitude'], df['dropoff_latitude'],
df['dropoff_longitude'])
    df['direction'] = get_angle_direction(df['pickup_latitude'],
df['pickup_longitude'], df['dropoff_latitude'],
df['dropoff_longitude'])
    return df

# Функция для добавления кластеров
def add_cluster_features(df, kmeans):
    coords = np.hstack((df[['pickup_latitude',
'pickup_longitude']],
                        df[['dropoff_latitude',
'dropoff_longitude']]))
    df['geo_cluster'] = kmeans.predict(coords)
    return df

import requests

def get_weather_data(lat, lon):
    """
    Получение текущей температуры и погодных условий в указанном
    месте с использованием API OpenWeatherMap.
    """
    API_KEY = "aa0cd1763c23475038222eb954fe1af"
    URL =
f"https://api.openweathermap.org/data/2.5/weather?lat={lat}&lon=
{lon}&appid={API_KEY}&units=metric"

    try:
        response = requests.get(URL)
        response.raise_for_status()
        data = response.json()

        temperature = data.get('main', {}).get('temp', 0.0) #
Значение по умолчанию - 0.0
        condition = data.get('weather', [{}])[0].get('main',
'').lower()

        event_mapping = {
            'rain': 'Rain',
            'drizzle': 'Rain',
            'thunderstorm': 'Rain',
            'snow': 'Snow',
            'mist': 'Fog',

```

```

        'fog': 'Fog',
        'haze': 'Fog',
        'smoke': 'Fog',
    }

    event = event_mapping.get(condition, 'None') # Если нет
совпадения, ставим 'None'

    print(f"Информация о погоде: Температура =
{temperature}, Событие = {event}")
    return temperature, event

except requests.exceptions.RequestException as e:
    print(f"Ошибка при получении информации о погоде: {e}")
    return 0.0, 'None' # Возвращаем безопасные значения

# Функция для добавления погодных данных
def add_weather_features(df1, df2):
    df2['date'] = pd.to_datetime(df2['time']).dt.date
    df2['hour'] = pd.to_datetime(df2['time']).dt.hour
    df1 = df1.merge(df2, how='left', left_on=['pickup_date',
'pickup_hour'], right_on=['date', 'hour'])
    df1.drop(columns=['time', 'windchill', 'heat index',
'humidity', 'pressure', 'dew Point', 'wind dir', 'gust speed',
'conditions', 'date', 'hour'], axis=1, inplace=True)
    return df1

# Функция для заполнения пропущенных значений
def fill_null_weather_data(df):
    col1 = ['temperature', 'wind speed', 'precip']
    df[col1] =
df[col1].fillna(df.groupby('pickup_date')[col1].transform('media
n'))
    df['events'] = df['events'].fillna('None')
    col2 = ['total_distance', 'total_travel_time',
'number_of_steps']
    df[col2] = df[col2].fillna(df[col2].median())
    return df

# Функция инициализации (выполняется один раз при запуске
сервера)
import os
import xgboost as xgb

import os
import joblib
import xgboost as xgb

def initialize():
    global model, scaler, best_features, one_hot_encoder, kmeans

```

```

model_path = "C:/data/parameters/xgb_model.json"
scaler_path = "C:/data/parameters/scaler.pkl"
encoder_path = "C:/data/parameters/one_hot_encoder.pkl"
kmeans_path = "C:/data/parameters/kmeans.pkl"
features_path = "C:/data/parameters/best_features.pkl"

# 1 Проверяем, есть ли сохраненная модель и параметры
if os.path.exists(model_path) and
os.path.exists(scaler_path) and os.path.exists(encoder_path) and
os.path.exists(kmeans_path) and os.path.exists(features_path):
    try:
        # Загружаем модель XGBoost
        model = xgb.Booster()
        model.load_model(model_path)
        print(f"Можель загружена из {model_path}")

        # Загружаем параметры
        scaler = joblib.load(scaler_path)
        one_hot_encoder = joblib.load(encoder_path)
        kmeans = joblib.load(kmeans_path)
        best_features = joblib.load(features_path)
        print(f"Параметры загружены")

        return # Выход из функции, модель загружена,
повторное обучение не требуется
    except Exception as e:
        print(f"Ошибка загрузки моделей или параметров: {e}.
Переобучение")

print("Не найдено обученной модели. Идёт обучение новой.")

# 2 Загружаем данные
taxi_data = pd.read_csv('C:/data/train.csv')
holiday_data = pd.read_csv('C:/data/holiday_data.csv',
sep=';')
osrm_data = pd.read_csv('C:/data/osrm_data_train.csv')
weather_data = pd.read_csv('C:/data/weather_data.csv')

weather_data = weather_data.drop('visibility', axis=1)
taxi_data = taxi_data.drop(['store_and_fwd_flag',
'vendor_id'], axis=1)

# 3 Обрабатываем данные
taxi_data['pickup_datetime'] =
pd.to_datetime(taxi_data['pickup_datetime'], format='%Y-%m-%d
%H:%M:%S')
taxi_data = add_datetime_features(taxi_data)
taxi_data = add_holiday_features(taxi_data, holiday_data)
taxi_data = add_osrm_features(taxi_data, osrm_data)

```

```

taxi_data = add_geographical_features(taxi_data)

coords = np.hstack((taxi_data[['pickup_latitude',
'pickup_longitude']],
                    taxi_data[['dropoff_latitude',
'dropoff_longitude']]))
kmeans = cluster.KMeans(n_clusters=10, n_init=10,
random_state=42)
kmeans.fit(coords)
taxi_data = add_cluster_features(taxi_data, kmeans)
taxi_data = add_weather_features(taxi_data, weather_data)
taxi_data = fill_null_weather_data(taxi_data)

# 4 Фильтрация данных
taxi_data = taxi_data[(taxi_data['trip_duration'] <= 24 * 60
* 60)]
avg_speed = taxi_data['total_distance'] /
taxi_data['trip_duration'] * 3.6
taxi_data = taxi_data[avg_speed <= 300]

# 5 Преобразование категориальных признаков
taxi_data['events'] = taxi_data['events'].astype('category')

# 6 EDA и выбор признаков
train_data = taxi_data.copy()
drop_columns = ['pickup_datetime', 'pickup_date',
'dropoff_datetime', 'id']
train_data = train_data.drop(drop_columns, axis=1)
train_data['trip_duration_log'] =
np.log(train_data['trip_duration'] + 1)
X = train_data.drop(['trip_duration', 'trip_duration_log'],
axis=1)
y_log = train_data['trip_duration_log']

# 7 OneHotEncoding
one_hot_encoder = OneHotEncoder(drop='first',
handle_unknown='ignore')
data_onehot =
one_hot_encoder.fit_transform(X[columns_to_change])
column_names =
one_hot_encoder.get_feature_names_out(columns_to_change)
data_onehot = pd.DataFrame(data_onehot.toarray(),
columns=column_names)
X =
pd.concat([X.reset_index(drop=True).drop(columns_to_change,
axis=1), data_onehot], axis=1)

# 8 Выбор лучших признаков
selector = SelectKBest(score_func=f_regression, k=25)
selector.fit(X, y_log)

```

```

        best_features = selector.get_feature_names_out().tolist()
        print(X[best_features].info()) # Логирование информации о
DataFrame

        # 9 Масштабирование
        scaler = MinMaxScaler()
        X_scaled =
pd.DataFrame(scaler.fit_transform(X[best_features]),
columns=best_features)

        # Обучение модели XGBoost
        dtrain = xgb.DMatrix(X_scaled, label=y_log,
feature_names=best_features)
        model = xgb.train(
            params={
                'min_child_weight': 20,
                'eta': 0.1,
                'colsample_bytree': 0.9,
                'max_depth': 6,
                'subsample': 0.9,
                'lambda': 1,
                'nthread': -1,
                'booster': 'gbtree',
                'eval_metric': 'rmse',
                'objective': 'reg:squarederror'
            },
            dtrain=dtrain,
            num_boost_round=50,
            verbose_eval=False
        )

        # Сохраняем модель и параметры
        model.save_model(model_path)
        joblib.dump(scaler, scaler_path)
        joblib.dump(one_hot_encoder, encoder_path)
        joblib.dump(kmeans, kmeans_path)
        joblib.dump(best_features, features_path)

        print(f"Модель и параметры сохранены")
        return jsonify({'status': 'ok'})

```

```

# Функция для препроцессинга входных данных
def preprocess_input(input_data):
    """
    Препроцессинг входных данных для предсказания.
    :param input_data: Словарь с входными данными
    (координатами).
    :return: Массив с признаками для предсказания.

```

```

"""
try:
    # Создание DataFrame из входных данных
    df = pd.DataFrame({
        'pickup_latitude': [input_data['pickup_latitude']],
        'pickup_longitude':
[input_data['pickup_longitude']],
        'dropoff_latitude':
[input_data['dropoff_latitude']],
        'dropoff_longitude':
[input_data['dropoff_longitude']],
        'passenger_count':
[input_data.get('passenger_count', 1)] # Значение по умолчанию
- 1
    })

    # Добавление временных признаков (примерно текущее
время)
    current_time = pd.Timestamp.now()
    df['pickup_datetime'] = current_time
    df = add_datetime_features(df)
    df['pickup_holiday'] = 0 # Предположим, что сегодня не
праздник

    # Добавление расстояния и направления
    df = add_geographical_features(df)

    # Получение данных из OSRM API
    osrm_data = get_osrm_data(
        df['pickup_latitude'].iloc[0],
        df['pickup_longitude'].iloc[0],
        df['dropoff_latitude'].iloc[0],
        df['dropoff_longitude'].iloc[0]
    )
    df['total_distance'] = osrm_data['total_distance']
    df['total_travel_time'] = osrm_data['total_travel_time']
    df['number_of_steps'] = osrm_data['number_of_steps']

    # Вывод количества шагов в терминал
    print(f"Total Distance: {df['total_distance'].iloc[0]}")
    print(f"Total Travel Time:
{df['total_travel_time'].iloc[0]}")
    print(f"Number of steps:
{df['number_of_steps'].iloc[0]}")

    # Кластеризация
    coords = np.hstack((df[['pickup_latitude',
'pickup_longitude']],
                        df[['dropoff_latitude',
'dropoff_longitude']]))

```

```

df['geo_cluster'] = kmeans.predict(coords)

# Погодные данные
df['temperature'], df['events'] = get_weather_data(
    df['pickup_latitude'].iloc[0],
df['pickup_longitude'].iloc[0]
)
df['wind speed'] = 5.0      # Примерная скорость ветра
df['precip'] = 0.0         # Без осадков

# Преобразование категориальных признаков
df['events'] = df['events'].astype('category')

# OneHotEncoding
data_onehot =
one_hot_encoder.transform(df[columns_to_change]).toarray()
column_names =
one_hot_encoder.get_feature_names_out(columns_to_change)
data_onehot = pd.DataFrame(data_onehot,
columns=column_names)
df =
pd.concat([df.reset_index(drop=True).drop(columns_to_change,
axis=1), data_onehot], axis=1)

print(f"Обработанные данные: {df}")

# Отбор признаков
X_input = df[best_features]
print(X_input.info()) # Логирование информации о
DataFrame

# Масштабирование
X_input_scaled = scaler.transform(X_input)

return X_input_scaled

except Exception as e:
    print("Ошибка в preprocess_input:", e)
    raise e

@app.route('/train_from_csv', methods=['POST'])
@cross_origin()
def train_from_csv():
    """
    Дообучение модели XGBoost на новых данных из CSV-файла,
    используя те же преобразования, что и при инициализации.
    """
    global model, scaler, best_features, one_hot_encoder, kmeans

```

```

try:
    print("Переобучение модели.")

    holiday_data = pd.read_csv('C:/data/holiday_data.csv',
sep=';')
    osrm_data = pd.read_csv('C:/data/osrm_data_train.csv')
    weather_data = pd.read_csv('C:/data/weather_data.csv')
    df = pd.read_csv('C:/data/train_new.csv')

    df = df.drop('store_and_fwd_flag', axis=1)
    df = df.drop('vendor_id', axis=1)
    weather_data = weather_data.drop('visibility', axis=1)

    if df.empty:
        return jsonify({'error': 'CSV file is empty'}), 400

    # Обработка данных
    df['pickup_datetime'] =
pd.to_datetime(df['pickup_datetime'], format='%Y-%m-%d
%H:%M:%S')
    df = add_datetime_features(df)
    df = add_holiday_features(df, holiday_data)
    df = add_osrm_features(df, osrm_data)
    df = add_geographical_features(df)
    df = add_cluster_features(df, kmeans)
    df = add_weather_features(df, weather_data)
    df = fill_null_weather_data(df)

    # Фильтрация данных
    df = df[(df['trip_duration'] <= 24 * 60 * 60)]
    avg_speed = df['total_distance'] / df['trip_duration'] *
3.6
    df = df[avg_speed <= 300]

    # Преобразование категориальных признаков
    df['events'] = df['events'].astype('category')

    # EDA и выбор признаков
    train_data = df.copy()
    drop_columns = ['pickup_datetime', 'pickup_date',
'dropoff_datetime', 'id']
    train_data = train_data.drop(drop_columns, axis=1)
    train_data['trip_duration_log'] =
np.log(train_data['trip_duration'] + 1)
    X = train_data.drop(['trip_duration',
'trip_duration_log'], axis=1)
    y_log = train_data['trip_duration_log']

```

```

        # OneHotEncoding
        data_onehot =
one_hot_encoder.transform(X[columns_to_change]).toarray() #
Преобразуем в массив
        column_names =
one_hot_encoder.get_feature_names_out(columns_to_change)
        data_onehot = pd.DataFrame(data_onehot,
columns=column_names)

        # Убедимся, что индексы совпадают перед объединением
X = X.reset_index(drop=True).drop(columns_to_change,
axis=1)
        data_onehot = data_onehot.reset_index(drop=True)
X = pd.concat([X, data_onehot], axis=1)

        # Масштабирование
X_scaled =
pd.DataFrame(scaler.transform(X[best_features]),
columns=best_features)

        # Проверка размерности данных
assert X_scaled.shape[1] == len(best_features),
"Размерность X_scaled не совпадает с best_features!"
        # Создание DMatrix
dtrain_new = xgb.DMatrix(X_scaled, label=y_log,
feature_names=best_features)
        model = xgb.train(
            params={
                'min_child_weight': 20,
                'eta': 0.1,
                'colsample_bytree': 0.9,
                'max_depth': 6,
                'subsample': 0.9,
                'lambda': 1,
                'nthread': -1,
                'booster': 'gbtree',
                'eval_metric': 'rmse',
                'objective': 'reg:squarederror'
            },
            dtrain=dtrain_new,
            num_boost_round=50,
            xgb_model="C:/data/parameters/xgb_model.json", #
Загружаем предыдущую модель для продолжения обучения
            verbose_eval=False
        )

        # Сохраняем обновленную модель
model.save_model("C:/data/parameters/xgb_model.json")
print("Model successfully updated from CSV with OSRM and
weather data!") # Логирование

```

```

        return jsonify({'message': 'Model updated successfully
from CSV with OSRM and weather data!'})

    except Exception as e:
        print("Ошибка в train_from_csv:", e)  # Логирование
ошибок
        return jsonify({'error': str(e)}), 400

# API для предсказания времени поездки
@app.route('/predict', methods=['POST'])
@cross_origin()
def predict():
    try:
        print("Получен POST запрос")  # Логирование получения
POST-запроса
        input_data = request.get_json()

        if not input_data:
            return jsonify({'error': 'No input data provided'}),
400

        print("Input data:", input_data)  # Логирование
полученных данных

        X_input_scaled = preprocess_input(input_data)
        dtest = xgb.DMatrix(X_input_scaled,
feature_names=best_features)
        prediction_log = model.predict(dtest)
        prediction = np.exp(prediction_log) - 1

        print("Prediction:", prediction)  # Логирование
предсказания

        return jsonify({'predicted_trip_duration':
prediction.tolist()})

    except Exception as e:
        print("Ошибка в предсказании:", e)  # Логирование ошибок
        return jsonify({'error': str(e)}), 400

if __name__ == '__main__':
    initialize()  # Инициализация модели и данных
    app.run(debug=True)

```

Файл «Leaflet.html»

```
<!DOCTYPE html>
<html lang="ru">
<head>
  <meta charset="UTF-8" />
  <meta name="viewport" content="width=device-width, initial-
scale=1.0" />
  <title>Вход | Такси</title>
  <link rel="stylesheet"
href="https://unpkg.com/leaflet/dist/leaflet.css" />
  <link rel="stylesheet" href="https://unpkg.com/leaflet-
routing-machine/dist/leaflet-routing-machine.css" />
  <style>
    body {
      margin: 0;
      font-family: 'Segoe UI', sans-serif;
      background: #f0f2f5;
    }
    .container {
      display: flex;
      height: 100vh;
    }
    .login-panel, .map-panel {
      margin: auto;
      background: white;
      border-radius: 12px;
      padding: 30px;
      box-shadow: 0 8px 24px rgba(0,0,0,0.15);
    }
    .login-panel {
      width: 300px;
    }
    .login-panel h2 {
      margin-bottom: 20px;
      text-align: center;
    }
    .leaflet-control-attribution {
      display: none !important;
    }
    input, button {
      width: 100%;
      padding: 10px;
      margin: 8px 0;
      border: 1px solid #ccc;
      border-radius: 6px;
      font-size: 14px;
    }
    button {
      background: #007bff;
```

					ПГУ 1.090301.44.001 ПЗ	Лист
Изм.	Лист	№ докум.	Подпись	Дата		92

```

        color: white;
        border: none;
        cursor: pointer;
    }
    button:hover {
        background: #0056b3;
    }
    #app-wrapper {
        display: none;
        height: 100vh;
    }
    #map {
        flex: 2;
        height: 100vh;
        position: relative;
    }
    #map-overlay {
        position: absolute;
        bottom: 0;
        left: 0;
        width: 100%;
        height: 50px;
        background: #f0f2f5;
        pointer-events: none;
        z-index: 999;
    }
    #sidebar {
        width: 300px;
        background: white;
        border-left: 1px solid #ddd;
        overflow-y: auto;
        padding: 15px;
        display: flex;
        flex-direction: column;
        justify-content: space-between;
    }
    .trip-item {
        padding: 10px;
        border: 1px solid #eee;
        border-radius: 6px;
        margin-bottom: 10px;
        cursor: pointer;
        background: #fafafa;
    }
    .trip-item:hover {
        background: #f0f0f0;
    }
    .sidebar-header {
        font-weight: bold;
        font-size: 16px;

```

```

        margin-bottom: 10px;
    }
    .hidden { display: none; }
    #action-button {
        margin-top: 10px;
        font-weight: bold;
        width: 100%;
        display: none;
    }
    #admin-controls {
        display: none;
        margin-top: 20px;
        padding: 10px;
        border-top: 1px solid #ddd;
    }
    #logout-button {
        margin-top: 15px;
        background: red;
    }
</style>
</head>
<body>
<div class="container" id="login-screen">
    <div class="login-panel">
        <h2>Вход в систему</h2>
        <input type="text" id="login" placeholder="Логин" />
        <input type="password" id="password" placeholder="Пароль" />
        <button onclick="handleLogin()">Войти</button>
        <p id="login-error" style="color: red; text-align:
center;"></p>
    </div>
</div>

<div class="container" id="app-wrapper">
    <div id="map">
    </div>
    <div id="sidebar">
        <div id="sidebar-content">
            <div class="sidebar-header">История поездок</div>
            <div id="trip-list"></div>
        </div>
        <select id="passenger-count" style="margin-top:10px;
display: none;">
            <option value="" disabled selected>Выберите количество
пассажиров</option>
            <option value="1">1 пассажир</option>
            <option value="2">2 пассажира</option>
            <option value="3">3 пассажира</option>
            <option value="4">4 пассажира</option>
        </select>
    </div>
</div>

```

```

        <button id="action-button"
onclick="handleTripAction()">Начать движение</button>
        <div id="admin-controls">
            <h3>Управление пользователями</h3>
            <input type="text" id="new-user" placeholder="Логин" />
            <input type="password" id="new-pass" placeholder="Пароль"
/>
            <button onclick="addUser()">Добавить пользователя</button>
            <input type="text" id="delete-user" placeholder="Логин для
удаления" />
            <button onclick="deleteUser()">Удалить
пользователя</button>
            <button onclick="updateModel()" style="background-
color:#28a745; color:white; margin-top:10px;">Обновить
модель</button>
        </div>
        <div id="prediction-result" style="margin-top:15px; font-
weight:bold;"></div>
        <button id="logout-button" onclick="logout()">Выйти из
системы</button>
    </div>
</div>

<script>
let map;
let predictedSeconds = null;
let tripTimer = null;
let actualSeconds = 0;
let startPoint = null;
let endPoint = null;
let routeControl;
let tripStarted = false;
let isHistoryTrip = false;
let role = null;
let startMarker = null;
let endMarker = null;

function updateModel() {
    fetch('http://localhost:5000/train_from_csv', {
        method: 'POST'
    })
        .then(res => res.json())
        .then(data => alert(data.message || 'Модель обновлена'))
        .catch(err => alert('Ошибка при обновлении модели: ' +
err));
}

function fetchPrediction(pickup_lat, pickup_lng, dropoff_lat,
dropoff_lng, passenger_count) {
    fetch('http://localhost:5000/predict', {

```

```

method: 'POST',
headers: { 'Content-Type': 'application/json' },
body: JSON.stringify({
  pickup_latitude: pickup_lat,
  pickup_longitude: pickup_lng,
  dropoff_latitude: dropoff_lat,
  dropoff_longitude: dropoff_lng,
  passenger_count: passenger_count
}))
}))
.then(res => res.json())
.then(data => {
  if (data.error) {
    console.error('Ошибка предсказания:', data.error);
    document.getElementById('prediction-result').innerText =
'Ошибка предсказания';
    return;
  }
  predictedSeconds = data.predicted_trip_duration[0];
  const minutes = Math.ceil(predictedSeconds / 60);
  document.getElementById('prediction-result').innerText =
`Предсказанное время: ${minutes} минут`;
}))
.catch(err => {
  console.error('Ошибка при получении предсказания:', err);
  document.getElementById('prediction-result').innerText =
'';
}));
}

```

```

function handleLogin() {
  const username = document.getElementById('login').value;
  const password = document.getElementById('password').value;

  fetch('http://localhost:5000/login', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, password })
  })
  .then(res => res.json())
  .then(data => {
    if (data.status === 'ok') {
      role = data.role;
      document.getElementById('login-screen').style.display =
'none';
      document.getElementById('app-wrapper').style.display =
'flex';
      document.getElementById('logout-button').style.display =
'block';
    }
  })
}

```

```

        if (role === 'admin') {
            document.getElementById('admin-
controls').style.display = 'block';
        } else {
            document.getElementById('action-button').style.display
= 'none';
            document.getElementById('passenger-
count').style.display = 'none';
        }
        loadMap();
        loadTrips(username);
    } else {
        document.getElementById('login-error').innerText =
data.message || 'Ошибка входа';
    }
});
}

function loadMap() {
    map = L.map('map').setView([53.19, 45.01], 13);

    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png'
, {
        maxZoom: 18,
        attribution: '© OpenStreetMap'
    }).addTo(map);

    map.on('click', function(e) {
        isHistoryTrip = false;
        const { lat, lng } = e.latlng;

        if (startMarker && endMarker) {
            resetTrip(); // если уже есть маршрут – сбрасываем
            document.getElementById('prediction-result').innerText =
'';
        }

        if (!startMarker) {
            startPoint = [lat, lng];
            startMarker = L.marker(startPoint).addTo(map);
        } else if (!endMarker) {
            endPoint = [lat, lng];
            endMarker = L.marker(endPoint).addTo(map);
            drawRoute();

            const selected = document.getElementById('passenger-
count');
            selected.selectedIndex = 0; // сбрасываем выбор
        }
    });
}

```

```

}

function drawRoute() {
  if (routeControl) map.removeControl(routeControl);
  routeControl = L.Routing.control({
    waypoints: [L.latLng(startPoint), L.latLng(endPoint)],
    router: L.Routing.osrmv1(),
    routeWhileDragging: false,
    showAlternatives: false
  }).addTo(map);

  if (role !== 'admin') {
    document.getElementById('action-button').style.display =
'block';
    document.getElementById('action-button').disabled = false;
  }
  if (!isHistoryTrip && role !== 'admin') {
    document.getElementById('passenger-count').style.display =
'block';
  }
  document.getElementById('action-button').style.display =
'none'; // Скрываем кнопку до выбора пассажиров
}

function handleTripAction() {
  const button = document.getElementById('action-button');

  if (!tripStarted) {
    document.getElementById('passenger-count').style.display =
'none';
    // Старт поездки
    tripStarted = true;
    button.innerText = 'Закончить поездку';
    window.pickupTime = new Date();

    // Убираем предсказание и запускаем таймер
    document.getElementById('prediction-result').innerText = '';
    actualSeconds = 0;
    tripTimer = setInterval(() => {
      actualSeconds++;
      const mins = Math.floor(actualSeconds / 60);
      const secs = actualSeconds % 60;
      document.getElementById('prediction-result').innerText =
`Прошло: ${mins}м ${secs}с`;
    }, 1000);

  } else {
    // Завершение поездки
    tripStarted = false;
    button.innerText = 'Начать движение';
  }
}

```

```

clearInterval(tripTimer);

const dropoffTime = new Date();
const pickupStr = window.pickupTime.toISOString().slice(0,
19).replace('T', ' ');
const dropoffStr = dropoffTime.toISOString().slice(0,
19).replace('T', ' ');

const username = document.getElementById('login').value;

const tripData = {
  pickup_datetime: pickupStr,
  dropoff_datetime: dropoffStr,
  pickup_latitude: startPoint[0],
  pickup_longitude: startPoint[1],
  dropoff_latitude: endPoint[0],
  dropoff_longitude: endPoint[1],
  passenger_count: getPassengerCount(),
  username: username,
  actual_duration: actualSeconds // ← передаём реальную
длительность
};

fetch('http://localhost:5000/record_trip', {
  method: 'POST',
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify(tripData)
})
.then(res => res.json())
.then(data => {
  if (data.status === 'ok') {
    alert('Поездка сохранена');
    loadTrips(username);
  } else {
    alert('Ошибка сохранения: ' + data.message);
  }
})
.catch(err => {
  alert('Ошибка отправки: ' + err);
});

resetTrip();
}
}

function resetTrip() {
  startPoint = null;
  endPoint = null;
  if (routeControl) map.removeControl(routeControl);

```

```

    if (startMarker) map.removeLayer(startMarker);
    if (endMarker) map.removeLayer(endMarker);
    if (tripTimer) clearInterval(tripTimer);
    actualSeconds = 0;
    startMarker = null;
    endMarker = null;
    document.getElementById('action-button').style.display =
'none';
    document.getElementById('prediction-result').innerText = '';
    document.getElementById('passenger-count').style.display =
'none';
    document.getElementById('passenger-count').selectedIndex = 0;
    tripStarted = false;
}

function logout() {
    location.reload();
}

function loadTrips(username) {
    fetch('http://localhost:5000/user_trips', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username, role })
    })
        .then(res => res.json())
        .then(data => renderTrips(data.trips))
        .catch(err => console.error('Ошибка при загрузке истории:',
err));
}

function renderTrips(trips) {
    const tripList = document.getElementById('trip-list');
    tripList.innerHTML = '';
    trips.forEach(trip => {
        const item = document.createElement('div');
        item.className = 'trip-item';
        item.innerHTML = `${trip.pickup_datetime} →
${trip.dropoff_datetime}`;
        item.onclick = () => {
            resetTrip(); // сбросим всё
            document.getElementById('passenger-count').style.display =
'none';
            startPoint = [trip.pickup_latitude,
trip.pickup_longitude];
            endPoint = [trip.dropoff_latitude,
trip.dropoff_longitude];

            startMarker = L.marker(startPoint).addTo(map);
            endMarker = L.marker(endPoint).addTo(map);

```

```

        isHistoryTrip = true;
        drawRoute();

        document.getElementById('action-button').style.display =
'none';

        // Показываем реальное время поездки
        const duration = Math.ceil(trip.trip_duration / 60);
        document.getElementById('prediction-result').innerText =
`Реальное время: ${duration} минут`;

    };
    tripList.appendChild(item);
  });
}

function addUser() {
  const username = document.getElementById('new-user').value;
  const password = document.getElementById('new-pass').value;

  if (!username || !password) {
    alert('Введите логин и пароль');
    return;
  }

  fetch('http://localhost:5000/add_user', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, password })
  })
    .then(res => res.json())
    .then(data => alert(data.message || 'Пользователь
добавлен'))
    .catch(err => alert('Ошибка добавления пользователя: ' +
err));
}

function addUser() {
  const username = document.getElementById('new-user').value;
  const password = document.getElementById('new-pass').value;

  if (!username || !password) {
    alert('Введите логин и пароль');
    return;
  }

  fetch('http://localhost:5000/add_user', {
    method: 'POST',
    headers: { 'Content-Type': 'application/json' },
    body: JSON.stringify({ username, password })
  })

```

```

    })
    .then(res => res.json())
    .then(data => alert(data.message || 'Пользователь
добавлен'))
    .catch(err => alert('Ошибка добавления пользователя: ' +
err));
}

function deleteUser() {
    const username = document.getElementById('delete-user').value;

    if (!username) {
        alert('Введите логин для удаления');
        return;
    }

    fetch('http://localhost:5000/delete_user', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ username })
    })
    .then(res => res.json())
    .then(data => {
        if (data.status === 'ok') {
            alert(data.message);
        } else {
            alert('Ошибка: ' + data.message);
        }
    })
    .catch(err => alert('Ошибка удаления: ' + err));
}

function updateModel() {
    fetch('http://localhost:5000/train_from_csv', {
        method: 'POST'
    })
    .then(res => res.json())
    .then(data => alert(data.message || 'Модель обновлена'))
    .catch(err => alert('Ошибка при обновлении модели: ' +
err));
}

function getPassengerCount() {
    const select = document.getElementById('passenger-count');
    return parseInt(select.value) || 1;
}

document.getElementById('passenger-
count').addEventListener('change', () => {

```

```

    document.getElementById('action-button').style.display =
    'block';

    // Вызываем предсказание
    if (startPoint && endPoint) {
        fetchPrediction(
            startPoint[0], startPoint[1],
            endPoint[0], endPoint[1],
            getPassengerCount()
        );
    }
    });

</script>

<script
src="https://unpkg.com/leaflet/dist/leaflet.js"></script>
<script src="https://unpkg.com/leaflet-routing-
machine/dist/leaflet-routing-machine.js"></script>
</body>
</html>

```

					ПГУ 1.090301.44.001 ПЗ	Лист
						103
Изм.	Лист	№ докум.	Подпись	Дата		

Приложение В

(обязательное)

Разработка системы предсказания времени поездки на такси на основе

машинного обучения

Презентация

					ПГУ 1.090301.44.001 ПЗ	Лист
						104
Изм.	Лист	№ докум.	Подпись	Дата		

Федеральное государственное бюджетное образовательное
учреждение высшего образования
Пензенский государственный университет



ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

«Программа планирования маршрутов городского такси»

Сивущин Артём Сергеевич

Направление подготовки:
09.03.01 «Информатика и вычислительная
техника»

Руководитель:
к.т.н., доцент Федюнин Роман Николаевич

Актуальность

Более эффективное
планирование логистики

Улучшение пользовательского
опыта

Проблема точного
прогнозирования времени
поездки

Сложности: пробки, погода,
человеческий фактор

Решение: интеграция
машинного обучения и внешних
данных

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

105

Цель

Разработка системы предсказания времени поездки на такси на основе машинного обучения и внешних данных (OSRM, OpenWeather API).

Объект исследования

Процесс маршрутизации и прогнозирования времени поездки

Предмет исследования

Модель машинного обучения, интегрированная в веб-сервис для предоставления предсказаний

Задачи

Сбор и анализ данных о поездках, погоде и маршрутах движения.

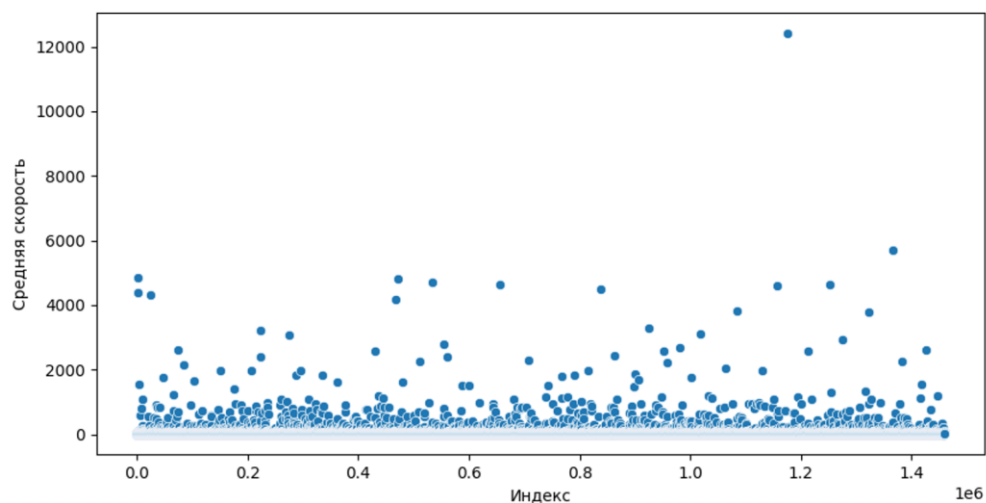
Разработка модели машинного обучения

Интеграция модели в веб-приложение

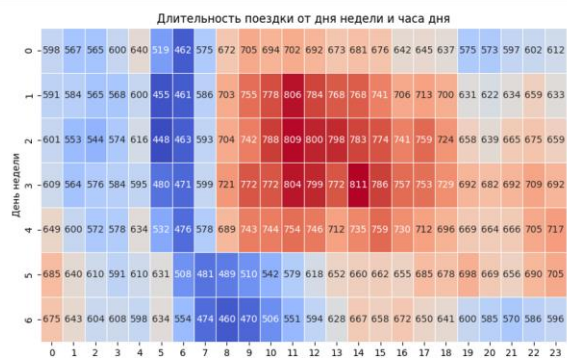
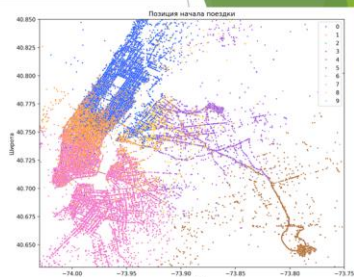
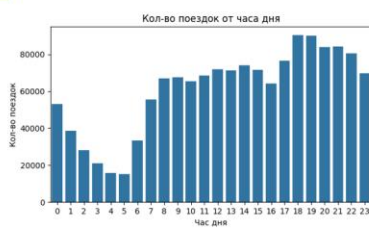
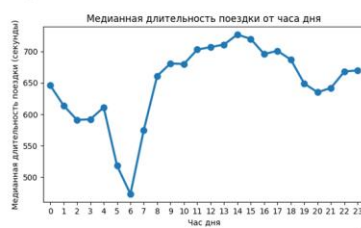
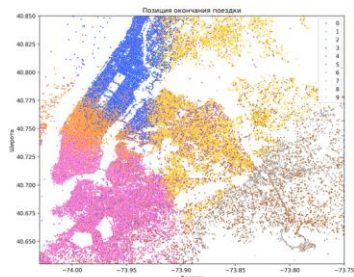
Тестирование и оценка качества предсказаний

Создание интерфейса визуализации маршрутов

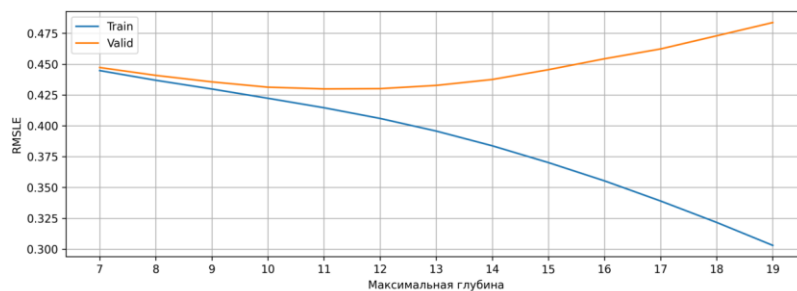
Очистка данных



Разведывательный анализ данных



Моделирование



Train RMSLE: 0.53
Valid RMSLE: 0.54



Train RMSLE: 0.48
Valid RMSLE: 0.48



Train RMSLE: 0.41
Valid RMSLE: 0.43



Train RMSLE: 0.3715
Valid RMSLE: 0.393

Изм.	Лист	№ докум.	Подпись	Дата

ПГУ 1.090301.44.001 ПЗ

Лист

108