Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет Электроники и вычислительной техники   
Направление (специальность) Информатика и вычислительная техника . .  
Кафедра Электронно-вычислительные машины и системы"

Дисциплина Системы обработки больших данных

|  |  |
| --- | --- |
|  | Утверждаю  Зав. кафедрой\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ |
|  | «\_\_\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г. |

ЗАДАНИЕ

на курсовую работу (проект)

Студент Куканов Александр Дмитриевич .

(фамилия, имя, отчество)

Группа САПР-1.1

1. Тема: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Утверждена приказом от «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_ 20\_\_\_ г. № \_\_\_\_\_\_\_\_\_

2. Срок представления работы (проекта) к защите «\_\_\_»\_\_\_\_\_\_\_\_\_\_\_\_\_20\_\_ г.

3. Содержание расчетно-пояснительной записки: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

4. Перечень графического материала: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Дата выдачи задания «\_\_\_\_\_» \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_20 \_\_\_ г.

Руководитель работы (проекта)\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

подпись, дата инициалы и фамилия

Задание принял к исполнению\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_ .

подпись, дата инициалы и фамилия

Минобрнауки России

Федеральное государственное бюджетное образовательное учреждение высшего образования

«Волгоградский государственный технический университет»

Факультет \_Электроники и вычислительной техники\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Кафедра Системы автоматизированного проектирования и поискового конструирования

**ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

**к курсовой работе (проекту)**

по дисциплине Системы обработки больших данных

на тему .

Студент Куканов Александр Дмитриевич .

(фамилия, имя, отчество)

Группа САПР-1.1

Руководитель работы (проекта) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись и дата подписания) (инициалы и фамилия)

Нормоконтролер \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(подпись, дата подписания) (инициалы и фамилия)

Волгоград 2023 г.

Содержание

[Введение 4](#_Toc155913011)

[1. РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ PYSPARK 5](#_Toc155913012)

[1.1 Постановка задачи 5](#_Toc155913013)

[1.2 Определение типов признаков в датасете 5](#_Toc155913014)

[1.3 Определение пропущенных значений и их устранение 9](#_Toc155913015)

[1.4 Определение выбросов и их устранение 9](#_Toc155913016)

[1.6 Расчет статистических показателей признаков 12](#_Toc155913017)

[1.7 Визуализация распределения наиболее важных признаков 12](#_Toc155913018)

[1.8 Корреляция между признаками 16](#_Toc155913019)

[1.9. Выводы 18](#_Toc155913020)

[2. МАШИННОЕ ОБУЧЕНИЕ НА БОЛЬШИХ ДАННЫХ 19](#_Toc155913021)

[2.1. Постановка задачи 19](#_Toc155913022)

[2.1. Подготовка данных для модели регрессии 19](#_Toc155913023)

[2.2. Определение пайплайна для задачи регрессии 19](#_Toc155913024)

[2.3. Генерация предсказаний случайного леса 20](#_Toc155913025)

[2.4. Оценка модели регрессии 20](#_Toc155913026)

[2.5. Настройка параметров регрессии 20](#_Toc155913027)

[2.6. Подготовка данных для модели классификации 22](#_Toc155913028)

[2.7. Подготовка пайплайна логистической регрессии 23](#_Toc155913029)

[2.8. Генерация предсказаний модели бинарной классификации 23](#_Toc155913030)

[2.9. Настройка параметров бинарной классификации 25](#_Toc155913031)

[2.11. Выводы 26](#_Toc155913032)

[ЗАКЛЮЧЕНИЕ 27](#_Toc155913033)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28](#_Toc155913034)

[ПРИЛОЖЕНИЕ А – исходный код разведочного анализа датасета 29](#_Toc155913035)

[ПРИЛОЖЕНИЕ Б – исходный код машинного обучения на больших данных 34](#_Toc155913036)

# Введение

В современном мире, где объемы данных растут с небывалой скоростью, специалисты во всех отраслях сталкиваются с необходимостью их анализа и обработки. Эти данные, известные как "большие данные" (Big Data), характеризуются своим объемом, скоростью поступления и разнообразием, что ставит перед исследователями и инженерами новые задачи и требует особого подхода к их обработке.

Ключевым инструментом был выбран PySpark - интерфейс Apache Spark для языка программирования Python. Используя PySpark, на практике были освоины основы работы с RDD (Resilient Distributed Datasets) и DataFrame API, которые являются ключевыми абстракциями в Apache Spark, позволяя обрабатывать данные эффективно и интуитивно понятно.

# 1. РАЗВЕДОЧНЫЙ АНАЛИЗ ДАННЫХ С ПОМОЩЬЮ PYSPARK

## 1.1 Постановка задачи

Выполнить разведочный анализ датасета авиабилетов из Expedia согласно варианту с определением:

1. типов признаков в датасете;
2. пропущенных значений и их устранением;
3. выбросов и их устранением;
4. расчетом статистических показателей признаков (средних, квартилей и т.д.);
5. визуализацией распределения наиболее важных признаков;
6. корреляций между признаками.

Источник: <https://www.kaggle.com/datasets/dilwong/flightprices>

## 1.2 Определение типов признаков в датасете

Датасет содержит информацию о ценах на полеты в одну сторону самолетами согласно Expedia на период с 16.04.2022 до 05.10.2022.

Датасет представляет собой CSV-файл, где каждая строка - купленный билет в/из следующих аэропортов: ATL, DFW, DEN, ORD, LAX, CLT, MIA, JFK, EWR, SFO, DTW, BOS, PHL, LGA, IAD, OAK.

Определения столбцов:

1. legId: Идентификатор рейса;
2. searchDate: Дата (ГГГГ-ММ-ДД), когда эта запись была взята из Expedia;
3. flightDate: Дата (ГГГГ-ММ-ДД) полета;
4. startingAirport: Трехсимвольный код аэропорта IATA для начального местоположения;
5. destinationAirport: Трехсимвольный код аэропорта IATA для места прибытия;
6. fareBasisCode: Код основного тарифа;
7. travelDuration: Продолжительность поездки в часах и минутах;
8. elapsedDays: Количество прошедших дней (обычно 0);
9. isBasicEconomy: Булево значение, указывающее, предназначен ли билет для базовой экономики;
10. isRefundable: Булево значение, указывающее, можно ли вернуть деньги за билет;
11. isNonStop: Булево значение, указывающее, является ли рейс беспосадочным;
12. baseFare: Цена билета (в USD);
13. totalFare: Цена билета (в USD), включая налоги и другие сборы;
14. seatsRemaining: Целое число для количества оставшихся мест;
15. totalTravelDistance: Общее расстояние поездки в милях. Эти данные иногда отсутствуют;
16. segmentsDepartureTimeEpochSeconds: Строка, содержащая время отправления (время Unix) для каждого этапа поездки;
17. segmentsDepartureTimeRaw: Строка, содержащая время отправления (формат ISO 8601: YYYY-MM-DDThh:mm:ss.000±[hh]:00) для каждого этапа поездки.
18. segmentsArrivalTimeEpochSeconds: Строка, содержащая время прибытия (время Unix) для каждого этапа поездки.
19. segmentsArrivalTimeRaw: Строка, содержащая время прибытия (формат ISO 8601: YYYY-MM-DDThh:mm:ss.000±[hh]:00) для каждого этапа поездки.
20. segmentsArrivalAirportCode: Строка, содержащая код аэропорта IATA для места прибытия для каждого этапа поездки.
21. segmentsDepartureAirportCode: Строка, содержащая код аэропорта IATA для места отправления на каждом этапе путешествия.
22. segmentsAirlineName: Строка, содержащая название авиакомпании, обслуживающей каждый этап путешествия.
23. segmentsAirlineCode: Строка, содержащая двухбуквенный код авиакомпании, обслуживающей каждый этап путешествия.
24. segmentsEquipmentDescription: Строка, содержащая тип самолета, используемого на каждом этапе путешествия (например, "Airbus A321" или "Boeing 737-800").
25. segmentsDurationInSeconds: Строка, содержащая продолжительность полета (в секундах) для каждого этапа путешествия.
26. segmentsDistance: Строка, содержащая пройденное расстояние (в милях) для каждого этапа поездки.
27. segmentsCabinCode: Строка, содержащая код салона для каждого этапа поездки (например, "coach").

Данные представляют собой значения следующих типов данных: integer, double, string, date, boolean. Типы колонок представлены в таблице 1.

Таблица 1 – типы признаков

| № | Название колонки | Тип данных | Тип признака |
| --- | --- | --- | --- |
| 0 | legId | string | Категориальный |
| 1 | searchDate | date | Порядковый |
| 2 | flightDate | date | Порядковый |
| 3 | startingAirport | string | Категориальный |
| 4 | destinationAirport | string | Категориальный |
| 5 | fareBasisCode | string | Категориальный |
| 6 | travelDuration | string | Категориальный |
| 7 | elapsedDays | int | Бинарный |
| 8 | isBasicEconomy | boolean | Бинарный |
| 9 | isRefundable | boolean | Бинарный |
| 10 | isNonStop | boolean | Бинарный |
| 11 | baseFare | double | Количественный |
| 12 | totalFare | double | Количественный |
| 13 | seatsRemaining | int | Количественный |
| 14 | totalTravelDistance | int | Количественный |
| 15 | segmentsDepartureTimeEpochSeconds | string | Категориальный |
| 16 | segmentsDepartureTimeRaw | string | Категориальный |
| 17 | segmentsArrivalTimeEpochSeconds | string | Категориальный |
| 18 | segmentsArrivalTimeRaw | string | Категориальный |
| 19 | segmentsArrivalAirportCode | string | Категориальный |
| 20 | segmentsDepartureAirportCode | string | Категориальный |
| 21 | segmentsAirlineName | string | Категориальный |
| 22 | segmentsAirlineCode | string | Категориальный |
| 23 | segmentsEquipmentDescription | string | Категориальный |
| 24 | segmentsDurationInSeconds | string | Категориальный |
| 25 | segmentsDistance | string | Категориальный |
| 26 | segmentsCabinCode | string | Категориальный |

Всего в датасете 1047402 строки.

## 1.3 Определение пропущенных значений и их устранение

Для корректного анализа данных необходимо убедиться, что датасет не имеет какие-либо пропущенные и аномальные нулевые значения. С помощью кода было найдено количество пустых и нулевых значений, в каких колонках были найдены эти значения, а также удаление этих строк:

# Подсчет пропущенных значений

dataframe.select([col(column).isNull().cast("int").alias(column) for column in dataframe.columns]).groupBy().sum().show()

# Удаление строк с пропущенными значениями

cleaned\_dataframe **=** dataframe**.**dropna**()**

# Вывод количество строк после удаления пустых значений

cleaned\_dataframe**.**count**()**

В результате выполнения кода было обнаружено 64719 пустых значений в столбце totalTravelDistance и 22163 в столбце travelDuration. После удаления пропущенных значений осталось 961815 строк в датафрейме.

## 1.4 Определение выбросов и их устранение

Для повышения качества данных для анализа необходимо определить и устранить в них выбросы.

# Фильтрация выбросов для каждого столбца

for column in selected\_columns:

# Рассчет квартилей - первый (0.25) и третий (0.75)

# 0.0 - максимально возможная точность

quartiles = cleaned\_dataframe.stat.approxQuantile(column, [0.25, 0.75], 0.0)

# Рассчет интерквартильного размаха между первым и третьим квартилем

IQR = quartiles[1] - quartiles[0]

# Определение границ для определения выбросов

lower\_bound = quartiles[0] - 1.5 \* IQR

upper\_bound = quartiles[1] + 1.5 \* IQR

# Фильтрация данных и подсчет значений ниже и выше границ выбросов

below\_quartile\_count\_before = cleaned\_dataframe.filter(col(column) < lower\_bound).count()

above\_quartile\_count\_before = cleaned\_dataframe.filter(col(column) > upper\_bound).count()

В фрагменте кода выше для определения и устранения выбросов, реализующий следующий цикл:

1. рассчитываются первый (25%) и третий (75%) квартили, а также интерквартильный размах;
2. определяются границы для определения выбросов с использованием коэффициента 1.5;
3. подсчитываются значения, находящиеся ниже и выше границ выбросов до фильтрации, а также максимальное, минимальное и медианное значения;
4. производится фильтрация данных и повторно подсчитываются значения после обработки выбросов.

Результат устранения выбросов:

Таблица 2 – результат устранения выбросов

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Столбец | (было) Выбросы ниже границы | (стало)  Выбросы ниже границы | (было) Выбросы выше границы | (стало)  Выбросы выше границы | (было)  макс. значение | (стало)  макс. значение | (было)  мин. значение | (стало)  мин. значение | (было) медиана | (стало)  медиана |
| baseFare | 0 | 0 | 21046 | 0 | 3531.16 | 750.7 | 8.72 | 8.72 | 277.21 | 272.55 |
| totalFare | 0 | 0 | 1466 | 0 | 847.8 | 816.1 | 23.97 | 23.97 | 314.2 | 313.6 |
| totalTravelDistance | 0 | 0 | 0 | 0 | 3958 | 3958 | 121 | 121 | 1392.0 | 1392.0 |
| travelDuration | 0 | 0 | 7758 | 0 | 1433 | 979 | 62 | 62 | 388 | 385 |

По итогу устранения выбросов качество данных для дальнейшей обработки увеличилось.

## 1.6 Расчет статистических показателей признаков

При помощи следующего фрагмента кода выведем статистические показатели для таких признаков, как цена билета, цена билета с учетом налогов, преодоленное расстояние и длительность полета:

# Расчет статистических показателей

statistics\_summary=cleaned\_dataframe.select('baseFare',totalFare',

'totalTravelDistance','travelDuration').describe()

# Вывод сводной статистики

statistics\_summary.toPandas()

Таблица 3 – статистические параметры признаков

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| summary | baseFare | totalFare | totalTravelDistance | travelDuration |
| count | 931545 | 931545 | 931545 | 931545 |
| mean | 287.10 | 330.49 | 1550.64 | 400.20 |
| stddev | 147.75 | 160.97 | 824.39 | 204.51 |
| min | 8.72 | 23.97 | 121 | 62 |
| max | 743.72 | 816.1 | 3958 | 979 |

## 1.7 Визуализация распределения наиболее важных признаков

Визуализации распределения подвергнутся признаки, используемые ранее для выявления статистических параметров, а также пункты вылета и назначения, тариф, наличие пересадок, количество пустующих мест. Далее приведены диаграммы распределения признаков.

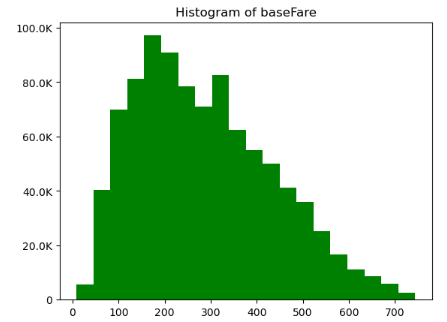


Рисунок 1 – распределение базовой стоимости билета

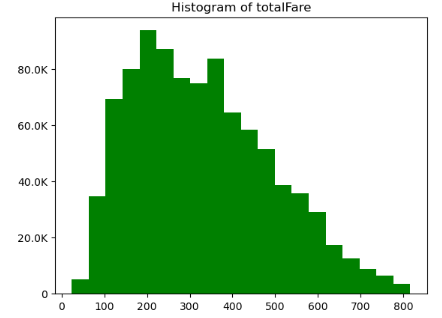


Рисунок 2 – распределение итоговой стоимости билета

Распределение стоимости с преобладанием в значениях среднего и ниже среднего гласит о преимущественном количестве билетов эконом-класса. Проверим эту гипотезу позднее с рассмотрением распределения билетов по эконом-классу и бизнесс-класу.

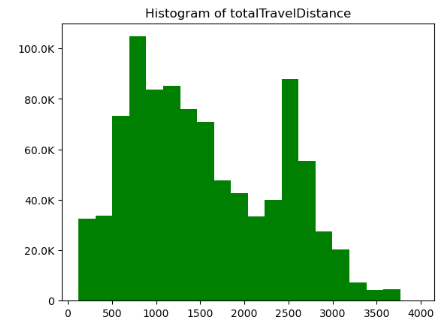


Рисунок 3 – распределение дистанции полета

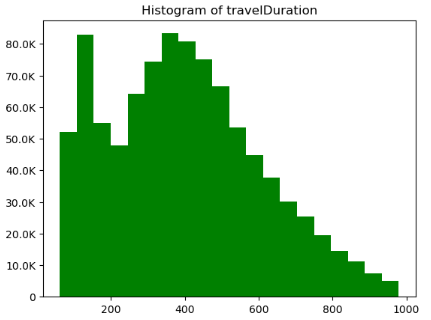


Рисунок 4 – распределение длительности полета

По данным на рисунке 4 видно, что длительность полета в большинстве случаев составляет меньше десяти часов.

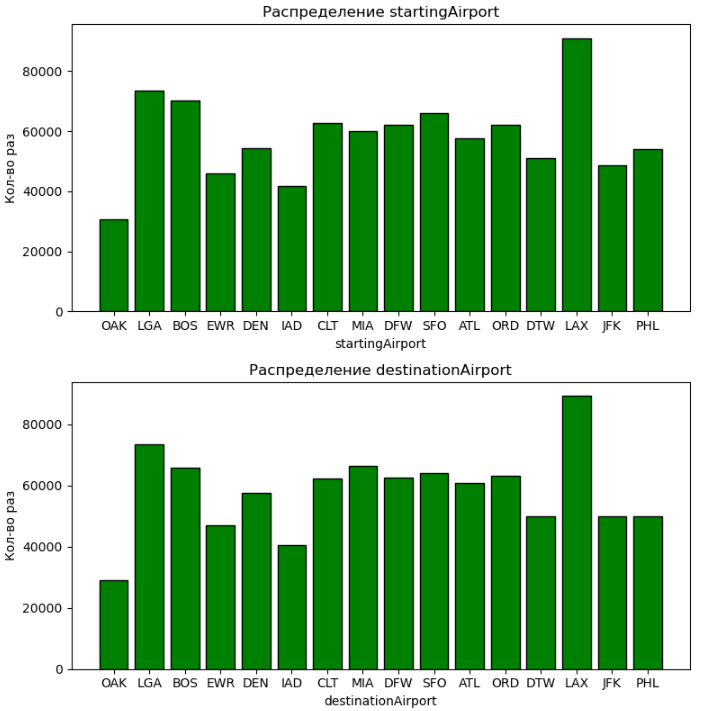


Рисунок 5 – распределение отправных и приемных аэропортов

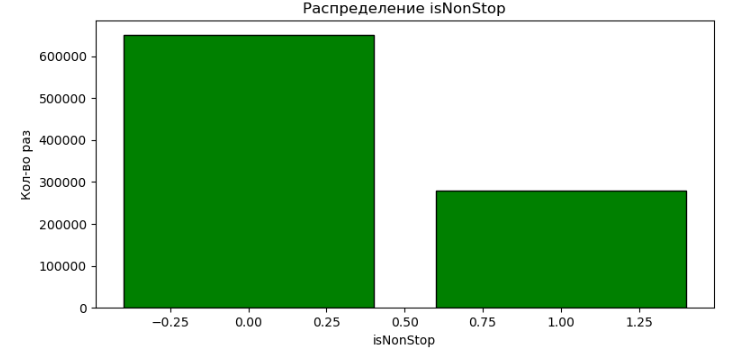


Рисунок 6 – распределение билетов с пересадками и без

На рисунке 6 заметно преобладание билетов без пересадок более чем в два раза.

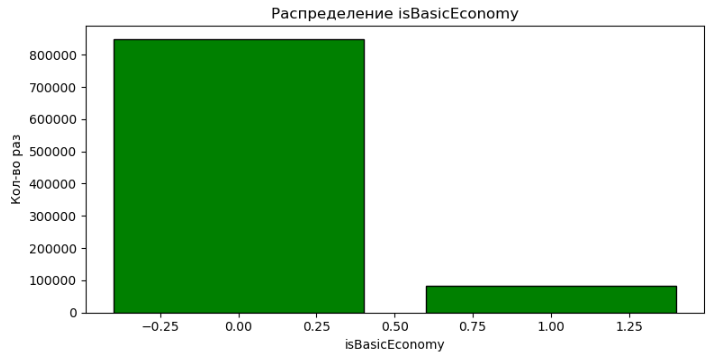


Рисунок 7 – распределение билетов эконом-класса и бизнес-класса

На рисунке 7 замечается преобладание билетов эконом-класса, что подтверждает гипотезу о преобладании билетов эконом-класса.

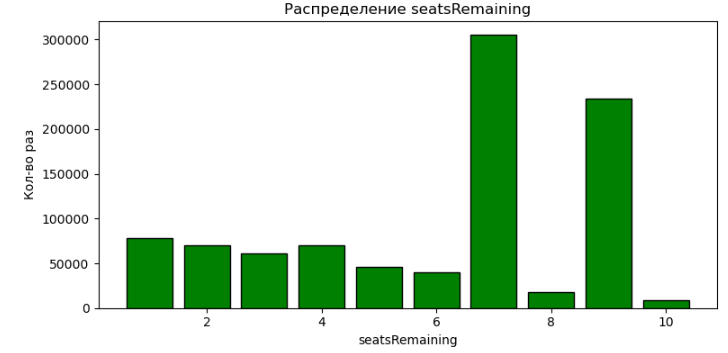


Рисунок 8 – распределение пустующих мест

## 1.8 Корреляция между признаками

Построим матрицу корреляции с использованием следующего фрагмента кода:

# Создаем VectorAssembler для объединения признаков в один вектор

# vector\_assembler - трансформатор, объединяющий столбцы feature\_cols в вектор-столбец

feature\_cols = ['baseFare', 'totalFare','totalTravelDistance', 'travelDuration']

vector\_assembler = VectorAssembler(inputCols=feature\_cols, outputCol="features")

# Применяем VectorAssembler к данным

data = vector\_assembler.transform(cleaned\_dataframe)

# Вычисляем матрицу корреляции

correlation\_matrix = Correlation.corr(data, "features").head()

correlation\_array = correlation\_matrix[0].toArray()

# Выводим матрицу корреляции

correlation\_df = pd.DataFrame(correlation\_array, columns=feature\_cols)

# Вычисление корреляционной матрицы

correlation\_df.corr()

Для наглядности представим матрицу корреляции в виде тепловой карты.

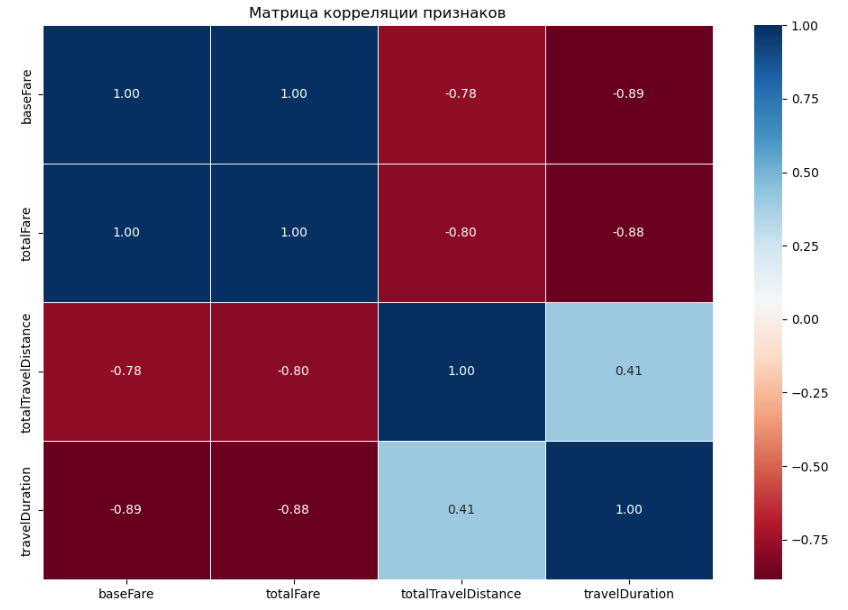


Рисунок 9 – матрица корреляции

На рисунке 9 можно заметить высокую корреляцию между базовой стоимостью билета и итоговой стоимостью, что логично, т.к. итоговая представляет стоимость представляет базовую с учетом налогов. Положительная корреляция наблюдается у длительности полета и его дистанции. Учтем эти признаки для дальнейшей работы с данными.

## 1.9. Выводы

В рамках данного раздела был проведен разведочный анализ датасета авиабилетов. Были выявлены типы признаков в датасете, определены и устранены пропущенные значения и выбросы, рассчитаны статистические показатели признаков, визуализация наиболее важных признаков, а также определена корреляция между ними.

# 2. МАШИННОЕ ОБУЧЕНИЕ НА БОЛЬШИХ ДАННЫХ

## 2.1. Постановка задачи

Выполнить анализ выбранного ранее датасета с помощью алгоритма случайный лес. Задача регресии – прогнозирование длительности полета.

## 2.1. Подготовка данных для модели регрессии

Для обучения модели используются случайные 70% данных, а оставшиеся 30% для тестирования. Разделение выполняется при помощи команды .randomSplit([0.7 , 0.3]).

## 2.2. Определение пайплайна для задачи регрессии

Для создания столбца признаков необходимо предпринять следующий порядок действий:

1. преобразование столбцов с начальным и конечным аэропортом в числовые индексы, а также закодировать их One-hot кодированием;
2. объединить категориальные признаки в один вектор-столбец;
3. масштабировать оставшиеся числовые признаки;
4. объединить категориальные и числовые признаки в один вектор столбец.

Соответственно эти же этапы входят в пайплайн для задачи регрессии, который будет иметь следующий вид: reg\_pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, catIdx, numVect, minMax, featVect, rf]).

## 2.3. Генерация предсказаний случайного леса

Рассмотрим результаты обучения модели на тренировочных данных.

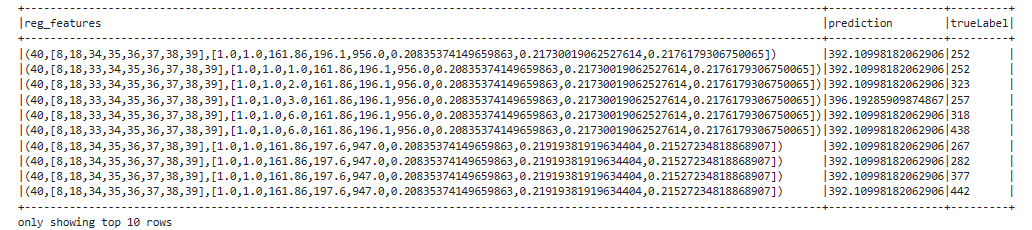


Рисунок 10 – результаты обучения модели на тренировочных данных

## 2.4. Оценка модели регрессии

Оценка модели регрессии проводится вычислением следующих метрик: RMSE, R2, MAE, stddev. Вычисляются метрики при помощи .RegressionEvaluator().

Таблица 4 – оценка качества модели

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | RMSE | R2 | MAE |
| Значение | 109.472 | 0.714 | 78.039 |

На основании данных из таблицы можно заметить отклонение между фактическими и предсказанными значениями, однако хоть модель не является идеальной, метрики (в особенности, R2 и RMSE) находятся выше среднего.

## 2.5. Настройка параметров регрессии

В целях улучшить качество модели настроим гиперпараметры модели. В данном случае будут настраиваться такие гиперпараметры, как максимальная глубина дерева и количество деревьев в случайном лесу. Для гиперпараметра глубины дерева зададим следующие значения: 2, 5, 10. Для количества деревьев – 5, 10, 20.

Для выбора наилучшей комбинации необходимо провести кросс-валидацию с созданием соответствующего объекта:

## Создание объекта для кросс-валидации

# estimator - оцениваемая модель (наш пайплайн c моделью и преобразованием данных)

# estimatorParamMaps - сетка параметров для оцениваемой модели

# evaluator - оценка качества модели на каждом этапе кросс-валидации (по умолчанию RMSE)

# numFolds - количество складываний для кросс-валидации, т.е. на сколько частей будет разделен набор данных (в нашем случае разделение на 3 части и обучение 3 раза)

cv = CrossValidator(estimator=reg\_pipeline,

estimatorParamMaps=param\_grid,

evaluator=RegressionEvaluator(),

numFolds=3)

# Обучение модели с использованием кросс-валидации

cv\_model = cv.fit(train)

# Получение наилучшей модели после обучения с кросс-валидацией

best\_cv\_model = cv\_model.bestModel

# Оценка производительности на тестовом наборе

cv\_prediction = best\_cv\_model.transform(test)

Рассмотрим метрики для оценки качества модели после подбора гиперпараметров и сравним с изначальными.

Таблица 5 – сравнение метрик до и после подбора гиперпараметров

|  |  |  |  |
| --- | --- | --- | --- |
| Метрика | RMSE | R2 | MAE |
| Значение (до) | 109.472 | 0.714 | 78.039 |
| Значение (после) | 100.63 | 0.758 | 69.390 |

Снижение значения RMSE после подбора гиперпараметров указывает на улучшение точности модели. Меньшее значение RMSE говорит о том, что предсказанные значения ближе к фактическим. Увеличение значения R2 после подбора гиперпараметров свидетельствует о лучшей объяснительной силе модели. Большее значение R2 указывает на более точные и надежные прогнозы. Уменьшение значения MAE после подбора гиперпараметров говорит о том, что модель лучше предсказывает реальные значения. Меньшая средняя абсолютная ошибка указывает на более точные прогнозы.

На основе сравнения метрик, можно утверждать, что модель после подбора гиперпараметров предоставляет более точные и качественные прогнозы, так как все три метрики (RMSE, R2, MAE) улучшились. Таким образом, вариант после подбора гиперпараметров считается лучшим.

## 2.6. Подготовка данных для модели классификации

В рамках задачи бинарной классификации необходимо установить, будет ли длительность полета составлять больше 375 минут. Соответственно, необходимо преобразовать столбец с длительностью полета в бинарный: 0 при длительности меньше 375 минут, а 1 – при длительности больше 375 минут. Разделим данные на тренировочные (70%) и проверочные (30%).

## 2.7. Подготовка пайплайна логистической регрессии

Для создания столбца признаков необходимо предпринять следующий порядок действий:

1. преобразование столбцов с начальным и конечным аэропортом в числовые индексы, а также закодировать их One-hot кодированием;
2. объединить категориальные признаки в один вектор-столбец;
3. масштабировать оставшиеся числовые признаки;
4. объединить категориальные и числовые признаки в один вектор столбец.

Соответственно пайплайн для решения задачи бинарной классификации будет следующим: pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, catIdx, numVect, minMax, featVect, lr]).

## 2.8. Генерация предсказаний модели бинарной классификации

Рассмотрим результаты генерации.

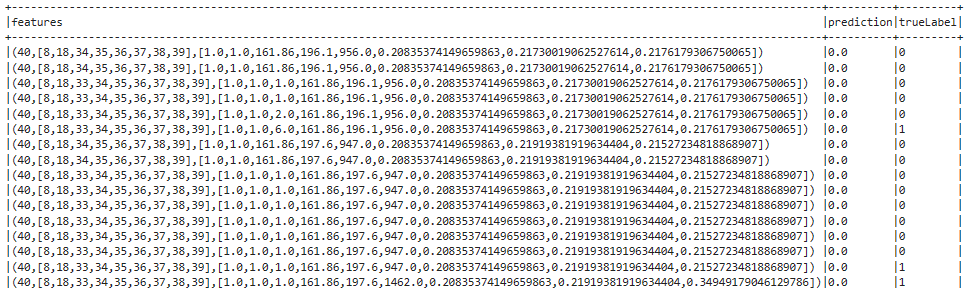


Рисунок 11 - предсказания модели бинарной классификации

На первый взгляд, предсказания модели удовлетворительные, но необходимо рассмотреть метрики для оценки точности модели:

1. TP – количество истинно-положительных значений;
2. FP – количество ложно-положительных значений;
3. TN – количество истинно-отрицательных значений;
4. FN – количество ложно-отрицательных значений;
5. Precision – точность;
6. Recall – чувствительность;
7. F1 – среднее гармоническое между precision и recall.

Таблица 6 – оценка точности модели бинарной классификации

| Метрика | Значение |
| --- | --- |
| TP | 123775.0 |
| FP | 30391.0 |
| TN | 103205.0 |
| FN | 22097.0 |
| Precision | 0.8028683367279426 |
| Recall | 0.848517878688165 |
| F1 | 0.8250621587932194 |

На основании метрик можно сделать следующие выводы:

1. модель имеет высокий процент правильных предсказаний положительных случаев (Precision);
2. модель хорошо "ловит" истинно-положительные случаи (Recall);
3. F1 Score учитывает баланс между Precision и Recall и также находится на уровне, что говорит о хорошем балансе между точностью и чувствительностью.

Таким образом, основываясь на предоставленных метриках, можно сказать, что модель хорошо справляется с задачей бинарной классификации

## 2.9. Настройка параметров бинарной классификации

В целях повысить точность модели, произведен подбор гиперпараметров. В данном случае это коэффициент регуляции (0.3 и 0.2), максимальное количество итераций (15 и 10), порог классификации (0.35 и 0.45). Код кросс-валидации аналогичен коду для задачи регрессии. Рассмотрим метрики после подбора гиперпараметров.

Таблица 7 – сравнение метрик до и после подбора гиперпараметров в рамках задачи бинарной классификации

| Метрика | Значение (до) | Значение (после) |
| --- | --- | --- |
| TP | 123775.0 | 133773.0 |
| FP | 30391.0 | 40758.0 |
| TN | 103205.0 | 92838.0 |
| FN | 22097.0 | 12099.0 |
| Precision | 0.8028683367279426 | 0.7664712859033639 |
| Recall | 0.848517878688165 | 0.9170574202040145 |
| F1 | 0.8250621587932194 | 0.8350296345539836 |

Замечается небольшое уменьшение точности модели, однако повышается чувствительность и среднее гармоническое, что может говорить о том, что подбор гиперпараметров повысил общую точность модели.

## 2.11. Выводы

В рамках данного раздела были поставлены и решены задачи линейнойр регрессии и бинарной классификации с использованием алгоритмов случайного леса и логистической регрессии соответственно. Были предприняты попытки повысить точность моделей путем подбора гиперпараметров, что незначительно, но положительно сказалось на точности модели. Для большей точности моделей необходима более глубокая настройка, что можно включить в перспективы работы.

# ЗАКЛЮЧЕНИЕ

Выполнение данной работы по анализу данных и применению машинного обучения в контексте больших данных с использованием PySpark позволило получить ценные практические навыки и познакомиться с ключевыми концепциями этого фреймворка. Освоение работы с PySpark, а также применение различных методов анализа и визуализации данных на примере датасета авиабилетов позволило лучше понять специфику обработки больших объемов информации.

В ходе разведочного анализа были успешно выявлены и устранены пропущенные значения и выбросы, а также проведено исследование важных признаков с использованием статистических методов и визуализации. Это позволило получить более четкое представление о структуре данных и их взаимосвязях.

Работа также охватила задачи линейной регрессии и бинарной классификации, применяя алгоритмы случайного леса и логистической регрессии. Подбор гиперпараметров несмотря на небольшой прирост точности, подчеркнул важность тщательной настройки моделей для достижения лучших результатов. Это стало стимулом для дальнейшего исследования и оптимизации.

В перспективе, для дальнейшего улучшения точности моделей, можно провести более глубокий исследовательский анализ данных, а также более тщательно подобрать гиперпараметры. Исследование может быть дополнено использованием других методов машинного обучения и ансамблевых моделей. Также важно обратить внимание на работу с большими объемами данных и оптимизацию производительности при масштабировании задач.

В итоге, выполнение этой работы дало ценный опыт работы с Apache Spark, применение методов анализа данных и машинного обучения на практике, что представляет собой важный вклад в профессиональное развитие в области обработки больших данных и искусственного интеллекта.

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Изучаем Spark: молниеносный анализ данных / Х. Карау, Э. Конвински, П. Венделл, М.М. Захария // ДМК Пресс, 2015. — 304 с.: ил.
2. Data Exploration // Learning Apache Spark with Python [Электронный ресурс] / W. Feng. - [2021]. - Режим доступа : <https://runawayhorse001.github.io/LearningApacheSpark/exploration.html> (дата обращ. 19.09.2022).
3. Advanced Pyspark for Exploratory Data Analysis [Электронный ресурс]. – [2022]. – Режим доступа : [https://www.kaggle.com/code/tientd95/advanced-  
   pyspark-for-exploratory-data-analysis](https://www.kaggle.com/code/tientd95/advanced-pyspark-for-exploratory-data-analysis) (дата обращ. 19.09.2022).
4. Exploratory Data Analysis (EDA) with PySpark on Databricks [Электронный ресурс]. – [2020]. – Режим доступа : <https://towardsdatascience.com/exploratory-data-analysis-eda-with-pyspark-on-databricks-e8d6529626b1> (дата обращ. 19.09.2022).
5. Exploratory data analysis with pySpark [Электронный ресурс]. – [2020]. – Режим доступа : <https://github.com/roshankoirala/pySpark_tutorial/blob/master/Exploratory_data_analysis_with_pySpark.ipynb> (дата обращ. 19.09.2022).
6. Официальный сайт Apache Spark [Электронный ресурс]. – [2022]. – Режим доступа : <https://spark.apache.org/> (дата обращ. 19.09.2022).
7. Уайт, Т. Hadoop: Подробное руководство / Т. Уайт. — 3-е изд. — СПб. : Питер, 2013. — 672 с.: ил.

# ПРИЛОЖЕНИЕ А – исходный код разведочного анализа датасета

# Import other modules not related to PySpark

!pip install pyspark\_dist\_explore

import os

import sys

import pandas as pd

from pandas import DataFrame

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.ticker as mtick

import matplotlib

from mpl\_toolkits.mplot3d import Axes3D

import math

from IPython.core.interactiveshell import InteractiveShell

from datetime import \*

import statistics as stats

from functools import reduce

import seaborn as sns

from typing import List

# This helps auto print out the items without explixitly using 'print'

InteractiveShell.ast\_node\_interactivity = "all"

%matplotlib inline

# Import PySpark related modules

import pyspark

from pyspark.rdd import RDD

from pyspark.sql import Row

from pyspark.sql import DataFrame

from pyspark.sql import SparkSession

from pyspark.sql import SQLContext

from pyspark.sql import functions as F

from pyspark.sql.functions import isnan, when, count, col, isnull

from pyspark.sql.types import IntegerType

from pyspark.ml.feature import VectorAssembler,Bucketizer

from pyspark.ml.stat import Correlation

from pyspark.sql.functions import lit, desc, col, size, array\_contains\

, isnan, udf, hour, array\_min, array\_max, countDistinct, expr

from pyspark.sql.types import \*

MAX\_MEMORY = '8G'

# Initialize a spark session.

conf = pyspark.SparkConf().setMaster("local[\*]") \

.set('spark.executor.heartbeatInterval', 10000) \

.set('spark.network.timeout', 10000) \

.set("spark.core.connection.ack.wait.timeout", "3600") \

.set("spark.executor.memory", MAX\_MEMORY) \

.set("spark.driver.memory", MAX\_MEMORY)

def init\_spark():

spark = SparkSession \

.builder \

.appName("Lab\_1\_SOBD") \

.config(conf=conf) \

.getOrCreate()

return spark

#Инициализация объекта SparkSession

spark = init\_spark()

#Определение пути к каталогу, где находится файл

directory\_path = "data/"

selected\_file = 'itineraries.csv'

# Чтение выбранного файла и создание DataFrame

# header - содержание в первой строке заголовков столбцов

# inferSchema - автоматическое определение типов

# sep - разделитель

dataframe = spark.read.csv(os.path.join(directory\_path, selected\_file), header=True, inferSchema=True, sep=',')

# Преобразование в Pandas Dataframe и вывод первых строк

dataframe.limit(5).toPandas().head()

# Вывод структуры Dataframe

dataframe.printSchema()

# Подсчет уникальных значений

# countDistinct - Подсчет уникальных значений для каждого столбца (с)

# \*() распаковка

# agg - принятие и выполнение выражений для датафрейма

uniq\_dataframe = dataframe.agg(\*(countDistinct(col(c)).alias(c) for c in dataframe.columns))

uniq\_dataframe.toPandas()

# Перевод времени полета в минуты

dataframe = dataframe.select('startingAirport', 'destinationAirport', 'isNonStop',

'isBasicEconomy', 'isRefundable', 'baseFare', 'totalFare',

'seatsRemaining', 'totalTravelDistance', 'travelDuration')

# Извлечение часов и минут из travelDuration

# SUBSTRING - выбор подстроки начиная с 3 элемента (отсекаем PT)

# SPLIT - массив строк начинающийся с часов/минут

# CAST - преобразование часов/минут в целочисленный тип

# withColumn - добавление столбца с названием часы/минуты

dataframe = dataframe.withColumn("hours", expr("CAST(SPLIT(SUBSTRING(travelDuration, 3), 'H')[0] AS INT)"))

dataframe = dataframe.withColumn("minutes", expr("CAST(SPLIT(SPLIT(SUBSTRING(travelDuration, 3), 'H')[1], 'M')[0] AS INT)"))

# Преобразование в минуты и запись в столбец travelDuration

dataframe = dataframe.withColumn("travelDuration", expr("hours \* 60 + minutes"))

dataframe = dataframe.drop("hours", "minutes")

dataframe.limit(5).toPandas()

dataframe.printSchema()

# Подсчет пропущенных значений

dataframe.select([col(column).isNull().cast("int").alias(column) for column in dataframe.columns]).groupBy().sum().show()

# Вывод количества строк до удаления пустых значений

dataframe.count()

# Вывод количества уникальных значений

dataframe.agg(\*(countDistinct(col(c)).alias(c) for c in dataframe.columns)).show()

# Удаление строк с пропущенными значениями

cleaned\_dataframe = dataframe.dropna()

# Вывод количество строк после удаления пустых значений

cleaned\_dataframe.count()

from pyspark.sql import functions as F

selected\_columns = ['baseFare', 'totalFare', 'totalTravelDistance', 'travelDuration']

# Фильтрация выбросов для каждого столбца

for column in selected\_columns:

# Рассчет квартилей - первый (0.25) и третий (0.75)

# 0.0 - максимально возможная точность

quartiles = cleaned\_dataframe.stat.approxQuantile(column, [0.25, 0.75], 0.0)

# Рассчет интерквартильного размаха между первым и третьим квартилем

IQR = quartiles[1] - quartiles[0]

# Определение границ для определения выбросов

lower\_bound = quartiles[0] - 1.5 \* IQR

upper\_bound = quartiles[1] + 1.5 \* IQR

# Фильтрация данных и подсчет значений ниже и выше границ выбросов

below\_quartile\_count\_before = cleaned\_dataframe.filter(col(column) < lower\_bound).count()

above\_quartile\_count\_before = cleaned\_dataframe.filter(col(column) > upper\_bound).count()

print(f"Столбец (было)'{column}': Ниже границы выбросов - {below\_quartile\_count\_before}, Выше границы выбросов - {above\_quartile\_count\_before}")

# Рассчет максимального и минимального значения до фильтрации

# collect - преобразование результатов операции из распределенного кластера в локальную структуру данных на драйвере PySpark

max\_value\_before = cleaned\_dataframe.agg(F.max(col(column))).collect()[0][0]

min\_value\_before = cleaned\_dataframe.agg(F.min(col(column))).collect()[0][0]

# Рассчет медианы до фильтрации, т.е. квантиль 50% (0.5) с максимальной точностью (0.0)

median\_value\_before = cleaned\_dataframe.approxQuantile(column, [0.5], 0.0)[0]

print(f"Столбец (было)'{column}': Максимальное значение - {max\_value\_before}, Минимальное значение - {min\_value\_before}, Медиана - {median\_value\_before}")

# Фильтрация данных - значения выше нижней границы и ниже верхней границы

cleaned\_dataframe = cleaned\_dataframe.filter((col(column) >= lower\_bound) & (col(column) <= upper\_bound))

# Фильтрация данных и подсчет значений ниже и выше границ выбросов

below\_quartile\_count\_after = cleaned\_dataframe.filter(col(column) < lower\_bound).count()

above\_quartile\_count\_after = cleaned\_dataframe.filter(col(column) > upper\_bound).count()

print(f"Столбец (стало) '{column}': Ниже границы выбросов - {below\_quartile\_count\_after}, Выше границы выбросов - {above\_quartile\_count\_after}")

# Рассчет максимального и минимального значения после фильтрации

max\_value\_after = cleaned\_dataframe.agg(F.max(col(column))).collect()[0][0]

min\_value\_after = cleaned\_dataframe.agg(F.min(col(column))).collect()[0][0]

# Рассчет медианы после фильтрации

median\_value\_after = cleaned\_dataframe.approxQuantile(column, [0.5], 0.0)[0]

print(f"Столбец (стало) '{column}': Максимальное значение - {max\_value\_after}, Минимальное значение - {min\_value\_after}, Медиана - {median\_value\_after}")

# Расчет статистических показателей

statistics\_summary = cleaned\_dataframe.select('baseFare', 'totalFare','totalTravelDistance','travelDuration').describe()

# Вывод сводной статистики

statistics\_summary.toPandas()

### Построение гистограммы для каждой выбранной колонки

# num\_buckets - количество бакетов, на которые разделяется диапазон значений в столбце

# step - шаг разделения для формирования бакета

# splits - границы бакетов - от -бесконечность до +бесконечность и промежуточные значения на основе шага step

from pyspark\_dist\_explore import hist

for column in selected\_columns:

fix, ax = plt.subplots()

hist(ax, cleaned\_dataframe.select(column), bins=20, color='green')

ax.set\_title(f'Histogram of {column}')

fix, ax = plt.subplots()

hist(ax, cleaned\_dataframe.select('travelDuration'), bins=20, color='green')

ax.set\_title(f'Histogram of travelDuration')

selected\_features = ['startingAirport', 'destinationAirport', 'isNonStop',

'isBasicEconomy', 'isRefundable',

'seatsRemaining']

fig, axs = plt.subplots(len(selected\_features), 1, figsize=(8, 4 \* len(selected\_features)))

# Построение гистограмм для каждого выбранного признака

for i, feature in enumerate(selected\_features):

# Сгруппировать по признаку и подсчитать количество

data\_grouped = cleaned\_dataframe.groupBy(feature).count().collect()

# Извлечение данных для построения графика

categories = [row[0] for row in data\_grouped]

counts = [row[1] for row in data\_grouped]

axs[i].bar(categories, counts, color='green', edgecolor='black')

axs[i].set\_title(f'Распределение {feature}')

axs[i].set\_xlabel(feature)

axs[i].set\_ylabel('Кол-во раз')

# Регулировка расположения подграфиков

plt.tight\_layout()

# Отображение графика

plt.show()

# Создаем VectorAssembler для объединения признаков в один вектор

# vector\_assembler - трансформатор, объединяющий столбцы feature\_cols в вектор-столбец

feature\_cols = ['baseFare', 'totalFare','totalTravelDistance', 'travelDuration']

vector\_assembler = VectorAssembler(inputCols=feature\_cols, outputCol="features")

# Применяем VectorAssembler к данным

data = vector\_assembler.transform(cleaned\_dataframe)

# Вычисляем матрицу корреляции

correlation\_matrix = Correlation.corr(data, "features").head()

correlation\_array = correlation\_matrix[0].toArray()

# Выводим матрицу корреляции

correlation\_df = pd.DataFrame(correlation\_array, columns=feature\_cols)

# Вычисление корреляционной матрицы

correlation\_df.corr()

# Тепловая карта корреляции с использованием Seaborn

plt.figure(figsize=(12, 8))

sns.heatmap(correlation\_df.corr(), annot=True, cmap="RdBu", fmt=".2f", linewidths=.5)

plt.title('Матрица корреляции признаков')

plt.show()

cleaned\_dataframe.coalesce(1).write.options(header='True', delimiter=';').csv('new\_csv')

# ПРИЛОЖЕНИЕ Б – исходный код машинного обучения на больших данных

import numpy as np

import pandas as pd

import os

from pyspark.sql.types import \*

from pyspark.sql.functions import \*

from pyspark.sql import SparkSession

from pyspark.ml import Pipeline

from pyspark.ml.feature import VectorAssembler, StringIndexer, VectorIndexer, MinMaxScaler, OneHotEncoder

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.tuning import ParamGridBuilder, CrossValidator

from pyspark.ml.evaluation import BinaryClassificationEvaluator, RegressionEvaluator

from pyspark.ml.regression import RandomForestRegressor

spark = SparkSession.builder.master("local[\*]").getOrCreate()

dataframe = spark.read.csv('new\_csv/itineraries2.csv', inferSchema=True, header=True, sep=';')

dataframe.limit(5).toPandas()

data = dataframe.select(

"startingAirport",

"destinationAirport",

"isNonStop",

"isBasicEconomy",

"isRefundable",

"baseFare",

"totalFare",

"seatsRemaining",

"totalTravelDistance",

((col("travelDuration")).alias("label")))

data.show(10)

splits = data.randomSplit([0.7, 0.3])

train = splits[0]

test = splits[1].withColumnRenamed("label", "trueLabel")

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

## Создание столбца признаков для задачи регрессии

# StringIndexer - преобразование текстовых значений в категориальных столбцах в числовые индексы

# OneHotEncoder - OHE-кодирование числовых индексов от StringIndexer

# VectorAssembler - объединение всех категориальных признаков в один вектор

# VectorIndexer - автоматическое определение категориальных признаков вектора

# MinMaxScaler - масштабирование числовых признаков в диапазон от 0 до 1

strIdx = StringIndexer(inputCols=["startingAirport", "destinationAirport"], outputCols=["startingAirport\_idx", "destinationAirport\_idx"])

oneHotEnc = OneHotEncoder(inputCols=["startingAirport\_idx", "destinationAirport\_idx"], outputCols=["startingAirport\_enc", "destinationAirport\_enc"])

catVect = VectorAssembler(inputCols=["startingAirport\_enc", "destinationAirport\_enc", "isNonStop", "isBasicEconomy", "isRefundable", "seatsRemaining"], outputCol="reg\_catFeatures")

catIdx = VectorIndexer(inputCol=catVect.getOutputCol(), outputCol="reg\_idxCatFeatures")

numVect = VectorAssembler(inputCols=["baseFare", "totalFare", "totalTravelDistance"], outputCol="reg\_numFeatures")

minMax = MinMaxScaler(inputCol=numVect.getOutputCol(), outputCol='reg\_normFeatures')

featVect = VectorAssembler(inputCols=["reg\_idxCatFeatures", "reg\_numFeatures", "reg\_normFeatures"], outputCol="reg\_features")

# Создание модели RandomForestRegressor

rf = RandomForestRegressor(labelCol="label", featuresCol="reg\_features", numTrees=10)

# Создание и выполнение пайплайна для задачи регрессии

reg\_pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, catIdx, numVect, minMax, featVect, rf])

reg\_model = reg\_pipeline.fit(train)

reg\_prediction = reg\_model.transform(test)

# Вывод результатов

reg\_prediction.select("reg\_features", "prediction", "trueLabel").show(10, truncate=False)

# RMSE - среднеквадратичное отклонение между фактическими и предсказанными значениями

reg\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")

rmse = reg\_evaluator.evaluate(reg\_prediction)

print("RMSE = ", rmse)

# R² - доля общей дисперсии

reg\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="r2")

r2 = reg\_evaluator.evaluate(reg\_prediction)

print("R² = ", r2)

# MAE - абсолютное отклонение между фактическими и предсказанными значениями

mae\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="mae")

mae = mae\_evaluator.evaluate(reg\_prediction)

print(f"Mean Absolute Error (MAE): {mae}")

# Стандартное отклонение

data.select(stddev('label')).show()

## Настройка гиперпараметров модели

# maxDepth - максимальная глубина дерева, выбор наилучшего 2,5,10

# numTrees - количество деревьев в случайном лесу, выбор наилучшего из 5,10,20

param\_grid = (ParamGridBuilder()

.addGrid(rf.maxDepth, [2, 5, 10])

.addGrid(rf.numTrees, [5, 10, 20])

.build())

## Создание объекта для кросс-валидации

# estimator - оцениваемая модель (наш пайплайн c моделью и преобразованием данных)

# estimatorParamMaps - сетка параметров для оцениваемой модели

# evaluator - оценка качества модели на каждом этапе кросс-валидации (по умолчанию RMSE)

# numFolds - количество складываний для кросс-валидации, т.е. на сколько частей будет разделен набор данных (в нашем случае разделение на 3 части и обучение 3 раза)

cv = CrossValidator(estimator=reg\_pipeline,

estimatorParamMaps=param\_grid,

evaluator=RegressionEvaluator(),

numFolds=3)

# Обучение модели с использованием кросс-валидации

cv\_model = cv.fit(train)

# Получение наилучшей модели после обучения с кросс-валидацией

best\_cv\_model = cv\_model.bestModel

# Оценка производительности на тестовом наборе

cv\_prediction = best\_cv\_model.transform(test)

reg\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="rmse")

rmse = reg\_evaluator.evaluate(cv\_prediction)

print("RMSE = ", rmse)

# Расчет метрик для задачи регрессии

reg\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="r2")

r2 = reg\_evaluator.evaluate(cv\_prediction)

print("R² = ", r2)

mae\_evaluator = RegressionEvaluator(labelCol="trueLabel", predictionCol="prediction", metricName="mae")

mae = mae\_evaluator.evaluate(cv\_prediction)

print(f"Mean Absolute Error (MAE): {mae}")

# 0 - для рейса с длительностью полета менее 400 минут,

# 1 - для рейса с длительностью полета более 400 минут

data = dataframe.select(

"startingAirport",

"destinationAirport",

"isNonStop",

"isBasicEconomy",

"isRefundable",

"baseFare",

"totalFare",

"seatsRemaining",

"totalTravelDistance",

((col("travelDuration") > 375).cast("Int").alias("label")))

data.show(10)

splits = data.randomSplit([0.7, 0.3])

train = splits[0]

test = splits[1].withColumnRenamed("label", "trueLabel")

train\_rows = train.count()

test\_rows = test.count()

print("Training Rows:", train\_rows, " Testing Rows:", test\_rows)

positive\_count = train.filter(col("label") == 1).count()

negative\_count = train.filter(col("label") == 0).count()

balance\_ratio = positive\_count / negative\_count

print("Positive to Negative Class Ratio:", balance\_ratio)

# Создание столбца признаков для задачи регрессии

strIdx = StringIndexer(inputCols=["startingAirport", "destinationAirport"], outputCols=["startingAirport\_idx", "destinationAirport\_idx"])

oneHotEnc = OneHotEncoder(inputCols=["startingAirport\_idx", "destinationAirport\_idx"], outputCols=["startingAirport\_enc", "destinationAirport\_enc"])

catVect = VectorAssembler(inputCols=["startingAirport\_enc", "destinationAirport\_enc", "isNonStop", "isBasicEconomy", "isRefundable", "seatsRemaining"], outputCol="reg\_catFeatures")

catIdx = VectorIndexer(inputCol=catVect.getOutputCol(), outputCol="reg\_idxCatFeatures")

numVect = VectorAssembler(inputCols=["baseFare", "totalFare", "totalTravelDistance"], outputCol="reg\_numFeatures")

minMax = MinMaxScaler(inputCol=numVect.getOutputCol(), outputCol='reg\_normFeatures')

featVect = VectorAssembler(inputCols=["reg\_idxCatFeatures", "reg\_numFeatures", "reg\_normFeatures"], outputCol="features")

lr = LogisticRegression(labelCol="label",featuresCol="features",maxIter=10,regParam=0.3)

pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, catIdx, numVect, minMax, featVect, lr])

pipelineModel = pipeline.fit(train)

prediction = pipelineModel.transform(test)

predicted = prediction.select("features", "prediction", "trueLabel")

predicted.show(100, truncate=False)

tp = float(predicted.filter("prediction == 1.0 AND truelabel == 1").count())

fp = float(predicted.filter("prediction == 1.0 AND truelabel == 0").count())

tn = float(predicted.filter("prediction == 0.0 AND truelabel == 0").count())

fn = float(predicted.filter("prediction == 0.0 AND truelabel == 1").count())

pr = tp / (tp + fp)

re = tp / (tp + fn)

metrics = spark.createDataFrame([

("TP", tp),

("FP", fp),

("TN", tn),

("FN", fn),

("Precision", pr),

("Recall", re),

("F1", 2\*pr\*re/(re+pr))],["metric", "value"])

metrics.show()

# Метрика Area Under the ROC Curve

evaluator = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="rawPrediction", metricName="areaUnderROC")

aur = evaluator.evaluate(prediction)

print ("AUR = ", aur)

## Настройка гиперпараметров

# regParam - коэфф регуляции

# maxIter - максимальное кол-во итераций при обучении

# threshold порог классификации

paramGrid = (ParamGridBuilder()

.addGrid(lr.regParam, [0.3, 0.2])

.addGrid(lr.maxIter, [15, 10])

.addGrid(lr.threshold,[0.45, 0.35])

.build())

cv = CrossValidator(estimator=pipeline,

evaluator=BinaryClassificationEvaluator(),

estimatorParamMaps=paramGrid,

numFolds=2)

model = cv.fit(train)

newPrediction = model.transform(test)

newPredicted = prediction.select("features", "prediction", "trueLabel")

newPredicted.show()

# Recalculate confusion matrix

tp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel == 1").count())

fp2 = float(newPrediction.filter("prediction == 1.0 AND truelabel == 0").count())

tn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel == 0").count())

fn2 = float(newPrediction.filter("prediction == 0.0 AND truelabel == 1").count())

pr2 = tp2 / (tp2 + fp2)

re2 = tp2 / (tp2 + fn2)

metrics2 = spark.createDataFrame([

("TP", tp2),

("FP", fp2),

("TN", tn2),

("FN", fn2),

("Precision", pr2),

("Recall", re2),

("F1", 2\*pr2\*re2/(re2+pr2))],["metric", "value"])

metrics2.show()

# Recalculate the Area Under ROC

evaluator2 = BinaryClassificationEvaluator(labelCol="trueLabel", rawPredictionCol="prediction", metricName="areaUnderROC")

aur2 = evaluator.evaluate(prediction)

print("AUR2 = ", aur2)