Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Факультет Электроники и вычислительной техники

Кафедра Электронно-вычислительные машины и системы

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
к курсовой работе (проекту)

по дисциплине Системы обработки больших данных

на тему Исследование датасета авиабилетов из Expedia с использованием фреймворка Apache Spark

Студент Плотников Иван Николаевич
(фамилия, имя, отчество)

Группа САПР-1.4

Руководитель работы (проекта) _____ П.Д. Кравченя
(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.

Министерство науки и высшего образования Российской Федерации

Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Волгоградский государственный технический университет»

Факультет Электроники и вычислительной техники

Направление (специальность) 09.04.01 – Информатика и вычислительная техника

Кафедра Электронно-вычислительные машины и системы

Дисциплина Системы обработки больших данных

Утверждаю
Зав. кафедрой Андреев А.Е.

«_____» _____ 20 ___ г.

ЗАДАНИЕ
на курсовую работу (проект)

Студент Плотников Иван Николаевич
(фамилия, имя, отчество)

Группа САПР-1.4

1. Тема: Исследование датасета авиабилетов из Expedia с использованием фреймворка Apache Spark

Утверждена приказом от «_____» _____ 20___ г. № _____

2. Срок представления работы (проекта) к защите «___»_____20__ г.

3. Содержание расчетно-пояснительной записки: Разведочный анализ данных, машинное обучение на больших данных

4. Перечень графического материала: _____

5. Дата выдачи задания «_____» _____20 ___ г.

Руководитель работы (проекта)_____ П.Д. Кравченя
подпись, дата — инициалы и фамилия

Задание принял к исполнению_____ И.Н. Плотников
подпись, дата — инициалы и фамилия

СОДЕРЖАНИЕ

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 освоены 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 В PYSPARK

1.1 Постановка задачи

Выполнить разведочный анализ датасета авиабилетов из Expedia согласно варианту с определением: типов признаков в датасете; пропущенных значений и их устранением; выбросов и их устранением; расчетом статистических показателей признаков (средних, квартилей и т.д.); визуализацией распределения наиболее важных признаков; корреляций между признаками.

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.

Данные представляют собой значения следующих типов данных: integer, double, string, date, boolean. Типы данных представлены на рисунке 1.

Числовые признаки: legId, travelDuration, elapsedDays, baseFare, totalFare, seatsRemaining, totalTravelDistance, segmentsDurationInSeconds

Категориальные признаки: startingAirport, destinationAirport, fareBasisCode, segmentsArrivalAirportCode,segmentsDepartureAirportCode,segmentsAirlineName, segmentsAirlineCode, segmentsEquipmentDescription, segmentsDistance, segmentsCabinCode

Бинарные признаки: isBasicEconomy, isRefundable, isNonStop

	Column Name	Data type
0	legId	string
1	searchDate	date
2	flightDate	date
3	startingAirport	string
4	destinationAirport	string
5	fareBasisCode	string
6	travelDuration	string
7	elapsedDays	int
8	isBasicEconomy	boolean
9	isRefundable	boolean
10	isNonStop	boolean
11	baseFare	double
12	totalFare	double
13	seatsRemaining	int
14	totalTravelDistance	int
15	segmentsDepartureTimeEpochSeconds	string
16	segmentsDepartureTimeRaw	string
17	segmentsArrivalTimeEpochSeconds	string
18	segmentsArrivalTimeRaw	string
19	segmentsArrivalAirportCode	string
20	segmentsDepartureAirportCode	string
21	segmentsAirlineName	string
22	segmentsAirlineCode	string
23	segmentsEquipmentDescription	string
24	segmentsDurationInSeconds	string
25	segmentsDistance	string
26	segmentsCabinCode	string

Рисунок 1 – Типы данных в датасете

С помощью представленного ниже кода проводим разведку на предмет количества уникальных значений для каждого столбца в датасете:

```
df.agg(*(countDistinct(col(c)).alias(c) for c in df.columns)).show()
```

Так как булев столбец isRefundable в нашей выборке имеет только одно уникальное значение, то его можно удалить из датасета. Уникальные значения для каждого столбца приведены на рисунке 2.

```
+------------------+------------+--------------+-----------+--------------+------------+---------+
|destinationAirport|fareBasisCode|travelDuration|elapsedDays|isBasicEconomy|isRefundable|isNonStop|
+------------------+------------+--------------+-----------+--------------+------------+---------+
|                16|        4463|          1654|          3|             2|           1|        2|
+------------------+------------+--------------+-----------+--------------+------------+---------+
```

Рисунок 2 – Уникальные значения для каждого столбца

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.

1. Для числовых колонок, допускающих значения ноль, проверим на None и NaN;
2. Для числовых колонок, недопускающих значения ноль, проверим на нули, None и NaN;
3. Для булевых колонок проверим на None и null;
4. Для колонок с датами проверим на None и null.

Были найдены столбцы, содержащие пропущенные значения, а также количество таковых в конкретных столбцах. Программный код для поиска и удаления пустых и нулевых значений представлен ниже:

```
# Инициализация словаря для хранения количества пропущенных значений для каждого столбца
missing_values = {}
# Итерация по столбцам DataFrame и подсчет пропущенных значений для каждого типа
for index, column in enumerate(df.columns):
    if column in string_columns: # check None and Null
        missing_count = df.filter(col(column).eqNullSafe(None) |
        col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in numeric_with_zeroes_columns: # check None, NaN and Null
        missing_count = df.filter(col(column) == None |
        isnan(col(column)) | col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in numeric_without_zeroes_columns:  # check zeroes, None, NaN and Null
        missing_count = df.filter(col(column).isin([0,None]) |
```

```
                isnan(col(column)) | col(column).isNull()).count()
                missing_values.update({column:missing_count})
        if column in boolean_columns:  # check None and Null
                missing_count = df.filter(col(column).eqNullSafe(None) |
                col(column).isNull()).count()
                missing_values.update({column:missing_count})
        if column in date_columns:  # check None and Null
                missing_count = df.filter(col(column).eqNullSafe(None) |
                col(column).isNull()).count()
                missing_values.update({column:missing_count})
# Создание DataFrame из словаря missing_values
missing_df = pd.DataFrame.from_dict([missing_values])
missing_df
# Определение столбцов с пропущенными значениями
columns_with_missing_values = []
for column in missing_df:
        if missing_df[column].values[0] != 0:
        columns_with_missing_values.append(column)
# Вывод информации о пропущенных значениях
missing_df[columns_with_missing_values]
print(missing_values)
```

1.4 Определение и удаление выбросов

Для удаления выбросов их необходимо определить. Для определения выбросов через квартили можно воспользоваться методом межквартильного размаха (IQR). Для этого нужно выполнить следующие шаги:

1. Найти первый (Q1) и третий (Q3) квартили данных. Для этого применяется функция approxQuantile в Apache Spark (конкретного процентного значения) в наборе данных.

2. Вычислить межквартильный размах (IQR) как разницу между Q3 и Q1: IQR = Q3 - Q1.

3. Определить нижнюю границу выбросов как Q1 - 1.5 * IQR.

4. Определить верхнюю границу выбросов как Q3 + 1.5 * IQR.

5. Любое значение, которое меньше нижней границы или больше верхней границы, считается выбросом.

Квартили — это значения, которые делят упорядоченный набор данных на три равные части. Они показывают распределение данных и включают в себя:

1. Первый квартиль (Q1), который отделяет первые 25% данных;

2. Второй квартиль (Q2), который также известен как медиана, отделяет первые 50% данных;

3. Третий квартиль (Q3), который отделяет первые 75% данных;

Программный код для поиска квартилей представлен ниже:

```
from pyspark.sql import functions as F
selected_columns = ['baseFare', 'totalFare', 'totalTravelDistance']
# Фильтрация выбросов для каждого столбца
for column in selected_columns:
    # Рассчет квартилей
    quartiles = cleaned_dataframe.stat.approxQuantile(column, [0.25, 0.75], 0.0)
    # Рассчет межквартильного размаха
    IQR = quartiles[1] - quartiles[0]
    # Определение границ выбросов как первый квартиль - 1.5 * межквартильный размах
    lower_bound = quartiles[0] - 1.5 * IQR
    upper_bound = quartiles[1] + 1.5 * IQR
    # Фильтрация данных и подсчет выбросов после фильтрации
    # Подсчитываются значения, находящиеся ниже и выше границ межквартильного размаха после фильтрации
    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 (до)'{column}': Снизу выбросов -
        {below_quartile_count_before}, Сверху выбросов -
        {above_quartile_count_before}")
        # Рассчет максимального и минимального значения до фильтрации
        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
        median_value_before = cleaned_dataframe.approxQuantile(column,
        [0.5], 0.0)[0]
        print(f"Столбец (до) '{column}': Максимальное значение -
        {max_value_before}, Минимальное значение - {min_value_before},
        Медиана - {median_value_before}")
        # Фильтрация данных
        cleaned_dataframe = cleaned_dataframe.filter((col(column) >=
        lower_bound) & (col(column) <= upper_bound))
        # Фильтрация данных и подсчет выбросов после фильтрации
        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 (после)'{column}': Снизу выбросов -
        {below_quartile_count_after}, Сверху 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 (после) '{column}': Максимальное значение -
{max_value_after}, Минимальное значение - {min_value_after},
Медиана - {median_value_after}")
```

Результат поиска выбросов приведен на рисунке 3. Как видно из рисунка, выбросы имеется в колонках “totalFare”, “baseFare”.

```
Столбец (до)'baseFare': Снизу выбросов - 0, Сверху выбросов - 4783
Столбец (до) 'baseFare': Максимальное значение - 7344.19, Минимальное значение - 13.37, Медиана - 206.87
Столбец (после)'baseFare': Снизу выбросов - 0, Сверху выбросов - 0
Столбец (после) 'baseFare': Максимальное значение - 682.79, Минимальное значение - 13.37, Медиана - 202.79
Столбец (до)'totalFare': Снизу выбросов - 0, Сверху выбросов - 703
Столбец (до) 'totalFare': Максимальное значение - 771.3, Минимальное значение - 28.97, Медиана - 238.6
Столбец (после)'totalFare': Снизу выбросов - 0, Сверху выбросов - 0
Столбец (после) 'totalFare': Максимальное значение - 735.71, Минимальное значение - 28.97, Медиана - 238.6
Столбец (до)'totalTravelDistance': Снизу выбросов - 0, Сверху выбросов - 0
Столбец (до) 'totalTravelDistance': Максимальное значение - 4421, Минимальное значение - 89, Медиана - 1517.0
Столбец (после)'totalTravelDistance': Снизу выбросов - 0, Сверху выбросов - 0
Столбец (после) 'totalTravelDistance': Максимальное значение - 4421, Минимальное значение - 89, Медиана - 1517.0
```

Рисунок 3 – Результаты поиска выбросов

1.5 Расчет статистических показателей признаков и их визуализация

Рассчитаем такие показатели, как:

Минимальное, среднее и максимальное значения;

Среднеквадратичное отклонение;

Квартили;

1. Минимальное, среднее и максимальное значения:
   1.1 Минимальное значение — это наименьшее число в наборе данных.
   1.2 Среднее значение (или среднее арифметическое) — это сумма всех чисел в наборе, деленная на их количество.
   1.3 Максимальное значение — это наибольшее число в наборе данных.

2. Среднеквадратичное отклонение (или стандартное отклонение). Стандартное отклонение показывает, насколько в среднем значения в наборе отличаются от среднего значения.
3. Квартили — это значения, которые делят упорядоченный набор данных на три равные части. Они показывают распределение данных и включают в себя:

- Первый квартиль (Q1), который отделяет первые 25% данных;
- Второй квартиль (Q2), который также известен как медиана, отделяет первые 50% данных;
- Третий квартиль (Q3), который отделяет первые 75% данных;

Программный код представлен ниже:

```
# Минимум, максимум и среднеее
def calculate_min_mean_max_statistic_indicators(df, column):
                                                    min_value =
df.agg(min(column).alias(f'min_{column}')).collect()[0][f'min_{column}'
]
                                                mean_value =
df.agg(mean(column).alias(f'mean_{column}')).collect()[0][f'mean_{colu
mn}']
                                                    max_value =
df.agg(max(column).alias(f'max_{column}')).collect()[0][f'max_{column
}']
    return (min_value, mean_value, max_value)

# Среднеквадратическое отклонение
def calculate_stdev(df, column):
                                                stddev_value =
df.agg(stddev(column).alias(f'stddev_{column}')).collect()[0][f'stddev_{c
olumn}']
    return stddev_value

# Квартили
def calculate_quartiles(df, column):
    q1, median, q3 = df.approxQuantile(column, [0.25, 0.5, 0.75], 0.01)
    return q1, median, q3

# Вывод
def calculate_statistical_indicators(df, column):
```

```
                                    min_value, mean_value, max_value
=calculate_min_mean_max_statistic_indicators(df, column)
    stddev_value = calculate_stdev(df, column)
    q1, median, q3 = calculate_quartiles(df, column)
     return  (min_value, mean_value, max_value, stddev_value, q1, median,
q3)
statistical_indicators = {}

for col in numeric_with_zeroes_columns:
                                                statistical_indicators[col]          =
calculate_statistical_indicators(cleaned_dataframe, col)

for col in numeric_without_zeroes_columns:
                                                statistical_indicators[col]          =
calculate_statistical_indicators(cleaned_dataframe, col)

for key, value in statistical_indicators.items():
    print(f"""
        Column: {key}
          -- Min: {value[0]}
          -- Mean: {value[1]}
          -- Max: {value[2]}
          -- Stddev: {value[3]}
          -- q1: {value[4]}
          -- q2(median): {value[5]}
          -- q3: {value[6]}
        """)
```

Расчет статических показателей произведен на рисунке 4.

```
Column: elapsedDays
  -- Min: 0
  -- Mean: 0.1452832657333508
  -- Max: 1
  -- Stddev: 0.3523867826809697
  -- q1: 0.0
  -- q2(median): 0.0
  -- q3: 0.0


Column: seatsRemaining
  -- Min: 0
  -- Mean: 6.959181604082167
  -- Max: 10
  -- Stddev: 2.1560235108981853
  -- q1: 7.0
  -- q2(median): 7.0
  -- q3: 9.0


Column: baseFare
  -- Min: 13.37
  -- Mean: 238.62285032056624
  -- Max: 662.34
  -- Stddev: 138.72446019414505
  -- q1: 124.65
  -- q2(median): 200.0
  -- q3: 336.74
```

Рисунок 4 – Расчет статических показателей

Из приведенного анализа можно сделать вывод о количестве элементов, мин. и макс. значении, среднее и среднеквадратичное отклонение.

Для визуализации распределения наиболее важных признаков были использованы следующие графики:

1. Гистограммы числовых признаков, пример которых иллюстрирует рисунок 5;
2. Круговая диаграмма категориальных признаков, пример изображён на рисунке 6;
3. Гистограммы количества встречающихся показателей категориальных и бинарных признаков на рисунке 7;

Код для построения гистограммы числовых признаков:

```
def plot_histogram(df, column, bins='auto', figsize=(10, 6)):
```

```
    data = df.select(column).rdd.flatMap(lambda x: x).collect()
    plt.figure(figsize=figsize)
    plt.hist(data, bins=bins, color='blue')
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
```

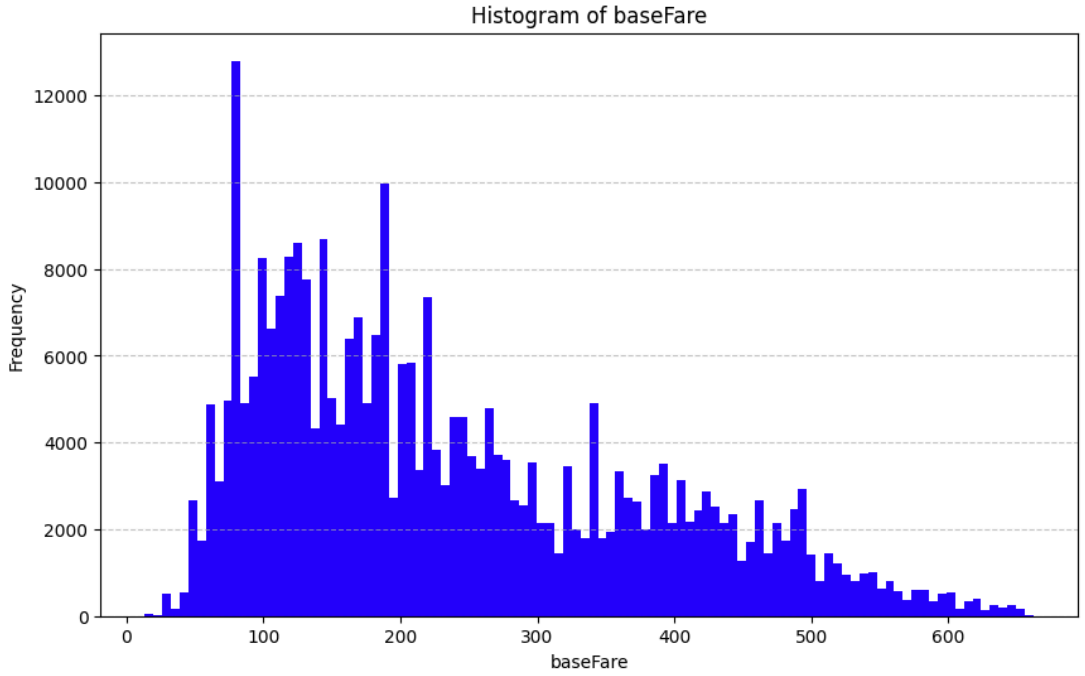

Рисунок 5 – Визуализация гистограммы распределения базовой стоимости

Код для построения круговой диаграммы, отрисованной на рисунке 6:

```
def pie_chart(df, column):
    pandas_df = df.groupby(column).count().toPandas()
    pandas_df = pandas_df.set_index(pandas_df.columns[0])
        fig, ax = plt.subplots(figsize=(12, 7), subplot_kw=dict(aspect='equal'),
dpi=120)
    data = pandas_df['count']
    categories = pandas_df.index
    plt.pie(data, labels = categories, autopct="%1.1f%%")
```

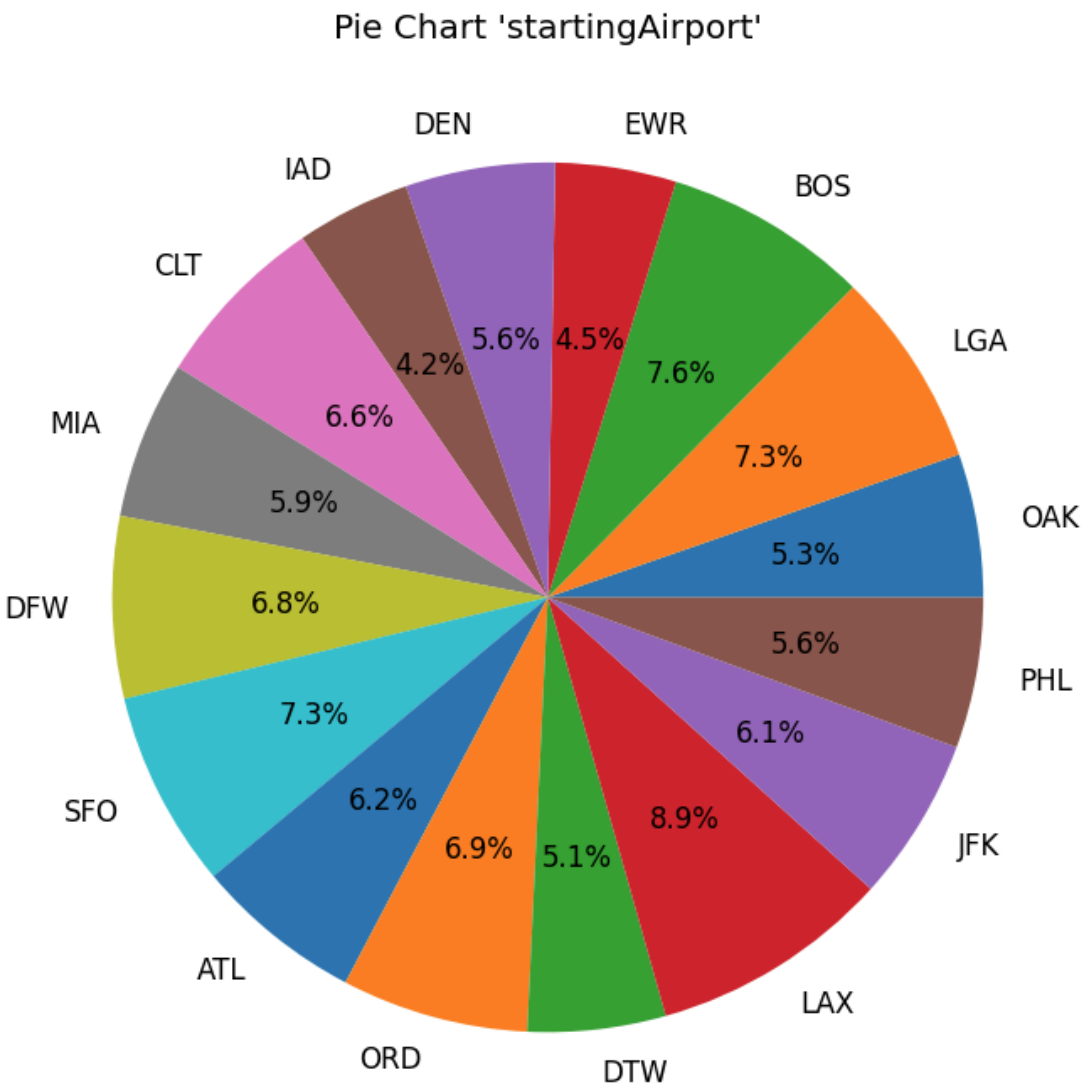

Рисунок 6 – Визуализация круговой диаграммы распределения индексов аэропортов взлета.

Код для построения гистограммы категориальных и бинарных признаков:

```
# Определение списка признаков для построения гистограмм для категориальных и бинарных признаков
selected_features = ['startingAirport', 'destinationAirport', 'isNonStop',
                     'isBasicEconomy',
                     'seatsRemaining']

# Создание фигуры и массива подграфиков
fig, axs = plt.subplots(len(selected_features), 1, figsize=(8, 3 * 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]
        # Построение гистограммы для текущего признака на соответствующем подграфике
    axs[i].bar(categories, counts, color='darkgreen', edgecolor='black')
    axs[i].set_title(f'Distribution of {feature}')
    axs[i].set_xlabel(feature)
    axs[i].set_ylabel('Count')

# Регулировка расположения подграфиков
plt.tight_layout();

# Отображение графика
plt.show();
```

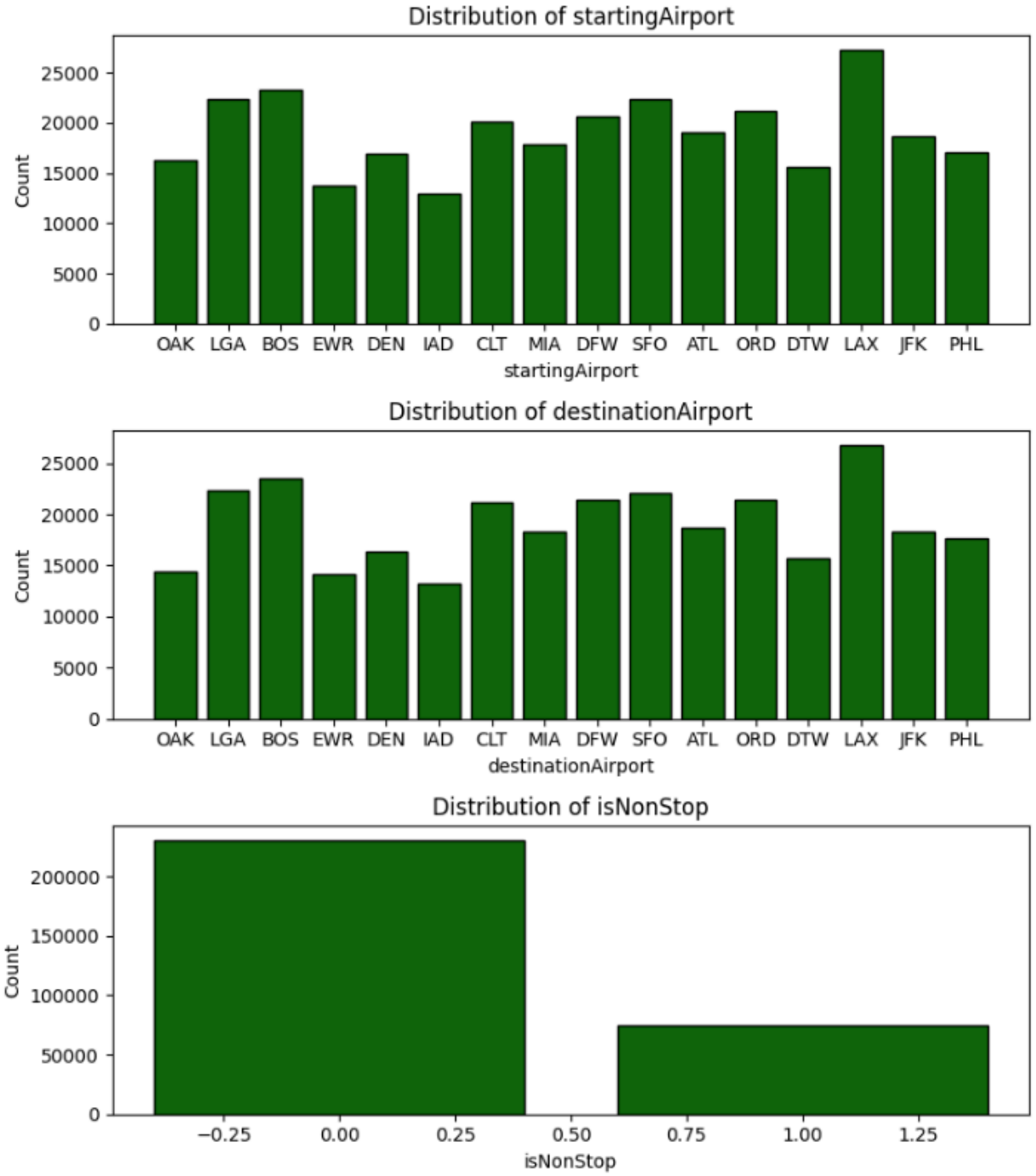

Рисунок 6– Визуализация гистограмм категориальных и бинарных признаков.

1.6 Корреляции между признаками

Чтобы выявить связи между признаками, можно построить матрицу корреляций. Создается переменная vector_col, которая представляет собой имя столбца, в который будут сохраняться векторы признаков. Создается список numeric_columns, который содержит все числовые столбцы из двух предоставленных списков. Используется VectorAssembler из библиотеки PySpark для создания векторов признаков из числовых столбцов. inputCols указывает на входные столбцы, а outputCol указывает на столбец, в который будут сохранены векторы .Происходит применение VectorAssembler к DataFrame cleaned_df, и из результата выбирается только столбец с векторами признаков. Используется метод corr из объекта Correlation для вычисления матрицы корреляции между векторами признаков. Преобразование полученной матрицы корреляции в список списков (матрицу) с помощью методов toArray и tolist. Создание DataFrame на основе полученной матрицы корреляции с использованием библиотеки Pandas. Столбцы и индексы DataFrame соответствуют числовым признакам. Создание объекта фигуры для графика с заданным размером. Использование библиотеки Seaborn для создания тепловой карты (heatmap) на основе матрицы корреляции. Оси X и Y меток обозначают числовые признаки, а цветовая шкала показывает уровень корреляции между признаками. Параметр annot=True включает отображение числовых значений в ячейках тепловой карты. Матрица корреляций показана на рисунке 7. Из корреляционной матрицы видно, что baseFare и totalFare демонстрируют тесную положительную взаимосвязь, поскольку цена билета всегда включает одну и ту же сумму налогов. Цена билета и расстояние в километрах имеют умеренную положительную связь, поскольку цена билета зависит от расстояния.

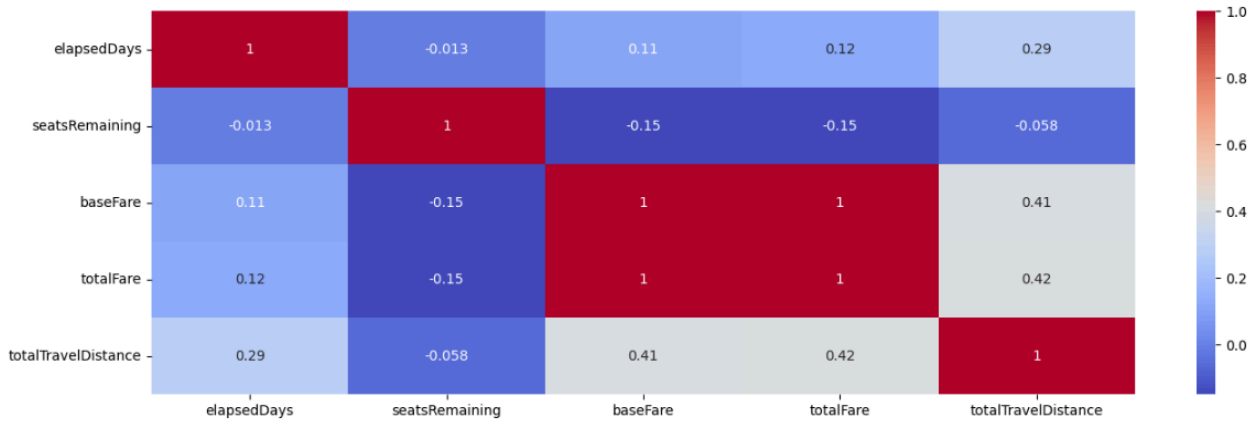

Рисунок 8 – Матрица корреляций

1.7 Выводе о работе

В ходе выполнения работы был выполнен разведочный анализ данных с помощью инструментов Apache Spark, которые он предоставляет для обработки больших данных. Проведенная работа заключается в определении типов признаков датасета, устранении пропущенных значений и выбросов, расчете статистических показателей и визуализации распределения признаков и их корреляции. Полный код проделанной работы представлен в Приложении А.

2 МАШИННОЕ ОБУЧЕНИЕ НА БОЛЬШИХ ДАННЫХ

2.1 Постановка задачи

Цель - предсказать значение 'totalTravelDistance' на основе входных признаков. Для этого используется набор данных, в котором выбраны следующие признаки:

1. 'startingAirport' - код аэропорта вылета
2. 'destinationAirport' - код аэропорта прилета
3. 'elapsedDays' - сколько дней занял полет
4. 'isBasicEconomy' (приведен к типу Int) - базовый ли билет
5. 'isNonStop' (приведен к типу Int) - прямой ли рейс
6. 'baseFare' - базовая стоимость билета
7. 'totalFare' - итоговая стоимость билета
8. 'totalTravelDistance' - пройденное за рейс расстояние

Эти признаки будут использованы для построения модели линейной регрессии, которая позволит предсказать значение 'label' на основе входных данных.

Задача: выяснить, превышает ли путь самолета 1500 миль.

Целевая переменная (label) будет равна 1, если рейс пролетел более среднего расстояния 1500 миль, и 0, если рейс пролетел менее этого расстояния.

Признаки (features) будут включать следующие столбцы: 'startingAirport', 'destinationAirport', 'elapsedDays', 'isBasicEconomy', 'isNonStop', 'baseFare', и 'totalFare'.

Критерии: используем метрику AUC-ROC для оценки качества модели

2.2 Подготовка данных для модели регрессии

При выполнении разведочного анализа были определены признаки, имеющие наибольшую корреляцию, возьмем их для последующего анализа.

Для построения модели регрессии выберем прогнозируемую переменную 'totalTravelDistance' с новым псевдонимом label из очищенного датафрейма cleanded_df:

```
data = cleaned_dataframe.select(
    'startingAirport',
    'destinationAirport',
    'elapsedDays',
    col('isBasicEconomy').cast('Int').alias('isBasicEconomy'),
    col('isNonStop').cast('Int').alias('isNonStop'),
    'baseFare',
    'totalFare',
    col('totalTravelDistance').alias('label')
)
```

Для дальнейшего обучения данные необходимо разделить на обучающий набор данных для нашей модели и тестовый. Было решено разделить данные на 80% тренировочных и 20% тестовых. Данные были распределены случайным образом с помощью метода randomSplit([0.8, 0.2])

```
splits = data.randomSplit([0.8, 0.2])
train = splits[0]
test = splits[1].withColumnRenamed('label', 'trueLabel')
```

2.3 Обучение модели линейной регрессии

Обычно алгоритмы машинного обучения показывают лучшие результаты и сходятся быстрее, когда различные признаки (переменные) имеют меньший масштаб. Поэтому перед обучением моделей машинного обучения данные обычно нормализуются [1]. Для этого данные сначала преобразуются в единый вектор при помощи:

```
numVect = VectorAssembler(inputCols = ['baseFare', 'totalFare', 'elapsedDays'], outputCol='numFeatures')
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(), outputCol='normFeatures')
```

featVect = VectorAssembler(inputCols=['catFeatures', 'normFeatures'], outputCol='features')

Для задачи регрессии был применен алгоритм линейной регрессии, где в качестве предсказываемого признака был установлен 'totalTravelDistance', а в качестве признаков, на основе которых будет вычисляться предсказание-вектор, полученный ранее features:

lr = LinearRegression(labelCol='label', featuresCol='features')

Укажем порядок обработки объектов конвейеру:

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

Полученные результаты предсказаний показаны на рисунке 9.

features	prediction	trueLabel
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2164258861101681])	1027.8787254622098	956
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2164258861101681])	1027.8787254622098	956
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2164258861101681])	1027.8787254622098	956
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2164258861101681])	1027.8787254622098	956
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2164258861101681])	1027.8787254622098	956
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	947
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	1462
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	1462
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	1462
(36,[8,18,33,34],[1.0,1.0,0.2069934985064137,0.2183118956910968])	1022.9588343420718	1856
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2855795707442194])	1111.892282392525 4	956
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	947
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	947
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	947
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	947
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	1856
(36,[8,18,33,34],[1.0,1.0,0.2761445196869552,0.2874655803251481])	1106.9723912723875	1856
(36,[8,18,33,34],[1.0,1.0,0.33146803995512486,0.34982963046785615])	1155.8259798755257	947
(36,[8,18,33,34],[1.0,1.0,0.340280875336226,0.3497164698930004])	1189.8184048055962	956
(36,[8,18,33,34],[1.0,1.0,0.340280875336226,0.3497164698930004])	1189.8184048055962	956
(36,[8,18,33,34],[1.0,1.0,0.3591365583985511,0.37045000188600963])	1207.8300058850396	947
(36,[8,18,33,34],[1.0,1.0,0.46978359893489047,0.4810958973004917])	1342.2723701216812	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.06474493367 6	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.064744933676	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.064744933676	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.064744933676	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.064744933676	947
(36,[8,18,33,34],[1.0,1.0,0.48361109984726225,0.49492663422730193])	1359.064744933676	947

Рисунок 9 – Результаты предсказаний линейной регрессии

Для оценки точности модели, приведенной на рисунке 9, вычисление метрик оценки качества модели, таких как точность, полнота, F1-мера и т.д.

2.4 Оценка модели регрессии

Оценить качество моделей было решено с помощью Root Mean Squared Error (RMSE) и R2 для задачи регрессии. RMSE также называемая среднеквадратичная ошибка - показатель, указывающий нам среднее расстояние между прогнозируемыми значениями из модели и фактическими значениями в наборе данных. Значение RMSE = 0 указывает на идеальное соответствие данным. Оценка R2 – коэффициент детерминации или показатель, который используется для оценки производительности модели машинного обучения на основе регрессии. Он показывает, насколько хорошо модель соответствует данным, где значение 1 означает идеальное соответствие, а значение 0 означает отсутствие соответствия. Суть его работы заключается в измерении количества отклонений в прогнозах, объясненных набором данных. Результаты метрик приведены на рисунке 10. Программный код для оценки модели представлен ниже:

```
evaluator_mse = RegressionEvaluator(labelCol='trueLabel', predictionCol='prediction', metricName="mse")
mse = evaluator_mse.evaluate(predictions)
print(f'Metric "MSE" on test data: {mse:.3f}')
evaluator_mae = RegressionEvaluator(labelCol='trueLabel', predictionCol='prediction', metricName="mae")
mae = evaluator_mae.evaluate(predictions)
print(f'Metric "mae" on test data: {mae:.3f}')
evaluator_rmse = RegressionEvaluator(labelCol='trueLabel', predictionCol='prediction', metricName="rmse")
rmse = evaluator_rmse.evaluate(predictions)
print(f'Metric "rmse" on test data: {rmse:.3f}')
evaluator_r2 = RegressionEvaluator(labelCol='trueLabel', predictionCol='prediction', metricName="r2")
r2 = evaluator_r2.evaluate(predictions)
print(f'Metric "R^2" on test data: {r2:.3f}')
```

```
Metric "MSE" on test data: 280043.962
Metric "mae" on test data: 398.396
Metric "rmse" on test data: 529.192
Metric "R^2" on test data: 0.595
```

Рисунок 10 – Показатели метрик регрессии

Исходя из результатов, приведенных на рисунке 10, можно сделать следующие выводы:

1. Модель не всегда точно предсказывает целевую переменную, учитывая метрики RMSE, MSE. Чем ближе к 0, тем лучше.
2. Значение R^2 от 0 до 1, и чем ближе к 1, тем лучше модель объясняет изменение в зависимой переменной. Значение 0.573 говорит о том, что модель объясняет примерно 57.3% дисперсии в данных, что может быть считаться умеренно хорошим результатом.

2.5 Настройка параметров регресии

Полученные метрики указывают на неточные предсказания модели. Попробуем их улучшить с помощью CrossValidator в Spark. Кросс-валидация позволяет оценить производительность модели путем разделения данных на обучающие и тестовые наборы несколько раз и вычисления среднего значения метрик производительности. При использовании кросс-валидации мы определяем сетку параметров, которая содержит различные значения гиперпараметров модели. Гиперпараметры - это настраиваемые параметры, которые влияют на процесс обучения и определяют характеристики модели, такие как сложность, регуляризация и т. д. Они отличаются от параметров модели, которые обучаются непосредственно из данных. Гиперпараметры можно рассматривать как параметры "верхнего уровня", которые влияют на процесс обучения и влияют на конечные параметры модели [2]. Были установлены: maxDepth – гиперпараметр, который определяет максимальную глубину каждого дерева решений в случайном лесу. Увеличение этого параметра может привести к более сложным моделям, которые могут лучше соответствовать обучающим данным, но могут также увеличить риск

переобучения; numTrees – количество деревьев в случайном лесу. Увеличение этого параметра может привести к более устойчивой модели, но также может увеличить время обучения; maxBins – гиперпараметр, который определяет максимальное количество корзин (bins), используемых при разбиении функций при построении деревьев решений в случайном лесу. Увеличение этого параметра может повысить точность модели, особенно если данные содержат много категориальных признаков. Установим для нашей модели следующие гиперпараметры:

```
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.0, 0.3, 0.5]) \
    .addGrid(lr.maxIter, [50, 100, 150]).build()
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(),
                          numFolds=3)
model = crossval.fit(train)
```

2.6 Проверка сбалансированности распределения классов

Проверим данные на сбалансированность распределения классов для обучающего набора с помощью кода:

```
splits = data_class.randomSplit([0.8, 0.2])
train = splits[0]
test = splits[1].withColumnRenamed('label', 'trueLabel')
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)
```

```
Positive to Negative Class Ratio: 0.8865827570038682
```

Рисунок 11 – Распределение классов до балансировки

Значение ближе к 1 указывает на относительно сбалансированные классы Значения, отличные от 1, указывают на дисбаланс классов.

2.7 Генерация предсказаний модели бинарной классификации

Для задачи бинарной классификации был применен метод прогнозирования «градиентный бустинг», порядок действий такой же, как и для задачи регрессии, рассмотренной выше: gbt = GBTClassifier(labelCol='label', featuresCol='features', maxDepth=4, maxBins=16). Результаты предсказаний бинарной классификации представлены на рисунке 12.

labelCol='label': указывает столбец в наборе данных, который содержит метки классов (целевую переменную), которую модель будет предсказывать.

featuresCol='features': указывает столбец в наборе данных, который содержит вектор признаков, на основе которых модель будет делать предсказания.

maxDepth=4: это гиперпараметр, определяющий максимальную глубину каждого дерева решений в ансамбле градиентного бустинга.

maxBins=16: это гиперпараметр, определяющий максимальное количество корзин (bins) для разделения функций при построении деревьев.

features	prediction	trueLabel
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.04545730636476233])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.04347554536060231,0.0458534369610973])	0.0	1
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.05998209489704565])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.05998209489704565])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.05799956848093935,0.06037822549338065])	0.0	1
(36,[8,19,33,34],[1.0,1.0,0.07093946241809655,0.07331849164032397])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.071470344420345,0.07345317604307787])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.071470344420345,0.07345317604307787])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.071470344420345,0.07384930663941287])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.08414621682697221,0.0880017324111413])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.09867023994730924,0.10104763338377395])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.09867023994730924,0.10104763338377395])	0.0	0
(36,[8,19,33,34],[1.0,1.0,0.10157447678313895,0.10395259109023061])	0.0	0

Рисунок 12 – Результаты предсказаний бинарной классификации LogisticRegression

2.8 Оценка модели бинарной классификации

Оценить качество моделей было решено с помощью метода Apache Spark: BinaryClassificationEvaluator. Он принимает на вход предсказания модели и вычисляет метрики качества, такие как confusion matrix(Матрица ошибок), Precision, Recall, F и AUR(areaUnderROC). Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм. F-мера - среднее гармоническое precision и recall. F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю. Метрики, рассчитанные для бинарной классификации, изображены на рисунке 13.

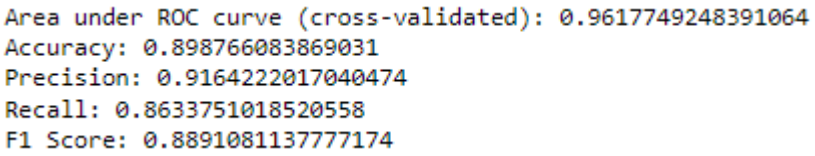

Рисунок 13 – Рассчитанные метрики бинарной классификации

В данном случае Area under ROC curve (AUC-ROC): значение 0.9617 говорит о высокой эффективности модели в разделении классов. AUC-ROC является показателем качества классификации, где значение ближе к 1 указывает на лучшую производительность модели. Accuracy (Точность): значение 0.8988 означает, что модель правильно классифицировала примерно 89.88% всех случаев. Precision (Точность): значение 0.9164 говорит о том, что из всех примеров, которые модель классифицировала как положительные, около 91.64% действительно принадлежат к положительному классу. Это измеряет точность положительных предсказаний. Recall (Полнота): значение 0.8634 указывает на то, что модель уловила около 86.34% всех положительных случаев из общего числа положительных случаев. Это измеряет способность модели обнаруживать все положительные примеры. F1 Score (F1-мера): значение 0.8891 является средним гармоническим между точностью и полнотой. Он предоставляет баланс между двумя метриками.

2.9 Настройка параметров бинарной классификации

Чтобы найти наиболее эффективные параметры, мы можем использовать класс CrossValidator для оценки каждой комбинации параметров, определенных в ParameterGrid, Установим следующие параметры с помощью программного кода:

```
paramGrid = (ParamGridBuilder() \
        .addGrid(gbt.maxDepth, [2, 4, 6]) \
        .addGrid(gbt.maxBins, [8, 16, 32]) \
        .build())
```

```
crossval  = CrossValidator(
    estimator=pipeline,
    evaluator=BinaryClassificationEvaluator(),
    estimatorParamMaps=paramGrid,
    numFolds=2
)
model  = crossval.fit(train)
```

RegParam: параметр указывает на добавление в сетку гиперпараметров значения для коэффициента регуляризации (regParam) модели логистической регрессии; maxIter: этот параметр указывает на добавление в сетку гиперпараметров значения для максимального количества итераций (maxIter) модели логистической регрессии; elasticNetParam – этот параметр указывает на добавление в сетку гиперпараметров значения для параметра эластичной сети (elasticNetParam) модели логистической регрессии.

Были подобраны лучшие результаты для модели градиентного бустинга.

```
best_model = model.bestModel
print("Лучшие параметры модели:")
for param_name, param_value in best_model.stages[-1].extractParamMap().items():
    print(f"{param_name.name}: {param_value}")
```

Результаты представлены на рисунке 14.

cacheNodeIds: Булев параметр, указывающий, следует ли кэшировать идентификаторы узлов. Кэширование может улучшить производительность в случае многократного использования узлов.

checkpointInterval: периодичность (в количество итераций), с которой следует выполнять контрольные точки. Контрольные точки используются для сохранения состояния модели и могут быть полезными при восстановлении после сбоев.

featureSubsetStrategy: стратегия выбора подмножества признаков для обучения каждого дерева. Значение "all" означает использование всех признаков.

featuresCol: Название столбца, содержащего признаки. В данном случае, "features".

impurity: критерий для измерения качества разделения в деревьях. В данном случае, "variance" используется для регрессии.

labelCol: название столбца, содержащего целевую переменную, "label".

leafCol: название столбца, в который будет записан номер листа, к которому относится предсказание.

lossType: тип функции потерь для градиентного бустинга. "logistic" означает логистическую функцию потерь, что подходит для бинарной классификации.

maxBins: максимальное количество бинов, используемых при разделении категориальных признаков.

maxDepth: максимальная глубина каждого дерева в композиции. maxIter: максимальное количество итераций (деревьев) для обучения.

maxMemoryInMB: максимальный объем памяти в мегабайтах для кэширования узлов.

minInfoGain: минимальный информационный выигрыш, необходимый для разделения узла.

minInstancesPerNode: минимальное количество экземпляров, требуемых для образования узла.

minWeightFractionPerNode: минимальная доля веса, необходимая для образования узла.

predictionCol: название столбца, в который будет записан результат предсказания.

probabilityCol: название столбца, в который будут записаны вероятности предсказания классов.

rawPredictionCol: название столбца, в который будут записаны сырые предсказания перед применением функции потерь.

seed: зерно для воспроизводимости результатов.

stepSize: размер шага для обновления весов при градиентном спуске.

subsamplingRate: доля данных, используемых для обучения каждого дерева.

validationTol: параметр, определяющий, когда остановить обучение на основе изменения ошибки на валидационных данных.

Гиперпараметры:

maxDepth: Максимальная глубина каждого дерева.

checkpointInterval: Это гиперпараметр, который определяет интервал (в количестве итераций) для создания контрольных точек в процессе обучения модели.

maxIter: Максимальное количество итераций (деревьев) в градиентном бустинге.

minInfoGain: Минимальное значение информационного выигрыша для узла дерева.

minInstancesPerNode: Минимальное количество экземпляров для создания узла дерева.

minWeightFractionPerNode: Минимальная доля веса суммируемого экземпляра для создания узла дерева.

stepSize: Размер шага для оптимизации.

subsamplingRate: Гиперпараметр, который устанавливает долю подвыборки, используемую при обучении модели.

validationTol: Этот гиперпараметр устанавливает порог для остановки обучения модели при использовании валидационного набора данных.

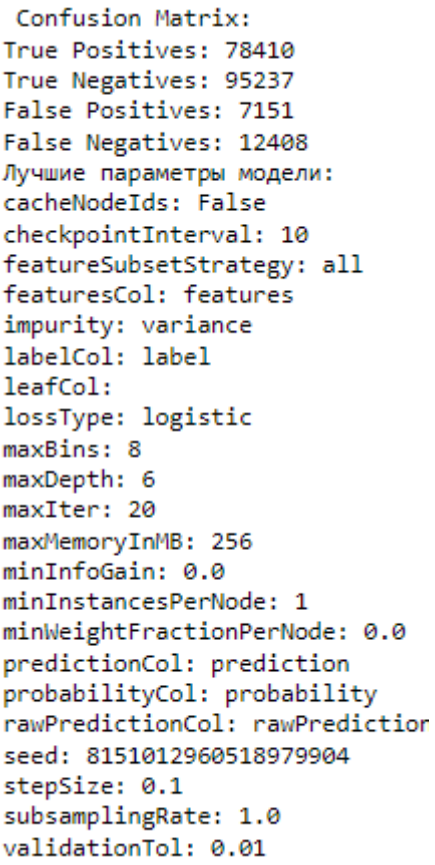

Рисунок 14 – Результаты для лучшей модели

2.10 Выводы

В данном разделе были подготовлены данные для машинного обучения, проведен процесс обучения моделей регрессии и бинарной классификации, а также проведена кросс-валидация для нахождения наилучших показателей у моделей для разных наборов гиперпараметров. В процессе работы познакомились с машинным обучением в Apache Spark, также были изучены методы регрессии по алгоритмам LinearRegression и классификации при помощи GradientBoostingMachine. При проведении кросс-валидации были выявлены лучшие параметры для моделей регрессии и классификации, а также обучены на основе данных параметров.

ЗАКЛЮЧЕНИЕ

Таким образом в данной работе было проведено исследование данных. Исследование было проведено с использованием технологии больших данных. В данной работе был проведен разведочный анализ данных датасета с Kaggle по ссылке, указанной в пункте 1.2 с помощью системы PySpark. Разведочный анализ включал в себя определение типов признаков в датасете, определение пропущенных значений и их устранение, определение выбросов и их устранение, расчет статистических показателей признаков, вывод корреляции между признаками, визуализации распределения признаков. Было проведено машинное обучение обработанных данных датасета с помощью двух алгоритмов машинного обучения - задача регрессии, а именно LinearRegression, и задача бинарной классификации, а именно GradientBoostingMachine.

Эффективность полученных моделей была рассмотрена с помощью расчета метрик классификации, а именно матрицы ошибок и площади под кривой ROC (AUR), и метрик регрессии, а именно
среднеквадратическая ошибка (RMSE) и коэффициент детерминации (R2).

Для улучшения эффективности моделей, был выполнен подбор гиперпараметров модели по сетке. Улучшение модели из задачи бинарной классификации, согласно матрице ошибок не было, это связано с малыми ресурсами устройства, на котором проводилось обучение, однако согласно остальным метрикам, модели улучшились. Улучшение эффективности было доказано с помощью повторного расчета метрик, описанных выше, и сравнение их с метриками, рассчитанными для изначальной модели.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Официальный сайт Apache Spark [Электронный ресурс]. – [2023]. – Режим доступа : https://spark.apache.org/ (дата обращения 09.01.2024).
2. Старовойтов, В. В. Нормализация данных в машинном обучении /

В. В. Старовойтов, Ю. И. Голуб // Информатика. – 2021. – Т. 18. – № 3. – С. 83-96. – DOI 10.37661/1816-0301-2021-18-3-83-96. – EDN JKAHKM.
3. Advanced Pyspark for Exploratory Data Analysis [Электронный ресурс]. – [2022]. – Режим доступа : https://www.kaggle.com/code/tientd95/advanced-pyspark-for-exploratory-data-analysis (дата обращения 09.01.2024).
4. Учебник по машинному обучению [Электронный ресурс]. – [2023]. – Режим доступа : https://academy.yandex.ru/handbook/ml (дата обращения 09.01.2024).
5. Исследовательский анализ данных с помощью pySpark [Электронный ресурс].– [2020].– Режим доступа:https://github.com/roshankoirala/pySpark_tutorial/blob/master/Exploratory_data_анализ_with_pySpark.ipynb(дата обращения 09.01.2024).
6. Advanced Pyspark для исследовательского анализа данных [Электронный ресурс].– [2022].– Режим доступа:https://www.kaggle.com/code/tientd95/advanced-pyspark-for-exploratory-data-analysis(дата обращения 09.01.2024).
7. Исследование данных // Изучение Apache Spark с помощью Python [Электронный ресурс] / У. Фэн.- [2021].- Режим доступа:https://runawayhorse001.github.io/LearningApacheSpark/exploration.html(дата обращения 09.01.2024).
8. Исследовательский анализ данных (EDA) с PySpark на Databricks [Электронный ресурс]. – [2020]. – Режим доступа: https://towardsdatascience.com/exploratory-data-анализ-eda-with-pyspark-on-databricks-e8d6529626b1 (дата обращения 09.01.2024).

ПРИЛОЖЕНИЕ А – Программный код разведочного анализа

```
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
import pylab
import seaborn as sns
import scipy.stats as scipy_stats
from scipy.stats import probplot
from functools import reduce
MAX_MEMORY = '15G'
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("Tp_Lab1") \
        .config(conf=conf) \
        .getOrCreate()
    return spark
spark = init_spark()
filename_data = 'itineraries.csv'
df = spark.read.options(inferSchema='True', header='True',
```

```
print('Data frame type: ' + str(type(df)))
df.printSchema()
df.limit(10).toPandas()
df.select("legId").show(10)
print('Data overview')
df.printSchema()
print('Columns overview')
pd.DataFrame(df.dtypes, columns = ['Column Name','Data type'])
print('Data frame describe (string and numeric columns only):')
print(f'Total rows: {df.count()}')
df.describe().toPandas()
string_columns = [
    'legId', 'startingAirport', 'destinationAirport', 'fareBasisCode', 'travelDuration',
    'segmentsDepartureTimeEpochSeconds', 'segmentsDepartureTimeRaw',
'segmentsArrivalTimeEpochSeconds',
    'segmentsArrivalTimeRaw', 'segmentsArrivalAirportCode',
'segmentsDepartureAirportCode',
    'segmentsAirlineName', 'segmentsAirlineCode', 'segmentsEquipmentDescription',
'segmentsDurationInSeconds',
    'segmentsDistance', 'segmentsCabinCode'
]
numeric_with_zeroes_columns = ['elapsedDays', 'seatsRemaining']
numeric_without_zeroes_columns = ['baseFare', 'totalFare', 'totalTravelDistance']
boolean_columns = ['isBasicEconomy', 'isNonStop']
date_columns = ['searchDate', 'flightDate']
missing_values = {}
for index, column in enumerate(df.columns):
    if column in string_columns: # check None and Null
        missing_count = df.filter(col(column).eqNullSafe(None) |
col(column).isNull()).count()
        missing_values.update({column:missing_count})
```

```
    if column in numeric_with_zeroes_columns: # check None, NaN and Null
        missing_count = df.filter(col(column) == None | isnan(col(column)) |
col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in numeric_without_zeroes_columns:  # check zeroes, None, NaN and
Null
        missing_count = df.filter(col(column).isin([0,None]) | isnan(col(column)) |
col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in boolean_columns:  # check None and Null
        missing_count = df.filter(col(column).eqNullSafe(None) |
col(column).isNull()).count()
        missing_values.update({column:missing_count})
    if column in date_columns:  # check None and Null
        missing_count = df.filter(col(column).eqNullSafe(None) |
col(column).isNull()).count()
        missing_values.update({column:missing_count})
missing_df = pd.DataFrame.from_dict([missing_values])
missing_df
columns_with_missing_values = []
for column in missing_df:
    if missing_df[column].values[0] != 0:
        columns_with_missing_values.append(column)
missing_df[columns_with_missing_values]
print(missing_values)
mean_value = df.agg(mean(df['totalTravelDistance'])).collect()[0][0]
mean_value
df_fill = df.withColumn('totalTravelDistanceWasNull',
when(df['totalTravelDistance'].isNull(), 1).otherwise(0))
df_fill=df_fill.na.fill(value=mean_value,subset=["totalTravelDistance"])
```

```
df_fill.select('totalTravelDistance','totalTravelDistanceWasNull').limit(100).toPandas(
)
print(f'Number of rows before deleting na values: {df.count()}')
df = df.na.drop(subset=columns_with_missing_values)
print(f'Number of rows after deleting na values: {df.count()}')
print(f'Number of rows before deleting na values: {df_fill.count()}')
df_fill = df_fill.na.drop(subset=columns_with_missing_values)
print(f'Number of rows after deleting na values: {df_fill.count()}')
cleaned_dataframe = df.dropna()
cleaned_dataframe.count()
from pyspark.sql import functions as F
selected_columns = ['baseFare', 'totalFare', 'totalTravelDistance']
for column in selected_columns:
    quartiles = cleaned_dataframe.stat.approxQuantile(column, [0.25, 0.75], 0.0)
    IQR = quartiles[1] - quartiles[0]
    lower_bound = quartiles[0] - 1.5 * IQR
    upper_bound = quartiles[1] + 1.5 * IQR
    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 (до)'{column}': Снизу выбросов -
{below_quartile_count_before}, Сверху выбросов -
{above_quartile_count_before}")
    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]
    median_value_before = cleaned_dataframe.approxQuantile(column, [0.5], 0.0)[0]
    print(f"Столбец (до) '{column}': Максимальное значение - {max_value_before},
Минимальное значение - {min_value_before}, Медиана -
{median_value_before}")
```

```
    cleaned_dataframe = cleaned_dataframe.filter((col(column) >= lower_bound) &
(col(column) <= upper_bound))
    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 (после)'{column}': Снизу выбросов -
{below_quartile_count_after}, Сверху выбросов - {above_quartile_count_after}")
    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]
    median_value_after = cleaned_dataframe.approxQuantile(column, [0.5], 0.0)[0]
    print(f"Столбец (после) '{column}': Максимальное значение -
{max_value_after}, Минимальное значение - {min_value_after}, Медиана -
{median_value_after}")
from pyspark.sql.functions import lit, desc, col, size, array_contains, isnan, udf, hour,
array_min, array_max, countDistinct, expr
dataframe = df.select('startingAirport', 'destinationAirport', 'isNonStop',
                      'isBasicEconomy', 'baseFare', 'totalFare',
                      'seatsRemaining', 'totalTravelDistance', 'travelDuration')
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)"))
dataframe = dataframe.withColumn("travelDuration", expr("hours * 60 + minutes"))
dataframe = dataframe.drop("hours", "minutes")
dataframe.limit(5).toPandas()
def plot_histogram(df, column):
    data = df.select(collect_list(column)).first()[0]
    plt.figure(figsize=(10, 6))
    plt.hist(data, bins='auto', color='blue')
```

```
    plt.title(f'Histogram of {column}')
    plt.xlabel(column)
    plt.ylabel('Frequency')
    plt.grid(axis='y', linestyle='--', alpha=0.7)
    plt.show()
def plot_boxplot(df, column):
    data = df.select(collect_list(column)).first()[0]
    plt.figure(figsize=(10, 6))
    plt.boxplot(data, vert=False)  # vert=False для горизонтального ящика
    plt.title(f"Box plot of '{column}'")
    plt.xlabel(column)
    plt.show()
for column in numeric_with_zeroes_columns:
    print(f'Column: {column}')
    plot_histogram(cleaned_df, column)
for column in numeric_without_zeroes_columns:
    print(f'Column: {column}')
    plot_histogram(cleaned_df, column)
for column in numeric_with_zeroes_columns:
    print(f'Column: {column}')
    plot_boxplot(cleaned_df, column)
for column in numeric_without_zeroes_columns:
    print(f'Column: {column}')
    plot_boxplot(cleaned_df, column)
def pie_chart(df, column):
    pandas_df = df.groupby(column).count().toPandas()
    pandas_df = pandas_df.set_index(pandas_df.columns[0])
    fig, ax = plt.subplots(figsize=(12, 7), subplot_kw=dict(aspect='equal'), dpi=120)
    data = pandas_df['count']
    categories = pandas_df.index
    plt.pie(data, labels = categories, autopct="%1.1f%%")
```

```
    ax.set_title(f"Pie Chart '{column}'")
    plt.show()
pie_chart(cleaned_df, 'startingAirport')
pie_chart(cleaned_df, 'destinationAirport')
pie_chart(cleaned_df, 'seatsRemaining')
from pyspark.ml.feature import VectorAssembler,Bucketizer
num_buckets = 20
step = 4000 / num_buckets
splits = [float("-inf")] + [i * step for i in range(0, num_buckets)] + [float("inf")]
selected_columns = ['baseFare', 'totalFare','totalTravelDistance']
for selected_column in selected_columns:
    bucketizer = Bucketizer(splits=splits, inputCol=selected_column,
outputCol="bucketFeature")
    df_bucket = bucketizer.transform(cleaned_dataframe)
    bucket_counts =
df_bucket.groupBy("bucketFeature").count().orderBy("bucketFeature")
    bucket_counts.show()
    bucket_counts_pd = bucket_counts.toPandas()
    plt.bar(bucket_counts_pd["bucketFeature"], bucket_counts_pd["count"],
align="center", label=selected_column)
    plt.title(f"Distribution of {selected_column}")
    plt.xlabel("Values")
    plt.ylabel("Frequency")
    plt.show()
selected_features = ['startingAirport', 'destinationAirport', 'isNonStop',
                     'isBasicEconomy',
                     'seatsRemaining']
fig, axs = plt.subplots(len(selected_features), 1, figsize=(8, 3 *
len(selected_features)))
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='darkgreen', edgecolor='black')
    axs[i].set_title(f'Distribution of {feature}')
    axs[i].set_xlabel(feature)
    axs[i].set_ylabel('Count')
vector_col = 'corr_features'
numeric_columns = numeric_with_zeroes_columns +
numeric_without_zeroes_columns
assembler = VectorAssembler(inputCols=numeric_columns, outputCol=vector_col)
df_vector = assembler.transform(cleaned_df).select(vector_col)
matrix = Correlation.corr(df_vector, vector_col).collect()[0][0]
corr_matrix = matrix.toArray().tolist()
corr_matrix_df = pd.DataFrame(data=corr_matrix, columns=numeric_columns,
index=numeric_columns)
plt.figure(figsize=(16,5))
sns.heatmap(
    corr_matrix_df,
    xticklabels=corr_matrix_df.columns.values,
    yticklabels=corr_matrix_df.columns.values,
    cmap= 'coolwarm',
    annot=True
)
```

ПРИЛОЖЕНИЕ Б – Программный код задачи регрессии

```
data = cleaned_dataframe.select(
    'startingAirport',
```

```
    'destinationAirport',
    'elapsedDays',
    col('isBasicEconomy').cast('Int').alias('isBasicEconomy'),
    col('isNonStop').cast('Int').alias('isNonStop'),
    'baseFare',
    'totalFare',
    col('totalTravelDistance').alias('label')
)
data.show(10)
from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler,
VectorIndexer, MinMaxScaler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml import Pipeline
from pyspark.sql.functions import col
from pyspark.ml.regression import LinearRegression
from pyspark.sql import SparkSession
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler,
MinMaxScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
from pyspark.ml.evaluation import RegressionEvaluator
# применяется для преобразования категориальных переменных в числовой
форма
strIdx = StringIndexer(inputCols = ['startingAirport', 'destinationAirport'], outputCols
= ['startingAirportIdx', 'destinationAirportIdx'])
oneHotEnc = OneHotEncoder(inputCols=['startingAirportIdx',
'destinationAirportIdx'], outputCols=['startingAirportEnc', 'destinationAirportEnc'])
catVect = VectorAssembler(inputCols = ['startingAirportEnc', 'destinationAirportEnc',
'isBasicEconomy', 'isNonStop'], outputCol='catFeatures')
```

```
numVect = VectorAssembler(inputCols = ['baseFare', 'totalFare', 'elapsedDays'],
outputCol='numFeatures')
minMax = MinMaxScaler(inputCol = numVect.getOutputCol(),
outputCol='normFeatures')
featVect = VectorAssembler(inputCols=['catFeatures', 'normFeatures'],
outputCol='features')
lr = LinearRegression(labelCol='label', featuresCol='features')
pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, numVect, minMax, featVect,
lr])
splits = data.randomSplit([0.8, 0.2])
train = splits[0]
test = splits[1].withColumnRenamed('label', 'trueLabel')
paramGrid = ParamGridBuilder() \
    .addGrid(lr.regParam, [0.0, 0.3, 0.5]) \
    .addGrid(lr.maxIter, [50, 100, 150]).build()
crossval = CrossValidator(estimator=pipeline,
                          estimatorParamMaps=paramGrid,
                          evaluator=RegressionEvaluator(),
                          numFolds=3)
model = crossval.fit(train)
predictions = model.transform(test)
predictions = predictions.select('features', 'prediction', 'trueLabel')
predictions.show(100, truncate=False)
```

```
evaluator_mse = RegressionEvaluator(labelCol='trueLabel',
predictionCol='prediction', metricName="mse")
mse = evaluator_mse.evaluate(predictions)
print(f'Metric "MSE" on test data: {mse:.3f}')
```

```
evaluator_mae = RegressionEvaluator(labelCol='trueLabel',
predictionCol='prediction', metricName="mae")
mae = evaluator_mae.evaluate(predictions)
print(f"Metric "mae" on test data: {mae:.3f}")
evaluator_rmse = RegressionEvaluator(labelCol='trueLabel',
predictionCol='prediction', metricName="rmse")
rmse = evaluator_rmse.evaluate(predictions)
print(f"Metric "rmse" on test data: {rmse:.3f}")
evaluator_r2 = RegressionEvaluator(labelCol='trueLabel', predictionCol='prediction',
metricName="r2")
r2 = evaluator_r2.evaluate(predictions)
print(f"Metric "R^2" on test data: {r2:.3f}")
# Вывод лучших параметров
best_model = model.bestModel
best_parameters = best_model.stages[-1].extractParamMap()
print("Лучшие параметры модели LinearRegression:")
for param, value in best_parameters.items():
    print(f"{param.name}: {value}")
```

ПРИЛОЖЕНИЕ В – Программный код задачи регрессии

```
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator
data_class = cleaned_df.select(
```

```
    'startingAirport',
    'destinationAirport',
    'elapsedDays',
    col('isBasicEconomy').cast('Int').alias('isBasicEconomy'),
    col('isNonStop').cast('Int').alias('isNonStop'),
    'baseFare',
    'totalFare',
    (col('totalTravelDistance') > 1500).cast('Int').alias('label')
)
data_class.show(10)
strIdx = StringIndexer(inputCols = ['startingAirport', 'destinationAirport'], outputCols
= ['startingAirportIdx', 'destinationAirportIdx'])
oneHotEnc = OneHotEncoder(inputCols=['startingAirportIdx',
'destinationAirportIdx'], outputCols=['startingAirportEnc', 'destinationAirportEnc'])
catVect = VectorAssembler(inputCols=['startingAirportEnc', 'destinationAirportEnc',
'isBasicEconomy', 'isNonStop'], outputCol='catFeatures')
numVect = VectorAssembler(inputCols=['baseFare', 'totalFare', 'elapsedDays'],
outputCol='numFeatures')
minMax = MinMaxScaler(inputCol=numVect.getOutputCol(),
outputCol='normFeatures')
featVect = VectorAssembler(inputCols=['catFeatures', 'normFeatures'],
outputCol='features')
gbt = GBTClassifier(labelCol='label', featuresCol='features', maxDepth=4,
maxBins=16)
pipeline = Pipeline(stages=[strIdx, oneHotEnc, catVect, numVect, minMax, featVect,
gbt])
splits = data_class.randomSplit([0.8, 0.2])
train = splits[0]
test = splits[1].withColumnRenamed('label', 'trueLabel')
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)
paramGrid = (ParamGridBuilder() \
            .addGrid(gbt.maxDepth, [2, 4, 6]) \
            .addGrid(gbt.maxBins, [8, 16, 32]) \
            .build())
crossval  = CrossValidator(
   estimator=pipeline,
   evaluator=BinaryClassificationEvaluator(),
   estimatorParamMaps=paramGrid,
   numFolds=2
)
model  = crossval.fit(train)
predictions = model.transform(test)
predictions = predictions.select('features', 'prediction', 'trueLabel')
predictions.show(50, truncate=False)
evaluator = BinaryClassificationEvaluator(labelCol='trueLabel',
rawPredictionCol='rawPrediction', metricName="areaUnderROC")
area_under_roc_cv = evaluator.evaluate(prediction)
print(f"Area under ROC curve (cross-validated): {area_under_roc_cv}")
true_positives = predictions.filter("prediction = 1.0 AND label = 1").count()
true_negatives = predictions.filter("prediction = 0.0 AND label = 0").count()
false_positives = predictions.filter("prediction = 1.0 AND label = 0").count()
false_negatives = predictions.filter("prediction = 0.0 AND label = 1").count()
accuracy = (true_positives + true_negatives) / (true_positives + true_negatives +
false_positives + false_negatives)
print(f"Accuracy: {accuracy}")
precision = true_positives / (true_positives + false_positives)
print(f"Precision: {precision}")
recall = true_positives / (true_positives + false_negatives)
print(f"Recall: {recall}")
```

```
f1_score = 2 * (precision * recall) / (precision + recall)
print(f"F1 Score: {f1_score}")
print("\n Confusion Matrix:")
print(f"True Positives: {true_positives}")
print(f"True Negatives: {true_negatives}")
print(f"False Positives: {false_positives}")
print(f"False Negatives: {false_negatives}")
best_model = model.bestModel
print("Лучшие параметры модели:")
for param_name, param_value in best_model.stages[-1].extractParamMap().items():
    print(f"{param_name.name}: {param_value}")
```