Geekbrains

**Разведочный анализ с помощью PySpark на примере датасета Expedia**

Программа: Разработчик-аналитик

Специализация Data Engineer

Мерберг Вепроника Александровна

Москва, 2024

СОДЕРЖАНИЕ

[Введение 4](#_heading=h.30j0zll)

[1. Разведочный анализ данных с помощью PySpark 5](#_heading=h.1fob9te)

[1.1 Постановка задачи 5](#_heading=h.3znysh7)

[1.2 Определение типов признаков в датасете 5](#_heading=h.2et92p0)

[1.3 Определение пропущенных значений и их устранение 9](#_heading=h.tyjcwt)

[1.4 Определение и удаление выбросов 11](#_heading=h.3dy6vkm)

[1.5 Расчет статистических показателей признаков и их визуализация 12](#_heading=h.1t3h5sf)

[1.6 Корреляций между признаками 15](#_heading=h.4d34og8)

[Чтобы выявить связи между признаками, можно построить матрицу корреляций. Матрица корреляций показана на рисунке 8. 15](#_heading=h.2s8eyo1)

[1.7 Выводе о работе 16](#_heading=h.17dp8vu)

[2 МАШИННОЕ ОБУЧЕНИЕ НА БОЛЬШИХ ДАННЫХ 16](#_heading=h.3rdcrjn)

[2.1 Постановка задачи 16](#_heading=h.26in1rg)

[2.2 Подготовка данных для модели регресии 16](#_heading=h.lnxbz9)

[2.3 Обучение модели линейной регрессии 17](#_heading=h.35nkun2)

[2.4 Оценка модели регрессии 18](#_heading=h.1ksv4uv)

[2.5 Настройка параметров регресии 20](#_heading=h.44sinio)

[2.6 Проверка сбалансированности распределения классов 21](#_heading=h.2jxsxqh)

[2.7 Генерация предсказаний модели бинарной классификации 21](#_heading=h.z337ya)

[2.8 Оценка модели бинарной классификации 22](#_heading=h.3j2qqm3)

[2.9 Настройка параметров бинарной классификации 23](#_heading=h.1y810tw)

[2.10 Выводы 26](#_heading=h.4i7ojhp)

[ЗАКЛЮЧЕНИЕ 27](#_heading=h.2xcytpi)  
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 28 ПРИЛОЖЕНИЕ А – Программный код разведочного анализа 31  
ПРИЛОЖЕНИЕ Б – Программный код задачи регрессии 38  
ПРИЛОЖЕНИЕ В – Программный код задачи регрессии 41

# 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 В PYSPARK

# 1.1 Постановка задачи

Выполнить разведочный анализ датасета авиабилетов из Expedia 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  
 Булевые признаки:isBasicEconomy, isRefundable, isNonStop



Рисунок 1 – Типы данных в датасете

## 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.

Было найдено количество пустых и нулевых значений, а также в каких колонках были найдены эти значения. Программный код для удаления пустых и нулевых значений представлен ниже:

for index, column in enumerate(df.columns):

if column in string\_columns:

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:

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:

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})

## 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.functions import lit, desc, col, size, array\_contains, isnan, udf, hour, array\_min, array\_max, countDistinct, expr

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]

Результат поиска выбросов приведен на рисунке 2. Как видно из рисунка, выбросы имеется в колонках “totalFare”, “baseFare”.

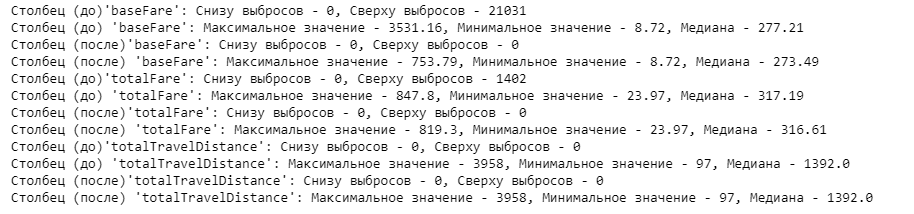


Рисунок 2 – Результаты поиска выбросов

## 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. Минимальное значение — это наименьшее число в наборе данных.
   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.

1. Среднеквадратичное отклонение (или стандартное отклонение). Стандартное отклонение показывает, насколько в среднем значения в наборе отличаются от среднего значения. Программный код представлен ниже:

print('Data frame describe (string and numeric columns only):')

print(f'Total rows: {df.count()}')

df.describe().toPandas()

Расчет статических показателей произведен на рисунке 3.

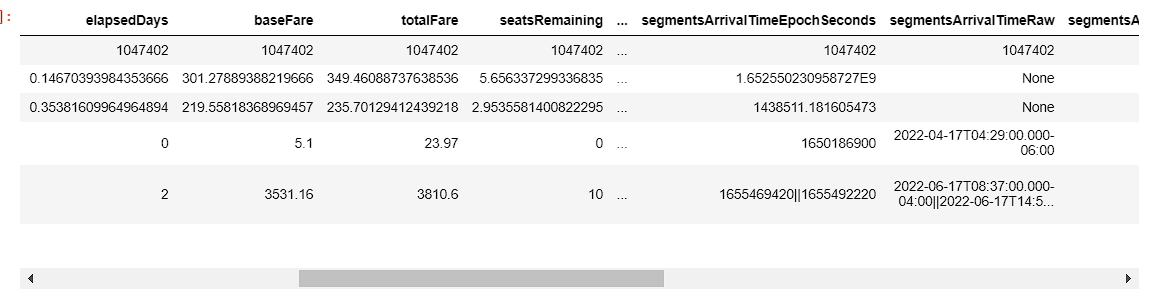


Рисунок 3 – Расчет статических показателей

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, пример которых иллюстрирует рисунок 4;

2. Box plot, пример изображён на рисунке 5;

3. Круговая диаграмма пример изображён на рисунке 6.

Для построения графиков используется следующий алгоритм:

1. Импортируем необходимые классы Bucketizer и VectorAssembler

2. Далее — определяем количество корзин и шаг для разделения значений  
 3. Выбираем столбцы, которые необходимы для построения графика

4. Для каждого столбца создается экземлляр Bucketizer, который разбивает значения столбца на корзины (20)

5. Применяем Bucketizer к очищенному набору данных и подсчитывает количество значений в каждой корзине

Bucketizer — трансформатор в библиотеке для Apache Spark, который используется для разделения числовых значений на «корзины».

Код для построения гистограммы:

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()

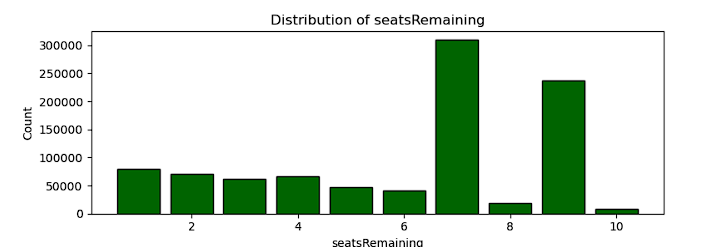


Рисунок 4 – Визуализация гистограммы распределения занятых мест

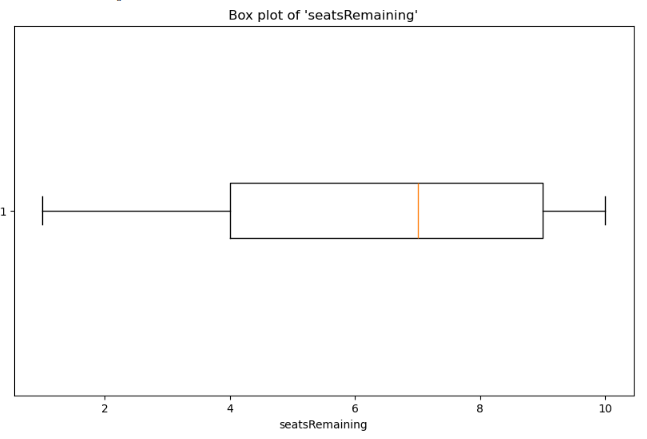


Рисунок 5 – Визуализация Box plot распределения количества оставшихся мест.

Код для построения круговой диаграммы, отрисованной на рисунке 5:

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%%")

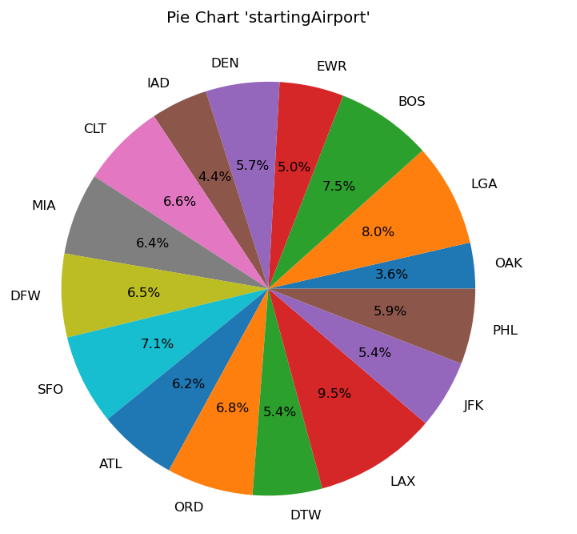
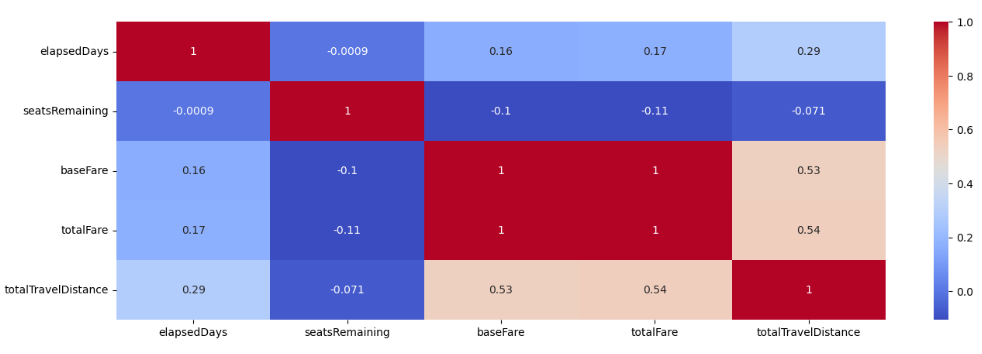


Рисунок 6– Визуализация круговой диаграммы распределения индексов начальных аэропортов.

## 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 7 – 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'
2. 'destinationAirport'
3. 'elapsedDays'
4. 'isBasicEconomy' (приведен к типу Int)
5. 'isRefundable' (приведен к типу Int)
6. 'isNonStop' (приведен к типу Int)
7. 'baseFare'
8. 'totalFare'

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', 'isRefundable', 'isNonStop', 'baseFare', и 'totalFare'.

Критерии: используем метрику AUC-ROC для оценки качества модели

## 2.2 Подготовка данных для модели регресии

При выполнении разведочного анализа были определены признаки,

имеющие наибольшую корреляцию, возьмем их для последующего анализа.

Для построения модели регресси выберем прогнозируемую переменную

'totalTravelDistance' с новым псевдонимом label из очищенного датафреймa cleanded\_df:

data = cleaned\_dataframe.select(

'startingAirport',

'destinationAirport',

'elapsedDays',

col('isBasicEconomy').cast('Int').alias('isBasicEconomy'),

col('isRefundable').cast('Int').alias('isRefundable'),

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 8.

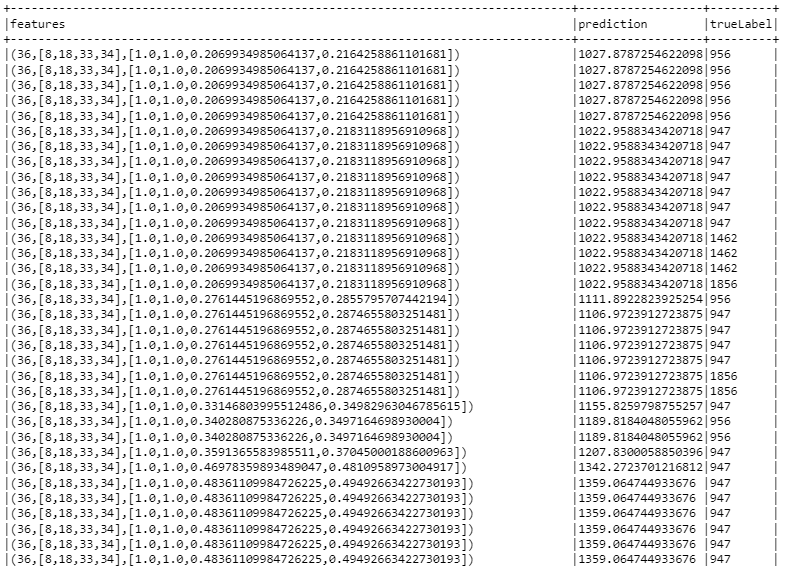


Рисунок 8 – Результаты предсказаний линейной регрессии

Для оценки точности модели, приведенной на рисунке 8, 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 9. 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}')

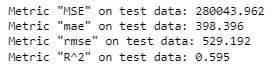


Рисунок 9– Показатели метрик регрессии

Исходя из результатов, приведенных на рисунке 8, 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)



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.

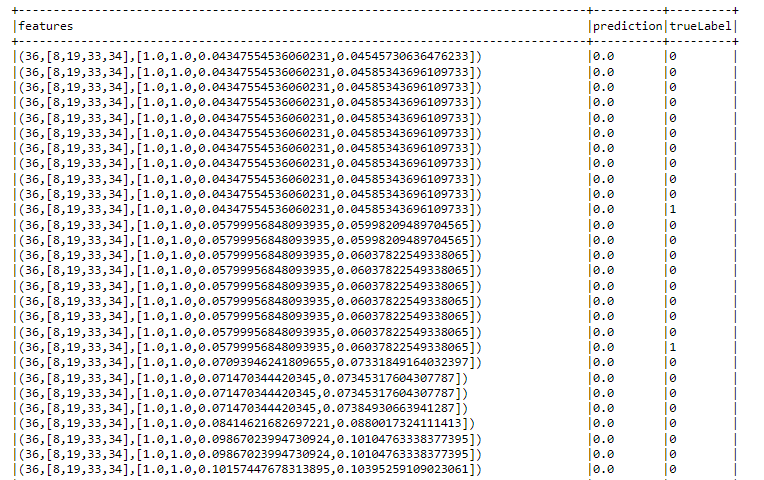


Рисунок 12 – Результаты предсказаний бинарной классификации

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.

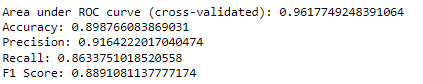


Рисунок 13 – Рассчитанные метрики бинарной классификации

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),

где 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.

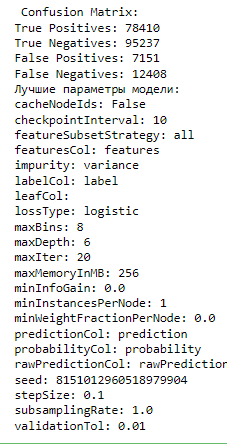


Рисунок 14 – Результаты для лучшей модели

## 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/Explorato

ry\_data\_anaлиз\_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.ht

ml(дата обращения 09.01.2024).

8. Исследовательский анализ данных (EDA) с PySpark на Databricks

[Электронный ресурс]. – [2020]. – Режим доступа:

https://towardsdatascience.com/exploratory-data-anaлиз-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', 'isRefundable', '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', 'isRefundable', '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', 'isRefundable',

'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('isRefundable').cast('Int').alias('isRefundable'),

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', 'isRefundable', '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('isRefundable').cast('Int').alias('isRefundable'),

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', 'isRefundable', '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}")