Цель работы: выполнить разведочный анализ датасета большого объёма с помощью фреймворка Apache Spark.

Задачи:

1. Осуществить поиск готового дата-сета;

2. Загрузить дата-сет на локальную файловую систему и HDFS;

3. Оценить состояние дата-сета на осуществление проведение по нему анализа путем чтения данных;

4. Провести очистку данных (удаление ненужных столбцов, изменение типа данных по столбцам);

5. Осуществить разведочных анализ дата-сета (типов признаков в датасете, пропущенных значений и их устранением, выбросов и их устранением, расчетом статистических показателей признаков, визуализацией распределения наиболее важных признаков, корреляций между признаками);

6. Сделать выводы о состоянии той области, который описывает дата-сет.

Очистка данных

В качестве данных будем использовать готовый дата-сет. Этот набор данных представляет собой CSV-файл, где каждая строка – это билет, купленный на Expedia в период с 16.04.2022 по 05.10.2022 в/из следующих аэропортов: ATL, DFW, DEN, ORD, LAX, CLT, MIA, JFK, EWR, SFO, DTW, BOS, PHL, LGA, IAD, OAK.

Дата-сет «itineraries.csv» (название файла) хранится на кластере в распределенной файловой системе HDFS.

Выполнение разведочного анализа больших данных будет проходить с помощью фреймворка Apache Spark.

Для этого необходимо подключить библиотеки:

import os

from pyspark.sql import SparkSession, DataFrame

from pyspark import SparkConf

from pyspark.sql.functions import (

regexp\_replace,

regexp\_extract\_all,

col,

lit

)

Формирование объекта конфигурации для Apache Spark произведем с помощью указания необходимых параметров:

def create\_spark\_configuration() -> SparkConf:

user\_name = os.getenv("USER")

…

conf = SparkConf()

conf.setAppName("lab 1 Test")

…

f"hdfs:///user/{user\_name}/warehouse")

…

    return conf

После этого создадим объект конфигурации:

conf = create\_spark\_configuration()

Создадим и сессию Apache Spark. В процессе создания сессии происходит подключение к кластеру Apache Hadoop.

spark = SparkSession.builder.config(conf=conf).getOrCreate()

Ранее указывалась информация про дата-сет, и теперь необходимо указать в HDFS путь до него:

path = "hdfs:///user/user3/andreev-directory/itineraries.csv"

Заполним дата-фрейм данными из файла:

df = (spark.read.format("csv")

.option("header", "true")

.load(path)

)

Выведем на экран часть дата-фрейма с помощью метода .show(), чтобы проверить его заполненность данными (рис. 1).

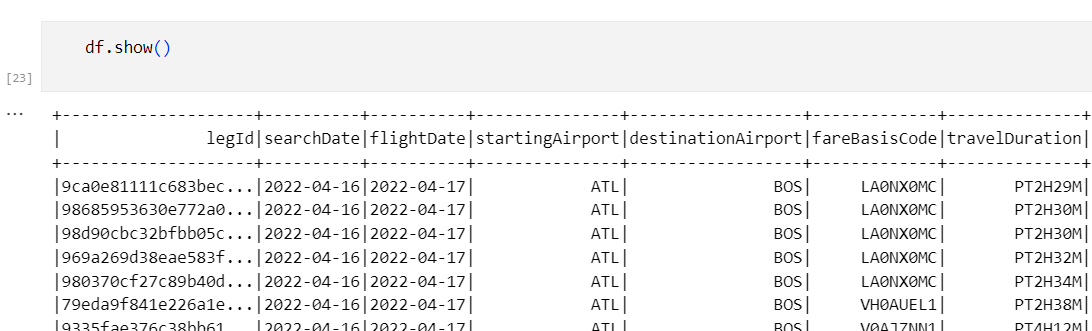


Рисунок 1 – Вывод дата-фрейма

На рисунке 1 изображена часть столбцов, которые входят в дата-сет, поэтому приведем подробное описание каждого столбца.

Описание столбцов:

**legId** – идентификатор рейса (String)

searchDate – дата поиска билета

flightDate – дата вылета рейса

**startingAirport** – аэропорт вылета (String)

**destinationAirport** – аэропорт назначения (String)

fareBasisCode – тарифный код условий

travelDuration – продолжительность полета (формат ISO)

elapsedDays – количество дней между датой поиска и вылета

isBasicEconomy – признак базового эконом-класса

**isRefundable** – признак возврата билета (boolean)

isNonStop – признак прямого рейса

**baseFare** – базовая цена билета (без сборов и налогов) (double)

**totalFare** – общая стоимость билета (включая сборы и налоги) (double)

seatsRemaining – оставшееся количество мест на рейсе

totalTravelDistance – общее расстояние путешествия

**segmentsDepartureTimeEpochSeconds** – время вылета сегмента рейса в формате Unix Epoch (секунды с 1970 года) (int)

segmentsDepartureTimeRaw – время вылета сегмента в формате ISO 8601

segmentsArrivalTimeEpochSeconds – время прилета сегмента рейса в формате Unix Epochs

segmentsArrivalTimeRaw – время прилета сегмента в формате ISO 8601

segmentsArrivalAirportCode – код аэропорта прибытия сегмента рейса

segmentsDepartureAirportCode – код аэропорта вылета сегмента рейса

**segmentsAirlineName** – название авиакомпании (String)

segmentsAirlineCode – код авиакомпании

**segmentsEquipmentDescription** – описание самолета, выполняющего сегмент рейса (String)

**segmentsDurationInSeconds** – длительность сегмента рейса в секундах (String)

**segmentsDistance** – расстояние сегмента рейса (String)

**segmentsCabinCode** – код класса обслуживания (String)

В целях сохранения ясности информации в дата-сете и сокращения расчетного времени при разведочном анализе имеет смысл рассматривать не все столбцы. Оставим следующие колонки, удалив остальные:

legId – идентификатор рейса (String)

startingAirport – аэропорт вылета (String)

destinationAirport – аэропорт назначения (String)

isRefundable – признак возврата билета (boolean)

baseFare – базовая цена билета (без сборов и налогов) (double)

totalFare – общая стоимость билета (включая сборы и налоги) (double)

segmentsDepartureTimeEpochSeconds – время вылета сегмента рейса в формате

segmentsAirlineName – название авиакомпании (String)

segmentsEquipmentDescription – описание самолета, выполняющего сегмент рейса (String)

segmentsDurationInSeconds – длительность сегмента рейса в секундах (String)

segmentsDistance – расстояние сегмента рейса (String)

segmentsCabinCode – код класса обслуживания (String)

С помощью метода .select() выберем эти столбцы и переопределим дата-фрейм:

df = df.select(

… )

Выведем мета-данные дата-сета с помощью метода .printSchema (рис. 2). Эта информация необходима для дальнейшего расчета, при проведении анализа.

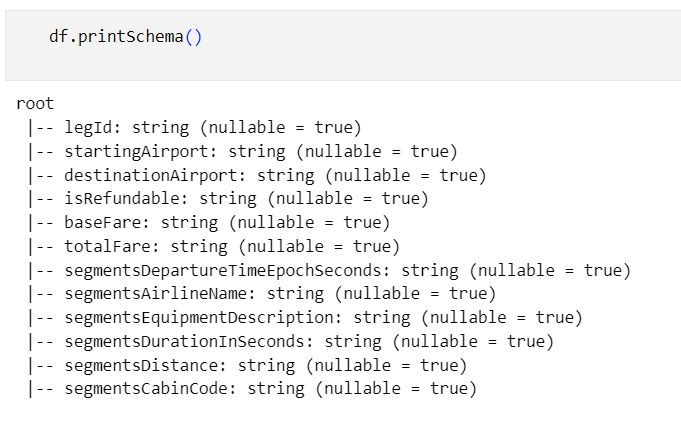


Рисунок 2 – Вывод мета-данных

Исходя из рисунка видно, что все столбцы датасета содержат строковый тип данных (String), что не соответствует ожиданиям. Выполним преобразования типов данных некоторых столбцов с помощью метода:

def transform\_dataframe(data: DataFrame) -> DataFrame:

    data = data.withColumn("isRefundable",

                            col("isRefundable").cast("Boolean"))

    …

    data = data.withColumn("segmentsDepartureTimeEpochSeconds",

                            col("segmentsDepartureTimeEpochSeconds").cast("Integer"))

    return data

Применим этот метод:

df = transform\_dataframe(df)

Выведем на экран мета-данные, после преобразования дата-фрейма (рис. 3).

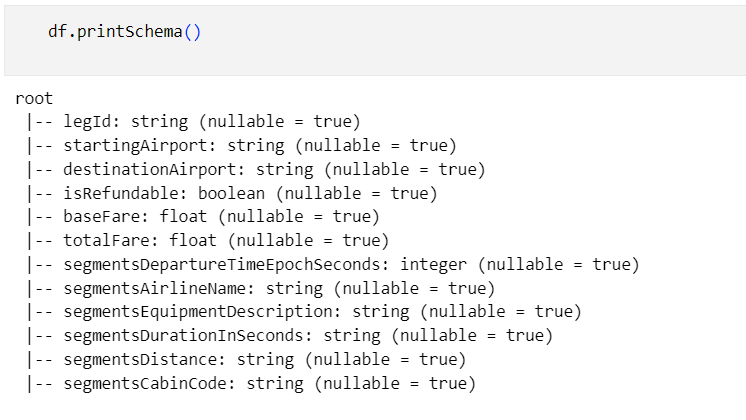


Рисунок 3 – Мета-данные дата-фрейма

Исходя из рисунка видно, что теперь столбцы дата-фрейма содержат значения корректных типов.

Полученный дата-фрейм сохраним в таблицу Apache Iceberg для дальнейшего использования.

Сначала создадим базу данных, в которой будет расположена таблица:

database\_name = "andreev\_database\_new"

Создадим инструкцию SQL для добавления базы данных в каталог Apache Spark:

create\_database\_sql = f""" CREATE DATABASE IF NOT EXISTS spark\_catalog.{database\_name} """

spark.sql(create\_database\_sql)

Установим созданную базу данных как текущую:

spark.catalog.setCurrentDatabase(database\_name)

Записываем преобразованный дата-фрейм в таблицу «sobd\_lab1\_table»:

df.writeTo("sobd\_lab1\_table").using("iceberg").create()

Разведочный анализ

Подготовим подключение необходимых библиотек для проведения разведочного анализа:

import os

from pyspark.sql import SparkSession, DataFrame

from pyspark import SparkConf

…

Сформируем объект конфигурации для Apache Spark, указав необходимые параметры:

def create\_spark\_configuration() -> SparkConf:

    …

    conf.setAppName("lab 1 Test")

    …

 conf.set("spark.sql.catalog.spark\_catalog.warehouse", f"hdfs:///user/{user\_name}/warehouse")

    …

return conf

Используя данный метод, создадим объект конфигурации:

conf = create\_spark\_configuration()

Создадим сессию Apache Spark:

spark = SparkSession.builder.config(conf=conf).getOrCreate()

Укажем базу данных и установим её, как текущую:

database\_name = "andreev\_database\_new"

spark.catalog.setCurrentDatabase(database\_name)

Прочитаем сохранённую таблицу и загрузим её в Spark Dataframe и выведем её на экран с помощью метода .show() (рис. 4).

df = spark.table("sobd\_lab1\_table")

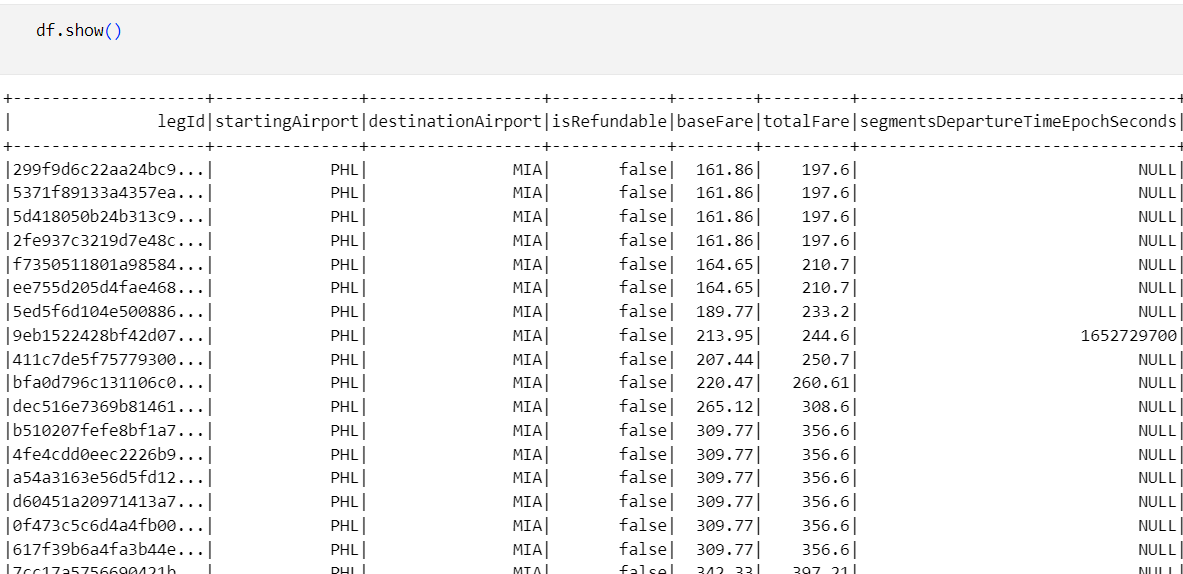


Рисунок 4 – Вывод на экран таблицы

Как видим, осуществился корректный вывод данных. Теперь выведем мета-данные таблицы с помощью метода .printSchema() (рис. 5).

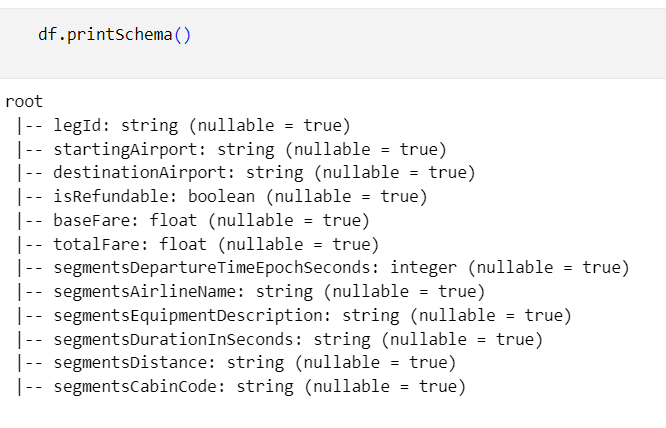


Рисунок 5 – Мета-данные таблицы

Выведем информацию о количестве строк в дата-фрейме с помощью метода .count() (рис. 6).

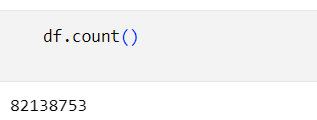


Рисунок 6 – Информация о количестве строк

Исходя из рисунка видно, что количество строк составляет около 82 млн. (82138753).

2.1 Анализ столбца «legId»

Отсортируем дата-фрейм по столбцу «legId», который может рассматриваться в качестве первичного ключа таблицы:

df.orderBy("legId", ascending=False).show()

Результаты покажем на рисунке 7.

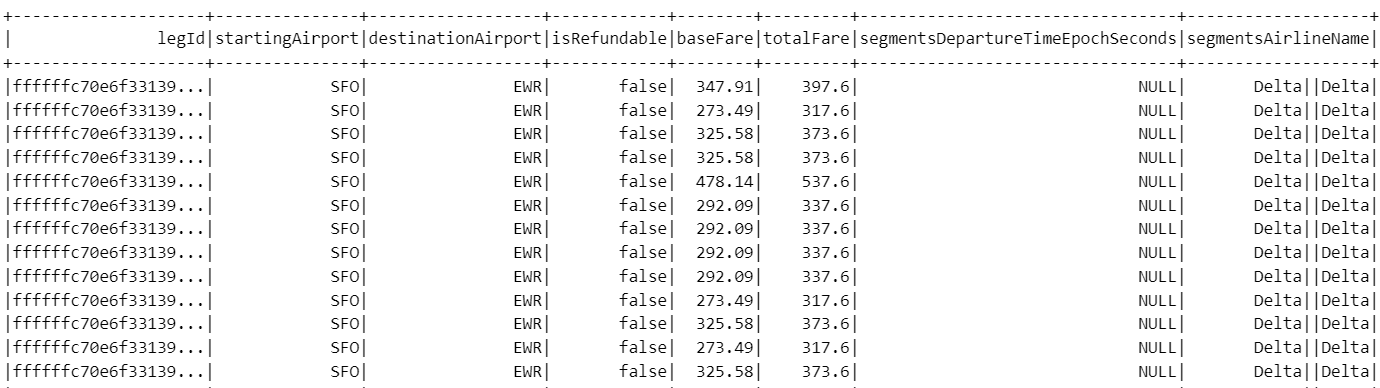


Рисунок 7 – Вывод результатов

Проверим dataframe на наличие пустых данных в столбце legId:

empty\_airline\_names = df.filter(col("legId").isNull() | (col("legId") == "")).count()

print(empty\_airline\_names)

Вывод: 0

Проверим наличие дубликатов в датафрейме.

(

    df

    .groupBy("legId")

    …

)

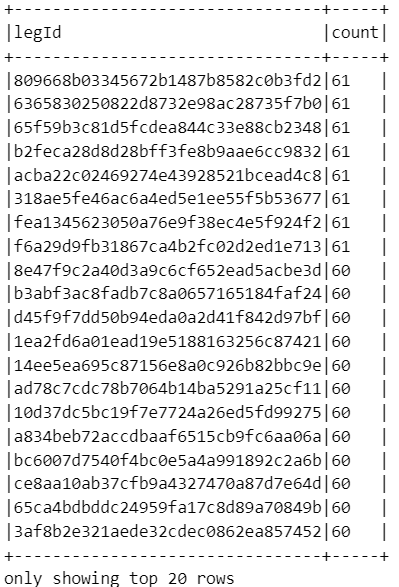


Рисунок 8 – Вывод дубликатов

Дубликаты есть. Покажем на рисунке 9, что они собой представляют (на примере одной записи):

df.filter(col("legId") == "809668b03345672b1487b8582c0b3fd2").show(truncate=False)

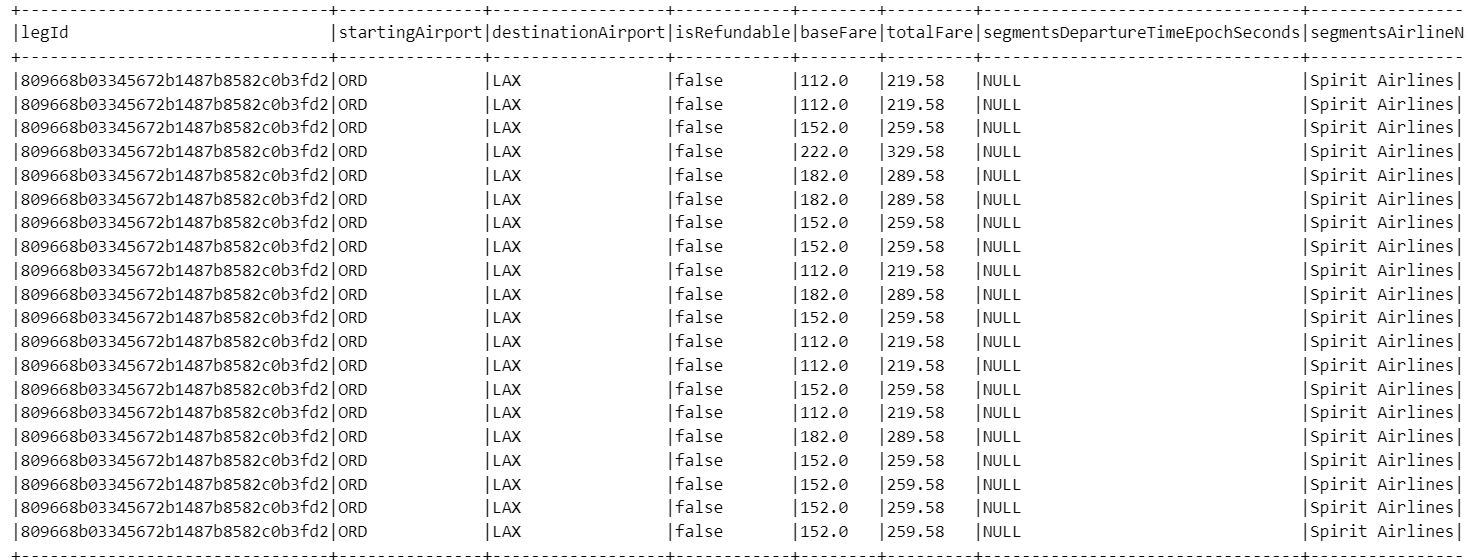


Рисунок 9 – Фильтр данных по ключу

Здесь наблюдается идентичность строк, однако столбцы «baseFare» и «totalFare» разные. Это может говорить о том, что на один рейс покупаются билеты, создавая запись с практически одинаковыми данными. Можно предположить, что время бронирование каждого билета разное, поэтому цены могут отличаться.

Для сокращения времени расчета будем использовать уникальные строки с «ledId». Это приведет к тому, что в рассматриваемой работе будут учитываться только уникальные рейсы (будет представляться анализ рейсов), а не билетов.

df = df.dropDuplicates(["legId"])

Выведем на экран итоговое количество строк после чистки

df.count()

Вывод: 5999739

2.2 Анализ столбца «startingAirport»

Данный столбец представляет собой категориальный признак (название аэропорта из которого начинается рейс).

Определим количество пустых значений с помощью заданного метода:

def count\_nulls(data: DataFrame, column\_name: str) -> None:

    …

    print(f"Число колонок с NULL: {null\_counts} " …

И теперь вызовем данный метод, указав необходимый столбец:

count\_nulls(data=df, column\_name="startingAirport")

Вывод: Число колонок с NULL: 0 (0.00%)

Поскольку пустых значений нету, то сразу перейдем к созданию функции расчета и визуализации распределения категориальных признаков.

def plot\_cat\_distribution(data: DataFrame, column\_name: str, top\_n: int = 20) -> None:

…

plt.show()

Вызовем данный метод, указав необходимый столбец:

plot\_cat\_distribution(data=df, column\_name="startingAirport")

Продемонстрируем построенный график на рисунке 10.

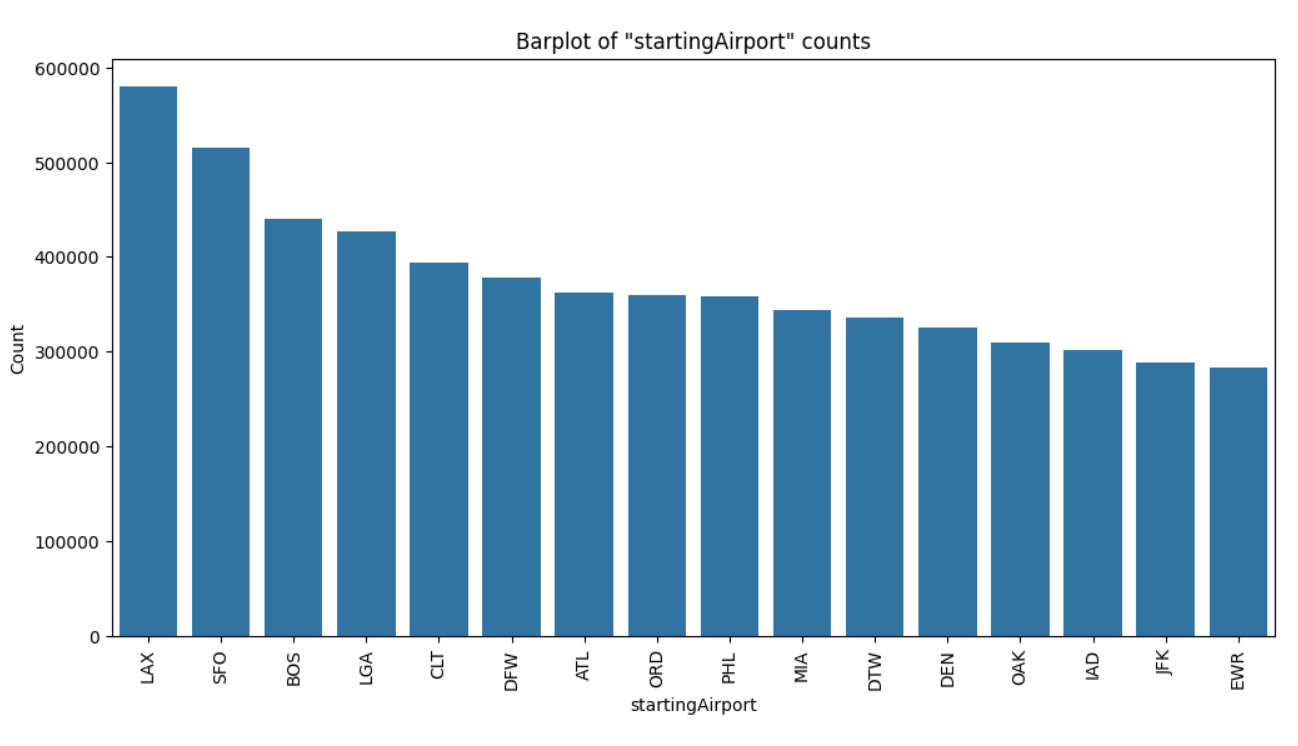


Рисунок 10 – Распределение категориальных признаков по столбцу «startingAirport»

Количество категорий признаков: 16. То есть в дата-сете представлено 16 аэропортов из которых осуществляет вылет.

Один из самых лидирующих аэропортов, откуда вылетают люди является: LAX. Он располагается в Лос-Анджелесе (показатель около 600 тыс.).

2.3 Анализ столбца «destinationAirport»

Данный столбец представляет собой категориальный признак. Первоначально узнаем, сколько нулевых значений присутствует:

count\_nulls(data=df, column\_name="destinationAirport")

Число колонок с NULL: 0 (0.00%)

Поскольку снова значений нету, то сразу перейдем к созданию функции расчета и визуализации распределения категориальных признаков (рис. 11).

plot\_cat\_distribution(data=df, column\_name="destinationAirport")

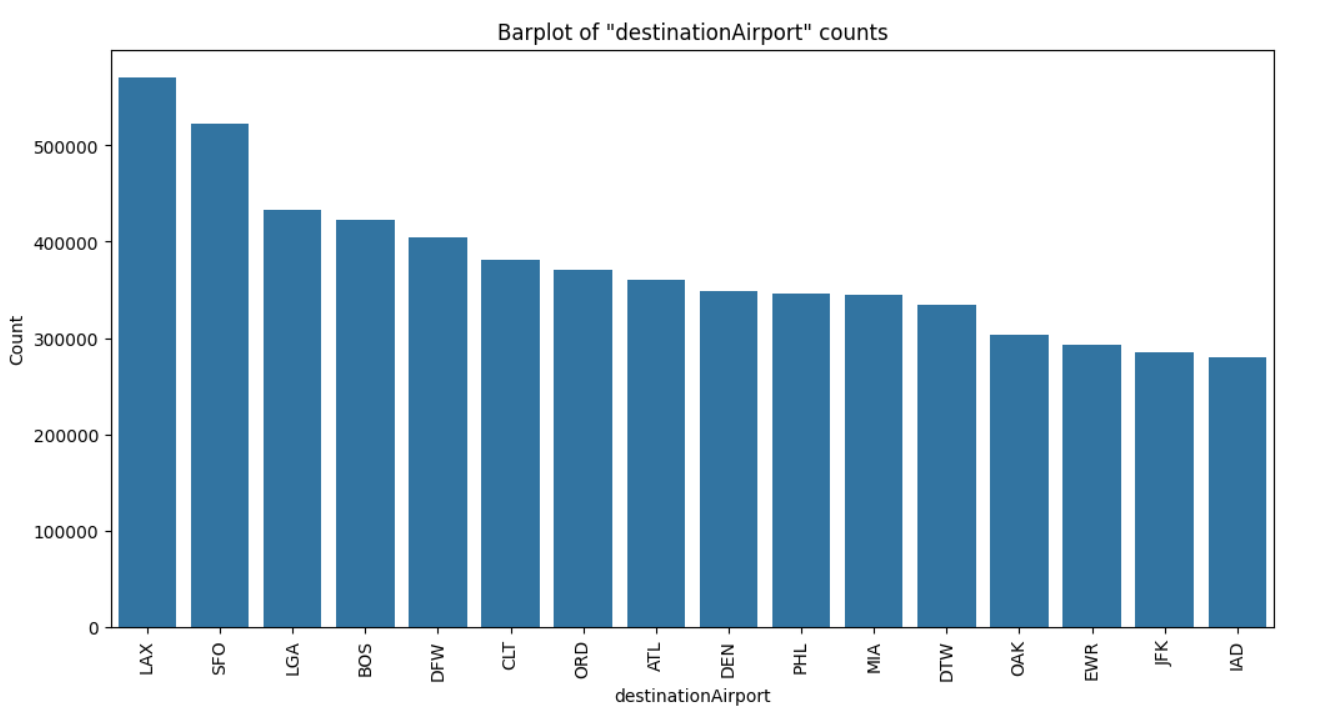


Рисунок 11 – Аэропорты назначения

Как видно из рисунка, самым популярным аэропортом назначения является «LAX» (более 500 тыс. раз)

2.4 Анализ столбца «isRefundable»

Данный столбец представляет собой информацию, которая показывает, что за этот рейс можно вернуть деньги обратно. Столбец хранит только данные True или False, таким образом, его признак является булевым.

Проверим столбец на пустые значения:

count\_nulls(data=df, column\_name="isRefundable")

Число колонок с NULL: 0 (0.00%)

Построим график и продемонстрируем его (рис. 12):

plot\_cat\_distribution(data=df, column\_name="isRefundable")

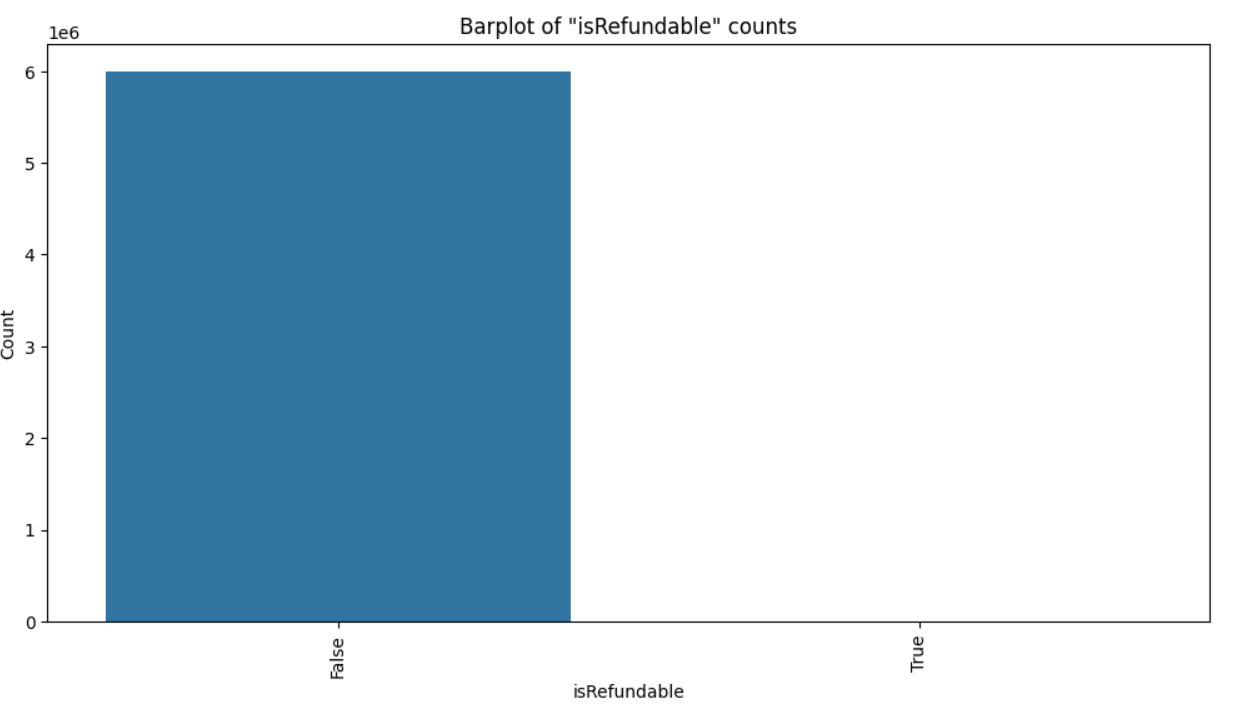


Рисунок 12 – Распределение категориальных признаков по столбцу «isRefundable»

Исходя из графика можно сделать вывод о том, что ни за один рейс нельзя вернуть деньги.

2.5 Анализ столбца «baseFare», «totalFare»

Данные столбцы приведены в одном разделе, поскольку имеют практически одинаковый смысл. Столбец «baseFare» (количественный признак) представляют собой базовую стоимость билета, а столбец «totalFare» (количественный признак) учитывает налоги и сборы вместо со стоимостью.

Представим функцию, позволяющую рассчитывать статистические показатели данных в столбцах и построим диаграмму «ящик с усами» для оценки наличия выбросов:

def plot\_boxplots(data: DataFrame, columns: list[str], sample\_fraction: float = 0.1) -> None:

…

    print(f"Минимальное значение:          {min\_value:.2f}")

    print(f"Среднее значение:              {mean\_value:.2f}")

    print(f"Среднеквадратичное отклонение: {std\_value:.2f}")

    print(f"Первый квартиль:               {q1:.2f}")

    print(f"Медиана:                       {median:.2f}")

    print(f"Третий квартиль:               {q3:.2f}")

print(f"Максимальное значение:         {max\_value:.2f}")

…

Применим данную данную функцию к столбцу «baseFare»:

plot\_boxplots(data=df, columns=["baseFare"])

Представим результаты на рисунке 13:

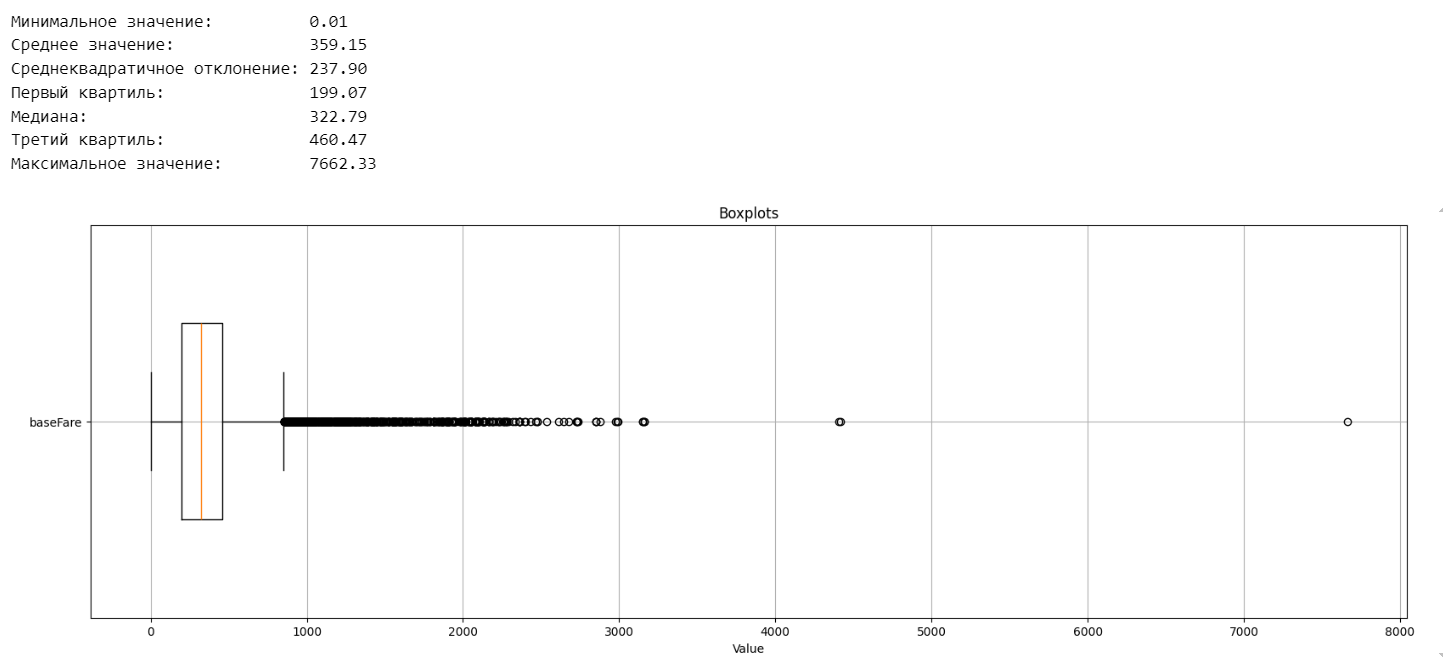


Рисунок 13 – Диаграмма «ящик с усами»

Как видно из графика, наблюдается большое количество выбросов. Создадим функцию для визуального распределения категориального признака для более тщательного исследования.

def plot\_quant\_distribution(data: DataFrame, column: str, num\_bins: int = 200) -> None:

…

plt.xlabel("Value")

    plt.ylabel("Count")

    plt.title(f"Распределение количественного признака \"{column}\"")

    plt.grid(True)

    plt.show()

Применим данный метод и представим график на рисунке 14:

plot\_quant\_distribution(data=df, column="baseFare")



Рисунок 14 – Распределение количественного признака «baseFare»

Установим, что данные не превышают 1500. Обрежем эти выбросы, установив границу в 1500:

df = df.withColumn(

    "baseFare",

    when(col("baseFare") > 1500.0, 1500.0)

        .otherwise(col("baseFare"))

)

Представим график (рис. 15):

plot\_quant\_distribution(data=df, column="baseFare")

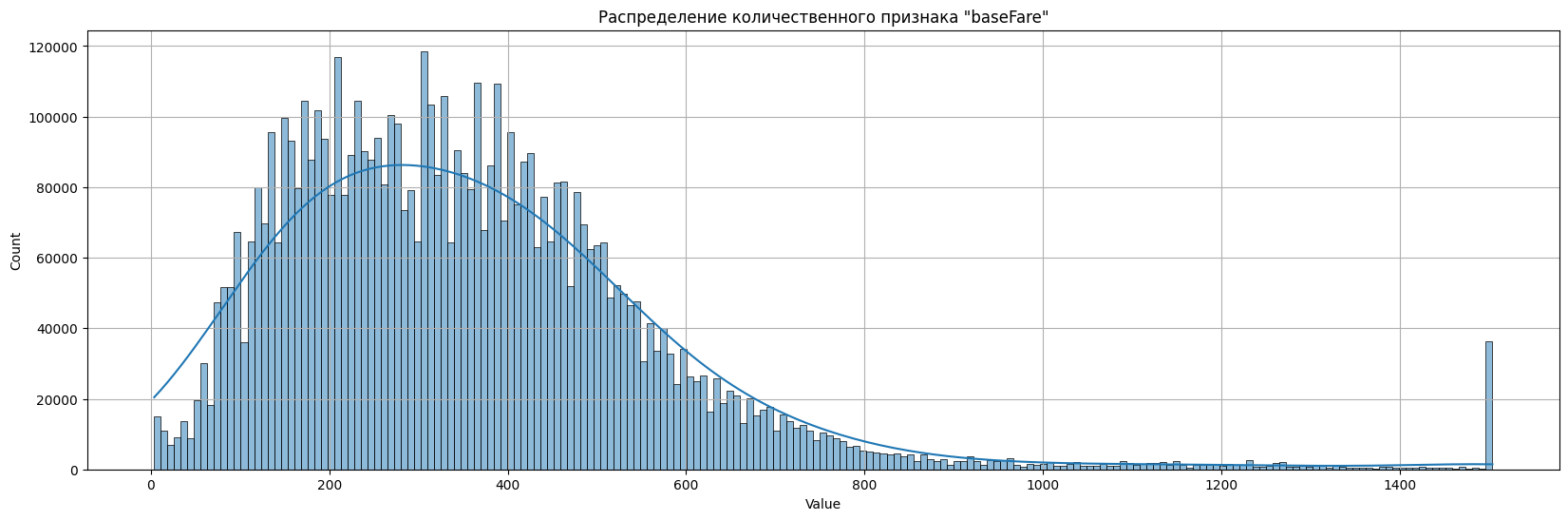


Рисунок 15 – Распределение количественного признака «baseFare»

Проведем аналогичную работу со столбцов «totalFare». Построим диаграмму «ящик с усами» (рис. 16):

plot\_boxplots(data=df, columns=["totalFare"])

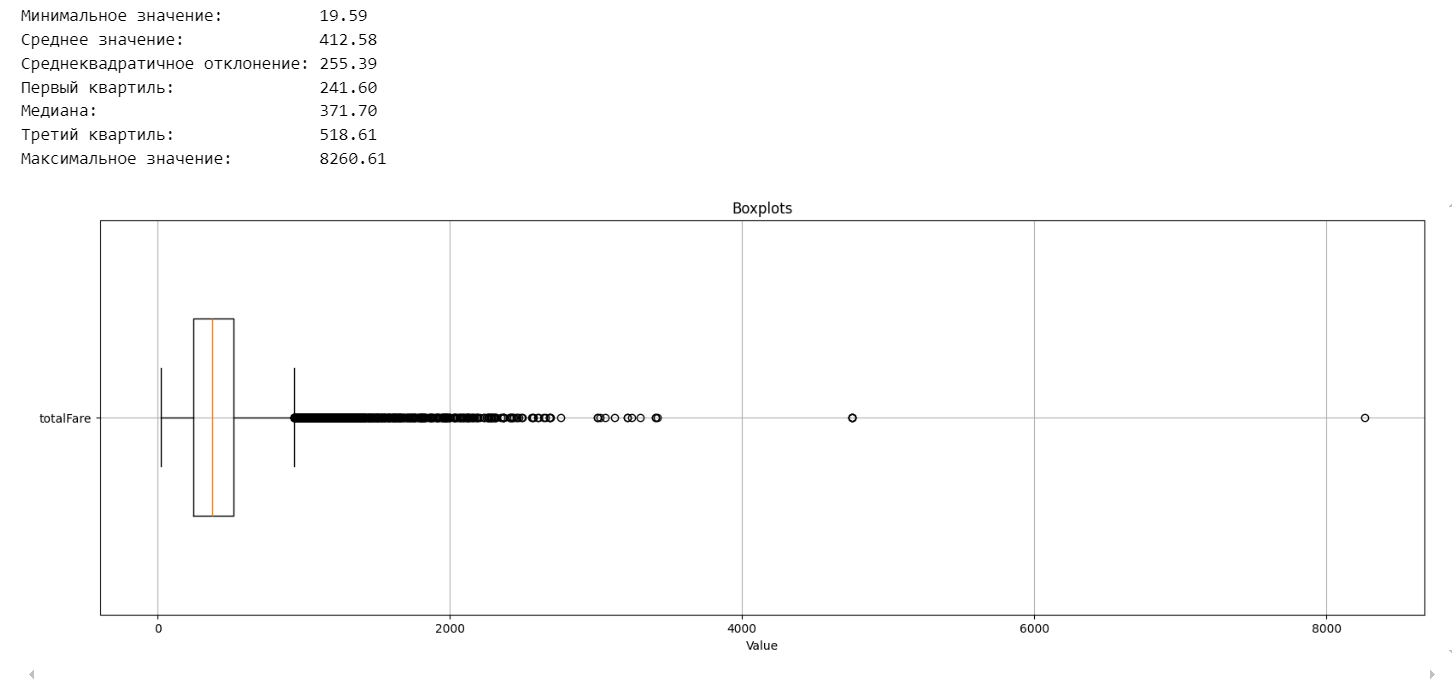


Рисунок 16 – Диаграмма «ящик с усами»

Представим функцию для визуального распределения категориального признака (рис. 17):

plot\_quant\_distribution(data=df, column="totalFare")

Обрежем выбросы, установив границу в 1800.

df = df.withColumn(

    "totalFare",

    when(col("totalFare") > 1750.0, 1750.0)

        .otherwise(col("totalFare"))

)

plot\_quant\_distribution(data=df, column="totalFare")



Рисунок 17 – Распределение количественного признака «totalFare»

Исходя из столбца «totalFare», средний показатель ценник за билет составляет 412,58 USD, а показатель по столбцу «baseFare» 359,15 USD.

В рамках рассматриваемых значений большое количество рейсов имеют небольшую дистанцию полета (исходя из цен). Поэтому выбросами считались те значения, которые имели большую стоимость за билет, а значит и большое расстояние. Математическое обоснование приведено в последующем пункте, разбирающий столбец «segmentsDistance».

2.5 Анализ столбцов «segmentsDepartureTimeEpochSeconds» и «segmentsDurationInSeconds»

Приведенные столбцы в данном пункте имеют схожий смысл. Первый представляет собой время вылета, а второй время полета.

В столбце «segmentsDepartureTimeEpochSeconds» данные располагают числовой (количественный признак), а в «segmentsDurationInSeconds» (из-за разделителя, хотя по своей сути данные несут количественный признак).

Введем функцию, определяющую количество NULL-значений в столбце:

count\_nulls(data=df, column\_name="segmentsDepartureTimeEpochSeconds")

Вывод: Число колонок с NULL: 5348124 (89.14%)

Видно, что столбец «segmentsDepartureTimeEpochSeconds» содержит большое количество пропущенных значений, поэтому целесообразно удалить столбец.

Проверим колонку «segmentsDurationInSeconds» на пустые значения:

count\_nulls(data=df, column\_name="segmentsDurationInSeconds")

Вывод: Число колонок с NULL: 0 (0.00%)

Столбец «segmentsDurationInSeconds» необходимо преобразовать в количественный признак, для этого нужно удалить «||» и просуммировать числа. Приведем метод:

def parse\_duration(duration\_str):

    …

    return total\_duration

Таким образом, данный метод будет суммировать значения в ячейке и переводить секунды в часы (для удобства).

…

parse\_duration\_udf = F.udf(parse\_duration, FloatType())

df = df.withColumn(“totalDurationInHour”,

parse\_duration\_udf(F.col("segmentsDurationInSeconds")))

…

Построим диаграмму ящик с усами на основании нового столбца «totalDurationInHour» (рис 18):

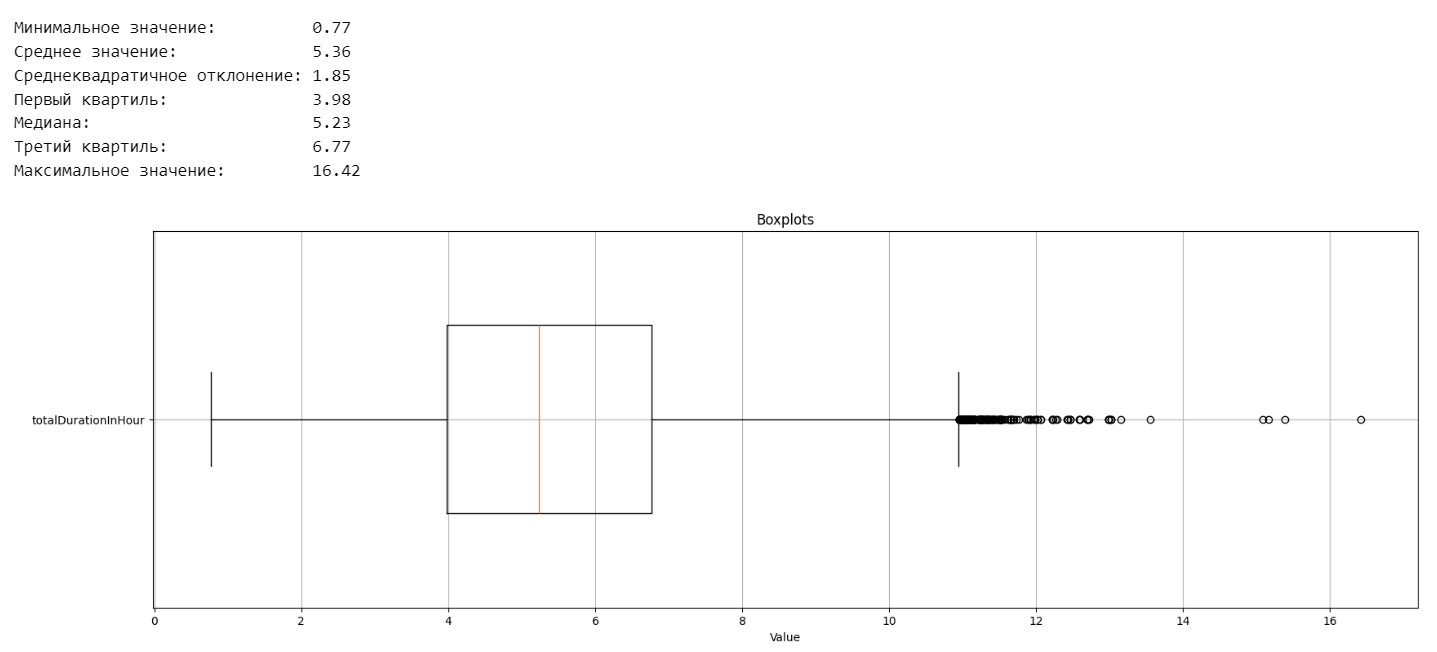


Рисунок 18 – Диаграмма «ящик с усами»

Для более тщательного исследования построим функцию распределения количественного признака (рис. 19).

plot\_quant\_distribution(data=df, column="totalDurationInHour")

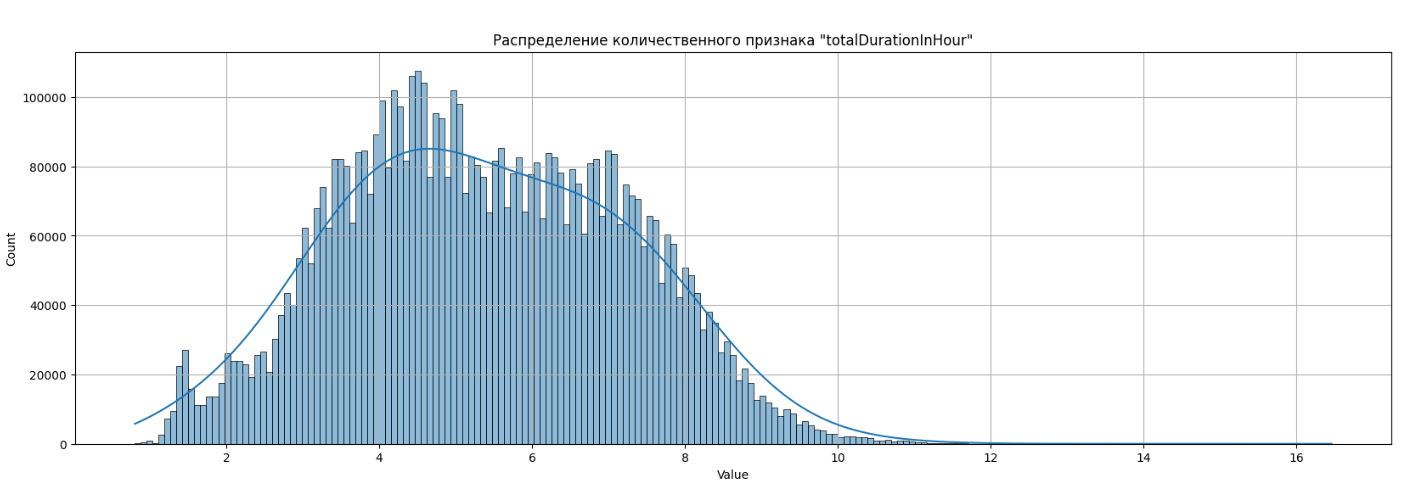


Рисунок 19 – Функция распределения количественного признака

Исходя из графика наблюдается, что почти все данные не превышают значения 11, но при этом наблюдается малое количество довольно сильных выбросов. Обрежем эти выбросы, установив для них максимальную границу.

df = df.withColumn(

    "totalDurationInHour",

    when(col("totalDurationInHour") > 11.0, 11.0)

        .otherwise(col("totalDurationInHour"))

)

plot\_quant\_distribution(data=df, column="totalDurationInHour")

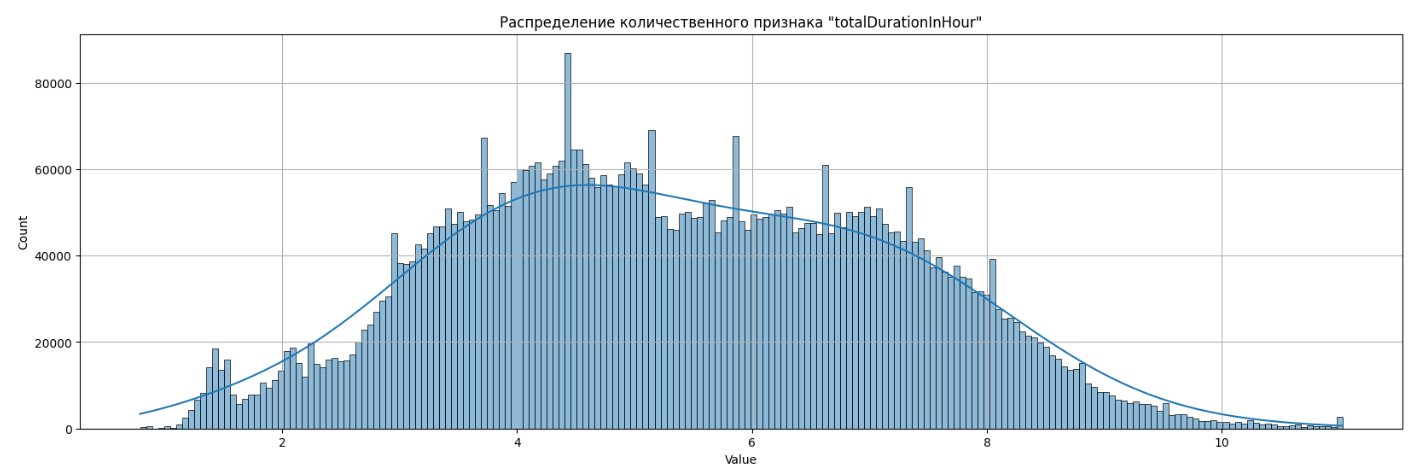


Рисунок 20 – Функция распределения количественного признака

Как видно из показанных рисунков, полет на самолете из текущих рейсов в среднем может занимать от 5 часов.

2.6 Анализ столбца «segmentsAirlineName»

Данный столбец представляет собой категориальный признак (название авиакомпаний).

Проведем поиск количества пустых значений:

count\_nulls(data=df, column\_name="segmentsAirlineName")

Вывод: Число колонок с NULL: 0 (0.00%)

Поскольку пустых значений нету, то выполним дальнейшую обработку данных и выведем результаты на графике (рис.

Выполним обработку данных

def split\_names\_and\_replace\_unknown(df: DataFrame, column\_name: str) -> DataFrame:

…

df\_cleaned = df\_exploded.withColumn(column\_name,

F.when(F.trim(F.col(column\_name)) == "",

F.lit("Unknown")).otherwise(F.col(column\_name)))

    return df\_cleaned

def prepare\_data\_for\_plot(df: DataFrame, column\_name: str) -> DataFrame:

return split\_names\_and\_replace\_unknown(df, column\_name)

prepared\_df = prepare\_data\_for\_plot(df, column\_name="segmentsAirlineName")

plot\_cat\_distribution(data=prepared\_df, column\_name="segmentsAirlineName")

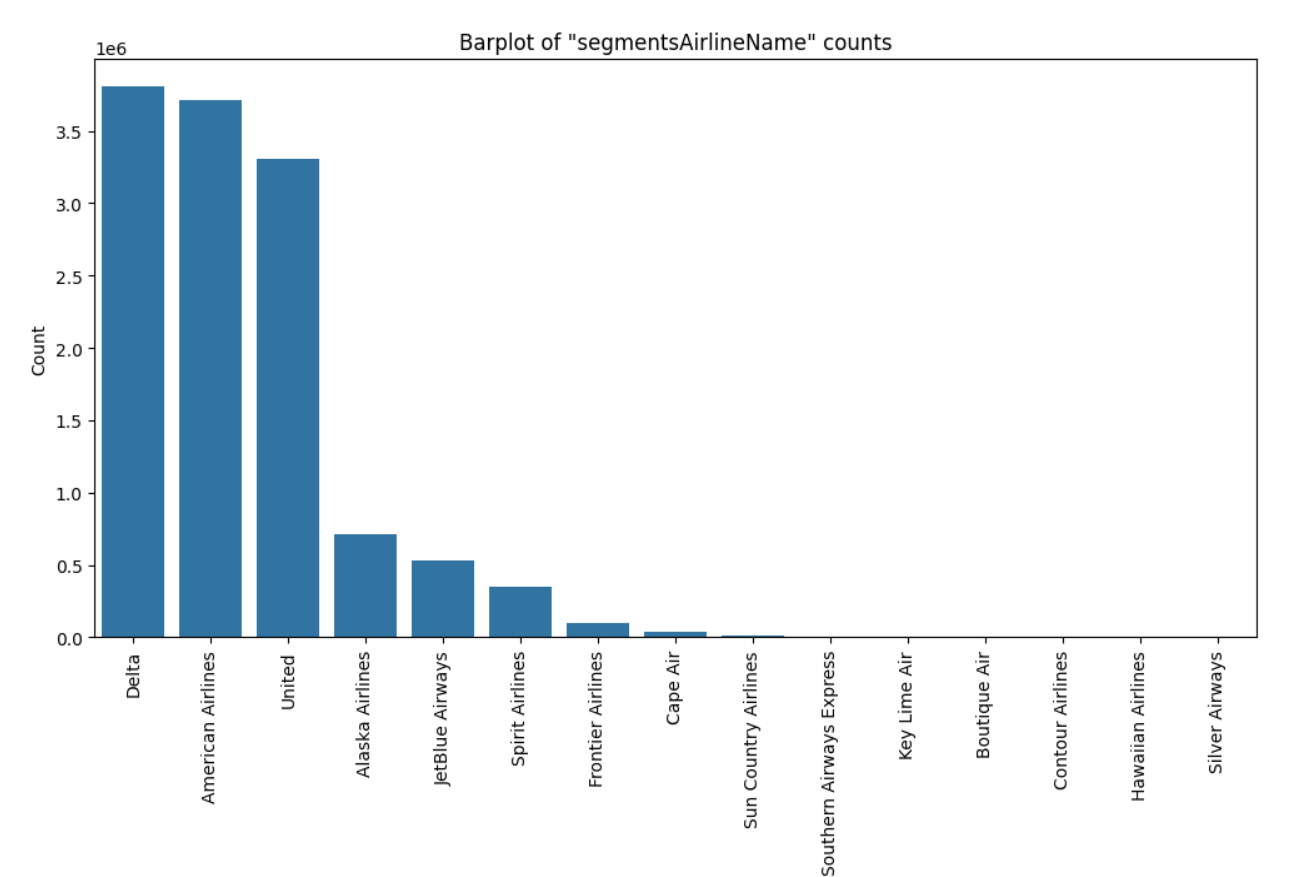


Рисунок 19 – Популярный авиакомпании

Как видно из графика, популярная авиакомпания – «Delta».

2.7 Анализ столбца «segmentsEquipmentDescription»

Данный столбец, который несет категориальный признак, представляет собой описание самолётов, которые выполняют рейс.

Выведем на экран количество пропущенных значений:

count\_nulls(data = df, column\_name = "segmentsEquipmentDescription")

Вывод: Число колонок с NULL: 48536 (0.81%)

Поскольку пропущенные значения присутствуют, то заменим их на «||».

df = df.fillna({"segmen tsEquipmentDescription": "||"})

После преобразования дата-фрейма выведем на экран количество пропущенных значений:

count\_nulls(data=df, column\_name="segmentsEquipmentDescription")

Вывод: Число колонок с NULL: 0 (0.00%)

Построим график (рис. 20).

prepared\_df = prepare\_data\_for\_plot(df,

column\_name="segmentsEquipmentDescription")

plot\_cat\_distribution(data=prepared\_df, column\_name="segmentsEquipmentDescription")

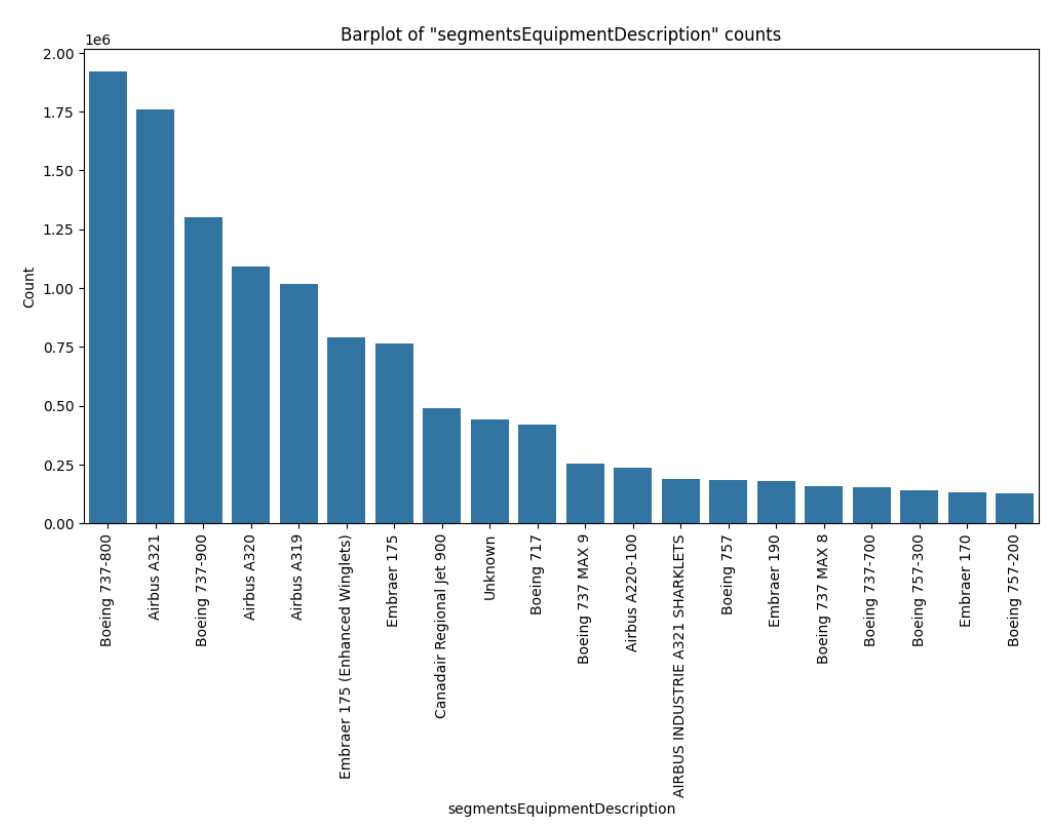


Рисунок 19 – Самолеты, выполняющие рейс

Один из самых популярных самолётов является «Boeing 737-800».

2.8 Анализ столбца «segmentsDistance»

Данный столбец представляет собой информацию о расстоянии рейса. Он несет количественный показатель.

Найдем нулевые значения:

count\_nulls(data=df, column\_name="segmentsDistance")

Вывод: Число колонок с NULL: 0 (0.00%)

Поскольку нулевых значений нету, то проведем дальнейшую обработку данных и выведем результаты на графике (рис. 20):

df = df.filter(~(col('segmentsDistance').isin("None", "None||None", "None||None||None", "None||None||None||None")))

…

def calculate\_distance(distance\_str):

…

…

Построим диаграмму «ящик с усами» (рис. 20):

plot\_boxplots(data=df, columns=["calculatedDistance"])

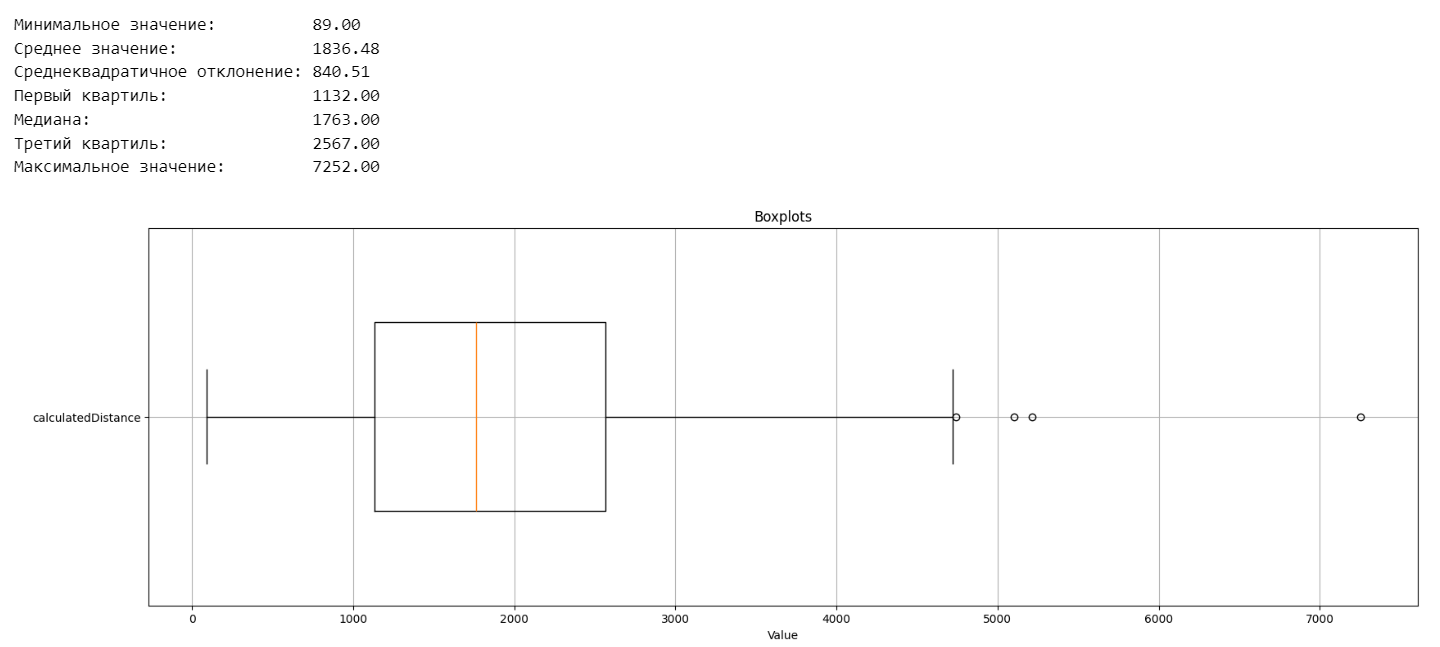


Рисунок 20 – Диаграмма «ящик с усами»

Построим функцию распределения количественного признака (рис. 21).

plot\_quant\_distribution(data=df, column="calculatedDistance")

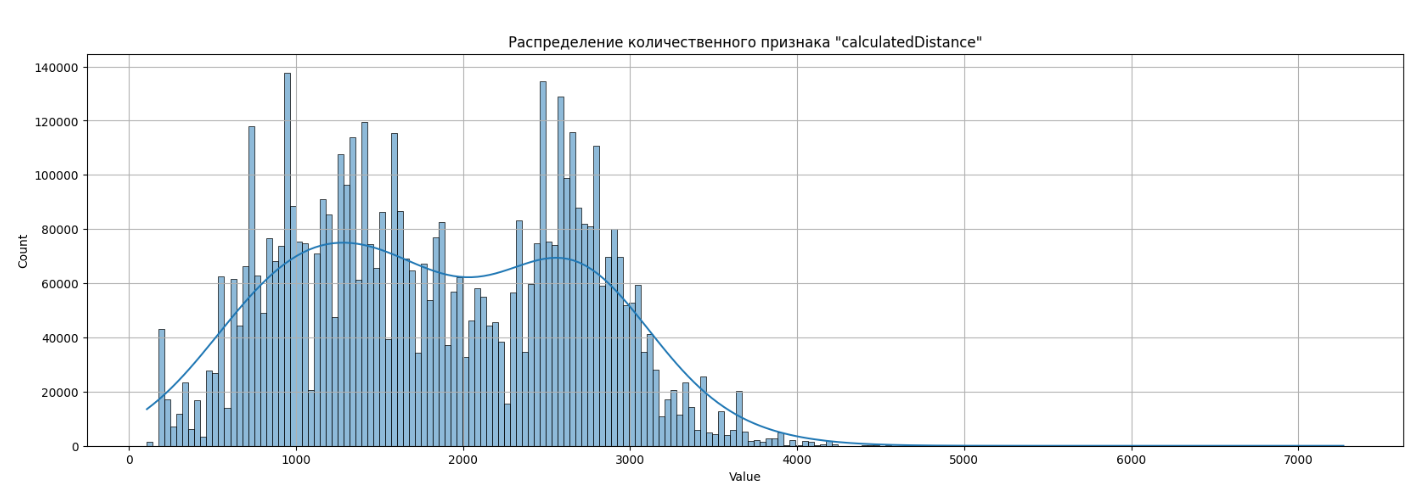


Рисунок 21 – Функция распределения количественного признака

Исходя из представленной информации среднее расстояние рейса составляет 1800 км. Поскольку это последний столбец, представляющий количественный признак, то изобразим корреляционную матрицу (рис. 22), чтобы увидеть, насколько сильно связаны между собой столбцы. Для этого опишем метод:  
 def compute\_and\_visualize\_correlation\_matrix(data: DataFrame, columns: list[str]) ->

None:

…

Внесем необходимые столбцы при использовании метода:

compute\_and\_visualize\_correlation\_matrix(

    data=df, columns=[

        "baseFare", "totalFare",

        "totalDurationInHour", "calculatedDistance"

    ])

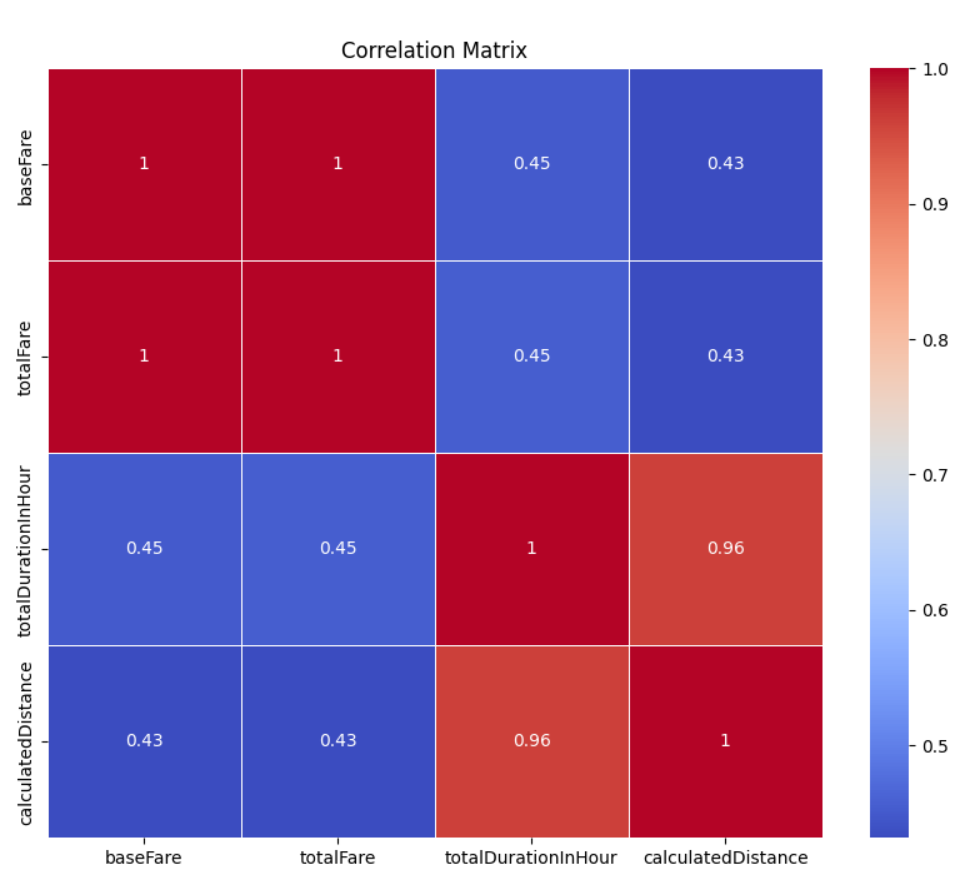


Рисунок 22 – Корреляционная матрица

Исходя из рисунка видна зависимость между показателями: «baseFare» и «totalFare», «totalDuration» и «calculatedDistance».

2.9 Анализ столбца «segmentsCabinCode»

Данный столбец представляет собой код класса обслуживания (бизнес, премиум и т. д.). Он представляет собой категориальный признак данных.

count\_nulls(data=df, column\_name="segmentsCabinCode")

Вывод: Число колонок с NULL: 0 (0.00%)

prepared\_df = prepare\_data\_for\_plot(df, column\_name="segmentsCabinCode")

plot\_cat\_distribution(data=prepared\_df, column\_name="segmentsCabinCode")

Построим график, показывающий каким классом обслуживания чаще всего пользуются пассажиры (рис. 21).

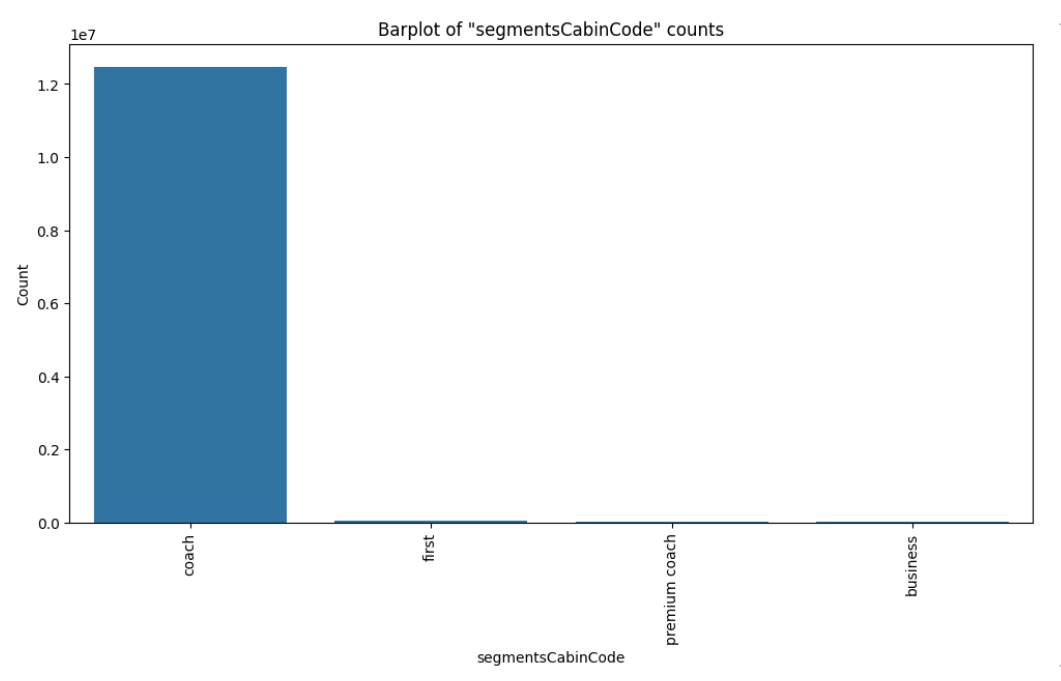


Рисунок 21 – Классы обслуживания

Исходя из графика, можно увидеть, что чаще всего пользуются «coach» (эконом класс). Однако другие классы трудно разглядеть из-за популярности «coach». Поэтому построим логарифмический график, для более наглядного просмотра (рис. 22).

class\_distribution = prepared\_df.groupBy('segmentsCabinCode').count().toPandas()

class\_distribution.columns = ['serviceClass', 'count']

…

plt.show()

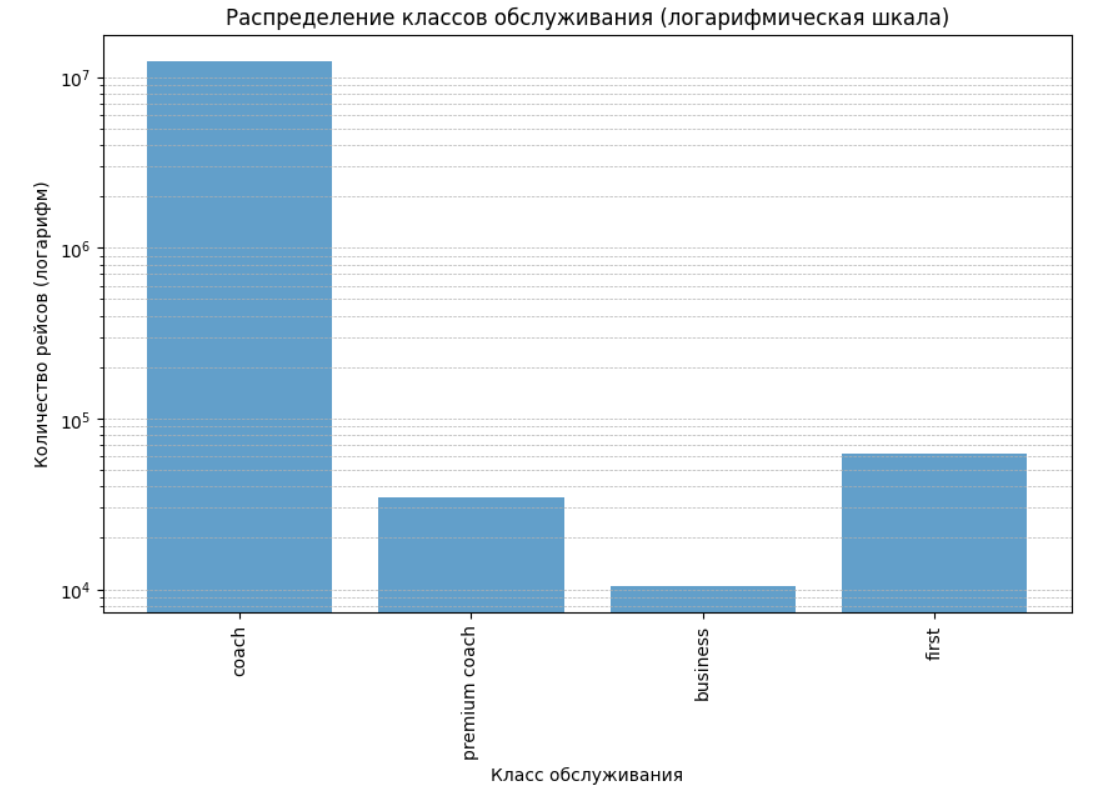


Рисунок 22 – Логарифмический график распределения классов обслуживания

Логарифмический график предоставил возможность выявить, что класс бизнес класс является редко используемым классом в рейсах.

После проведения дополнительной очистки данных в дата-сете при проведении разведочного анализа, количество строк сократилось до 5536924.

Вывод. В результате проведения анализа дата-сета выяснилось, что почти за полгода компания Expedia (туристическая компания) совершила около 6 миллионов рейсов, из них продано около 82 млн. билетов. В среднем каждый рейс занимает 6 часов полета и самый популярный аэропорт назначения (из которого вылетают или в который вылетают – LAX). Данный аэропорт расшифровывается, как международный аэропорт Лос-Анджелес. Однако делать ещё более глубокие выводы трудно, поскольку приведены показатели только от одной компании за период апрель – октябрь во времена/послековидных времен.