В рамках данной курсовой работы было необходимо провести разведочный анализ выбранного набора данных 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.

Для выбора наборы данных были установлены следующие требование:

* Вес более нескольких ГБ
* Отсутствие звуковых, графических и иных сложных видов данных в наборе

Для работы был выбран данные цифровой торговли из неизвестного магазина, который содержит в себе 42448764 записи, имеет вес более чем в несколько ГБ, и несёт в себе только текстовые и числовые значения, что удовлетворяет условиям, предъявляемым к выбираемому набору данных.

Ссылка на датасет: https://www.kaggle.com/datasets/mkechinov/ecommerce-behavior-data-from-multi-category-store?select=2019-Oct.csv

В изначальном наборе данных имеется 9 столбов:

* event\_time – Время операции
* event\_type – Тип операции
* product\_id – Номер продукта
* category\_id – Номер категории
* category\_code – Код категории (название)
* brand - Бренд
* price - Цена
* user\_id – Номер пользователя
* user\_session – Номер сессии

В первой части работы были рассмотрены:

* Загрузка данных из HDFS
* Базовые преобразования данных
* Загрузка преобразованных данных в таблицу Apache Airflow

Датасет был загружен на удалённый узел кластера, после чего был загружен в распределённую файловую систему HDFS с помощью команды:

hdfs dfs -copyFromLocal /path

**Импорт необходимых библиотек и создание объекта конфигурации для Apache Spark**

**import** os

**from** pyspark.sql **import** SparkSession, DataFrame

**from** pyspark **import** SparkConf

**from** pyspark.sql.functions **import** (

regexp\_replace,

regexp\_extract\_all,

col,

lit

)

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

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

conf = SparkConf()

conf.setAppName("lab 1 Test")

conf.setMaster("yarn")

conf.set("spark.submit.deployMode", "client")

conf.set("spark.executor.memory", "12g")

conf.set("spark.executor.cores", "8")

conf.set("spark.executor.instances", "2")

conf.set("spark.driver.memory", "4g")

conf.set("spark.driver.cores", "2")

conf.set("spark.jars.packages", "org.apache.iceberg:iceberg-spark-runtime-3.5\_2.12:1.6.0")

conf.set("spark.sql.extensions", "org.apache.iceberg.spark.extensions.IcebergSparkSessionExtensions")

conf.set("spark.sql.catalog.spark\_catalog", "org.apache.iceberg.spark.SparkCatalog")

conf.set("spark.sql.catalog.spark\_catalog.type", "hadoop")

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

conf.set("spark.sql.catalog.spark\_catalog.io-impl", "org.apache.iceberg.hadoop.HadoopFileIO")

return conf

conf = create\_spark\_configuration()

**Из датасета был создан датафрейм:**

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

**.**option("header", "true")

**.**load(path)

)

Удалены столбцы: event\_time, category\_id, user\_id, user\_session:

df **=** df**.**select(

"event\_type", "product\_id","category\_code", "brand", "price"

)

**Заменён тип данных на нужный:**

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

Args:

data (DataFrame): Исходный DataFrame.

Returns:

DataFrame: Преобразованный DataFrame.

data **=** data**.**withColumn("product\_id",

col("product\_id")**.**cast("Integer"))

data **=** data**.**withColumn("price",

col("price")**.**cast("Float"))

**return** data

df **=** transform\_dataframe(df)

**Создана база данных и сохранена таблица в ней:**

database\_name **=** "lopin\_database1"

create\_database\_sql **=** f"""

CREATE DATABASE IF NOT EXISTS spark\_catalog.{database\_name}

spark**.**sql(create\_database\_sql)

spark**.**catalog**.**setCurrentDatabase(database\_name)

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

Во второй части работы был проведён:

* Разведочный анализ данных
* Проведена работа с Dataframe API фреймворка Apache Spark

Анализ столбца event\_type:

df**.**orderBy("event\_type", ascending**=False**)**.**show()

Отсортировали по столбцу event\_type

df**.**select('event\_type')**.**distinct()**.**show()

Получили 3 уникальных значения: Purchase, view, cart. Категориальный признак

Анализ столбца category\_code:

Создадим функцию для подсчёта нулевых значений в указанном столбце и подсчитаем их:

def count\_nulls(data: DataFrame,

column\_name: str) -> None:

Args:

data (DataFrame): DataFrame, содержащий данные.

column\_name (str): Имя столбца для подсчета null и not null значений.

Returns:

None

null\_counts = data.select(

sum(col(column\_name).isNull().cast("int"))

).collect()[0][0]

not\_null\_counts = data.select(

sum(col(column\_name).isNotNull().cast("int"))

).collect()[0][0]

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

f"({100 \* null\_counts / (null\_counts + not\_null\_counts):.2f}%)")

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

Получили 13515609 (31.84%) колонок с Null. Заменим их на Unknown:

df = df.fillna({"category\_code": "Unknown"})

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

В результате визуализации получили подобную диаграмму:

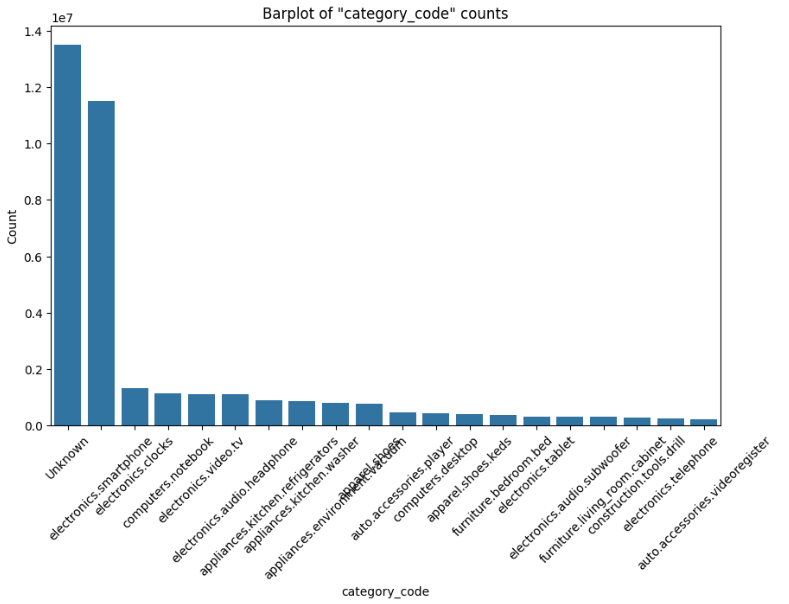


Рисунок 1 – Диаграмма по столбцу category\_code

Анализ столбца product\_id:

Проверим пустые значения:

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

Проверим на совпадения

(

df

.groupBy("product\_id")

.count()

.where("count > 1")

.orderBy("count", ascending=False)

.show()

)

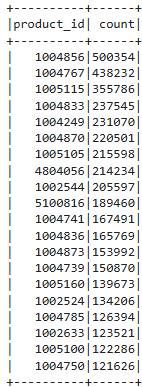


Рисунок 2 – Проверка совпадений

Посмотрим отдельную запись

df.filter(col("product\_id") == "1004856").show()

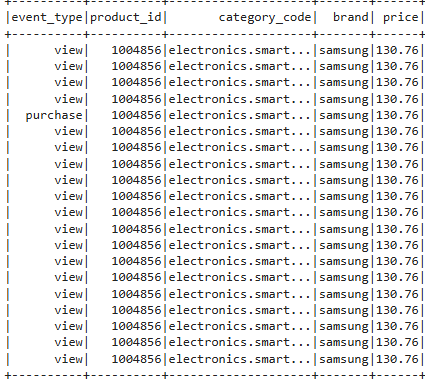


Рисунок 3 – Проверка отдельной записи

В силу специфика набора данных такого рода данные являются нормальными.

Анализ столбца brand:

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

Получили 6113008 (14.40%) колонок с Null. Заменим их на Unidentified:  
count\_nulls(data=df, column\_name="brand"). Визуализируем данные по столбцу brand:

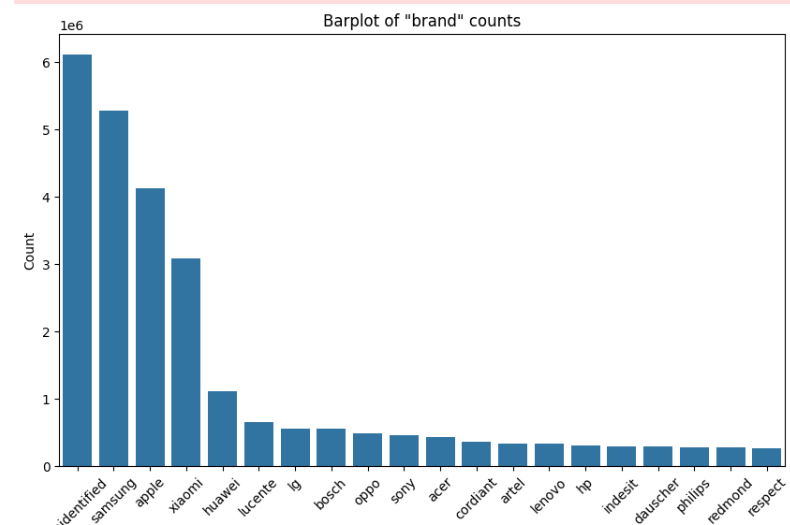


Рисунок 4 – Визуализация по столбцу brand

Анализ столбца price:

Создадим функцию для создания диаграммы: «ящик с усами»:

def plot\_boxplots(data: DataFrame,

columns: list[str],

sample\_fraction: float = 0.1) -> None:

box\_data = []

for column in columns:

quantiles = data.approxQuantile(column, [0.25, 0.5, 0.75], 0.01)

q1, median, q3 = quantiles

iqr = q3 - q1

lower\_bound = q1 - 1.5 \* iqr

upper\_bound = q3 + 1.5 \* iqr

filtered\_df = data.filter((col(column) >= lower\_bound) & (col(column) <= upper\_bound))

outliers\_df = data.filter((col(column) < lower\_bound) | (col(column) > upper\_bound))

min\_value = data.agg({column: "min"}).collect()[0][0]

mean\_value = data.agg({column: "mean"}).collect()[0][0]

std\_value = data.agg({column: "std"}).collect()[0][0]

max\_value = data.agg({column: "max"}).collect()[0][0]

lower\_bound = max(lower\_bound, min\_value)

upper\_bound = min(upper\_bound, max\_value)

outliers = []

if not outliers\_df.isEmpty():

sampled\_outliers\_df = outliers\_df.sample(sample\_fraction)

outliers = (

sampled\_outliers\_df

.select(column)

.limit(1000)

.collect()

)

outliers = [row[column] for row in outliers]

if min\_value < lower\_bound and min\_value not in outliers:

outliers.append(min\_value)

if max\_value > upper\_bound and max\_value not in outliers:

outliers.append(max\_value)

box\_data.append({

'whislo': lower\_bound, # Нижняя граница усов

'q1': q1, # Первый квартиль

'med': median, # Медиана

'q3': q3, # Третий квартиль

'whishi': upper\_bound, # Верхняя граница усов

'fliers': outliers # Выбросы

})

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}")

fig, ax = plt.subplots(figsize=(20, 6))

ax.bxp(box\_data,

vert=False,

positions=range(1, len(columns) + 1), widths=0.5)

ax.set\_yticks(range(1, len(columns) + 1))

ax.set\_yticklabels(columns)

ax.set\_xlabel('Value')

ax.set\_title('Boxplots')

ax.grid(True)

plt.show()

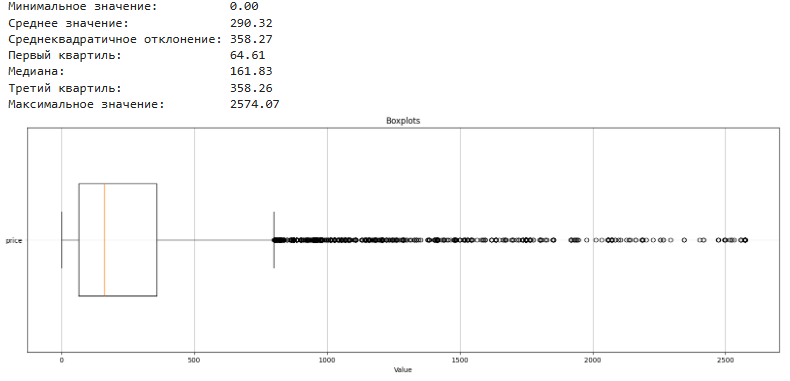


Рисунок 5 – Диаграмма «Ящик с усами» для столбца price

Создадим функцию для построения гистограммы для количественной переменной

def plot\_quant\_distribution(data: DataFrame,

column: str,

num\_bins: int = 200) -> None:

min\_value = data.agg({column: "min"}).collect()[0][0]

max\_value = data.agg({column: "max"}).collect()[0][0]

bin\_size = (max\_value - min\_value) / num\_bins

data = data.withColumn(

"bin",

floor((col(column) - min\_value) / bin\_size)

)

bin\_counts = data.groupBy("bin").count()

bin\_counts\_pd = bin\_counts.limit(1000).toPandas()

bin\_edges = [min\_value + i \* bin\_size for i in range(num\_bins + 2)]

bin\_centers = [

(bin\_edges[i] + bin\_edges[i + 1]) / 2 for i in range(num\_bins + 1)

]

bin\_counts\_pd['bin\_center'] = bin\_counts\_pd['bin'].apply(

lambda x: bin\_centers[int(x)]

)

plt.figure(figsize=(20, 6))

sns.histplot(data=bin\_counts\_pd, x="bin\_center",

weights="count", kde=True, bins=num\_bins + 1)

plt.xlabel("Value")

plt.ylabel("Count")

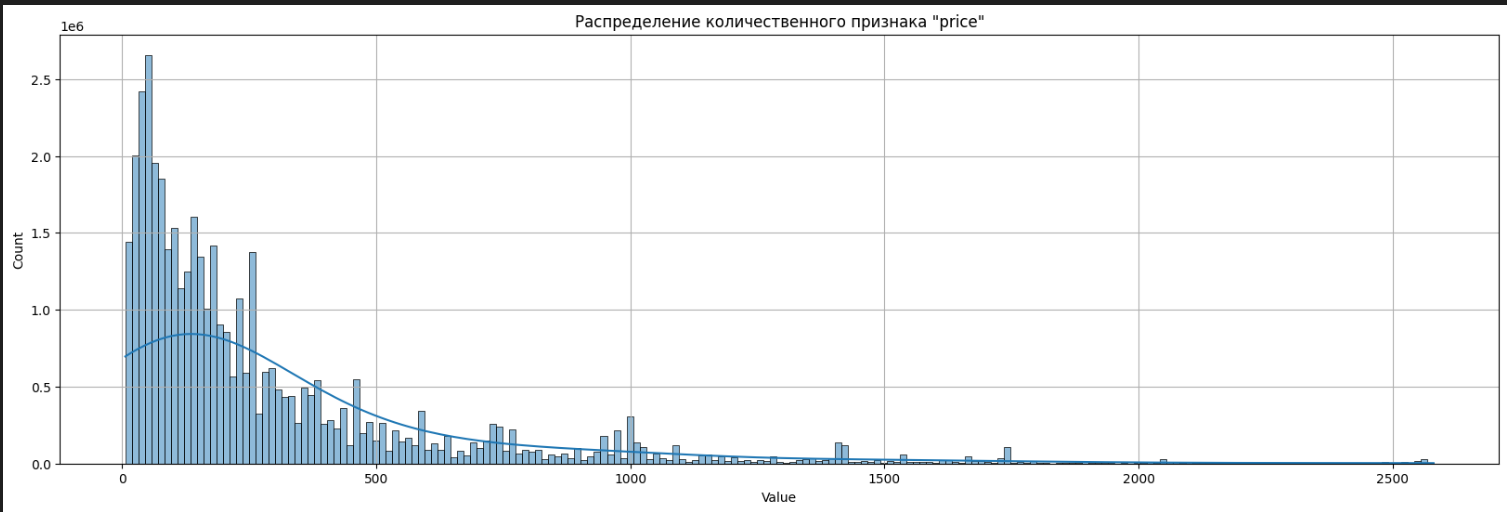
plt.title(f"Распределение количественного признака \"{column}\"")

plt.grid(True)

plt.show()

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

На выводе получим:



Ограничим данные, чтобы нивелировать выбросы в пределах 2000:

df = df.withColumn(

"price",

when(col("price") > 2000, 2000)

.otherwise(col("price"))

)



В датасете только один количественный признак, поэтому ради практики была проведена проверка на корреляцию между price и product\_id:

Создадим функцию для вычисления и визуализации корреляционной матрицы:  
def compute\_and\_visualize\_correlation\_matrix(data: DataFrame,

columns: list[str]) -> None:

corr\_matrix = {}

for col1 in columns:

corr\_matrix[col1] = {}

for col2 in columns:

corr\_value = data.select(corr(col1, col2)).collect()[0][0]

corr\_matrix[col1][col2] = corr\_value

corr\_matrix\_pd = pd.DataFrame(corr\_matrix)

plt.figure(figsize=(10, 8))

sns.heatmap(corr\_matrix\_pd, annot=True, cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Matrix')

plt.show()

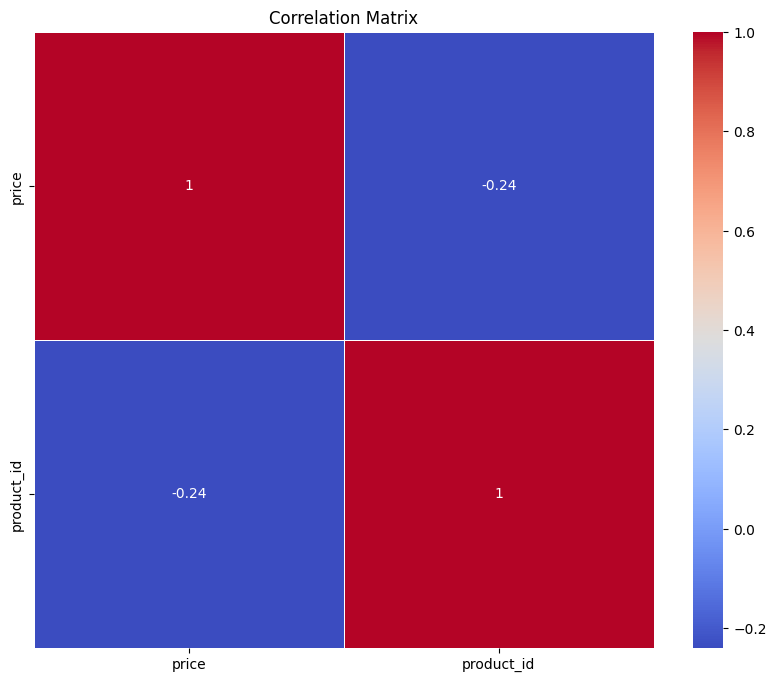
compute\_and\_visualize\_correlation\_matrix(

data**=**df, columns**=**[

"price", "product\_id"

]

)



Выводы по проведённой работе:

* Познакомился с понятиями большие данные и способами их обработки
* Познакомился с инструментом Apache Spark и возможностями, которые он предоставляет
* Получил представление об инструментах экосистемы Hadoop: HDFS и YARN
* Поработал с табличным форматом для больших данных Apache Iceberg
* Получил навыки выполнения разведочного анализа данных использованием Pyspark