Департамент образования и науки города Москвы
Государственное автономное образовательное учреждение
высшего образования города Москвы
«Московский городской педагогический университет»
Институт цифрового образования
Департамент информатики управления и технологий

Ли Александр Андреевич БД-241м

# Практическая работа 1.2. Обработка данных с использованием Apache Spark и Python (PySpark)

Направление подготовки/специальность
38.04.05 - Бизнес-информатика
Бизнес-аналитика и большие данные
(очная форма обучения)
ФИО преподавателя
Босенко Тимур Муртазович
Вариант 12

Москва
2025

Цель: освоение основ работы с Apache Spark и его интеграцией с Python через библиотеку PySpark. Научиться обрабатывать большие объемы данных, используя распределенные вычисления, а также применение базовых операции с RDD (Resilient Distributed Datasets) и DataFrame, работа с SQL-запросами в Spark SQL, а также визуализация результатов обработки данных.

Ход работы

Запустили Hadoop и yarn

**Start-dfs.sh**

**Start-yarm.sh**

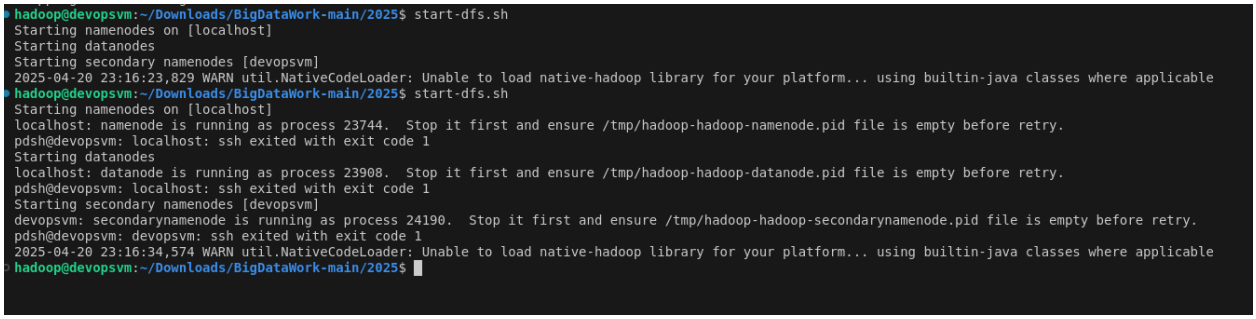


рис.1 запуск Hadoop

Проверяем запущен ли Hadoop и yarn

Перейдем по ссылке http://localhost:9870

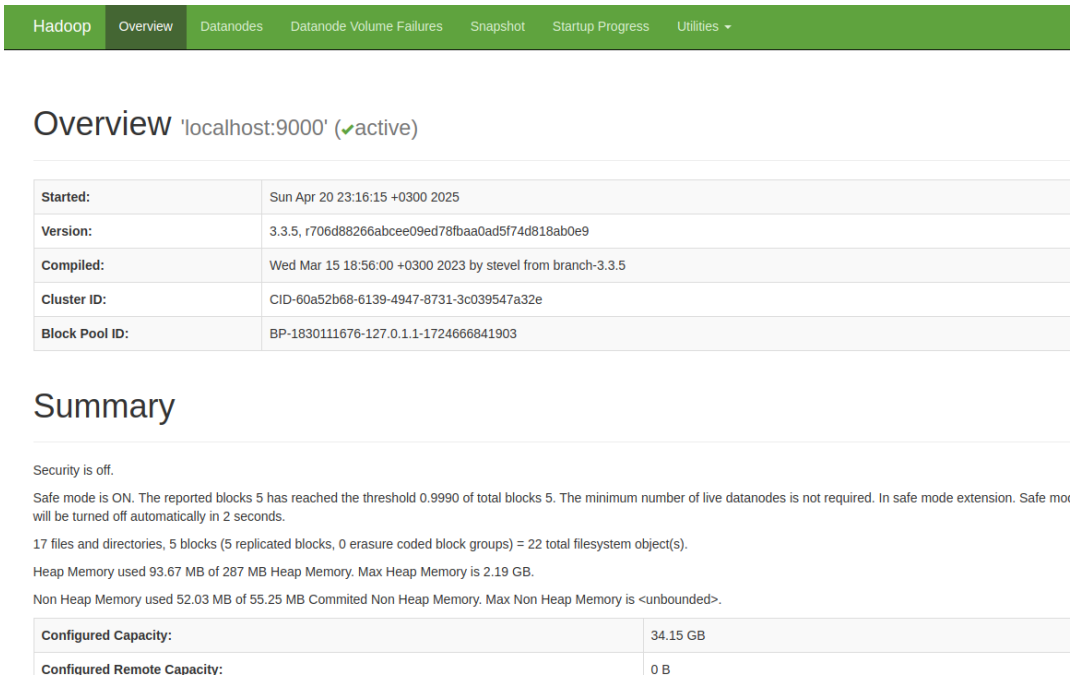


Рис.2 Проверяем запущен ли Hadoop

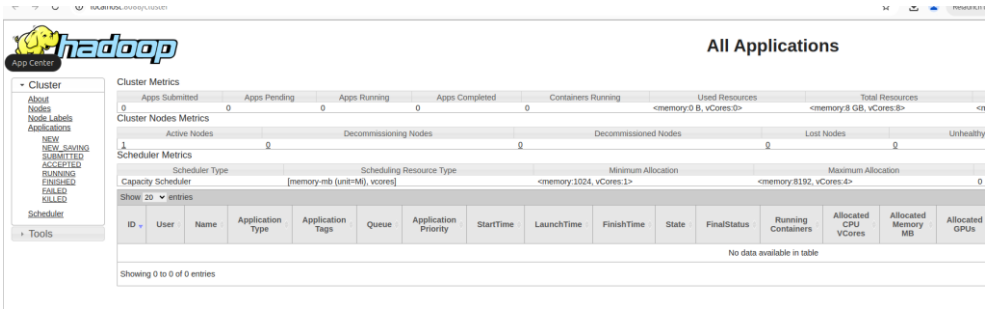


Рис.3 Проверяем запущен ли yarn

Создаем пользователя user7

**Hdfs dfs -mkdir -p /user7/sparkdir**

```
hadoop@devopsvm:~$ hdfs dfs -mkdir -p /user7/sparkdir
2025-04-21 00:13:16,333 WARN util.NativeCodeLoader: Unable to load native-hadoop library for your pla
mkdir: Call From devopsvm/127.0.1.1 to localhost:9000 failed on connection exception: java.net.Connec
hadoop/ConnectionRefused
hadoop@devopsvm:~$
```

Рис.4 Создание директории для данных

Проверяем созданного пользователя user7

| | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| ☐ | drwxr-xr-x | hadoop | supergroup | 0 B | Apr 21 00:15 | 0 | 0 B | user7 | 🗑 |

Рис.5 Проверяем созданного пользователя user 7

Загружаем данные

**Hdfs dfs -put /home/Hadoop/lab_01_2/BigDataWork-main/2025/data/data* /user7/sparkdir/**

Show 25 entries Search:

| | Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name | |
|---|---|---|---|---|---|---|---|---|---|
| ☐ | -rw-r--r-- | hadoop | supergroup | 847.88 KB | Apr 21 00:25 | 1 | 128 MB | access.log | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 197.95 MB | Apr 21 00:25 | 1 | 128 MB | brooklyn_sales_map.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 483.13 KB | Apr 21 00:25 | 1 | 128 MB | digits.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 222.08 MB | Apr 21 00:25 | 1 | 128 MB | food-inspections.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 4.47 KB | Apr 21 00:25 | 1 | 128 MB | idiot.txt | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 6.44 KB | Apr 21 00:25 | 1 | 128 MB | ip.tsv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 2.85 KB | Apr 21 00:25 | 1 | 128 MB | kmeans.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 2.39 KB | Apr 21 00:25 | 1 | 128 MB | logit_test.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 612 B | Apr 21 00:25 | 1 | 128 MB | logit_train.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 2.33 MB | Apr 21 00:25 | 1 | 128 MB | ratings.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 3.45 MB | Apr 21 00:25 | 1 | 128 MB | reviews_sample.json | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 120.63 KB | Apr 21 00:25 | 1 | 128 MB | sales.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 114 B | Apr 21 00:25 | 1 | 128 MB | sales_header.csv | 🗑 |
| ☐ | -rw-r--r-- | hadoop | supergroup | 954 B | Apr 21 00:25 | 1 | 128 MB | stopwords.txt | 🗑 |

Showing 1 to 14 of 14 entries Previous 1 Next

Рис.6 Загруженные данные

Запускаем файл sparkwordcount_1hadoop.ipynb, создаем спарксессию и читаем данные из hdfs.

```
from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("WordCount App") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Установка количества разделов для shuffle операций
spark.conf.set("spark.sql.shuffle.partitions", "50")

# Чтение данных из HDFS (текстовый файл
file_path = "hdfs://localhost:9000/user7/sparkdir/data/idiot.txt"

df = spark.read.text(file_path)  # Используем .text, так как это текстовый файл
udio Code
# Печать первых нескольких строк
df.show()
```

```
25/04/21 01:03:58 WARN SparkSession: Using an existing Spark session; only runtime SQL configurations will take effect.
```

```
+--------------------+
|               value|
+--------------------+
|Beauty will save ...|
|Don’t let us forg...|
|It is better to b...|
|There is somethin...|
|Lack of originali...|
|A fool with a hea...|
|Grown-up people d...|
|I am a fool with ...|
|One can't underst...|
|In every idea of ...|
|It wasn't the New...|
|Sometimes you dre...|
|I almost do not e...|
|It's life that ma...|
|God knows what is...|
|One man doesn't b...|
|The prince says t...|
+--------------------+
```

Рис.7 Чтение данных из hdfs

Считаем количество часто встречающихся слов.

```
def clean_word(word):
    # Приводим слово к нижнему регистру
    word = word.lower()
    # Убираем все спецсимволы и цифры
    word = re.sub(r'[^a-zA-Za-яA-ЯёЁ]', '', word)  # Исправлено регулярное выражение
    # Убираем начальные и конечные пробелы
    word = word.strip()
    return word

# Очистка и фильтрация слов
text_rdd_cleaned = (text_rdd
                    .filter(lambda x: x is not None)
                    .map(lambda x: clean_word(x))  # Используем функцию очистки
                    .filter(lambda x: len(x) > 3)  # Убираем слова длиной 3 и менее
                    .filter(lambda x: len(x) > 0)  # Убираем пустые строки
                    .collect())

# Подсчет частоты слов
word_counts = (spark.sparkContext.parallelize(text_rdd_cleaned)  # Используем spark.sparkContext для создания RDD
               .map(lambda word: (word, 1))
               .reduceByKey(lambda x, y: x + y)
               .filter(lambda x: x[1] > 1))  # Оставляем только слова, встречающиеся более одного раза

# Преобразуем в DataFrame
word_counts_df = spark.createDataFrame(word_counts, ["word", "count"])

# Сортируем по убыванию частоты и выбираем топ-10
top_10_words = word_counts_df.orderBy("count", ascending=False).limit(10)

# Создаем красивую таблицу с помощью pandas
table_data = top_10_words.toPandas()
print("Топ-10 наиболее частых слов:")
print(table_data.to_string(index=False))

# Визуализация
plt.figure(figsize=(12, 6))
sns.barplot(
    data=table_data,
    x='count',
    y='word',
    hue='word',  # Явно указываем hue для устранения предупреждения
    palette='viridis',
    legend=False  # Отключаем легенду
)
plt.title('Топ-10 наиболее часто встречающихся слов', pad=20)
plt.xlabel('Частота')
```

Рис.8 подсчет часто встречающихся слов

Визуализируем подсчитанные слова.

```
plt.show()
```

```
Топ-10 наиболее частых слов:
      word  count
      that     24
      your     15
      with     12
     every      7
 something      7
     dream      7
      them      6
      were      6
     heart      6
      know      5
```

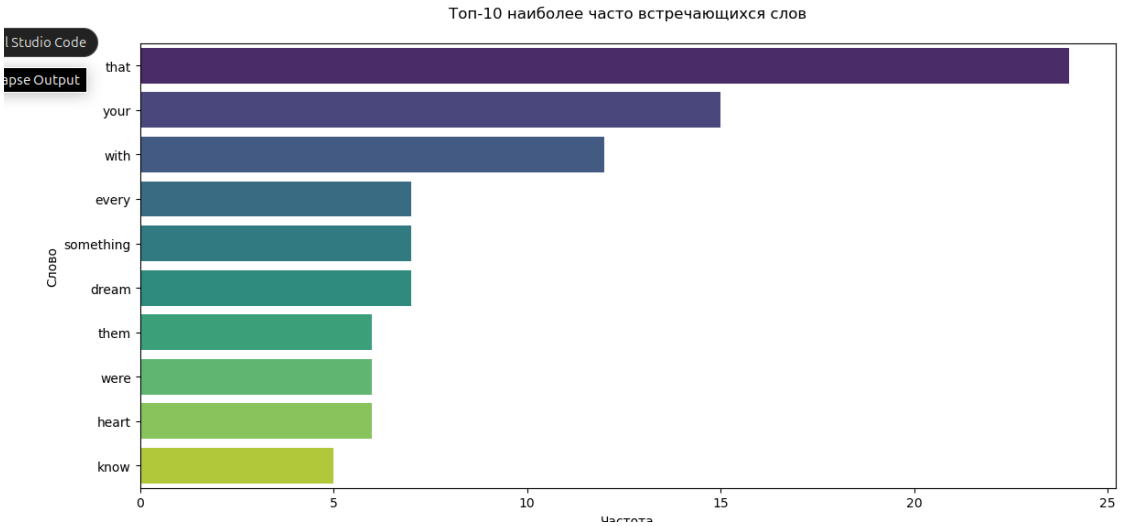


Рис.9 визуализация часто встречающихся слов

Запускаем sparksql и читаем данные из hdfs

```
[5]: from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("SQL App") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Установка количества разделов для shuffle операций
spark.conf.set("spark.sql.shuffle.partitions", "50")

# Чтение данных из HDFS (текстовый файл)
file_path = "hdfs://localhost:9000/user7/sparkdir/data/sales.csv"
header_path = "hdfs://localhost:9000/user7/sparkdir/data/sales_header.csv"
```

```
25/04/21 00:53:59 WARN Utils: Your hostname, devopsvm resolves to a loopback address: 127.0.1.1; using 192.168.31.
165 instead (on interface enp0s3)
25/04/21 00:53:59 WARN Utils: Set SPARK_LOCAL_IP if you need to bind to another address
Setting default log level to "WARN".
To adjust logging level use sc.setLogLevel(newLevel). For SparkR, use setLogLevel(newLevel).
25/04/21 00:54:00 WARN NativeCodeLoader: Unable to load native-hadoop library for your platform... using builtin-j
ava classes where applicable
```

Рис. 10 Чтение данных из hdfs

Парсим строки и читаем данные с использованием RDD

```
[6]: def parse_row(line):
         # Парсим строку (разделение по запятой)
         return line.split(',')

     # Чтение данных с использованием RDD
     sales_rdd = spark.sparkContext.textFile(file_path).map(parse_row)

     # Чтение заголовка из HDFS
     sales_header_rdd = spark.sparkContext.textFile(header_path).take(1)  # Читаем первую строку заголовка
     sales_header = sales_header_rdd[0].split(',')  # Разделяем по запятой
```

Рис.11 Парсинг строк

Выводим полученные данные.

```
[6]: # Преобразуем RDD в DataFrame с заголовками
     sales_df = spark.createDataFrame(sales_rdd, sales_header)

     # Регистрация временной таблицы
     sales_df.createOrReplaceTempView('sales')

     # Выполнение SQL-запроса для получения всех данных
     sales_total_df = spark.sql("SELECT * FROM sales")

     # Печать первых 10 строк
     sales_total_df.show(10)
```

```
+----------------+--------+-----+------------+----------------+--------------------+------------+-------------+--------------+------------+----------+----------+
|Transaction_date| Product|Price|Payment_Type|            Name|                City|       State|      Country|Account_Created|  Last_Login|  Latitude| Longitude|
+----------------+--------+-----+------------+----------------+--------------------+------------+-------------+--------------+------------+----------+----------+
|     1/2/09 6:17|Product1| 1200|  Mastercard|         carolina|           Basildon|     England|United Kingdom|    1/2/09 6:00| 1/2/09 6:08|      51.5|-1.1166667|
|     1/2/09 4:53|Product1| 1200|        Visa|           Betina|Parkville        ...|          MO| United States|    1/2/09 4:42| 1/2/09 7:49|    39.195| -94.68194|
|    1/2/09 13:08|Product1| 1200|  Mastercard|Federica e Andrea|Astoria          ...|          OR| United States|   1/1/09 16:21|1/3/09 12:32|   46.18806|   -123.83|
|    1/3/09 14:44|Product1| 1200|        Visa|            Gouya|              Echuca|    Victoria|     Australia|  9/25/05 21:13|1/3/09 14:22|-36.1333333|    144.75|
|    1/4/09 12:56|Product2| 3600|        Visa|          Gerd W |Cahaba Heights   ...|          AL| United States| 11/15/08 15:47|1/4/09 12:45|   33.52056|  -86.8025|
|    1/4/09 13:19|Product1| 1200|        Visa|         LAURENCE|Mickleton        ...|          NJ| United States|  9/24/08 15:19|1/4/09 13:04|      39.79| -75.23806|
|    1/4/09 20:11|Product1| 1200|  Mastercard|            Fleur|Peoria           ...|          IL| United States|    1/3/09 9:38|1/4/09 19:45|   40.69361| -89.58889|
|    1/2/09 20:09|Product1| 1200|  Mastercard|            adam|Martin           ...|          TN| United States|   1/2/09 17:43|1/4/09 20:01|   36.34333| -88.85028|
|    1/4/09 13:17|Product1| 1200|  Mastercard|  Renee Elisabeth|           Tel Aviv|     Tel Aviv|       Israel|   1/4/09 13:03|1/4/09 22:10| 32.0666667|34.7666667|
|    1/4/09 14:11|Product1| 1200|        Visa|           Aidan|             Chatou|Ile-de-France|       France|    6/3/08 4:22| 1/5/09 1:17| 48.8833333|      2.15|
+----------------+--------+-----+------------+----------------+--------------------+------------+-------------+--------------+------------+----------+----------+
only showing top 10 rows
```

Рис.12 Вывод данных.

Делаем визуализацию распределения цен по странам

```
[7]: # Визуализация распределения цен по странам
     # Преобразуем DataFrame в pandas для визуализации
     sales_total_df_pd = sales_total_df.toPandas()

     # Построение графика для анализа цен по странам
     plt.figure(figsize=(12, 6))
     sns.boxplot(
         data=sales_total_df_pd,
         x='Country',  # Страна
         y='Price',    # Цена
         palette='viridis'
     )

     plt.title('Распределение цен по странам')
     plt.xlabel('Страна')
     plt.ylabel('Цена')
     plt.xticks(rotation=90)  # Поворачиваем подписи на оси X для лучшей читаемости
     plt.tight_layout()
     plt.show()
```

```
/tmp/ipykernel_6500/1618239209.py:7: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in v0.14.0. Assign the `x` variable to `hue` and set `legend=False` for the same effect.

  boxplot(
```

Collapse Output

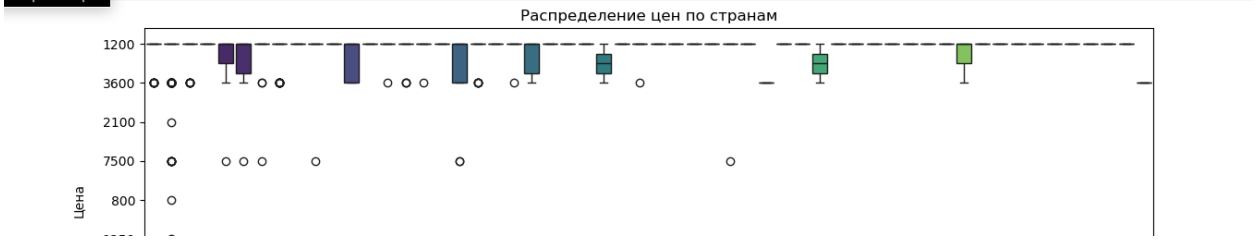


Рис. 13 визуализацию распределения цен по странам

Выводим доход по странам и штатам.

```
[8]: # TODO: Вывод дохода по стране и штату
revenue_by_country_state = spark.sql("""
    SELECT Country, State, SUM(Price) as Total_Revenue
    FROM sales
    GROUP BY Country, State
    ORDER BY Total_Revenue DESC
""")

# Отображаем результаты
revenue_by_country_state.show()
```

```
+--------------+----------------+-------------+
|       Country|           State|Total_Revenue|
+--------------+----------------+-------------+
|United Kingdom|         England|     120000.0|
| United States|              CA|     113350.0|
| United States|              NY|      61200.0|
| United States|              TX|      55500.0|
| United States|              FL|      51600.0|
|        Canada|         Ontario|      46800.0|
| United States|              VA|      40400.0|
|        Canada|British Columbia|      28800.0|
|       Ireland|          Dublin|      28800.0|
| United States|              GA|      28200.0|
|        Canada|         Alberta|      26400.0|
| United States|              WA|      24000.0|
| United States|              IL|      24000.0|
|   Netherlands|    Zuid-Holland|      23100.0|
| United States|              NJ|      22800.0|
| United States|              MD|      22800.0|
|     Australia| New South Wales|      20400.0|
| United States|              PA|      20400.0|
| United States|              TN|      19500.0|
| United States|              MN|      19200.0|
+--------------+----------------+-------------+
only showing top 20 rows
```

Рис.14 Запрос для вывода доход по странам и штатам.

Открываем файл sparklogit_2.ipynb

Загружаем библиотеки

```
[3] from pyspark.ml import Pipeline
    from pyspark.ml.classification import LogisticRegression
    from pyspark.ml.feature import HashingTF, Tokenizer
    from pyspark.sql.functions import UserDefinedFunction
    from pyspark.sql.types import *
```

ДАТАСЕТ С ДОМАМИ НА ПРОДАЖУ

В качестве примера мы будем использовать датасет Kaggle, который содержит данные о домах на продажу в Бруклине с 2003 по 2017 года и доступен для скачивания. Он содержит 111 атрибутов (столбцов) и 390883 записей (строк). В атрибуты включены: дата продажи, дата постройки, цена на дом, налоговый класс, соседние регионы, долгота, ширина и др.

Рис.15 импорт библиотек

Монтируем диск

```
[4]  from google.colab import drive
     drive.mount('/content/drive')

     Mounted at /content/drive
```

Рис.16 монтирурем диск

Указываем путь к данным

```
[6]  os.chdir("/content/drive/MyDrive/Colab Notebooks/data")
     os.listdir()

     ['stopwords.txt',
      'sales_header.csv',
      'sales.csv',
      'ratings.csv',
      'logit_train.csv',
      'kmeans.csv',
      'logit_test.csv',
      'reviews_sample.json',
      'ip.tsv',
      'idiot.txt',
      'digits.csv',
      'access.log',
      'food-inspections.csv',
      'brooklyn_sales_map.csv']
```

Рис.17 указываем путь к данным

Импортируем данные из hdfs.

```
import findspark
findspark.init()
from pyspark.sql import SparkSession
spark = SparkSession.builder.master("local[*]").getOrCreate()
data = spark.read.csv(
    '/content/drive/MyDrive/Colab Notebooks/data/brooklyn_sales_map.csv',
    inferSchema=True, header=True)
```

Рис. 18 импорт данных

## Делаем бинарную классификацию

те, которые принадлежат классу 1, и остальные. В Python это делается очень просто, нужно просто вызвать метод replace:

```
[9] by_1 = ['1', '1A', '1B', '1C']
    by_others = ['2', '2A', '2B', '2C', '3', '4']
    data = data.replace(by_others, '0', ['tax_class'])
    data = data.replace(by_1, '1', ['tax_class'])
```

Кроме того, алгоритмы Machine Learning в PySpark работают с числовым значениями, а не со строками. Поэтому преобразуем значения столбца tax_class в тип int:

```
data = data.withColumn('tax_class', data.tax_class.cast('int'))
```

Рис.19 бинарная классификация

## Делаем подбор признаков

```
[11] from pyspark.ml.feature import StringIndexer

     indexer = StringIndexer(inputCol="neighborhood", outputCol="neighborhood_id")
     data = indexer.fit(data).transform(data)
```

Рис.20 подбор признаков

Преобразованные категории имеют вид:

```
data.groupBy('neighborhood_id').count().show()
```

```
+---------------+-----+
|neighborhood_id|count|
+---------------+-----+
|            8.0|13215|
|            0.0|27279|
|            7.0|13387|
|           49.0| 2271|
|           29.0| 5074|
|           47.0| 2422|
|           42.0| 3086|
|           44.0| 2802|
|           35.0| 4000|
|           18.0| 7342|
|            1.0|21206|
|           39.0| 3396|
|           37.0| 3894|
|           34.0| 4037|
|           25.0| 5809|
|           36.0| 3984|
|           41.0| 3138|
|            4.0|14608|
|           23.0| 6374|
|           56.0|  985|
+---------------+-----+
only showing top 20 rows
```

Рис.21 Группируем neighborhood_id

Теперь выберем необходимые признаки, а также отбросим строки с пустыми значениями с помощью метода dropna в PySpark:

```
features = ['year_of_sale', 'sale_price', 'neighborhood_id']
target = 'tax_class'
attributes = features + [target]
sample = data.select(attributes).dropna()
```

Рис.22 выбор признаков

Делаем векторизация признаков

Для начала выберем в качестве признака для преобразования — цену на дом. Код на Pyt

```
[14] from pyspark.ml.feature import VectorAssembler

     assembler = VectorAssembler(inputCols=['sale_price'],
                                 outputCol='features')
     output = assembler.transform(sample)
```

Полученный после векторизации DataFrame выглядит следующим образом:

```
output.show(5)
```

```
+------------+------------+--------------+---------+--------------+
|year_of_sale|  sale_price|neighborhood_id|tax_class|      features|
+------------+------------+--------------+---------+--------------+
|        2008|4.99401179E8|          48.0|        0|[4.99401179E8]|
|        2016|      3.45E8|          41.0|        0|      [3.45E8]|
|        2016|       3.4E8|          27.0|        0|       [3.4E8]|
|        2012|   2.76947E8|          52.0|        0|   [2.76947E8]|
|        2017|     2.025E8|          27.0|        0|     [2.025E8]|
+------------+------------+--------------+---------+--------------+
only showing top 5 rows
```

Рис.23 векторизация признаков

Разделяем датасет на обучающиеся и тренировочные данные

Python это выглядит так:

```
[ ] train, test = output.randomSplit([0.8, 0.2])
```

Теперь воспользуемся логистической регрессией (Logistic Regression) [1], ко

Рис.24 разделение данных

Обучаем модель используя логистическую регрессию

```
[18] from pyspark.ml.classification import LogisticRegression

     lr = LogisticRegression(featuresCol='features',
                             labelCol='tax_class')
     model = lr.fit(train)
```

Осталось только получить предсказания. Для этого вызывается метод transform, который принимает тестовую выборку:

```
[19] predictions = model.transform(test)
```

Проверим эффективность модели, используя метрику качества. И в этом случае PySpark нас выручает, поскольку у него есть класс BinaryClassificationEvaluator. Нужно лишь указать целевой признак (tax class), а затем вызвать метод evaluate и передать в него наши предсказания. В Python это выглядит так:

Рис.25 логистическая регрессия

Проверяем эффективность модели

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol='tax_class')
print('Evaluation:', evaluator.evaluate(predictions))
```

```
Evaluation: 0.5244628900395347
```

Как видим, мы получили точность только 52%, что очень мало. Попробуем добавить ещё неск

Рис.26 вывод точности модели

Добавляем признаки

Векторизуем год постройки и соседние регионы.

```
[21] features = ['year_of_sale', 'sale_price', 'neighborhood_id']

     assembler = VectorAssembler(inputCols=features,
                                 outputCol='features')
     output = assembler.transform(sample)
```

```
[22] output.show(5)
```

```
+------------+------------+--------------+---------+--------------------+
|year_of_sale|  sale_price|neighborhood_id|tax_class|            features|
+------------+------------+--------------+---------+--------------------+
|        2008|4.99401179E8|          48.0|        0|[2008.0,4.9940117...|
|        2016|      3.45E8|          41.0|        0|[2016.0,3.45E8,41.0]|
|        2016|       3.4E8|          27.0|        0| [2016.0,3.4E8,27.0]|
|        2012|   2.76947E8|          52.0|        0|[2012.0,2.76947E8...|
|        2017|     2.025E8|          27.0|        0|[2017.0,2.025E8,2...|
+------------+------------+--------------+---------+--------------------+
only showing top 5 rows
```

Рис.27 Добавление признакров

Разделяем данные и обучаем

```
[23] train, test = output.randomSplit([0.8, 0.2])
```

```
from pyspark.ml.classification import LogisticRegression

lr = LogisticRegression(featuresCol='features',
                        labelCol='tax_class')
model = lr.fit(train)
```

Рис.28 разделение данных

Смотрим предсказание модели

```
[26] predictions = model.transform(test)
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

evaluator = BinaryClassificationEvaluator(labelCol='tax_class')
print('Evaluation:', evaluator.evaluate(predictions))
```

```
Evaluation: 0.6031776747259999
```

Python-код для остальных шагов – разделение на тестовую и обучающую выборки, обучение тем же. В итоге, мы смогли повысить точность до 60%:

Рис.29 вывод предсказания модели

Индивидуальное задание вариант 12

**Анализ отзывов: загрузить reviews.csv в HDFS, классифицировать отзывы**

Загружаем данные в hdfs и проверяем их загрузку

```
hadoop@devopsvm:~$ hdfs dfs -put /home/hadoop/Review.csv /user7/sparkdir/
```

Show 25 entries Search:

| Permission | Owner | Group | Size | Last Modified | Replication | Block Size | Name |
|---|---|---|---|---|---|---|---|
| -rw-r--r-- | hadoop | supergroup | 2.84 MB | Apr 21 02:15 | 1 | 128 MB | Review.csv |
| drwxr-xr-x | hadoop | supergroup | 0 B | Apr 21 00:25 | 0 | 0 B | data |

Рис.30 загрузка данных

Импортируем библиотеки

```
[7]: # import libraries
import pyspark
from pyspark import SparkConf
from pyspark.sql import SparkSession

from pyspark.sql.window import Window
from pyspark.sql.functions import udf, isnan, min, max, sum, count, desc, expr, avg
from pyspark.sql.types import IntegerType, LongType
from pyspark.sql import SparkSession
from pyspark.ml.feature import StandardScaler, VectorAssembler, MinMaxScaler
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.tuning import CrossValidator, ParamGridBuilder, CrossValidatorModel
from pyspark.ml import Pipeline
from pyspark.ml.classification import DecisionTreeClassifier, RandomForestClassifier, LogisticRegression, GBTClassifier, LogisticRegressionModel, GBTClassificationModel, Ran
Editor
import matplotlib.pyplot as plt
import seaborn as sns
import datetime
import numpy as np
import pandas as pd
import re  # Добавляем импорт модуля re
from tabulate import tabulate
import matplotlib.pyplot as plt
```

Рис.31 импорт библиотек

Создаем сессию и читаем данные из hdfs

```
8]: from pyspark.sql import SparkSession

# Создание SparkSession
spark = SparkSession.builder \
    .appName("Review") \
    .config("spark.hadoop.fs.defaultFS", "hdfs://localhost:9000") \
    .config("spark.ui.port", "4050") \
    .getOrCreate()

# Чтение данных из HDFS (текстовый файл
file_path = "hdfs://localhost:9000/user7/sparkdir/Review.csv"

df = spark.read.text(file_path)  # Используем .text, так как это текстовый файл

# Печать первых нескольких строк
df.show()
```

```
+--------------------+
|               value|
+--------------------+
|        review,class|
|"Fantastic spot f...|
|                  ",1|
|"Love, love, love...|
|                  ",1|
|"Love this place....|
|                  ",1|
|"It's everything ...|
|                  ",1|
|"I came here befo...|
|                  ",1|
|"Olive or Twist i...|
|                  ",1|
|"A beautiful litt...|
|                  ",1|
|"My favorite bar ...|
|                  ",1|
|"The location is ...|
|                  ",1|
|"THIS PLACE IS OP...|
+--------------------+
only showing top 20 rows
```

Рис.32 вывод полученных данных

**SQL-анализ: определить средний рейтинг по категориям**

Нашел датасет на Kaggle “Amazon Sales Dataset”

https://www.kaggle.com/datasets/karkavelrajaj/amazon-sales-dataset

Набор данных содержит данные о рейтингах и обзорах более 1000 продуктов Amazon в соответствии с их данными, указанными на официальном сайте Amazon.

Загружаем данные в colab

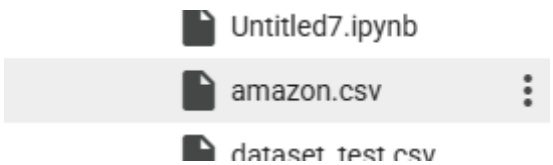


Рис.33 загрузка данных в colab

Загружаем и импортируем библиотеки.

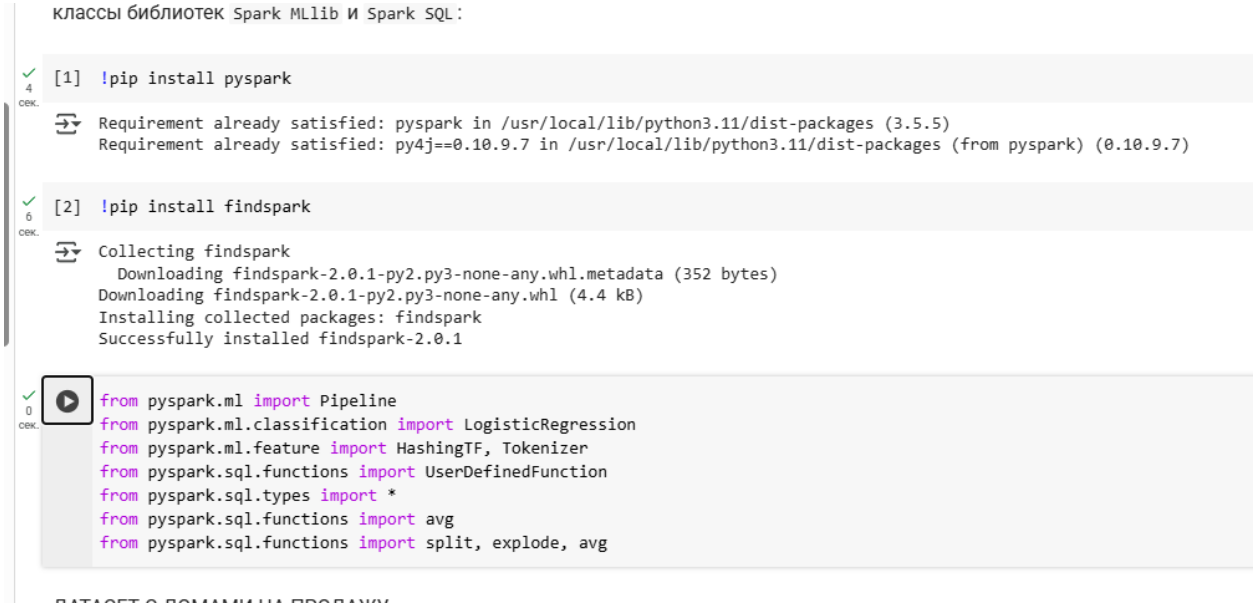


Рис.34 загрузка и импорт библиотек

Монтируем драйв, для загрузки данных.

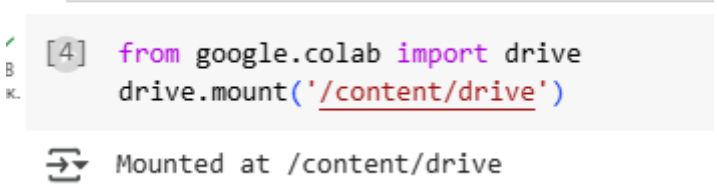


Рис.35 монтируем google drive

Инициализирует SparkSession, запускаем spark локально и загружаем данные из CSV-файла.

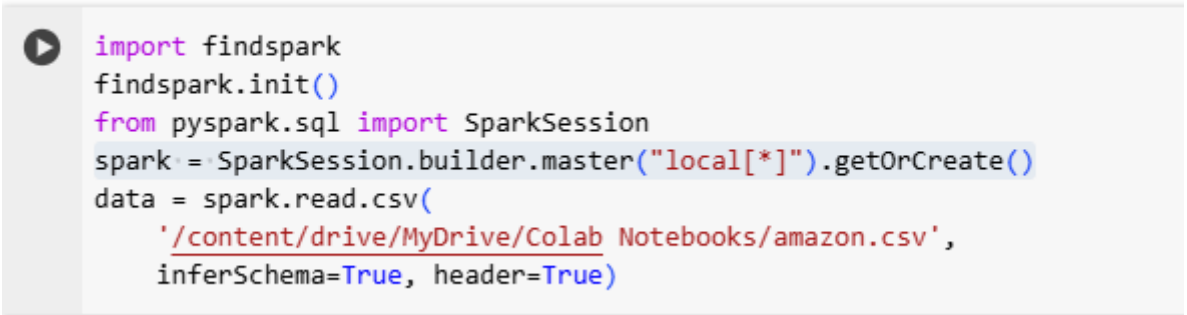

Рис.36 загрузка данных

Проверяем загруженные данные

```
data.printSchema()
data.show(5)
```

```
root
 |-- product_id: string (nullable = true)
 |-- product_name: string (nullable = true)
 |-- category: string (nullable = true)
 |-- discounted_price: string (nullable = true)
 |-- actual_price: string (nullable = true)
 |-- discount_percentage: string (nullable = true)
 |-- rating: string (nullable = true)
 |-- rating_count: string (nullable = true)
 |-- about_product: string (nullable = true)
 |-- user_id: string (nullable = true)
 |-- user_name: string (nullable = true)
 |-- review_id: string (nullable = true)
 |-- review_title: string (nullable = true)
 |-- review_content: string (nullable = true)
 |-- img_link: string (nullable = true)
 |-- product_link: string (nullable = true)

+----------+--------------------+--------------------+----------------+------------+-------------------+------+------------+--------------------+--------------------+--------------------+
|product_id|        product_name|            category|discounted_price|actual_price|discount_percentage|rating|rating_count|       about_product|             user_id|           user_name|
+----------+--------------------+--------------------+----------------+------------+-------------------+------+------------+--------------------+--------------------+--------------------+
|B07JW9H4J1|Wayona Nylon Brai...|Computers&Accesso...|            ₹399|      ₹1,099|                64%|   4.2|      24,269| High Compatibilit...|AG3D6O4STAQKAY2UV...|Manav,Adarsh gupt...|
|B098NS6PVG|Ambrane Unbreakab...|Computers&Accesso...|            ₹199|        ₹349|                43%|   4.0|      43,994| Compatible with a...|AECPFYFQVRUWC3KGN...|ArdKn,Nirbhay kum...|
|B096MSW6CT|Sounce Fast Phone...|Computers&Accesso...|            ₹199|      ₹1,899|                90%|   3.9|       7,928| 【 Fast Charger& D...|AGU3BBQ2V2DDAMOAK...|Kunal,Himanshu,vi...|
|B08HDJ86NZ|boAt Deuce USB 30...|Computers&Accesso...|            ₹329|        ₹699|                53%|   4.2|      94,363| The boAt Deuce US...|AEWAZDZZJLQUYVOVG...|Omkar dhale,JD,HE...|
|B08CF3B7N1|Portronics Konnec...|Computers&Accesso...|            ₹154|        ₹399|                61%|   4.2|      16,905| [CHARGE & SYNC FU...|AE3Q6KSUK5P75D5HF...|rahuls6099,Swasat...|
+----------+--------------------+--------------------+----------------+------------+-------------------+------+------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

Рис.37 Проверка загруженных данных

Так как у товары могут относиться к нескольким категориям, создаем отдельные строки для каждой категории.

Регистрируем временную таблицу.

Далее, группируем данные по категориям и считаем средний рейтинг.

```
#  разделяем категории на отдельные строки и очищаем их от лишних пробелов
data = data.withColumn("Category", explode(split(trim(col("category")), "\|")))

#  регистрация временной таблицы
data.createOrReplaceTempView("amazon_data")

query = """
WITH exploded_categories AS (
    SELECT
        TRIM(Category) AS Category,
        rating
    FROM amazon_data
)
SELECT
    Category,
    ROUND(AVG(rating), 2) AS Average_Rating
FROM exploded_categories
GROUP BY Category
ORDER BY Average_Rating DESC
"""


# 6. Показываем результаты
category_avg_rating_sql.show()
```

```
+--------------------+--------------+
|            Category|Average_Rating|
+--------------------+--------------+
|             Tablets|           4.6|
|          Projectors|           4.5|
|              Memory|           4.5|
|MediaStreamingDev...|           4.5|
|     SurgeProtectors|           4.5|
|                Film|           4.5|
|    PowerAccessories|           4.5|
|    PowerLANAdapters|           4.5|
|               Basic|           4.5|
|      CordManagement|           4.5|
|    StreamingClients|           4.5|
|       CoffeePresses|           4.5|
|Maintenance,Upkee...|          4.47|
|           AirFryers|          4.46|
|       DeepFatFryers|          4.46|
|   OfficeElectronics|          4.45|
```

Рис.38 считаем средний рейтинг

### 3.Визуализировать распределение оценок

Визуализируем данные из задания 2 “Amazon sales”

Данные в датасете записаны как string

```
data.dtypes
```

```
[('product_id', 'string'),
 ('product_name', 'string'),
 ('category', 'string'),
 ('discounted_price', 'string'),
 ('actual_price', 'string'),
 ('discount_percentage', 'string'),
 ('rating', 'string'),
 ('rating_count', 'string'),
 ('about_product', 'string'),
 ('user_id', 'string'),
 ('user_name', 'string'),
 ('review_id', 'string'),
 ('review_title', 'string'),
 ('review_content', 'string'),
 ('img_link', 'string'),
 ('product_link', 'string')]
```

Рис.39 Тип данных в датасете

Преобразуем rating из string в float для использования его в визуализации

```
# Преобразование столбца "rating" в числовой формат
data = data.withColumn("rating", data["rating"].cast("float"))
```

Рис.40 Преобразования столбца rating

Строим гистограмму для распределения рейтинга

```python
import matplotlib.pyplot as plt

#берем столбец rating
ratings = data.select("rating")

# преобразуем данные из Spark в Pandas
ratings_pd = ratings.toPandas()

plt.figure(figsize=(10, 6))
plt.hist(ratings_pd["rating"], bins=20, color="blue",edgecolor="black")
plt.title("Распределение оценок (рейтингов)")
plt.xlabel("Рейтинг")
plt.ylabel("Количество продуктов")
plt.grid(axis="y")
plt.show()
```

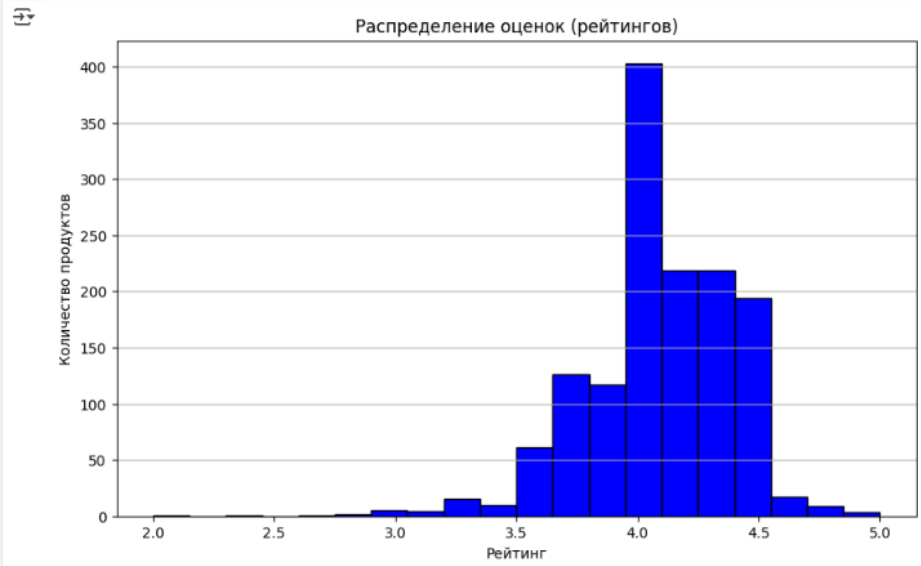


Рис.41гистограмма распределения рейтингов

Строим график плотности для распределения рейтингов

```python
# Построение графика плотности
plt.figure(figsize=(10, 6))
ratings_pd["rating"].plot(kind="density", color="red")
plt.title("Плотность распределения оценок (рейтингов)")
plt.xlabel("Рейтинг")
plt.ylabel("Плотность")
plt.grid(axis="y")
plt.show()
```

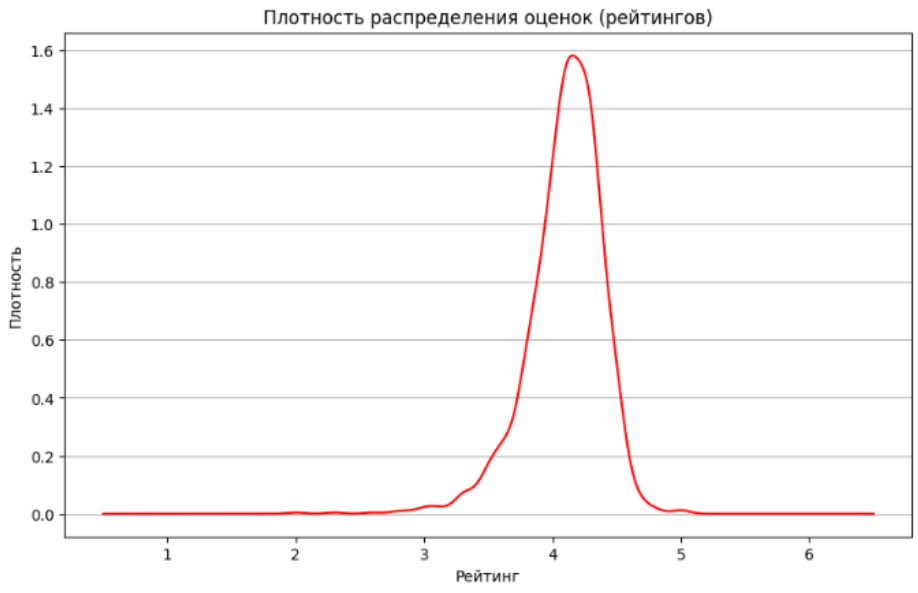


Рис.42 график плотности

## Заключение

В ходе проделанной лабораторной работы, были получены практические навыки работы с Apache Spark и его интеграцией с Python через библиотеку PySpark, обработка больших объемов данных, используя распределенные вычисления, а также применение базовых операции с RDD и DataFrame, работа с SQL-запросами в Spark SQL, а также визуализация результатов обработки данных.