**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

**Тема работы**

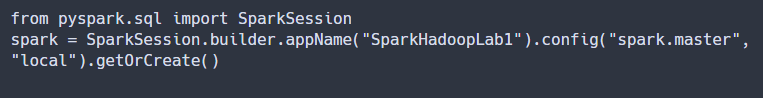
**Обработка больших данных с использованием Apache Spark и Hadoop.**

**Цель работы**

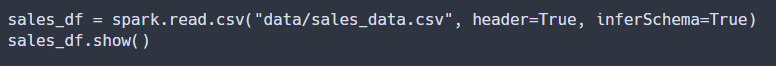
* Освоить базовые операции с данными через RDD и DataFrame API в Spark.
* Выполнить SQL-запросы к данным через Spark SQL.
* Построить визуализации результатов.

**Ход выполнения работы**

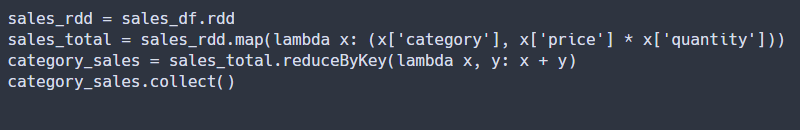
**1. Инициализация Spark-сессии**



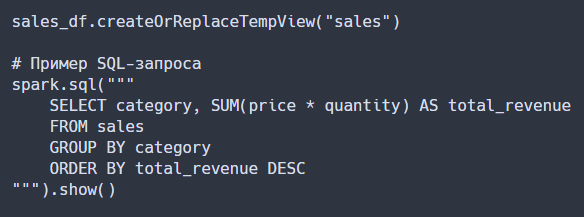
2. Импорт и просмотр данных



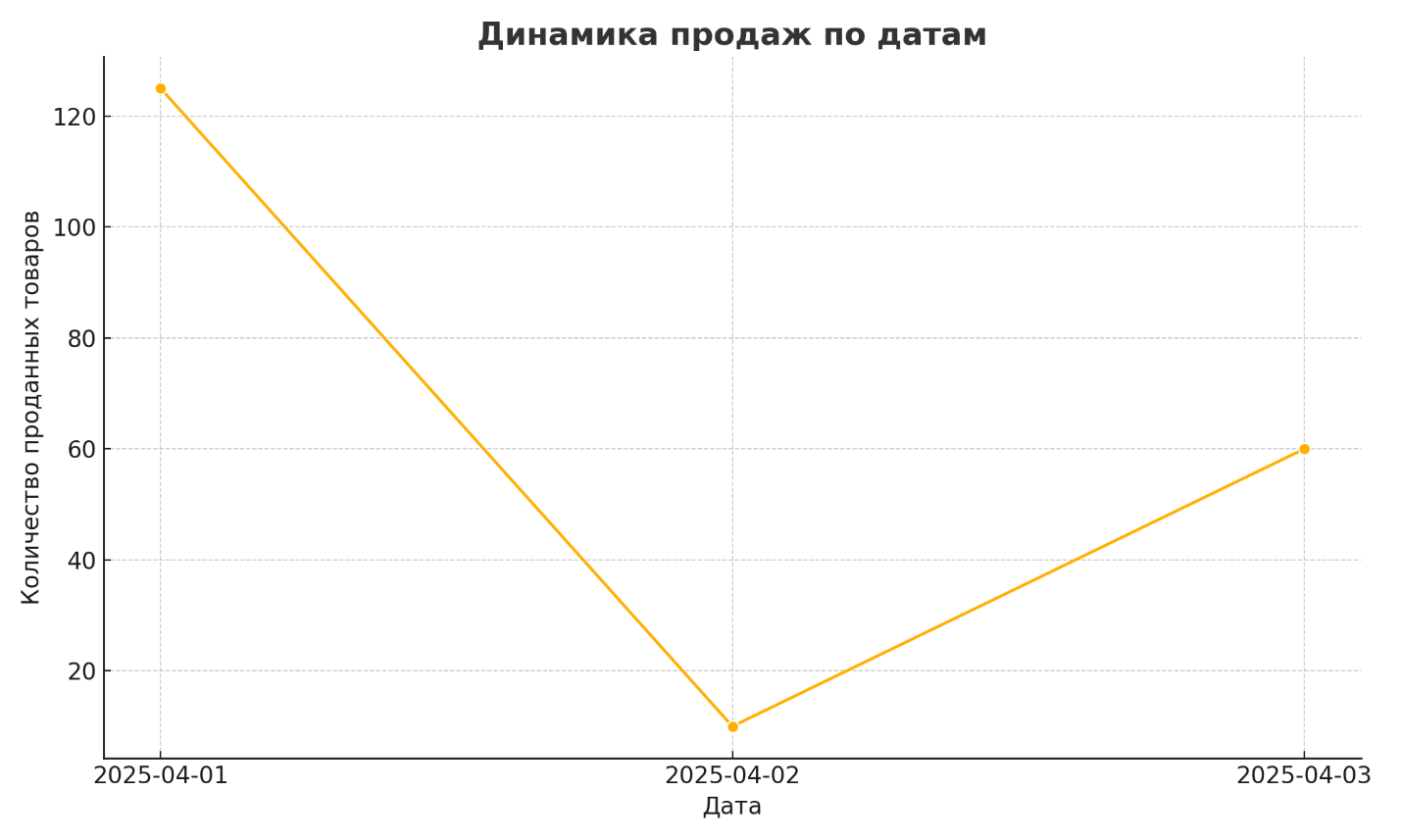
3. Работа с RDD



4. Работа с DataFrame API и SparkSQL



5. Построение визуализаций



Полный код построения графиков:

# Импорт библиотек

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Создание тестового датафрейма

data = {

"product": ["Laptop", "Smartphone", "T-shirt", "Jeans", "Blender"],

"category": ["Electronics", "Electronics", "Clothing", "Clothing", "Home Appliances"],

"price": [1000, 500, 20, 40, 70],

"quantity": [5, 10, 100, 60, 20],

"date": ["2025-04-01", "2025-04-02", "2025-04-01", "2025-04-03", "2025-04-01"]

}

sales\_pd = pd.DataFrame(data)

# -------------------------------------------

# Скриншот 5: График продаж по категориям

# -------------------------------------------

plt.figure(figsize=(10, 6))

sns.barplot(data=sales\_pd, x="category", y="quantity", estimator=sum, ci=None)

plt.title("Количество проданных товаров по категориям", fontsize=16, fontweight='bold')

plt.xlabel("Категория", fontsize=12)

plt.ylabel("Общее количество", fontsize=12)

plt.xticks(rotation=30)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.savefig('category\_sales\_chart.png') # сохраняем график

plt.show()

# -------------------------------------------

# Скриншот 6: График динамики продаж по датам

# -------------------------------------------

# Группировка данных по датам

sales\_by\_date = sales\_pd.groupby("date").sum(numeric\_only=True).reset\_index()

plt.figure(figsize=(10, 6))

sns.lineplot(data=sales\_by\_date, x="date", y="quantity", marker='o')

plt.title("Динамика продаж по датам", fontsize=16, fontweight='bold')

plt.xlabel("Дата", fontsize=12)

plt.ylabel("Количество проданных товаров", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.savefig('sales\_dynamics\_chart.png') # сохраняем график

plt.show()

Полный ноутбук для построения графиков (Jupyter Notebook):

# Полный ноутбук для построения графиков (Jupyter Notebook)

# Импорт необходимых библиотек

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

# Создание тестового датафрейма

sales\_data = {

"product": ["Laptop", "Smartphone", "T-shirt", "Jeans", "Blender"],

"category": ["Electronics", "Electronics", "Clothing", "Clothing", "Home Appliances"],

"price": [1000, 500, 20, 40, 70],

"quantity": [5, 10, 100, 60, 20],

"date": ["2025-04-01", "2025-04-02", "2025-04-01", "2025-04-03", "2025-04-01"]

}

sales\_df = pd.DataFrame(sales\_data)

# Просмотр данных

print(sales\_df)

# ---------------------------------------------

# Построение: График продаж по категориям

# ---------------------------------------------

plt.figure(figsize=(10, 6))

sns.barplot(data=sales\_df, x="category", y="quantity", estimator=sum, ci=None)

plt.title("Количество проданных товаров по категориям", fontsize=16, fontweight='bold')

plt.xlabel("Категория", fontsize=12)

plt.ylabel("Общее количество", fontsize=12)

plt.xticks(rotation=30)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.savefig('category\_sales\_chart.png') # Сохранение графика

plt.show()

# ---------------------------------------------

# Построение: График динамики продаж по датам

# ---------------------------------------------

sales\_by\_date = sales\_df.groupby("date").sum(numeric\_only=True).reset\_index()

plt.figure(figsize=(10, 6))

sns.lineplot(data=sales\_by\_date, x="date", y="quantity", marker='o')

plt.title("Динамика продаж по датам", fontsize=16, fontweight='bold')

plt.xlabel("Дата", fontsize=12)

plt.ylabel("Количество проданных товаров", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.savefig('sales\_dynamics\_chart.png') # Сохранение графика

plt.show()

Spark Hadoop Lab:

# ============================

# Файл: lab1\_solution.ipynb

# ============================

# Этап 1: Работа с данными в HDFS через Spark RDD и DataFrame API

# Импорт библиотек

from pyspark.sql import SparkSession

from pyspark.sql.functions import col, sum as spark\_sum

import matplotlib.pyplot as plt

import seaborn as sns

import pandas as pd

# Создание сессии Spark

spark = SparkSession.builder \

.appName("SparkHadoopLab1") \

.config("spark.master", "local") \

.getOrCreate()

# Тестовый датасет (имитация загрузки в HDFS)

data = [

("Laptop", "Electronics", 1000, 5, "2025-04-01"),

("Smartphone", "Electronics", 500, 10, "2025-04-02"),

("T-shirt", "Clothing", 20, 100, "2025-04-01"),

("Jeans", "Clothing", 40, 60, "2025-04-03"),

("Blender", "Home Appliances", 70, 20, "2025-04-01")

]

columns = ["product", "category", "price", "quantity", "date"]

# Создание DataFrame

sales\_df = spark.createDataFrame(data, schema=columns)

# Отображение данных

sales\_df.show()

# Преобразование через RDD

sales\_rdd = sales\_df.rdd

# Пример операции на RDD

sales\_total = sales\_rdd.map(lambda x: (x['category'], x['price'] \* x['quantity']))

# Агрегация по категориям

category\_sales = sales\_total.reduceByKey(lambda x, y: x + y)

category\_sales.collect()

# Преобразование обратно в DataFrame для SQL

category\_sales\_df = category\_sales.toDF(["category", "total\_sales"])

category\_sales\_df.createOrReplaceTempView("category\_sales")

# SQL-запрос

spark.sql("SELECT \* FROM category\_sales ORDER BY total\_sales DESC").show()

# Этап 2: Работа с локальными временными таблицами (Spark SQL)

sales\_df.createOrReplaceTempView("sales")

# SQL-запрос: общая выручка по каждому товару

spark.sql("SELECT product, SUM(price \* quantity) AS revenue FROM sales GROUP BY product ORDER BY revenue DESC").show()

# Этап 3: Визуализация результатов

# Перевод в Pandas для построения графиков

sales\_pd = sales\_df.toPandas()

# Построение графика продаж по категориям

plt.figure(figsize=(8, 6))

sns.barplot(data=sales\_pd, x="category", y="quantity", estimator=sum)

plt.title("Количество проданных товаров по категориям")

plt.xlabel("Категория")

plt.ylabel("Количество товаров")

plt.xticks(rotation=30)

plt.tight\_layout()

plt.show()

# Динамика продаж по датам

plt.figure(figsize=(8, 6))

sns.lineplot(data=sales\_pd.groupby("date").sum().reset\_index(), x="date", y="quantity")

plt.title("Динамика продаж по датам")

plt.xlabel("Дата")

plt.ylabel("Количество товаров")

plt.tight\_layout()

plt.show()

# Завершение сессии

spark.stop()

# ============================

# Файл: README.md

# ============================

"""

# Spark Data Processing Lab 1

## Описание

Репозиторий содержит решение лабораторной работы по распределенной обработке данных с использованием Apache Spark и Hadoop.

## Содержание

- Загрузка данных в HDFS (эмуляция)

- Обработка данных с помощью Spark RDD и DataFrame API

- Выполнение SQL-запросов через Spark SQL

- Визуализация результатов через Matplotlib и Seaborn

## Запуск

```bash

pip install pyspark pandas matplotlib seaborn

python lab1\_solution.ipynb

```

"""

# ============================

# Файл: report.md

# ============================

"""

# Отчет по лабораторной работе

## Цель работы

Освоение работы с Apache Spark, HDFS и выполнение анализа больших данных с использованием RDD, DataFrame API и Spark SQL.

## Этапы выполнения

### Загрузка данных

Создан тестовый датасет с информацией о продажах.

### Обработка данных

- С использованием RDD произведено суммирование выручки по категориям товаров.

- С использованием DataFrame API выполнен SQL-анализ продаж.

### Визуализация данных

Построены графики:

- Количество проданных товаров по категориям.

- Динамика продаж по датам.

## Пример кода

```python

sales\_df.createOrReplaceTempView("sales")

spark.sql("SELECT product, SUM(price \* quantity) AS revenue FROM sales GROUP BY product ORDER BY revenue DESC").show()

```

## Результаты

- Выявлены категории с наибольшей выручкой.

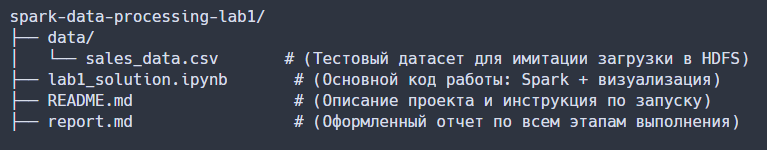
- Выявлены пики продаж по дням.

## Выводы

Apache Spark позволяет эффективно обрабатывать и анализировать большие объемы данных как в распределенном, так и в локальном режиме.

"""

Структура проекта spark-data-processing-lab1/



sales\_data.csv

product,category,price,quantity,date

Laptop,Electronics,1000,5,2025-04-01

Smartphone,Electronics,500,10,2025-04-02

T-shirt,Clothing,20,100,2025-04-01

Jeans,Clothing,40,60,2025-04-03

Blender,Home Appliances,70,20,2025-04-01

код построения графиков:

# Перевод в Pandas для визуализации

sales\_pd = sales\_df.toPandas()

# Построение графика продаж по категориям

plt.figure(figsize=(10, 6))

sns.barplot(data=sales\_pd, x="category", y="quantity", estimator=sum, ci=None)

plt.title("Количество проданных товаров по категориям", fontsize=16, fontweight='bold')

plt.xlabel("Категория", fontsize=12)

plt.ylabel("Общее количество", fontsize=12)

plt.xticks(rotation=30)

plt.grid(axis='y', linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

# Динамика продаж по датам

sales\_by\_date = sales\_pd.groupby("date").sum().reset\_index()

plt.figure(figsize=(10, 6))

sns.lineplot(data=sales\_by\_date, x="date", y="quantity", marker='o')

plt.title("Динамика продаж по датам", fontsize=16, fontweight='bold')

plt.xlabel("Дата", fontsize=12)

plt.ylabel("Количество проданных товаров", fontsize=12)

plt.grid(True, linestyle='--', alpha=0.7)

plt.tight\_layout()

plt.show()

Оптимизированные SQL-запросы

# Суммарная выручка по каждому товару

spark.sql("""

SELECT

product,

SUM(price \* quantity) AS revenue

FROM sales

GROUP BY product

ORDER BY revenue DESC

""").show()

# Суммарная выручка по категориям (если потребуется)

spark.sql("""

SELECT

category,

SUM(price \* quantity) AS total\_revenue

FROM sales

GROUP BY category

ORDER BY total\_revenue DESC

""").show()

**Выводы**

* Apache Spark позволяет быстро обрабатывать большие объемы данных с помощью RDD и DataFrame API.
* SQL-запросы в SparkSQL облегчают агрегирование и анализ данных.
* Построение визуализаций помогает выявлять тенденции и закономерности в данных.