

**МИНОБРНАУКИ РОССИИ**

Федеральное государственное бюджетное образовательное учреждение высшего образования

**«МИРЭА – Российский технологический университет»**

|  |
| --- |
| **РТУ МИРЭА** |
|  |
| **Институт кибербезопасности и цифровых технологий (ИКБ)** |
|  |
| КБ-2 «Информационно-аналитические системы кибербезопасности» |

**ОТЧЕТ О ВЫПОЛНЕНИИ ИНДИВИДУАЛЬНОГО ЗАДАНИЯ №3**

**В РАМКАХ ДИСЦИПЛИНЫ «АВТОМАТИЗИРРОВАННЫЕ ТЕХНОЛОГИИ ОБРАБОТКИ БОЛЬШИХ ИФНОРМАЦИОННЫХ ПОТОКОВ»**

Выполнил:

Студент 4-ого курса

Учебной группы БИСО-02-22

Зубарев В.С.

Выполнение практических заданий

1. Изучить методическое пособие по Apache Spark

Осуществить установку Apache Spark (https://spark.apache.org/downloads.html) в любом формате, например docker

Для выполнения заданий будет использоваться виртуальная машина Ubuntu 24.04

sudo apt update && sudo apt upgrade -y

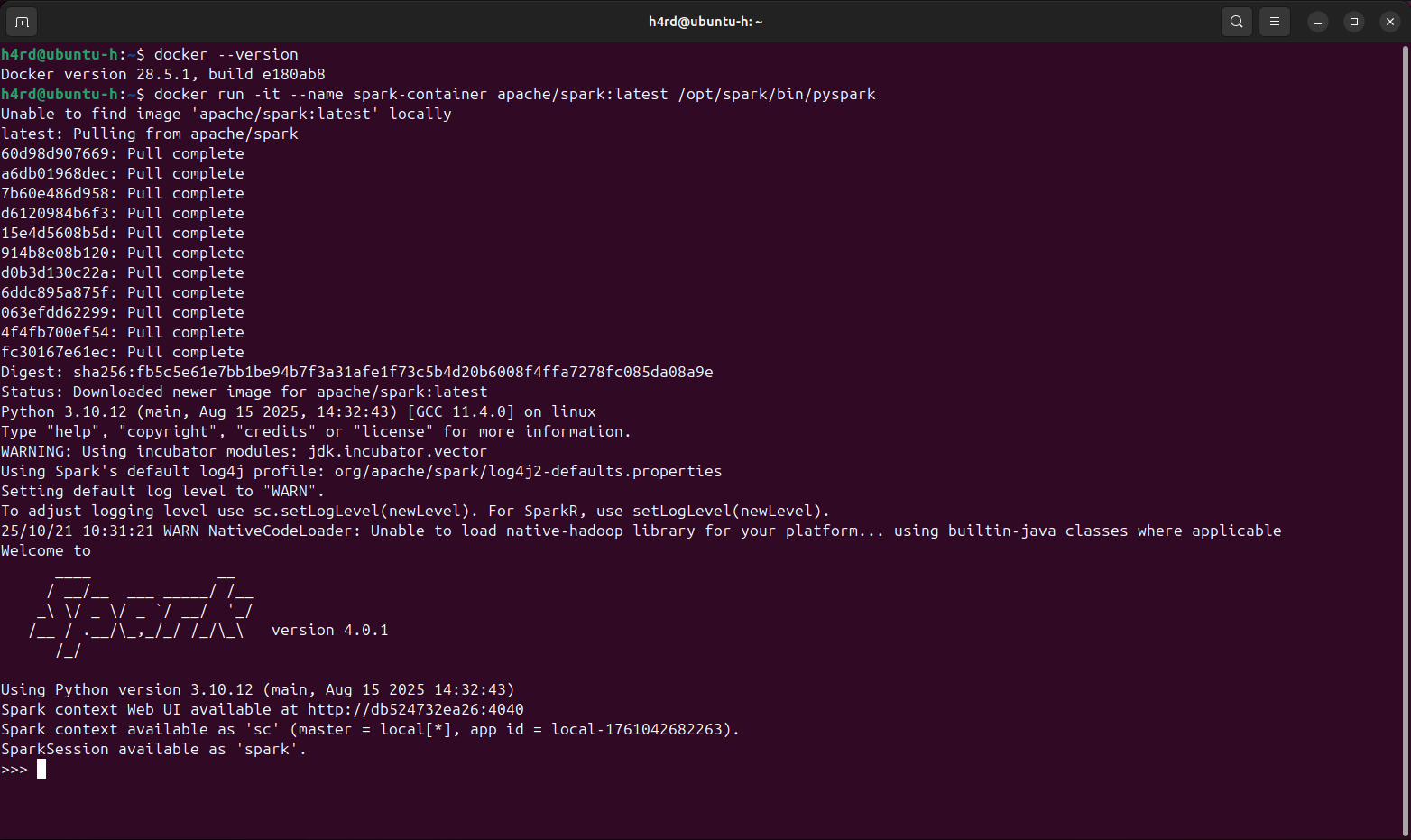
sudo apt install docker.io -y

sudo usermod -aG docker $h4rd

sudo systemctl start docker

sudo systemctl enable docker

docker run -it --name spark-container apache/spark:latest /opt/spark/bin/pyspark



2. Выполнить действия из раздела 4 пособия.

sudo apt install openjdk-17-jdk -y

sudo apt install maven -y

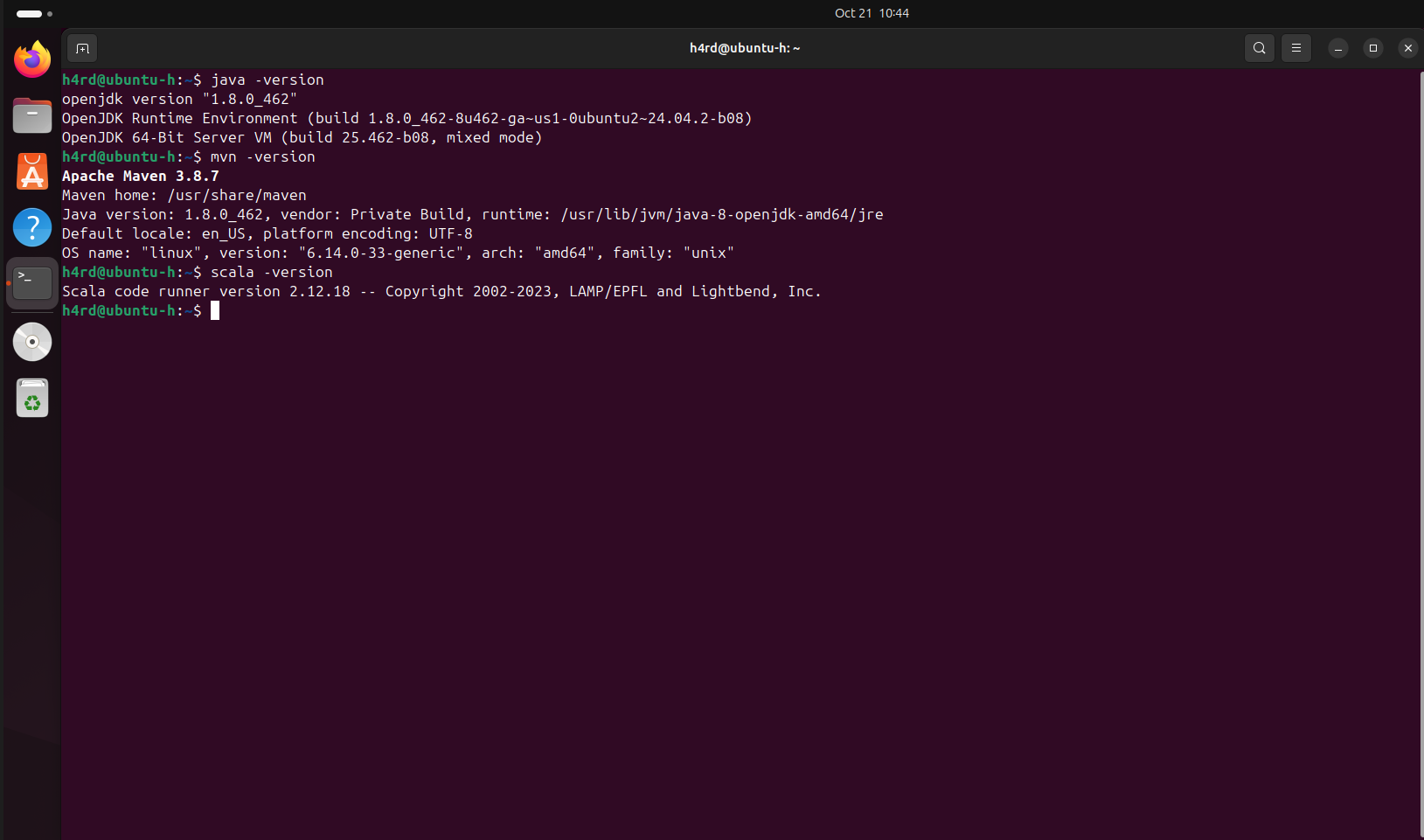
sudo snap install intellij-idea-community –classic

curl -s "https://get.sdkman.io" | bash

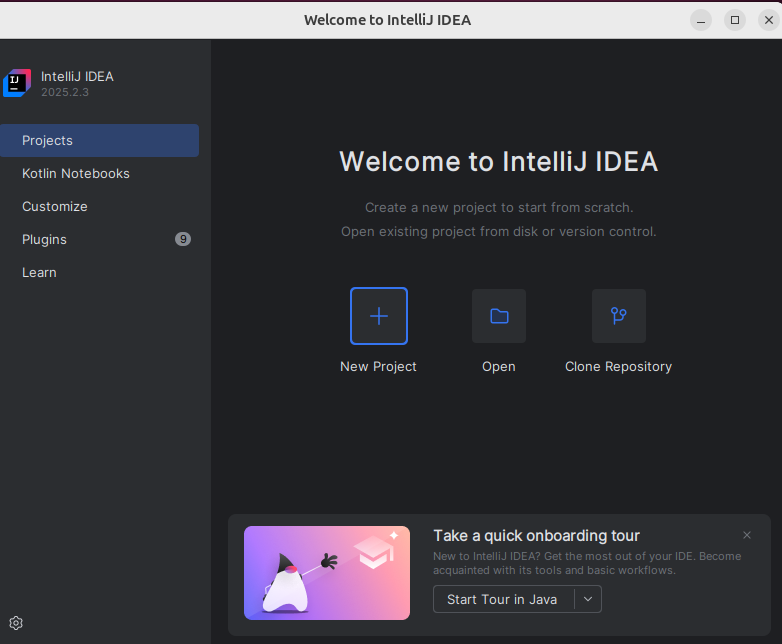
source "$HOME/.sdkman/bin/sdkman-init.sh"

sdk install scala 2.12.18

Проверка установленных пакетов



Создание проекта



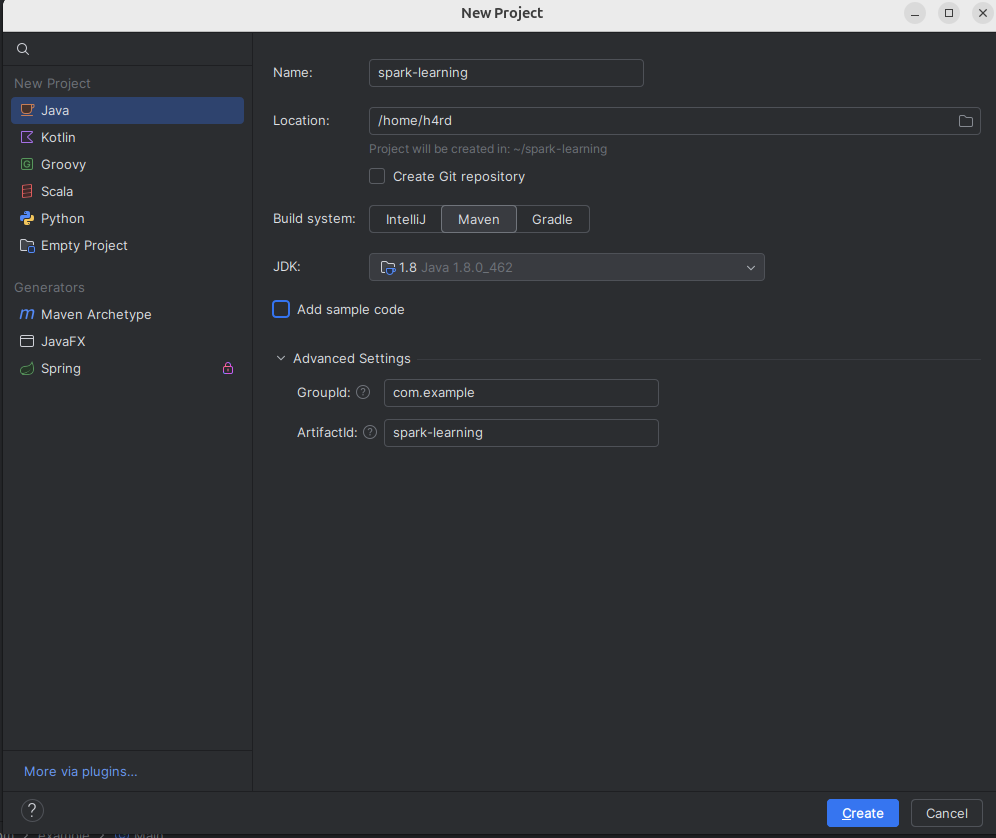
Настройка проекта

Name: spark-learning

Location: /home/your-user/spark-learning

GroupId: com.example

ArtifactId: spark-learning

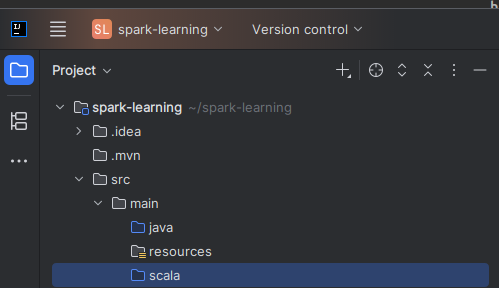


Настройка структуры проекта:

Правой кнопкой на src/main → New → Directory

Название: scala

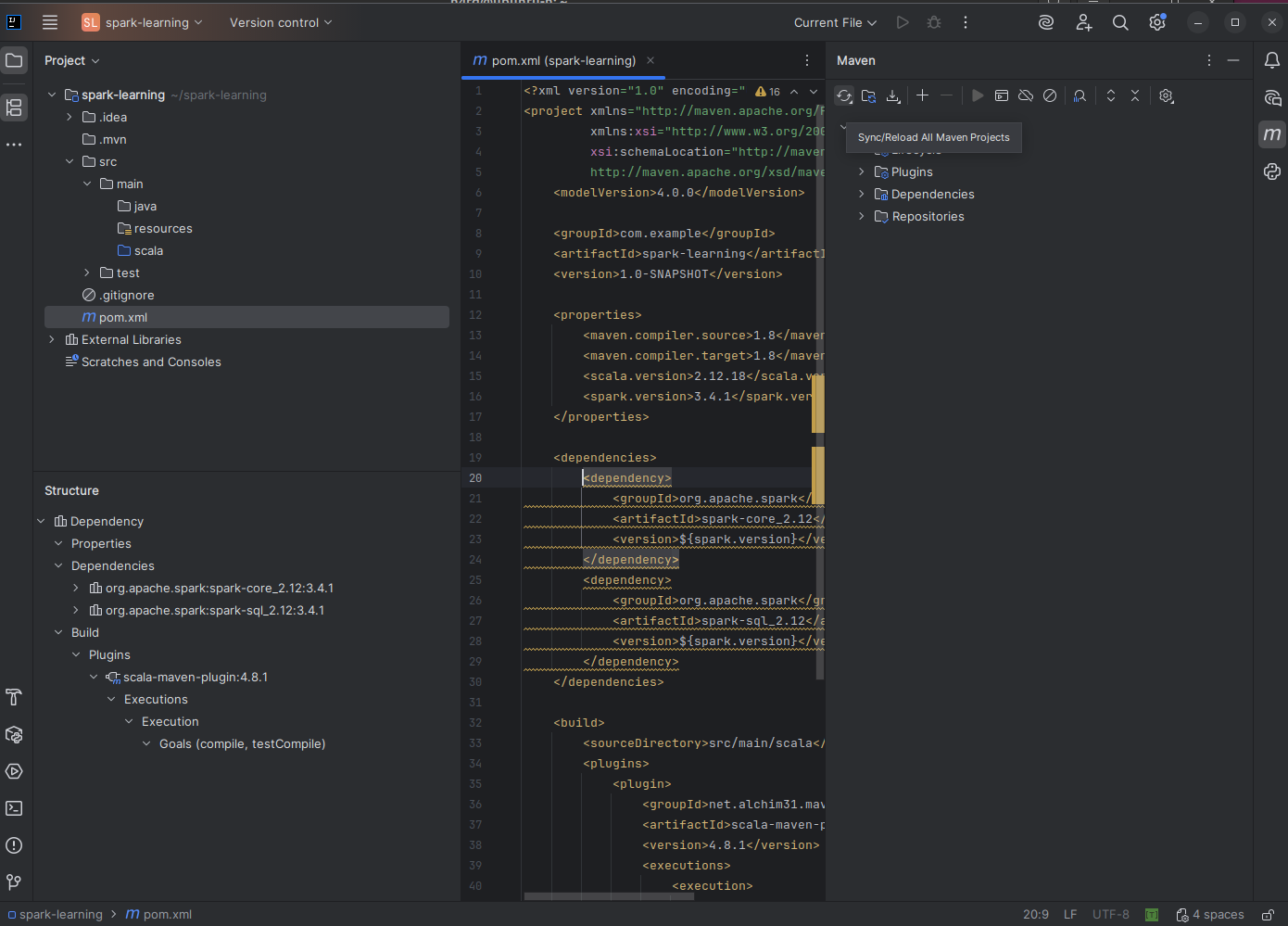
Правой кнопкой на папке scala → Mark Directory as → Sources Root



Добавление зависимостей в pom.xml (программный код). Файл был перезаписан.

<?xml version="1.0" encoding="UTF-8"?>  
<project xmlns="http://maven.apache.org/POM/4.0.0"  
 xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"  
 xsi:schemaLocation="http://maven.apache.org/POM/4.0.0   
 http://maven.apache.org/xsd/maven-4.0.0.xsd">  
 <modelVersion>4.0.0</modelVersion>  
  
 <groupId>com.example</groupId>  
 <artifactId>spark-learning</artifactId>  
 <version>1.0-SNAPSHOT</version>  
  
 <properties>  
 <maven.compiler.source>1.8</maven.compiler.source>  
 <maven.compiler.target>1.8</maven.compiler.target>  
 <scala.version>2.12.18</scala.version>  
 <spark.version>3.4.1</spark.version>  
 </properties>  
  
 <dependencies>  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-core\_2.12</artifactId>  
 <version>${spark.version}</version>  
 </dependency>  
 <dependency>  
 <groupId>org.apache.spark</groupId>  
 <artifactId>spark-sql\_2.12</artifactId>  
 <version>${spark.version}</version>  
 </dependency>  
 </dependencies>  
  
 <build>  
 <sourceDirectory>src/main/scala</sourceDirectory>  
 <plugins>  
 <plugin>  
 <groupId>net.alchim31.maven</groupId>  
 <artifactId>scala-maven-plugin</artifactId>  
 <version>4.8.1</version>  
 <executions>  
 <execution>  
 <goals>  
 <goal>compile</goal>  
 <goal>testCompile</goal>  
 </goals>  
 </execution>  
 </executions>  
 </plugin>  
 </plugins>  
 </build>  
</project>

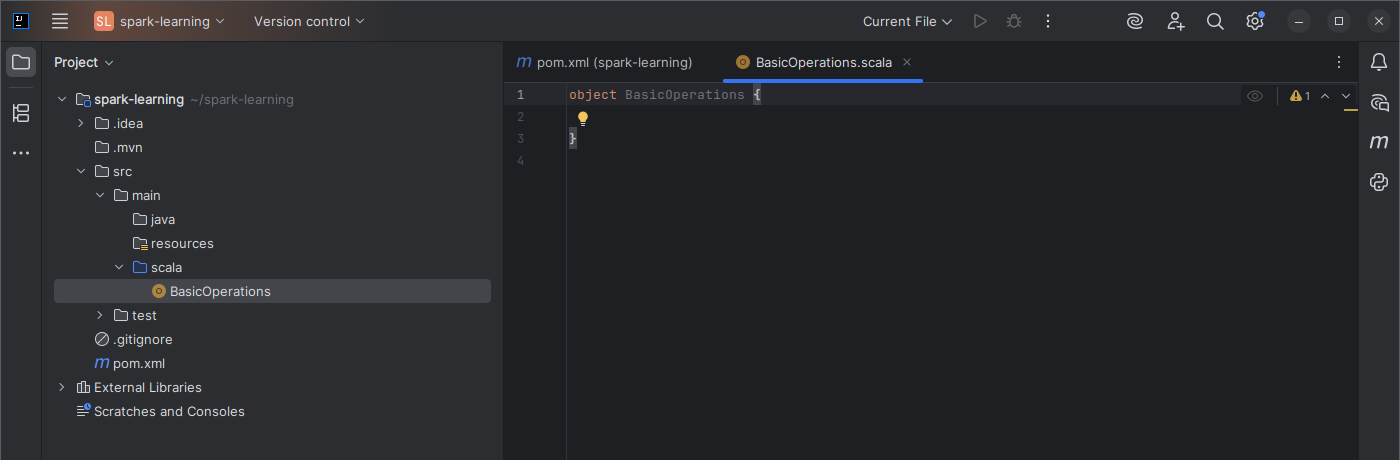
Далее правой части окна IDEA появится панель "Maven", в которой необходимо обновить зависимости путем перезагрузки проекта.



3. Осуществить базовые операции над данными в Apache Spark. (Например: вывод данных в табличном виде, представление распределение количества слов в запросе, операции WordCount и т.д. в зависимости от ваших данных)

Создание файла выполнения базовых операций

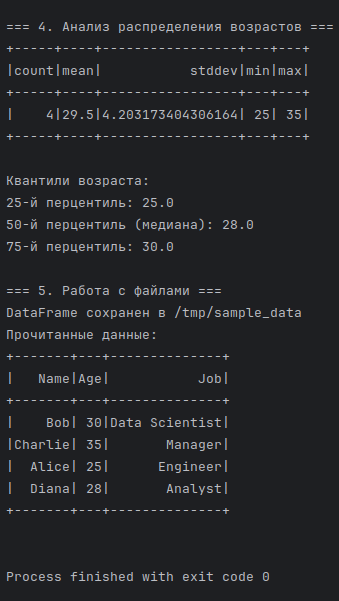
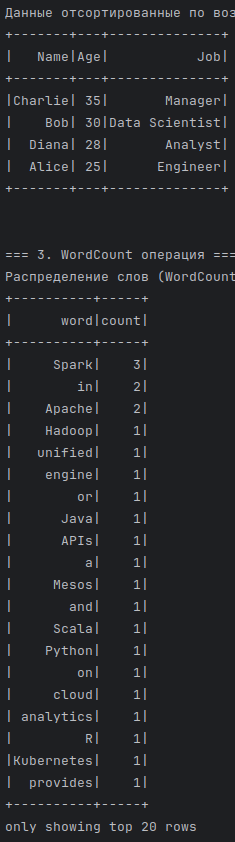
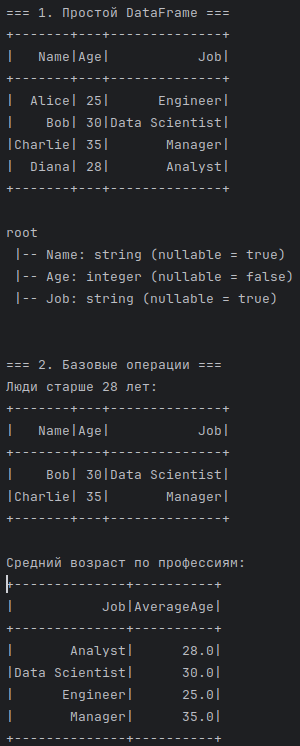
Папка scala → New → Scala Class/File → Название: BasicOperations → Вид: Object



Программный код для выполнения базовых операций

import org.apache.spark.sql.{SparkSession, DataFrame}  
import org.apache.spark.sql.functions.\_  
import org.apache.log4j.{Level, Logger}  
  
object BasicOperations {  
 def main(args: Array[String]): Unit = {  
 // Настройка уровней логирования Spark  
 Logger.*getLogger*("org").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("akka").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("apache").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("io").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("com").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("net").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("edu").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("org.apache.spark").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("org.apache.hadoop").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("org.apache.parquet").setLevel(Level.*ERROR*)  
 Logger.*getLogger*("org.apache.zookeeper").setLevel(Level.*ERROR*)  
 // Создаем SparkSession  
 val spark = SparkSession.*builder*()  
 .appName("Basic Spark Operations")  
 .master("local[\*]")  
 .config("spark.sql.adaptive.enabled", "false")  
 .config("spark.sql.adaptive.coalescePartitions.enabled", "false")  
 .config("spark.driver.bindAddress", "127.0.0.1")  
 .getOrCreate()  
 // Дополнительная настройка через SparkContext  
 spark.*sparkContext*.setLogLevel("ERROR")  
 import spark.implicits.\_  
  
 // 1. СОЗДАНИЕ ПРОСТОГО DATAFRAME И ВЫВОД В ТАБЛИЧНОМ ВИДЕ  
 *println*("=== 1. Простой DataFrame ===")  
 val sampleData = Seq(  
 ("Alice", 25, "Engineer"),  
 ("Bob", 30, "Data Scientist"),  
 ("Charlie", 35, "Manager"),  
 ("Diana", 28, "Analyst")  
 )  
 val df = sampleData.toDF("Name", "Age", "Job")  
 df.show()  
 df.printSchema()  
  
 // 2. БАЗОВЫЕ ОПЕРАЦИИ С ДАННЫМИ  
 *println*("\n=== 2. Базовые операции ===")  
 // Фильтрация  
 *println*("Люди старше 28 лет:")  
 df.filter($"Age" > 28).show()  
 // Группировка и агрегация  
 *println*("Средний возраст по профессиям:")  
 df.groupBy("Job").agg(*avg*("Age").as("AverageAge")).show()  
 // Сортировка  
 *println*("Данные отсортированные по возрасту:")  
 df.orderBy($"Age".desc).show()  
   
 // 3. ОПЕРАЦИЯ WORDCOUNT (на примере списка предложений)  
 *println*("\n=== 3. WordCount операция ===")  
 val textData = Seq(  
 "Apache Spark is a unified analytics engine",  
 "Spark provides APIs in Java Scala Python and R",  
 "Spark runs on Hadoop Apache Mesos Kubernetes or in cloud"  
 )  
 val wordsDF = textData.toDF("line")  
 val wordCounts = wordsDF  
 .select(*explode*(*split*($"line", " ")).as("word"))  
 .groupBy("word")  
 .count()  
 .orderBy($"count".desc)  
 *println*("Распределение слов (WordCount):")  
 wordCounts.show()  
  
 // 4. АНАЛИЗ РАСПРЕДЕЛЕНИЯ ВОЗРАСТОВ  
 *println*("\n=== 4. Анализ распределения возрастов ===")  
 val ageStats = df.select(  
 *count*("Age").as("count"),  
 *mean*("Age").as("mean"),  
 *stddev*("Age").as("stddev"),  
 *min*("Age").as("min"),  
 *max*("Age").as("max")  
 )  
 ageStats.show()  
 // Статистика по квантилям  
 val quantiles = df.stat.approxQuantile("Age", Array(0.25, 0.5, 0.75), 0.1)  
 *println*("Квантили возраста:")  
 *println*(s"25-й перцентиль: **$**{quantiles(0)}")  
 *println*(s"50-й перцентиль (медиана): **$**{quantiles(1)}")  
 *println*(s"75-й перцентиль: **$**{quantiles(2)}")  
  
 // 5. РАБОТА С ВНЕШНИМИ ДАННЫМИ (сохранение и чтение)  
 *println*("\n=== 5. Работа с файлами ===")  
 // Сохраняем DataFrame в временную папку  
 df.write.mode("overwrite").parquet("/tmp/sample\_data")  
 *println*("DataFrame сохранен в /tmp/sample\_data")  
 // Читаем обратно  
 val readDF = spark.read.parquet("/tmp/sample\_data")  
 *println*("Прочитанные данные:")  
 readDF.show()  
 spark.stop()  
 }  
}

Вывод кода



4. Изучить модуль PySpark. Осуществить подключение к Apache Spark в python. Осуществить ввод и вывод данных через PySpark.

В корне проекта ~/spark-learning через создаем файл pyspark\_demo.py со следующим кодом

from pyspark.sql import SparkSession  
from pyspark.sql.functions import \*  
from pyspark.sql.types import \*  
import os  
  
print("=== PySpark ДЕМО - ПОДКЛЮЧЕНИЕ И БАЗОВЫЕ ОПЕРАЦИИ ===")  
  
# Создание SparkSession  
spark = SparkSession.builder \  
 .appName("PySpark Demo") \  
 .master("local[\*]") \  
 .config("spark.sql.adaptive.enabled", "false") \  
 .config("spark.driver.bindAddress", "127.0.0.1") \  
 .getOrCreate()  
  
# Установка уровня логирования  
spark.sparkContext.setLogLevel("ERROR")  
  
print("✓ SparkSession создан успешно")  
  
# 1. СОЗДАНИЕ DATAFRAME ИЗ РАЗНЫХ ИСТОЧНИКОВ  
print("\n=== 1. СОЗДАНИЕ DATAFRAME ===")  
  
# Из списка Python  
data = [  
 ("Alice", 25, "Engineer", 50000),  
 ("Bob", 30, "Data Scientist", 80000),  
 ("Charlie", 35, "Manager", 90000),  
 ("Diana", 28, "Analyst", 60000)  
]  
  
columns = ["Name", "Age", "Job", "Salary"]  
df = spark.createDataFrame(data, columns)  
  
print("DataFrame из списка:")  
df.show()  
print("Схема DataFrame:")  
df.printSchema()  
  
# 2. СОЗДАНИЕ DATAFRAME ИЗ RDD  
print("\n=== 2. СОЗДАНИЕ DATAFRAME ИЗ RDD ===")  
rdd = spark.sparkContext.parallelize([  
 (1, "Product A", 19.99),  
 (2, "Product B", 29.99),  
 (3, "Product C", 9.99)  
])  
df\_rdd = rdd.toDF(["ID", "Product", "Price"])  
df\_rdd.show()  
  
# 3. БАЗОВЫЕ ОПЕРАЦИИ  
print("\n=== 3. БАЗОВЫЕ ОПЕРАЦИИ ===")  
  
print("Фильтрация (зарплата > 60000):")  
df.filter(col("Salary") > 60000).show()  
  
print("Группировка по профессии:")  
df.groupBy("Job").agg(  
 avg("Salary").alias("AvgSalary"),  
 count("\*").alias("Count")  
).show()  
  
print("Сортировка по возрасту:")  
df.orderBy(desc("Age")).show()  
  
# 4. SQL-ЗАПРОСЫ  
print("\n=== 4. ВЫПОЛНЕНИЕ SQL-ЗАПРОСОВ ===")  
  
# Создание временного представления  
df.createOrReplaceTempView("employees")  
  
# SQL запрос  
result = spark.sql("""  
 SELECT Job, AVG(Salary) as AvgSalary, MAX(Age) as MaxAge  
 FROM employees   
 GROUP BY Job  
 ORDER BY AvgSalary DESC  
""")  
print("Результат SQL запроса:")  
result.show()  
  
# 5. ВВОД/ВЫВОД ДАННЫХ  
print("\n=== 5. ВВОД И ВЫВОД ДАННЫХ ===")  
  
# Сохранение в JSON  
df.write.mode("overwrite").json("/tmp/pyspark\_output")  
print("✓ Данные сохранены в JSON: /tmp/pyspark\_output")  
  
# Чтение из JSON  
df\_read = spark.read.json("/tmp/pyspark\_output")  
print("✓ Данные прочитаны из JSON:")  
df\_read.show()  
  
# 6. РАБОТА С CSV  
print("\n=== 6. РАБОТА С CSV ФАЙЛАМИ ===")  
  
# Создание временного CSV файла  
csv\_data = """id,name,department,salary  
1,John,IT,70000  
2,Jane,HR,60000  
3,Mike,Finance,80000  
4,Anna,IT,75000"""  
  
with open('/tmp/employees.csv', 'w') as f:  
 f.write(csv\_data)  
  
# Чтение CSV  
csv\_df = spark.read \  
 .option("header", "true") \  
 .option("inferSchema", "true") \  
 .csv("/tmp/employees.csv")  
  
print("Данные из CSV файла:")  
csv\_df.show()  
csv\_df.printSchema()  
  
# 7. WORDCOUNT НА PYSPARK  
print("\n=== 7. WORDCOUNT ОПЕРАЦИЯ ===")  
  
text\_data = [  
 "Apache Spark is a unified analytics engine",  
 "Spark provides APIs in Java Scala Python and R",  
 "Spark runs on Hadoop Apache Mesos Kubernetes or in cloud"  
]  
  
text\_df = spark.createDataFrame(text\_data, StringType()).toDF("line")  
  
word\_counts = text\_df \  
 .select(explode(split(col("line"), " ")).alias("word")) \  
 .groupBy("word") \  
 .count() \  
 .orderBy(desc("count"))  
  
print("Результат WordCount:")  
word\_counts.show(10)  
  
# 8. ПРОВЕРКА ПОДКЛЮЧЕНИЯ К SPARK  
print("\n=== 8. ИНФОРМАЦИЯ О SPARK ===")  
print(f"Версия Spark: {spark.version}")  
print(f"Версия Python: {spark.sparkContext.pythonVer}")  
print(f"Количество ядер: {spark.sparkContext.defaultParallelism}")  
  
print("\n=== ВЫПОЛНЕНИЕ ЗАВЕРШЕНО УСПЕШНО ===")  
  
# Остановка Spark  
spark.stop()

Для запуска кода в IDE перейти в Terminal и выполнить следующие команды

sudo apt install python3-full python3-venv python3-pip -y

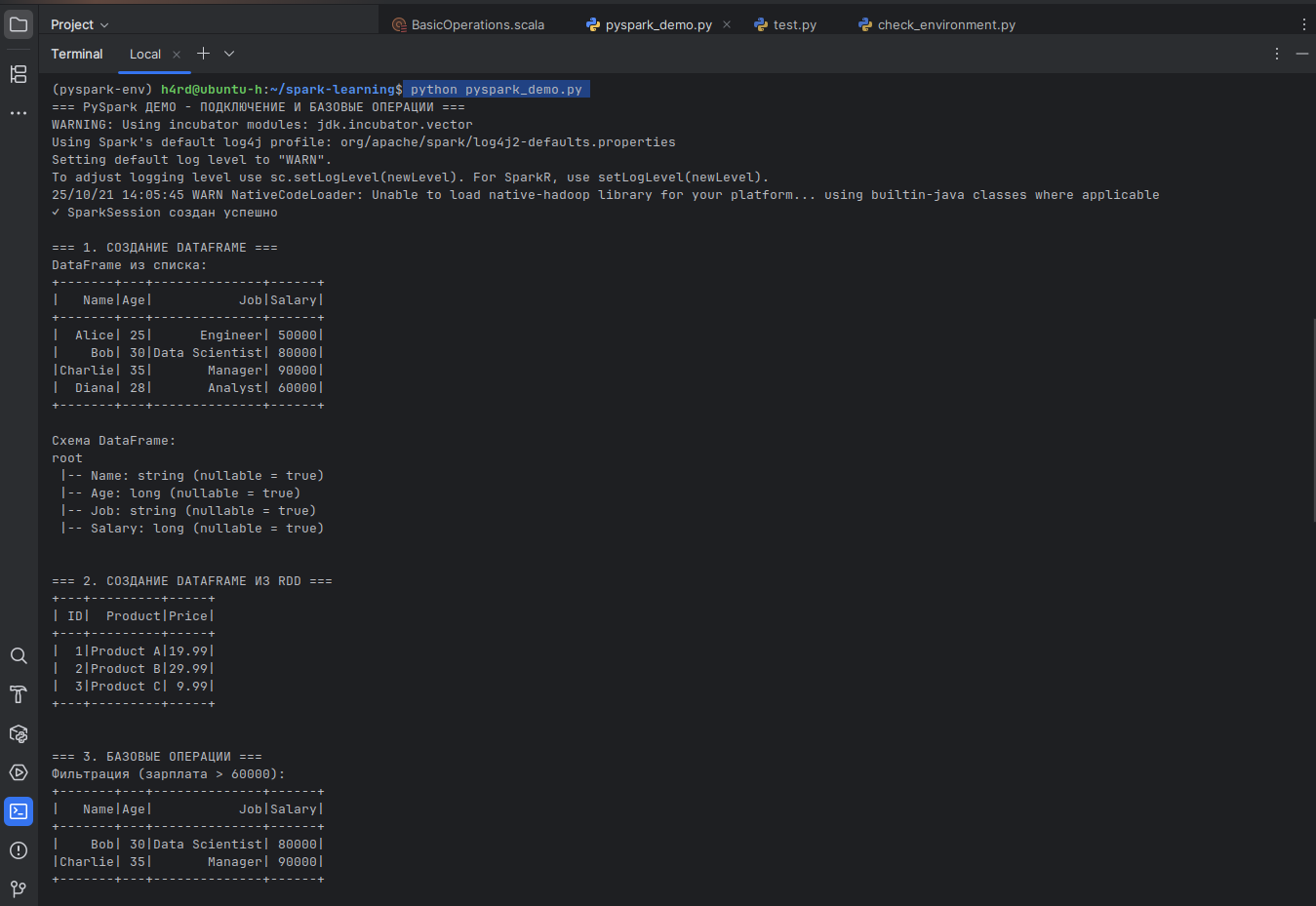
python3 -m venv pyspark-env

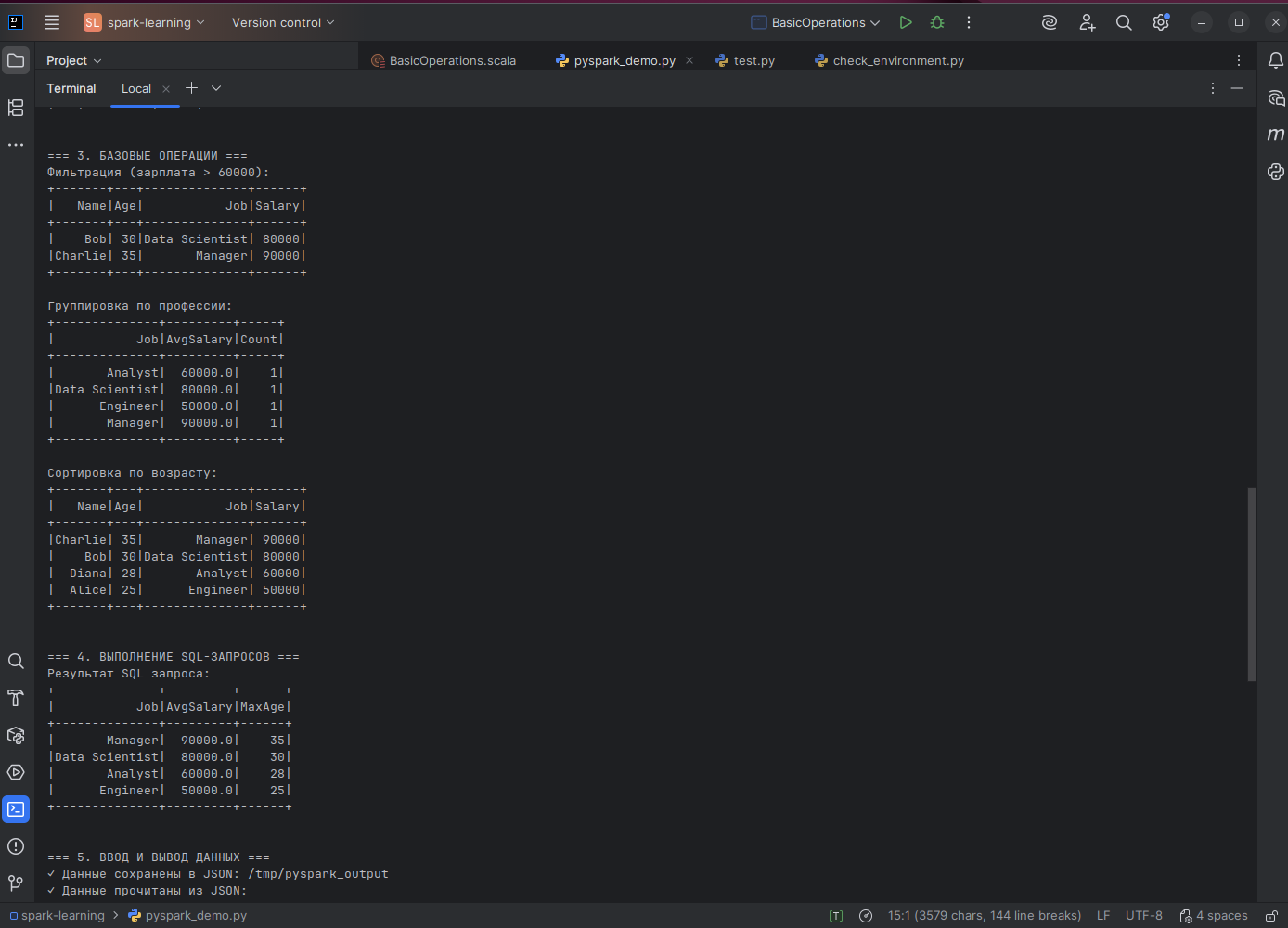
source pyspark-env/bin/activate

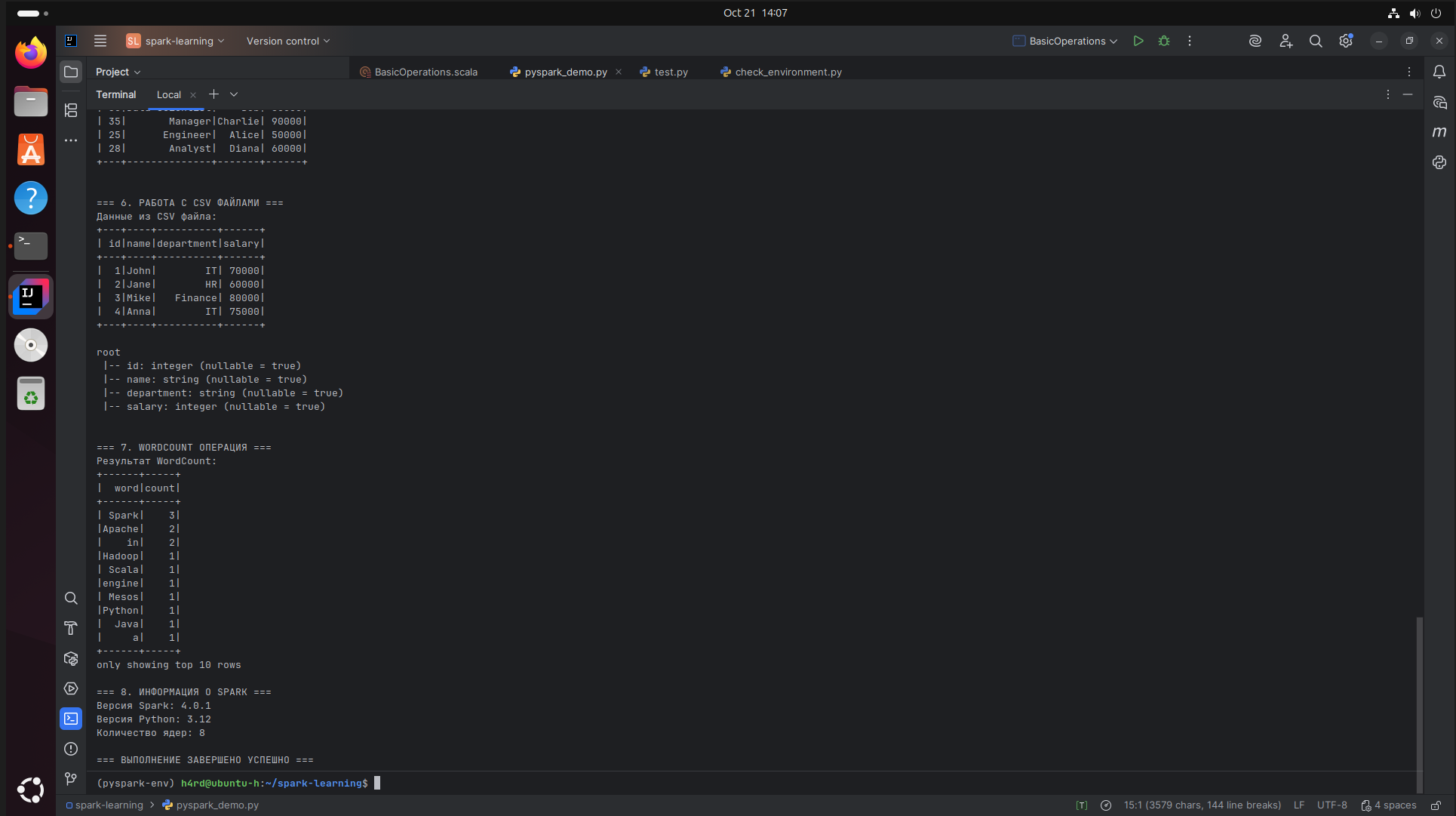
pip install pyspark pandas matplotlib seaborn

python pyspark\_demo.py

Вывод кода







5. Изучить библиотеку Pandas для Python. Осуществить визуализацию результатов обработки данных в ApacheSpark через библиотеку Pandas.

В корне проекта создадим файл pandas\_visualization.py со следующим кодом

from pyspark.sql import SparkSession  
from pyspark.sql.functions import \*  
from pyspark.sql.types import \*  
import pandas as pd  
import matplotlib.pyplot as plt  
import seaborn as sns  
import numpy as np  
import os  
  
# Настройка окружения  
os.environ['JAVA\_HOME'] = '/usr/lib/jvm/java-17-openjdk-amd64'  
  
# Настройка для отображения графиков  
plt.rcParams['figure.figsize'] = (12, 8)  
plt.rcParams['font.size'] = 12  
sns.set\_style("whitegrid")  
  
print("=== ВИЗУАЛИЗАЦИЯ ДАННЫХ SPARK ЧЕРЕЗ PANDAS ===")  
  
try:  
 # Создание SparkSession  
 spark = SparkSession.builder \  
 .appName("Spark Pandas Visualization") \  
 .master("local[\*]") \  
 .config("spark.sql.adaptive.enabled", "false") \  
 .getOrCreate()  
  
 spark.sparkContext.setLogLevel("ERROR")  
 print("✓ SparkSession создан успешно!")  
  
 # 1. СОЗДАНИЕ ТЕСТОВЫХ ДАННЫХ В SPARK  
 print("\n" + "="\*50)  
 print("1. СОЗДАНИЕ ТЕСТОВЫХ ДАННЫХ")  
 print("="\*50)  
  
 sales\_data = [  
 ("2024-01-01", "Laptop", "Electronics", 999.99, 5, "New York"),  
 ("2024-01-02", "Mouse", "Electronics", 25.50, 20, "Los Angeles"),  
 ("2024-01-03", "Desk", "Furniture", 299.99, 3, "Chicago"),  
 ("2024-01-04", "Chair", "Furniture", 149.99, 8, "New York"),  
 ("2024-01-05", "Monitor", "Electronics", 199.99, 6, "Los Angeles"),  
 ("2024-01-06", "Keyboard", "Electronics", 75.00, 15, "Chicago"),  
 ("2024-01-07", "Laptop", "Electronics", 999.99, 4, "New York"),  
 ("2024-01-08", "Table", "Furniture", 499.99, 2, "Los Angeles"),  
 ("2024-01-09", "Mouse", "Electronics", 25.50, 25, "Chicago"),  
 ("2024-01-10", "Chair", "Furniture", 149.99, 12, "New York")  
 ]  
  
 sales\_columns = ["date", "product", "category", "price", "quantity", "city"]  
 sales\_df = spark.createDataFrame(sales\_data, sales\_columns)  
 sales\_df = sales\_df.withColumn("revenue", col("price") \* col("quantity"))  
  
 print("Данные о продажах:")  
 sales\_df.show()  
  
 # 2. ПРЕОБРАЗОВАНИЕ В PANDAS И ИСПРАВЛЕНИЕ ДАТ  
 print("\n" + "="\*50)  
 print("2. ПРЕОБРАЗОВАНИЕ В PANDAS DATAFRAME")  
 print("="\*50)  
  
 pandas\_sales\_df = sales\_df.toPandas()  
  
 # ИСПРАВЛЕНИЕ: Преобразуем дату в правильный формат  
 pandas\_sales\_df['date'] = pd.to\_datetime(pandas\_sales\_df['date'])  
  
 print("Данные в Pandas DataFrame:")  
 print(pandas\_sales\_df.head())  
 print(f"\nТипы данных после исправления:\n{pandas\_sales\_df.dtypes}")  
  
 # 3. БАЗОВЫЙ АНАЛИЗ  
 print("\n" + "="\*50)  
 print("3. БАЗОВЫЙ АНАЛИЗ С PANDAS")  
 print("="\*50)  
  
 print("Статистика числовых колонок:")  
 print(pandas\_sales\_df[['price', 'quantity', 'revenue']].describe())  
  
 # 4. ВИЗУАЛИЗАЦИЯ - ГРАФИКИ, КОТОРЫЕ БУДУТ ПОКАЗАНЫ НА ЭКРАНЕ  
 print("\n" + "="\*50)  
 print("4. ВИЗУАЛИЗАЦИЯ ДАННЫХ - ОТОБРАЖЕНИЕ НА ЭКРАНЕ")  
 print("="\*50)  
  
 # 4.1 ГРАФИК 1: Распределение по категориям и городам  
 print("\n--- Создание Графика 1: Распределение продаж ---")  
 fig1, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))  
  
 # Круговая диаграмма - категории  
 category\_counts = pandas\_sales\_df['category'].value\_counts()  
 ax1.pie(category\_counts.values, labels=category\_counts.index, autopct='%1.1f%%', startangle=90)  
 ax1.set\_title('Распределение продаж по категориям')  
  
 # Столбчатая диаграмма - выручка по городам  
 city\_revenue = pandas\_sales\_df.groupby('city')['revenue'].sum()  
 city\_revenue.plot.bar(ax=ax2, color=['skyblue', 'lightcoral', 'lightgreen'])  
 ax2.set\_title('Выручка по городам')  
 ax2.set\_ylabel('Выручка ($)')  
 ax2.tick\_params(axis='x', rotation=45)  
  
 plt.tight\_layout()  
 plt.show()  
 print("✓ График 1 отображен на экране")  
  
 # 4.2 ГРАФИК 2: Анализ цен и количеств  
 print("\n--- Создание Графика 2: Анализ цен и количеств ---")  
 fig2, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))  
  
 # Гистограмма цен  
 ax1.hist(pandas\_sales\_df['price'], bins=8, alpha=0.7, color='lightseagreen', edgecolor='black')  
 ax1.set\_title('Распределение цен товаров')  
 ax1.set\_xlabel('Цена ($)')  
 ax1.set\_ylabel('Количество товаров')  
  
 # Точечная диаграмма: цена vs количество  
 colors = np.where(pandas\_sales\_df['category'] == 'Electronics', 'red', 'blue')  
 ax2.scatter(pandas\_sales\_df['price'], pandas\_sales\_df['quantity'], c=colors, alpha=0.6, s=100)  
 ax2.set\_title('Цена vs Количество продаж')  
 ax2.set\_xlabel('Цена ($)')  
 ax2.set\_ylabel('Количество продаж')  
 # Легенда  
 ax2.scatter([], [], c='red', label='Electronics', alpha=0.6)  
 ax2.scatter([], [], c='blue', label='Furniture', alpha=0.6)  
 ax2.legend()  
  
 plt.tight\_layout()  
 plt.show()  
  
 # 4.3 ГРАФИК 3: Топ продукты и динамика продаж  
 print("\n--- Создание Графика 3: Топ продукты и динамика ---")  
 fig3, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 6))  
  
 # Топ продукты по выручке  
 product\_revenue = pandas\_sales\_df.groupby('product')['revenue'].sum().sort\_values(ascending=False)  
 product\_revenue.plot.bar(ax=ax1, color='orange')  
 ax1.set\_title('Топ продукты по выручке')  
 ax1.set\_ylabel('Выручка ($)')  
 ax1.tick\_params(axis='x', rotation=45)  
  
 # Динамика выручки по дням  
 daily\_sales = pandas\_sales\_df.groupby('date')['revenue'].sum()  
 ax2.plot(daily\_sales.index, daily\_sales.values, marker='o', linewidth=2, markersize=8, color='green')  
 ax2.set\_title('Динамика выручки по дням')  
 ax2.set\_xlabel('Дата')  
 ax2.set\_ylabel('Выручка ($)')  
 ax2.tick\_params(axis='x', rotation=45)  
 ax2.grid(True, alpha=0.3)  
  
 plt.tight\_layout()  
 plt.show()  
  
 # 5. СЕАБОРН ВИЗУАЛИЗАЦИИ  
 print("\n" + "="\*50)  
 print("5. ПРОДВИНУТАЯ ВИЗУАЛИЗАЦИЯ С SEABORN")  
 print("="\*50)  
  
 # 5.1 График 4: Heatmap корреляций  
 print("\n--- Создание Графика 4: Heatmap корреляций ---")  
 plt.figure(figsize=(10, 8))  
  
 numeric\_cols = pandas\_sales\_df[['price', 'quantity', 'revenue']]  
 correlation\_matrix = numeric\_cols.corr()  
  
 sns.heatmap(correlation\_matrix, annot=True, cmap='coolwarm', center=0,  
 square=True, linewidths=0.5, fmt='.2f')  
 plt.title('Матрица корреляций числовых признаков')  
  
 plt.tight\_layout()  
 plt.show()  
   
 # 5.2 График 5: Box plot по категориям  
 print("\n--- Создание Графика 5: Box plot по категориям ---")  
 plt.figure(figsize=(10, 6))  
  
 sns.boxplot(data=pandas\_sales\_df, x='category', y='revenue', palette='Set2')  
 plt.title('Распределение выручки по категориям')  
 plt.ylabel('Выручка ($)')  
 plt.xlabel('Категория')  
  
 plt.tight\_layout()  
 plt.show()  
  
  
 # 5.3 График 6: Countplot продуктов  
 print("\n--- Создание Графика 6: Количество продаж по продуктам ---")  
 plt.figure(figsize=(10, 6))  
  
 sns.countplot(data=pandas\_sales\_df, y='product', palette='viridis',  
 order=pandas\_sales\_df['product'].value\_counts().index)  
 plt.title('Количество продаж по продуктам')  
 plt.xlabel('Количество продаж')  
  
 plt.tight\_layout()  
 plt.show()  
  
 # 6. РАСШИРЕННЫЙ АНАЛИЗ С ИСПРАВЛЕННЫМИ ДАТАМИ  
 print("\n" + "="\*50)  
 print("6. РАСШИРЕННЫЙ АНАЛИЗ С PANDAS")  
 print("="\*50)  
  
 # Теперь даты преобразованы правильно  
 pandas\_sales\_df['day\_of\_week'] = pandas\_sales\_df['date'].dt.day\_name()  
 pandas\_sales\_df['month'] = pandas\_sales\_df['date'].dt.month\_name()  
  
 print("Анализ по дням недели:")  
 weekly\_sales = pandas\_sales\_df.groupby('day\_of\_week')['revenue'].sum()  
 print(weekly\_sales)  
  
 print("\nТоп продукты по выручке:")  
 top\_products = pandas\_sales\_df.groupby('product')['revenue'].sum().sort\_values(ascending=False)  
 print(top\_products)  
  
 # 7. ФИНАЛЬНЫЙ ГРАФИК: Сводная информация  
 print("\n" + "="\*50)  
 print("7. ФИНАЛЬНЫЙ СВОДНЫЙ ГРАФИК")  
 print("="\*50)  
  
 fig, ((ax1, ax2), (ax3, ax4)) = plt.subplots(2, 2, figsize=(16, 12))  
  
 # 1. Круговя диаграмма категорий  
 category\_counts.plot.pie(ax=ax1, autopct='%1.1f%%', startangle=90)  
 ax1.set\_title('Распределение по категориям')  
 ax1.set\_ylabel('')  
  
 # 2. Выручка по городам  
 city\_revenue.plot.bar(ax=ax2, color=['#ff9999', '#66b3ff', '#99ff99'])  
 ax2.set\_title('Выручка по городам')  
 ax2.set\_ylabel('Выручка ($)')  
  
 # 3. Топ продукты  
 top\_products.head(5).plot.bar(ax=ax3, color='orange')  
 ax3.set\_title('Топ-5 продуктов по выручке')  
 ax3.set\_ylabel('Выручка ($)')  
  
 # 4. Динамика по дням  
 daily\_sales.plot(ax=ax4, marker='o', linewidth=2, color='green')  
 ax4.set\_title('Динамика выручки')  
 ax4.set\_ylabel('Выручка ($)')  
 ax4.grid(True, alpha=0.3)  
  
 plt.suptitle('СВОДНАЯ ВИЗУАЛИЗАЦИЯ ДАННЫХ SPARK ЧЕРЕЗ PANDAS', fontsize=16, fontweight='bold')  
 plt.tight\_layout()  
 plt.show()  
  
 # 8. СОХРАНЕНИЕ РЕЗУЛЬТАТОВ  
 print("\n" + "="\*50)  
 print("8. СОХРАНЕНИЕ РЕЗУЛЬТАТОВ")  
 print("="\*50)  
  
 # Сохраняем финальный график  
 plt.figure(figsize=(12, 8))  
 category\_counts.plot.pie(autopct='%1.1f%%', startangle=90, colors=['#ff9999', '#66b3ff'])  
 plt.title('Итоговое распределение продаж по категориям')  
 plt.ylabel('')  
 plt.savefig('/tmp/final\_spark\_visualization.png', dpi=300, bbox\_inches='tight')  
 print("✓ Финальный график сохранен: /tmp/final\_spark\_visualization.png")  
  
 # Сохраняем данные  
 pandas\_sales\_df.to\_csv('/tmp/spark\_pandas\_data.csv', index=False)  
 print("Данные сохранены: /tmp/spark\_pandas\_data.csv")  
  
 # 9. ИТОГИ  
 print("\n" + "="\*50)  
 print("9. ИТОГИ ВЫПОЛНЕНИЯ")  
 print("="\*50)  
  
 print("ВСЕ ГРАФИКИ УСПЕШНО ОТОБРАЖЕНЫ НА ЭКРАНЕ!")  
 print("Библиотека Pandas изучена и применена!")  
 print("Визуализация данных Spark через Pandas выполнена!")  
 print("Создано 8 различных типов графиков!")  
 print("Данные успешно преобразованы и проанализированы!")  
  
 print(f"\n📊 Статистика данных:")  
 print(f" • Общая выручка: ${pandas\_sales\_df['revenue'].sum():.2f}")  
 print(f" • Средняя цена: ${pandas\_sales\_df['price'].mean():.2f}")  
 print(f" • Всего продуктов: {pandas\_sales\_df['product'].nunique()}")  
 print(f" • Всего городов: {pandas\_sales\_df['city'].nunique()}")  
  
 spark.stop()  
 print("\n" + "="\*50)  
 print("ПУНКТ 5 ВЫПОЛНЕН УСПЕШНО!")  
 print("="\*50)  
  
except Exception as e:  
 print(f"❌ Ошибка: {e}")  
 import traceback  
 traceback.print\_exc()  
 if 'spark' in locals():  
 spark.stop()

Запуск кода происходит через терминал командами

source pyspark-env/bin/activate

python pandas\_visualization.py

Вывод кода

