

CHS
Unit-4 Task(Spark)

AMAN GUPTA(2021UCS1602)

GARIMA(2021UCS1605)

TARUN MITTAL(2021UCS1608)

INDEX

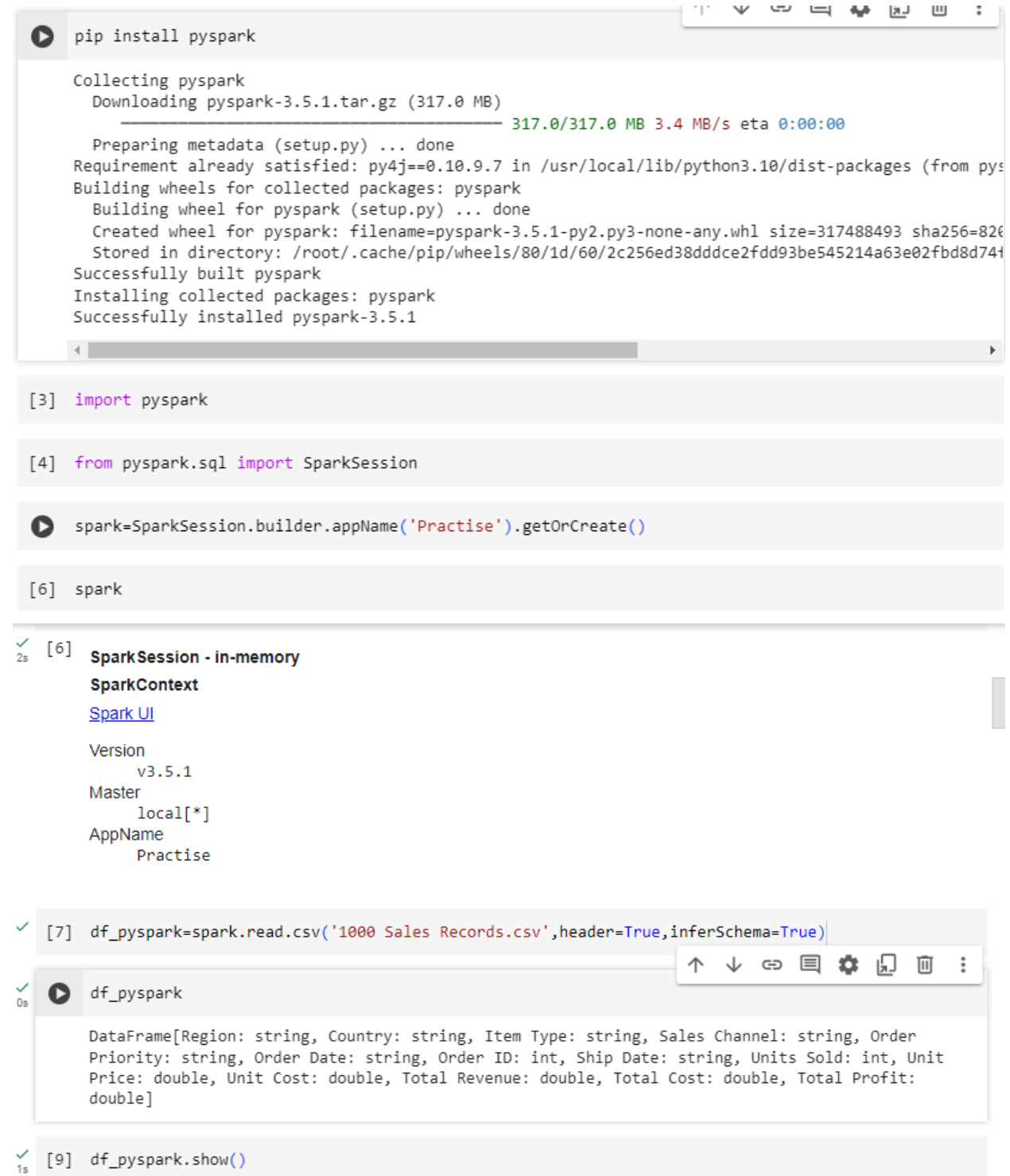
S.No	Questions
1.	Explore RDD in spark
2.	In PySpark, create a program that reads a CSV file containing sales data, performs data cleaning by handling missing values and removing duplicates, calculates the total sales amount for each product, and finally, outputs the results to a new CSV file. Ensure to use transformations and actions in your PySpark script

Q1) Explore RDD in spark

In Spark, RDD stands for Resilient Distributed Dataset. RDD is the fundamental data structure of Spark, representing an immutable, distributed collection of objects that can be operated on in parallel across a cluster. RDDs are designed to handle large-scale data processing tasks efficiently by distributing the data across multiple nodes in a cluster and enabling parallel processing operations.

- **Resilient:** RDDs are resilient because they can automatically recover from failures. They achieve this resilience through lineage, which is a record of the transformations applied to the base data set. If a partition of an RDD is lost due to a worker node failure, Spark can use the lineage information to recompute the lost partition from the original data.
- **Distributed:** RDDs are distributed across multiple nodes in a cluster, allowing for parallel processing. Spark automatically distributes the data among worker nodes and partitions it to optimize parallelism and data locality.
- **Dataset:** RDDs represent collections of objects, which can be of any type, including primitive types, custom objects, or even other RDDs. RDDs can be created from various data sources, including HDFS, HBase, Amazon S3, local file systems, etc.
- **Immutable:** RDDs are immutable, meaning once created, they cannot be changed. Instead, transformations on RDDs create new RDDs, preserving the original data. This immutability ensures that RDDs are thread-safe and can be efficiently parallelized.
- **Transformations and Actions:** RDDs support two types of operations: transformations and actions. Transformations are lazy operations that define how to transform one RDD into another (e.g., map, filter, join), while actions are operations that trigger computation and return results to the driver program (e.g., count, collect, save).
- **Lineage:** RDDs maintain a lineage graph, which is a directed acyclic graph (DAG) representing the series of transformations applied to the base data to produce the current RDD. This lineage information enables fault tolerance by allowing Spark to recompute lost partitions of an RDD in case of failures.
- **Caching:** RDDs support caching and persistence, allowing users to persist intermediate results in memory or disk for faster access in subsequent operations. Caching is particularly useful when an RDD is used multiple times in a computation or when the RDD fits entirely in memory.

Q2) In PySpark, create a program that reads a CSV file containing sales data, performs data cleaning by handling missing values and removing duplicates, calculates the total sales amount for each product, and finally, outputs the results to a new CSV file. Ensure to use transformations and actions in your PySpark script



The screenshot displays a Jupyter Notebook interface with the following content:

```
pip install pyspark
```

Collecting pyspark
Downloading pyspark-3.5.1.tar.gz (317.0 MB)
317.0/317.0 MB 3.4 MB/s eta 0:00:00
Preparing metadata (setup.py) ... done
Requirement already satisfied: py4j==0.10.9.7 in /usr/local/lib/python3.10/dist-packages (from pyspark)
Building wheels for collected packages: pyspark
Building wheel for pyspark (setup.py) ... done
Created wheel for pyspark: filename=pyspark-3.5.1-py2.py3-none-any.whl size=317488493 sha256=826...
Stored in directory: /root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd8d74...
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1

```
[3] import pyspark
```

```
[4] from pyspark.sql import SparkSession
```

```
spark=SparkSession.builder.appName('Practise').getOrCreate()
```

```
[6] spark
```

SparkSession - in-memory
SparkContext
[Spark UI](#)
Version
v3.5.1
Master
local[*]
AppName
Practise

```
[7] df_pyspark=spark.read.csv('1000 Sales Records.csv',header=True,inferSchema=True)
```

```
df_pyspark
```

DataFrame[Region: string, Country: string, Item Type: string, Sales Channel: string, Order Priority: string, Order Date: string, Order ID: int, Ship Date: string, Units Sold: int, Unit Price: double, Unit Cost: double, Total Revenue: double, Total Cost: double, Total Profit: double]

```
[9] df_pyspark.show()
```

Data cleaning by handling missing values and removing duplicates

```
[10] df_pyspark.head(3)
```

```
[Row(Region='Middle East and North Africa', Country=None, Item Type='Cosmetics', Sales Channel='Offline', Order Priority='M', Order Date='10/18/2014', Order ID=686800706, Ship Date='10/31/2014', Units Sold=8446, Unit Price=437.2, Unit Cost=263.33, Total Revenue=3692591.2, Total Cost=2224085.18, Total Profit=1468506.02),
 Row(Region='North America', Country='Canada', Item Type='Vegetables', Sales Channel='Online', Order Priority='M', Order Date='11/7/2011', Order ID=185941302, Ship Date='12/8/2011', Units Sold=None, Unit Price=154.06, Unit Cost=90.93, Total Revenue=None, Total Cost=274426.74, Total Profit=190526.34),
 Row(Region='Middle East and North Africa', Country='Libya', Item Type='Baby Food', Sales Channel='Offline', Order Priority='C', Order Date='10/31/2016', Order ID=246222341, Ship Date='12/9/2016', Units Sold=1517, Unit Price=None, Unit Cost=159.42, Total Revenue=387259.76, Total Cost=None, Total Profit=145419.62)]
```

```
df_pyspark=df_pyspark.drop('Ship Date')
```

```
[12] df_pyspark
```

```
DataFrame[Region: string, Country: string, Item Type: string, Sales Channel: string, Order Priority: string, Order Date: string, Order ID: int, Units Sold: int, Unit Price: double, Unit Cost: double, Total Revenue: double, Total Cost: double, Total Profit: double]
```

```
[13] df_pyspark.show()
```

```
[14] df_pyspark=df_pyspark.na.drop(how="any",subset=['Country'])
```

```
df_pyspark.show()
```

```
[16] from pyspark.ml.feature import Imputer
```

```
imputer = Imputer(
    inputCols=['Units Sold', 'Unit Price', 'Unit Cost', 'Total Revenue', 'Total Cost'],
    outputCols=["{}_imputed".format(c) for c in ['Units Sold', 'Unit Price', 'Unit Cost', 'Total Revenue', 'Total Cost']],
    ).setStrategy("median")
```

```
imputer.fit(df_pyspark).transform(df_pyspark).show()
```

```
▶ imputed_df = imputer.fit(df_pyspark).transform(df_pyspark)
imputed_df.show()
```

```
▶ imputed_df.head()
```

```
▶ # Drop specified columns
columns_to_drop = ['Units Sold', 'Unit Price', 'Unit Cost', 'Total Revenue', 'Total Cost']
imputed_df = imputed_df.drop(*columns_to_drop)

# Show the DataFrame after dropping columns
imputed_df.show()
```

```
[23] imputed_df.write.csv("imputed_data.csv", header=True)
```

Output :

