Vaishnavi Saggurthi
vaishnavisaggurthi@gmail.com

DATABRICKS

What is Databricks?

Databricks is a cloud-based data and AI platform designed to store, process, analyse, and model large datasets in a fast and scalable way. It is built on Apache Spark, which enables distributed computing, allowing multiple machines to work together for high-speed data processing. Databricks provides a collaborative workspace with notebooks, clusters, and data tools, making it easier for teams to perform data engineering, analytics, and machine learning tasks.

Why Databricks Instead of Pandas or Hadoop?

Pandas:

- Works on a single machine.
- Best suited for small datasets.
- Easy to use but cannot handle big data.
- Not distributed.

Hadoop:

- Used for large-scale storage and batch processing.
- Reliable but slow because it reads from disk.
- Complex setup and maintenance.
- Not suitable for real-time processing.

Databricks:

- Cloud-based and highly scalable.
- Uses Apache Spark for fast, distributed, in-memory processing.
- Supports SQL, machine learning, streaming, and AI.
- Suitable for both real-time and large historical datasets.

Vaishnavi Saggurthi
vaishnavisaggurthi@gmail.com

Lakehouse Architecture

The Lakehouse architecture combines the advantages of Data Lakes and Data Warehouses into one system.

Why it is needed:

- Data lakes are cheap but unorganized.
- Data warehouses are organized but expensive.

Benefits of Lakehouse:

- Single storage layer for all workloads.
- Lower cost.
- High performance analytics.
- Supports ETL, BI, machine learning, and streaming in one place.
- No duplicate pipelines.

Medallion Structure:

- Bronze – Raw data.
- Silver – Cleaned and structured data.
- Gold – Aggregated and analytics-ready data.

Databricks Workspace Structure

Workspace:

- Central area containing folders, notebooks, and project files.

Notebooks:

- Interactive environment for writing and running code.
- Supports Python, SQL, and Scala.
- Displays results immediately.

Compute (Clusters):

- Machines that process your code.
- Can scale automatically depending on the workload.
- You pay only when the cluster is running.

Data Explorer:

- Interface to view databases, tables, and schemas.
- Useful for inspection and validation.

Vaishnavi Saggurthi
vaishnavisaggurthi@gmail.com

Storage vs Compute

Storage:

- Permanent location where data is saved.
- Stored in cloud-based systems like S3, ADLS, or GCS.
- Low-cost and scalable.

Compute:

- Temporary machines used to process data.
- Can be started or stopped as needed.
- Used for ETL, analytics, ML, and streaming.

Importance of Separation:

- Reduces cost.
- Allows unlimited storage.
- Multiple compute clusters can access the same data.

Industry Use Cases

Netflix:

- Processes large volumes of user activity.
- Builds real-time recommendation systems.
- Analyses viewing behaviour.

Shell:

- Uses sensor data for predictive maintenance.
- Optimizes equipment performance.
- Improves safety and reduces downtime.

Comcast:

- Monitors network performance.
- Detects issues in real-time.
- Improves customer service and reliability.

Vaishnavi Saggurthi
vaishnavisaggurthi@gmail.com

PYSPARK NOTEBOOK

Creating a DataFrame:

- Convert Python data into Spark DataFrame
- Display using. show()

Inspecting Data:

- printSchema()
- columns
- count()
- describe()

Column Operations:

- select() → choose columns
- withColumn() → add column
- withColumnRenamed() → rename column

Filtering Data:

- filter() → pick specific rows

Sorting & Grouping:

- orderBy()
- groupBy().agg()

Spark maps data **positionally** (like a zipper).
DataFrames are **immutable** → every change creates a new one.

BASIC PYSPARK COMMANDS

Data Inspection:

- printSchema()
- columns
- count()
- describe()

Row & Column Operations:

- select()
- filter()

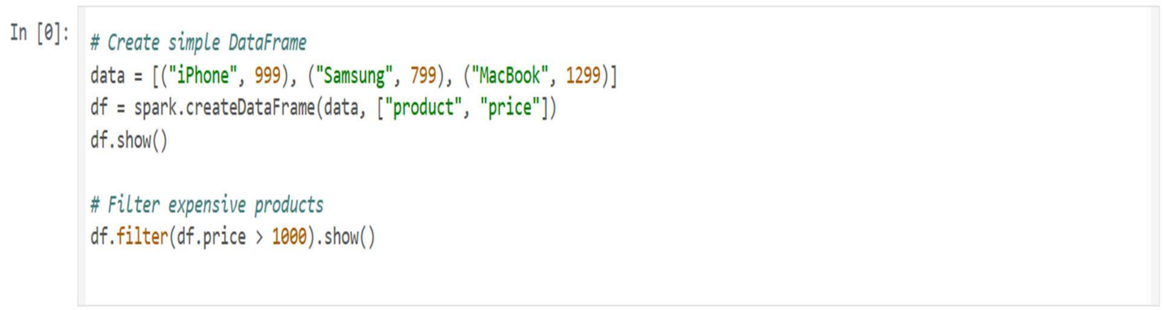
Vaishnavi Saggurthi
vaishnavisaggurthi@gmail.com

- withColumn()
- withColumnRenamed()

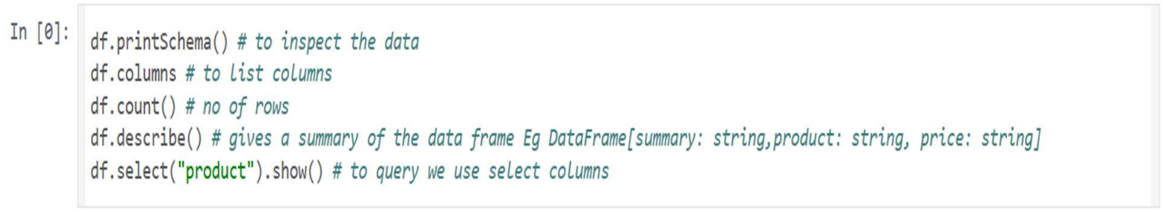
Ordering & Aggregation:

- orderBy()
- groupBy().agg()

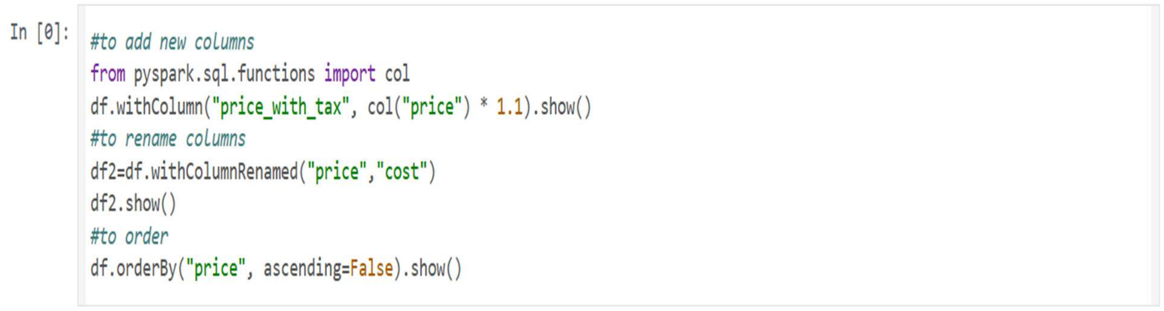
PYSPARK COMMANDS PRACTICE



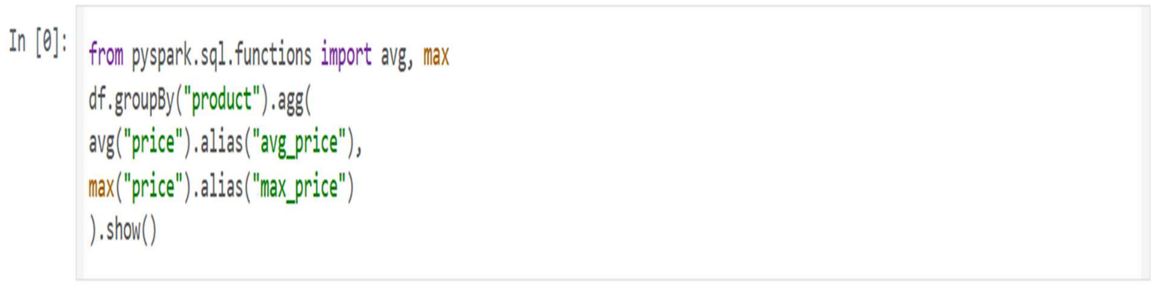
```
In [0]:
# Create simple DataFrame
data = [("iPhone", 999), ("Samsung", 799), ("MacBook", 1299)]
df = spark.createDataFrame(data, ["product", "price"])
df.show()

# Filter expensive products
df.filter(df.price > 1000).show()
```



```
In [0]:
df.printSchema() # to inspect the data
df.columns # to list columns
df.count() # no of rows
df.describe() # gives a summary of the data frame Eg DataFrame[summary: string,product: string, price: string]
df.select("product").show() # to query we use select columns
```



```
In [0]:
#to add new columns
from pyspark.sql.functions import col
df.withColumn("price_with_tax", col("price") * 1.1).show()
#to rename columns
df2=df.withColumnRenamed("price","cost")
df2.show()
#to order
df.orderBy("price", ascending=False).show()
```



```
In [0]:
from pyspark.sql.functions import avg, max
df.groupBy("product").agg(
avg("price").alias("avg_price"),
max("price").alias("max_price")
).show()
```