**📝 HiveQL - WEEK 7 Notes**

**✨ DDL (Data Definition Language)**

Used to define or modify the structure of tables.

➡️ **CREATE TABLE**

CREATE TABLE employee (

id INT,

name STRING,

salary FLOAT

)

ROW FORMAT DELIMITED

FIELDS TERMINATED BY ','

STORED AS TEXTFILE;

➡️ **ALTER TABLE**

ALTER TABLE employee ADD COLUMNS (department STRING);

➡️ **DROP TABLE**

DROP TABLE employee;

➡️ **TRUNCATE TABLE**

TRUNCATE TABLE employee;

**✨ DML (Data Manipulation Language)**

Used to insert, load, and retrieve data.

➡️ **LOAD DATA**

LOAD DATA LOCAL INPATH '/user/hive/input/employee.csv' INTO TABLE employee;

➡️ **INSERT INTO**

INSERT INTO TABLE employee VALUES (1, 'Alice', 50000.0, 'IT');

➡️ **SELECT** (to read data)

SELECT \* FROM employee;

**✨ SELECT Query Examples**

✔️ Select specific columns:

SELECT name, salary FROM employee;

✔️ With WHERE condition:

SELECT \* FROM employee WHERE salary > 40000;

✔️ ORDER BY:

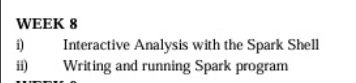
SELECT \* FROM employee ORDER BY name ASC;

✔️ GROUP BY with COUNT:

SELECT department, COUNT(\*) FROM employee GROUP BY department;

Great! Here's a **simple, handwritten-style note** for **WEEK 8** topic: Spark Shell & Spark Programs — perfect for quick revision before your exam 📘✨

**📝 WEEK 8 – Apache Spark**



**🔹 i) Interactive Analysis with the Spark Shell**

✅ **Spark Shell** is an interactive environment where we can type Spark code and see results immediately.

🚀 To start Spark shell (Scala):

$ spark-shell

📌 Inside Spark Shell:

val data = Array(10, 20, 30, 40)

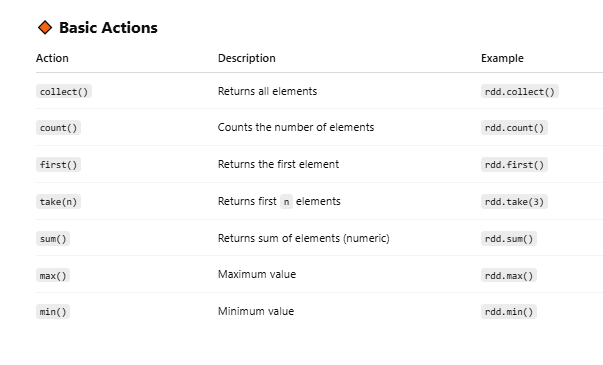
val rdd = sc.parallelize(data)

rdd.collect()

➡️ sc.parallelize() → converts data into RDD  
➡️ collect() → returns all elements from RDD

💡 You can perform actions like:

* rdd.count()
* rdd.sum()
* rdd.max()
* rdd.filter(x => x > 25).collect()





**🔹 ii) Writing and Running Spark Program**

✅ We can write Spark programs in:

* Scala
* Python
* Java

👉 Example in **Python (PySpark)**:

from pyspark import SparkContext

sc = SparkContext("local", "SimpleApp")

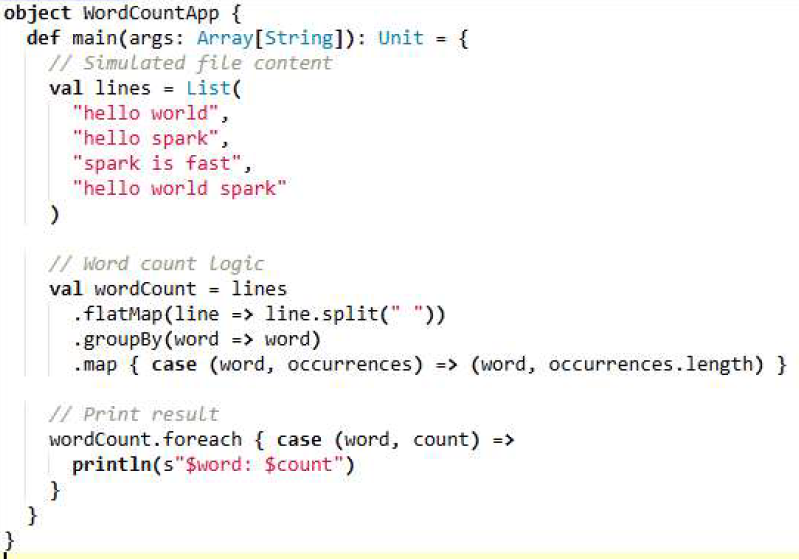
data = [1, 2, 3, 4, 5]

rdd = sc.parallelize(data)

print("Sum is:", rdd.sum())

➡️ Save this as: spark\_example.py  
➡️ Run using:

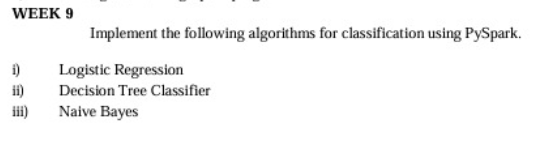
$ spark-submit spark\_example.py



**⭐ Key Spark Concepts:**

* **RDD** → Resilient Distributed Dataset
* **Actions** → collect(), count(), sum()
* **Transformations** → map(), filter()

✍️ **Week 9 – PySpark Classification Algorithms**



**Code :**

**from pyspark.sql import SparkSession**

**from pyspark.ml.feature import VectorAssembler**

**from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier, NaiveBayes**

**from pyspark.ml.evaluation import MulticlassClassificationEvaluator**

**spark = SparkSession.builder.appName("Week9\_Classification").getOrCreate()**

**data = [**

**(0.0, 1.0, 3.0, 0.0),**

**(1.0, 2.0, 1.0, 1.0),**

**(0.0, 2.0, 2.0, 0.0),**

**(1.0, 3.0, 3.0, 1.0)**

**]**

**columns = ["feature1", "feature2", "feature3", "label"]**

**df = spark.createDataFrame(data, columns)**

**assembler = VectorAssembler(inputCols=["feature1", "feature2", "feature3"], outputCol="features")**

**assembled\_df = assembler.transform(df)**

**final\_data = assembled\_df.select("features", "label")**

**lr = LogisticRegression(featuresCol='features', labelCol='label')**

**lr\_model = lr.fit(final\_data)**

**lr\_results = lr\_model.transform(final\_data)**

**lr\_results.select("features", "label", "prediction").show()**

**dt = DecisionTreeClassifier(featuresCol='features', labelCol='label')**

**dt\_model = dt.fit(final\_data)**

**dt\_results = dt\_model.transform(final\_data)**

**dt\_results.select("features", "label", "prediction").show()**

**nb = NaiveBayes(featuresCol='features', labelCol='label')**

**nb\_model = nb.fit(final\_data)**

**nb\_results = nb\_model.transform(final\_data)**

**nb\_results.select("features", "label", "prediction").show()**

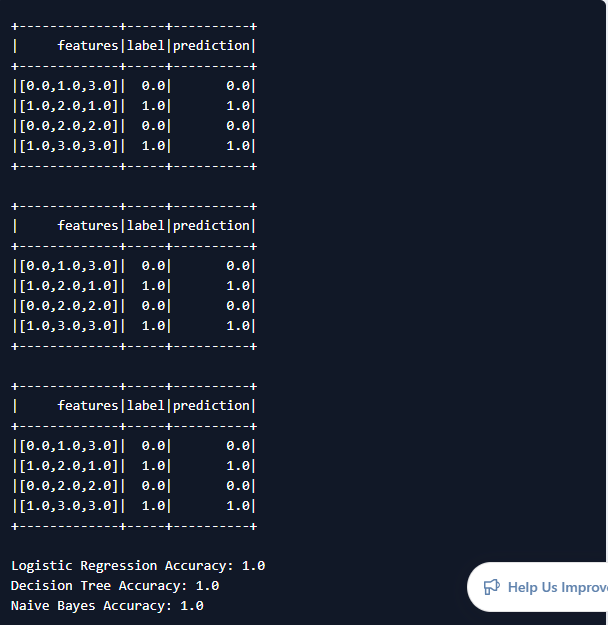
**evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction", metricName="accuracy")**

**print("Logistic Regression Accuracy:", evaluator.evaluate(lr\_results))**

**print("Decision Tree Accuracy:", evaluator.evaluate(dt\_results))**

**print("Naive Bayes Accuracy:", evaluator.evaluate(nb\_results))**

**results :**



**Week-11 : Implement collaborative filtering using spark ML library**

**Code :**

**from pyspark.sql import SparkSession**

**from pyspark.ml.recommendation import ALS**

**from pyspark.ml.evaluation import RegressionEvaluator**

**spark = SparkSession.builder.appName("OnlineALSExample").getOrCreate()**

**data = [**

**(0, 0, 4.0),**

**(0, 1, 2.0),**

**(1, 1, 3.0),**

**(1, 2, 5.0),**

**(2, 1, 4.0),**

**(2, 2, 3.0),**

**(3, 0, 2.0),**

**(3, 2, 4.0),**

**(4, 1, 5.0),**

**(4, 0, 3.0)**

**]**

**columns = ["userId", "movieId", "rating"]**

**ratings = spark.createDataFrame(data, columns)**

**(training, test) = ratings.randomSplit([0.8, 0.2], seed=42)**

**als = ALS(**

**maxIter=10,**

**regParam=0.1,**

**userCol="userId",**

**itemCol="movieId",**

**ratingCol="rating",**

**coldStartStrategy="drop"**

**)**

**model = als.fit(training)**

**predictions = model.transform(test)**

**predictions.show()**

**if predictions.count() > 0:**

**evaluator = RegressionEvaluator(**

**metricName="rmse",**

**labelCol="rating",**

**predictionCol="prediction"**

**)**

**rmse = evaluator.evaluate(predictions)**

**print(f"Root Mean Squared Error = {rmse:.2f}")**

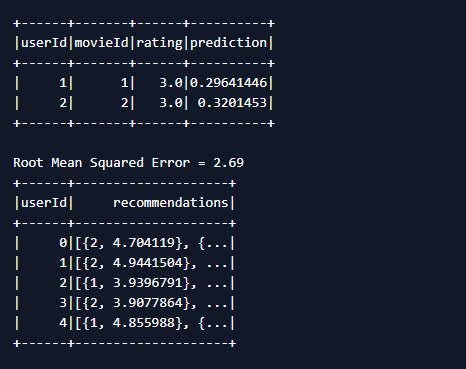
**else:**

**print("Test set is empty, skipping RMSE evaluation.")**

**userRecs = model.recommendForAllUsers(3)**

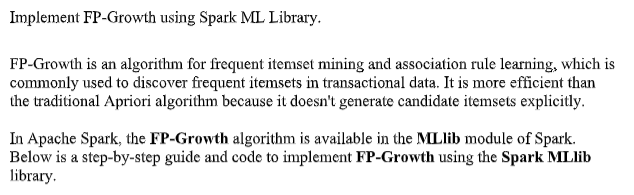
**userRecs.show()**

**spark.stop()**



**Week-12**

**Code:**



**Program:**

**from pyspark.sql import SparkSession**

**from pyspark.ml.fpm import FPGrowth**

**spark = SparkSession.builder.appName("OnlineFPGrowth").getOrCreate()**

**data = [**

**(0, ["milk", "bread", "butter"]),**

**(1, ["bread", "butter"]),**

**(2, ["milk", "bread"]),**

**(3, ["milk", "bread", "butter", "jam"]),**

**(4, ["bread", "butter"]),**

**(5, ["milk", "bread"]),**

**(6, ["jam", "bread"]),**

**]**

**columns = ["id", "items"]**

**df = spark.createDataFrame(data, columns)**

**fpGrowth = FPGrowth(itemsCol="items", minSupport=0.3, minConfidence=0.6)**

**model = fpGrowth.fit(df)**

**print("🔹 Frequent Itemsets:")**

**model.freqItemsets.show()**

**print("🔹 Association Rules:")**

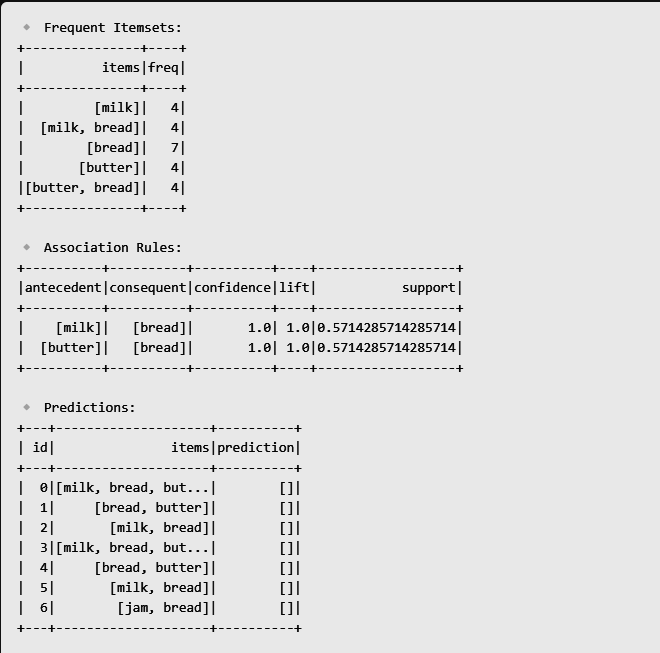
**model.associationRules.show()**

**print("🔹 Predictions:")**

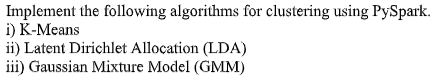
**model.transform(df).show()**

**spark.stop()**

**Output:**

****

**Week – 10**



**Program:**

**from pyspark.sql import SparkSession**

**from pyspark.ml.clustering import KMeans, GaussianMixture, LDA**

**from pyspark.ml.feature import VectorAssembler**

**from pyspark.ml.evaluation import ClusteringEvaluator**

**from pyspark.sql import Row**

**from pyspark.sql.functions import col**

**spark = SparkSession.builder.appName("ClusteringExample").getOrCreate()**

**data = [**

**Row(feature1=1.0, feature2=2.0), Row(feature1=1.5, feature2=1.8),**

**Row(feature1=5.0, feature2=8.0), Row(feature1=8.0, feature2=8.0),**

**Row(feature1=1.0, feature2=0.6), Row(feature1=9.0, feature2=11.0),**

**Row(feature1=8.0, feature2=9.0), Row(feature1=7.0, feature2=6.0),**

**Row(feature1=5.0, feature2=5.0), Row(feature1=6.0, feature2=4.0),**

**]**

**df = spark.createDataFrame(data)**

**print("Original Dataset:")**

**df.show()**

**assembler = VectorAssembler(inputCols=["feature1", "feature2"], outputCol="features")**

**assembled\_df = assembler.transform(df)**

**evaluator = ClusteringEvaluator(predictionCol="prediction", featuresCol="features")**

**train\_df = assembled\_df**

**test\_df = assembled\_df**

**print("\n=== K-Means Clustering ===")**

**kmeans = KMeans(k=2, featuresCol="features", predictionCol="prediction", seed=123)**

**kmeans\_model = kmeans.fit(train\_df)**

**kmeans\_predictions = kmeans\_model.transform(test\_df)**

**kmeans\_silhouette = evaluator.evaluate(kmeans\_predictions)**

**print(f"K-Means Silhouette Score: {kmeans\_silhouette:.4f}")**

**kmeans\_predictions.select("features", "prediction").show()**

**print("\n=== Gaussian Mixture Model ===")**

**gmm = GaussianMixture(k=2, featuresCol="features", predictionCol="prediction", seed=123)**

**gmm\_model = gmm.fit(train\_df)**

**gmm\_predictions = gmm\_model.transform(test\_df)**

**gmm\_silhouette = evaluator.evaluate(gmm\_predictions)**

**print(f"GMM Silhouette Score: {gmm\_silhouette:.4f}")**

**gmm\_predictions.select("features", "prediction").show()**

**print("\n=== LDA Topic Modeling ===")**

**lda\_data = assembled\_df.withColumn(**

**"feature1\_int", (col("feature1") \* 10).cast("integer") + 1**

**).withColumn(**

**"feature2\_int", (col("feature2") \* 10).cast("integer") + 1**

**)**

**lda\_assembler = VectorAssembler(inputCols=["feature1\_int", "feature2\_int"], outputCol="word\_counts")**

**lda\_prepared = lda\_assembler.transform(lda\_data)**

**lda = LDA(k=2, featuresCol="word\_counts", maxIter=10, seed=123)**

**lda\_model = lda.fit(lda\_prepared)**

**lda\_results = lda\_model.transform(lda\_prepared)**

**lda\_model.describeTopics(2).show(truncate=False)**

**lda\_results.select("word\_counts", "topicDistribution").show(truncate=False)**

**spark.stop()**

**Output:**

