1. Spark Setup

python

CopyEdit

import findspark

findspark.init()

- Makes Spark accessible in your Python environment (only needed in local setups or Colab).

---

python

CopyEdit

from pyspark.sql import SparkSession

from pyspark.ml.feature import VectorAssembler, StandardScaler, PCA, StringIndexer

from pyspark.ml.classification import LogisticRegression

from pyspark.ml.stat import Correlation

from pyspark.ml.evaluation import MulticlassClassificationEvaluator

import matplotlib.pyplot as plt

import pandas as pd

- Importing PySpark and visualization libraries.

---

python

CopyEdit

spark = SparkSession.builder.appName("Multivariate_Analysis_Iris").getOrCreate()

- Starts a Spark session named "Multivariate_Analysis_Iris".

📊 2. Load and Clean Data

python

CopyEdit

df = spark.read.csv("/content/Iris.csv", header=True, inferSchema=True)

- Loads the **Iris dataset** CSV file into a Spark DataFrame.
- inferSchema=True: Automatically infers data types.
- header=True: Uses the first row as column names.

---

python

CopyEdit

```
for old_name in df.columns:
   new_name = old_name.replace(".", "_").replace(" ", "_")
   df = df.withColumnRenamed(old_name, new_name)
df = df.withColumnRenamed(df.columns[-1], "label")
```

- Cleans column names by removing dots or spaces.
- Renames the last column (species name) to **label**.

🔲 3. Feature Engineering

python

CopyEdit

```
feature_cols = df.columns[:-1]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df = assembler.transform(df)
```

- Selects all columns **except** the label.
- Combines them into a single column called features.

📈 4. Correlation Matrix

python

CopyEdit

```
correlation_matrix = Correlation.corr(df, "features", method="pearson").head()[0]
print("Correlation Matrix:\n", correlation_matrix)
```

- Calculates the **Pearson correlation** between all numeric features.
- **Output** will be a 4x4 symmetric matrix showing how each feature correlates with others.

Example (partial output):

python-repl

CopyEdit

```
Correlation Matrix:
DenseMatrix([
[ 1.00,  0.73,  0.87,  0.81],
```

[0.73, 1.00, 0.82, 0.77],

...

])

📏 5. Standardization

python

CopyEdit

```
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withMean=True, withStd=True)

scaler_model = scaler.fit(df)

df = scaler_model.transform(df)
```

- Scales all features to **mean = 0** and **standard deviation = 1**.
- Essential before applying PCA.

🔬 6. Principal Component Analysis (PCA)

python

CopyEdit

```
pca = PCA(k=2, inputCol="scaled_features", outputCol="pca_features")

pca_model = pca.fit(df)

df = pca_model.transform(df)
```

- Applies PCA to reduce dimensionality from 4 features ➞ 2 principal components.
- The result is stored in a new column: pca_features.

🔤 7. Convert Labels to Numbers

python

CopyEdit

```
indexer = StringIndexer(inputCol="label", outputCol="indexedLabel")

df = indexer.fit(df).transform(df)
```

- Converts labels like Iris-setosa, Iris-versicolor to numbers: 0, 1, 2.
- Adds a new column indexedLabel.

---

8. Split Dataset

python

CopyEdit

```
train_df, test_df = df.randomSplit([0.8, 0.2], seed=1)
```

- Splits the data: **80% for training**, **20% for testing**.

🤖 9. Train Logistic Regression

python

CopyEdit

```
lr = LogisticRegression(featuresCol="pca_features", labelCol="indexedLabel", maxIter=100)
lr_model = lr.fit(train_df)
```

- Trains a **logistic regression classifier** using the two PCA components as input.

✅ 10. Make Predictions

python

CopyEdit

```
predictions = lr_model.transform(test_df)
```

- Applies the model to the test data to make predictions.

📏 11. Evaluate Model

python

CopyEdit

```
evaluator = MulticlassClassificationEvaluator(labelCol="indexedLabel", predictionCol="prediction", metricName="accuracy")
accuracy = evaluator.evaluate(predictions)
print(f"Model Accuracy: {accuracy:.2f}")
```

- Measures how accurate the model is.
- **Output example**:

yaml

CopyEdit

```
Model Accuracy: 0.97
```

📊 12. Confusion Matrix

python

CopyEdit

```
predictions.groupBy("indexedLabel", "prediction").count().show()
```

- Shows how many samples were correctly or incorrectly predicted.
- Example output:

diff

CopyEdit

```
+-------------+----------+-----+
|indexedLabel |prediction|count|
+-------------+----------+-----+
|0.0          |0.0       |10   |
|1.0          |1.0       |9    |
|2.0          |2.0       |10   |
+-------------+----------+-----+
```

🎨 13. PCA Visualization

python

CopyEdit

```
pandas_df = predictions.select("pca_features", "prediction").toPandas()
pandas_df["PCA1"] = pandas_df["pca_features"].apply(lambda x: x[0])
pandas_df["PCA2"] = pandas_df["pca_features"].apply(lambda x: x[1])
```

- Extracts the two PCA columns into a Pandas DataFrame for plotting.

---

python

CopyEdit

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(pandas_df["PCA1"], pandas_df["PCA2"], c=pandas_df["prediction"], cmap="Set1", alpha=0.7)
plt.xlabel("Principal Component 1")
```

plt.ylabel("Principal Component 2")

plt.title("Multivariate PCA Projection of Iris Classification")

plt.grid(True)

plt.colorbar(scatter)

plt.show()

- Plots a **2D scatter plot** of the test data using PCA results.
- Different colors represent different predicted classes.

🛑 14. Stop Spark Session

python

CopyEdit

spark.stop()

- Ends the Spark session.

📌 Summary of Outputs:

Step	Output	Description
Correlation Matrix	DenseMatrix	Shows how features are correlated
Accuracy	0.95 - 1.00 (typical)	How accurate the model is
Confusion Matrix	Table with counts	Actual vs Predicted labels
Scatter Plot	PCA 2D graph	Visual separation of classes

X-axis: Principal Component 1

- **Y-axis**: Principal Component 2
  These are new axes created by PCA that capture the most variance in the dataset.

🎨 Color Coding

- Each point is a data sample (a flower).
- Colors indicate the **class** of each flower (species of Iris):

    - Typically:
        - 0 = Iris-setosa (e.g., gray)
        - 1 = Iris-versicolor (e.g., orange)
        - 2 = Iris-virginica (e.g., red)
- The **color bar** on the right shows the label value range (0.00 to 2.00).

📊 Interpretation

- The clusters suggest that PCA has done a decent job of **separating the classes**.
- The red cluster (likely Iris-virginica) is well separated.
- The orange (likely Iris-versicolor) and gray (likely Iris-setosa) clusters show some overlap, indicating a bit more similarity.

✅ Purpose

- Helps in **visualizing the separability** of classes.
- Useful for **checking the effectiveness** of dimensionality reduction before applying classification algorithms.