✅ **STEP 1: Setup Spark and Import Libraries**

python

CopyEdit

```
import findspark
findspark.init()
```

- **findspark** helps us connect Jupyter/Colab/IDEs with PySpark.
- **findspark.init()** sets up the environment to use Spark.

python

CopyEdit

```
from pyspark.sql import SparkSession
from pyspark.ml.feature import VectorAssembler, StandardScaler, PCA
from pyspark.ml.clustering import KMeans
import matplotlib.pyplot as plt
```

- Importing:
  - **SparkSession**: For creating a Spark application.
  - **VectorAssembler**: Combines multiple columns into a single features column.
  - **StandardScaler**: Standardizes the data (mean = 0, std = 1).
  - **PCA**: Reduces the number of features (dimensionality).
  - **KMeans**: For clustering data.
  - **Matplotlib**: For plotting the PCA output.

✅ **STEP 2: Initialize Spark Session**

python

CopyEdit

```
spark = SparkSession.builder.appName("PCA_with_KMeans").getOrCreate()
```

- This starts a new Spark session named "PCA_with_KMeans".

✅ **STEP 3: Load Dataset**

python

CopyEdit

```
df = spark.read.csv("/content/segmentation data.csv", header=True, inferSchema=True)
```

- Loads a CSV file.
- **header=True**: Treat the first row as column headers.
- **inferSchema=True**: Automatically detect the data types.

✅ **STEP 4: Clean Column Names**

python

CopyEdit

```
for col_name in df.columns:
    cleaned_name = col_name.replace(".", "_").replace(" ", "_")
    df = df.withColumnRenamed(col_name, cleaned_name)
```

- Loops through all column names.
- Replaces . and spaces with _ to avoid errors while processing.

✅ **STEP 5: Assemble Features**

python

CopyEdit

```
feature_columns = df.columns[:-1]
vector_assembler = VectorAssembler(inputCols=feature_columns, outputCol="features")
df = vector_assembler.transform(df).select("features")
```

- Combines all feature columns into one column called "features" using VectorAssembler.
- [:-1] means we exclude the last column (possibly labels).
- We then select only this new column.

✅ **STEP 6: Standardize Features**

python

CopyEdit

```
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True, withMean=True)
scaler_model = scaler.fit(df)
df = scaler_model.transform(df).select("scaled_features")
```

- Scales the features:

- withStd=True → divide by std deviation.
- withMean=True → subtract mean.

- Fits the scaler and applies the transformation.

✅ **STEP 7: Apply PCA (Dimensionality Reduction)**

python

CopyEdit

```
pca = PCA(k=2, inputCol="scaled_features", outputCol="pca_features")
pca_model = pca.fit(df)
df = pca_model.transform(df).select("pca_features")
```

- Reduces the dataset to 2 dimensions (2 principal components).
- Output column is pca_features.
- This makes it easier to **visualize in 2D**.

✅ **STEP 8: Define Function to Run KMeans and Plot**

python

CopyEdit

```
def kmeans_with_pca():
    kmeans = KMeans(featuresCol="pca_features", k=3, seed=42)
    model = kmeans.fit(df)
    predictions = model.transform(df)
```

- Trains a KMeans clustering model with 3 clusters (k=3).
- Applies the model to the PCA-transformed data.

python

CopyEdit

```
    pandas_df = predictions.select("pca_features", "prediction").toPandas()
```

- Converts the Spark DataFrame to a Pandas DataFrame for visualization.

python

CopyEdit

```
    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 individual PCA components (x and y) for plotting.

python

CopyEdit

```
plt.figure(figsize=(8, 6))
scatter = plt.scatter(pandas_df["PCA1"], pandas_df["PCA2"], c=pandas_df["prediction"], cmap="viridis", alpha=0.7)
plt.colorbar(scatter, label="Cluster")
plt.xlabel("PCA Component 1")
plt.ylabel("PCA Component 2")
plt.title("PCA with KMeans Clustering (PySpark)")
plt.show()
```

- Plots the clusters using a scatter plot.
- Points are colored by their cluster label (0, 1, or 2).
- This helps us **visualize how KMeans grouped the data** after reducing it to 2D using PCA.