Cluster (Python)

Import notebook

Customer Segmentation and Sales Forecast

Big Data Analytics 2025
NOVA IMS MDSAA

[NOTE]

In this project, we are going to work on 3 notebooks:

- 1. Cleaning: For EDA and Data Preprocessing
- 2. Clustering: For clustering
- 3. Project Forecasting: For Sales Forecast.

This notebook is 2. Clustering.

Group 77

	Student Name	Student ID
1	Hassan Bhatti	20241023
2	Moeko Mitani	20240670
3	Oumayma Ben Hfaiedh	20240699
4	Ricardo Pereira	20240745

1. Data Integration

Import Libraries

```python
# ─────────────────────────────────────────────
# Spark Core
# ─────────────────────────────────────────────
from pyspark.sql import SparkSession
from pyspark.sql import functions as F
from pyspark.sql.functions import (
    col, lit, to_timestamp, to_date, year, month, dayofmonth,
    count, countDistinct, sum, avg, min, max,
    round, when, datediff, current_date, concat_ws,
    monotonically_increasing_id
)
from pyspark.sql.functions import max as spark_max

# ─────────────────────────────────────────────
# Spark MLlib
# ─────────────────────────────────────────────
from pyspark.ml.feature import (
    VectorAssembler, PCA, StringIndexer, StandardScaler,
    MinMaxScaler
)
from pyspark.ml.clustering import KMeans
from pyspark.ml.stat import Correlation
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.ml import Pipeline

# ─────────────────────────────────────────────
# Python Built-ins and Data Science Libraries
# ─────────────────────────────────────────────
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.cm as cm
from matplotlib.colors import ListedColormap
from sklearn.metrics import (
    silhouette_samples, silhouette_score,
    confusion_matrix
)

# ─────────────────────────────────────────────
# Utilities
# ─────────────────────────────────────────────
from itertools import combinations
```

```python
# Start Spark session
spark = SparkSession.builder.appName("Project_Group77").getOrCreate()
```

Import CSV File from Cleaning Notebook

;

```
# File location and type
file_location = "/FileStore/df_clustering.csv"
file_type = "csv"

# CSV options
infer_schema = "true"
first_row_is_header = "true"
delimiter = ","

# The applied options are for CSV files. For other file types, these will be ignored.
df_clustering = spark.read.format(file_type) \
  .option("inferSchema", infer_schema) \
  .option("header", first_row_is_header) \
  .option("sep", delimiter) \
  .load(file_location)

df_clustering.limit(10).display()
df_clustering.schema
```

▸ df_clustering: pyspark.sql.dataframe.DataFrame = [Customer ID: integer, num_invoices: integer ... 34 more fields]

Table

```
Out[3]: StructType([StructField('Customer ID', IntegerType(), True), StructField('num_invoices', IntegerType(), True), StructField('num_products', IntegerTyp
e(), True), StructField('total_quantity', IntegerType(), True), StructField('total_price', DoubleType(), True), StructField('avg_unit_price', DoubleType(), T
rue), StructField('first_purchase_date', DateType(), True), StructField('last_purchase_date', DateType(), True), StructField('purchase_span_days', IntegerTyp
e(), True), StructField('avg_quantity_per_invoice', DoubleType(), True), StructField('recency_days', IntegerType(), True), StructField('2022-12', IntegerType
(), True), StructField('2023-1', IntegerType(), True), StructField('2023-10', IntegerType(), True), StructField('2023-11', IntegerType(), True), StructField
('2023-12', IntegerType(), True), StructField('2023-2', IntegerType(), True), StructField('2023-3', IntegerType(), True), StructField('2023-4', IntegerType
(), True), StructField('2023-5', IntegerType(), True), StructField('2023-6', IntegerType(), True), StructField('2023-7', IntegerType(), True), StructField('2
023-8', IntegerType(), True), StructField('2023-9', IntegerType(), True), StructField('2024-1', IntegerType(), True), StructField('2024-10', IntegerType(), T
rue), StructField('2024-11', IntegerType(), True), StructField('2024-12', IntegerType(), True), StructField('2024-2', IntegerType(), True), StructField('2024
-3', IntegerType(), True), StructField('2024-4', IntegerType(), True), StructField('2024-5', IntegerType(), True), StructField('2024-6', IntegerType(), Tru
e), StructField('2024-7', IntegerType(), True), StructField('2024-8', IntegerType(), True), StructField('2024-9', IntegerType(), True)])
```

2. Clustering

2.1. Feature Selection

We are going to check the correlation between metric features to select the features for clustering.

```
# Define the list of metric features
new_metric_features =
['num_invoices','num_products','total_quantity','total_price','avg_unit_price','purchase_span_days','avg_quantity_per_invoice','recency_days','2022-
12','2023-1','2023-10','2023-11','2023-12','2023-2','2023-3','2023-4','2023-5','2023-6','2023-7','2023-8','2023-9','2024-1','2024-10','2024-
11','2024-12','2024-2','2024-3','2024-4','2024-5','2024-6','2024-7','2024-8','2024-9']

# VectorAssembler is a Spark ML tool that combines multiple columns into a single vector column
assembler = VectorAssembler(inputCols=new_metric_features, outputCol="features_vector")
df_vector = assembler.transform(df_clustering).select("features_vector") # Output

# Compute Spearman correlation matrix
correlation_result = Correlation.corr(df_vector, "features_vector", method="spearman").head()
corr_matrix = correlation_result[0].toArray() # toArray() converts Spark's matrix to a NumPy array

# Convert Spark correlation matrix to Pandas DataFrame
corr_df = pd.DataFrame(corr_matrix, index=new_metric_features, columns=new_metric_features)

# Create a mask to hide the upper triangle of the correlation matrix
mask = np.triu(np.ones_like(corr_df, dtype=bool))

# Plot heatmap using Seaborn & Matplotlib
plt.figure(figsize=(10, 8))
sns.heatmap(data=corr_df, mask=mask, annot=False, cmap='PRGn', cbar=True, square=False, linewidths=0.5)
plt.title('Correlation Heatmap', fontsize=20)
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.tight_layout()

# Display the plot in Databricks
plt.show()
```

▸ df_vector: pyspark.sql.dataframe.DataFrame

We are going to check the features highly correlated with threshold |0.8|.

```
# Extract pairs with strong correlations (|correlation| > 0.8)
strong_corr_pairs = []

for i, j in combinations(range(len(new_metric_features)), 2):
    corr_value = corr_df.iloc[i, j]
    if abs(corr_value) > 0.8:
        strong_corr_pairs.append((new_metric_features[i], new_metric_features[j], corr_value))

# Create a Pandas DataFrame from the results
strong_corr_df = pd.DataFrame(strong_corr_pairs, columns=["Feature 1", "Feature 2", "Correlation"])

# Sort by absolute correlation descending
strong_corr_df["AbsCorrelation"] = strong_corr_df["Correlation"].abs()
strong_corr_df.sort_values(by="AbsCorrelation", ascending=False, inplace=True)
strong_corr_df.drop(columns="AbsCorrelation", inplace=True)

# Display the result in Databricks
display(strong_corr_df)
```

Table

From the analysis above, we decided to drop 2 features: *purchase_span_days* and *total_quantity*
Moreover, we exclude time base features (e.g., 2022-12) for clustering. We will used them for profiling.

```
vb_features = ["Customer ID", "num_products", "num_invoices", "total_price", "avg_unit_price", "recency_days", "avg_quantity_per_invoice"]

df_vb = df_clustering.select(*vb_features)
df_vb.limit(10).display()
```

```
df_vb:  pyspark.sql.dataframe.DataFrame = [Customer ID: integer, num_products: integer ... 5 more fields]
```

Table

2.2. Clustering Using K-MEANS

We are going to use K-Means algorithm for clustering.
We are going to used MinMaxScaler to Scale the data since our data is not normally distributed. MinMaxScaler is useful when we want to ensure that all features have similar scales and are constrained within a certain range. This can improve the performance of algorithms that are sensitive to the scale of input features, such as gradient-based optimization algorithms.

```
# Start Spark session
spark = SparkSession.builder.appName("CustomerClustering").getOrCreate()

# Assemble all features into a single vector
# Exclude 'clientID' from the list of features
feature_cols = [col for col in df_vb.columns if col != "Customer ID"]
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features_unscaled")
assembled = assembler.transform(df_vb)

# Scale features using MinMaxScaler
scaler = MinMaxScaler(inputCol="features_unscaled", outputCol="features")
scaler_model = scaler.fit(assembled)
scaled_data = scaler_model.transform(assembled)
```

```
assembled:  pyspark.sql.dataframe.DataFrame
scaled_data:  pyspark.sql.dataframe.DataFrame
```

We are going to check how many clusters are best for our data by using elbow method.

```
cost = []
k_values = list(range(2, 11))

for k in k_values:
    kmeans = KMeans(featuresCol="features", k=k, seed=42)
    model = kmeans.fit(scaled_data)
    cost.append(model.summary.trainingCost)

# Plot the elbow curve
import matplotlib.pyplot as plt

plt.figure(figsize=(8, 5))
plt.plot(k_values, cost, marker="o", linestyle="--")
plt.xlabel("Number of Clusters (K)")
plt.ylabel("Cost (Within Set Sum of Squared Errors)")
plt.title("Elbow Method for Optimal K (Value-Based)")
plt.grid(True)
plt.tight_layout()
plt.show()
```

From the plot above, we are going to use K=3.

```
# Create clusters with k=3
kmeans = KMeans(featuresCol="features", k=3, seed=1)
model = kmeans.fit(scaled_data)
predictions = model.transform(scaled_data)

# Evaluate clustering
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print(f"Silhouette Score: {silhouette}")
```

▸ predictions: pyspark.sql.dataframe.DataFrame

```
Silhouette Score: 0.7685266695191887
```

```
# Show number of customers per cluster
predictions.groupBy("prediction").count().orderBy("prediction").show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|         0| 2109|
|         1| 4524|
|         2| 2810|
+----------+-----+
```

```
display(predictions)
```

Table

```
# Select only 'Customer ID' and 'prediction' from predictions
predictions_selected = predictions.select("Customer ID", "prediction")

# Perform the left join
df_final = df_clustering.join(predictions_selected, on="Customer ID", how="left")
df_final.limit(10).display()
```

▸ df_final: pyspark.sql.dataframe.DataFrame = [Customer ID: integer, num_invoices: integer ... 35 more fields]
▸ predictions_selected: pyspark.sql.dataframe.DataFrame = [Customer ID: integer, prediction: integer]

Table

3. Export CSV File

```
df_pandas = df_final.toPandas()
```

```
df_pandas.to_csv("/tmp/df_final.csv", index=False)
```

```
dbutils.fs.cp("file:/tmp/df_final.csv", "dbfs:/FileStore/df_final.csv")
```

```
Out[15]: True
```

4. Profiling

Now, we are going to profile each customer clusters.

4.1. Time Base Profiling

```
profiling_time = ['2022-12','2023-1','2023-10','2023-11','2023-12','2023-2','2023-3','2023-4','2023-5','2023-6','2023-7','2023-8','2023-9','2024-
1','2024-10','2024-11','2024-12','2024-2','2024-3','2024-4','2024-5','2024-6','2024-7','2024-8','2024-9']
```

```
# Group by 'prediction' and sum selected columns
agg_exprs = [F.sum(col).alias(col) for col in profiling_time]

df_time_cluster = df_final.groupBy('prediction').agg(*agg_exprs)
df_time_cluster.limit(10).display()
```

df_time_cluster: pyspark.sql.dataframe.DataFrame = [prediction: integer, 2022-12: long ... 24 more fields]

Table

Plot using Pandas

```
# Make a function
def plot_profiling_barplots_spark(df, merged_labels_col, features):
    # Aggregate in Spark
    # For each feature, sum the values grouped by the target label
    agg_exprs = [F.sum(col).alias(col) for col in features]
    grouped_pre_df = df.groupBy(merged_labels_col).agg(*agg_exprs)

    # Convert to Pandas
    grouped_df = grouped_pre_df.toPandas()

    # Plot using the same logic as before
    sns.set_style('white')

    # ensure the label column is in string format for consistent plotting
    if grouped_df[merged_labels_col].dtype == 'category' or not pd.api.types.is_numeric_dtype(grouped_df[merged_labels_col]):
        grouped_df[merged_labels_col] = grouped_df[merged_labels_col].astype(str)

    # Convert feature columns to numeric if necessary
    for feature in features:
        if not pd.api.types.is_numeric_dtype(grouped_df[feature]):
            grouped_df[feature] = pd.to_numeric(grouped_df[feature], errors='coerce')

    # Determine layout
    n_features = len(features)
    n_cols = 4
    n_rows = (n_features // n_cols) + (n_features % n_cols > 0)

    # Create subplot grid
    fig, axes = plt.subplots(n_rows, n_cols, figsize=(n_cols * 5, n_rows * 5))
    axes = axes.flatten()

    # Create bar plots for each feature
    for i, feature in enumerate(features):
        sns.barplot(x=merged_labels_col, y=feature, data=grouped_df, color='darkgreen', ax=axes[i])
        axes[i].set_title(f'{feature} by {merged_labels_col}')
        axes[i].set_xlabel(merged_labels_col)
        axes[i].set_ylabel('Counts')
        axes[i].tick_params(axis='x', rotation=90)

    for j in range(i + 1, len(axes)):
        axes[j].axis('off')

    plt.tight_layout()
    plt.show()
```

```
# Show the plot
plot_profiling_barplots_spark(df_final, 'prediction', profiling_time)
```

4.2. Behavior Base Profiling

```
profiling_metric =
['num_invoices','num_products','total_quantity','total_price','avg_unit_price','purchase_span_days','avg_quantity_per_invoice','recency_days']
```

```
# Create aggregation expressions to sum each selected profiling metric
# This generates a list of expressions like: F.sum('metric1').alias('metric1'), ...
agg_exprs = [F.sum(col).alias(col) for col in profiling_metric]

# Group the final DataFrame by the predicted cluster label
# Then, aggregate each profiling metric by summing its values within each cluster
df_metric_cluster = df_final.groupBy('prediction').agg(*agg_exprs)
```

```
df_metric_cluster:  pyspark.sql.dataframe.DataFrame = [prediction: integer, num_invoices: long ... 7 more fields]
```

```
display(df_metric_cluster)
```

Table

```
# Call the custom function to generate barplots for profiling metrics per cluster

# Parameters:
# - df_final: The Spark DataFrame containing the final data with predictions
# - 'prediction': The name of the column used for grouping (typically cluster labels)
# - profiling_metric: A list of feature names (columns) to sum and visualize per group

# Show the plot
plot_profiling_barplots_spark(df_final, 'prediction', profiling_metric)
```

4.3. Customer Profiling

Cluster 0: Inactive or Lost Customers

Size: 2,109 customers

Key Characteristics:
- Lowest in invoices, products, quantity, revenue, and average unit price.
- Very long recency – they haven't purchased in a long time.
- Short customer lifetime (Purchase Span Days).
- Lowest average quantity per invoice.
- Most recent purchases occurred mainly in 2022-2023 summer.
- No purchases at all in late 2023 or 2024.

Profile Summary:

These are inactive or churned customers. They used to engage at a low level but have now completely stopped purchasing. This group may have switched to competitors, closed their business, or found less value in our offerings.

Actionable Strategy:
- Restart Campaigns: Special reactivation offers or targeted email campaigns.
- Feedback collection: Try to learn why they stopped ordering.
- Consider removing them from active targeting if unresponsive after multiple campaigns.

Cluster 1: High-Value Loyal Customers

Size: 4,524 customers

Key Characteristics:
- Highest across all transactional metrics: invoices, products, quantity, revenue.
- Consistent across all months and years, highly active and reliable.
- Frequent purchases with large average quantities per invoice.
- Recent and ongoing engagement.
- Long customer lifetime (long span days).

Profile Summary:

These customers are our top-tier wholesale customers who are loyal, high-frequency, and high-value buyers. They show strong ongoing engagement and form the backbone of our business.

Actionable Strategy:
- Loyalty Programs: Reward frequent buyers such as discounts, early access, bulk deals.
- Develop recommendation system: Tailor product recommendations.

Cluster 2: Occasional or Premium Buyers

Size: 2,810 customers

Key Characteristics:
- Middle in invoice count, product variety, quantity, and revenue.
- Highest average unit price which means they buy fewer, but more expensive items.
- Somewhat consistent but limited engagement through 2023 and early 2024, disappeared the middle of 2024.
- Moderate recency tells that they are not fully inactive, but not as recent as Cluster 1.
- Average quantity per invoice is moderate.

Profile Summary:

These are likely small boutique resellers or premium buyers. Occasional purchases, preference for luxury and selected gift items, and small, high-margin orders. Their pattern shows selective engagement.

Actionable Strategy:
- Premium Product Bundles: Offer curated or seasonal premium boxes.
- Targeted product launches (high-margin or exclusive items).
- Encourage regular use with small loyalty rewards or "limited time" offers.

Cluster	Size	Activity Level	Customer Type	Recency	Spending	Strategy
0	2,109	Inactive	Churned/Lost	High	Low	Restart Campaign, survey
1	4,524	Active	High-value/Loyal	Low	High	Loyalty, upsell
2	2,810	Moderate	Occasional/Premium	Medium	Medium	Premium-focused