# Name: Sai Sravani Madabhushi

# Z number: Z23752172

```
from pyspark.sql import SparkSession
from pyspark.sql.functions import col, median, lit
from pyspark.ml.feature import VectorAssembler, StandardScaler
from pyspark.ml.regression import LinearRegression
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import RegressionEvaluator, ClusteringEvaluator
import networkx as nx
import matplotlib.pyplot as plt
import numpy as np

# Initializing Spark Session
spark = SparkSession.builder \
    .appName("SpotifyNetworkAnalysis") \
    .getOrCreate()

print("Spark Session Initialized")
```

```
Spark Session Initialized
```

1. Load the Dataset using PySpark (2 points)

o Start the Spark session and load the dataset using PySpark. Ensure that Spark is properly initialized before loading the data. You will need to do some exploratory analysis for the written report.

```
from pyspark.sql.functions import col
from pyspark.sql.types import DoubleType
from pyspark.sql.functions import mean

#  QUESTION 1: LOAD DATASET

# Define file paths
nodes_path = "nodes.csv"
edges_path = "edges.csv"

# Load the datasets
df_nodes = spark.read.csv(nodes_path, header=True, inferSchema=True)
df_edges = spark.read.csv(edges_path, header=True, inferSchema=True)

# PREPROCESSING
#  Handle Missing Values
df_nodes = df_nodes.na.drop(subset=["spotify_id", "name"])
df_edges = df_edges.na.drop(subset=["id_0", "id_1"])

# Fill missing numerical values (e.g., popularity, followers) with 0
df_nodes = df_nodes.na.fill(0, subset=["popularity", "followers"])

# Cast 'followers' column to DoubleType
df_nodes = df_nodes.withColumn("followers", col("followers").cast(DoubleType()))

# Show the schema and count to verify loading
print("Nodes Schema:")
df_nodes.printSchema()
print(f"Total Nodes: {df_nodes.count()}")
print(f"Total Edges: {df_edges.count()}")

# Calculate and print the average for followers and popularity
print("Averages")
df_nodes.select(mean("followers"), mean("popularity")).show()
```

```
Nodes Schema:
root
 |-- spotify_id: string (nullable = true)
 |-- name: string (nullable = true)
 |-- followers: double (nullable = true)
 |-- popularity: double (nullable = false)
 |-- genres: string (nullable = true)
 |-- chart_hits: string (nullable = true)

Total Nodes: 156422
Total Edges: 300386
Averages
+-----------------+------------------+
|   avg(followers)|   avg(popularity)|
+-----------------+------------------+
|86224.26419762558|21.157497027272377|
+-----------------+------------------+
```

2. Create the Graph using NetworkX (2 points)

o Construct the graph using NetworkX, where each node represents a user, and edges represent collaboration relationships.

```
#  QUESTION 2:  CREATE GRAPH

#  Create the full graph in NetworkX
full_edge_list = df_edges.select("id_0", "id_1").rdd.map(lambda row: (row[0], row[1])).collect()

G_full = nx.Graph()
G_full.add_edges_from(full_edge_list)
print(f"Full Graph Created. Nodes: {G_full.number_of_nodes()}, Edges: {G_full.number_of_edges()}")

# Extract Subsample using Breadth-First Search (BFS)
def bfs_sample(graph, start_node, target_size):
    sampled_nodes = set()
    queue = [start_node]
    sampled_nodes.add(start_node)

    while len(sampled_nodes) < target_size and queue:
        current_node = queue.pop(0)
        neighbors = list(graph.neighbors(current_node))

        for neighbor in neighbors:
            if neighbor not in sampled_nodes:
                sampled_nodes.add(neighbor)
                queue.append(neighbor)
                if len(sampled_nodes) >= target_size:
                    break
    # Create the subgraph from the nodes
    return graph.subgraph(sampled_nodes).copy()
```

```
Full Graph Created. Nodes: 153327, Edges: 300386
```

3. Visualize the Graph (or a Subset) using NetworkX

IN CASE YOU ARE WORKING WITH A GRAPH SAMPLE – ALL THE PROPERTIES AND THE REST OF THE INSTRUCTIONS SHOULD BE EXECUTED FOR THE SUBSAMPLE GRAPH ONLY

▪ Here are some suggestions on approaches to extract a graph subsample. Please use one of the following:

▪ Random walk

▪ Breadth first search method

▪ Forest fire sampling

▪ DO NOT EXTRACT A RANDOM SAMPLE OF NODES/EDGES, THIS DOES NOT PRESERVE GRAPH TOPOLOGY!

▪ IF YOU USE A CODE REFERENCE FOR SAMPLING IMPLENETATION, MAKE SURE TO CITE IT! IF NOT PROPERLY CITED, YOUR WORK WILL BE REPORTED AS A VIOLATION OF ACADEMIC INTEGRITY!

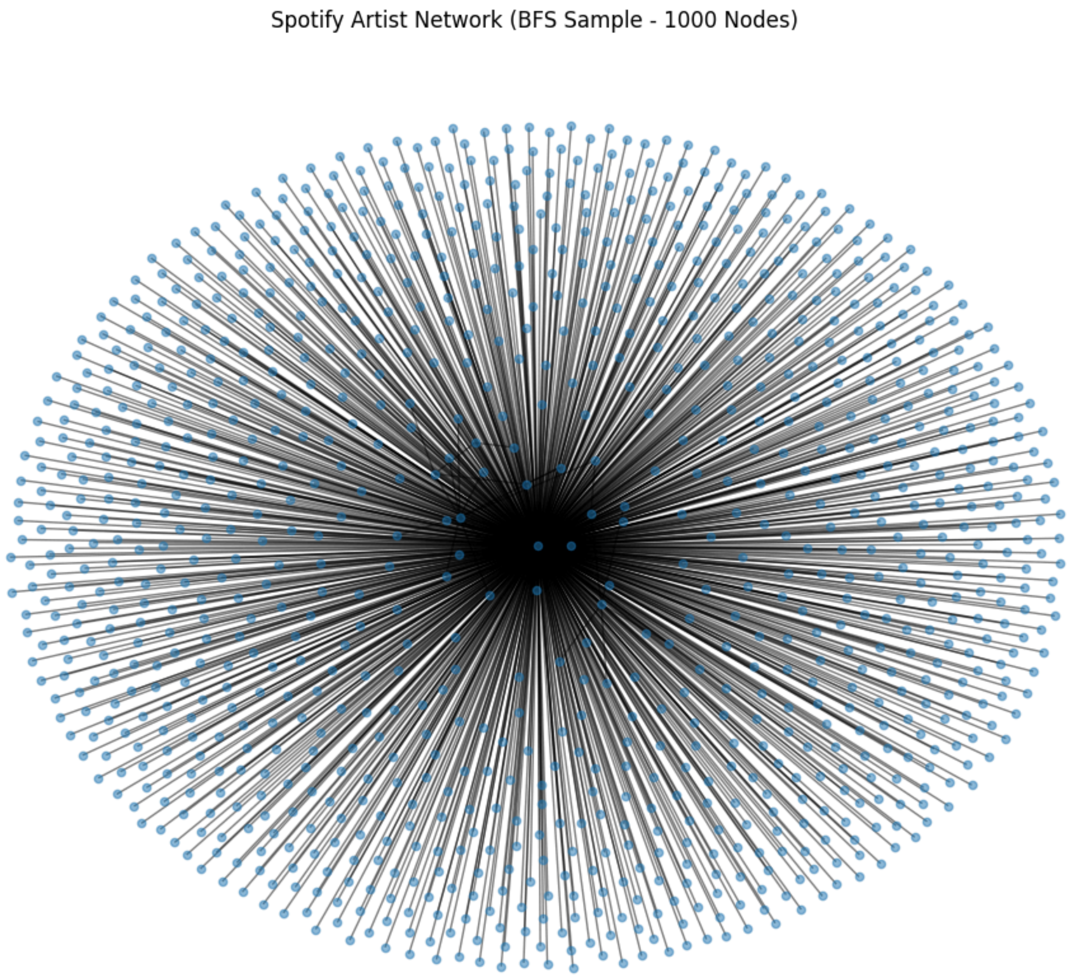
o Visualize the graph (or a small subset) using NetworkX's plotting functions to get an understanding of the network's structure. (0.5 points)

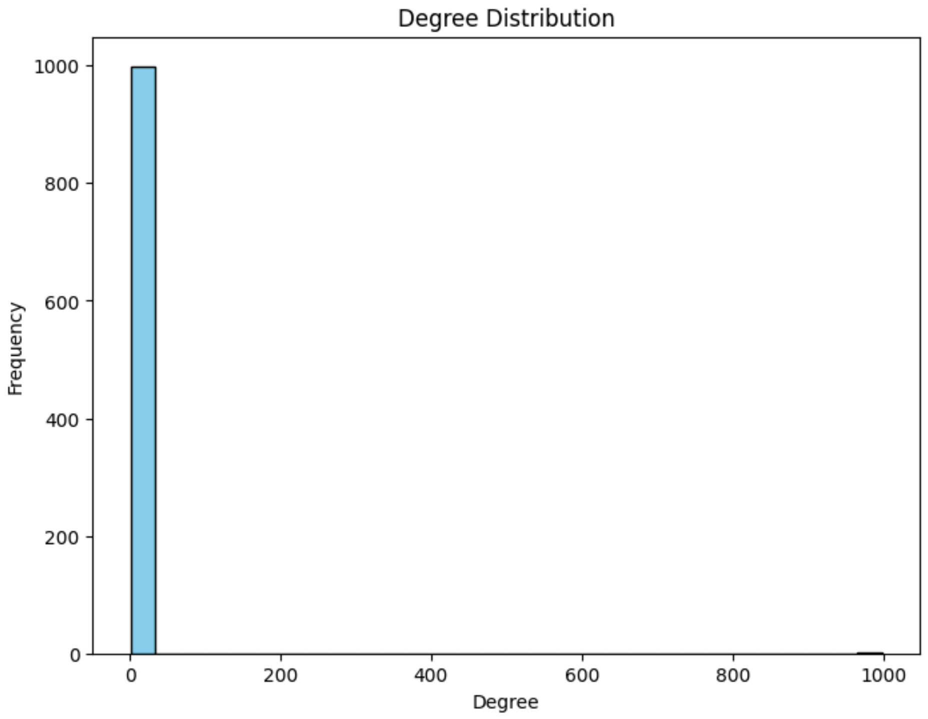
o Plot the degree distribution. (0.5 points)

```
#  QUESTION 3:  VISUALIZE GRAPH
# Pick a start node
start_node = sorted(G_full.degree, key=lambda x: x[1], reverse=True)[0][0]
TARGET_SIZE = 1000

G_sample = bfs_sample(G_full, start_node, TARGET_SIZE)
print(f"BFS Sample Graph Created. Nodes: {G_sample.number_of_nodes()}, Edges: {G_sample.number_of_edges()}")
```

```
# Visualize the Subsample
plt.figure(figsize=(10, 8))
pos = nx.spring_layout(G_sample, seed=42)
nx.draw(G_sample, pos, node_size=20, edge_color="black", alpha=0.5, with_labels=False)
plt.title(f"Spotify Artist Network (BFS Sample - {TARGET_SIZE} Nodes)")
plt.show()

#  Plot Degree Distribution
degrees = [d for n, d in G_sample.degree()]
plt.figure(figsize=(8, 6))
plt.hist(degrees, bins=30, color='skyblue', edgecolor='black')
plt.title("Degree Distribution")
plt.xlabel("Degree")
plt.ylabel("Frequency")
plt.show()
```

```
BFS Sample Graph Created. Nodes: 1000, Edges: 1023
```





4. Compute Network Properties

o For the report, you will also need network diameter, number of connected components, and edge density.

o Compute Centrality Measures (5 points)

▪ Calculate the following centrality measures for each node using NetworkX:

▪ Degree centrality

▪ Betweenness centrality

▪ Closeness centrality

▪ Eigenvector centrality

▪ PageRank

o Top 5 Most Influential Artists (2 points)

▪ Identify and print the top 5 most influential artists based on each centrality measure (i.e., the nodes with the highest values for each centrality measure). Make sure to print their name, not ID!

```
# QUESTION 4: NETWORK PROPERTIES & CENTRALITY


#  General Properties
print("Graph Properties for Sample Graph:")
print(f"Number of Nodes: {G_sample.number_of_nodes()}")
print(f"Number of Edges: {G_sample.number_of_edges()}")

# Diameter is calculated on the largest connected component
if nx.is_connected(G_sample):
    print(f"Network Diameter: {nx.diameter(G_sample)}")
else:
    largest_cc_nodes = max(nx.connected_components(G_sample), key=len)
    largest_cc = G_sample.subgraph(largest_cc_nodes)
    print(f"Network Diameter (Largest Component): {nx.diameter(largest_cc)}")

print(f"Number of Connected Components: {nx.number_connected_components(G_sample)}")
print(f"Edge Density: {nx.density(G_sample):.5f}")
avg_degree = 2 * G_sample.number_of_edges() / G_sample.number_of_nodes()
print(f"Average Degree: {avg_degree}")

#  Compute Centrality Measures

degree_centrality = nx.degree_centrality(G_sample)
betweenness_centrality = nx.betweenness_centrality(G_sample)
closeness_centrality = nx.closeness_centrality(G_sample)
eigenvector_centrality = nx.eigenvector_centrality(G_sample, max_iter=1000)
page_rank = nx.pagerank(G_sample)

# Top 5 Most Influential Artists
sample_ids = list(G_sample.nodes())
df_sample_names = df_nodes.filter(col("spotify_id").isin(sample_ids)).select("spotify_id", "name")
# Collect as a dict: {id: name}
id_to_name = {row['spotify_id']: row['name'] for row in df_sample_names.collect()}

def print_top_5(centrality_dict, metric_name):
    # Sort by score descending
    sorted_nodes = sorted(centrality_dict.items(), key=lambda x: x[1], reverse=True)[:5]
    print(f"\nTop 5 for {metric_name}:")
    for node_id, score in sorted_nodes:
        name = id_to_name.get(node_id, "Unknown")
        print(f"  {name}  : {score:.4f}")

print_top_5(degree_centrality, "Degree Centrality")
print_top_5(betweenness_centrality, "Betweenness Centrality")
print_top_5(closeness_centrality, "Closeness Centrality")
print_top_5(eigenvector_centrality, "Eigenvector Centrality")
print_top_5(page_rank, "PageRank")
```

```
Graph Properties for Sample Graph:
Number of Nodes: 1000
Number of Edges: 1023
Network Diameter: 2
Number of Connected Components: 1
Edge Density: 0.00205
Average Degree: 2.046
```

```
Top 5 for Degree Centrality:
  Johann Sebastian Bach  : 1.0000
  John Williams  : 0.0120
  Franz Xaver Gruber  : 0.0060
  Ludwig van Beethoven  : 0.0050
  Fazıl Say  : 0.0040

Top 5 for Betweenness Centrality:
  Johann Sebastian Bach  : 0.9999
  John Williams  : 0.0001
  Franz Xaver Gruber  : 0.0000
  Ludwig van Beethoven  : 0.0000
  Fazıl Say  : 0.0000

Top 5 for Closeness Centrality:
  Johann Sebastian Bach  : 1.0000
  John Williams  : 0.5030
  Franz Xaver Gruber  : 0.5015
  Ludwig van Beethoven  : 0.5013
  Fazıl Say  : 0.5010

Top 5 for Eigenvector Centrality:
  Johann Sebastian Bach  : 0.7067
  John Williams  : 0.0306
  Franz Xaver Gruber  : 0.0260
  Ludwig van Beethoven  : 0.0256
  Fazıl Say  : 0.0247

Top 5 for PageRank:
  Johann Sebastian Bach  : 0.4507
  John Williams  : 0.0044
  Franz Xaver Gruber  : 0.0023
  Ludwig van Beethoven  : 0.0018
  Fazıl Say  : 0.0015
```

5. Predicting User Popularity Using PySpark

o Linear Regression Model (5 points)

▪ Use the number of followers and the centrality measures (from step 4) as features in a Linear Regression model to predict user popularity (e.g., number of streams or followers). You'll need to create a Spark DataFrame with these features.

▪ Split the data into training and testing sets.

o Print Model Coefficients and Intercept (2 points)

▪ After training the model, print out the coefficients and intercept of the regression model.

o Evaluate Model Performance (2 points)

▪ Evaluate the model performance using RMSE (Root Mean Squared Error) and R² (coefficient of determination).

```
# QUESTION 5: PREDICTING USER POPULARITY (LINEAR REGRESSION)

data_list = []
for node in G_sample.nodes():
    data_list.append((
        node,
        float(degree_centrality[node]),
        float(betweenness_centrality[node]),
        float(closeness_centrality[node]),
        float(eigenvector_centrality[node]),
        float(page_rank[node])
    ))

# Schema for the centrality data
schema = ["spotify_id", "degree_centrality", "betweenness_centrality", "closeness_centrality", "eigenvector_centrality", "page_
df_centrality = spark.createDataFrame(data_list, schema=schema)

df_model_data = df_nodes.join(df_centrality, on="spotify_id", how="inner")

# Feature Engineering

feature_cols = ["followers", "degree_centrality", "betweenness_centrality", "closeness_centrality", "eigenvector_centrality", "
assembler = VectorAssembler(inputCols=feature_cols, outputCol="features")
df_final = assembler.transform(df_model_data)

#  Split Data
train_data, test_data = df_final.randomSplit([0.8, 0.2], seed=42)
```

```
# Train Model
lr = LinearRegression(featuresCol="features", labelCol="popularity")
lr_model = lr.fit(train_data)

#  Print Coefficients and Intercept
print("\n Model Coefficients ")
print(f"Intercept: {lr_model.intercept:.4f}")
for i, col_name in enumerate(feature_cols):
    print(f"{col_name}: {lr_model.coefficients[i]:.4f}")

#  Evaluate Performance
predictions = lr_model.transform(test_data)
evaluator = RegressionEvaluator(labelCol="popularity", predictionCol="prediction", metricName="rmse")
rmse = evaluator.evaluate(predictions)
r2 = evaluator.setMetricName("r2").evaluate(predictions)

print("\nEvaluation")
print(f"RMSE: {rmse:.4f}")
print(f"R2 Score: {r2:.4f}")
```

```
 Model Coefficients
Intercept: 8119099.2596
followers: 0.0000
degree_centrality: 4047468.9859
betweenness_centrality: 4071318.3230
closeness_centrality: -16242541.3835
eigenvector_centrality: 102151.5128
page_rank: -148598.6235

Evaluation
RMSE: 16.2564
R2 Score: 0.0026
```

6. Cluster Users Based on Their Characteristics Using PySpark

o Perform K-Means Clustering (2 points)

▪ Standardize the features

▪ Use PySpark's K-Means algorithm to cluster users based on the features you used for training the regression model (number of followers and centrality measures).

o Find the Optimal Number of Clusters (2 points)

▪ Determine the optimal number of clusters.

```
# QUESTION 6: CLUSTER USERS (K-MEANS)

# Standardize Features
scaler = StandardScaler(inputCol="features", outputCol="scaled_features", withStd=True, withMean=True)
scaler_model = scaler.fit(df_final)
df_clust = scaler_model.transform(df_final)

# Find Optimal Number of Clusters
print(" Finding Optimal K (2 to 10) ")
silhouette_scores = []
k_values = range(2, 11)

for k in k_values:
    kmeans = KMeans(featuresCol="scaled_features", k=k, seed=1)
    model = kmeans.fit(df_clust)
    predictions = model.transform(df_clust)

    # Evaluate with Silhouette Score
    evaluator = ClusteringEvaluator(featuresCol="scaled_features")
    score = evaluator.evaluate(predictions)
    silhouette_scores.append(score)
    print(f"K={k}: Silhouette Score = {score:.4f}")

# Select best K
best_k = k_values[np.argmax(silhouette_scores)]
print(f"\nOptimal K based on Silhouette: {best_k}")

#  Perform Final Clustering
final_kmeans = KMeans(featuresCol="scaled_features", k=best_k, seed=1)
final_model = final_kmeans.fit(df_clust)
final_predictions = final_model.transform(df_clust)
```

```
# Show a few results
final_predictions.select("name", "prediction").show(10)
```

```
 Finding Optimal K (2 to 10)
K=2: Silhouette Score = 0.9988
K=3: Silhouette Score = 0.9977
K=4: Silhouette Score = 0.9960
K=5: Silhouette Score = 0.9893
K=6: Silhouette Score = 0.9709
K=7: Silhouette Score = 0.9243
K=8: Silhouette Score = 0.8849
K=9: Silhouette Score = 0.8275
K=10: Silhouette Score = 0.7197

Optimal K based on Silhouette: 2
+--------------------+----------+
|                name|prediction|
+--------------------+----------+
|Michel Garcin-Marrou|         0|
|Wolfgang Schneide...|         0|
|  Christian Brückner|         0|
|     Martha Schuster|         0|
|   Daniel Lozakovich|         0|
|       Benjamin Bayl|         0|
|         Hakurō Mōri|         0|
|       Anton Dermota|         0|
|Deutsche Bachsoli...|         0|
|The Glenn Gould S...|         0|
+--------------------+----------+
only showing top 10 rows
```

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit

Double-click (or enter) to edit