Project Title: Using Apache Spark, develop a K-means algorithm to classify NBA player's records into 4 Comfortable zones

Task Description:
By analyzing the NBA shot log data (**Data**), we need to perform the following task:

GAME_ID	MATCHUP	LOCATION	W	FINAL_MARGIN
21.4m – 21.4m	1808 unique values	A 50% H 50%	W 50% L 50%	-53 – 53
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24
21400899	MAR 04, 2015 - CHA @ BKN	A	W	24

Task : For each player, we define the comfortable zone of shooting is a matrix of,

[SHOT DIST, CLOSE DEF DIST, SHOT CLOCK]

Develop a K-means algorithm to classify each player's records into 4 Comfortable zones. And identify best zone for James Harden, Chris Paul, Stephen Curry and Lebron James.

Dataset Description: The NBA dataset contains 21 Columns (e.g., player_name, CLOSE_DEF_DIST) and 128,069 rows (all players information). Missing values are dropped using specific code functions. Sample dataset is shown in Fig.1, which shows shots taken during the 2014-2015 season, who took the shot, where on the floor was the shot taken from, who was the nearest defender, how far away was the nearest defender, time on the shot clock, and much more. The column titles are generally self-explanatory.

GAME_ID	MATCHUP	LOCATION	W	FINAL_MA	SHOT_NUM	PERIOD	GAME_CLC	SHOT_CLO	DRIBBLES	TOUCH_TI	SHOT_DIST	PTS_TYPE	SHOT_RESI	CLOSEST_D	CLOSEST_D	CLOSE_DEF_DIST	PTS	player_nar	player_id
21400899	MAR 04, 2(	A	W	24	1	1	01:09	10.8	2	1.9	7.7	2	made	Anderson,	101187	1.3	1	2 brian rober	203148
21400899	MAR 04, 2(	A	W	24	2	1	00:14	3.4	0	0.8	28.2	3	missed	Bogdanovi	202711	6.1	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	3	1	00:00		3	2.7	10.1	2	missed	Bogdanovi	202711	0.9	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	4	2	11:47	10.3	2	1.9	17.2	2	missed	Brown, Ma	203900	3.4	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	5	2	10:34	10.9	2	2.7	3.7	2	missed	Young, Tha	201152	1.1	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	6	2	08:15	9.1	2	4.4	18.4	2	missed	Williams, D	101114	2.6	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	7	4	10:15	14.5	11	9	20.7	2	missed	Jack, Jarret	101127	6.1	0	0 brian rober	203148
21400899	MAR 04, 2(	A	W	24	8	4	08:00	3.4	3	2.5	3.5	2	made	Plumlee, M	203486	2.1	1	2 brian rober	203148

Fig.1 NBA Sample data set (missing values are dropped)

Data of Interest Matrix: A matrix of [SHOT_DIST, CLOSE_DEF_DIST, SHOT_CLOCK] is created from the sample dataset to identify the four comfortable zones of all players (Fig.2).

player_name	CLOSEST_DEFENDER	SHOT_RESULT	CLOSE_DEF_DIST	SHOT_DIST	SHOT_CLOCK
brian roberts	Anderson, Alan	made	1.3	7.7	10.8
brian roberts	Bogdanovic, Bojan	missed	6.1	28.2	3.4
brian roberts	Brown, Markel	missed	3.4	17.2	10.3
brian roberts	Young, Thaddeus	missed	1.1	3.7	10.9
brian roberts	Williams, Deron	missed	2.6	18.4	9.1
brian roberts	Jack, Jarrett	missed	6.1	20.7	14.5
brian roberts	Plumlee, Mason	made	2.1	3.5	3.4
brian roberts	Morris, Darius	missed	7.3	24.6	12.4
brian roberts	Ellington, Wayne	missed	19.8	22.4	17.4
brian roberts	Lin, Jeremy	missed	4.7	24.5	16
brian roberts	Lin, Jeremy	made	1.8	14.6	12.1
brian roberts	Hill, Jordan	made	5.4	5.9	4.3
brian roberts	Green, Willie	missed	4.4	26.4	4.4
brian roberts	Smart, Marcus	missed	5.3	22.8	6.8
brian roberts	Young, James	made	5.6	24.7	6.4
brian roberts	Jerebko, Jonas	missed	5.4	25	17.6
brian roberts	Crowder, Jae	missed	5.1	25.6	8.7
brian roberts	Thomas, Isaiah	made	11.1	24.2	20.8
brian roberts	Brooks, Aaron	missed	3.5	25.4	17.5

Fig. 2 Data of Interest (only 20 representative rows are shown)

Tools:

- **Distributed Processing Engine:** Apache Spark
- **Cloud Platform:** Google Cloud Platform (integrated with 3-node clusters and Jupyter notebook)
- Github, Gitbash, Bash Scripting

Process Design:

The entire process consists of following steps (shown in Fig.3).

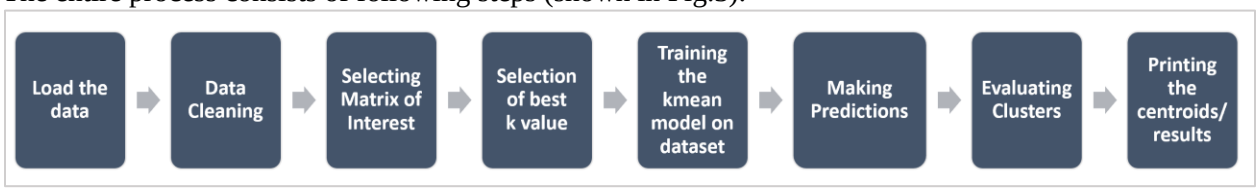

Fig.3 Design Steps

i) *Data Preprocessing:* After loading the dataset, player wise matrix of interest is selected. The matrix contains no null values.

Matrix of Interest: [SHOT_DIST, CLOSE_DEF_DIST, SHOT_CLOCK]

ii) *Exploration of k value:* It is important to select the best value of 'k' before training kmean model on any dataset. In this example, at first Silhouette with squared Euclidean distance (SED) is determined for all the k values (2 to 7) shown in Fig. 4 and then one best k value is selected where homogeneity in the dataset is high. A graph between k values and SED is plotted to select the best k value and identify the homogenous data zones.

```
With K=2
Silhouette with squared euclidean distance = 0.7749790418883465
With K=3
Silhouette with squared euclidean distance = 0.6857632125279227
With K=4
Silhouette with squared euclidean distance = 0.6068364199429537
With K=5
Silhouette with squared euclidean distance = 0.5593931268060487
With K=6
Silhouette with squared euclidean distance = 0.575362923277882
With K=7
Silhouette with squared euclidean distance = 0.5015644786009681
```

Fig.4 Silhouette with squared euclidean distance for k ranging from 2 to 7

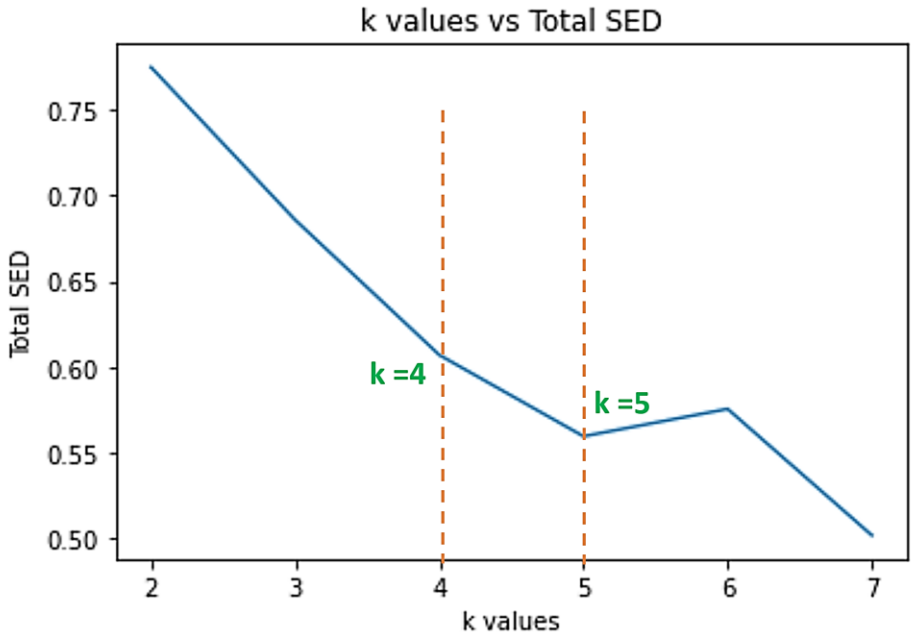

Fig. 5 Graph between Silhouette with squared euclidean distance and k values ranging from 2 to 7. Either k=4 or k=5 would represent the highest homogenity in the dataset, so any one value could be chosen.

iii) *Model Training and Making Predictions:* The k means model is trained at k=4, and comfortable zones of all players are predicted based on this value. The clusters are evaluated by computing Silhouette score for each player.

iv) *Printing final output:* Cluster centers for each player is generated as output.

NBA K-means Code:

```
from __future__ import print_function

from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
from pyspark.sql.functions import concat_ws
from pyspark.sql.types import StructField,IntegerType
from pyspark.ml.feature import VectorAssembler
from pyspark.sql import SparkSession, SQLContext
from pyspark.sql.types import StructType
from pyspark.sql.types import DoubleType
from pyspark.sql.types import*

import pandas as pd
import matplotlib.pyplot as plt

spark = SparkSession\
        .builder\
        .appName("KMeansExample")\
        .getOrCreate()
```

```
df = spark.read.load(sys.argv[1],format="csv", sep=",", inferSchema="true", heade
r="true")

#LOAD DATA
dataset = spark.read.format("libsvm").load("/mapreduce-test/pro_1/shot_logs.csv")
df = df.drop(columns=['GAME_ID','SHOT_RESULT','MATCHUP','LOCATION','W','FINAL_MAR
GIN','SHOT_NUMBER','PERIOD','GAME_CLOCK','DRIBBLES','CLOSEST_DEFENDER','CLOSEST_D
EFENDER_PLAYER_ID','FGM','PTS','player_id','PTS_TYPE','TOUCH_TIME'])
df = df[df.SHOT_RESULT != 'missed']
df = df.groupby('player_name')
df = df.mean()

#FILTER DATA TO GROUP BY PLAYER
training_set = x = df[['SHOT_CLOCK','SHOT_DIST','CLOSE_DEF_DIST']]

#EXPLORATION OF KVALUES

#Convert dataset into VectorRow data cells
data_of_interest = dataset.withColumn('CLOSE_DEF_DIST', data_cleaned['CLOSE_DEF_D
IST'].cast(DoubleType())).withColumn('SHOT_DIST', data_cleaned['SHOT_DIST, '].cas
t(DoubleType())).withColumn('SHOT_CLOCK', data_cleaned['SHOT_CLOCK'].cast(DoubleT
ype()))
feature_vector = VectorAssembler(inputCols=['CLOSE_DEF_DIST', 'SHOT_DIST', 'SHOT_
CLOCK'], outputCol="features")
transform_data = feature_vector.transform(data_of_interest)
player_names = transform_data.select("player_name").distinct().collect()
list_items = list()
evaluator = ClusteringEvaluator()

#Getting Silhouette with squared euclidean distance for k value ranging from 2 to
 8
TotalSED = []
for player in player_name:
    features = transform_data.where(transform_data["player_name"] == player[0]).s
elect("features")
    for k in range(2,8):
        kmeans = KMeans(featuresCol = 'features', k=k)
        model = kmeans.fit(features)
        predictions = model.transform(features)
        silhouette = evaluator.evaluate(predictions)
        print("With K={}".format(k))
        print("Silhouette with squared euclidean distance = " + str(silhouette))
        TotalSED.append(silhouette)
    break
```

```
#plotting kvalues and Total_SED
plt.plt(range(2,9), TSED); plt.xlabel("No_of_Clusters"); plt.ylabel("Total_SED");
 plt.xticks(k)


#ESTABLISH MODEL WITH KMEANS
kmeans = KMeans().setK(4).setSeed(1)
model = kmeans.fit(training_set)

# Make predictions
predictions = model.transform(training_set)

# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(predictions)
print("Silhouette with squared euclidean distance = " + str(silhouette))

# Shows the result.
centers = model.clusterCenters()

def e_dist(stat_matrix,cent_list):
    mat_dist = []
    for i in range(0,len(centers)):
        dist_sqrd = 0
        for j in range(0,len(centers[i])):
            dist_sqrd += (float(stat_matrix[j]) - float(cent_list[i][j]))**2

        dist = math.sqrt(dist_sqrd)
        mat_dist.append(dist)

spark.stop()
```

test.sh file

```
#!/bin/bash
source ../../env.sh
/usr/local/hadoop/bin/hdfs dfs -rm -r /kmeans/input/
/usr/local/hadoop/bin/hdfs dfs -rm -r /kmeans/output/
/usr/local/hadoop/bin/hdfs dfs -mkdir -p /kmeans/input/
/usr/local/hadoop/bin/hdfs dfs -copyFromLocal ../../test-
data/shotlogs.txt /nba/input/
/usr/local/spark/bin/spark-submit --
master=spark://$SPARK_MASTER:7077 ./kmeans.py hdfs://$SPARK_MASTER:9000/kmeans/in
put/
```

Final Output :

Comfortable Zone of four players (based on their success)

```
lebron james [12.90796976 12.88713801  4.38209669]
chris paul [12.74732632 17.01261435  4.98419107]
james harden [12.90796976 12.88713801  4.38209669]
stephen curry [12.74732632 17.01261435  4.98419107]
```

Appendix:

Zones of all players (just for information)

```
Silhouette with squared euclidean distance = 8.6868364199429537
['james johnson', [array([2.60735294, 4.80735294, 5.78735294]), array([ 3.12857143,  3.77272727, 20.34025974]), ar
ray([ 6.97758621, 21.82413793,  9.43103448]), array([ 2.38712871,  4.59287921, 12.3968396 ])]]
['danny green', [array([ 6.53414634, 23.46439024, 17.84829268]), array([2.80724638, 7.83333333, 7.7884058 ]), arra
y([ 2.72318841,  3.12753623, 19.47536232]), array([ 5.49666667, 23.54166667,  7.67333333])]]
['mason plumlee', [array([ 3.09210526,  3.76491228, 21.69912281]), array([2.44044118, 4.3125    , 6.76176471]), ar
ray([45.3,  5.4, 24. ]), array([ 2.52446843,  4.24748281, 14.01726619])]]
['donald sloan', [array([ 4.17866667, 22.85866667,  4.976     ]), array([ 2.37529412,  4.94941176, 17.13529412]),
array([2.40686061, 6.49393939, 7.82878788]), array([ 4.68815789, 21.76710526, 14.29736842])]]
['eric bledsoe', [array([ 5.84162679, 21.81674641, 17.3277512 ]), array([ 3.78363636,  3.48681818, 19.17545455]),
array([ 5.32810458, 21.49346405,  7.45947712]), array([2.84444444, 4.89691358, 8.56728395])]]
['david west', [array([2.47891156, 7.05442177, 8.96734694]), array([ 5.1511811 , 18.16141732,  5.96771654]), array
([ 2.29574468,  3.23617021, 20.64688851]), array([ 6.19709302, 19.22267442, 13.88430233])]]
['shabazz napier', [array([ 4.44545455, 21.78545455,  3.83818182]), array([ 5.79459459, 22.58648649, 16.7351351
4]), array([ 3.14583333,  4.89791667, 11.81875   ]), array([ 5.26111111, 23.22407407, 10.07777778])]]
['danilo gallinai', [array([ 6.9 , 34.35, 23.5 ]), array([ 4.43431373, 23.43039216,  6.63921569]), array([ 2.80769
231,  6.82967033, 13.18989011]), array([ 5.83153846, 23.88923077, 15.51615385])]]
['jared dudley', [array([14.655, 22.365, 14.765]), array([ 5.78383838, 21.00888081,  7.23636364]), array([ 3.14556
962,  4.78607595, 14.82658228]), array([ 6.07842254, 22.36971831, 16.15985915])]]
['andre iguodala', [array([ 6.38041237, 21.65257732, 15.31958763]), array([ 3.60864198,  3.99876543, 19.5456790
1]), array([ 5.68880808, 22.47171717,  6.65454545]), array([2.88980392, 6.31372549, 8.61176471])]]
['kawhi leonard', [array([ 5.72283465, 22.10787402,  7.81417323]), array([ 3.81078431,  4.61478588, 18.97941176]),
array([2.93741007, 7.93597122, 8.97913669]), array([ 5.37387692, 19.87      , 16.11238769])]]
['beno urdih', [array([ 4.57117117, 16.83873874, 15.17837838]), array([3.856, 9.32 , 6.206]), array([ 3.19056604,
2.7490566 , 14.95849057]), array([ 5.41651376, 21.59541284,  8.80366972])]]
['dirk nowtizski', [array([ 2.47171717,  5.19393939, 11.64646465]), array([ 3.57932331, 14.28195489, 10.1863989
8]), array([ 5.12832086, 20.99411765,  8.58938481]), array([ 5.82489627, 22.55228216, 16.81991701])]]
['elfrid payton', [array([2.52866667, 5.63533333, 7.72666667]), array([ 6.525     , 18.875     , 14.61041667]), ar
ray([ 4.51578947, 18.22105263,  4.71578947]), array([ 2.67489091,  4.70989091, 17.96363636])]]
['ramon sessions', [array([ 2.51678832,  6.51021898, 13.39788029]), array([ 6.1    , 21.74375,  5.38125]), array([
5.96190476, 19.66666667, 16.31428571]), array([ 6.80833333, 24.75416667, 12.17916667])]]
['marreese speights', [array([ 6.4511811 , 18.47559055, 15.53543307]), array([ 2.43671875,  3.81953125, 20.1554687
5]), array([2.34867257, 4.79646018, 9.91150442]), array([ 5.21351351, 17.97927928,  7.98828829])]]
['matt barnes', [array([ 7.39534884, 23.51395349, 16.38372093]), array([ 3.21630435,  4.10217391, 19.55188696]), a
rray([ 6.3       , 24.82592593,  7.59555556]), array([2.49759036, 5.62771084, 8.19156627])]]
['damian lillard', [array([ 5.18849673, 24.25915033, 16.65898039]), array([ 3.16891892,  4.2981982 , 17.7153153
2]), array([ 4.71404959, 23.86983471,  8.20286612]), array([2.84774194, 5.9516129 , 8.44774194])]]
['taj gibson', [array([ 4.80204882, 14.89795918,  9.27857143]), array([ 1.88271685,  2.88395062, 23.88888889]), ar
ray([2.16581197, 4.56324786, 6.43247863]), array([ 2.83364486,  3.82856075, 13.83738318])]]
['rudy gay', [array([2.49337748, 8.89172185, 7.74503311]), array([ 5.08417722, 21.5028481 , 11.61178886]), array([
2.58336134,  6.54369748, 17.49957983]), array([21.78,  5.14, 21.38])]]
['udonis haslem', [array([ 5.52592593, 17.22962963,  4.71481481]), array([ 5.99677419, 17.49032258, 12.8       ]),
array([ 2.31111111,  2.55925926, 21.31481481]), array([1.94897959, 2.80612245, 7.45306122])]]
['marcin gortat', [array([ 2.75121951,  3.21382114, 28.85609756]), array([ 5.89833333, 14.98416667, 10.31666667]),
array([ 2.7816092 ,  5.51264368, 13.8637931 ]), array([2.84100719, 6.74316547, 6.88647482])]]
```

Only selected output is shown in this figure