Name: Mayuri Birari

Assignment 02

## Initial Setup:

1. Python 3.x installed on your system
2. PySpark Python package installed using – pip install pyspark

## Question 1 Part 1:

Importing libraries and reading data

```python
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
```

```python
spark=SparkSession.builder.appName('Parking_ticket_analysis').getOrCreate()
```

```python
data=spark.read.csv('C:/Users/mayur/OneDrive/Desktop/CoursesSem2/ECC/A02/Parking_Violations_Issued_-_Fiscal_Year_2023 (1).csv',header=True,inferSchema=True)
```

```python
data.show(10)
```

Python

```
+--------------+--------+------------------+----------+----------+--------------+-----------------+-------------+--------------+------------+------------+------------+------------------
|Summons Number|Plate ID|Registration State|Plate Type|Issue Date|Violation Code|Vehicle Body Type|Vehicle Make|Issuing Agency|Street Code1|Street Code2|Street Code3|Vehicle Expiration
+--------------+--------+------------------+----------+----------+--------------+-----------------+-------------+--------------+------------+------------+------------+------------------
|    1484697303| JER1863|                NY|       PAS|06/10/2022|            67|              SDN|        TOYOT|             P|       34330|         179|           0|              202
|    1484697315| KEV4487|                NY|       PAS|06/13/2022|            51|             SUBN|         JEEP|             K|       34310|       16400|       11010|              202
|    1484697625|  H73NYD|                NJ|       PAS|06/19/2022|            63|              SDN|         JEEP|             N|       30640|       13015|       28540|
|    1484697674| GJC9296|                NY|       PAS|06/19/2022|            63|             SUBN|        LEXUS|             N|       30640|       13015|       28540|              202
|    1484697686|  M51PUV|                NJ|       PAS|06/19/2022|            63|              SDN|        HYUND|             N|       30640|       13015|       28540|
|    1484697698|  H73NYD|                NJ|       PAS|06/23/2022|            63|             null|         JEEP|             K|       30640|       13015|       28540|
|    1484697728|  M51PUV|                NJ|       PAS|06/23/2022|            63|             null|        HYUND|             K|       30640|       13015|       28540|
|    1484698204| KJJ8637|                NY|       PAS|06/20/2022|            67|             SUBN|        TOYOT|             N|       11585|       26390|       15010|              202
|    1484698381| JEC8631|                NY|       PAS|06/19/2022|            98|             SUBN|        NISSA|             N|           0|           0|           0|              202
|    1484698721|  K21PNH|                NJ|       PAS|06/25/2022|            10|             SUBN|         JEEP|             K|       33190|       25190|       31990|
+--------------+--------+------------------+----------+----------+--------------+-----------------+-------------+--------------+------------+------------+------------+------------------
```

1: **When are tickets most likely to be issued?**
First I grouped by issue_date and count the number of tickets issued on each day
Then, I ordered the result in descending order of ticket count to get the most ticketed days

```python
#When are tickets most likely to be issued?
most_ticketed_days = data.groupBy('Issue Date').count().orderBy(desc('count')).take(10)
most_ticketed_days
```

```
[Row(Issue Date='08/04/2022', count=66726),
 Row(Issue Date='08/05/2022', count=65393),
 Row(Issue Date='08/02/2022', count=64876),
 Row(Issue Date='06/30/2022', count=64846),
 Row(Issue Date='07/19/2022', count=64815),
 Row(Issue Date='11/25/2022', count=64411),
 Row(Issue Date='08/11/2022', count=64192),
 Row(Issue Date='08/18/2022', count=63975),
 Row(Issue Date='07/12/2022', count=63780),
 Row(Issue Date='07/15/2022', count=63646)]
```

**Ans: Tickets are most likely to be issued on 08/04/2022**

**2: What are the most common years and types of cars to be ticketed?**

I grouped by vehicle_year and count the number of tickets issued for each year. I then grouped by vehicle_make to get the count of tickets for each car make. Then, I ordered the result in descending order of ticket count to get the most ticketed years and car makes

```
# What are the most common years and types of cars to be ticketed?

most_ticketed_years = data.groupBy('Vehicle Year').count().orderBy(desc('count')).take(10)
most_ticketed_car_makes = data.groupBy('Vehicle Make').count().orderBy(desc('count')).take(10)
```

```
most_ticketed_car_makes
```

```
[Row(Vehicle Make='HONDA', count=1394250),
 Row(Vehicle Make='TOYOT', count=1333368),
 Row(Vehicle Make='FORD', count=1045269),
 Row(Vehicle Make='NISSA', count=954362),
 Row(Vehicle Make='CHEVR', count=610002),
 Row(Vehicle Make='ME/BE', count=594653),
 Row(Vehicle Make='BMW', count=583146),
 Row(Vehicle Make='JEEP', count=533310),
 Row(Vehicle Make='HYUND', count=392388),
 Row(Vehicle Make='LEXUS', count=293765)]
```

Ans: The types of cars to be ticketed is Honda and the common years of cars to be ticketed is 2021

**3: Where are tickets most commonly issued?**

I grouped by violation_precinct and count the number of tickets issued for each precinct. Then, I ordered the result in descending order of ticket count to get the most ticketed precincts

```
# Where are tickets most commonly issued?

most_ticketed_precincts = data.groupBy('Violation Precinct').count().orderBy(desc('count')).take(10)
most_ticketed_precincts
```

```
[Row(Violation Precinct=0, count=5349526),
 Row(Violation Precinct=19, count=282466),
 Row(Violation Precinct=13, count=254057),
 Row(Violation Precinct=6, count=224686),
 Row(Violation Precinct=114, count=221523),
 Row(Violation Precinct=14, count=190012),
 Row(Violation Precinct=18, count=176733),
 Row(Violation Precinct=9, count=162228),
 Row(Violation Precinct=1, count=152429),
 Row(Violation Precinct=109, count=137833)]
```

```
Ans: Precinct where tickets are most commonly issued is Precinct 0
```

Ans: Precinct where tickets are most commonly issued is Precinct 0

**4: Which color of the vehicle is most likely to get a ticket?**

I grouped by vehicle_color and count the number of tickets issued for each color. Then, ordered the result in descending order of ticket count to get the most ticketed colors.

```python
# Which color of the vehicle is most likely to get a ticket?

most_ticketed_colors = data.groupBy('Vehicle Color').count().orderBy(desc('count')).take(10)
most_ticketed_colors
```

```
[Row(Vehicle Color='GY', count=2275457),
 Row(Vehicle Color='WH', count=2055818),
 Row(Vehicle Color='BK', count=1992788),
 Row(Vehicle Color=None, count=1032007),
 Row(Vehicle Color='BL', count=760235),
 Row(Vehicle Color='WHITE', count=671757),
 Row(Vehicle Color='RD', count=435989),
 Row(Vehicle Color='BLACK', count=424056),
 Row(Vehicle Color='GREY', count=308993),
 Row(Vehicle Color='SILVE', count=151063)]
```

Ans: The color of the vehicle is most likely to get a ticket is Gray

### Question 1 Part 2:

### Importing libraries and reading data

```python
from pyspark.sql.functions import col
from pyspark.sql import SparkSession
from pyspark.sql.functions import *
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
import pyspark.sql.functions as F
```

```python
spark=SparkSession.builder.appName('Parking_ticket_analysis').getOrCreate()
```

```python
# Load the dataset
parking_df = spark.read.format("csv").option("header", "true").load("C:/Users/mayur/OneDrive/Desktop/CoursesSem2/ECC/A02/Parking_Violations_Issued_-_Fiscal_Year_2023 (1).csv")
```

```python
parking_df.show()
```

| Summons Number | Plate ID | Registration State | Plate Type | Issue Date | Violation Code | Vehicle Body Type | Vehicle Make | Issuing Agency | Street Code1 | Street Code2 | Street Code3 | Vehicle Expiration |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1484697303 | JER1863 | NY | PAS | 06/10/2022 | 67 | SDN | TOYOT | P | 34330 | 179 | 0 | 202 |
| 1484697315 | KEV4487 | NY | PAS | 06/13/2022 | 51 | SUBN | JEEP | K | 34310 | 16400 | 11010 | 202 |
| 1484697625 | H73NYD | NJ | PAS | 06/19/2022 | 63 | SDN | JEEP | N | 30640 | 13015 | 28540 | |
| 1484697674 | GJC9296 | NY | PAS | 06/19/2022 | 63 | SUBN | LEXUS | N | 30640 | 13015 | 28540 | 202 |
| 1484697686 | M51PUV | NJ | PAS | 06/19/2022 | 63 | SDN | HYUND | N | 30640 | 13015 | 28540 | |
| 1484697698 | H73NYD | NJ | PAS | 06/23/2022 | 63 | null | JEEP | K | 30640 | 13015 | 28540 | |
| 1484697728 | M51PUV | NJ | PAS | 06/23/2022 | 63 | null | HYUND | K | 30640 | 13015 | 28540 | |
| 1484698204 | KJJ8637 | NY | PAS | 06/20/2022 | 67 | SUBN | TOYOT | N | 11585 | 26390 | 15010 | 202 |
| 1484698381 | JEC8631 | NY | PAS | 06/19/2022 | 98 | SUBN | NISSA | N | 0 | 0 | 0 | 202 |
| 1484698721 | K21PNH | NJ | PAS | 06/25/2022 | 10 | SUBN | JEEP | K | 33190 | 25190 | 31990 | |
| 1484698769 | LVK1404 | PA | PAS | 06/05/2022 | 10 | SUBN | NISSA | P | 33190 | 25190 | 31990 | |
| 1484699683 | KSX6366 | FL | PAS | 06/25/2022 | 51 | SUBN | SUBAR | K | 30640 | 24050 | 0 | 202 |
| 1484699750 | GCX5397 | NY | PAS | 06/19/2023 | 63 | SUBN | CHEVR | N | 30640 | 13015 | 28540 | 202 |
| 1484703261 | KUH5328 | NY | PAS | 06/09/2022 | 45 | SDN | NISSA | K | 0 | 0 | 0 | 202 |
| 1484710629 | KUH1765 | NY | OMS | 06/30/2022 | 14 | SDN | TOYOT | K | 33340 | 0 | 0 | 202 |
| 1484717909 | HJJ9998 | NY | PAS | 06/10/2022 | 20 | SUBN | HONDA | K | 34440 | 0 | 0 | 202 |
| 1484720581 | GHR524 | FL | PAS | 07/03/2022 | 68 | VAN | CHRYS | K | 0 | 0 | 0 | |
| 1484720600 | G15PTU | NJ | PAS | 07/03/2022 | 68 | SUBN | CHEVR | K | 0 | 0 | 0 | |
| 1484720829 | JLJ6406 | NY | PAS | 06/18/2022 | 27 | SDN | NISSA | K | 11210 | 22695 | 0 | 202 |
| 1484721317 | KNZ4744 | NY | PAS | 06/29/2022 | 20 | SUBN | ME/BE | K | 23904 | 25680 | 21950 | 202 |

## Filtered for black vehicles parked at the specified street codes

```python
# Filter for black vehicles parked at the specified street codes
black_vehicles_list = ['BLAC','Black','BK','BLK.','BK/','BLK','BC','B LAC','BK.','BCK','BLACK']
filtered_df = parking_df.filter((col("Vehicle Color").isin(black_vehicles_list ) ))
filtered_df.show()
```

Python

| Summons Number | Plate ID | Registration State | Plate Type | Issue Date | Violation Code | Vehicle Body Type | Vehicle Make | Issuing Agency | Street Code1 | Street Code2 | Street Code3 | Vehicle Expiration |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1484697303 | JER1863 | NY | PAS | 06/10/2022 | 67 | SDN | TOYOT | P | 34330 | 179 | 0 | 2022 |
| 1484698381 | JEC8631 | NY | PAS | 06/19/2022 | 98 | SUBN | NISSA | N | 0 | 0 | 0 | 2023 |
| 1484703261 | KUH5328 | NY | PAS | 06/09/2022 | 45 | SDN | NISSA | K | 0 | 0 | 0 | 2024 |
| 1484710629 | KUH1765 | NY | OMS | 06/30/2022 | 14 | SDN | TOYOT | K | 33340 | 0 | 0 | 2022 |
| 1484720829 | JLJ6406 | NY | PAS | 06/18/2022 | 27 | SDN | NISSA | K | 11210 | 22695 | 0 | 2022 |
| 1484721809 | GDM5641 | NY | PAS | 07/04/2022 | 20 | SDN | LEXUS | K | 0 | 0 | 0 | 2023 |
| 1484725955 | KSG1672 | NY | PAS | 06/15/2022 | 27 | SDN | VOLKS | K | 0 | 13820 | 0 | 2023 |
| 1484726364 | JHJ2036 | NY | PAS | 06/17/2022 | 14 | SUBN | ROVER | K | 0 | 0 | 0 | 2023 |
| 1484726844 | HSZ5866 | NY | PAS | 06/04/2022 | 68 | SUBN | BMW | K | 13820 | 0 | 0 | 2022 |
| 1484741780 | KEX2744 | NY | PAS | 06/30/2022 | 20 | SUBN | HONDA | N | 29520 | 12335 | 0 | 2022 |
| 1484751632 | LYD2422 | PA | PAS | 06/18/2022 | 20 | null | SUBAR | K | 9020 | 0 | 0 | |
| 1484751759 | KDT8949 | NY | PAS | 06/11/2022 | 14 | SDN | BMW | K | 22320 | 22425 | 0 | 2026 |
| 1484752272 | KSZ7897 | NY | PAS | 06/06/2022 | 14 | SDN | LEXUS | K | 36420 | 58870 | 0 | 2023 |
| 1484752752 | KKD1739 | NY | PAS | 06/10/2022 | 20 | SUBN | TOYOT | K | 22620 | 22425 | 0 | 2023 |
| 1484763269 | GZJ6596 | NY | PAS | 06/11/2022 | 20 | SUBN | FORD | K | 29090 | 24940 | 0 | 2023 |
| 1484767329 | KNIGHT | CA | PAS | 06/22/2022 | 51 | MOTO | null | P | 53950 | 49630 | 0 | |
| 1484768127 | KKE4605 | NY | PAS | 07/03/2022 | 14 | SUBN | HONDA | P | 28430 | 25370 | 23830 | 2023 |
| 1484768668 | GD2556 | NY | PAS | 06/23/2022 | 17 | SUBN | NISSA | P | 86530 | 86730 | 28430 | 2022 |
| 1484772192 | HRF7476 | NY | PAS | 06/11/2022 | 98 | SDN | BMW | P | 72230 | 77730 | 51030 | 2023 |
| 1484772209 | JPB6390 | NY | PAS | 07/02/2022 | 14 | SUBN | MITSU | P | 28430 | 61010 | 25370 | 2024 |

```python
filtered_df = filtered_df.select(filtered_df['Street Code1'],filtered_df['Street Code2'],filtered_df['Street Code3'])
filtered_df.show()
```

| Street Code1 | Street Code2 | Street Code3 |
|---|---|---|
| 34330 | 179 | 0 |
| 0 | 0 | 0 |
| 0 | 0 | 0 |
| 33340 | 0 | 0 |
| 11210 | 22695 | 0 |
| 0 | 0 | 0 |
| 0 | 13820 | 0 |
| 0 | 0 | 0 |
| 13820 | 0 | 0 |
| 29520 | 12335 | 0 |
| 9020 | 0 | 0 |
| 22320 | 22425 | 0 |
| 36420 | 58870 | 0 |
| 22620 | 22425 | 0 |
| 29090 | 24940 | 0 |
| 53950 | 49630 | 0 |
| 28430 | 25370 | 23830 |
| 86530 | 86730 | 28430 |
| 72230 | 77730 | 51030 |
| 28430 | 61010 | 25370 |

## Selecting the columns for clustering

```
# Select the columns for clustering
cluster_df = filtered_df.select(col('Street Code1').cast('int'),col('Street Code2').cast('int'),col('Street Code3').cast('int'))
cluster_df.show()
```

```
+------------+------------+------------+
|Street Code1|Street Code2|Street Code3|
+------------+------------+------------+
|       34330|         179|           0|
|           0|           0|           0|
|           0|           0|           0|
|       33340|           0|           0|
|       11210|       22695|           0|
|           0|           0|           0|
|           0|       13820|           0|
|           0|           0|           0|
|       13820|           0|           0|
|       29520|       12335|           0|
|        9020|           0|           0|
|       22320|       22425|           0|
|       36420|       58870|           0|
|       22620|       22425|           0|
|       29090|       24940|           0|
|       53950|       49630|           0|
|       28430|       25370|       23830|
|       86530|       86730|       28430|
|       72230|       77730|       51030|
|       28430|       61010|       25370|
+------------+------------+------------+
```

## Convert the features to a vector

```
# Convert the features to a vector
assembler = VectorAssembler(inputCols=cluster_df.columns, outputCol="features")
df = assembler.transform(cluster_df)
```

```
df.show()
```

```
+------------+------------+------------+--------------------+
|Street Code1|Street Code2|Street Code3|            features|
+------------+------------+------------+--------------------+
|       34330|         179|           0| [34330.0,179.0,0.0]|
|           0|           0|           0|           (3,[],[])|
|           0|           0|           0|           (3,[],[])|
|       33340|           0|           0|   [33340.0,0.0,0.0]|
|       11210|       22695|           0|[11210.0,22695.0,...|
|           0|           0|           0|           (3,[],[])|
|           0|       13820|           0|   [0.0,13820.0,0.0]|
|           0|           0|           0|           (3,[],[])|
|       13820|           0|           0|   [13820.0,0.0,0.0]|
|       29520|       12335|           0|[29520.0,12335.0,...|
|        9020|           0|           0|    [9020.0,0.0,0.0]|
|       22320|       22425|           0|[22320.0,22425.0,...|
|       36420|       58870|           0|[36420.0,58870.0,...|
|       22620|       22425|           0|[22620.0,22425.0,...|
|       29090|       24940|           0|[29090.0,24940.0,...|
|       53950|       49630|           0|[53950.0,49630.0,...|
|       28430|       25370|       23830|[28430.0,25370.0,...|
|       86530|       86730|       28430|[86530.0,86730.0,...|
|       72230|       77730|       51030|[72230.0,77730.0,...|
|       28430|       61010|       25370|[28430.0,61010.0,...|
+------------+------------+------------+--------------------+
only showing top 20 rows
```

**Calculating the silhouette score for various values of k**

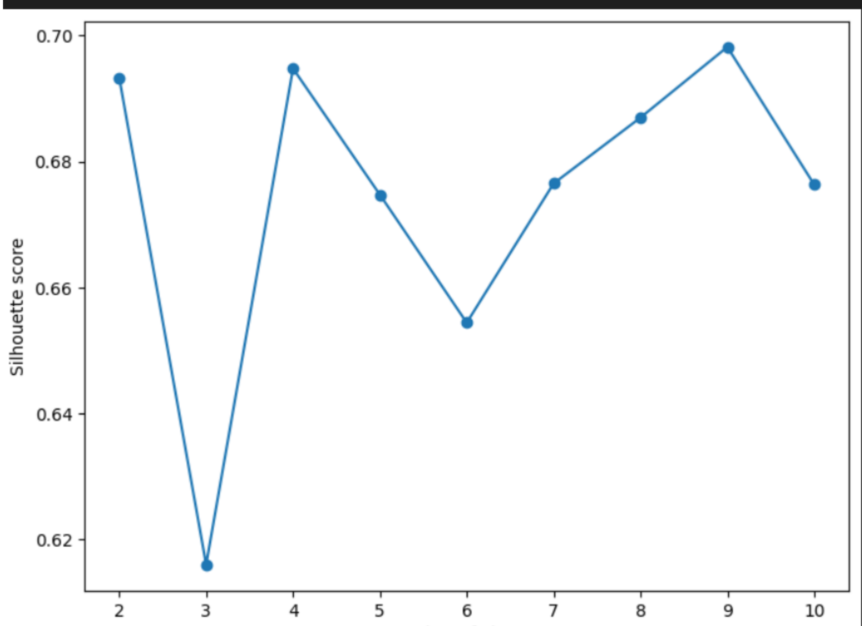
```python
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
import matplotlib.pyplot as plt
import numpy as np
```

```python
# Define a function to calculate silhouette scores
def calculate_silhouette_score(data, num_clusters):
    # Train a KMeans model
    kmeans = KMeans(k=num_clusters, seed=1, featuresCol='features')
    model = kmeans.fit(data)

    # Make predictions and evaluate the model
    predictions = model.transform(data)
    evaluator = ClusteringEvaluator()
    silhouette_score = evaluator.evaluate(predictions)
    return silhouette_score
```

```python
# Calculate silhouette scores for different number of clusters
silhouette_scores = []
for k in range(2, 11):
    score = calculate_silhouette_score(df, k)
    silhouette_scores.append(score)
```

**Plotting the silhouette scores and selecting the elbow point as k**

```python
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(range(2, 11), silhouette_scores, marker='o')
ax.set_xlabel('Number of clusters')
ax.set_ylabel('Silhouette score')
ax.set_xticks(np.arange(2, 11))
plt.show()
```




**Implementing k-means using the silhouette score obtained from above i.e. k=3**

```
# Number of clusters is 3
km = KMeans(featuresCol='features', k=3)
model = km.fit(df)
pred = model.transform(df)
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(pred)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

```
Silhouette with squared euclidean distance = 0.6489755173296069
```

**Converting dense vectors to sparse vectors and applying it as a user-defined function (UDF) on a Spark DataFrame column**

```
from pyspark.ml.linalg import VectorUDT, SparseVector
from pyspark.sql.functions import udf, col

def to_sparse_vector(v):
    indices = [i for i in range(len(v)) if v[i] != 0]
    values = [v[i] for i in range(len(v)) if v[i] != 0]
    return SparseVector(len(v), indices, values)

to_sparse_udf = udf(to_sparse_vector, VectorUDT())

prediction_df = pred.withColumn('features', to_sparse_udf(col('features'))) \
                .select('Street Code1', 'Street Code2', 'Street Code3', 'features', 'prediction')
```

**Creating a spark dataframe “street_codes" with a single row of data containing three integer values, and printing the schema of the DataFrame.**

```
street_codes_list = [(34510,10030,34050)]
street_codes = spark.createDataFrame(data=street_codes_list,schema=cluster_df.columns)
street_codes.printSchema()
```

```
root
 |-- Street Code1: long (nullable = true)
 |-- Street Code2: long (nullable = true)
 |-- Street Code3: long (nullable = true)
```

**Performing clustering on the data in "street_codes" using a model, and extracting the predicted cluster label for the first row of the data**

```
predictions_data_df = model.transform(assembler.transform(street_codes)).toPandas()
```

**Performing clustering on the data using a trained model and returning the cluster to which the first row of the transformed data belongs**

```
data_df = model.transform(assembler.transform(street_codes)).select(['prediction']).toPandas()
cluster = data_df.iloc[0,0]
print("Cluster", cluster)
```

```
Cluster 2
```

**Creating a temporary view called "NYCParking", quering the total count of rows in the view, and then constructing a query to count the number of rows in the view that have a specific value for the "prediction" column**

```
prediction_df.createOrReplaceTempView("NYCParking")
```

```
total_count = spark.sql("SELECT COUNT(*) as Total from NYCParking")
total_count.show()
```

```
+-------+
|  Total|
+-------+
|2504872|
+-------+
```

```
q = f"""SELECT COUNT(*) as total_in_cluster FROM NYCParking WHERE prediction== {cluster}"""
```

```
total_count_prediction = spark.sql(q)
total_count_prediction.show()
```

```
+----------------+
|total_in_cluster|
+----------------+
|          802771|
+----------------+
```

**Calculating the probability of an event by dividing the count of occurrences in a subset by the total count of occurrences.**

```
from pyspark.sql.functions import expr

prob_df = total_count.join(total_count_prediction)
prob_df = prob_df.withColumn("Probability", expr("total_in_cluster / Total"))
prob_df.show()
```

✓ 10.0s

```
+-------+----------------+-------------------+
|  Total|total_in_cluster|        Probability|
+-------+----------------+-------------------+
|2504872|          803051|0.32059562324941154|
+-------+----------------+-------------------+
```

## Question 2 Part 1:

### Importing libraries and reading the data

```python
from pyspark.sql.functions import *
from pyspark.sql.types import *
from pyspark.sql import SparkSession
```

```python
spark=SparkSession.builder.appName('Parking_ticket_analysis').getOrCreate()
```

```python
# Load the NBA shot logs dataset into a PySpark dataframe
nba_df = spark.read.csv("C:/Users/mayur/OneDrive/Desktop/CoursesSem2/ECC/A02/shot_logs.csv",header=True,inferSchema=True)
```

```python
nba_df.show()
```

Python

| GAME_ID | MATCHUP | LOCATION | W | FINAL_MARGIN | SHOT_NUMBER | PERIOD | GAME_CLOCK | SHOT_CLOCK | DRIBBLES | TOUCH_TIME | SHOT_DIST | PTS_TYPE | SHOT_RESULT | CLOSEST_DEFENDER | CLOSEST_ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 1 | 1 | 2023-04-21 01:09:00 | 10.8 | 2 | 1.9 | 7.7 | 2 | made | Anderson, Alan | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 2 | 1 | 2023-04-21 00:14:00 | 3.4 | 0 | 0.8 | 28.2 | 3 | missed | Bogdanovic, Bojan | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 3 | 1 | 2023-04-21 00:00:00 | null | 3 | 2.7 | 10.1 | 2 | missed | Bogdanovic, Bojan | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 4 | 2 | 2023-04-21 11:47:00 | 10.3 | 2 | 1.9 | 17.2 | 2 | missed | Brown, Markel | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 5 | 2 | 2023-04-21 10:34:00 | 10.9 | 2 | 2.7 | 3.7 | 2 | missed | Young, Thaddeus | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 6 | 2 | 2023-04-21 08:15:00 | 9.1 | 2 | 4.4 | 18.4 | 2 | missed | Williams, Deron | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 7 | 4 | 2023-04-21 10:15:00 | 14.5 | 11 | 9.0 | 20.7 | 2 | missed | Jack, Jarrett | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 8 | 4 | 2023-04-21 08:00:00 | 3.4 | 3 | 2.5 | 3.5 | 2 | made | Plumlee, Mason | |
| 21400899 | MAR 04, 2015 - CH... | A | W | 24 | 9 | 4 | 2023-04-21 05:14:00 | 12.4 | 0 | 0.8 | 24.6 | 3 | missed | Morris, Darius | |
| 21400890 | MAR 03, 2015 - CH... | H | W | 1 | 1 | 2 | 2023-04-21 11:32:00 | 17.4 | 0 | 1.1 | 22.4 | 3 | missed | Ellington, Wayne | |
| 21400890 | MAR 03, 2015 - CH... | H | W | 1 | 2 | 2 | 2023-04-21 06:30:00 | 16.0 | 8 | 7.5 | 24.5 | 3 | missed | Lin, Jeremy | |
| 21400890 | MAR 03, 2015 - CH... | H | W | 1 | 3 | 4 | 2023-04-21 11:32:00 | 12.1 | 14 | 11.9 | 14.6 | 2 | made | Lin, Jeremy | |
| 21400890 | MAR 03, 2015 - CH... | H | W | 1 | 4 | 4 | 2023-04-21 08:55:00 | 4.3 | 2 | 2.9 | 5.9 | 2 | made | Hill, Jordan | |
| 21400882 | MAR 01, 2015 - CH... | A | W | 15 | 1 | 4 | 2023-04-21 09:10:00 | 4.4 | 0 | 0.8 | 26.4 | 3 | missed | Green, Willie | |
| 21400859 | FEB 27, 2015 - CH... | A | L | -8 | 1 | 1 | 2023-04-21 00:48:00 | 6.8 | 0 | 0.5 | 22.8 | 3 | missed | Smart, Marcus | |
| 21400859 | FEB 27, 2015 - CH... | A | L | -8 | 2 | 2 | 2023-04-21 10:38:00 | 6.4 | 3 | 2.7 | 24.7 | 3 | made | Young, James | |
| 21400859 | FEB 27, 2015 - CH... | A | L | -8 | 3 | 2 | 2023-04-21 08:27:00 | 17.6 | 6 | 5.1 | 25.0 | 3 | missed | Jerebko, Jonas | |
| 21400859 | FEB 27, 2015 - CH... | A | L | -8 | 4 | 4 | 2023-04-21 10:55:00 | 8.7 | 1 | 0.9 | 25.6 | 3 | missed | Crowder, Jae | |
| 21400859 | FEB 27, 2015 - CH... | A | L | -8 | 5 | 4 | 2023-04-21 10:29:00 | 20.8 | 0 | 1.2 | 24.2 | 3 | made | Thomas, Isaiah | |
| 21400845 | FEB 25, 2015 - CH... | A | W | 12 | 1 | 1 | 2023-04-21 03:35:00 | 17.5 | 2 | 2.2 | 25.4 | 3 | missed | Brooks, Aaron | |

### Grouping the NBA dataframe by player name, closest defender, and shot result, and then aggregating the count of occurrences for each group.

```python
shot_result_df = (nba_df
                  .groupBy(['player_name', 'CLOSEST_DEFENDER', 'SHOT_RESULT'])
                  .agg(count('*').alias('count'))
                 )
```

```python
shot_result_df = nba_df.show()
```

| player_name | CLOSEST_DEFENDER | SHOT_RESULT | count |
|---|---|---|---|
| al jefferson | Aldrich, Cole | made | 2 |
| al jefferson | Morris, Marcus | made | 2 |
| gary neal | Hibbert, Roy | made | 1 |
| gerald henderson | James, LeBron | missed | 2 |
| gerald henderson | Ross, Terrence | missed | 2 |
| kemba walker | Green, Willie | missed | 1 |
| lance stephenson | Barnes, Harrison | made | 1 |
| marvin williams | Pondexter, Quincy | made | 1 |
| jason maxiell | Acy, Quincy | missed | 2 |
| gordon hayward | Oladipo, Victor | missed | 1 |
| enes kanter | Bosh, Chris | made | 2 |
| enes kanter | Hansbrough, Tyler | missed | 2 |
| jon ingles | Morris, Markieff | missed | 2 |

**Grouping NBA player names and closest defenders, pivoting on shot results, and calculating a fear score based on the proportion of shots made versus missed.**

```
# Select required columns
shot_result_df_1 = nba_df.select(['player_name', 'CLOSEST_DEFENDER', 'SHOT_RESULT'])

# Group by player name and closest defender, and pivot on shot result
fear_playerwise_count = shot_result_df_1.groupBy(['player_name', 'CLOSEST_DEFENDER']) \
                        .pivot('SHOT_RESULT') \
                        .count() \
                        .na.fill(0)

# Calculate fear score
fear_playerwise_count = fear_playerwise_count.withColumn('fear_score', col('made')/(col('made') + col('missed')))
```

```
shot_result_df_1.show()
```

```
+------------+----------------+-----------+
|  player_name| CLOSEST_DEFENDER|SHOT_RESULT|
+------------+----------------+-----------+
|brian roberts|    Anderson, Alan|       made|
|brian roberts|Bogdanovic, Bojan|     missed|
|brian roberts|Bogdanovic, Bojan|     missed|
|brian roberts|     Brown, Markel|     missed|
|brian roberts|   Young, Thaddeus|     missed|
|brian roberts|   Williams, Deron|     missed|
|brian roberts|     Jack, Jarrett|     missed|
|brian roberts|    Plumlee, Mason|       made|
|brian roberts|    Morris, Darius|     missed|
|brian roberts| Ellington, Wayne|     missed|
```

```
shot_result_df_1.show()
```

```
+--------------+-----------------+----+------+-----------------+
|   player_name|  CLOSEST_DEFENDER|made|missed|       fear_score|
+--------------+-----------------+----+------+-----------------+
|  nene hilario|Westbrook, Russell|   1|     0|              1.0|
| brian roberts|       Gasol, Pau|   0|     1|              0.0|
|   kyle korver|     Meeks, Jodie|   1|     1|              0.5|
|    mike scott|   Griffin, Blake|   0|     4|              0.0|
|     john wall| Robinson, Thomas|   1|     0|              1.0|
|  tyreke evans|    McDaniels, KJ|   1|     1|              0.5|
|harrison barnes|     Harris, Devin|   2|     1|0.6666666666666666|
|    nick young|      Love, Kevin|   1|     3|             0.25|
|     gary neal|    Smart, Marcus|   0|     4|              0.0|
|  trevor booker|  Frye, Channing|   1|     1|              0.5|
|    paul pierce|      Tucker, PJ|   3|     5|            0.375|
|  ryan anderson|    Butler, Jimmy|   1|     1|              0.5|
|    paul pierce|   Afflalo, Arron|   0|     1|              0.0|
|     john wall|  Prigioni, Pablo|   1|     0|              1.0|
|  rasual butler|Speights, Marreese|   1|     0|              1.0|
|draymond green|Westbrook, Russell|   0|     3|              0.0|
|   luke babbitt|     Jones, Perry|   0|     1|              0.0|
|    jeremy lin|     Gobert, Rudy|   0|     1|              0.0|
|    kobe bryant|Valanciunas, Jonas|   1|     2|0.3333333333333333|
| garrett temple| Mbah a Moute, Luc|   0|     1|              0.0|
+--------------+-----------------+----+------+-----------------+
only showing top 20 rows
```

**Grouping the NBA DataFrame by player name, closest defender, and shot result, and calculating the count of occurrences for each group.**

```
shot_result_df_1 = (nba_df.groupBy(['player_name', 'CLOSEST_DEFENDER', 'SHOT_RESULT'])
                    .agg(count('*').alias('count')))
```

**Grouping the "fear_playerwise_count" dataframe by player name, calculating the minimum fear score and selecting the closest defender with that minimum fear score for each player.**

```python
unwanted_defender_df = (
    fear_playerwise_count
    .groupBy("player_name")
    .agg(
        min("fear_score").alias("min_fear_score"),
        first("CLOSEST_DEFENDER").alias("most_unwanted_defender")
    )
)
```

**Q: For each pair of the players (A, B), we define the fear sore of A when facing B is the hit rate, such that B is closet defender when A is shooting. Based on the fear sore, for each player, please find out who is his ”most unwanted defender”.**

**Final output**

```
+---------------+-------------+----------------------+
|    player_name|min_fear_score|most_unwanted_defender|
+---------------+-------------+----------------------+
|   aaron brooks|          0.0|          Smith, Jason|
|   aaron gordon|          0.0|         Rivers, Austin|
| al farouq aminu|         0.0|         Lee, Courtney|
|     al horford|          0.0|        Nowitzki, Dirk|
|   al jefferson|          0.0|             Len, Alex|
|  alan anderson|          0.0|          Zeller, Cody|
|    alan crabbe|          0.0|      Sefolosha, Thabo|
|       alex len|          0.0|       Knight, Brandon|
|  alexis ajinca|          0.0|        Hawes, Spencer|
|     alonzo gee|          0.0|         Dragic, Goran|
|amare stoudemire|         0.0|        Garnett, Kevin|
|    amir johnson|         0.0|          Hibbert, Roy|
|  andre drummond|         0.0|         Millsap, Paul|
|  andre iguodala|         0.0|        Hummel, Robbie|
|    andre miller|         0.0|      Middleton, Khris|
|  andre roberson|         0.0|           Ingles, Joe|
|    andrew bogut|         0.0|         O'Quinn, Kyle|
|  andrew wiggins|         0.0|        Afflalo, Arron|
| anthony bennett|         0.0|         Ajinca, Alexis|
|   anthony davis|         0.0|        Wallace, Gerald|
+---------------+-------------+----------------------+
only showing top 20 rows
```

The most unwanted defender corresponding to every player

**Question 2 Part 2:**

**Importing libraries and reading data**

```python
import matplotlib.pyplot as plt
import pandas as pd

from pyspark.sql import SparkSession
from pyspark.sql.types import StructType, StructField, StringType, IntegerType
import pyspark.sql.functions as F

from pyspark.ml.feature import StandardScaler,VectorAssembler
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator

from datetime import date, datetime, timedelta
import time

from pyspark.sql.functions import col, dayofweek, dayofmonth, month, to_date, year
from pyspark.sql.types import *
```

```python
spark = SparkSession.builder.appName("Comfortable_Zone").getOrCreate()
```

```python
nba_df = spark.read.csv("C:/Users/mayur/OneDrive/Desktop/CoursesSem2/ECC/A02/shot_logs.csv",header=True,inferSchema=True)
```

```python
nba_df.show(20)
```

```
+--------+--------------------+--------+---+------------+-----------+------+-------------------+----------+--------+----------+---------+--------+-----------+-----------------+--------
| GAME_ID|             MATCHUP|LOCATION|  W|FINAL_MARGIN|SHOT_NUMBER|PERIOD|         GAME_CLOCK|SHOT_CLOCK|DRIBBLES|TOUCH_TIME|SHOT_DIST|PTS_TYPE|SHOT_RESULT| CLOSEST_DEFENDER|CLOSEST_
+--------+--------------------+--------+---+------------+-----------+------+-------------------+----------+--------+----------+---------+--------+-----------+-----------------+--------
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          1|     1|2023-04-21 01:09:00|      10.8|       2|       1.9|      7.7|       2|       made|   Anderson, Alan|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          2|     1|2023-04-21 00:14:00|       3.4|       0|       0.8|     28.2|       3|     missed|Bogdanovic, Bojan|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          3|     1|2023-04-21 00:00:00|      null|       3|       2.7|     10.1|       2|     missed|Bogdanovic, Bojan|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          4|     2|2023-04-21 11:47:00|      10.3|       2|       1.9|     17.2|       2|     missed|    Brown, Markel|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          5|     2|2023-04-21 10:34:00|      10.9|       2|       2.7|      3.7|       2|     missed|  Young, Thaddeus|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          6|     2|2023-04-21 08:15:00|       9.1|       2|       4.4|     18.4|       2|     missed|  Williams, Deron|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          7|     4|2023-04-21 10:15:00|      14.5|      11|       9.0|     20.7|       2|     missed|    Jack, Jarrett|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          8|     4|2023-04-21 08:00:00|       3.4|       3|       2.5|      3.5|       2|       made|   Plumlee, Mason|
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          9|     4|2023-04-21 05:14:00|      12.4|       0|       0.8|     24.6|       3|     missed|   Morris, Darius|
|21400890|MAR 03, 2015 - CH...|       H|  W|           1|          1|     2|2023-04-21 11:32:00|      17.4|       0|       1.1|     22.4|       3|     missed| Ellington, Wayne|
|21400890|MAR 03, 2015 - CH...|       H|  W|           1|          2|     2|2023-04-21 06:30:00|      16.0|       8|       7.5|     24.5|       3|     missed|      Lin, Jeremy|
|21400890|MAR 03, 2015 - CH...|       H|  W|           1|          3|     4|2023-04-21 11:32:00|      12.1|      14|      11.9|     14.6|       2|       made|      Lin, Jeremy|
|21400890|MAR 03, 2015 - CH...|       H|  W|           1|          4|     4|2023-04-21 08:55:00|       4.3|       2|       2.9|      5.9|       2|       made|     Hill, Jordan|
|21400882|MAR 01, 2015 - CH...|       A|  W|          15|          1|     4|2023-04-21 09:10:00|       4.4|       0|       0.8|     26.4|       3|     missed|    Green, Willie|
|21400859|FEB 27, 2015 - CH...|       A|  L|          -8|          1|     1|2023-04-21 00:48:00|       6.8|       0|       0.5|     22.8|       3|     missed|    Smart, Marcus|
```

**Grouping the "fear_playerwise_count" dataframe by player name, calculating the minimum fear score and selecting the closest defender with that minimum fear score for each player.**

```python
new_df = (nba_df.dropna().toPandas())
```

```
C:\Users\mayur\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.10_qbz5n2kfra8p0\LocalCache\local-packages\Python310\site-packages\pyspark\sql\pand
  series = series.astype(t, copy=False)
```

```python
df = nba_df.select('SHOT_DIST', 'CLOSE_DEF_DIST', 'SHOT_CLOCK').dropna()
```

**Spark's VectorAssembler to combine all columns in the DataFrame "df" into a single column of feature vectors called "features"**

```python
assembler = VectorAssembler()
assembler.setInputCols(df.columns)
assembler.setOutputCol('features')
assembler_data = assembler.transform(df)
```

```python
assembler_data.show(5)
```

```
+---------+--------------+----------+--------------+
|SHOT_DIST|CLOSE_DEF_DIST|SHOT_CLOCK|      features|
+---------+--------------+----------+--------------+
|      7.7|           1.3|      10.8| [7.7,1.3,10.8]|
|     28.2|           6.1|       3.4| [28.2,6.1,3.4]|
|     17.2|           3.4|      10.3|[17.2,3.4,10.3]|
|      3.7|           1.1|      10.9| [3.7,1.1,10.9]|
|     18.4|           2.6|       9.1| [18.4,2.6,9.1]|
+---------+--------------+----------+--------------+
only showing top 5 rows
```

## Calculating silhouette scores for various values for k

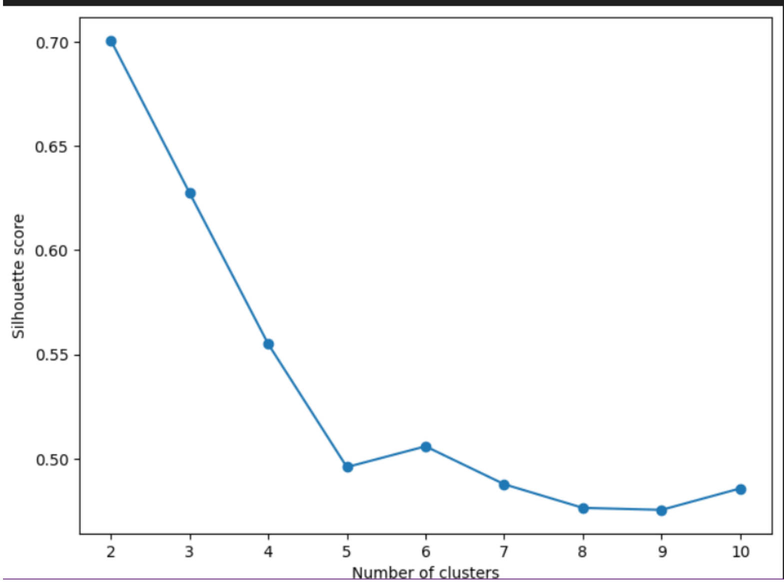
```python
def calculate_silhouette_score(data, num_clusters):
    # Train a KMeans model
    kmeans = KMeans(k=num_clusters, featuresCol='features')
    model = kmeans.fit(data)

    # Make predictions and evaluate the model
    predictions = model.transform(data)
    evaluator = ClusteringEvaluator()
    silhouette_score = evaluator.evaluate(predictions)
    return silhouette_score
```

```python
# Calculate silhouette scores for different number of clusters
silhouette_scores = []
for k in range(2, 11):
    score = calculate_silhouette_score(assembled_data, k)
    print(score, k)
    silhouette_scores.append(score)
```

```
0.7005693569346803 2
0.6274971328256687 3
0.5550339306822153 4
0.496023939066375 5
0.5059735416306909 6
0.48779556198511104 7
0.47646810182324506 8
0.4755061270394926 9
```

## Plotting the silhouette scores

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.plot(range(2, 11), silhouette_scores, marker='o')
ax.set_xlabel('Number of clusters')
ax.set_ylabel('Silhouette score')
ax.set_xticks(np.arange(2, 11))
plt.show()
```




**Implementing k-means on k=5**

```
# Number of clusters is 3
km = KMeans(featuresCol='features', k=5)
model = km.fit(assembler_data)
pred = model.transform(assembler_data)
# Evaluate clustering by computing Silhouette score
evaluator = ClusteringEvaluator()
silhouette = evaluator.evaluate(pred)
print("Silhouette with squared euclidean distance = " + str(silhouette))
```

```
Silhouette with squared euclidean distance = 0.496023939066375
```

**Collecting predictions from a Spark DataFrame and creating a new Pandas DataFrame with selected columns.**

```
# Collect the predictions
predictions_collected = pred.collect()

# Select columns of interest and convert to Pandas dataframe
pred_df = pred.select(['prediction','SHOT_DIST','CLOSE_DEF_DIST','SHOT_CLOCK']).toPandas()

# Create a new Pandas dataframe with selected columns
pred_data = pred_df[['prediction','SHOT_DIST','CLOSE_DEF_DIST','SHOT_CLOCK']]
```

**Concatenating two dataframes (new_df and pred_data) horizontally and keeping only the common columns (intersection) in the resulting dataframe "all_data".**

```python
all_data = pd.concat([new_df, pred_data], axis=1, join='inner')
all_data.head()
```

Python

| | GAME_ID | MATCHUP | LOCATION | W | FINAL_MARGIN | SHOT_NUMBER | PERIOD | GAME_CLOCK | SHOT_CLOCK | DRIBBLES | ... | CLOSEST_DEFENDER_PLAYER_ID | CLOSE_DEF_DIST | FGM | PTS | player_r |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 1 | 1 | 2023-04-21 01:09:00 | 10.8 | 2 | ... | 101187 | 1.3 | 1 | 2 | brian ro |
| 1 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 2 | 1 | 2023-04-21 00:14:00 | 3.4 | 0 | ... | 202711 | 6.1 | 0 | 0 | brian ro |
| 2 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 4 | 2 | 2023-04-21 11:47:00 | 10.3 | 2 | ... | 203900 | 3.4 | 0 | 0 | brian ro |
| 3 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 5 | 2 | 2023-04-21 10:34:00 | 10.9 | 2 | ... | 201152 | 1.1 | 0 | 0 | brian ro |
| 4 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 6 | 2 | 2023-04-21 08:15:00 | 9.1 | 2 | ... | 101114 | 2.6 | 0 | 0 | brian ro |

**Merging two data frames based on the columns 'SHOT_DIST', 'CLOSE_DEF_DIST', and 'SHOT_CLOCK'.**

```python
all_data = new_df.merge(pred_data, on=['SHOT_DIST','CLOSE_DEF_DIST','SHOT_CLOCK'])
all_data.head()
```

Python

| | GAME_ID | MATCHUP | LOCATION | W | FINAL_MARGIN | SHOT_NUMBER | PERIOD | GAME_CLOCK | SHOT_CLOCK | DRIBBLES | ... | PTS_TYPE | SHOT_RESULT | CLOSEST_DEFENDER | CLOSEST_DEFENDER_PI |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 1 | 1 | 2023-04-21 01:09:00 | 10.8 | 2 | ... | 2 | made | Anderson, Alan | |
| 1 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 2 | 1 | 2023-04-21 00:14:00 | 3.4 | 0 | ... | 3 | missed | Bogdanovic, Bojan | |
| 2 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 4 | 2 | 2023-04-21 11:47:00 | 10.3 | 2 | ... | 2 | missed | Brown, Markel | |
| 3 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 5 | 2 | 2023-04-21 10:34:00 | 10.9 | 2 | ... | 2 | missed | Young, Thaddeus | |
| 4 | 21400899 | MAR 04, 2015 - CHA @ BKN | A | W | 24 | 6 | 2 | 2023-04-21 08:15:00 | 9.1 | 2 | ... | 2 | missed | Williams, Deron | |

**Creating a new Spark DataFrame from the data in 'all_data' and displays the first 10 rows, but it doesn't do anything with the 'players_list' variable.**

```python
new_df=spark.createDataFrame(all_data)
players_list = ['james harden','chris paul','stephen curry','lebron james']
new_df.head(10)
```

Python

```
[Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=1, PERIOD=1, GAME_CLOCK=datetime.datetime(2023, 4, 21, 1, 9), SHOT_CLOCK=10.
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=2, PERIOD=1, GAME_CLOCK=datetime.datetime(2023, 4, 21, 0, 14), SHOT_CLOCK=3.
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=4, PERIOD=2, GAME_CLOCK=datetime.datetime(2023, 4, 21, 11, 47), SHOT_CLOCK=1
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=5, PERIOD=2, GAME_CLOCK=datetime.datetime(2023, 4, 21, 10, 34), SHOT_CLOCK=1
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=6, PERIOD=2, GAME_CLOCK=datetime.datetime(2023, 4, 21, 8, 15), SHOT_CLOCK=9.
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=7, PERIOD=4, GAME_CLOCK=datetime.datetime(2023, 4, 21, 10, 15), SHOT_CLOCK=1
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=7, PERIOD=4, GAME_CLOCK=datetime.datetime(2023, 4, 21, 10, 15), SHOT_CLOCK=1
 Row(GAME_ID=21400053, MATCHUP='NOV 04, 2014 - ORL @ CHI', LOCATION='A', W='L', FINAL_MARGIN=-8, SHOT_NUMBER=3, PERIOD=1, GAME_CLOCK=datetime.datetime(2023, 4, 21, 4, 6), SHOT_CLOCK=14.
 Row(GAME_ID=21400053, MATCHUP='NOV 04, 2014 - ORL @ CHI', LOCATION='A', W='L', FINAL_MARGIN=-8, SHOT_NUMBER=3, PERIOD=1, GAME_CLOCK=datetime.datetime(2023, 4, 21, 4, 6), SHOT_CLOCK=14.
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=8, PERIOD=4, GAME_CLOCK=datetime.datetime(2023, 4, 21, 8, 0), SHOT_CLOCK=3.4
```

**Code creates an empty Spark DataFrame with a defined schema for player data.**

```python
# Create an empty RDD
empty_rdd = spark.sparkContext.emptyRDD()

# Define the schema for the DataFrame
player_schema = StructType([
    StructField('name', StringType(), False),
    StructField('group', StringType(), False),
    StructField('hits', IntegerType(), False),
    StructField('total', IntegerType(), False)
])

# Create a DataFrame with the empty RDD and the defined schema
player_df = spark.createDataFrame(empty_rdd, schema=player_schema)
```

**Creating a new DataFrame by filtering a given list of player names from an existing DataFrame, creates an empty DataFrame with a given schema using Spark, and creating a temporary view named "NBAData" from the filtered DataFrame.**

```
filtered_data = new_df.filter(new_df.player_name.isin(players_list))
data_rdd = spark.createDataFrame(data=empty_rdd,schema=player_schema)
filtered_data.createOrReplaceTempView("NBAData")
```

**Grouping and aggregating the data by player name and prediction to count the number of successful shots made by each player, and displaying the results in a table.**

```
from pyspark.sql.functions import count, when

df = filtered_data.filter(filtered_data.SHOT_RESULT == 'made') \
            .groupBy('player_name', 'prediction') \
            .agg(count('SHOT_RESULT').alias('hits')) \
            .select('player_name', 'prediction', 'hits')
df.show()
```

```
+-------------+----------+----+
|  player_name|prediction|hits|
+-------------+----------+----+
| lebron james|         4|  94|
|stephen curry|         3| 205|
| james harden|         1| 124|
| james harden|         0| 129|
|stephen curry|         1|  84|
| lebron james|         2|  82|
|   chris paul|         3| 173|
|stephen curry|         2|  33|
|stephen curry|         0|  65|
|stephen curry|         4|  98|
| james harden|         3|  95|
| lebron james|         0| 110|
|   chris paul|         4|  76|
| james harden|         2|  40|
```

**Grouping and aggregating data based on player name and prediction, and then selects and displays the player name, prediction, and total number of shots for each player.**

```python
result_df = filtered_data \
    .groupBy("player_name", "prediction") \
    .agg(count("*").alias("total_shots"), \
         filtered_data["player_name"].alias("player"), \
         filtered_data["prediction"].alias("cluster_group")) \
    .select("player", "cluster_group", "total_shots")

result_df.show()
```

```
+------------+-------------+-----------+
|      player|cluster_group|total_shots|
+------------+-------------+-----------+
| james harden|            3|        299|
| lebron james|            4|        186|
|stephen curry|            3|        478|
| lebron james|            1|        189|
|   chris paul|            3|        352|
| james harden|            1|        217|
|stephen curry|            0|        162|
| lebron james|            0|        301|
| james harden|            4|        186|
|   chris paul|            0|        267|
| lebron james|            3|        180|
| james harden|            0|        297|
|   chris paul|            2|         70|
|stephen curry|            1|        126|
| lebron james|            2|        147|
|stephen curry|            2|         57|
|stephen curry|            4|        165|
|   chris paul|            1|         45|
|   chris paul|            4|        157|
| james harden|            2|         88|
+------------+-------------+-----------+
```

**Joining two queries on player name and cluster, and then adding a new column "Hit_rate" to the resulting DataFrame, which is calculated as hits divided by total shots.**

```python
join_data = query1.join(query2,(query1.player_name==query2.player) & (query1.cluster==query2.cluster_group))
join_data
```

```
DataFrame[player_name: string, cluster: bigint, hits: bigint, player: string, cluster_group: bigint, total_shots: bigint]
```

```python
join_data = join_data.withColumn("Hit_rate",(F.col("hits")/F.col("total_shots")))
```

```python
join_data.show()
```

```
+-------------+-------+----+-------------+-------------+-----------+-------------------+
|  player_name|cluster|hits|       player|cluster_group|total_shots|           Hit_rate|
+-------------+-------+----+-------------+-------------+-----------+-------------------+
| james harden|      3|  95| james harden|            3|        299| 0.3177257525083612|
| lebron james|      4|  94| lebron james|            4|        186| 0.5053763440860215|
|stephen curry|      3| 205|stephen curry|            3|        478|0.42887029288702927|
| lebron james|      1| 136| lebron james|            1|        189| 0.7195767195767195|
|   chris paul|      3| 173|   chris paul|            3|        352| 0.4914772727272727|
| james harden|      1| 124| james harden|            1|        217| 0.5714285714285714|
|stephen curry|      0|  65|stephen curry|            0|        162| 0.4012345679012346|
| lebron james|      0| 110| lebron james|            0|        301| 0.3654485049833887|
| james harden|      4|  99| james harden|            4|        186|  0.532258064516129|
|   chris paul|      0| 117|   chris paul|            0|        267|0.43820224719101125|
| lebron james|      3|  72| lebron james|            3|        180|                0.4|
| james harden|      0| 129| james harden|            0|        297|0.43434343434343436|
|   chris paul|      2|  37|   chris paul|            2|         70| 0.5285714285714286|
|stephen curry|      1|  84|stephen curry|            1|        126| 0.6666666666666666|
| lebron james|      2|  82| lebron james|            2|        147| 0.5578231292517006|
|stephen curry|      2|  33|stephen curry|            2|         57| 0.5789473684210527|
|stephen curry|      4|  98|stephen curry|            4|        165|  0.593939393939394|
|   chris paul|      1|  28|   chris paul|            1|         45| 0.6222222222222222|
|   chris paul|      4|  76|   chris paul|            4|        157| 0.4840764331210191|
| james harden|      2|  40| james harden|            2|         88|0.45454545454545453|
+-------------+-------+----+-------------+-------------+-----------+-------------------+
```

**SQL query on a Spark DataFrame named "HitRate" to select the player_name, cluster, and Hit_rate where the Hit_rate is the maximum value for each player_name.**

```
predicted = spark.sql("SELECT player_name,cluster as comfort_zone,Hit_rate FROM HitRate WHERE Hit_rate in (SELECT max(Hit_rate) FROM HitRate GROUP BY player_name)")
predicted.show()
```

```
+------------+------------+-----------------+
| player_name|comfort_zone|         Hit_rate|
+------------+------------+-----------------+
|  chris paul|           1|0.6222222222222222|
| lebron james|          1|0.7195767195767195|
|stephen curry|          1|0.6666666666666666|
| james harden|          1|0.5714285714285714|
+------------+------------+-----------------+
```

**Q: For each player, we define the comfortable zone of shooting is a matrix of,{SHOT DIST, CLOSE DEF DIST, SHOT CLOCK}. Please develop a Spark-based algorithm to classify each player's records into 4 comfortable zones. Considering the hit rate, which zone is the best for James Harden, Chris Paul,Stephen Curry, and Lebron James.**

**Ans:**

**chris paul Comfort zone 1**

**lebron james Comfort zone 1**

**stephen curry Comfort zone 1**

**james harden Comfort zone 1**