Using google.colab to import datasets from google drive by mounting the drive. Have uploaded both the datasets directly there. Can use the upload functionality of the same. But I prefer mounting the drive.

```
!pip install pyspark


from google.colab import drive
drive.mount('/content/drive')
```

```
Collecting pyspark
  Downloading pyspark-3.5.1.tar.gz (317.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 317.0/317.0 MB 2.8 MB/s eta
0:00:00
etadata (setup.py) ... ent already satisfied: py4j==0.10.9.7 in
/usr/local/lib/python3.10/dist-packages (from pyspark) (0.10.9.7)
Building wheels for collected packages: pyspark
  Building wheel for pyspark (setup.py) ... e=pyspark-3.5.1-py2.py3-
none-any.whl size=317488491
sha256=2b1c9293c4f8f0e4a508310a7fb45da1fa42df0afeb1ce0d94da1023b3202c6
b
  Stored in directory:
/root/.cache/pip/wheels/80/1d/60/2c256ed38dddce2fdd93be545214a63e02fbd
8d74fb0b7f3a6
Successfully built pyspark
Installing collected packages: pyspark
Successfully installed pyspark-3.5.1
Mounted at /content/drive
```

Importing Libraries

```
import os
import numpy as np
import pandas as pd
import shutil
import pyspark
from pyspark.sql.functions import when, count, col, sum,
regexp_replace
from pyspark import SparkContext
import pyspark.sql.functions as psf
from pyspark.sql import SparkSession
from pyspark.sql import SparkSession, SQLContext, Window
from pyspark.sql.types import IntegerType
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler

spkObj =
pyspark.sql.SparkSession.builder.appName("vekal_Assignment_2").getOrCr
eate()
```

```
spkObj.conf.set("spark.sql.repl.eagerEval.enabled", True)

spkObj
```

```
<pyspark.sql.session.SparkSession at 0x7f91a7fc5c90>
```

QUESTION 1

```
QUE1_NYC = spkObj.read.format("csv").option("header",
"true").option("inferSchema",
"true").load('/content/drive/MyDrive/Parking_Violations_Issued_-
_Fiscal_Year_2024_20240411.csv')

QUE1_NYC.show(n=2, truncate=False, vertical=True)
```

```
-RECORD 0-----------------------------------------
 Summons Number                     | 1159637337
 Plate ID                           | KZH2758
 Registration State                 | NY
 Plate Type                         | PAS
 Issue Date                         | 06/09/2023
 Violation Code                     | 67
 Vehicle Body Type                  | VAN
 Vehicle Make                       | HONDA
 Issuing Agency                     | P
 Street Code1                       | 0
 Street Code2                       | 0
 Street Code3                       | 0
 Vehicle Expiration Date            | 20250201
 Violation Location                 | 43
 Violation Precinct                 | 43
 Issuer Precinct                    | 43
 Issuer Code                        | 972773
 Issuer Command                     | 0043
 Issuer Squad                       | 0000
 Violation Time                     | 0911A
 Time First Observed                | NULL
 Violation County                   | BX
 Violation In Front Of Or Opposite  | NULL
 House Number                       | NULL
 Street Name                        | I/O TAYLOR AVE
 Intersecting Street                | GUERLAIN
 Date First Observed                | 0
 Law Section                        | 408
 Sub Division                       | E5
 Violation Legal Code               | NULL
 Days Parking In Effect             | BBBBBBB
 From Hours In Effect               | ALL
```

```
 To Hours In Effect                | ALL
 Vehicle Color                     | BLUE
 Unregistered Vehicle?             | 0
 Vehicle Year                      | 2006
 Meter Number                      | -
 Feet From Curb                    | 0
 Violation Post Code               | NULL
 Violation Description             | NULL
 No Standing or Stopping Violation | NULL
 Hydrant Violation                 | NULL
 Double Parking Violation          | NULL
-RECORD 1-----------------------------------------
 Summons Number                    | 1252960645
 Plate ID                          | JPD8746
 Registration State                | NY
 Plate Type                        | PAS
 Issue Date                        | 06/30/2023
 Violation Code                    | 87
 Vehicle Body Type                 | SUBN
 Vehicle Make                      | LINCO
 Issuing Agency                    | M
 Street Code1                      | 17870
 Street Code2                      | 25390
 Street Code3                      | 32670
 Vehicle Expiration Date           | 20240210
 Violation Location                | 14
 Violation Precinct                | 14
 Issuer Precinct                   | 968
 Issuer Code                       | 271057
 Issuer Command                    | 0968
 Issuer Squad                      | 0000
 Violation Time                    | 0717A
 Time First Observed               | NULL
 Violation County                  | NY
 Violation In Front Of Or Opposite | O
 House Number                      | 51
 Street Name                       | E 44TH ST
 Intersecting Street               | NULL
 Date First Observed               | 0
 Law Section                       | 408
 Sub Division                      | D
 Violation Legal Code              | NULL
 Days Parking In Effect            | BBBBBBB
 From Hours In Effect              | ALL
 To Hours In Effect                | ALL
 Vehicle Color                     | GRAY
 Unregistered Vehicle?             | 0
 Vehicle Year                      | 2020
 Meter Number                      | -
```

```
 Feet From Curb                     | 0
 Violation Post Code                | NULL
 Violation Description              | NULL
 No Standing or Stopping Violation | NULL
 Hydrant Violation                  | NULL
 Double Parking Violation           | NULL
only showing top 2 rows
```

Number of Rows and Columns

```
print("Total Number of  Rows: " , QUE1_NYC.count())
print("Total Number of Columns: " , len(QUE1_NYC.columns))
```

```
Total Number of  Rows:  10717482
Total Number of Columns:  43
```

Schema

```
QUE1_NYC.printSchema()
```

```
root
 |-- Summons Number: long (nullable = true)
 |-- Plate ID: string (nullable = true)
 |-- Registration State: string (nullable = true)
 |-- Plate Type: string (nullable = true)
 |-- Issue Date: string (nullable = true)
 |-- Violation Code: integer (nullable = true)
 |-- Vehicle Body Type: string (nullable = true)
 |-- Vehicle Make: string (nullable = true)
 |-- Issuing Agency: string (nullable = true)
 |-- Street Code1: integer (nullable = true)
 |-- Street Code2: integer (nullable = true)
 |-- Street Code3: integer (nullable = true)
 |-- Vehicle Expiration Date: integer (nullable = true)
 |-- Violation Location: integer (nullable = true)
 |-- Violation Precinct: integer (nullable = true)
 |-- Issuer Precinct: integer (nullable = true)
 |-- Issuer Code: integer (nullable = true)
 |-- Issuer Command: string (nullable = true)
 |-- Issuer Squad: string (nullable = true)
 |-- Violation Time: string (nullable = true)
 |-- Time First Observed: string (nullable = true)
 |-- Violation County: string (nullable = true)
 |-- Violation In Front Of Or Opposite: string (nullable = true)
 |-- House Number: string (nullable = true)
 |-- Street Name: string (nullable = true)
 |-- Intersecting Street: string (nullable = true)
 |-- Date First Observed: integer (nullable = true)
```

```
 |-- Law Section: integer (nullable = true)
 |-- Sub Division: string (nullable = true)
 |-- Violation Legal Code: string (nullable = true)
 |-- Days Parking In Effect    : string (nullable = true)
 |-- From Hours In Effect: string (nullable = true)
 |-- To Hours In Effect: string (nullable = true)
 |-- Vehicle Color: string (nullable = true)
 |-- Unregistered Vehicle?: integer (nullable = true)
 |-- Vehicle Year: integer (nullable = true)
 |-- Meter Number: string (nullable = true)
 |-- Feet From Curb: integer (nullable = true)
 |-- Violation Post Code: string (nullable = true)
 |-- Violation Description: string (nullable = true)
 |-- No Standing or Stopping Violation: string (nullable = true)
 |-- Hydrant Violation: string (nullable = true)
 |-- Double Parking Violation: string (nullable = true)
```

Sample Data

```
QUE1_NYC.selectExpr("*").show(1)
```

```
+--------------+--------+------------------+----------+----------
+--------------+-----------------+------------+-------------
+------------+------------+------------+----------------------
+------------------+------------------+---------------+----------
+--------------+------------+--------------+------------------
+----------------+-----------------------------------+------------
+--------------+--------------------+--------------------+----------
+------------+---------------------+-------------------------
+--------------------+------------------+-------------
+----------------------+------------+------------+--------------
+--------------------+--------------------
+---------------------------------+-----------------
+-------------------------+
|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 Date|Violation
Location|Violation Precinct|Issuer Precinct|Issuer Code|Issuer
Command|Issuer Squad|Violation Time|Time First Observed|Violation
County|Violation In Front Of Or Opposite|House Number|   Street Name|
Intersecting Street|Date First Observed|Law Section|Sub Division|
Violation Legal Code|Days Parking In Effect    |From Hours In Effect|
To Hours In Effect|Vehicle Color|Unregistered Vehicle?|Vehicle Year|
Meter Number|Feet From Curb|Violation Post Code|Violation Description|
No Standing or Stopping Violation|Hydrant Violation|Double Parking
Violation|
+--------------+--------+------------------+----------+----------
+--------------+-----------------+------------+-------------
```

```
+------------+------------+------------+----------------------
+------------------+------------------+--------------+----------
+--------------+------------+--------------+------------------
+----------------+-----------------------------------+-----------
+--------------+-------------------+-------------------+----------
+------------+--------------------+-------------------------
+--------------------+------------------+-------------
+---------------------+------------+------------+-------------
+-------------------+--------------------
+---------------------------------+-----------------
+-------------------------+
|    1159637337| KZH2758|                NY|       PAS|06/09/2023|
67|              VAN|       HONDA|             P|           0|
0|           0|               20250201|                43|
43|             43|     972773|          0043|        0000|
0911A|               NULL|              BX|
NULL|        NULL|I/O TAYLOR AVE|           GUERLAIN|
0|        408|          E5|                NULL|
BBBBBBB|                 ALL|               ALL|         BLUE|
0|        2006|           -|              0|               NULL|
NULL|                              NULL|              NULL|
NULL|
+--------------+--------+------------------+----------+----------
+--------------+-----------------+------------+-------------
+------------+------------+------------+----------------------
+------------------+------------------+--------------+----------
+--------------+------------+--------------+------------------
+----------------+-----------------------------------+-----------
+--------------+-------------------+-------------------+----------
+------------+--------------------+-------------------------
+--------------------+------------------+-------------
+---------------------+------------+------------+-------------
+-------------------+--------------------
+---------------------------------+-----------------
+-------------------------+
only showing top 1 row
```

Null Values in the dataset

```
from pyspark.sql.functions import isnan, when, count, col
QUE1_NYC.select([count(when(col(column).isNull(),
column)).alias(column) for column in
QUE1_NYC.columns]).show(vertical=True)
```

```
-RECORD 0-----------------------------------
 Summons Number                    | 0
 Plate ID                          | 1
 Registration State                | 0
```

```
Plate Type                          | 0
Issue Date                          | 0
Violation Code                      | 0
Vehicle Body Type                   | 28486
Vehicle Make                        | 10679
Issuing Agency                      | 0
Street Code1                        | 0
Street Code2                        | 0
Street Code3                        | 0
Vehicle Expiration Date             | 0
Violation Location                  | 4923863
Violation Precinct                  | 0
Issuer Precinct                     | 0
Issuer Code                         | 0
Issuer Command                      | 4918591
Issuer Squad                        | 5292644
Violation Time                      | 336
Time First Observed                 | 10087137
Violation County                    | 102892
Violation In Front Of Or Opposite   | 4973482
House Number                        | 5015875
Street Name                         | 1507
Intersecting Street                 | 4980009
Date First Observed                 | 0
Law Section                         | 0
Sub Division                        | 1767
Violation Legal Code                | 5799044
Days Parking In Effect              | 5014718
From Hours In Effect                | 7338955
To Hours In Effect                  | 7338965
Vehicle Color                       | 1015121
Unregistered Vehicle?               | 10490502
Vehicle Year                        | 0
Meter Number                        | 9381186
Feet From Curb                      | 0
Violation Post Code                 | 5519326
Violation Description               | 227812
No Standing or Stopping Violation   | 10717482
Hydrant Violation                   | 10717482
Double Parking Violation            | 10717482
```

Pre-processing and Handling Null Values

```
QUE1_NYC = QUE1_NYC.dropna(subset=['Violation Time'])

QUE1_NYC = QUE1_NYC.dropna(subset=['Vehicle Body Type'])
```

```
QUE1_NYC = QUE1_NYC.dropna(subset=['Violation Location'])

QUE1_NYC = QUE1_NYC.dropna(subset=['Vehicle Color'])

QUE1_NYC.select([count(when(col(column).isNull(),
column)).alias(column) for column in
QUE1_NYC.columns]).show(vertical=True)
```

```
-RECORD 0-----------------------------------
 Summons Number                     | 0
 Plate ID                           | 0
 Registration State                 | 0
 Plate Type                         | 0
 Issue Date                         | 0
 Violation Code                     | 0
 Vehicle Body Type                  | 0
 Vehicle Make                       | 5004
 Issuing Agency                     | 0
 Street Code1                       | 0
 Street Code2                       | 0
 Street Code3                       | 0
 Vehicle Expiration Date            | 0
 Violation Location                 | 0
 Violation Precinct                 | 0
 Issuer Precinct                    | 0
 Issuer Code                        | 0
 Issuer Command                     | 0
 Issuer Squad                       | 372077
 Violation Time                     | 0
 Time First Observed                | 5135396
 Violation County                   | 16740
 Violation In Front Of Or Opposite  | 49635
 House Number                       | 90803
 Street Name                        | 442
 Intersecting Street                | 4776952
 Date First Observed                | 0
 Law Section                        | 0
 Sub Division                       | 617
 Violation Legal Code               | 5757591
 Days Parking In Effect             | 95438
 From Hours In Effect               | 2418019
 To Hours In Effect                 | 2418029
 Vehicle Color                      | 0
 Unregistered Vehicle?              | 5566827
 Vehicle Year                       | 0
 Meter Number                       | 4458269
 Feet From Curb                     | 0
 Violation Post Code                | 562687
 Violation Description              | 190942
 No Standing or Stopping Violation  | 5757591
```

```
Hydrant Violation                | 5757591
Double Parking Violation         | 5757591
```

Create View for NYC Parking Data

```
QUE1_NYC = QUE1_NYC.withColumn('Issue Year',
psf.year(psf.to_date(QUE1_NYC['Issue Date'], 'MM/dd/yyyy')))
QUE1_NYC.createOrReplaceTempView("QUE1_NYCView")
```

When are tickets most likely to be issued? (15 pts)

```
spkObj.sql("SELECT `Violation Time`, COUNT(*) AS Ticket_Frequency FROM
QUE1_NYCView GROUP BY `Violation Time` ORDER BY Ticket_Frequency
DESC").show()
```

```
+--------------+----------------+
|Violation Time|Ticket_Frequency|
+--------------+----------------+
|         0836A|           16176|
|         0838A|           15559|
|         0839A|           15544|
|         0840A|           15381|
|         0906A|           15142|
|         0841A|           15017|
|         0837A|           14901|
|         0842A|           14724|
|         0908A|           14488|
|         0843A|           14430|
|         0845A|           14408|
|         0910A|           14388|
|         0909A|           14379|
|         0907A|           14263|
|         0844A|           14226|
|         1140A|           13978|
|         0846A|           13899|
|         1141A|           13789|
|         1139A|           13713|
|         0911A|           13706|
+--------------+----------------+
only showing top 20 rows
```

Answer:

- We have our maximum number of violators i.e 16176 at 0836A which 8:36 AM.

Used violation time to find this out

What are the most common years and types of cars to be ticketed? (15 pts)

```
spkObj.sql("SELECT `Vehicle Body Type` as `Type of Car`,`Issue Year`,
COUNT(*) AS `Violation_Count` FROM QUE1_NYCView WHERE (`Vehicle Year`
> 0) GROUP BY `Vehicle Body Type`, `Issue Year` ORDER BY
`Violation_Count` DESC").show()
```

```
+-----------+----------+---------------+
|Type of Car|Issue Year|Violation_Count|
+-----------+----------+---------------+
|       SUBN|      2023|        1455740|
|       4DSD|      2023|         762700|
|       SUBN|      2024|         470615|
|        VAN|      2023|         438855|
|       4DSD|      2024|         233471|
|        VAN|      2024|         137466|
|       PICK|      2023|         100055|
|       DELV|      2023|          85359|
|       2DSD|      2023|          56158|
|        SDN|      2023|          49267|
|       REFG|      2023|          39034|
|       PICK|      2024|          32876|
|       DELV|      2024|          25891|
|       2DSD|      2024|          16746|
|       CONV|      2023|          15146|
|       REFG|      2024|          12309|
|       UTIL|      2023|          11866|
|       TRAC|      2023|          10499|
|        SDN|      2024|           9530|
|       SEDN|      2023|           8616|
+-----------+----------+---------------+
only showing top 20 rows
```

Answer:

- The most common year seems to be 2023 and the most common car seems to be SUBN.

Issue Year Column was created with the help of issue date. Which was then used to find most common years and the type of vehicle.

Where are tickets most commonly issued? (15 pts)

```
spkObj.sql("SELECT `Violation Location`, COUNT(*) AS No_of_tickets
FROM QUE1_NYCView GROUP BY `Violation Location` ORDER BY count(*)
DESC").show()
```

```
+------------------+-------------+
|Violation Location|No_of_tickets|
+------------------+-------------+
|                 19|       275694|
|                114|       211702|
|                  6|       207346|
|                 13|       188822|
|                 14|       177747|
|                109|       153290|
|                  1|       147416|
|                 18|       147053|
|                  9|       141548|
|                115|       135019|
|                 61|       116063|
|                 66|       115531|
|                 20|       115337|
|                112|       109566|
|                 70|       107169|
|                 84|       103834|
|                103|       103248|
|                108|       102315|
|                 52|       101996|
|                 46|        98129|
+------------------+-------------+
only showing top 20 rows
```

Answer:

- Most number of tickets have been issued at Location 19

The Violation Location was used to find the location where most tickets were issued.

Which color of the vehicle is most likely to get a ticket? (15 pts)

```
spkObj.sql("SELECT `Vehicle Color`, COUNT(*) as Ticket_Count FROM
QUE1_NYCView GROUP BY `Vehicle Color` ORDER BY COUNT(*) DESC").show()
```

```
+-------------+------------+
|Vehicle Color|Ticket_Count|
+-------------+------------+
|           WH|     1100571|
|           GY|     1013560|
|           BK|      867338|
|        WHITE|      598915|
|        BLACK|      389161|
|           BL|      348480|
|         GREY|      301436|
|           RD|      194491|
|         BLUE|      136469|
|        SILVE|      129919|
|        BROWN|      129532|
|          RED|       98815|
|           GR|       72157|
|           TN|       36762|
|        OTHER|       33153|
|           BR|       32054|
|          BLK|       31434|
|        GREEN|       25426|
|           YW|       24872|
|           GL|       21431|
+-------------+------------+
only showing top 20 rows
```

Answer :

- WH is the most ticket issued vehicle with 1100571 tickets issued.

The vehicle colour column was used to find which coloured vehicle was most frequently ticketed.

```
spkObj.stop()
```

Based on a K-Means algorithm, please try to answer the following question: Given a Black vehicle parking illegally at 34510, 10030, 34050 (street codes). What is the probability that it will get an ticket? (very rough prediction). (20 pts)

```
spkObj = SparkSession.builder \
.appName('vekal_Assignment_2') \
.master('local[*]') \
.config('spark.sql.execution.arrow.pyspark.enabled', True) \
.config('spark.sql.session.timeZone', 'UTC') \
.config('spark.driver.memory','32G') \
.config('spark.ui.showConsoleProgress', True) \
.config('spark.sql.repl.eagerEval.enabled', True) \
.getOrCreate()

spkObj
```

```
<pyspark.sql.session.SparkSession at 0x7f91a7f91f30>
```

Reading the Data

```
QUE1_NYC = spkObj.read.format("csv").option("header",
"true").option("inferSchema","true").load('/content/drive/MyDrive/
Parking_Violations_Issued_-
_Fiscal_Year_2024_20240411.csv').select('Street Code1', 'Street
Code2', 'Street Code3', 'Vehicle Color')

QUE1_NYC = QUE1_NYC.select(QUE1_NYC['Street Code1'].cast('float'),
QUE1_NYC['Street Code2'].cast('float'), QUE1_NYC['Street
Code3'].cast('float'), QUE1_NYC['Vehicle Color'])

from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
import pyspark.sql.functions as psf
import numpy as np
```

First I created a List of all possible Black colours. The street code column was added for the requirements of the question. For which spark vector assembler was used. For the k means value of K is 4. Then a function calculates the probability of a black vehicle being in a cluster by using the list I created. Finally the closest cluster is found using Euclidean Distance, and the closest cluster index is returned.

```
def vectorize_street_codes(data_frame):
```

```
    assembler = VectorAssembler(inputCols=["Street Code1", "Street
Code2", "Street Code3"], outputCol="vectorized_street_codes")
    return assembler.transform(data_frame)

def initialize_kmeans(vectorized_data):

    kmeans_model = KMeans(k=4, featuresCol="vectorized_street_codes")
    model_fit =
kmeans_model.fit(vectorized_data.select('vectorized_street_codes'))
    cluster_centers =
np.array(model_fit.clusterCenters()).astype(float)
    return model_fit.transform(vectorized_data).cache(),
cluster_centers

def calculate_black_car_probability(data_frame, black_colors):

    color_distribution = data_frame.groupBy('prediction').agg(
        psf.sum(psf.when(psf.col('Vehicle Color').isin(black_colors),
1)).alias('Black_Count'),
        psf.count('Vehicle Color').alias('Total_Cars')
    ).orderBy('prediction')

    return color_distribution.select(
        'prediction',
        'Black_Count',
        'Total_Cars',
        (psf.col('Black_Count') /
psf.col('Total_Cars')).alias('Probability')
    )


def find_closest_cluster(street_data, cluster_centers):

    street_data = np.array(street_data)

    distances = np.sum((cluster_centers - street_data)**2, axis=1)

    closest_cluster_index = np.argmin(distances)

    return closest_cluster_index


def display_cluster_probability(cluster_center_id, probabilities_df):

    print(f'Cluster ID for given Street Code: {cluster_center_id}')

print("----------------------------------------------------------------
--------")
    print("Probability for that Cluster:")
    probabilities_df.filter(psf.col('prediction') ==
cluster_center_id).show()
```

```
def calculate_and_display_probability(data_frame, black_colors,
street_code):

    vectorized_data = vectorize_street_codes(data_frame)

    clustered_data, cluster_centers =
initialize_kmeans(vectorized_data)

    probabilities_df = calculate_black_car_probability(clustered_data,
black_colors)

    closest_cluster_id = find_closest_cluster(street_code,
cluster_centers)

    display_cluster_probability(closest_cluster_id, probabilities_df)

blackColor=['BLK.', 'Black', 'BC', 'BLAC', 'BK/', 'BLK', 'BK.', 'BCK',
'BK', 'B LAC']
streetCode=[34510, 10030, 34050]
calculate_and_display_probability(QUE1_NYC,blackColor,streetCode )
```

```
Cluster ID for given Street Code (34510, 10030, 34050): 0
-------------------------------------------------------------------------
-
Probability for that Specific Cluster:
+----------+-----------+----------+--------------------+
|prediction|Black_Count|Total_Cars|         Probability|
+----------+-----------+----------+--------------------+
|         0|     327327|   2342418|0.13973893643235324|
+----------+-----------+----------+--------------------+
```

The above model is built for K = 4 for which probability is 0.1476759875634194

```
spkObj.stop()
```

QUESTION 2

```
spkObj =
pyspark.sql.SparkSession.builder.appName("vekal_Assignment_2").getOrCr
eate()

spkObj.conf.set("spark.sql.repl.eagerEval.enabled", True)

QUE2_NBA = spkObj.read.format("csv").option("header",
"true",).option("inferSchema","true").load('/content/drive/MyDrive/
shot_logs.csv')

QUE2_NBA.show(n=1)
```

```
+--------+--------------------+--------+---+-----------+-----------
+------+------------------+---------+--------+---------+---------
+--------+-----------+---------------+-------------------------
+--------------+---+---+------------+---------+
| 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_DEFENDER_PLAYER_ID|
CLOSE_DEF_DIST|FGM|PTS|  player_name|player_id|
+--------+--------------------+--------+---+-----------+-----------
+------+------------------+---------+--------+---------+---------
+--------+-----------+---------------+-------------------------
+--------------+---+---+------------+---------+
|21400899|MAR 04, 2015 - CH...|       A|  W|          24|          1|
1|2024-04-12 01:09:00|      10.8|       2|       1.9|      7.7|
2|       made|  Anderson, Alan|                  101187|
1.3|  1|  2|brian roberts|   203148|
+--------+--------------------+--------+---+-----------+-----------
+------+------------------+---------+--------+---------+---------
+--------+-----------+---------------+-------------------------
+--------------+---+---+------------+---------+
only showing top 1 row
```

Number of Rows and Columns

```
print("Total Number of Rows: " , QUE2_NBA.count())
print("Total Number of Columns: " , len(QUE2_NBA.columns))

Total Number of Rows:  128069
Total Number of Columns:  21
```

Schema

```
QUE2_NBA.printSchema()

root
 |-- GAME_ID: integer (nullable = true)
 |-- MATCHUP: string (nullable = true)
 |-- LOCATION: string (nullable = true)
 |-- W: string (nullable = true)
 |-- FINAL_MARGIN: integer (nullable = true)
 |-- SHOT_NUMBER: integer (nullable = true)
 |-- PERIOD: integer (nullable = true)
 |-- GAME_CLOCK: timestamp (nullable = true)
 |-- SHOT_CLOCK: double (nullable = true)
 |-- DRIBBLES: integer (nullable = true)
 |-- TOUCH_TIME: double (nullable = true)
 |-- SHOT_DIST: double (nullable = true)
 |-- PTS_TYPE: integer (nullable = true)
```

```
 |-- SHOT_RESULT: string (nullable = true)
 |-- CLOSEST_DEFENDER: string (nullable = true)
 |-- CLOSEST_DEFENDER_PLAYER_ID: integer (nullable = true)
 |-- CLOSE_DEF_DIST: double (nullable = true)
 |-- FGM: integer (nullable = true)
 |-- PTS: integer (nullable = true)
 |-- player_name: string (nullable = true)
 |-- player_id: integer (nullable = true)
```

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”. (10 pts)

madeCond and missedCond are used to create a separate dataframe. Where we use the player name and closest defender as a pair. Then I calculated a ratio of total shots made and total shots attempted, by usinfg the same pair mentioned above. We drop the null and duplicate values along the way based on Player ID and Defender ID and display the answer.

```
dataframe = (
    QUE2_NBA
    .groupBy(
        psf.col("player_id").alias("Player ID"),
        psf.col("CLOSEST_DEFENDER_PLAYER_ID").alias("Defender ID")
    )
    .agg(
        psf.sum(psf.when(psf.col("SHOT_RESULT") == "made",
1).otherwise(0)).alias("Scored"),
        psf.sum(psf.when(psf.col("SHOT_RESULT") == "missed",
1).otherwise(0)).alias("Not Scored")
    )
)

dataframe.show(10)
```

```
+---------+-----------+------+----------+
|Player ID|Defender ID|Scored|Not Scored|
+---------+-----------+------+----------+
|   203148|     101179|     0|         1|
```

```
|   202687|     201980|     1|          0|
|     2744|       1717|     0|          2|
|   203469|     202329|     1|          1|
|   201945|     202322|     0|          3|
|   202689|     202699|     6|          8|
|   202689|     203924|     1|          0|
|   203077|       2730|     1|          0|
|   203077|     201584|     2|          0|
|   202362|     201188|     2|          0|
+---------+-----------+------+-----------+
only showing top 10 rows
```

```
dataframe = dataframe.withColumn(
    "HR",
    psf.col("Scored") / (psf.col("Scored") + psf.col("Not Scored"))
)
dataframe.show(10)
```

```
+---------+-----------+------+----------+-------------------+
|Player ID|Defender ID|Scored|Not Scored|                 HR|
+---------+-----------+------+----------+-------------------+
|   203148|     101179|     0|         1|                0.0|
|   202687|     201980|     1|         0|                1.0|
|     2744|       1717|     0|         2|                0.0|
|   203469|     202329|     1|         1|                0.5|
|   201945|     202322|     0|         3|                0.0|
|   202689|     202699|     6|         8|0.42857142857142855|
|   202689|     203924|     1|         0|                1.0|
|   203077|       2730|     1|         0|                1.0|
|   203077|     201584|     2|         0|                1.0|
|   202362|     201188|     2|         0|                1.0|
+---------+-----------+------+----------+-------------------+
only showing top 10 rows
```

```
dataframe = dataframe.filter(psf.col("HR").isNotNull())

dataframe = dataframe.dropDuplicates(subset=["Player ID", "HR"])

tempFrame = dataframe.groupBy("Player
ID").agg(psf.min("HR").alias("HR"))

dataframe = dataframe.join(tempFrame, ["Player ID",
"HR"]).select("Player ID", "Defender ID")


dataframe = dataframe.join(
    QUE2_NBA,
    (QUE2_NBA["player_id"] == dataframe["Player ID"]) &
(QUE2_NBA["CLOSEST_DEFENDER_PLAYER_ID"] == dataframe["Defender ID"])
```

```
).withColumn("Player Name", col("player_name")).withColumn("Most
Unwanted Defender", col("CLOSEST_DEFENDER"))

dataframe = dataframe.dropDuplicates(["Player ID", "Defender ID"])

dataframe.select("Player Name", "Most Unwanted Defender").show(10)
```

```
+--------------+----------------------+
|   Player Name|Most Unwanted Defender|
+--------------+----------------------+
| kevin garnett|           Exum, Dante|
|   kobe bryant|        Anderson, Kyle|
|    tim duncan|        Roberts, Brian|
|  vince carter|       Crawford, Jamal|
|dirk nowtizski|           Hickson, JJ|
|   paul pierce|         Waiters, Dion|
|  andre miller|       Splitter, Tiago|
|  shawn marion|     Tolliver, Anthony|
|   jason terry|          Lopez, Brook|
| manu ginobili|      Bennett, Anthony|
+--------------+----------------------+
only showing top 10 rows
```

Answer:

From the above table, we can see that most unwanted defender for each of the player. So, if Kevin Garnett is the shooter, the most the unwanted defender is the Exum, Dante

```
spkObj.stop()
```

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. (10 pts)

```
spkObj=
pyspark.sql.SparkSession.builder.appName("vekal_Assignment_2").getOrCr
eate()
spkObj.conf.set("spark.sql.repl.eagerEval.enabled", True)
```

```
QUE2_NBA = spkObj.read.format("csv").option("header",
"true",).option("inferSchema","true").load('/content/drive/MyDrive/
shot_logs.csv').select("player_name","SHOT_DIST","CLOSE_DEF_DIST",
"SHOT_CLOCK", "SHOT_RESULT").na.drop()
```

Transforming the column into binary 1 for successful shot 0 for missed. (Basically One-Hot Encoding)

```
QUE2_NBA = QUE2_NBA.withColumn('SHOT_RESULT',
psf.when(psf.col('SHOT_RESULT') == 'made',
1).otherwise(0).cast('float'))
comfortMatrix = ["SHOT_DIST", "CLOSE_DEF_DIST", "SHOT_CLOCK"]
```

Chnaging every column data type to float.

```
for i in comfortMatrix:
    QUE2_NBA = QUE2_NBA.withColumn(i, psf.col(i).cast("float"))
```

The value of K is 4, and we fit the model to the data. The data is filtered for the required player. The fitted model is then used to predict the cluster assignment. Since the K means clustering algorithm assigns clusters randomly every time we run the code we might get different clusters. But few players will belong to the same cluster only. In this example except for Chris Paul, everyone else will be in the same zone.

```
vecAssembler = VectorAssembler(inputCols=comfortMatrix,
outputCol="Zones")
QUE2_NBA = vecAssembler.transform(QUE2_NBA).select('player_name',
'Zones', 'SHOT_RESULT')


kmeans = KMeans(k=4, featuresCol="Zones")
kmeansFitData = kmeans.fit(QUE2_NBA)

pD = QUE2_NBA.filter(QUE2_NBA['player_name'].isin(['james harden',
'chris paul', 'stephen curry', 'lebron james']))

pred = kmeansFitData.transform(pD).select('player_name', 'prediction',
'SHOT_RESULT')
```

Following code snippet basically calculates the average shot result for every player. Then max avg shot result for each player is identified using join operation. Only where the avg is highest.

```
pred.createOrReplaceTempView("player_zones")

result = spkObj.sql("""SELECT player_name, prediction,
AVG(SHOT_RESULT) AS avgShotResult FROM player_zones GROUP BY
player_name, prediction ORDER BY player_name, prediction """)
```

```
maxAvgShot =
result.groupBy("player_name").agg(psf.max("avgShotResult").alias("maxA
vgShotResult"))

best_Zone = result.alias("f1").join(maxAvgShot.alias("f2"),
(psf.col("f1.player_name") == psf.col("f2.player_name")) &
(psf.col("f1.avgShotResult") == psf.col("f2.maxAvgShotResult")))

best_Zone = best_Zone.select("f1.*")
best_Zone.show()
```

```
+-------------+----------+------------------+
|  player_name|prediction|     avgShotResult|
+-------------+----------+------------------+
| lebron james|         1|0.6613545816733067|
|   chris paul|         0|0.5563380281690141|
| james harden|         1|0.5604395604395604|
|stephen curry|         1|0.6350710900473934|
+-------------+----------+------------------+
```

Answer:

Zone-1 corresponds to a prediction value of 0, Zone-2 to 1, Zone-3 to 2, and Zone-4 to 3 in the 'prediction' column of the dataset. To analyze each player's comfort zone, the data was organized by player and by zone, and the average score was computed for each grouping.

```
spkObj.stop()
```