ECC ASSIGNMENT 2

QUESTION 1. NY PARKING VIOLATIONS

Loading the dataset in spark dataframe -

```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("ECC-Assignment-2").getOrCreate()

df = spark.read.csv("/Users/aditisonawane/Desktop/ECC/a2/Parking_Violations_Issued_-_Fiscal_Year_2023.csv",
                    header=True, inferSchema=True)

```

Visualizing data -

```
In [41]: df.head(5)
```

```
Out[41]: [Row(Summons Number=1484697303, Plate ID='JER1863', Registration State='NY', Plate Type='PAS', Issue Date='06/10/202
2', Violation Code=67, Vehicle Body Type='SDN', Vehicle Make='TOYOT', Issuing Agency='P', Street Code1=34330, Street
Code2=179, Street Code3=0, Vehicle Expiration Date=20221210, Violation Location=10, Violation Precinct=10, Issuer Pre
cinct=1, Issuer Code=160195, Issuer Command='0001', Issuer Squad='0000', Violation Time='1037A', Time First Observed=
None, Violation County='NY', Violation In Front Of Or Opposite='F', House Number=None, Street Name='W 28TH ST', Inter
secting Street='CHELSEA PK', Date First Observed=0, Law Section=408, Sub Division='E3', Violation Legal Code=None, Da
ys Parking In Effect    ='BBBBBBB', From Hours In Effect='ALL', To Hours In Effect='ALL', Vehicle Color='BLK', Unregi
stered Vehicle?=0, Vehicle Year=2004, Meter Number='-', Feet From Curb=0, Violation Post Code=None, Violation Descrip
tion=None, No Standing or Stopping Violation=None, Hydrant Violation=None, Double Parking Violation=None),
 Row(Summons Number=1484697315, Plate ID='KEV4487', Registration State='NY', Plate Type='PAS', Issue Date='06/13/202
2', Violation Code=51, Vehicle Body Type='SUBN', Vehicle Make='JEEP', Issuing Agency='K', Street Code1=34310, Street
Code2=16400, Street Code3=11010, Vehicle Expiration Date=20220908, Violation Location=10, Violation Precinct=10, Issu
er Precinct=1, Issuer Code=160195, Issuer Command='0001', Issuer Squad='0000', Violation Time='1045A', Time First Obs
erved=None, Violation County='NY', Violation In Front Of Or Opposite='O', House Number='436', Street Name='27TH DR',
Intersecting Street=None, Date First Observed=0, Law Section=408, Sub Division='F3', Violation Legal Code=None, Days
Parking In Effect    ='BBBBBBB', From Hours In Effect='ALL', To Hours In Effect='ALL', Vehicle Color='GRAY', Unregist
ered Vehicle?=0, Vehicle Year=2017, Meter Number='-', Feet From Curb=0, Violation Post Code=None, Violation Descripti
on=None, No Standing or Stopping Violation=None, Hydrant Violation=None, Double Parking Violation=None),
 Row(Summons Number=1484697625, Plate ID='H73NYD', Registration State='NJ', Plate Type='PAS', Issue Date='06/19/202
2', Violation Code=63, Vehicle Body Type='SDN', Vehicle Make='JEEP', Issuing Agency='N', Street Code1=30640, Street C
ode2=13015, Street Code3=28540, Vehicle Expiration Date=0, Violation Location=1, Violation Precinct=1, Issuer Precinc
t=1, Issuer Code=161315, Issuer Command='0001', Issuer Squad='0000', Violation Time='1116A', Time First Observed=Non
e, Violation County='NY', Violation In Front Of Or Opposite=None, House Number='109', Street Name='SOUTH ST', Interse
cting Street=None, Date First Observed=0, Law Section=408, Sub Division='F3', Violation Legal Code=None, Days Parking
In Effect    ='BBBBBBB', From Hours In Effect='ALL', To Hours In Effect='ALL', Vehicle Color='GRAY', Unregistered Veh
icle?=0, Vehicle Year=0, Meter Number='-', Feet From Curb=0, Violation Post Code=None, Violation Description=None, No
Standing or Stopping Violation=None, Hydrant Violation=None, Double Parking Violation=None),
 Row(Summons Number=1484697674, Plate ID='GJC9296', Registration State='NY', Plate Type='PAS', Issue Date='06/19/202
2', Violation Code=63, Vehicle Body Type='SUBN', Vehicle Make='LEXUS', Issuing Agency='N', Street Code1=30640, Street
Code2=13015, Street Code3=28540, Vehicle Expiration Date=20230820, Violation Location=1, Violation Precinct=1, Issuer
Precinct=1, Issuer Code=161315, Issuer Command='0001', Issuer Squad='0000', Violation Time='1052A', Time First Observ
ed=None, Violation County='NY', Violation In Front Of Or Opposite=None, House Number='109', Street Name='SOUTH ST', I
ntersecting Street=None, Date First Observed=0, Law Section=408, Sub Division='F3', Violation Legal Code=None, Days P
arking In Effect    ='BBBBBBB', From Hours In Effect='ALL', To Hours In Effect='ALL', Vehicle Color=None, Unregistere
d Vehicle?=0, Vehicle Year=0, Meter Number='-', Feet From Curb=0, Violation Post Code=None, Violation Description=Non
e, No Standing or Stopping Violation=None, Hydrant Violation=None, Double Parking Violation=None),
 Row(Summons Number=1484697686, Plate ID='M51PUV', Registration State='NJ', Plate Type='PAS', Issue Date='06/19/202
2', Violation Code=63, Vehicle Body Type='SDN', Vehicle Make='HYUND', Issuing Agency='N', Street Code1=30640, Street
```

Printing schema -

```
In [2]: 1 df.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)
```

Count of rows in the dataframe -

```
In [42]: 1 print(df.count())
```

```
[Stage 70:==========================>                              (8 + 8) / 17]
11535314
```

Checking NULL values in each column of df -

In [43]:
```
for column in df.columns:
    print(column, ":","Null values: ", df.filter(df[column].isNull()).count())
```

```
Summons Number : Null values:  0
Plate ID : Null values:  1
Registration State : Null values:  0
Plate Type : Null values:  0
Issue Date : Null values:  0
Violation Code : Null values:  0
Vehicle Body Type : Null values:  27240
Vehicle Make : Null values:  9690
Issuing Agency : Null values:  0
Street Code1 : Null values:  0
Street Code2 : Null values:  0
Street Code3 : Null values:  0
Vehicle Expiration Date : Null values:  0
Violation Location : Null values:  5349526
Violation Precinct : Null values:  0
Issuer Precinct : Null values:  0
Issuer Code : Null values:  0
Issuer Command : Null values:  5344155
Issuer Squad : Null values:  5741691
Violation Time : Null values:  123
Time First Observed : Null values:  10881550
Violation County : Null values:  42634
```

```
Violation In Front Of Or Opposite : Null values:  5404277

House Number : Null values:  5446068

Street Name : Null values:  1310

Intersecting Street : Null values:  5369958

Date First Observed : Null values:  0

Law Section : Null values:  0

Sub Division : Null values:  1681

Violation Legal Code : Null values:  6191157

Days Parking In Effect     : Null values:  5444990

From Hours In Effect : Null values:  7964864

To Hours In Effect : Null values:  7964879

Vehicle Color : Null values:  1032007
```


```
To Hours In Effect : Null values:  7964879

Vehicle Color : Null values:  1032007
```


```
Unregistered Vehicle? : Null values:  11297515

Vehicle Year : Null values:  0

Meter Number : Null values:  10238122

Feet From Curb : Null values:  0

Violation Post Code : Null values:  5979457

Violation Description : Null values:  238417

No Standing or Stopping Violation : Null values:  11535314

Hydrant Violation : Null values:  11535314
[Stage 199:=======================================>                (12 + 5) / 17]
Double Parking Violation : Null values:  11535314
```

Q1.1 -
I have calculated the maximum number of tickets issued based on two factors - Issue date and Violation time.

In case of the Issue Date column, I have converted it from String to Date format. Then, I have grouped the rows by date and count of the number of rows in each group.
Output - 10 dates with the highest number of tickets issued.

In [44]:
```
#1.1
from pyspark.sql.functions import to_date, count
from pyspark.sql.types import DateType

df = df.withColumn("Issue Date", to_date(df["Issue Date"], "MM/dd/yyyy").cast(DateType()))

ncount = df.groupBy("Issue Date").agg(count("*").alias("Total number of tickets issued"))
ncount = ncount.orderBy("Total number of tickets issued", ascending=False)
ncount.show(10)
```

```
[Stage 202:=====================================================>   (16 + 1) / 17]
```

```
+----------+-----------------------------+
|Issue Date|Total number of tickets issued|
+----------+-----------------------------+
|2022-08-04|                        66726|
|2022-08-05|                        65393|
|2022-08-02|                        64876|
|2022-06-30|                        64846|
|2022-07-19|                        64815|
|2022-11-25|                        64411|
|2022-08-11|                        64192|
|2022-08-18|                        63975|
|2022-07-12|                        63780|
|2022-07-15|                        63646|
+----------+-----------------------------+
only showing top 10 rows
```

For the Violation Time, similar to Issue Date, I have used Violation time and converted it to 12hr time format and displayed top 10 times when the highest number of tickets are issued.

In [45]:
```
#1.1
from pyspark.sql.functions import col, concat, lit, substring

vtcount = df.groupBy("Violation Time").agg(count("*").alias("Total number of tickets issued"))
vtcount = vtcount.orderBy("Total number of tickets issued", ascending=False)

vtcount = vtcount.withColumn("Time", concat(substring(col("Violation Time"), 1, 2), lit(":"),
                                            substring(col("Violation Time"), 3, 2), lit(" AM")))
vtcount = vtcount.drop("Violation Time")
vtcount.show(10,False)

```

```
[Stage 205:=====================================================>   (16 + 1) / 17]
```

```
+-----------------------------+--------+
|Total number of tickets issued|Time    |
+-----------------------------+--------+
|22958                        |08:36 AM|
|21772                        |08:39 AM|
|21712                        |09:06 AM|
|21588                        |08:40 AM|
|21585                        |08:38 AM|
|21074                        |08:37 AM|
|20883                        |08:41 AM|
|20765                        |09:08 AM|
|20660                        |08:42 AM|
|20564                        |11:41 AM|
+-----------------------------+--------+
only showing top 10 rows
```

Q1.2 - The most common years and types of cars to be ticketed can be calculated either by using the Vehicle year and Vehicle body type columns or the Issue date and Vehicle body type columns.

In the below approach, I have grouped rows by Vehicle Year and Vehicle body type and counted the maximum number of tickets issued. To avoid the 0 entries in the Vehicle year column, I have added the condition on line number 4.

In [48]:
```
1 #1.2
2
3 vycount = df.groupBy('Vehicle Year','Vehicle Body Type').agg(count("*").alias("Total number of tickets issued"))
4 vycount = vycount.filter(vycount['Vehicle Year']!=0)
5 vycount = vycount.orderBy("Total number of tickets issued", ascending=False)
6 vycount.show(10)
```

```
[Stage 211:==================================================>   (16 + 1) / 17]

+------------+-----------------+-----------------------------+
|Vehicle Year|Vehicle Body Type|Total number of tickets issued|
+------------+-----------------+-----------------------------+
|        2021|             SUBN|                       574105|
|        2020|             SUBN|                       474581|
|        2022|             SUBN|                       450725|
|        2019|             SUBN|                       427511|
|        2018|             SUBN|                       330480|
|        2017|             SUBN|                       268854|
|        2016|             SUBN|                       216894|
|        2015|             SUBN|                       206983|
|        2017|             4DSD|                       203948|
|        2018|             4DSD|                       199032|
+------------+-----------------+-----------------------------+
only showing top 10 rows
```

Similar to the above approach, here I have used the year from the Issue Date column and the Vehicle body type and printed the top 10 rows with the maximum number of tickets issued.

In [47]:
```
1 #1.2
2 from pyspark.sql.functions import year, count
3
4 total = df.select(year('Issue Date').alias('Year'), 'Vehicle Body Type').groupBy('Year', 'Vehicle Body Type')
5         .agg(count('*').alias('Total Number of Tickets')).orderBy('Total Number of Tickets', ascending=False)
6
7 total.show(10)
8
```

```
[Stage 208:==================================================>   (16 + 1) / 17]

+----+-----------------+----------------------+
|Year|Vehicle Body Type|Total Number of Tickets|
+----+-----------------+----------------------+
|2022|             SUBN|               3909852|
|2022|             4DSD|               2432083|
|2023|             SUBN|               1050497|
|2022|              VAN|                676542|
|2023|             4DSD|                635016|
|2022|               UT|                317138|
|2022|             PICK|                258757|
|2022|             DELV|                196646|
|2022|               SD|                192466|
|2023|              VAN|                183923|
+----+-----------------+----------------------+
only showing top 10 rows
```

Q1.3 - To answer the question - Where are tickets most commonly issued, I have selected the Violation Location column from the df and filtered the rows with null entries. Then, I counted the number of rows for each violation location and printed the top 10 violation locations with the highest number of tickets issued.

In [49]:
```
#1.3
wcount = df.groupBy("Violation Location").agg(count("*").alias("Total number of tickets issued"))
wcount = wcount.filter(col("Violation Location").isNotNull())
wcount = wcount.orderBy("Total number of tickets issued", ascending=False)
wcount.show(10)

```

```
[Stage 214:================================================>      (15 + 2) / 17]
```

```
+------------------+-----------------------------+
|Violation Location|Total number of tickets issued|
+------------------+-----------------------------+
|                19|                       282466|
|                13|                       254057|
|                 6|                       224686|
|               114|                       221523|
|                14|                       190012|
|                18|                       176733|
|                 9|                       162228|
|                 1|                       152429|
|               109|                       137833|
|               115|                       127523|
+------------------+-----------------------------+
only showing top 10 rows
```

Output - The Violation Location 19 has the highest number of tickets issued.

Q1.4 - The color of the vehicle that is most likely to get a ticket can be calculated using the Vehicle color column. I have used aggregate function count to calculate the total number of rows for each vehicle color and displayed the top vehicles colors with the maximum number of tickets issued.

In [65]:
```
#1.4
ccount = df.groupBy("Vehicle Color").agg(count("*").alias("Total number of tickets issued"))
ccount = ccount.filter(col("Vehicle Color").isNotNull())
ccount = ccount.orderBy("Total number of tickets issued", ascending=False)
ccount.show(10)

```

```
[Stage 227:================================================>    (16 + 1) / 17]
```

```
+-------------+-----------------------------+
|Vehicle Color|Total number of tickets issued|
+-------------+-----------------------------+
|           GY|                      2275457|
|           WH|                      2055818|
|           BK|                      1992788|
|           BL|                       760235|
|        WHITE|                       671757|
|           RD|                       435989|
|        BLACK|                       424056|
|         GREY|                       308993|
|        SILVE|                       151063|
|         BLUE|                       150435|
+-------------+-----------------------------+
only showing top 10 rows
```

Output - The Grey color vehicles have a maximum number of tickets issued.

Q1.5 - KMEANS

After analyzing the Vehicle color column, it can be concluded that there are multiple variations of the Black color vehicles.

In [72]:

```
# Kmeans
from pyspark.sql.functions import col, lower
unique = df.select(col("Vehicle Color")).distinct().filter(col("Vehicle Color").startswith("B"))
unique.show(unique.count(), truncate=False)
```

```
[Stage 272:=========================>                          (8 + 8) / 17]
```

```
+-------------+
|Vehicle Color|
+-------------+
|BKMR         |
|BKGL         |
|BUR          |
|BLTN         |
|BLK          |
|BRWN         |
|BRRD         |
|BKBR         |
|BIEGE        |
|BW           |
|BRBL         |
|BKI          |
|BRCDF        |
|BRN          |
|BLACK        |
|BURGU        |
|BUS          |
|BUE          |
|BLAAC        |
|BKWH         |
|BRGL         |
|BU           |
|BKBK         |
|BKTN         |
|BRONZ        |
|BLB          |
|BL/          |
|BEIGE        |
|BEGE         |
|BLPR         |
```

The black color codes array is created with all possible variations of black vehicle color. The number of clusters selected is 3. The probability is calculated based on the all vehicle colors and the street codes mentioned in the question. The probabilities are calculated in accordance to the black color code array.

Output - 1. The feature values with the cluster number for all vehicle colors and street codes are printed in the first table below.
2. Test data is passed to the k-means model for the given street code values and black vehicle colors. For each cluster, probability is printed in the second table below.
3. Finally the condition - Black vehicle parking illegally at 34510, 10030, 34050, the probability that a ticket is issued is 0.2. And this type of cars are street codes that are present in 2nd cluster number.

In [106]:

```
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler
from pyspark.sql.functions import col, when

black_color_codes = ['BLK', 'BK.', 'BCK', 'BK', 'BLK.', 'Black', 'BC', 'BLAC', 'BK/', 'B LAC','BLAAC',
                     'BLAK','B K','BLCK','BACK','BL.','BLC','BLA','BLACL','B L','B LK','BLACK']

df2 = df.select('Vehicle Color', 'Street Code1', 'Street Code2', 'Street Code3')
va = VectorAssembler(inputCols=['Street Code1', 'Street Code2', 'Street Code3'], outputCol='features')
df2 = va.transform(df2).select('Vehicle Color', 'Street Code1', 'Street Code2', 'Street Code3','features')

kmeans = KMeans(k=3)
model = kmeans.fit(df2.select('features'))

td = model.transform(df2)
td.show()

probabilities = td.groupBy('prediction').agg(count(when(col('Vehicle Color').isin(black_color_codes),1))
                                             .alias('Black cars'),count('Vehicle Color').alias('Total'))
                                            .orderBy('prediction')
probabilities = probabilities.select('prediction','Black cars','Total',(col('Black cars')/col('Total'))
                                     .alias('Prob'))

print(probabilities.show())

test_df = spark.createDataFrame([[34510, 10030, 34050]], ['Street Code1', 'Street Code2', 'Street Code3'])
test_df = va.transform(test_df)
label = model.transform(test_df).select('prediction').collect()[0]['prediction']

print(label)
print(probabilities.filter(col('prediction') == label).show())


```

```
+-------------+------------+------------+------------+--------------------+----------+
|Vehicle Color|Street Code1|Street Code2|Street Code3|            features|prediction|
+-------------+------------+------------+------------+--------------------+----------+
|          BLK|       34330|         179|           0| [34330.0,179.0,0.0]|         2|
|         GRAY|       34310|       16400|       11010|[34310.0,16400.0,...|         2|
|         GRAY|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|         null|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|         BLUE|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|         null|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|         BLUE|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|        WHITE|       11585|       26390|       15010|[11585.0,26390.0,...|         2|
|          BLK|           0|           0|           0|           (3,[],[])|         1|
|          RED|       33190|       25190|       31990|[33190.0,25190.0,...|         2|
|           GY|       33190|       25190|       31990|[33190.0,25190.0,...|         2|
|         GRAY|       30640|       24050|           0|[30640.0,24050.0,...|         2|
|          WHT|       30640|       13015|       28540|[30640.0,13015.0,...|         2|
|          BLK|           0|           0|           0|           (3,[],[])|         1|
|        BLACK|       33340|           0|           0|   [33340.0,0.0,0.0]|         1|
|          RED|       34440|           0|           0|   [34440.0,0.0,0.0]|         2|
|          WHT|           0|           0|           0|           (3,[],[])|         1|
|          WHT|           0|           0|           0|           (3,[],[])|         1|
|        BLACK|       11210|       22695|           0|[11210.0,22695.0,...|         1|
|        WHITE|       23904|       25680|       21950|[23904.0,25680.0,...|         2|
+-------------+------------+------------+------------+--------------------+----------+
only showing top 20 rows
```

```
+----------+----------+-------+-------------------+
|prediction|Black cars|  Total|               Prob|
+----------+----------+-------+-------------------+
|         0|    342934|1391343|0.24647696506181438|
|         1|   1334197|5226789| 0.2552613086160547|
|         2|    828135|3885175|0.21315256069546418|
+----------+----------+-------+-------------------+

None
2
```

```
[Stage 675:=======================================>                 (12 + 5) / 17]
```

```
+----------+----------+-------+-------------------+
|prediction|Black cars|  Total|               Prob|
+----------+----------+-------+-------------------+
|         2|    828135|3885175|0.21315256069546418|
+----------+----------+-------+-------------------+

None
```

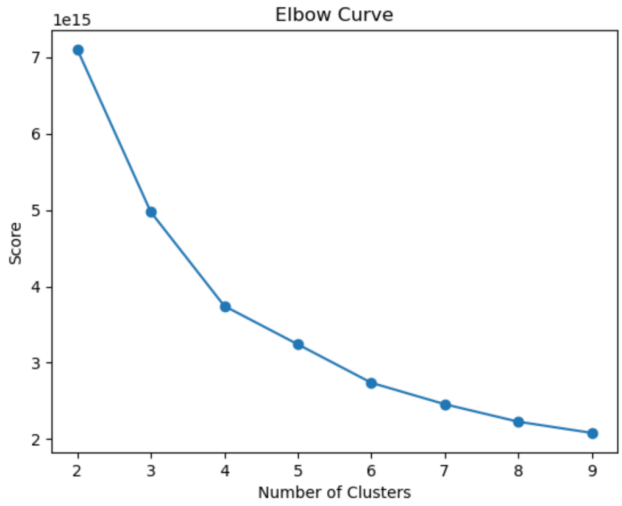
The silhouette score is calculated to evaluate the clustering.
Output - Silhouette score = 0.6 infers that the number of clusters selected is good.
The cluster values have been printed below.

In [108]:
```
from pyspark.ml.evaluation import ClusteringEvaluator

evaluator = ClusteringEvaluator()

silhouette = evaluator.evaluate(td)
print("Silhouette with squared euclidean distance = " + str(silhouette))

centers = model.clusterCenters()
print("Cluster Centers: ")
for center in centers:
    print(center)
```

```
Silhouette with squared euclidean distance = 0.6398167912595234
Cluster Centers:
[51903.41498312 53190.09694159 53098.34032493]
[1886.27459238  919.48245195  439.49030395]
[27085.43106032 21832.8069534  22121.2570675 ]
```

Performance of different values of k can be evaluated using the Elbow method.
Output - From the plot we can infer that number of clusters = 3 can be considered ideal.

In [15]:
```
import pandas as pd
import pylab as pl
kdf = pd.DataFrame(errors[2:])
kdf.columns = ["errors"]
new_col = range(2,10)
kdf.insert(0, 'cluster', new_col)
pl.plot(kdf.cluster, kdf.errors, '-o')
pl.xlabel('Number of Clusters')
pl.ylabel('Score')
pl.title('Elbow Curve')
pl.show()
```



QUESTION 2. NBA Shot Logs

Loading the dataset in py-spark dataframe and analyzing the data in it.

In [52]:
```
from pyspark.sql import SparkSession

spark = SparkSession.builder.appName("ECC-Assignment-2.2").getOrCreate()

df = spark.read.csv("/Users/aditisonawane/Desktop/ECC/a2/shot_logs.csv", header=True, inferSchema=True)


```

In [2]:
```
df.head(5)
```

Out[2]:
```
[Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=1, PERIO
D=1, GAME_CLOCK=datetime.datetime(2023, 4, 11, 1, 9), SHOT_CLOCK=10.8, DRIBBLES=2, TOUCH_TIME=1.9, SHOT_DIST=7.7, PTS
_TYPE=2, SHOT_RESULT='made', CLOSEST_DEFENDER='Anderson, Alan', CLOSEST_DEFENDER_PLAYER_ID=101187, CLOSE_DEF_DIST=1.
3, FGM=1, PTS=2, player_name='brian roberts', player_id=203148),
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=2, PERIO
D=1, GAME_CLOCK=datetime.datetime(2023, 4, 11, 0, 14), SHOT_CLOCK=3.4, DRIBBLES=0, TOUCH_TIME=0.8, SHOT_DIST=28.2, PT
S_TYPE=3, SHOT_RESULT='missed', CLOSEST_DEFENDER='Bogdanovic, Bojan', CLOSEST_DEFENDER_PLAYER_ID=202711, CLOSE_DEF_DI
ST=6.1, FGM=0, PTS=0, player_name='brian roberts', player_id=203148),
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=3, PERIO
D=1, GAME_CLOCK=datetime.datetime(2023, 4, 11, 0, 0), SHOT_CLOCK=None, DRIBBLES=3, TOUCH_TIME=2.7, SHOT_DIST=10.1, PT
S_TYPE=2, SHOT_RESULT='missed', CLOSEST_DEFENDER='Bogdanovic, Bojan', CLOSEST_DEFENDER_PLAYER_ID=202711, CLOSE_DEF_DI
ST=0.9, FGM=0, PTS=0, player_name='brian roberts', player_id=203148),
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=4, PERIO
D=2, GAME_CLOCK=datetime.datetime(2023, 4, 11, 11, 47), SHOT_CLOCK=10.3, DRIBBLES=2, TOUCH_TIME=1.9, SHOT_DIST=17.2,
PTS_TYPE=2, SHOT_RESULT='missed', CLOSEST_DEFENDER='Brown, Markel', CLOSEST_DEFENDER_PLAYER_ID=203900, CLOSE_DEF_DIST
=3.4, FGM=0, PTS=0, player_name='brian roberts', player_id=203148),
 Row(GAME_ID=21400899, MATCHUP='MAR 04, 2015 - CHA @ BKN', LOCATION='A', W='W', FINAL_MARGIN=24, SHOT_NUMBER=5, PERIO
D=2, GAME_CLOCK=datetime.datetime(2023, 4, 11, 10, 34), SHOT_CLOCK=10.9, DRIBBLES=2, TOUCH_TIME=2.7, SHOT_DIST=3.7, P
TS_TYPE=2, SHOT_RESULT='missed', CLOSEST_DEFENDER='Young, Thaddeus', CLOSEST_DEFENDER_PLAYER_ID=201152, CLOSE_DEF_DIS
T=1.1, FGM=0, PTS=0, player_name='brian roberts', player_id=203148)]
```

Checking for NULL entries in the dataframe and removing if any -

In [2]:
```
for column in df.columns:
    print(column, ":", "Null values: ", df.filter(df[column].isNull()).count())
```

```
GAME_ID : Null values:  0
MATCHUP : Null values:  0
LOCATION : Null values:  0
W : Null values:  0
FINAL_MARGIN : Null values:  0
SHOT_NUMBER : Null values:  0
PERIOD : Null values:  0
GAME_CLOCK : Null values:  0
SHOT_CLOCK : Null values:  5567
DRIBBLES : Null values:  0
TOUCH_TIME : Null values:  0
SHOT_DIST : Null values:  0
PTS_TYPE : Null values:  0
SHOT_RESULT : Null values:  0
CLOSEST_DEFENDER : Null values:  0
CLOSEST_DEFENDER_PLAYER_ID : Null values:  0
CLOSE_DEF_DIST : Null values:  0
FGM : Null values:  0
PTS : Null values:  0
player_name : Null values:  0
player_id : Null values:  0
```

In [53]:
```
df = df.na.drop(subset=["SHOT_CLOCK"])
```

2.1 - In this question we have to calculate the fear score for each player against their defenders and find the most unwanted defender. I have calculated the fear score by summing the entries of 'missed' in column 'Shot Result' and they found the average of each missed shot by total shots missed by each player against each defender. Further, I have found the most unwanted defender based on the fear scores.

In [13]:

```
from pyspark.sql.functions import col, when, sum, count, rank
from pyspark.sql.window import Window

df2 = df.groupBy('player_name', 'CLOSEST_DEFENDER')
hit = df2.agg(sum(when(col('SHOT_RESULT') == 'missed', 1).otherwise(0)).alias('Missed Shots'),
              count('SHOT_RESULT').alias('Total shots missed'))
fear_scoredf = hit.withColumn('Fear Score', when(col('Total shots missed') == 0, 0)
                              .otherwise(col('Missed Shots')/col('Total shots missed')))
fear_scoredf.show()

w = Window.partitionBy('player_name').orderBy(col('Missed Shots').desc())
most_unwanted_defender = fear_scoredf.withColumn('rank', rank().over(w)).filter(col('rank') == 1)
                                     .select('player_name', 'CLOSEST_DEFENDER')

most_unwanted_defender.show()
```

```
+----------------+------------------+------------+------------------+-------------------+
|     player_name|  CLOSEST_DEFENDER|Missed Shots|Total shots missed|        Fear Score|
+----------------+------------------+------------+------------------+-------------------+
|    al jefferson| Hardaway Jr., Tim|           1|                 1|                1.0|
|     cody zeller|     Price, Ronnie|           1|                 1|                1.0|
|       gary neal|     Beal, Bradley|           0|                 3|                0.0|
|       gary neal|     Smart, Marcus|           4|                 4|                1.0|
|gerald henderson|    Bazemore, Kent|           2|                 2|                1.0|
|    kemba walker|     Williams, Lou|           1|                 2|                0.5|
|lance stephenson|    Fournier, Evan|           2|                 2|                1.0|
|  gordon hayward|Aldridge, LaMarcus|           2|                 4|                0.5|
|  gordon hayward|    Bazemore, Kent|           0|                 1|                0.0|
|   trevor booker|   Thompson, Jason|           0|                 3|                0.0|
|   trevor booker|    Frye, Channing|           1|                 2|                0.5|
|     enes kanter|   Chandler, Tyson|           5|                 6|0.8333333333333334|
|      dante exum|      Williams, Mo|           3|                 3|                1.0|
|      jon ingles|     Jack, Jarrett|           0|                 1|                0.0|
|      jon ingles|     Williams, Lou|           4|                 5|                0.8|
|     rudy gobert|       Foye, Randy|           0|                 1|                0.0|
|     rudy gobert|      Wade, Dwyane|           0|                 1|                0.0|
|   carlos boozer|     Jefferson, Al|           0|                 2|                0.0|
+----------------+------------------+------------+------------------+-------------------+
```

```
+----------------+------------------+
|     player_name|  CLOSEST_DEFENDER|
+----------------+------------------+
|    aaron brooks|     Pressey, Phil|
|    aaron gordon|      Ibaka, Serge|
| al farouq aminu|   Faried, Kenneth|
|      al horford|    Gortat, Marcin|
|    al jefferson|   Vucevic, Nikola|
|   alan anderson|      Korver, Kyle|
|   alan anderson|  Sefolosha, Thabo|
|     alan crabbe|       Smith, J.R.|
|        alex len|       Gasol, Marc|
|        alex len|   Jordan, DeAndre|
|   alexis ajinca|Stoudemire, Amar'e|
|   alexis ajinca| Perkins, Kendrick|
|      alonzo gee|         Gay, Rudy|
|      alonzo gee| Muhammad, Shabazz|
|      alonzo gee|  Vasquez, Greivis|
|      alonzo gee| Williams, Derrick|
|amare stoudemire|      Lopez, Brook|
|    amir johnson|     Millsap, Paul|
|  andre drummond|   Mozgov, Timofey|
|  andre drummond| Thompson, Tristan|
+----------------+------------------+
only showing top 20 rows
```

Output - Table 1 shows players with their closest defenders and their missed shots and fear score. In table 2 we can see that for Aaron brooks, Phil Pressey is the most unwanted defender and so on.

Q2.2 - In this question I have selected the required columns from dataframe and created new dataframe - data. Then I have used the VectorAssembler class to convert class variables into vectors and passed it to Kmeans function. I have taken k = 4. For the players given in question I have calculated Average shot. Based on this prediction I have calculated the best zone for each player mentioned in the question.

In [3]:

```
#2.2
from pyspark.sql.functions import when, count, col, sum, regexp_replace, avg, max
from pyspark.ml.clustering import KMeans
from pyspark.ml.feature import VectorAssembler


data = df.select('player_name', 'SHOT_DIST', 'CLOSE_DEF_DIST', 'SHOT_CLOCK', 'SHOT_RESULT')

data = data.withColumn('SHOT_RESULT', regexp_replace('SHOT_RESULT', 'made', '1'))
           .withColumn('SHOT_RESULT', regexp_replace('SHOT_RESULT', 'missed', '0'))
           .withColumn('SHOT_RESULT', col('SHOT_RESULT').cast('float'))

data = data.withColumn('SHOT_DIST', col('SHOT_DIST').cast('float')).withColumn('CLOSE_DEF_DIST',
                   col('CLOSE_DEF_DIST').cast('float')).withColumn('SHOT_CLOCK', col('SHOT_CLOCK').cast('float'))

columns = ['SHOT_DIST', 'CLOSE_DEF_DIST', 'SHOT_CLOCK']

va = VectorAssembler(inputCols=columns, outputCol='features')
nba = va.transform(data).select('player_name', 'features','SHOT_RESULT')

kmeans = KMeans(k=4,seed=1,featuresCol='features')
knba = kmeans.fit(nba)

players = ['james harden', 'chris paul', 'stephen curry', 'lebron james']
nba = nba.filter(col('player_name').isin(players))

pred = knba.transform(nba).select('player_name', 'prediction', 'SHOT_RESULT')
res = pred.groupBy('player_name', 'prediction').agg(avg('SHOT_RESULT').alias('avg_shot_result'))
                                               .orderBy('player_name', 'prediction')

res.show()

bestZone = res.join(res.groupBy('player_name').agg(max('avg_shot_result').alias('max_avg_shot_result')),
                   on='player_name').filter(col('avg_shot_result') == col('max_avg_shot_result'))
                   .select('player_name', 'prediction', 'avg_shot_result').orderBy('player_name')

bestZone.show()

```

```
+-------------+----------+-------------------+
|  player_name|prediction|    avg_shot_result|
+-------------+----------+-------------------+
|   chris paul|         0| 0.4914772727272727|
|   chris paul|         1|0.4950495049505049|
|   chris paul|         2|             0.4296875|
|   chris paul|         3| 0.5563380281690141|
| james harden|         0|0.3297872340425531|
| james harden|         1| 0.5604395604395604|
| james harden|         2|0.43617021276595747|
| james harden|         3|0.48520710059171596|
| lebron james|         0| 0.4124293785310734|
| lebron james|         1| 0.6613545816733067|
| lebron james|         2|0.35789473684210527|
| lebron james|         3| 0.5427350427350427|
|stephen curry|         0| 0.4319654427645788|
|stephen curry|         1| 0.6350710900473934|
|stephen curry|         2| 0.4189189189189189|
|stephen curry|         3| 0.5714285714285714|
+-------------+----------+-------------------+

+-------------+----------+-------------------+
|  player_name|prediction|    avg_shot_result|
+-------------+----------+-------------------+
|   chris paul|         3|0.5563380281690141|
| james harden|         1|0.5604395604395604|
| lebron james|         1|0.6613545816733067|
|stephen curry|         1|0.6350710900473934|
+-------------+----------+-------------------+
```

Output - We can infer from the above table that each player's most comfortable zone is determined by averaging the highest likelihood of scoring across all zones. Zone-1 is represented by Prediction Class 0, Zone-1 by Prediction Class 2, and Zone-4 by Prediction Class 3.