iation-rule-mining-miniproject-1

November 8, 2024

1 Read the following cricket dataset : https://taxila-aws.bits-pilani.ac.in/pluginfile.php/1390403/mod_assign/intro/cricketers.csv

a) Identify the all-rounder by the following logic -> Get the players with most runs whose wickets are more than the median wickets of all the players where the players must have taken atleast 1 wicket. [3 Marks]

b) Perform K-means clustering for different values of K (2,3,4,5) and evaluate the quality of clustering using the Silhouette score. [3 points]

c) For each value of K, plot clusters (all cluster points in the same cluster with the same color, cluster points in different clusters in different colors). Use PCA for dimensionality reduction so that the data points can be plotted in 2 - D. Take only 5 data points from each cluster and the point should be labelled with the player name, else 0 marks. [3 pt]

d) Draw a bar graph with X-axis as K value and Y-axis as silhouette score. [1.5 pt]

e) List down 10 players with each cluster and categorize them to the batsman, all-rounder, bowler, etc. [1.5 pt]

References: But not limited to the following:-

PCA: https://towardsdatascience.com/pca-using-python-scikit-learn-e653f8989e60Links to an external site.

Silhouette score: https://towardsdatascience.com/silhouette-coefficient-validating-clustering-techniques-e976bb81d10cLinks to an external site.

[185]:
```
# Importing required packages
import numpy as np
import pandas as pd
import warnings as war
war.filterwarnings("ignore")
```

[186]:
```
# Defining dataset csv Path
dataSetPath="C:\\Users\\ASUS\\jupyterworkspace\\Assignment & Mini␣
 ↪Project\Module_04_Unsupervised Learning and Association Rule␣
 ↪Mining\\MiniProject\\cricketers.csv"
# Loading dataSet
dataSetRead=pd.read_csv(dataSetPath, delimiter='\t')
```

[187]:
```
# Displaying first 5 records to confirming data loading
print("**************************************************Displaying below␣
 ↪first 5 records************************************************************")
dataSetRead.head()
```

```
**************************************************Displaying below first 5
records************************************************************
```

[187]:

	PLAYER	matches_played	innings_batted	runs_scored	highest_runs	balls_faced	average_runs	strike_rate	innings_bowled	overs	runs_given	wickets_obtained	average_runs_per_wicket	bowling_economy
0	Aaron Finch	10	9	134	46	100	16.75	144.00	0	0.0	0	0	0.0	0.0
1	AB de Villiers	12	11	480	90	275	53.33	174.54	0	0.0	0	0	0.0	0.0
2	Abhishek Sharma	3	3	63	46	33	63.00	190.90	0	0.0	0	0	0.0	0.0
3	Ajinkya Rahane	15	14	370	65	313	28.46	118.21	0	0.0	0	0	0.0	0.0
4	Alex Hales	6	6	148	45	118	24.66	125.42	0	0.0	0	0	0.0	0.0

[188]:
```
# Displaying last 5 records to confirming data loading
print("**************************************************Displaying below␣
 ↪last 5 records************************************************************")
dataSetRead.tail()
```

```
**************************************************Displaying below last 5
records************************************************************
```

[188]:

	PLAYER	matches_played	innings_batted	runs_scored	highest_runs	balls_faced	average_runs	strike_rate	innings_bowled	overs
104	Anureet Singh	0	0	0	0	0	0.0	0.0	3	4.0
105	Avesh Khan	0	0	0	0	0	0.0	0.0	6	19.0
106	Barinder Sran	0	0	0	0					
107	Basil Thampi	0	0	0	0					
108	Ben Laughlin	0	0	0	0					

```
106            0           0.0           0.0               6   22.0
107            0           0.0           0.0               4   10.1
108            0           0.0           0.0               7   21.0

     runs_given  wickets_obtained  average_runs_per_wicket   bowling_economy
104          42                 1                    42.00             10.50
105         204                 4                    51.00             10.73
106         229                 4                    57.25             10.40
107         114                 5                    22.80             10.20
108         212                 9                    22.44             10.04
```

[189]:
```
# Displaying all records to confirming data loading
print("**********************************************************Displaying below␣
 ↪all records**************************************************************")
dataSetRead
```

```
**********************************************************Displaying below all
records**************************************************************
```

[189]:
```
              PLAYER  matches_played  innings_batted  runs_scored  \
0        Aaron Finch              10               9          134
1     AB de Villiers              12              11          480
2    Abhishek Sharma               3               3           63
3     Ajinkya Rahane              15              14          370
4         Alex Hales               6               6          148
..               …             …               …          …
104    Anureet Singh               0               0            0
105       Avesh Khan               0               0            0
106    Barinder Sran               0               0            0
107     Basil Thampi               0               0            0
108     Ben Laughlin               0               0            0

     highest_runs  balls_faced  average_runs  strike_rate  innings_bowled  \
0              46          100         16.75       144.00               0
1              90          275         53.33       174.54               0
2              46           33         63.00       190.90               0
3              65          313         28.46       118.21               0
4              45          118         24.66       125.42               0
..            …           …           …           …              …
104             0            0          0.00         0.00               3
105             0            0          0.00         0.00               6
106             0            0          0.00         0.00               6
107             0            0          0.00         0.00               4
108             0            0          0.00         0.00               7

     overs  runs_given  wickets_obtained  average_runs_per_wicket   \
0      0.0           0                 0                     0.00
```

```
1      0.0           0                0                      0.00
2      0.0           0                0                      0.00
3      0.0           0                0                      0.00
4      0.0           0                0                      0.00
..     …           …                …                      …
104    4.0          42                1                     42.00
105   19.0         204                4                     51.00
106   22.0         229                4                     57.25
107   10.1         114                5                     22.80
108   21.0         212                9                     22.44

     bowling_economy
0                0.00
1                0.00
2                0.00
3                0.00
4                0.00
..                …
104             10.50
105             10.73
106             10.40
107             10.20
108             10.04

[109 rows x 14 columns]
```

[190]:
```
# Displaying dimension of dataSet
print("Dimention of Dataset:- {}".format(dataSetRead.shape[0:2]))
print("Total number of rows in Dataset:- {}".format(dataSetRead.shape[0]))
print("Total number of columns in Dataset:- {}".format(dataSetRead.shape[1]))
```

```
Dimention of Dataset:- (109, 14)
Total number of rows in Dataset:- 109
Total number of columns in Dataset:- 14
```

[191]:
```
# Displaying description & statistcal summary of the dataSet
dataSetRead.describe().T
```

[191]:

	count	mean	std	min	25%	50%	\
matches_played	109.0	9.559633	5.161164	0.0	5.00	10.00	
innings_batted	109.0	7.889908	4.982078	0.0	4.00	7.00	
runs_scored	109.0	173.633028	182.356522	0.0	36.00	99.00	
highest_runs	109.0	43.495413	29.830268	0.0	20.00	40.00	
balls_faced	109.0	124.669725	123.670265	0.0	33.00	76.00	
average_runs	109.0	22.360092	16.093453	0.0	11.80	21.16	
strike_rate	109.0	122.258716	48.576709	0.0	109.09	130.26	
innings_bowled	109.0	4.513761	5.449456	0.0	0.00	2.00	

```
overs                     109.0   13.894495   18.395495  0.0    0.00    3.00
runs_given                109.0  119.935780  152.807630  0.0    0.00   27.00
wickets_obtained          109.0    3.926606    5.602094  0.0    0.00    0.00
average_runs_per_wicket   109.0   17.334862   21.910816 -0.0    0.00    0.00
bowling_economy           109.0    5.149817    4.773435  0.0    0.00    7.28

                            75%     max
matches_played            14.00   17.00
innings_batted            13.00   17.00
runs_scored              260.00  735.00
highest_runs              62.00  128.00
balls_faced              188.00  516.00
average_runs              30.00   75.83
strike_rate              146.04  300.00
innings_bowled             8.00   17.00
overs                     26.00   68.00
runs_given               223.00  533.00
wickets_obtained           6.00   24.00
average_runs_per_wicket   28.36  108.00
bowling_economy            9.23   16.50
```

[192]:
```
# Displaying the columns and their respective data types
dataSetRead.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 109 entries, 0 to 108
Data columns (total 14 columns):
 #   Column                   Non-Null Count  Dtype
---  ------                   --------------  -----
 0   PLAYER                   109 non-null    object
 1   matches_played           109 non-null    int64
 2   innings_batted           109 non-null    int64
 3   runs_scored              109 non-null    int64
 4   highest_runs             109 non-null    int64
 5   balls_faced              109 non-null    int64
 6   average_runs             109 non-null    float64
 7   strike_rate              109 non-null    float64
 8   innings_bowled           109 non-null    int64
 9   overs                    109 non-null    float64
 10  runs_given               109 non-null    int64
 11  wickets_obtained         109 non-null    int64
 12  average_runs_per_wicket  109 non-null    float64
 13  bowling_economy          109 non-null    float64
dtypes: float64(5), int64(8), object(1)
memory usage: 12.1+ KB
```

1.1 Question (a):- Identify the all-rounder by the following logic -> Get the players with most runs whose wickets are more than the median wickets of all the players where the players must have taken atleast 1 wicket. [3 Marks]

[193]:
```
# displaying player name who obtained at least 1 wicket
dataSetRead_Players_withWicket = dataSetRead[dataSetRead['wickets_obtained'].
 ↪values >= 1]
```

[194]:
```
dataSetRead_Players_withWicket
```

[194]:

	PLAYER	matches_played	innings_batted	runs_scored	\
6	Andre Russell	16	14	316	
7	Andrew Tye	14	8	32	
8	Axar Patel	9	8	80	
9	Ben Cutting	9	6	96	
10	Ben Stokes	13	13	196	
11	Bhuvneshwar Kumar	12	4	13	
13	Carlos Brathwaite	4	4	75	
16	Chris Morris	4	4	46	
17	Chris Woakes	5	4	17	
18	Colin de Grandhomme	9	8	131	
20	Corey Anderson	3	3	17	
21	D'Arcy Short	7	7	115	
22	Dan Christian	4	3	26	
24	Deepak Chahar	12	4	50	
27	Dwayne Bravo	16	10	141	
31	Glenn Maxwell	12	12	169	
32	Harbhajan Singh	13	3	29	
33	Hardik Pandya	13	13	260	
34	Harshal Patel	5	2	60	
38	Jaydev UOdkat	15	7	49	
39	Jofra Archer	10	8	15	
46	KrishOppa Gowtham	15	13	126	
47	KruOl Pandya	14	13	228	
54	Marcus Stoinis	7	7	99	
56	Mayank Markande	14	6	21	
57	Mitchell Johnson	6	2	16	
58	Moeen Ali	5	4	77	
59	Mohammad Obi	2	2	18	
60	Mohammed Siraj	11	4	25	
62	Nitish RaO	15	15	304	
64	Piyush Chawla	15	7	27	
67	Rahul Tewatia	8	5	50	
69	Rashid Khan	17	7	59	
70	Ravichandran Ashwin	14	9	102	
71	Ravindra Jadeja	16	10	89	

79	Shakib Al Hasan	17	13	239
80	Shane Watson	15	15	555
81	Shardul Thakur	13	1	15
83	Shivam Mavi	9	4	13
85	Shreyas Gopal	11	4	50
89	Sunil Orine	16	16	357
92	Tim Southee	8	4	52
93	Tom Curran	5	4	23
94	Vijay Shankar	13	11	212
96	Washington Sundar	7	6	65
98	Yusuf Pathan	15	13	260
101	Amit Mishra	0	0	0
102	Ankit Rajpoot	0	0	0
104	Anureet Singh	0	0	0
105	Avesh Khan	0	0	0
106	Barinder Sran	0	0	0
107	Basil Thampi	0	0	0
108	Ben Laughlin	0	0	0

	highest_runs	balls_faced	average_runs	strike_rate	innings_bowled	\
6	88	171	28.72	184.79	15	
7	14	38	5.33	84.21	14	
8	19	69	13.33	115.94	8	
9	37	58	24.00	165.51	7	
10	45	161	16.33	121.73	12	
11	7	16	6.50	81.25	12	
13	43	48	25.00	156.25	4	
16	27	26	46.00	176.92	4	
17	11	19	8.50	89.47	5	
18	40	84	26.20	155.95	7	
20	15	22	5.66	77.27	3	
21	44	99	16.42	116.16	2	
22	13	33	13.00	78.78	4	
24	39	29	16.66	172.41	12	
27	68	91	35.25	154.94	16	
31	47	120	14.08	140.83	10	
32	19	36	9.66	80.55	12	
33	50	195	28.88	133.33	13	
34	36	33	60.00	181.81	5	
38	26	38	12.25	128.94	15	
39	8	21	3.00	71.42	10	
46	33	64	14.00	196.87	15	
47	41	157	22.80	145.22	13	
54	29	76	24.75	130.26	6	
56	7	24	10.50	87.50	14	
57	12	11	-0.00	145.45	6	
58	65	46	19.25	167.39	5	

59	14	12	9.00	150.00	2
60	14	22	12.50	113.63	11
62	59	232	23.38	131.03	5
64	12	34	6.75	79.41	15
67	24	43	16.66	116.27	8
69	34	31	11.80	190.32	17
70	45	71	12.75	143.66	14
71	27	74	17.80	120.27	14
79	35	197	21.72	121.31	17
80	117	359	39.64	154.59	11
81	15	5	-0.00	300.00	13
83	7	15	4.33	86.66	9
85	24	45	16.66	111.11	10
89	75	188	22.31	189.89	16
92	36	46	26.00	113.04	8
93	18	28	7.66	82.14	5
94	54	148	53.00	143.24	4
96	35	38	21.66	171.05	7
98	45	200	28.88	130.00	1
101	0	0	0.00	0.00	10
102	0	0	0.00	0.00	8
104	0	0	0.00	0.00	3
105	0	0	0.00	0.00	6
106	0	0	0.00	0.00	6
107	0	0	0.00	0.00	4
108	0	0	0.00	0.00	7

	overs	runs_given	wickets_obtained	average_runs_per_wicket	\
6	37.5	355	13	27.30	
7	56.0	448	24	18.66	
8	26.0	218	3	72.66	
9	17.0	168	2	84.00	
10	37.0	303	8	37.87	
11	46.1	354	9	39.33	
13	10.1	94	5	18.80	
16	14.0	143	3	47.66	
17	18.2	190	8	23.75	
18	15.0	129	2	64.50	
20	8.4	115	3	38.33	
21	3.0	19	1	19.00	
22	11.5	101	4	25.25	
24	38.1	278	10	27.80	
27	53.3	533	14	38.07	
31	16.0	132	5	26.40	
32	31.5	270	7	38.57	
33	42.4	381	18	21.16	
34	17.3	167	7	23.85	

38	50.2	486	11	44.18
39	38.5	325	15	21.66
46	40.0	312	11	28.36
47	40.1	284	12	23.66
54	11.0	120	3	40.00
56	44.0	368	15	24.53
57	21.0	216	2	108.00
58	13.1	97	3	32.33
59	5.0	47	1	47.00
60	41.0	367	11	33.36
62	6.1	44	4	11.00
64	49.0	412	14	29.42
67	22.0	173	6	28.83
69	68.0	458	21	21.80
70	50.4	410	10	41.00
71	41.0	303	11	27.54
79	57.0	456	14	32.57
80	28.0	251	6	41.83
81	46.4	431	16	26.93
83	28.0	270	5	54.00
85	31.0	236	11	21.45
89	61.0	467	17	27.47
92	29.0	261	5	52.20
93	10.1	118	6	19.66
94	5.0	58	1	58.00
96	20.0	192	4	48.00
98	2.0	14	1	14.00
101	37.0	264	12	22.00
102	26.1	223	11	20.27
104	4.0	42	1	42.00
105	19.0	204	4	51.00
106	22.0	229	4	57.25
107	10.1	114	5	22.80
108	21.0	212	9	22.44

	bowling_economy
6	9.38
7	8.00
8	8.38
9	9.88
10	8.18
11	7.66
13	9.24
16	10.21
17	10.36
18	8.60
20	13.26

```
21                6.33
22                8.53
24                7.28
27                9.96
31                8.25
32                8.48
33                8.92
34                9.54
38                9.65
39                8.36
46                7.80
47                7.07
54               10.90
56                8.36
57               10.28
58                7.36
59                9.40
60                8.95
62                7.13
64                8.40
67                7.86
69                6.73
70                8.09
71                7.39
79                8.00
80                8.96
81                9.23
83                9.64
85                7.61
89                7.65
92                9.00
93               11.60
94               11.60
96                9.60
98                7.00
101               7.13
102               8.52
104              10.50
105              10.73
106              10.40
107              10.20
108              10.04
```

[195]:
```
# Calculating Median of wickets obtained
Wickets_Median = dataSetRead_Players_withWicket['wickets_obtained'].median()
print("Meadian of wickets {}".format(Wickets_Median))
```

Meadian of wickets 7.0

[196]:
```
#  Fetching description details of all-rounders
Allrounders_dataSetRead =␣
 ↪dataSetRead_Players_withWicket[dataSetRead_Players_withWicket['wickets_obtained'].
 ↪values > Wickets_Median]
Allrounders_dataSetRead.describe()
```

[196]:

	matches_played	innings_batted	runs_scored	highest_runs	balls_faced
count	25.000000	25.000000	25.00000	25.000000	25.000000
mean	11.960000	7.520000	97.08000	28.720000	67.640000
std	5.191981	4.814215	107.14162	23.821419	67.859463
min	0.000000	0.000000	0.00000	0.000000	0.000000
25%	11.000000	4.000000	17.00000	11.000000	21.000000
50%	14.000000	7.000000	50.00000	26.000000	38.000000
75%	15.000000	13.000000	141.00000	41.000000	91.000000
max	17.000000	16.000000	357.00000	88.000000	197.000000

	average_runs	strike_rate	innings_bowled	overs	runs_given
count	25.000000	25.000000	25.000000	25.000000	25.00000
mean	13.240400	120.867200	12.720000	42.812000	354.24000
std	9.700819	67.651465	3.075711	11.699192	93.66175
min	0.000000	0.000000	5.000000	18.200000	190.00000
25%	6.500000	84.210000	11.000000	37.500000	284.00000
50%	12.500000	121.310000	13.000000	41.000000	355.00000
75%	17.800000	154.940000	15.000000	50.200000	431.00000
max	35.250000	300.000000	17.000000	68.000000	533.00000

	wickets_obtained	average_runs_per_wicket	bowling_economy
count	25.000000	25.000000	25.000000
mean	13.000000	28.103200	8.348800
std	3.937004	7.196091	0.993845
min	8.000000	18.660000	6.730000
25%	11.000000	22.000000	7.650000
50%	12.000000	27.300000	8.180000
75%	15.000000	32.570000	8.950000
max	24.000000	44.180000	10.360000

[197]:
```
# Displaying allrounders details in descending orders based on run scored
Allrounders_dataSetRead.sort_values(by='runs_scored',ascending=False)
```

[197]:

	PLAYER	matches_played	innings_batted	runs_scored
89	Sunil Orine	16	16	357
6	Andre Russell	16	14	316
33	Hardik Pandya	13	13	260
79	Shakib Al Hasan	17	13	239
47	Kru0l Pandya	14	13	228

10	Ben Stokes	13	13	196
27	Dwayne Bravo	16	10	141
46	KrishOppa Gowtham	15	13	126
70	Ravichandran Ashwin	14	9	102
71	Ravindra Jadeja	16	10	89
69	Rashid Khan	17	7	59
24	Deepak Chahar	12	4	50
85	Shreyas Gopal	11	4	50
38	Jaydev UOdkat	15	7	49
7	Andrew Tye	14	8	32
64	Piyush Chawla	15	7	27
60	Mohammed Siraj	11	4	25
56	Mayank Markande	14	6	21
17	Chris Woakes	5	4	17
39	Jofra Archer	10	8	15
81	Shardul Thakur	13	1	15
11	Bhuvneshwar Kumar	12	4	13
101	Amit Mishra	0	0	0
102	Ankit Rajpoot	0	0	0
108	Ben Laughlin	0	0	0

	highest_runs	balls_faced	average_runs	strike_rate	innings_bowled	\
89	75	188	22.31	189.89	16	
6	88	171	28.72	184.79	15	
33	50	195	28.88	133.33	13	
79	35	197	21.72	121.31	17	
47	41	157	22.80	145.22	13	
10	45	161	16.33	121.73	12	
27	68	91	35.25	154.94	16	
46	33	64	14.00	196.87	15	
70	45	71	12.75	143.66	14	
71	27	74	17.80	120.27	14	
69	34	31	11.80	190.32	17	
24	39	29	16.66	172.41	12	
85	24	45	16.66	111.11	10	
38	26	38	12.25	128.94	15	
7	14	38	5.33	84.21	14	
64	12	34	6.75	79.41	15	
60	14	22	12.50	113.63	11	
56	7	24	10.50	87.50	14	
17	11	19	8.50	89.47	5	
39	8	21	3.00	71.42	10	
81	15	5	-0.00	300.00	13	
11	7	16	6.50	81.25	12	
101	0	0	0.00	0.00	10	
102	0	0	0.00	0.00	8	
108	0	0	0.00	0.00	7	

	overs	runs_given	wickets_obtained	average_runs_per_wicket	\
89	61.0	467	17	27.47	
6	37.5	355	13	27.30	
33	42.4	381	18	21.16	
79	57.0	456	14	32.57	
47	40.1	284	12	23.66	
10	37.0	303	8	37.87	
27	53.3	533	14	38.07	
46	40.0	312	11	28.36	
70	50.4	410	10	41.00	
71	41.0	303	11	27.54	
69	68.0	458	21	21.80	
24	38.1	278	10	27.80	
85	31.0	236	11	21.45	
38	50.2	486	11	44.18	
7	56.0	448	24	18.66	
64	49.0	412	14	29.42	
60	41.0	367	11	33.36	
56	44.0	368	15	24.53	
17	18.2	190	8	23.75	
39	38.5	325	15	21.66	
81	46.4	431	16	26.93	
11	46.1	354	9	39.33	
101	37.0	264	12	22.00	
102	26.1	223	11	20.27	
108	21.0	212	9	22.44	

	bowling_economy
89	7.65
6	9.38
33	8.92
79	8.00
47	7.07
10	8.18
27	9.96
46	7.80
70	8.09
71	7.39
69	6.73
24	7.28
85	7.61
38	9.65
7	8.00
64	8.40
60	8.95
56	8.36

```
17              10.36
39               8.36
81               9.23
11               7.66
101              7.13
102              8.52
108             10.04
```

1.2 Question (b):- Perform K-means clustering for different values of K (2,3,4,5) and evaluate the quality of clustering using the Silhouette score. [3 points]¶

[198]:
```
# Importing required packages
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score
from sklearn.preprocessing import MinMaxScaler

# Selecting all numerical features for K-Means Cl
New_dataSetRead = dataSetRead.select_dtypes(include=['float64', 'int64']).copy()

# Standardize the data for K-means clustering
scaler = MinMaxScaler()
scaled_data = scaler.fit_transform(New_dataSetRead)

# Dictionary to store Silhouette scores for each K value
silhouette_scores = {}

# Perform K-means clustering for K values 2, 3, 4, and 5
for k in [2, 3, 4, 5]:
    kmeans = KMeans(n_clusters=k, random_state=42)
    cluster_labels = kmeans.fit_predict(scaled_data)
    score = silhouette_score(scaled_data, cluster_labels)
    silhouette_scores[k] = score
    print(f'Silhouette Score for K={k}: {score:.5f}')
# Identify the best K value based on the highest silhouette score
Best_K_Value = max(silhouette_scores, key=silhouette_scores.get)
print(f"\nBest K value: {Best_K_Value } with a Silhouette Score of␣
 ↪{silhouette_scores[Best_K_Value]:.5f}")
```

```
Silhouette Score for K=2: 0.37118
Silhouette Score for K=3: 0.38084
Silhouette Score for K=4: 0.38135
Silhouette Score for K=5: 0.37025

Best K value: 4 with a Silhouette Score of 0.38135
```

Analysis:- The silhouette score generally indicates how well-separated and compact the clusters are, with higher scores being preferable. In this case:

The highest silhouette score is at =4 (0.38135), suggesting this value likely offers the best cluster separation among the tested values.

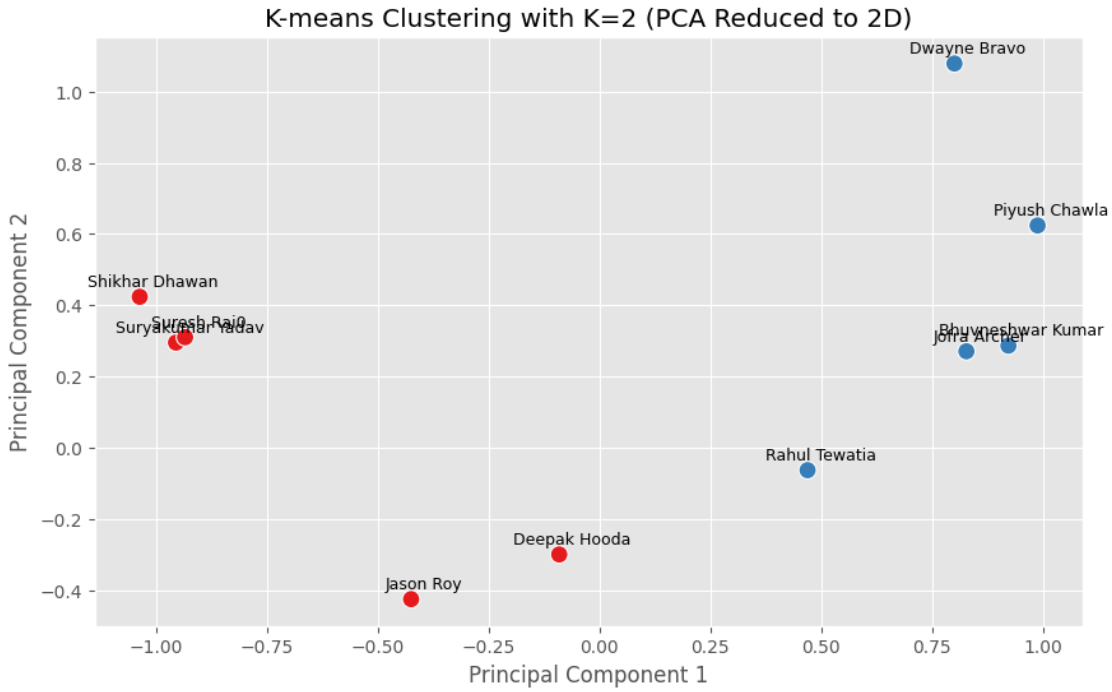
The score drops slightly at =5 indicating that increasing clusters beyond =4 may introduce some overlap or reduce the compactness of clusters.

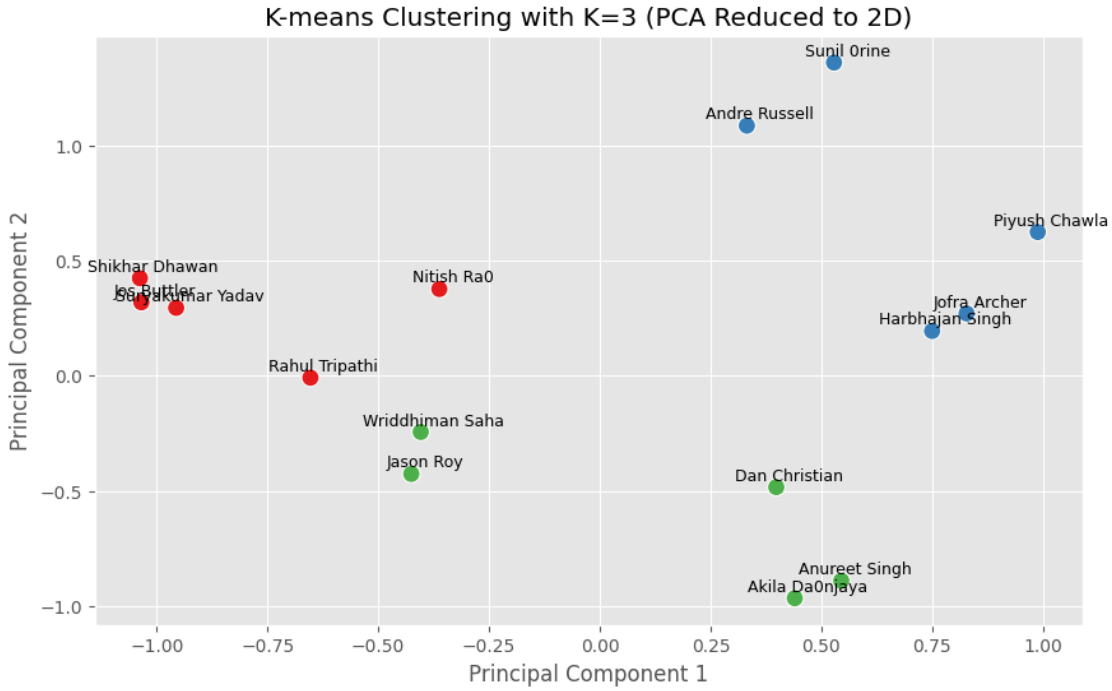
Based on these values, =4 seems like the optimal choice for achieving the best-defined clusters with the data you have.

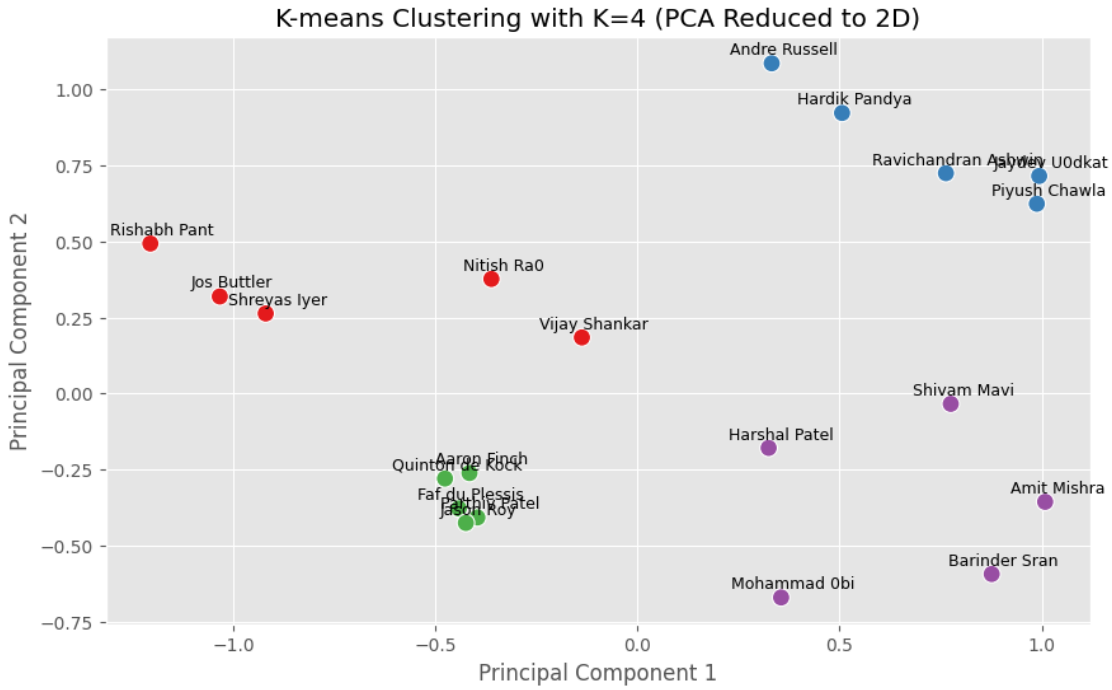
1.3 Question (c):-For each value of K, plot clusters (all cluster points in the same cluster with the same color, cluster points in different clusters in different colors). Use PCA for dimensionality reduction so that the data points can be plotted in 2 - D. Take only 5 data points from each cluster and the point should be labelled with the player name, else 0 marks. [3 pt]

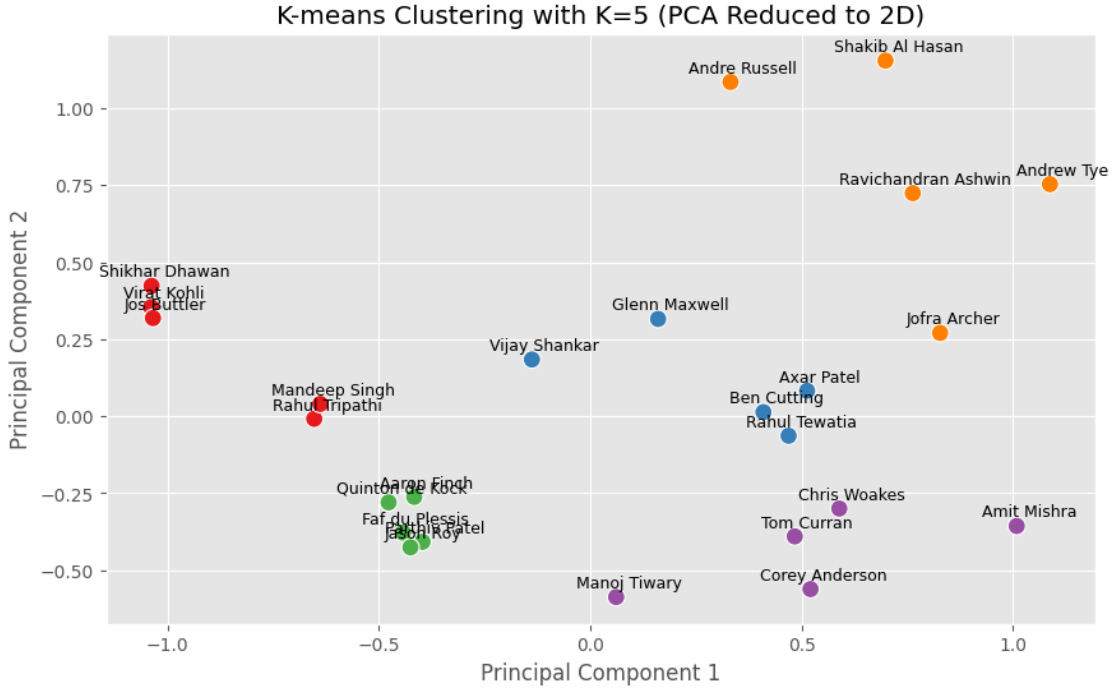
[199]:
```
# importing required packages
import matplotlib.pyplot as plt
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import seaborn as sns
import pandas as pd

# Perform PCA for dimensionality reduction (to 2D)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(scaled_data)

# Loop through each K (number of clusters)
for k in range(2, 6):
    kmeans = KMeans(n_clusters=k, random_state=42)
    clusters = kmeans.fit_predict(scaled_data)

    # Create an empty DataFrame to hold only the selected points
    selected_points = pd.DataFrame(columns=['PCA1', 'PCA2', 'Cluster',␣
 ↪'Player'])

    # Select only 5 points from each cluster for display
    for cluster_num in range(k):
        # Get indices of points in the current cluster
        cluster_indices = pd.Series(range(len(clusters)))[clusters ==␣
 ↪cluster_num]

        # Randomly select 5 indices from the cluster
        if len(cluster_indices) > 5:
            selected_indices = cluster_indices.sample(5, random_state=42).values
        else:
            selected_indices = cluster_indices.values  # If fewer than 5, take␣
 ↪all points
```

```
        # Collect selected points into a temporary DataFrame
        temp_df = pd.DataFrame({
            'PCA1': X_pca[selected_indices, 0],
            'PCA2': X_pca[selected_indices, 1],
            'Cluster': cluster_num,
            'Player': dataSetRead.loc[selected_indices, 'PLAYER']  # Get player␣
↪names from df
        })

        # Append temp_df to selected_points
        selected_points = pd.concat([selected_points, temp_df],␣
↪ignore_index=True)

    # Plot only the selected points with labels
    plt.figure(figsize=(10, 6))
    sns.scatterplot(data=selected_points, x='PCA1', y='PCA2', hue='Cluster',␣
↪palette="Set1", s=100, marker="o", legend=None)

   # Annotate each selected point with the player name
    for _, row in selected_points.iterrows():
        plt.text(row['PCA1'] + 0.03, row['PCA2'] + 0.03, row['Player'],␣
↪fontsize=9, ha='center', color='black')

    # Set plot title and labels
    plt.title(f'K-means Clustering with K={k} (PCA Reduced to 2D)')
    plt.xlabel('Principal Component 1')
    plt.ylabel('Principal Component 2')
    plt.show()
```


K-means Clustering with K=2 (PCA Reduced to 2D)
Dwayne Bravo
Piyush Chawla
Shikhar Dhawan
Suresh Raio
Suryakumar Yadav
Bhuvneshwar Kumar
Jofra Archer
Rahul Tewatia
Deepak Hooda
Jason Roy
Principal Component 2
Principal Component 1
1.0
0.8
0.6
0.4
0.2
0.0
−0.2
−0.4
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00


K-means Clustering with K=3 (PCA Reduced to 2D)
Sunil 0rine
Andre Russell
Piyush Chawla
Shikhar Dhawan
Nitish Ra0
Jofra Archer
Harbhajan Singh
Rahul Tripathi
Wriddhiman Saha
Jason Roy
Dan Christian
Anureet Singh
Akila Da0njaya
Principal Component 2
Principal Component 1
1.0
0.5
0.0
−0.5
−1.0
−1.00
−0.75
−0.50
−0.25
0.00
0.25
0.50
0.75
1.00

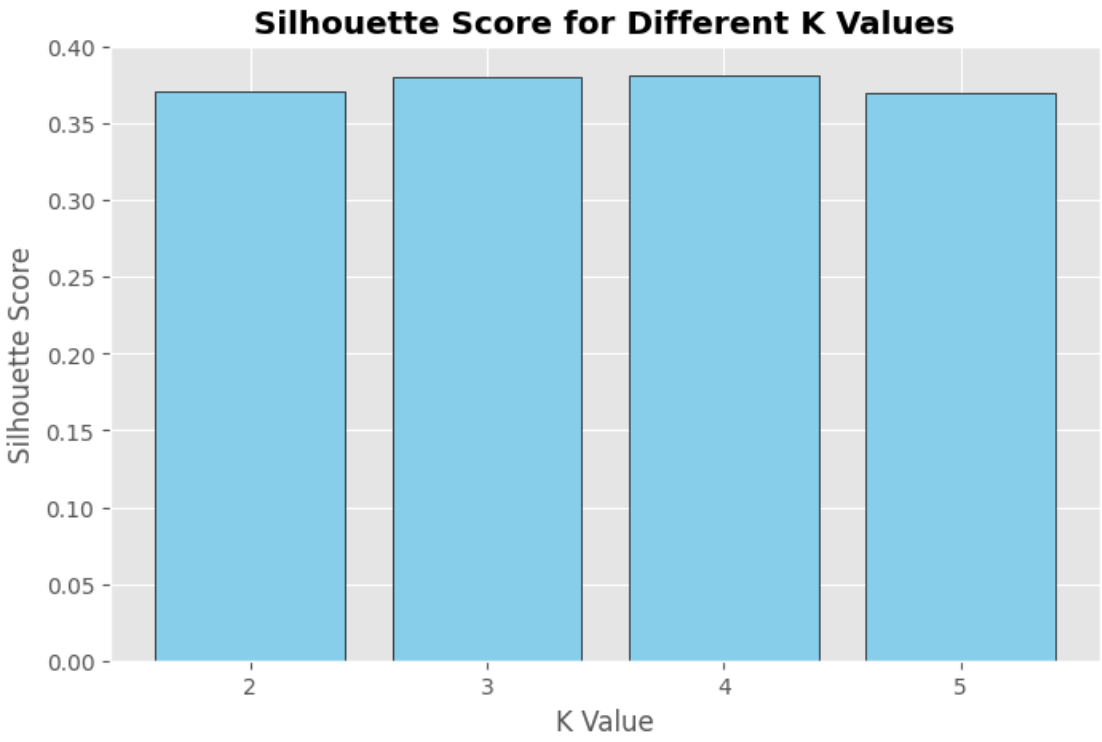

K-means Clustering with K=4 (PCA Reduced to 2D)
Andre Russell
Hardik Pandya
Ravichandran Ashwin
Jaydev U0dkat
Piyush Chawla
Rishabh Pant
Nitish Ra0
Jos Buttler
Shreyas Iyer
Vijay Shankar
Shivam Mavi
Harshal Patel
Aaron Finch
Quinton de Kock
Faf du Plessis
Amit Mishra
Barinder Sran
Mohammad 0bi
Principal Component 2
Principal Component 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.0
−0.5
0.0
0.5
1.0


K-means Clustering with K=5 (PCA Reduced to 2D)
Shakib Al Hasan
Andre Russell
Ravichandran Ashwin
Andrew Tye
Shikhar Dhawan
Virat Kohli
Jos Buttler
Glenn Maxwell
Jofra Archer
Vijay Shankar
Axar Patel
Ben Cutting
Rahul Tewatia
Mandeep Singh
Rahul Tripathi
Aaron Finch
Quinton de Kock
Faf du Plessis
Chris Woakes
Tom Curran
Amit Mishra
Manoj Tiwary
Corey Anderson
Principal Component 2
Principal Component 1
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−1.0
−0.5
0.0
0.5
1.0

1.4 Question (d):-Draw a bar graph with X-axis as K value and Y-axis as silhouette score. [1.5 pt]

[200]:
```
# Values for K and corresponding silhouette scores
print('*************************************************************************************')
print('The dictionary contains K values and corresponding silhouette score from␣
 ↪question (b): ')
print('  ')
print(silhouette_scores)
print('*************************************************************************************')
k_values = list(silhouette_scores.keys())
silhouette_values = list(silhouette_scores.values())

# Plotting the bar graph
plt.figure(figsize=(8, 5))
plt.style.use('ggplot')
plt.bar(k_values, silhouette_values, color='skyblue',edgecolor='black')
plt.xlabel('K Value')
plt.ylabel('Silhouette Score')
plt.title('Silhouette Score for Different K Values',fontweight='bold')
plt.xticks(k_values)
plt.show()
```

```
*************************************************************************************
*************************************
The dictionary contains K values and corresponding silhouette score from
question (b):

{2: 0.3711813404418582, 3: 0.3808399859723335, 4: 0.38135494702813433, 5:
0.3702497715206442}
*************************************************************************************
*************************************
```



1.5 Question (e):-List down 10 players with each cluster and categorize them to the batsman, all-rounder, bowler, etc. [1.5 pt]

[202]:
```
# Step (e): List down 10 players with each cluster and categorize them
print(silhouette_scores)
print('best_k:', best_k)
# Final clustering step to get cluster labels for the best K (highest␣
 ↪silhouette score)
best_k = max(silhouette_scores, key=silhouette_scores.get)
final_kmeans = KMeans(n_clusters=best_k, random_state=42, n_init=10)
final_cluster_labels = final_kmeans.fit_predict(scaled_data)

# Add cluster labels to the dataframe
dataSetRead['cluster'] = final_cluster_labels

# Group players by their cluster and list down 10 players from each
clustered_players = {}
for cluster in range(best_k):
    players_in_cluster = dataSetRead[dataSetRead['cluster'] ==␣
 ↪cluster]['PLAYER'].head(10).tolist()
    clustered_players[f'Cluster {cluster + 1}'] = players_in_cluster
```

```
# Display the categorized players from each cluster
for cluster, players in clustered_players.items():
    print(f"{cluster}: {players}")
```

```
{2: 0.3711813404418582, 3: 0.3808399859723335, 4: 0.38135494702813433, 5:
0.3702497715206442}
best_k: 4
Cluster 1: ['AB de Villiers', 'Ajinkya Rahane', 'Ambati Rayudu', 'Chris Gayle',
'Chris Lynn', 'Dinesh Karthik', 'Evin Lewis', 'Ishan Kishan', 'Jos Buttler',
'Kane Williamson ']
Cluster 2: ['Andre Russell', 'Andrew Tye', 'Ben Stokes', 'Bhuvneshwar Kumar',
'Deepak Chahar', 'Dwayne Bravo', 'Glenn Maxwell', 'Harbhajan Singh', 'Hardik
Pandya', 'Jaydev U0dkat']
Cluster 3: ['Aaron Finch', 'Abhishek Sharma', 'Alex Hales', 'Brendon McCullum',
'Colin Munro', "D'Arcy Short", 'David Miller', 'Deepak Hooda', 'Faf du Plessis',
'Gautam Gambhir']
Cluster 4: ['Axar Patel', 'Ben Cutting', 'Carlos Brathwaite', 'Chris Morris',
'Chris Woakes', 'Colin de Grandhomme', 'Corey Anderson', 'Dan Christian',
'Harshal Patel', 'JP Duminy']
```

[205]:
```
dataSetRead['cluster'].value_counts()
```

[205]:
```
cluster
3    32
0    30
2    24
1    23
Name: count, dtype: int64
```

[206]:
```
# Set option to display maximum rows in the DataFrame
pd.reset_option('display.max_rows', None)  # None displays all rows
```

[208]:
```
# Function to classify players based on statistical thresholds
def classify_player(row):
    if row['wickets_obtained'] > 10 and row['runs_scored'] < 100:
        return 'Bowler'
    elif row['wickets_obtained'] > 10 and row['runs_scored'] >= 100:
        return 'All-Rounder'
    elif row['runs_scored'] >= 200 and row['wickets_obtained'] < 10:
        return 'Batsman'
    elif row['wickets_obtained'] == 0:
        return 'Batsman'
    else:
        return 'All-Rounder'  # Default fallback

# Apply the classification function to the dataframe
dataSetRead['Role'] = dataSetRead.apply(classify_player, axis=1)
```

```python
# Group by Cluster and display 10 players from each cluster
clustered_players = dataSetRead.groupby('cluster').apply(lambda x: x.
 ↪sample(n=10, random_state=42) if len(x) >= 10 else x)
clustered_players = clustered_players.reset_index(drop=True)  # Reset index to␣
 ↪avoid ambiguity

# Display the result
for cluster, group in clustered_players.groupby('cluster'):
    print(f"\nCluster {cluster} - Sampled Players:")
    print(group[['PLAYER', 'Role']])
```

```
Cluster 0 - Sampled Players:
             PLAYER     Role
0     Vijay Shankar  Batsman
1        Nitish Ra0  Batsman
2      Shreyas Iyer  Batsman
3      Rishabh Pant  Batsman
4       Jos Buttler  Batsman
5  Kane Williamson   Batsman
6       Virat Kohli  Batsman
7      Shubman Gill  Batsman
8     Mandeep Singh  Batsman
9    AB de Villiers  Batsman

Cluster 1 - Sampled Players:
                 PLAYER         Role
10        Piyush Chawla       Bowler
11        Jaydev U0dkat       Bowler
12        Andre Russell  All-Rounder
13        Hardik Pandya  All-Rounder
14  Ravichandran Ashwin  All-Rounder
15         Kru0l Pandya  All-Rounder
16           Andrew Tye       Bowler
17      Mayank Markande       Bowler
18         Dwayne Bravo  All-Rounder
19           Ben Stokes  All-Rounder

Cluster 2 - Sampled Players:
             PLAYER         Role
20   Faf du Plessis      Batsman
21    Parthiv Patel      Batsman
22      Aaron Finch      Batsman
23  Quinton de Kock      Batsman
24        Jason Roy      Batsman
25   Gautam Gambhir      Batsman
```

```
26   Kieron Pollard      Batsman
27  Abhishek Sharma      Batsman
28    Sarfaraz Khan      Batsman
29     D'Arcy Short  All-Rounder

Cluster 3 - Sampled Players:
            PLAYER         Role
30   Barinder Sran  All-Rounder
31    Mohammad Obi  All-Rounder
32     Amit Mishra       Bowler
33     Shivam Mavi  All-Rounder
34   Harshal Patel  All-Rounder
35       JP Duminy      Batsman
36    Basil Thampi  All-Rounder
37   Ankit Rajpoot       Bowler
38  Marcus Stoinis  All-Rounder
39      Axar Patel  All-Rounder
```