Basics of Data Analysis
Homework report

December 2023
Moscow, Russia

Author: Marwan Bouabidi
Supervisor: Prof. Mirkin Boris Grigorievich

NATIONAL RESEARCH UNIVERSITY

Department of Computer Science

Contents

List of Figures

List of Tables

1 Introduction

In this small study, I will employ some powerful data analysis tools to reveal hidden patterns and relationships within the chosen dataset of boxing matches. The correlation coefficient will enable us to assess the connections between different variables. Principal Component Analysis (PCA) will assist in identifying the most significant features in the data. And K-means clustering will allow us to group matches based on shared characteristics, revealing hidden clusters of data. I will also use contingency tables to assess statistical relationship s between some features, and use the bootstrap method along with confidence intervals to compare means between 2 chosen clusters, and the grand mean with a cluster mean.
This analysis provides an interesting way of exploring the dynamics of the sport and seeing it from an unusual angle.

2 Dataset

This data set is a collection of boxing matches results, since I'm interested in the sport and thought it would be a good idea to apply some data analysis tools to this kind of information.
I removed rows with missing data (except for judges scores, as they are not always present) and was left with **7155** rows or objects in the resulting dataset. Initially, the features were:

- **age_A, age_B**: age of the boxer
- **height_A, height_B**: height (in cm)
- **reach_A, reach_B**: arms reach (in cm)
- **stance_A, stance_B**: type of stance, 'orthodox' or 'southpaw'
- **weight_A, weight_B**: weight (in lbs)
- **won_A, won_B**: number of wins
- **lost_A, lost_B**: number of losses
- **drawn_A, drawn_B**: number of draws
- **kos_A, kos_B**: number of KOs
- **result**: the match result. 'win_A', 'win_B', or 'draw'
- **decision**: match conclusion. 'UD' (unanimous decision), 'SD' (split draw), 'MD' (majority decision), 'PTS' (points), 'KO', 'TKO' (technical KO), 'TD', 'RTD' (retired), 'DQ' (disqualification)
  Please see this link for abbreviations:
  `https://www.boxingbase.com/boxing-results-explained/`

- **judge1_A, judge1_B**: scores of judge 1
- **judge2_A, judge2_B**: scores of judge 2
- **judge3_A, judge3_B**: scores of judge 3

However, I was advised to consider the matches from the point of view of Boxer A only. So the features became something like: **age_A, age_diff, height_A, height_diff, reach_A, reach_diff**... Where:

$$\text{feature\_diff} = \text{feature\_A} - \text{feature\_B}$$

And so, the new list of features is:

```
['age', 'age_diff', 'height', 'height_diff',
'reach', 'reach_diff', 'stance', 'stance_diff',
'weight', 'weight_diff', 'won', 'won_diff',
'lost', 'lost_diff', 'drawn', 'drawn_diff', 'kos',
'kos_diff', 'result', 'decision', 'judge1',
'judge1_diff', 'judge2', 'judge2_diff', 'judge3',
'judge3_diff']
```

So, if the difference is **positive**, the feature for the boxer we're considering is higher than that of his opponent, and vice-versa for the negative differences.

Source: `https://www.kaggle.com/datasets/mexwell/boxing-matches`

3 Analysis

3.1 Correlation coefficient

We need firstly to find 2 features with a linear-like relationship, and that is achieved by examining scatter-plots of feature pairs. I have chosen to examine the classic features of **height** and **reach** of the boxer, as their scatter-plot shows a linear pattern, with only a couple of points falling far from the rest, also known as **outliers**, which could greatly affect the linear regression model, in particular by inflating the correlation coefficient ρ. We can, of course, rule some of these points out as errors in measurements, because, for example, a reach of more than 4 meters doesn't seem humanly possible. Also, it is not irrational to assume some form of linear relationship between these 2 features, because based on everyday observations it seems they might be proportional (although, to my knowledge, there is no clear scientific evidence for this).
The scatter-plot below shows a linear-like pattern of points suitable enough to build a **Linear Regression** model, so let us examine this further.

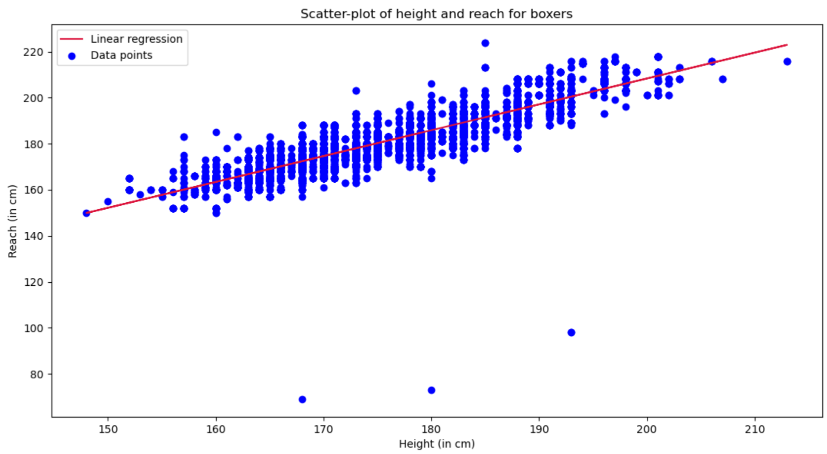

Figure 1: Scatter-plot of height and reach for different boxers with linear regression model

For 2 features **X** and **Y**, correlation coefficients are given by the following formulas:

$$\rho = \frac{\sum_{i=1}^{N}(x_i - \bar{X})(y_i - \bar{Y})/N}{\sigma(x)\sigma(y)}$$

$$a = \rho\frac{\sigma(y)}{\sigma(x)}$$

$$b = \bar{y} - a * \bar{x}$$

$$\text{Such that: } Y = a * X + b$$

Where σ is the standard deviation, ρ the **correlation coefficient**, Y representing the reach, and X the height.
In python, I calculated 2 versions of ρ, one with the **scipy.stats** library, and another using the formula. Rounded to two decimals, they're equal.
For these chosen features, I obtained:

$$\rho = 0.88$$

$$a = 1.12$$

$$b = -16.45$$

$$\text{Therefore: } Y = 1.12 * X - 16.45$$

The determinancy coefficient: $\rho^2 = 0.77 = 77\%$

The correlation coefficient measures the extent of linearity between X and Y, while the determinancy coefficient represents the percentage of variance of Y

that is explained by linear regression (of Y over X).
We compute two errors to verify the regression model:

- **Data Analysis error:** $\frac{|Y_{pred} - Y_{real}|}{|Y_{real}|} * 100$
- **Machine Learning error:** $\frac{|Y_{pred} - Y_{real}|}{|Y_{pred}|} * 100$

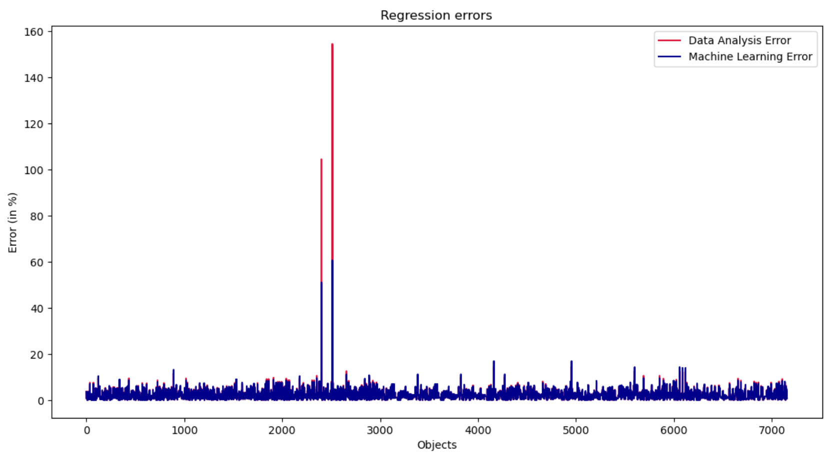

Figure 2: Errors of the Linear Regression model

Both errors are almost always below the 20% line, except for a few spikes, so we can conclude that the linear regression is a decent model to represent the relationship between these 2 features. The determinancy coefficient is 77%, leaving around 23% error margin.

3.2 Principal Component Analysis (PCA)

We assume our data are arranged in a matrix **X**, where the lines represent the objects, and the columns the features.
The features below were chosen because I believe they influence the outcome of the match the most.

```
['age', 'age_diff', 'won', 'won_diff', 'lost', 'lost_diff']
```

We start by standardizing the data (for every column), and for that we'll be using 2 methods:

- **Z-scoring:** $Y = \frac{X - \bar{X}}{\sigma_X}$
- **Range normalisation:** $Y = \frac{X - \bar{X}}{max - min}$

After, we apply SVD on the data matrix X for both cases (we'll note them X_z and X_r):

$$Z, \mu, C = SVD(X)$$
$$\text{Where: } X = Z * \mu * C^T$$

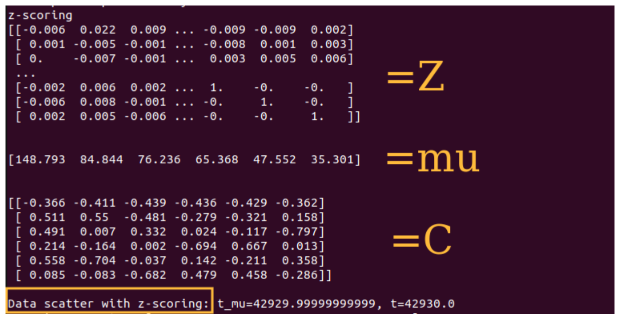

Figure 3: SVD with z-scoring

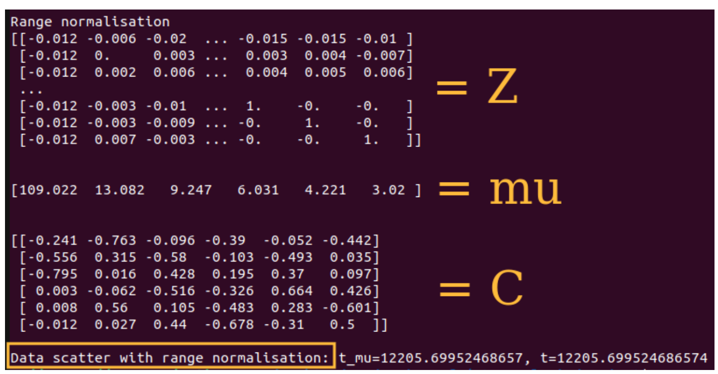

Figure 4: SVD with range normalisation

We notice from the SVD results and data scatter (which was calculated twice, with Y and μ) that:
Putting the square of the singular values in μ in percentage, we can see the contribution of each of the principal components to the data scatter:

```
mu2_z = [51.57, 16.77, 13.54, 9.95, 5.27, 2.9]
mu2_r = [54.59, 20.59, 11.23, 7.45, 4.18, 1.95]
```

Visualisation:
PCA helps us project our data objects on a 2D plane. For both cases, we visualise the first 2 components with the most contribution, which are the first 2 columns of Z: Z_1 and Z_2. And, depending on the values for C's columns, we

can consider C_i (and Z_i) or $-C_i$ (and $-Z_i$) because they are all eigenvectors of X^TX, since we have: $XC = \mu Z$ and $X^TZ = \mu C$ for a singular value μ. So, for visualization:

- For z-scoring: Z_1 and $-Z_2$
- For range normalisation: $-Z_1$ and $-Z_2$

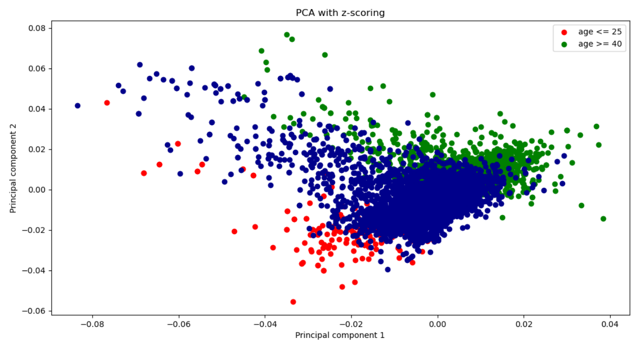

Figure 5: PC visualisation with z-scoring

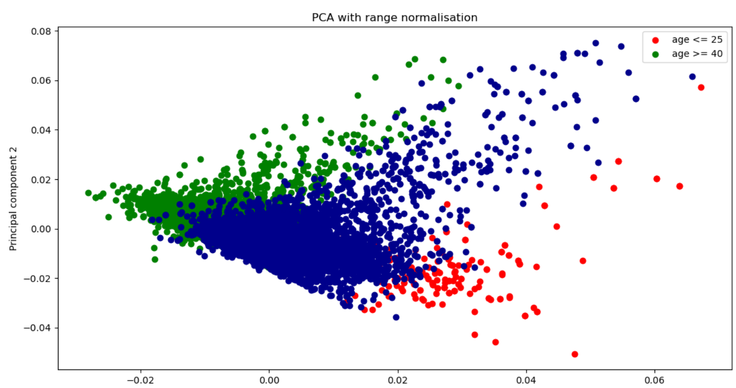

Figure 6: PC visualisation with range normalisation

By examining the corresponding scatters, we can see some clusters forming after age filters have been applied. The results of both standardisation methods look similar, since their respective principal components have similar contributions to the data scatter. But visually, it seems the data is more separated and clearer for interpretation with range normalisation.

Hidden factor:
To find a hidden factor, we start by normalising the data to get values between 0 and 100 using ranking normalisation:

$$Y = \frac{X - min}{max - min} * 100$$

Then, similarly, we apply SVD to get the principal components, we choose the most dominant, we get rid of negative values if they dominate, and finally the hidden factor is computed as:

$$\alpha = \frac{1}{sum(C)}$$

In our case, the value rounded to 2 decimals is $\alpha = 0.8 = 80\%$.

3.3 K-means Clustering

K-Means is an unsupervised Machine Learning algorithm used to partition or divide data objects into K groups or clusters, based on euclidean distances of these points to the K cluster centers.
For our dataset, we will apply this algorithm twice: one for **K=4**, and another for **K=7**. For each case we will run the algorithm 10 times, with random initialisations, and choose the best based on **inertia** value:

$$D(S,c) = \sum_{p=1}^{K} \sum_{i \in S_p} d(i, c_p) = \sum_{p=1}^{K} \sum_{i \in S_p} \sum_{v=1}^{V} (y_{iv} - c_{kv})^2$$

I chose the below features:

```
features = ['age','age_diff','reach','reach_diff','won',
'won_diff','kos','kos_diff']
```

These features, in my personal opinion, affect the most the outcome of the match. After standardizing with range normalisation (in%) the data and running the algorithm multiple times, we obtain a list of inertia values. We choose the iteration with minimum inertia value.
Something to note here: in python, the **KMeans** library specifically, there is an option *n_init* that controls how many times the algorithm will be executing, and then Python will return automatically the best iteration based on inertia values. It was set to 10 in my code, and I also executed KMeans 10 times manually, for the sake of showing results, so KMeans was actually executed a **100** times.

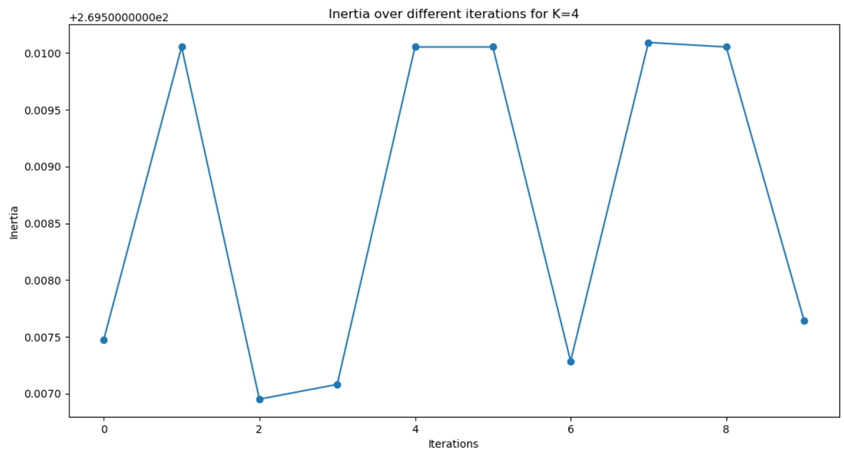

Figure 7: Inertia values over 10 iterations for K=4

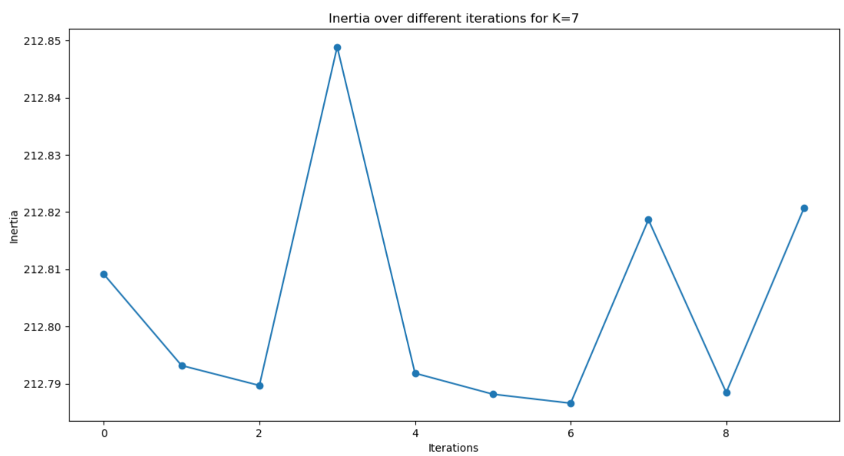

Figure 8: Inertia values over 10 iterations for K=7

3.3.1 K=4

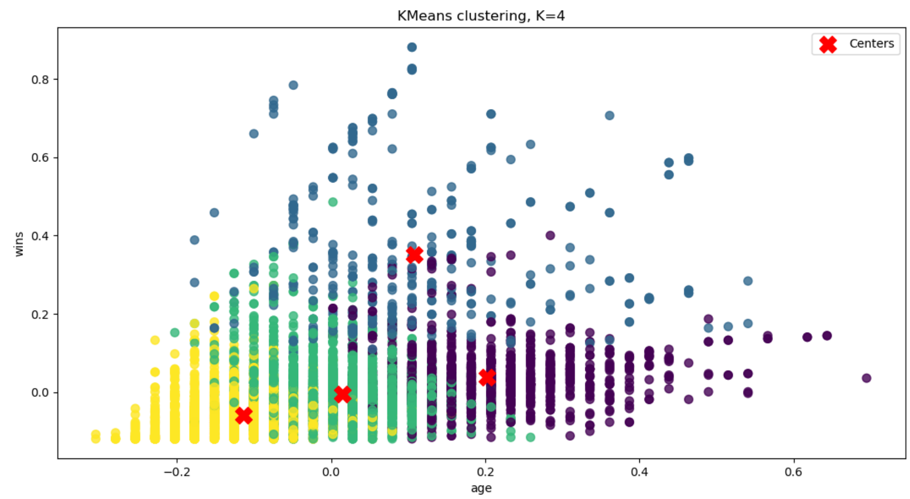

Figure 9: KMeans clusters for K=4 visualised for features **age and won** with range normalisation.

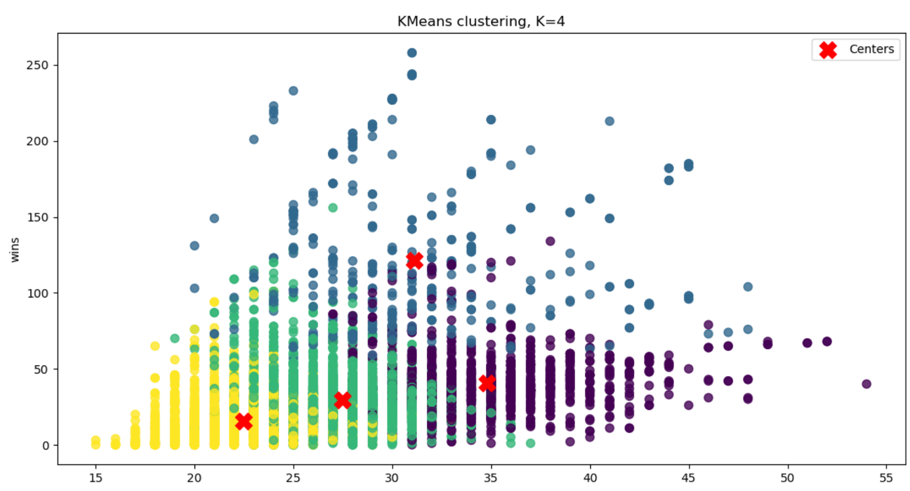

Figure 10: KMeans clusters for K=4 visualised for features **age and won** with real data.

3.3.2 K=7

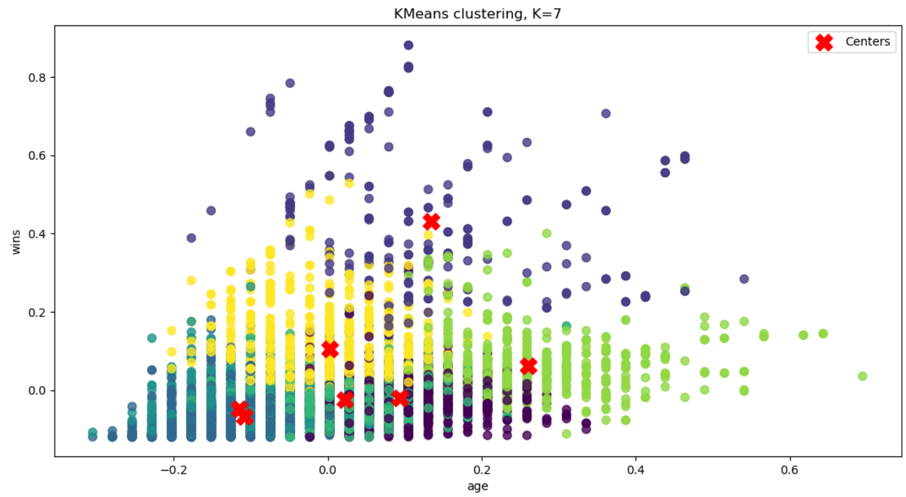

Figure 11: KMeans clusters for K=7 visualised for features **age and won** with range normalisation.

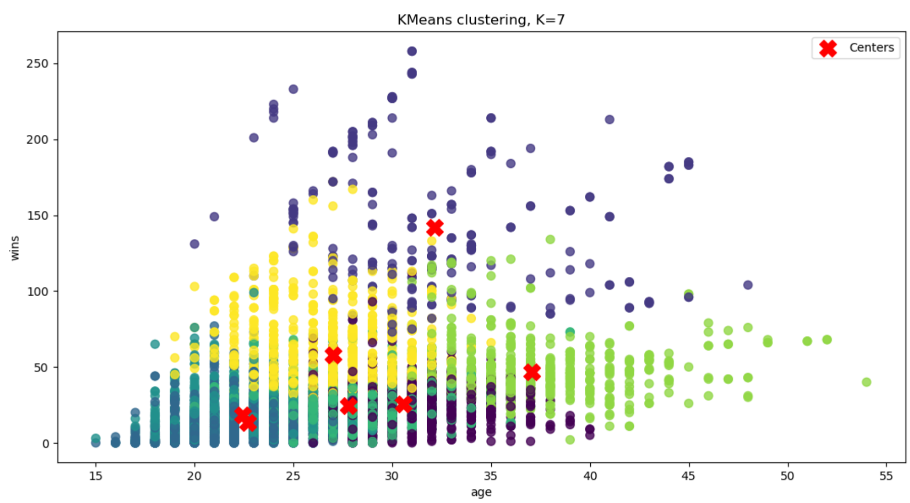

Figure 12: KMeans clusters for K=7 visualised for features **age and won** with real data.

3.3.3 Interpretation

By comparing the local means of clusters to the global mean for every chosen feature, I obtained the following results for K=4:

```
#Global means:
{'age': 26.91, 'age_diff': -1.41, 'reach': 181.46, 'reach_diff': 0.88,
'won': 30.75, 'won_diff': 3.26, 'kos': 18.23, 'kos_diff': 2.35}

#Means for cluster number 1 (size 1134) for K=4:
{'age': 34.8, 'age_diff': 6.34, 'reach': 185.4, 'reach_diff': 0.61,
'won': 40.48, 'won_diff': 14.46, 'kos': 25.38, 'kos_diff': 8.67}
#Relative differences to global means (%):
[22.67, 122.24, 2.13, 44.26, 24.04, 77.46, 28.17, 72.9]
Average (%):  49.23

#Means for cluster number 2 (size 376) for K=4:
{'age': 31.11, 'age_diff': 4.58, 'reach': 180.57, 'reach_diff': -2.39,
'won': 121.15, 'won_diff': 72.61, 'kos': 59.76, 'kos_diff': 36.6}
#Relative differences to global means (%):
[13.5, 130.79, 0.49, 136.82, 74.62, 95.51, 69.49, 93.58]
Average (%):  76.85

#Means for cluster number 3 (size 2871) for K=4:
{'age': 27.5, 'age_diff': -0.62, 'reach': 182.12, 'reach_diff': 0.82,
'won': 29.47, 'won_diff': 1.96, 'kos': 18.18, 'kos_diff': 1.96}
#Relative differences to global means (%):
[2.15, 127.42, 0.36, 7.32, 4.34, 66.33, 0.28, 19.9]
Average (%):  28.51

#Means for cluster number 4 (size 2774) for K=4:
{'age': 22.5, 'age_diff': -6.21, 'reach': 179.29, 'reach_diff': 1.49,
'won': 15.84, 'won_diff': -9.37, 'kos': 9.72, 'kos_diff': -4.46}
#Relative differences to global means (%):
[19.6, 77.29, 1.21, 40.94, 94.13, 134.79, 87.55, 152.69]
Average (%):  76.03
```

And **for K=7**, I obtained the following results:

```
#Global means:
{'age': 26.91, 'age_diff': -1.41, 'reach': 181.46, 'reach_diff': 0.88,
'won': 30.75, 'won_diff': 3.26, 'kos': 18.23, 'kos_diff': 2.35}

#Means for cluster number 1 (size 1149) for K=7:
{'age': 30.56, 'age_diff': 4.3, 'reach': 177.48, 'reach_diff': -1.58,
'won': 25.9, 'won_diff': 3.86, 'kos': 15.28, 'kos_diff': 1.13}
```

```
#Relative differences to global means (%):
[11.94, 132.79, 2.24, 155.7, 18.73, 15.54, 19.31, 107.96]
Average (%):  58.03

#Means for cluster number 2 (size 232) for K=7:
{'age': 32.16, 'age_diff': 5.66, 'reach': 180.43, 'reach_diff': -3.19,
'won': 141.76, 'won_diff': 95.12, 'kos': 66.2, 'kos_diff': 45.19}
#Relative differences to global means (%):
[16.32, 124.91, 0.57, 127.59, 78.31, 96.57, 72.46, 94.8]
Average (%):  76.44

#Means for cluster number 3 (size 1623) for K=7:
{'age': 22.7, 'age_diff': -1.34, 'reach': 175.15, 'reach_diff': 0.42,
'won': 13.47, 'won_diff': -0.95, 'kos': 7.89, 'kos_diff': -0.03}
#Relative differences to global means (%):
[18.55, 5.22, 3.6, 109.52, 128.29, 443.16, 131.05, 7933.33]
Average (%):  1096.59

#Means for cluster number 4 (size 1312) for K=7:
{'age': 22.42, 'age_diff': -9.7, 'reach': 180.31, 'reach_diff': 1.67,
'won': 18.59, 'won_diff': -15.98, 'kos': 11.6, 'kos_diff': -7.78}
#Relative differences to global means (%):
[20.03, 85.46, 0.64, 47.31, 65.41, 120.4, 57.16, 130.21]
Average (%):  65.83

#Means for cluster number 5 (size 1403) for K=7:
{'age': 27.76, 'age_diff': -3.78, 'reach': 191.0, 'reach_diff': 4.19,
'won': 24.58, 'won_diff': -5.84, 'kos': 16.51, 'kos_diff': -2.92}
#Relative differences to global means (%):
[3.06, 62.7, 4.99, 79.0, 25.1, 155.82, 10.42, 180.48]
Average (%):  65.2

#Means for cluster number 6 (size 593) for K=7:
{'age': 37.07, 'age_diff': 7.63, 'reach': 190.29, 'reach_diff': 2.19,
'won': 46.79, 'won_diff': 19.86, 'kos': 30.75, 'kos_diff': 13.2}
#Relative differences to global means (%):
[27.41, 118.48, 4.64, 59.82, 34.28, 83.59, 40.72, 82.2]
Average (%):  56.39

#Means for cluster number 7 (size 843) for K=7:
{'age': 27.0, 'age_diff': -0.82, 'reach': 179.02, 'reach_diff': -1.42,
'won': 57.96, 'won_diff': 18.68, 'kos': 33.32, 'kos_diff': 13.74}
#Relative differences to global means (%):
[0.33, 71.95, 1.36, 161.97, 46.95, 82.55, 45.29, 82.9]
Average (%):  61.66
```

The features **age** and **reach** seem to present the lowest relative errors in both partitions, while **age_diff, won_diff, reach_diff** (occasionally) and **kos_diff** seem to have the highest values.
Partitions for **K=4** present lower relative errors to the global mean, and also they look more separate graphically, therefore my choice is for **K=4** as the most suited for interpretation of the 2.

Looking at the results for the **K=4** partition, we can see 4 categories of boxers:

1. First, those with the highest wins (average of 121.15) spanning all over the age range, represented by **cluster 2 with 376 objects**, but mainly scattered around their mean age of **31.11** (31 often considered the prime age of boxers).

2. Those with the lower wins (looking at some of them, they have around 50 wins, which is a great number in boxing, but here we don't take into account some other factors such as the quality of those wins, and so the algorithm is considering only this number, along with the other features) divided into 3 age groups themselves: young (mean **22.5**) represented by **cluster 4 with 2774 objects**, prime age (mean **27.5**) represented by **cluster 3 with 2871 objects**, and old (mean **34.8**) represented by **cluster 1 with 1134 objects**.

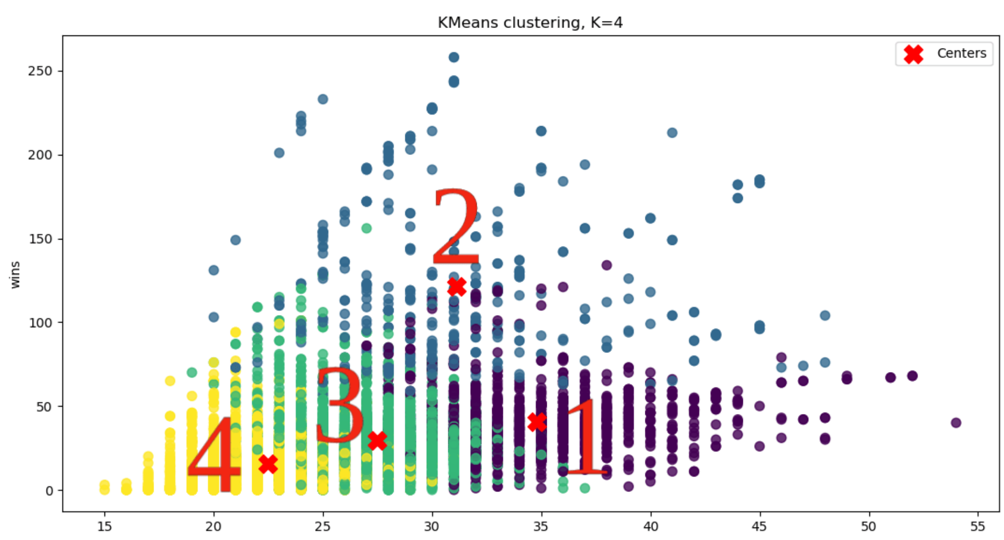

Figure 13: Numbered Kmeans clusters for K=4 visualised for features **age and won** with real data.

It seems, those with the highest number of wins are mainly those old and active enough to accumulate these wins with time, but nevertheless we can see some talents with high wins at relatively young age.
Now, looking at the computed means we can see some interesting observations:

- Boxers with high wins (cluster 2) tend to choose younger opponents (with an average age of 31 in this cluster, one could say that boxers 4.58 years younger are considered inexperienced), with much less wins and KOs (average wins difference is 72.6, and for KOs it's 36.6). So it seems the opponents are carefully chosen.

- Boxers with lower wins and old age (cluster 1): they choose younger opponents in their prime years, with less wins in general (14.46 average wins difference and 8.6 KOs difference with 40.48 average win record for this cluster, which seems acceptable)

- Boxers with lower wins and prime age (cluster 3): choosing tough opponents in their own age and win record level to keep their ranks or rise even more.

- Boxers with lower wins and young age (cluster 4): aspiring young boxers choosing older more experienced opponents (6.2 years older on average), with more wins (-9.37 win difference, -4.46 KOs difference) to rise in the ranks.

3.4 Contingency tables

In order to displays the multivariate frequency distribution of the variables, we form a contingency table with 4 bins for the two following chosen nominal features: **won** and **won_diff** to analyse the choice of opponents based on win records. The data has been discreticised into **4 bins** with the **quantile** strategy in Python, which means all bins in each feature have a similar number of points. The results are as follows:

```
For feature won, we have:
Bin number 1: [0, 13]
Bin number 2: [14, 23]
Bin number 3: [24, 36]
Bin number 4: [37, 258]
For feature won_diff, we have:
Bin number 1: [-205, -8]
Bin number 2: [-7, 0]
Bin number 3: [1, 9]
Bin number 4: [10, 211]
```

Figure 14: Bins interval ranges.

won_diff						
		G1	**G2**	**G3**	**G4**	**Total**
won	**H1**	379	694	596	13	1682
	H2	616	555	535	106	1812
	H3	410	343	519	543	1815
	H4	260	108	226	1252	1846
	Total	1665	1700	1876	1914	**7155**

Table 1: Contingency table

We deduce from the previous table, the table of relative frequencies by dividing every element by the total number of objects:

won_diff						
		G1	**G2**	**G3**	**G4**	**Total**
won	**H1**	0.05	0.1	0.08	0	0.24
	H2	0.09	0.08	0.07	0.01	0.25
	H3	0.06	0.05	0.07	0.08	0.25
	H4	0.04	0.02	0.03	0.17	0.26
	Total	0.23	0.24	0.26	0.27	1

Table 2: Frequency table

The conditional probability table is given by dividing every cell in the contingency table by the sum of elements of its corresponding row:

won_diff						
		G1	**G2**	**G3**	**G4**	**Total**
won	**H1**	0.23	0.41	0.35	0.01	1
	H2	0.34	0.31	0.3	0.06	1
	H3	0.23	0.19	0.29	0.3	1
	H4	0.14	0.06	0.12	0.68	1

Table 3: Conditional probability (or frequency) table

We can now compute the **Quetelet index table** defined as follows:

$$q_{ij} = q(j/i) = \frac{p(j/i) - p_j}{p_j} = \frac{NN_{ij}}{N_i N_j} - 1$$

Where $p(j/i) = p_{ij}/p_i$ is the (i, j) element from the conditional frequency table, and $p_j = N_j/N$ the probability for a column j.

won_diff					
		G1	**G2**	**G3**	**G4**
won	**H1**	-0.03	0.74	0.35	-0.97
	H2	0.46	0.29	0.13	-0.78
	H3	-0.03	-0.2	0.09	0.12
	H4	-0.39	-0.75	-0.53	1.54

Table 4: Quetelet index table

From here we can calculate:

Average Quetelet Index: $q_{avg} = \sum_{i,j} p_{ij} * q_{ij} = 0.39$

Chi-squared: $\Phi^2 = \sum_{i,j} \frac{(p_{ij} - p_i p_j)^2}{p_i p_j} = 0.39$

Number of degrees of freedom: $(N_{rows} - 1) * (N_{cols} - 1) = 9$

Pearson's Chi-squared: $N * \Phi^2 = 2771.9$; N the number of objects

Chi-squared or Φ^2 measures deviation from statistical independence. The average Quetelet index is interpreted as: the knowledge of a category of one feature (i) increases knowledge of (j) category by $100 * q_{avj}\% = 39\%$. So, knowing a boxer's win records gives a decent chance of knowing his opponent's win record, which makes sense.
According to chi-squared tables for 9 DF:

- For 95% confidence level we need 19.92/0.39 = 43.38 observations, so **44** minimum.

- For 99% confidence level we need 19.02/0.39 = 48.77 observations, so **49** minimum.

3.5 Bootstrap

Bootstrap is a statistical resampling technique used to estimate the sampling distribution of a statistic. Its main idea is to simulate multiple datasets that resemble the original data.

From the clustering section, we choose 2 clusters with the lowest average relative errors from the **K=4** partition. We perform bootstrap on these clusters with the nominal feature **won** (representing the number of wins).

My dataset has **N=7155 objects**. I generated a matrix of indices of size **5000*N**, where every line has N random indices between **0 and N-1**, then I calculated the averages for every line (for the chosen feature 'won') and stored it in a vector **mx**. To find a **95% confidence interval** for the **grand mean=30.75**, I used two method:

- **Pivotal method**:

  $$Ic = [mean(mx) - 1.96 * \sigma(mx),\ mean(mx) + 1.96 * \sigma(mx)] = [30.07,\ 31.42]$$

- **Non-pivotal method**: we sort the values of mx into **mxs**, then we remove 2.5% from each side of mxs, so we get:

  $$Ic = [mxs[125],\ mxs[4874]] = [30.08,\ 31.42]$$

To compare the means between clusters, or the mean of a cluster with the grand mean, we use a similar methodology. First, we construct 2 vectors of 5000 averages for each of the clusters. This is done as follows: for a chosen cluster CL_i, we through all the 5000 tries generated earlier, and for each try, containing N random indices, we select the indices that are in the cluster CL_i. Then we average these values, and we obtain one average for 1 try. We repeat this 5000 times to get the wanted vector. Let us call our chosen cluster, CL_1 and CL_2, and their respective vectors of averages $mx1$ and $mx2$.
We can compare the means of the clusters CL_1 and CL_2 by considering the vector $m12 = mx1 - mx2$, and then finding, as we previously did, a 95% confidence interval for the mean of this difference. If 0 is in that interval, we can accept the hypothesis that $mean(CL_1) = mean(CL_2)$.
Similarly, we compare the mean of one of the clusters to the grand mean (the latter represented by the vector mx).

After applying this, I got the following results:

- **Pivotal method**:

  $$Ic_{12} = [-12.03,\ -9.62]$$
  $$Ic_{1g} = [0.33,\ 1.81]$$
  $$Ic_{2g} = [-10.84,\ -8.68]$$

- **Non-pivotal method**:

  $$Ic_{12} = [-12.02,\ -9.6]$$
  $$Ic_{1g} = [0.34,\ 1.8]$$
  $$Ic_{2g} = [-10.8,\ -8.65]$$

So we can conclude that the means of the clusters and the grand mean are all different. None of the equality hypotheses were accepted since 0 is not in any of the confidence intervals found with our samples. The closest hypothesis to being true was: $mean(CL_1) = grand\ mean$.

```
Bootstrap method:
Confidence interval for the grand mean:
Pivotal Ic=[30.07344669472215, 31.420212620442072]
Non-pivotal Ic=[30.07756813417191, 31.41928721174004]
Grand mean:  30.749685534591194
Comparing cl1 and cl2 means:
Pivotal Ic=[-12.025649750535875, -9.62108178945205]
Non-pivotal Ic=[-12.025038132843271, -9.608605371882437]
Comparing cl1 and grand mean:
Pivotal Ic=[0.3269598652708896, 1.8094592653227726]
Non-pivotal Ic=[0.3431111051148683, 1.801782950292914]
Comparing cl2 and grand mean:
Pivotal Ic=[-10.83507999736562, -8.675232412028643]
Non-pivotal Ic=[-10.804203526650433, -8.645049329478887]
```

Figure 15: Bootstrap results.

4 Conclusion

By analysing this boxing dataset, I was able to visualise and analyse some of the effects of important factors in the sport such as win record, losses, number of KOs, age, height, reach, etc... relying on different data analysis tools.
I was also able to notice some interesting but known trends in the data when it comes to the choice of opponents for different groups of boxers.

5 Appendix

5.1 Python code

```python
#Data preprocessing: Line 20
#Correlation: Line 48
#PCA: Line 92
#KMeans Clustering: Line 188
#Contingency Table: Line 310
#Bootstrap: Line 368
#Main program: Line 462

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
from scipy.stats import pearsonr
from sklearn.cluster import KMeans
from sklearn.preprocessing import KBinsDiscretizer

#Cluster indices for bootstrap
cl1_indices = []
cl2_indices = []

def preprocessing():
    try:
        path = "Datasets/Boxing/"
        file_name = 'boxing_matches.csv'
        df = pd.read_csv(path+file_name)
        ignore_columns = ['judge1_A','judge1_B','judge2_A','judge2_B','judge3_A',
        'judge3_B']
        df.dropna(subset=df.columns.difference(ignore_columns), inplace=True)
        df.reset_index(drop=True, inplace=True)
        df['result'] = df['result'].replace('win_A','win').replace('win_B', 'loss')
        #Categories
        #print(df['result'].value_counts())
        cols = df.columns.tolist()
        for i,col in enumerate(cols):
            if ('_B' in col) and (df[col].dtype.kind in 'iufc'):
                df[col] = df[cols[i-1]] - df[col]
        new_cols = {col:col.replace('_A','').replace('_B','_diff') for col in cols}
        df.rename(columns=new_cols, inplace=True)
        #Reach of more than 4m isnt acceptable, drop rows
        df.drop(labels=[2805,2963,2893], axis=0, inplace=True)
        df.reset_index(drop=True, inplace=True)
        df.to_csv(path+file_name.replace('.csv', '_updated.csv'), index=False)
        return df
```

```
    except Exception as e:
        print(r'Unable to read data: {}'.format(str(e)))
        return pd.DataFrame()


def correlation(df):
    try:
        print('Correlation:')
        #plt.matshow(df.corr(numeric_only=True))
        #plt.show()
        #pd.plotting.scatter_matrix(df, alpha=0.5, figsize=(6, 6), diagonal='hist')
        X = df['height'].to_numpy()
        Y = df['reach'].to_numpy()
        #Reach of more than 4m is irrational, drop those rows
        #Rho from library
        rho = pearsonr(X,Y).statistic
        #Rho built with formula
        rho_ = sum([(X[i]-np.mean(X))*(Y[i]-np.mean(Y)) for i in range(len(X))])/
        (len(X)*np.std(X)*np.std(Y))

        print('Library rho = {}, calculated rho={}'.format(rho,rho_))
        a = rho*np.std(Y)/np.std(X)
        b = np.mean(Y) - a*np.mean(X)
        print('a={}, b={}'.format(a,b))
        print('Correlation coeff: {}, determinancy coeff: {}'.format(rho,rho**2))
        #Scatter-plot
        plt.plot(X, [a*x+b for x in X], color='crimson', label='Linear regression')
        plt.scatter(X, Y, color='blue', label='Data points')
        plt.xlabel('Height (in cm)')
        plt.ylabel('Reach (in cm)')
        plt.legend(loc='upper left')
        plt.title('Scatter-plot of height and reach for boxers')
        plt.show()
        #Compute errors
        labels = [i for i in range(len(X))]
        error_DS = [100*abs(a*x+b-Y[i])/abs(Y[i]) for i, x in enumerate(X)]
        error_ML = [100*abs(a*x+b-Y[i])/abs(a*x+b) for i, x in enumerate(X)]
        plt.plot(labels, error_DS, color='crimson', label='Data Analysis Error')
        plt.plot(labels, error_ML, color='darkblue', label='Machine Learning Error')
        plt.xlabel('Objects')
        plt.ylabel('Error (in %)')
        plt.legend(loc='upper right')
        plt.title('Regression errors')
        plt.show()
    except Exception as e:
        print(r'Unable to perform linear regression: {}'.format(str(e)))
```

```python
def PCA(df):
    try:
        print('\n\nPrincipal component analysis:')
        features = ['age','age_diff','won','won_diff','lost','lost_diff']
        PCA_df_z = pd.DataFrame()
        PCA_df_r = pd.DataFrame()
        HiddenFactor_df = pd.DataFrame()

        for feature in features:
            #Standardization: z-scoring
            sigma = np.std(df[feature])
            mean = np.mean(df[feature])
            max = df[feature].max()
            min = df[feature].min()
            PCA_df_z[feature] = df[feature].apply(lambda x: (x-mean)/sigma)
            PCA_df_r[feature] = df[feature].apply(lambda x: (x-mean)/(max-min))
            HiddenFactor_df[feature] = df[feature].apply(lambda x: 100*(x-min)/(max-min))


        #Visualisation indices
        df_inf = df.loc[df['age']>=40]
        df_sup = df.loc[df['age']<=25]
        #print(df_inf.index.values.tolist())
        inf_indices = df_inf.index.values.tolist()
        sup_indices = df_sup.index.values.tolist()
        med_indices = [i for i in range(len(df)) if i not in inf_indices + sup_indices]

        #####################################
        ########### PCA z-scoring ############
        #####################################

        print('z-scoring')
        X_z = PCA_df_z.to_numpy()
        Z_z, mu_z, C_z = np.linalg.svd(X_z, full_matrices=True)
        print(np.around(Z_z,decimals=3))
        print('\n')
        print(np.around(mu_z,decimals=3))
        print('\n')
        print(np.around(C_z,decimals=3))
        #Data scatter
        tm_z= sum([mu_i**2 for mu_i in mu_z])
        t_z= np.sum(X_z**2)
        print('\nData scatter with z-scoring: t_mu={}, t={}'.format(tm_z,t_z))
```

```python
print('mu_z in percentage', [round(100*mu_i**2/t_z, 2) for mu_i in mu_z])
Z_1 = Z_z[:,0]
Z_2 = -Z_z[:,1]
plt.scatter(Z_1[inf_indices], Z_2[inf_indices], color='red', label="age <= 25")
plt.scatter(Z_1[sup_indices], Z_2[sup_indices], color='green', label="age >= 40")
plt.scatter(Z_1[med_indices], Z_2[med_indices], color='darkblue')
plt.xlabel('Principal component 1')
plt.ylabel('Principal component 2')
plt.legend(loc='upper right')
plt.title('PCA with z-scoring')
plt.show()

#######################################
###### PCA Range Normalisation #######
#######################################

print('\n\nRange normalisation')
X_r = PCA_df_r.to_numpy()
Z_r, mu_r, C_r = np.linalg.svd(X_r, full_matrices=True)
print(np.around(Z_r,decimals=3))
print('\n')
print(np.around(mu_r,decimals=3))
print('\n')
print(np.around(C_r,decimals=3))
#Data scatter
tm_r = sum([mu_i**2 for mu_i in mu_r])
t_r = np.sum(X_r**2)
print('\nData scatter with range normalisation: t_mu={}, t={}'.format(tm_r,t_r))
print('mu_r in percentage', [round(100*mu_i**2/t_r,2) for mu_i in mu_r])
Z_1 = -Z_r[:,0]
Z_2 = -Z_r[:,1]
plt.scatter(Z_1[inf_indices], Z_2[inf_indices], color='red', label="age <= 25")
plt.scatter(Z_1[sup_indices], Z_2[sup_indices], color='green', label="age >= 40")
plt.scatter(Z_1[med_indices], Z_2[med_indices], color='darkblue')
plt.xlabel('Principal component 1')
plt.ylabel('Principal component 2')
plt.legend(loc='upper right')
plt.title('PCA with range normalisation')
plt.show()

#######################################
########### Hidden Factor ############
#######################################
X_h = HiddenFactor_df.to_numpy()
Z_h, mu_h, C_h = np.linalg.svd(X_h, full_matrices=True)
print(np.around(C_h, decimals=3))
```

```
        C_1 = -C_h[:,0]
        alpha = 1/sum(C_1)
        print(alpha)
    except Exception as e:
        print(r'Unable to apply PCA: {}'.format(str(e)))


def KMeansClustering(df):
    try:
        print('\n\nKMeans Clustering:')
        features = ['age','age_diff','reach','reach_diff','won','won_diff','kos',
        'kos_diff']
        KMeans_df = pd.DataFrame()
        global_means = {}
        #Standardisation of the data
        for feature in features:
            #Standardization: z-scoring
            sigma = np.std(df[feature])
            mean = np.mean(df[feature])
            global_means[feature] = round(mean,2)
            max = df[feature].max()
            min = df[feature].min()
            KMeans_df[feature] = df[feature].apply(lambda x: (x-mean)/(max-min))
            #KMeans_z_df[feature] = df[feature].apply(lambda x: 100*(x-mean)/(sigma))

        #Clustering
        X = KMeans_df.to_numpy()
        #X_z = KMeans_z_df
        K_list = [4,7]
        n_iterations = 10
        best_iterations = {}

        for K in K_list:
            kmeans = KMeans(n_clusters=K,init='random', n_init=10)
            inertia = []
            centers = []
            labels = []
            index_best = 0
            avg_rel_errs = []
            for i in range(n_iterations):
                kmeans.fit(X)
                inertia.append(kmeans.inertia_)
                centers.append(kmeans.cluster_centers_)
                labels.append(kmeans.labels_)
                if inertia[i]<inertia[index_best]:
                    index_best = i
```

```
        #Plot inertia
        plt.plot(range(10), inertia, marker='o')
        plt.xlabel('Iterations')
        plt.ylabel('Inertia')
        plt.title('Inertia over different iterations for K={}'.format(K))
        plt.show()
        #plot clusters
        #Age for x axis
        x_feat = 0
        #Number of wins for y axis
        y_feat = 4
        #Normalised centers
        centers_ = centers[index_best]
        #Real data
        X_real = df[features].to_numpy()
        #Real centers
        centers_real = np.empty(shape=(K,len(features)))
        for j, feature in enumerate(features):
            mean = np.mean(df[feature])
            max = df[feature].max()
            min = df[feature].min()
            for i in range(K):
                centers_real[i,j] = centers_[i,j]*(max-min)+mean

        #Visualisation
        #Normalised data
        #plt.scatter(X[:,x_feat],X[:,y_feat],c=labels[index_best],cmap='viridis',
        s=50,alpha=0.8)
        #plt.scatter(centers_[:,x_feat],centers_[:,y_feat],marker='X',color='red',
        s=200,label='Centers')
        #plt.xlabel('age')
        #plt.ylabel('wins')
        #plt.legend()
        #plt.title('KMeans clustering, K={}'.format(K))
        #plt.show()

        #Real data
        plt.scatter(X_real[:,x_feat],X_real[:,y_feat],c=labels[index_best],
        cmap='viridis',s=50,alpha=0.8)
        plt.scatter(centers_real[:,x_feat],centers_real[:,y_feat],marker='X',
        color='red',s=200,label='Centers')
        plt.xlabel('age')
        plt.ylabel('wins')
        plt.legend()
        plt.title('KMeans clustering, K={}'.format(K))
```

```
            plt.show()

            #Compare Cluster means
            #Get separate clusters
            print('Global means:')
            print(global_means)
            for j in range(K):
                means = {}
                for feature in features:
                    #Standardization: z-scoring
                    mean = np.mean(df[feature])
                    max = df[feature].max()
                    min = df[feature].min()
                    means[feature] = round(np.mean(df[feature].iloc
                    [labels[index_best] == j]),2)

                cl_size = len(df.iloc[labels[index_best] == j])
                print('Means for cluster number {} (size {}) for K={}:'.format(
                j+1,cl_size,K))
                print(means)
                relative_err = [round(100*abs(means[feature]-global_means[feature])/
                abs(means[feature]),2) for feature in features]

                print('Relative differences to global means (%): ')
                print(relative_err)
                print('Average (%): ', round(sum(relative_err)/len(relative_err),2))
                avg_rel_errs.append(sum(relative_err)/len(relative_err))
            #Choose clusters for Bootstrap
            if (K==4):
                global cl1_indices
                cl1_indices = labels[index_best] == np.argsort(avg_rel_errs)[0]
                global cl2_indices
                cl2_indices = labels[index_best] == np.argsort(avg_rel_errs)[1]
    except Exception as e:
        print(r'Unable to read data: {}'.format(str(e)))


def ContingencyTable(df):
    try:
        print('\n\nContingency table:')
        features = ['won','won_diff']
        N_Bins = 4
        N = len(df)
        Kbins_discret = KBinsDiscretizer(n_bins=N_Bins,encode='ordinal',
        strategy='quantile')
        binned_data = Kbins_discret.fit_transform(df[features])
```

```
        df_binned = pd.DataFrame(binned_data, columns=features)
        #Print bins interval ranges
        for feature in features:
            bins_indices  = [df_binned[df_binned[feature]==x].index.tolist()
            for x in [0.0,1.0,2.0,3.0]]
            print('For feature {}, we have:'.format(feature))
            for i, indices in enumerate(bins_indices):
                print('Bin number {}: [{}, {}]'.format(i+1,
                df[feature].iloc[indices].min(),df[feature].iloc[indices].max()))
        print('\n')
        c_table = pd.crosstab(df_binned[features[0]], df_binned[features[1]])
        print(c_table)
        X = c_table.to_numpy()
        CT1, CT2 = [], []
        for i in range(N_Bins):
            CT1.append(np.sum(X[:,i]))
            CT2.append(np.sum(X[i,:]))
        #Sum over columns
        sum_cols = np.array(CT1)
        #Sum over rows
        sum_rows = np.array(CT2)
        SumRowsMat = np.transpose(np.array([CT2,CT2,CT2,CT2]))
        #print(SumRowsMat)

        #Conditional probability
        print('Conditional frequency table:')
        cp_table = np.divide(X,SumRowsMat)
        #cp_table = np.divide(X,N)
        print(np.round(cp_table, decimals=2))
        #Quetelet index table and Pearson's Chi-squared
        print('Quetelet index table:')
        q_mat = np.empty(shape=(4,4))
        pearson_mat = np.empty(shape=(4,4))
        for i in range(N_Bins):
            for j in range(N_Bins):
                q_mat[i,j] = (N*X[i,j])/(sum_rows[i]*sum_cols[j])-1
                pearson_mat[i,j] = (X[i,j]-sum_rows[i]*sum_cols[j]/N)**2/
                (sum_rows[i]*sum_cols[j])

        print(np.round(q_mat,2))
        print('Average Quetelet index: ', round(np.sum(q_mat*np.divide(X,N)),2))
        print('Chi-squared: ', round(np.sum(pearson_mat),2))
        print('Degrees of freedom: ', 9)
        print('Pearson Chi-squared: ', round(N*np.sum(pearson_mat),2))

    except Exception as e:
```

```
            print(r'Unable to perform contingency table calculations: {}'.format(str(e)))


    def Bootstrap(df):
        try:
            print('\n\nBootstrap method:')
            N_iter = 5000
            N = len(df)
            feature = 'won'
            cl1 = df.iloc[cl1_indices]
            cl2 = df.iloc[cl2_indices]
            cl1_idx = cl1.index.tolist()
            cl2_idx = cl2.index.tolist()
            indices = np.empty(shape=(N_iter,N), dtype=int)
            means = []
            means_cl1 = []
            means_cl2 = []
            for i in range(N_iter):
                indices[i,:] = np.random.randint(0, N, size=N, dtype=int)
                means.append(np.mean(df[feature].iloc[indices[i,:]]))
                L1 = []
                L2 = []
                for index in indices[i,:].tolist():
                    if(cl1_indices[index]):
                        L1.append(df[feature].iloc[index])
                    if(cl2_indices[index]):
                        L2.append(df[feature].iloc[index])

                if(L1):
                    means_cl1.append(sum(L1)/len(L1))
                else:
                    means_cl1.append(0)
                if(L2):
                    means_cl2.append(sum(L2)/len(L2))
                else:
                    means_cl2.append(0)

            #Confidence intervals
            print('Confidence interval for the grand mean:')
            #Pivotal
            lp = np.mean(means)-1.96*np.std(means)
            rp = np.mean(means)+1.96*np.std(means)
            print('Pivotal Ic=[{}, {}]'.format(lp,rp))
            #Non pivotal
            sorted_means = means[:]
            sorted_means.sort()
```

```python
            lnp = sorted_means[125]
            rnp = sorted_means[4874]
            print('Non-pivotal Ic=[{}, {}]'.format(lnp,rnp))
            #Grand mean
            print('Grand mean: ',np.mean(df[feature]))

            #Compare cl1 and cl2 means
            #Pivotal
            m12 = [means_cl1[i] - means_cl2[i] for i in range(N_iter)]
            print('Comparing cl1 and cl2 means:')
            lp = np.mean(m12)-1.96*np.std(m12)
            rp = np.mean(m12)+1.96*np.std(m12)
            print('Pivotal Ic=[{}, {}]'.format(lp,rp))
            #Non pivotal
            sorted_means = m12[:]
            sorted_means.sort()
            lnp = sorted_means[125]
            rnp = sorted_means[4874]
            print('Non-pivotal Ic=[{}, {}]'.format(lnp,rnp))

            #Compare cl1 and grand mean
            #Pivotal
            m1g = [means[i] - means_cl1[i] for i in range(N_iter)]
            print('Comparing cl1 and grand mean:')
            lp = np.mean(m1g)-1.96*np.std(m1g)
            rp = np.mean(m1g)+1.96*np.std(m1g)
            print('Pivotal Ic=[{}, {}]'.format(lp,rp))
            #Non pivotal
            sorted_means = m1g[:]
            sorted_means.sort()
            lnp = sorted_means[125]
            rnp = sorted_means[4874]
            print('Non-pivotal Ic=[{}, {}]'.format(lnp,rnp))

            #Compare cl1 and cl2 means
            #Pivotal
            m2g = [means[i] - means_cl2[i] for i in range(N_iter)]
            print('Comparing cl2 and grand mean:')
            lp = np.mean(m2g)-1.96*np.std(m2g)
            rp = np.mean(m2g)+1.96*np.std(m2g)
            print('Pivotal Ic=[{}, {}]'.format(lp,rp))
            #Non pivotal
            sorted_means = m2g[:]
            sorted_means.sort()
            lnp = sorted_means[125]
            rnp = sorted_means[4874]
```

```
        print('Non-pivotal Ic=[{}, {}]'.format(lnp,rnp))
    except Exception as e:
        print(r'Unable to perform Bootstrap: {}'.format(str(e)))


if __name__=='__main__':
    df = preprocessing()
    correlation(df)
    PCA(df)
    KMeansClustering(df)
    ContingencyTable(df)
    Bootstrap(df)
```