

Правительство Российской Федерации
Федеральное государственное автономное образовательное учреждение
высшего профессионального образования
«Национальный исследовательский университет
“Высшая школа экономики”»

АНАЛИЗ СТРАН МИРА

ANALYSIS OF COUNTRIES OF THE WORLD

Homework Project 2018/2019

Команда «Fortune»
Маслов Дмитрий Васильевич
Эйдлина Мария Андреевна
Володкевич Анна Алексеевна

Учебная группа мНОД18
1 курс магистратуры
Программа: «Науки о данных»
Факультет компьютерных наук

“Fortune” team:
Dmitry Maslov
Maria Eidlina
Anna Volodkevich

Group mNOD18
MSc Program “Data Science”
1st Year

Moscow, 2018

Table of Contents

A1 Starting Report: Data Description	3
A2: K-means clustering	5
1. Features selection	5
2. K-means application	5
3. Results interpretation	7
3. Analysis with Bootstrap	9
A3: Contingency Table Analysis	12
1. Nominal features selection	12
2. Contingency tables building	13
3. Computation of summary Quetelet index	16
4. Hypothesis testing	16
A4 PCA: Hidden Factor & Data Visualization	17
1. Selection of a subset	17
2. Data scatter and contribution of principal components	17
3. Data visualization	17
4. Hidden factor computation and interpretation	19
A5: 2D Regression	20
Appendix 1	23
A2.1 Source code	23
A2.2 Source code	25
A3 Source code	27
A4 Source code	30
A5 Source code	35

A1 Starting Report: Data Description

The dataset used represents information about countries, located in different regions, their population, climate and economical indicators. The dataset contains both categorical (e.g. climate, region) and numerical features (e.g. Gross domestic product (GDP)), which, in our opinion, are in sufficient quantity to perform qualitative data analysis.

We assume that we will find some dependencies between the climate, GDP sources, and economics of different regions and between GDP and life quality (literacy, infant mortality). The dataset analysis may help to understand the components of the countries' economic success and indicate the regions with the best and worst economic and life quality state.

Data is taken from the site [kaggle.com](https://www.kaggle.com/fernandol/countries-of-the-world) (URL: <https://www.kaggle.com/fernandol/countries-of-the-world>). This dataset contains 227 countries, but there are some missing fields. Overall, there are 19 features in dataset. Complete data is available for 179 countries, which satisfies condition of the problem. The following is a list of the used features in dataset:

Feature	Description
Country	Name of the country
Region	Region the country belongs to
Population	Number of humans, living in the country
Area	Land area of the country (sq. mi.)
Pop. Density	A measurement of population per unit area or unit volume
Coastline	Area, where land meets sea or ocean (coast/area ratio)
Net Migration	The difference between the number of immigrants (people coming into an area) and the number of emigrants (people leaving an area) throughout the year.
Infant Mortality	The number of deaths per 1,000 live births of children under one year of age.
GDP	Gross domestic product (\$ per Capita)
Literacy	Number of people, that are able to read and write (in %)
Arable	The area of a land which is capable of being ploughed and used to grow crops. (in %)

Crops	The area of a land which is permanently used for growing crops. (in %)
Other	Left area of a land, which is not for arable or crops. (in %)
Climate	The type of climate.
Birth rate	The total number of live births in a country in a year or period.
Death rate	The total number of deaths in a country in a year or period.
Agriculture	Agricultural component of the GDP of a nation.
Industry	Industrial component of the GDP of a nation.
Service	Service component of the GDP of a nation.

A piece of data is presented below:

	Country	Region	Population	Area (sq. mi.)	Pop. Density (per sq. mi.)	Coastline (coast/area ratio)	Net migration	Infant mortality (per 1000 births)
0	Afghanistan	2	31056997	647500	48.000	0.000	23.060	163.070
1	Albania	7	3581655	28748	124.600	1.260	-4.930	21.520
2	Algeria	8	32930091	2381740	13.800	0.040	-0.390	31.000
3	Anguilla	1	13477	102	132.100	59.800	10.760	21.030
4	Antigua & Barbuda	1	69108	443	156.000	34.540	-6.150	19.460

	GDP (\$ per capita)	Literacy (%)	Arable (%)	Crops (%)	Other (%)	Climate	Birthrate	Deathrate	Agriculture	Industry	Service
	700.000	36.000	12.130	0.220	87.650	1.000	46.600	20.340	0.380	0.240	0.380
	4500.000	86.500	21.090	4.420	74.490	3.000	15.110	5.220	0.232	0.188	0.579
	6000.000	70.000	3.220	0.250	96.530	1.000	17.140	4.610	0.101	0.600	0.298
	8600.000	95.000	0.000	0.000	100.000	2.000	14.170	5.340	0.040	0.180	0.780
	11000.000	89.000	18.180	4.550	77.270	2.000	16.930	5.370	0.038	0.220	0.743

A2: K-means clustering

1. Features selection

We have selected the following features to perform K-means:

- Population,
- GDP (\$ per capita),
- Birth rate,
- Death rate,
- Population Density

We are intending to find clusters representing countries partition by economical performance and population size tendencies, e.g. european developed countries with good economic performance and low birth and death rates or asian developing countries with medium GDP, big population destiny and birth rate.

2. K-means application

We have performed k-means clustering for $k = 5, 9$ using KMeans method of sklearn.cluster Python library. We performed preprocessing using StandardScaler of sklearn.preprocessing Python library, which standardize features by removing the mean and scaling to unit variance. We have selected the lowest k-means criterion value for 10 iteration (with 1000 runs) of k-means for each k. (See the source code in Appendix 1. A2.1).

The following clusters were found for $k = 5$ (k-means criterion value = 215.8381):

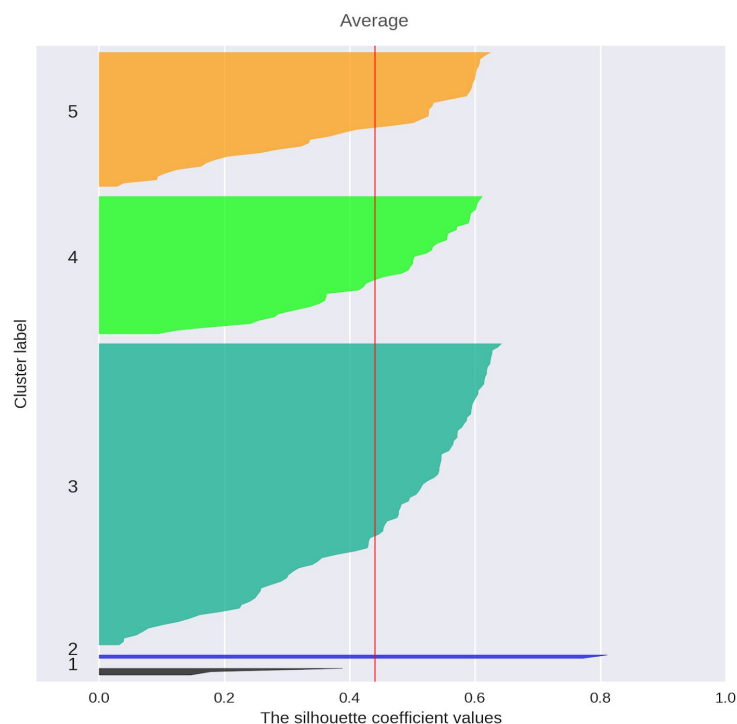
	Population	GDP (\$ per capita)	Birthrate	Deathrate	Pop. Density (per sq. mi.)
Cluster 1	3961902.333	23966.667	8.370	5.013	9673.633
Cluster 2	1204662854.000	3950.000	17.630	7.575	235.050
Cluster 3	23280250.385	5326.374	21.295	6.942	126.364
Cluster 4	16316436.595	1959.524	38.401	16.805	62.767
Cluster 5	21934834.805	24065.854	12.635	7.964	223.020
Grand Mean	34214149.520	9125.698	23.067	9.465	294.805

The following clusters were found for $k = 9$ (k-means criterion value =102.5877):

	Population	GDP (\$ per capita)	Birthrate	Deathrate	Pop. Density (per sq. mi.)
Cluster 1	26613258.077	4613.462	21.139	5.895	102.779
Cluster 2	10544514.500	6116.667	24.295	25.108	34.133
Cluster 3	21120747.160	1040.000	42.037	17.087	65.064
Cluster 4	1204662854.000	3950.000	17.630	7.575	235.050
Cluster 5	5716291.000	26250.000	8.315	5.285	6418.950
Cluster 6	20396246.556	2329.630	34.727	9.299	154.996
Cluster 7	9842271.316	11857.895	13.986	8.225	178.395
Cluster 8	30893892.154	28369.231	11.680	8.310	197.665
Cluster 9	453125.000	19400.000	8.480	4.470	16183.000
Grand Mean	34214149.520	9125.698	23.067	9.465	294.805

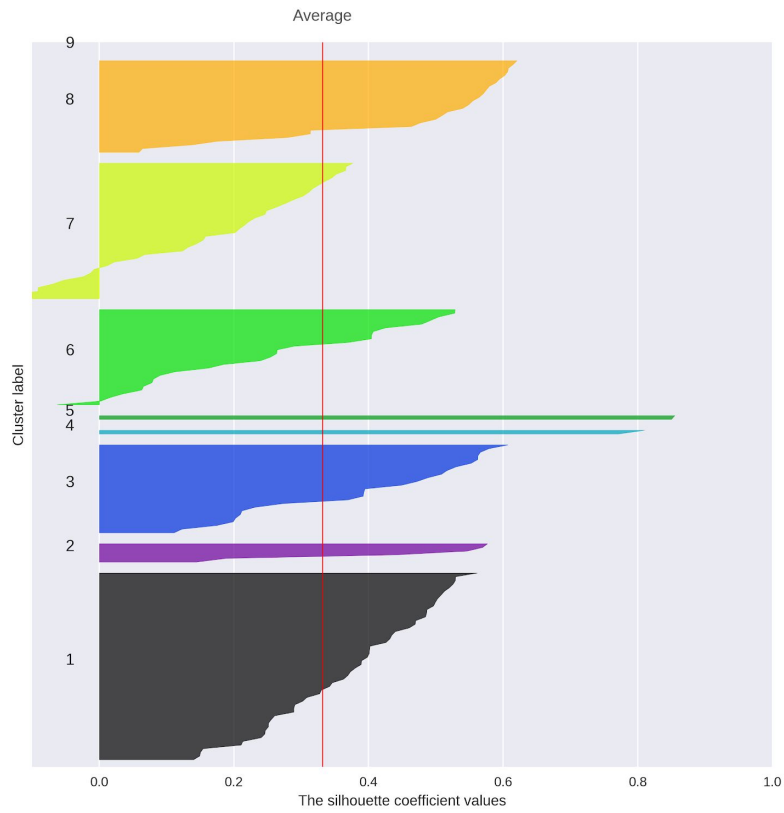
To build silhouettes, we used the Python language library `kmeansplots`. The average silhouette for a partition with $k = 5$ is larger than for a partition with $k = 9$. It follows from this conclusion that the first partition is preferable from the point of view of the silhouette criterion. In addition, there is a negative “tail” for the 6th and 7th clusters of the second partition, which indicates a poor division of these clusters.

Silhouette Analysis for KMeans Clustering $k = 5$



Silhouette plot for the partition with $k = 5$

Silhouette Analysis for KMeans Clustering k = 5



Silhouette plot for the partition with k = 9

3. Results interpretation

We calculated related differences percent using $c_{kv}/c_v - 1$ formula to interpret the clusters for k = 5:

	Population	GDP (\$ per capita)	Birthrate	Deathrate	Pop. Density (per sq. mi.)
Cluster 1	-88.420	162.628	-63.715	-47.034	3181.366
Cluster 2	3420.949	-56.716	-23.572	-19.969	-20.269
Cluster 3	-31.957	-41.633	-7.685	-26.656	-57.137
Cluster 4	-52.311	-78.527	66.472	77.551	-78.709
Cluster 5	-35.890	163.715	-45.224	-15.863	-24.350
Grand Mean	0.000	0.000	0.000	0.000	0.000

Cluster	Interpretation
Cluster 1	Countries with very big GDP and population destiny, e.g.Hong Kong.
Cluster 2	Countries with very big population, e.g.China
Cluster 3	Countries with medium population, GDP and birth and death rates, e.g.Brazil
Cluster 4	Poor countries with big birth and death rates, e.g.Afghanistan
Cluster 5	Countries with very big GDP and normal population destiny, e.g.Finland.

We calculated related differences percent for interpret the clusters for $k = 9$:

	Population	GDP (\$ per capita)	Birthrate	Deathrate	Pop. Density (per sq. mi.)
Cluster 1	-22.216	-49.445	-8.358	-37.717	-65.137
Cluster 2	-69.181	-32.973	5.321	165.272	-88.422
Cluster 3	-38.269	-88.604	82.234	80.523	-77.930
Cluster 4	3420.949	-56.716	-23.572	-19.969	-20.269
Cluster 5	-83.293	187.649	-63.954	-44.164	2077.354
Cluster 6	-40.387	-74.472	50.545	-1.756	-47.424
Cluster 7	-71.233	29.940	-39.371	-13.099	-39.487
Cluster 8	-9.704	210.872	-49.364	-12.200	-32.950
Cluster 9	-98.676	112.586	-63.238	-52.774	5389.391
Grand Mean	0.000	0.000	0.000	0.000	0.000

We concluded that with k growths the partition was detailed, some clusters remained unchanged, e.g. clusters of countries with very big population (Cluster 4), or countries with very big GDP and population destiny (Cluster 9), and new appeared, e.g. a group of countries with small population and big death rate (Cluster 2), or countries with big birth rate and small death rate (Cluster 6).

Thus, the results of both partitions are meaningful and no one is significantly better, but partition for $k = 5$ is simpler and easy to understand and therefore more preferable. This is confirmed by the built silhouette plots.

3. Analysis with Bootstrap

3.1. Compare one of the features between two clusters with using bootstrap

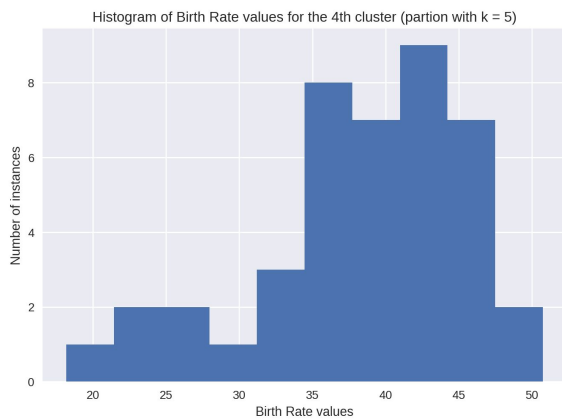
We have selected birth rate feature and clusters 4 and 5 of k-means partition with $k = 5$. We have made 5000 bootstrap samples for each feature, calculated means of each sample and calculated cluster differences. (See the source code in Appendix 1. A2.2).

Means and standard deviation for two clusters (birth rate feature):

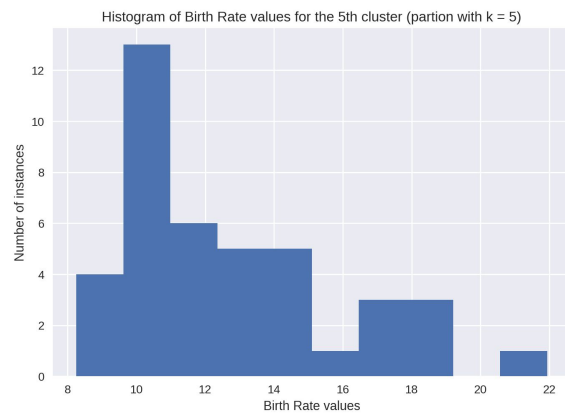
- Cluster4: mean = 38.4009, std = 7.2263;
- Cluster5: mean = 12.6354, std = 3.2287.

Feature difference between two clusters:

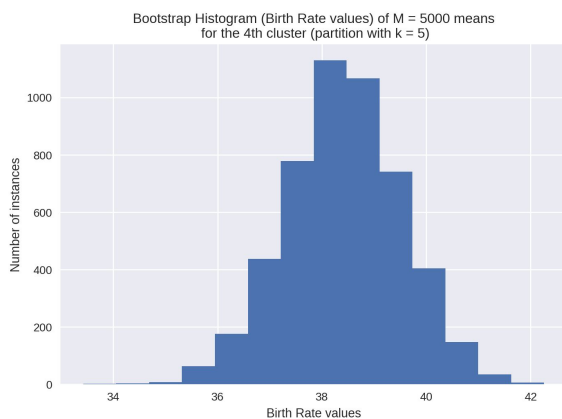
- Pivotal 95% confidence interval for the difference: [23.2883, 28.1165]
- Non Pivotal 95% confidence interval for the difference: [23.2889, 28.1228]



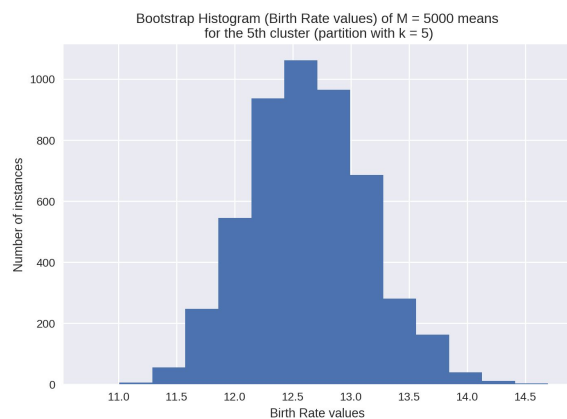
Histogram of Birth Rate for the 4th cluster
(partition with $k = 5$)



Histogram of Birth Rate for the 5th cluster
(partition with $k = 5$)



Bootstrap histogram (Birth Rate) of $M = 5000$ means for the 4th cluster (partition with $k = 5$)



Bootstrap histogram (Birth Rate) of $M = 5000$ means for the 5th cluster (partition with $k = 5$)

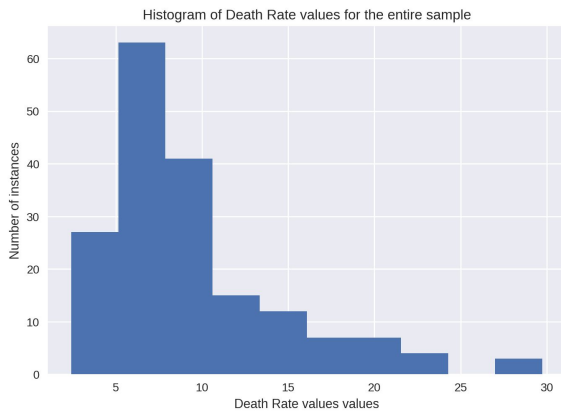
We calculated confidence intervals for the difference between two classes for the values of the feature. (pivotal/ non pivotal methods) This intervals don't include a zero value. Therefore, we concluded that the values of this feature (birth rate) are different between two clusters.

3.2. Take a feature, find the 95% confidence interval for its grand mean by using bootstrap

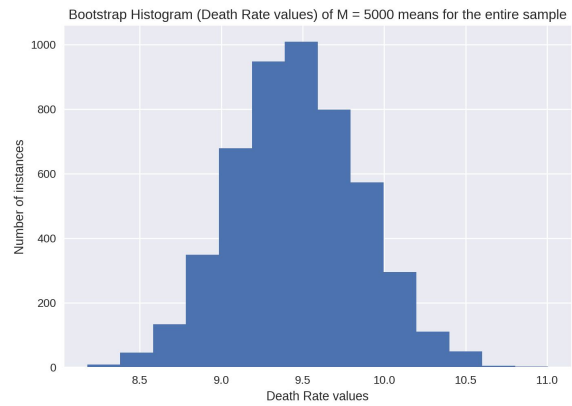
We have selected death rate feature to find the 95% confidence interval for its grand mean. (See the source code in Appendix 1. A2.2).

The grand mean and standard deviation of death rate (for the entire sample): mean = 9.4651, std = 5.1955.

- Pivotal 95% confidence interval for the grand mean of death rate: [8.7237, 10.2392]
- Non Pivotal 95% confidence interval for the grand mean of death rate: [8.7244, 10.2424].



Histogram of Death Rate for the entire sample



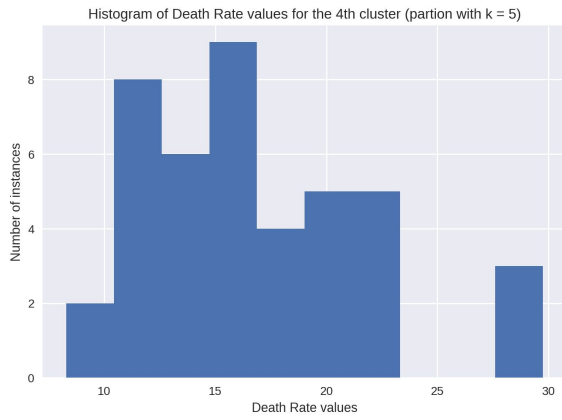
Bootstrap histogram (Death Rate) of M = 5000 means for the entire sample

3.3. Take a cluster, and compare the grand mean with the within-cluster mean for the feature by using bootstrap

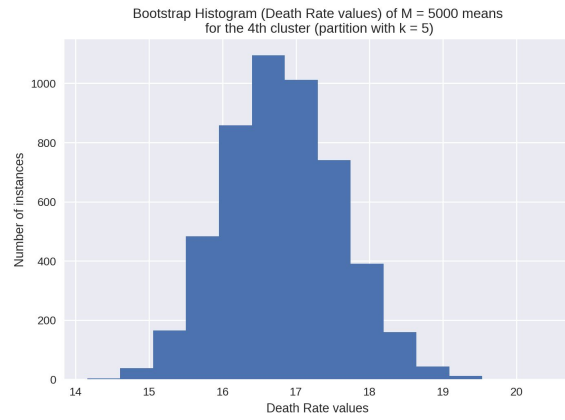
We have selected death rate feature and cluster 4 of k-means partition with k = 5 to compare the grand mean with the within-cluster mean confidence intervals for the feature.

The mean and standard deviation of death rate (for the 4th cluster): mean = 16.8055, std = 5.065.

- Pivotal 95% confidence interval for the cluster mean: [15.3590, 18.3781]
- Non Pivotal 95% confidence interval for the cluster mean: [15.3600, 18.3793]



Histogram of Death Rate for the 4th cluster
(partition with $k = 5$)



Bootstrap histogram (Death Rate) of $M = 5000$ means for the 4th cluster (partition with $k = 5$)

Comparing the intervals with grand mean interval (see 3.2) we conclude that the intervals for grand mean and for cluster mean do not intersect (both for pivotal and non-pivotal method).

A3: Contingency Table Analysis

1. Nominal features selection

We have selected the following features to perform contingency tables analysis:

- Climate (originally nominal)
- Agriculture
- Arable (%)

Climate categories are:



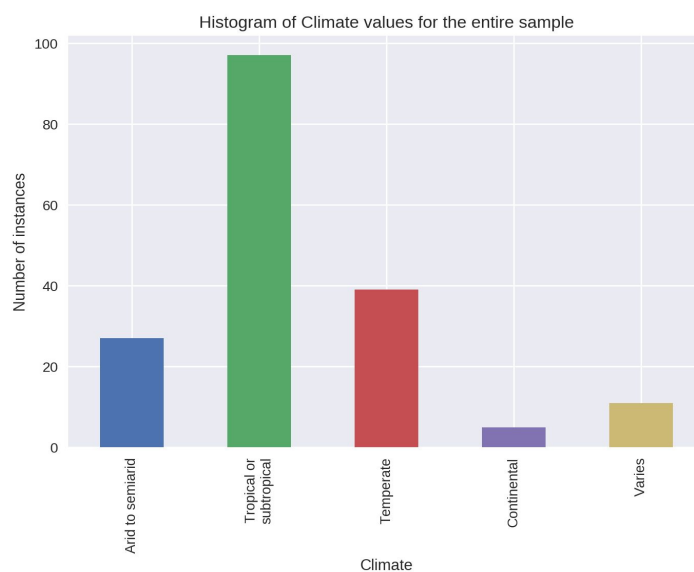
1. Arid to semiarid
2. Tropical or subtropical
3. Temperate
4. Continental
5. Varies greatly within the country

Agriculture categories are:

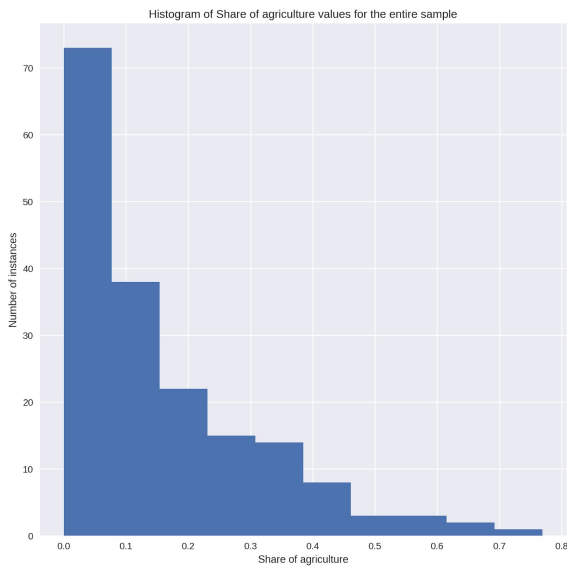
1. Non-agrarian countries (less than 15% of GDP is received from Agriculture)
2. Countries with agriculture income from 15 to 45%
3. Agrarian countries (more than 45%)

Arable categories are:

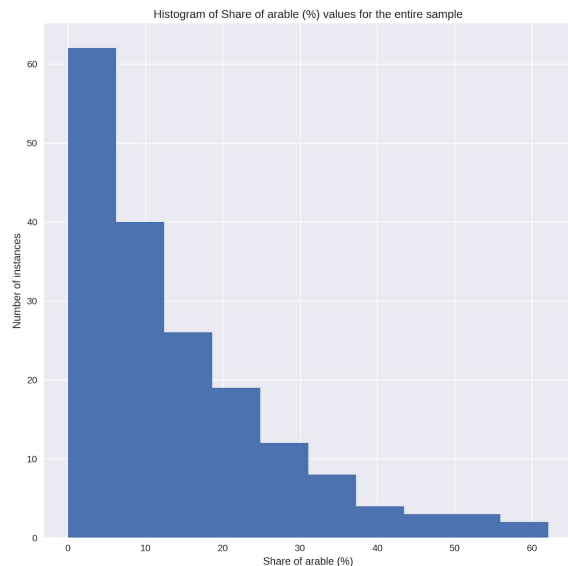
1. Countries, having lack of arable lands (less than 15%)
2. Countries with average percent of arable lands (from 12 to 32%)
3. Countries with arable land prevalence (more than 32% of arable lands)



Histogram of frequencies of categories for Climate (the entire sample)



Histogram for quantitative feature Agriculture (the entire sample)



Histogram for quantitative feature Arable (the entire sample)

After developing nominal features from quantitative features, we obtained for the one origin and two new nominal features the following frequencies of categories of nominal features (each category of feature is assigned a number, see above):

Climate Frequency		
0	climate_1	27
1	climate_2	97
2	climate_3	39
3	climate_4	5
4	climate_5	11

Distribution of the nominal feature “Climate”

Agriculture_cat	Frequency
agriculture_1	109
agriculture_2	59
agriculture_3	11

Distribution of the new nominal feature “Agriculture Categorized” (“Agriculture_cat” in source code)

Arable_cat	Frequency
arable_1	98
arable_2	63
arable_3	18

Distribution of the new nominal feature “Arable Categorized” (“Arable_cat” in source code)

2. Contingency tables building

We have built Contingency tables over selected features using crosstab function of Python’s Pandas library and calculated related Relative contingency tables, Conditional frequency tables and Quetelet relative index tables. (See the source code in Appendix 1. A3.1).

We built two Contingency tables: first one for Climate and Agriculture Categorized and second one for Climate and Arable Categorized.

	agriculture_1	agriculture_2	agriculture_3	Total
climate_1	17	9	1	27
climate_2	49	38	10	97
climate_3	33	6	0	39
climate_4	4	1	0	5
climate_5	6	5	0	11
Total	109	59	11	179

Contingency table for features Climate and Agriculture Categorized

	arable_1	arable_2	arable_3	Total
climate_1	21	6	0	27
climate_2	59	31	7	97
climate_3	11	19	9	39
climate_4	1	3	1	5
climate_5	6	4	1	11
Total	98	63	18	179

Contingency table for features Climate and Arable Categorized

On the basis of these tables we built Relative Contingency (Co-Occurrence) tables.

	agriculture_1	agriculture_2	agriculture_3	Total
climate_1	0.0950	0.0503	0.0056	0.1508
climate_2	0.2737	0.2123	0.0559	0.5419
climate_3	0.1844	0.0335	0.0000	0.2179
climate_4	0.0223	0.0056	0.0000	0.0279
climate_5	0.0335	0.0279	0.0000	0.0615
Total	0.6089	0.3296	0.0615	1.0000

Relative Contingency table for features Climate and Agriculture Categorized

	arable_1	arable_2	arable_3	Total
climate_1	0.1173	0.0335	0.0000	0.1508
climate_2	0.3296	0.1732	0.0391	0.5419
climate_3	0.0615	0.1061	0.0503	0.2179
climate_4	0.0056	0.0168	0.0056	0.0279
climate_5	0.0335	0.0223	0.0056	0.0615
Total	0.5475	0.3520	0.1006	1.0000

Relative Contingency table for features Climate and Arable Categorized



Also we built conditional frequency tables and Quetelet Relative Index tables.

	agriculture_1	agriculture_2	agriculture_3
climate_1	0.156	0.153	0.091
climate_2	0.450	0.644	0.909
climate_3	0.303	0.102	0.000
climate_4	0.037	0.017	0.000
climate_5	0.055	0.085	0.000

Conditional Frequency table for features Climate and Agriculture Categorized

	arable_1	arable_2	arable_3
climate_1	0.214	0.095	0.000
climate_2	0.602	0.492	0.389
climate_3	0.112	0.302	0.500
climate_4	0.010	0.048	0.056
climate_5	0.061	0.063	0.056

Conditional Frequency table for features Climate and Arable Categorized

	agriculture_1	agriculture_2	agriculture_3
climate_1	0.0340	0.0113	-0.3973
climate_2	-0.1704	0.1885	0.6776
climate_3	0.3896	-0.5332	-1.0000
climate_4	0.3138	-0.3932	-1.0000
climate_5	-0.1043	0.3790	-1.0000

Quetelet Relative Index tables for features Climate and Agriculture Categorized

	arable_1	arable_2	arable_3
climate_1	0.4206	-0.3686	-1.0000
climate_2	0.1110	-0.0920	-0.2824
climate_3	-0.4848	0.3842	1.2949
climate_4	-0.6347	0.7048	0.9889
climate_5	-0.0037	0.0332	-0.0960

Quetelet Relative Index tables for features Climate and Arable Categorized

As we can see from Conditional Frequency table for features Climate and Agriculture Categorized, there is an almost conceptual rule **agriculture_3** \Rightarrow **climate_2**, where **agriculture_3** - Agrarian country, **climate_2** - Tropical or subtropical. Indeed, many agricultural countries (such as Afghanistan, Myanmar or Uganda) have a tropical or subtropical climate.

For Conditional Frequency table for features Climate and Arable Categorized we can't make any significant conclusions, because all the values are less than 0.602.

However, the situation changes, when we turn to consideration of Quetelet Relative Index tables for features Climate and Arable Categorized. As we can see there are three values in the table that more than 0.7 (> 70 %): for **arable_3** - **climate_3** (129.49 %), **arable_3** - **climate_4** (98.89 %), **arable_2** - **climate_4** (70.48 %), where **arable_3** - Country with arable land prevalence, **arable_2** - Country with average percent of arable lands, **climate_3** - Temperate, **climate_4** - Continental.

It means that **climate_3**, given **arable_3**, is 129.49% more frequent than on average. This we could not see in Conditional Frequency table: $P(\text{climate}_3 | \text{arable}_3) = 0.5$. An even larger difference we see for pair **climate_4** - **arable_3**: $P(\text{climate}_4 | \text{arable}_3) = 0.056$ (while Quetelet Relative Index for the pair is 98.89 %).



The high values of the Quetelet Relative Index reflect the relationship between the categories considered.

3. Computation of summary Quetelet index

We have calculated χ^2 – *summary Quetelet index* (See the source code in Appendix 1. A3.2). The index value for Climate - Agriculture features pair is 0.0987, the index for Climate - Arable features pair is 0.1366. That is, on average knowledge of **agriculture_i** “adds” 9.87% to frequency of **climate_k**, while on average knowledge of **arable_i** “adds” 13.66% to frequency of **climate_k**.

4. Hypothesis testing

We have contingency tables with $K = 5$ and $L = 3$, so there are $f = (K - 1)(L - 1) = (5 - 1) * (3 - 1) = 8$ degrees of freedom. It follows that there is a 5% chance that the NX_2 value will be greater than 15.507 if the hypothesis of independence is true. Also, there is a 1% chance that the NX_2 value will be greater than 20.090 if the hypothesis of independence is true.

For the first table (Climate - Agriculture) we have $X_2 = 0.0987$, $N = 179$ so that $NX_2 = 17.667$. The hypothesis is to be rejected with 95% confidence. If we take $N' = 157$, we will receive $N'X_2 = 15.496 < 15.507$ and the independence cannot be rejected. For 99% confidence interval $17.667 < 20.090$, so we are not rejecting the hypothesis for $N' = N = 179$.

For the second table (Climate - Arable) we have $X_2 = 0.1366$, $N = 179$ so that $NX_2 = 24.451$. The hypothesis of independence is to be rejected with 95% confidence. If we take $N' = 113$, we will receive $N'X_2 = 15.436 < 15.507$ and the independence cannot be rejected. For 99% confidence interval $24.451 > 20.090$, so we are rejecting the hypothesis for $N = 179$. The hypothesis of independence cannot be rejected if $N' = 147$ ($N'X_2 = 20.081 < 20.090$)



A4 PCA: Hidden Factor & Data Visualization

1. Selection of a subset

We have selected the following features to perform hidden factor data analysis:

- GDP (\$ per capita),
- Death Rate,
- Birth Rate,
- Literacy (%)

We assume that the selected features are connected because high literacy level may influence higher GDP, lower death and birth rates and vice versa. We performed preprocessing using StandardScaler of sklearn.preprocessing Python library. (See the source code in Appendix 1. A4.1).

2. Data scatter and contribution of principal components

We standardized features by removing the mean and scaling to range [0, 1]. After standardization of the selected subset we computed its data scatter. Data Scatter is equal to 194.9646.

We calculated eigenvalues using singular matrix of SVD and determined contributions of all the principal components to the data scatter, naturally and per cent:

Component	Contribution, naturally	Contribution, %
1	0.81602	81.602
2	0.13231	13.231
3	0.03227	3.227
4	0.01940	1.940

3. Data visualization

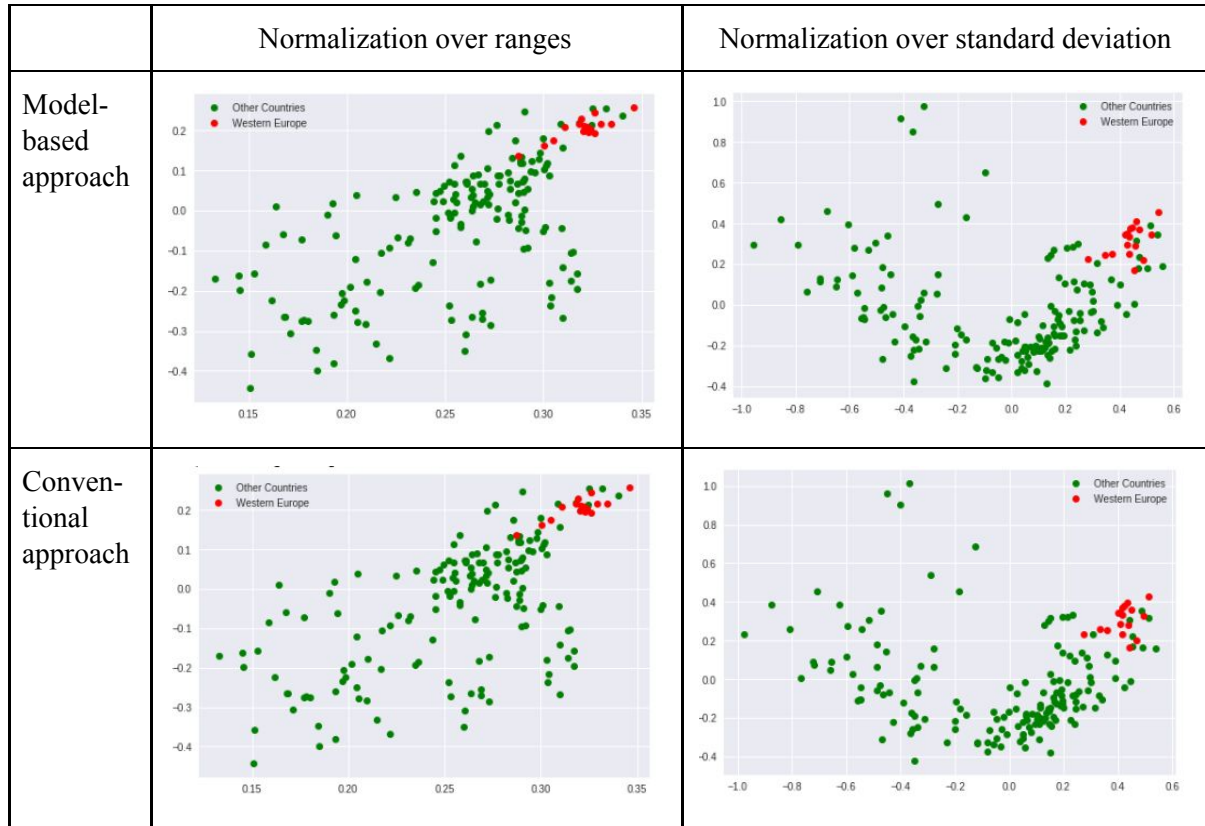
We have selected Western Europe countries as a pre-specified group of objects.

We standardize the data set with two versions of normalization: (a) over ranges and (b) over standard deviations (See the source code in Appendix 1. A4.1).

- Data scatter for standard deviations normalization is 712.00.
- Data scatter for range normalization is 194.96.

Also, we have visualized the data with selected features (GDP, Death Rate, Birth Rate, Literacy (%)) using model-based approach and conventional approach. At these visualizations, we have highlighted in red the points corresponding to the instances of the selected group of Western Europe countries.

There is no any difference in the resulting visualizations for the two approaches considered.



We determined contributions of the two first principal components to each data scatter, naturally and per cent:

Component	Model-based approach & Normalization over ranges		Model-based approach & Normalization over standard deviation		Conventional approach & Normalization over ranges		Conventional approach & Normalization over standard deviation	
	Contribution, naturally	Contribution, %	Contribution, naturally	Contribution, %	Contribution, naturally	Contribution, %	Contribution, naturally	Contribution, %
1	0.8160	81.60	0.6448	64.48	0.8160	81.60	0.6448	64.48
2	0.1323	13.23	0.1942	19.42	0.1323	13.23	0.1942	19.42

As we can see, there is no significant difference between visualizations with normalization over ranges and normalization over standard deviation. In the case of normalization over ranges, the

selected group on the plot is represented as an elongated oval, whereas in the case of normalization over standard deviation, this group has a circular shape, and the points of the group are concentrated near the center of group. That's why we can say that normalization over standard deviation is better in our case.

4. Hidden factor computation and interpretation

We scaled our data to achieve 0 - 100 scale, performed SVD and found first singular triplet. We calculated α value and computed a hidden factor behind the selected features (See the source code in Appendix 1. A4.2).

The following equation was computed: $Z = 0.156 \text{ GDP} + 0.153 \text{ Death Rate} + 0.201 \text{ Birthrate} + 0.490 \text{ Literacy}$.

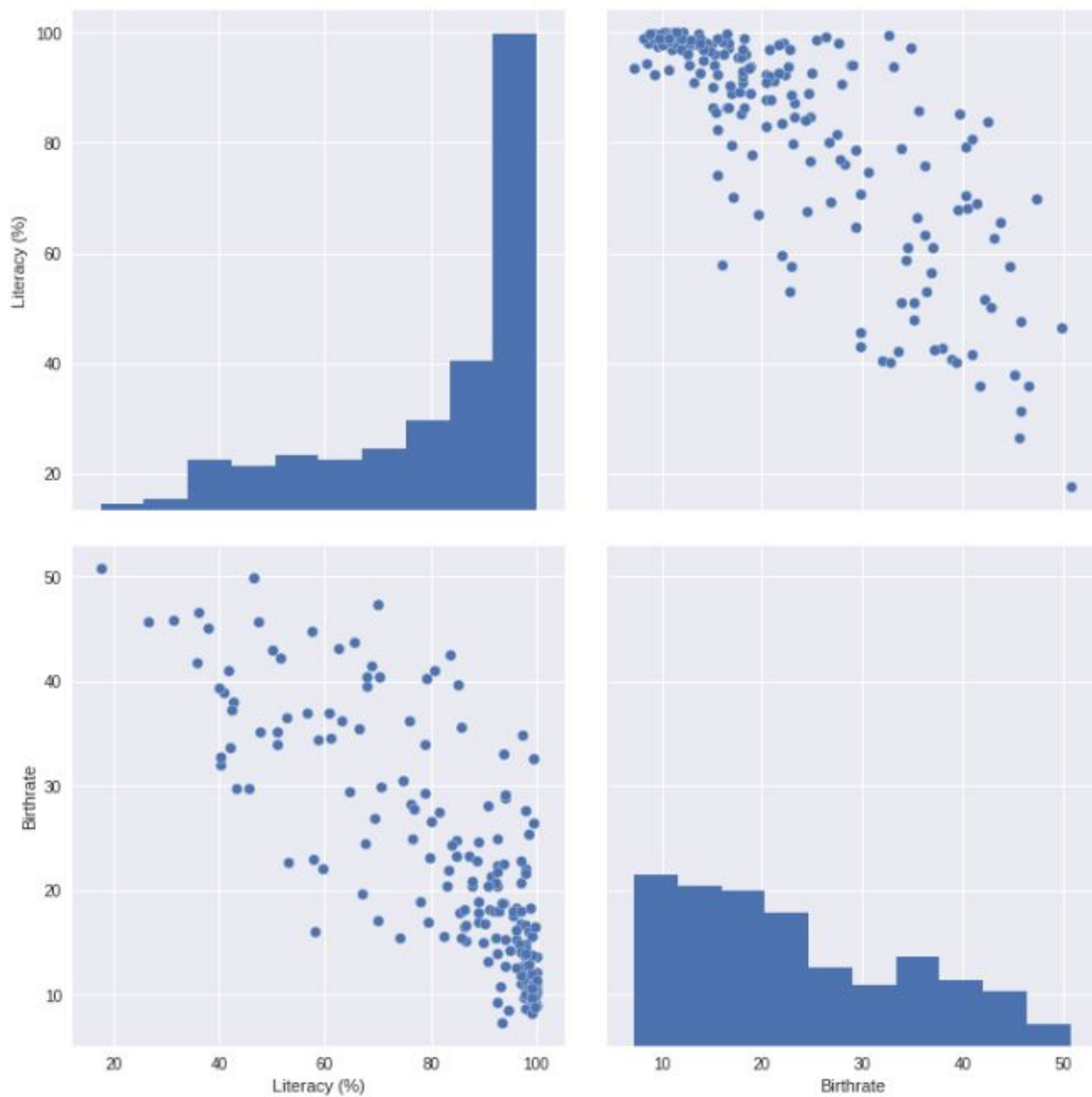
Our subset (Western Europe countries) shows greater hidden factor values, than the other countries (hidden factor mean for our subset is 65, hidden factor mean for all data set is 53 and hidden factor mean for other countries except for Western Europe is 52. We interpret the hidden factor as a factor of good life-quality and society development.



A5: 2D Regression

1. Features selection

We have built scatterplots for numerical features of the dataset using pairplot function of seaborn Python library. We have selected Birth Rate and Literacy as features to perform further analysis.



Scatter plots and histograms for Literacy and Birth Rate



We built a linear regression of birth rate over the literacy feature.

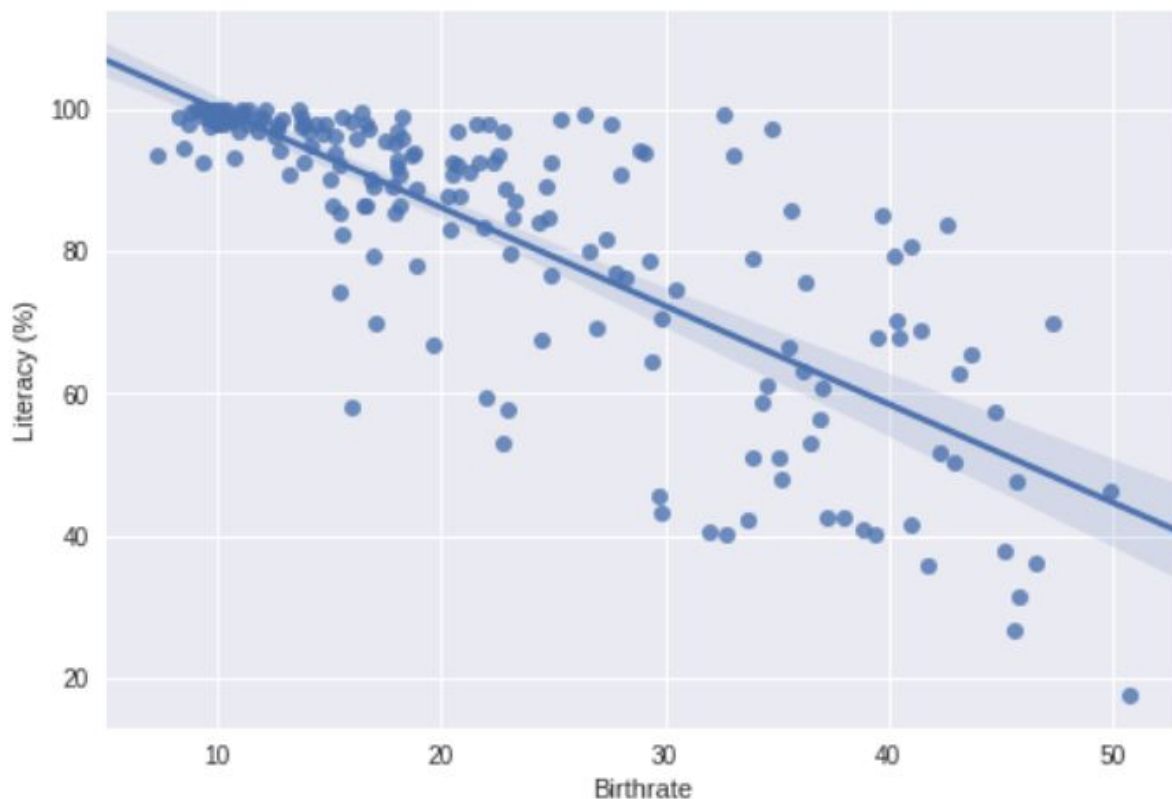
2. Linear regression building

We performed matrix multiplication with numpy dot pseudo-inverse matrix calculation with `linalg.pinv` to find linear regression coefficients (See the source code in Appendix 1. A5.1).

The following values were found:

- $k = -0.45$ (slope)
- $b = 59.84$ (intercept)

The slope is negative, which means, that there is inverse relationship between Birth Rate and Literacy - Birth Rate decreases with Literacy growth.



Scatteplot with linear regression line

3. Correlation and determinacy coefficients

We found determinacy coefficient using $R^2 = 1 - \frac{RSS}{TSS}$ formula (where $RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$, $TSS = \sum_{i=1}^n (y_i - \bar{y})^2$) and correlation coefficient using Pearson coefficient. The following values were found:


- Determinacy coefficient: 0.6215
- Correlation coefficient: -0.7883

Negative correlations imply that as x increases, y decreases. The coefficient is close to -1, which means negative linear relation (See the source code in Appendix 1. A5.2).

Determinacy coefficient shows the proportion of the variance $\sigma^2(y)$ taken into account by the linear regression of y over x. It is equal to 0.62, which is more than 0.5, but it's less than 0.8 (Models with a coefficient of determination above 0.8 can be considered quite good). So our model is acceptable (assuming that the coefficient of determination must be at least not less than 0.5).

4. Prediction of the target values for given two or three predictor' values

We randomly selected three literacy values and calculated prospective birth rates with previously calculated linear regression coefficients (See the source code in Appendix 1. A5.3) :

	Literacy	Birth rate 
Trial 1	45.88	39.24
Trial 2	63.68	31.26
Trial 3	85.82	21.33

The results we obtained shows the same inverse relationship between Birth Rate and Literacy - Birth Rate decreases with Literacy growth.

5. Comparison of the mean relative absolute error of the regression and the determinacy coefficient

We compared the mean relative absolute error of the regression on all points of the set and the determinacy coefficient using $\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2 / y_i$ formula. (See the source code in Appendix 1. A5.4).

Mean Relative Absolute Error value is 0.27. Considering the received Determinacy coefficient, which is equal to 0.62, the MRAE has a reasonable value (The smaller the Mean Relative Absolute Error, the better the approximation of the results).



Appendix 1

A2.1 Source code

```
# numpy is the fundamental package for scientific computing with Python
# pandas is an open source, BSD-licensed library providing high-performance, easy-to-use
data structures and data analysis tools
# sklearn.cluster.Kmeans is used to implement 'kmeans' algorithm
# StandardScaler is used for preprocessing the data (centering and std normalization)
import numpy as np
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler

# pd.read_csv method is used to read data (as a params we include the data path, the
seperation character and other stuff for successful read)

df1 = pd.read_csv("countries_Mirkin.csv", header = 0, decimal = ",", sep = ',')

# Dropping objects with missing values. inplace param stands for saving changes to the
same table.
# We are also excluding the unnecessary feature
df1.dropna(inplace = True)
df1.drop(['Phones (per 1000)'], axis = 1, inplace = True)

# Adding accurate index range (as long as it was shifted after dropping the objects with
missing values)
df1.index = range(0,179)

# Saving features name in array
numeric_features = ['Population', 'Area (sq. mi.)', 'Pop. Density (per sq.
mi.)', 'Coastline (coast/area ratio)', 'Net migration',
'Infant mortality (per 1000 births)', 'GDP ($ per capita)', 'Literacy (%)', 'Arable (%)',
'Crops (%)', 'Other (%)', 'Birthrate', 'Deathrate', 'Agriculture', 'Industry',
'Service']
categorical_features = ['Region', 'Climate']

# choosing features for the task
features_t2 = ['Population', 'GDP ($ per capita)', 'Birthrate', 'Deathrate', 'Pop.
Density (per sq. mi.)']
# 2.1.1
# constructing new table with features for this task
df_t1 = df1[features_t2]
# Standardization
scaler = StandardScaler()
df_t1_scaled = pd.DataFrame(data = scaler.fit_transform(df_t1.values), columns =
features_t2)

# Dividing data into 5 clusters
# Here we do 10 random initializations for our kmeans algorithm and choosing the best
one.
# Params if method Kmeans: init stands for random first position of clusters center,
```

```

n_clusters stands for a number of clusters
# kmean.inertia_ stands for K-means criterion
# Then we save all received results in the dictionary and choose the best one
best_init_5 = 0
lowest_criterion_5 = 300
k_5_dict = {}
for i in range(1,11):
    kmean = KMeans(init = 'random', n_clusters=5, random_state = i).fit(df_t1_scaled)
    print('initialization No.',i," : ", "K-means criterion: ", kmean.inertia_)
    if(lowest_criterion_5 > kmean.inertia_):
        lowest_criterion_5 = kmean.inertia_
        best_init_5 = i
    k_5_dict[i] = kmean

# printing the best results
print("best initialization for k = 5: ", best_init_5, " ,", "best result: ", k_5_dict[best_init_5].inertia_)

# Dividing data into 9 clusters. Same implementation as for k = 5.
best_init_9 = 0
lowest_criterion_9 = 300
k_9_dict = {}
for i in range(1,11):
    kmean = KMeans(init = 'random', n_clusters=9, random_state = i).fit(df_t1_scaled)
    print('initialization No.',i," : ", "K-means criterion: ", kmean.inertia_)
    if(lowest_criterion_9 > kmean.inertia_):
        lowest_criterion_9 = kmean.inertia_
        best_init_9 = i
    k_9_dict[i] = kmean

# printing the best result for dividing into 9 clusters
print("best initialization for k = 10: ", best_init_9, " ,", "K-means criterion: ", k_9_dict[best_init_9].inertia_)

# saving best result to different variables
kmean_5 = k_5_dict[best_init_5]
kmean_9 = k_9_dict[best_init_9]

# printing the clusters centers
kmean_5.cluster_centers_

# rewinding our scaling
kmean5_values = scaler.inverse_transform(kmean_5.cluster_centers_)
kmean5_values

# let's count Grand Mean
mean_array = df_t1.mean()
index_5 = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5', 'Grand Mean']
print(mean_array)

# constructing table with grand mean
kmean5_df = pd.DataFrame(data = np.vstack([kmean5_values, mean_array]), columns = features_t2, index = index_5)

```



```

kmean5_df2 = kmean5_df.copy(deep = True)

# 2.1.2

# Getting table with Ckv/Cv - 1 values (k = 5)
pd.set_option('display.float_format', lambda x: '%.3f' % x)
for i in range(0,5):
    kmean5_df2.iloc[: -1,i] = (kmean5_df.iloc[: -1,i]/kmean5_df.iloc[ -1,i] - 1) * 100

kmean5_df2

kmean9_values = scaler.inverse_transform(kmean_9.cluster_centers_)
index_9 = ['Cluster 1', 'Cluster 2', 'Cluster 3', 'Cluster 4', 'Cluster 5',
           'Cluster 6', 'Cluster 7', 'Cluster 8', 'Cluster 9', 'Grand Mean']

# constructing table with grand mean row
kmean9_df = pd.DataFrame(data = np.vstack([kmean9_values, mean_array]), columns =
features_t2, index = index_9)
kmean9_df2 = kmean9_df.copy(deep = True)

for i in range(0,5):
    kmean9_df2.iloc[: -1,i] = (kmean9_df.iloc[: -1,i]/kmean9_df.iloc[ -1,i] - 1) * 100

kmean9_df2

```

A2.2 Source code

```

#2.2.1

# this method constructs the interval with 100 - alpha confindce. stat stands for data
def stat_intervals(stat, alpha):
    boundaries = np.percentile(stat, [100 * alpha / 2., 100 * (1 - alpha / 2.)])
    return boundaries

# method to implement the bootstrap
def get_bootstrap_samples(data, n_samples):
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples

# Let's compare 3rd and 4th clusters
print(len(np.where(kmean_5.labels_ == 3)[0]))
print(len(np.where(kmean_5.labels_ == 4)[0]))

# implementing the bootstrap
cl4_bootstrap = get_bootstrap_samples(df1.loc[cluster4_indices].Birthrate.values, 5000)
cl5_bootstrap = get_bootstrap_samples(df1.loc[cluster5_indices].Birthrate.values, 5000)

mean_scores_cl4 = list(map(np.mean, cl4_bootstrap))
mean_scores_cl5 = list(map(np.mean, cl5_bootstrap))

# we need to make a comparison of a feature between two clusters, so we will take the

```

```

differences between received 5000 values
# from each cluster and construct and confidence interval for the difference between
values (95 % confidence)
cluster_differences = np.array(mean_scores_cl4) - np.array(mean_scores_cl5)

print("Feature difference between two clusters: ")
print("Pivotal 95% confidence interval for the mean of
BirthRate:",stat_intervals(cluster_differences, 0.05))
print("Non Pivotal 95% confidence interval for the difference: [",
      sorted(cluster_differences)[126], " ",sorted(cluster_differences)[4875],"]")

# mean and std for cluster 4:
values_birthrate_cluster4 = df1.loc[cluster4_indices].Birthrate.values
print(values_birthrate_cluster4.mean())
print(values_birthrate_cluster4.std())

# mean and std for cluster 5:
values_birthrate_cluster5 = df1.loc[cluster5_indices].Birthrate.values
print(values_birthrate_cluster5.mean())
print(values_birthrate_cluster5.std())

#2.2.2
# Pivotal method
np.random.seed(0)
mean_scores_dthr_piv = list(map(np.mean,get_bootstrap_samples(df1.Deathrate.values,
5000)))

print("Pivotal 95% confidence interval for the grand mean of
DeathRate:",stat_intervals(mean_scores_dthr_piv, 0.05))

# Non-pivotal method
np.random.seed(0)
mean_scores_dthr_non_piv = list(map(np.mean,get_bootstrap_samples(df1.Deathrate.values,
5000)))

# counting boundaries
lb = sorted(mean_scores_dthr_non_piv)[126]
rb = sorted(mean_scores_dthr_non_piv)[4875]

# 1% of 500 is 50
# 2.5% is 125; 97.5% is 4875
print("Non Pivotal 95% confidence interval for the grand mean of DeathRate: [",lb,"
",rb,"]")

print(df1.Deathrate.values.mean())
print(df1.Deathrate.values.std())

#2.2.3
# values for the 3rd cluster
df1.Deathrate.values[cluster4_indices]

# Pivotal method

```

```

np.random.seed(0)
mean_scores_dthr_piv_cl =
list(map(np.mean,get_bootstrap_samples(df1.Deathrate.values[cluster4_indices], 5000)))

print("Pivotal 95% confidence interval for the cluster mean:",
stat_intervals(mean_scores_dthr_piv_cl, 0.05))

# Non-pivotal method
np.random.seed(0)
mean_scores_dthr_non_piv_cl =
list(map(np.mean,get_bootstrap_samples(df1.Deathrate.values[cluster4_indices], 5000)))
# counting boundaries
lb_cl = sorted(mean_scores_dthr_non_piv_cl)[126]
rb_cl = sorted(mean_scores_dthr_non_piv_cl)[4875]

# 1% of 500 is 50
# 2.5% is 125; 97.5% is 4875
print(" Non Pivotal 95% confidence interval for the cluster mean: [",lb_cl,"
",rb_cl,"]")

print(df1.loc[cluster4_indices].Deathrate.values.mean())
print(df1.loc[cluster4_indices].Deathrate.values.std())

```

A3 Source code

```

#importing needed libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler

#3.1.1
#reading data
df1 = pd.read_csv("countries_Mirkin.csv",header = 0, decimal = ",", sep = ',')

#dropping missing values and unnecessary features
df1.dropna(inplace = True)
df1.drop(['Phones (per 1000)'], axis = 1, inplace = True)
df1.index = range(0,179)

# checking value counts for each climate
df1.Climate.value_counts()

# uniting two types of climate in one
df1['Climate'] = df1['Climate'].map({1:1,2:2,3:3,4:4,1.5:5,2.5:5})

# functions for transforming numeric features to categorical

#(0 - 0.15 - 0.45 - 1)
def transform_agriculture(x):
    if (x >= 0 and x < 0.15):

```

```

        return 1
    elif( x >= 0.15 and x < 0.45):
        return 2
    elif( x >= 0.45 and x <= 1):
        return 3

#(0 - 12 - 32 - 100)
def transform_arable(x):
    if (x >= 0 and x < 12):
        return 1
    elif( x >= 12 and x < 32):
        return 2
    elif( x >= 32 and x <= 100):
        return 3

# applying transform functions
df1['Service_cat'] = df1['Service'].apply(transform_service)
df1['Agriculture_cat'] = df1['Agriculture'].apply(transform_agriculture)
df1['Arable_cat'] = df1['Arable (%)'].apply(transform_arable)

#3.1.2
# getting crosstab values
ct1 = pd.crosstab(df1['Climate'],df1['Agriculture_cat'])
ct1
ct2 = pd.crosstab(df1['Climate'],df1['Arable_cat'])
ct2

# constructing new column and index names (as long as we need a little bit more
information in our table)
s_features = ['agriculture_1','agriculture_2','agriculture_3','Total']
m_features = ['arable_1','arable_2','arable_3','Total']

index_cl = ['climate_1','climate_2','climate_3','climate_4','climate_5','Total']

# some evaluations to get sum for each row and each column
Total_column1 = [ sum(x) for x in ct1.values]
Total_row1 = [ sum(x) for x in ct1.values.T]
Total_row1.append(179)
Total_column2 = [ sum(x) for x in ct2.values]
Total_row2 = [ sum(x) for x in ct2.values.T]
Total_row2.append(179)

# here we constructed a contingency table for climate and agriculture
con_table1 = pd.DataFrame(data =
np.vstack([np.hstack([ct1.values,np.array(Total_column1).reshape(5,1)]),Total_row1]),
columns = s_features, index = index_cl)

# here we constructed a contingency table for climate and arable
con_table2 = pd.DataFrame(data =
np.vstack([np.hstack([ct2.values,np.array(Total_column2).reshape(5,1)]),Total_row2]),
columns = m_features, index = index_cl)

# now we need to build conditional freq table and quetelet relative index tables

```

```

con_freq_table1 = con_table1.copy(deep = True)
con_freq_table2 = con_table2.copy(deep = True)

#relative tables
con_table1 / 179

con_table2 / 179

#Conditional frequency tables

for i in range(0,3):
    con_freq_table1.iloc[: -1,i] = con_freq_table1.iloc[: -1,i]/con_freq_table1.iloc[5,i]

pd.set_option('display.float_format', lambda x: '%.4f' % x)
con_freq_table1

for i in range(0,3):
    con_freq_table2.iloc[: -1,i] = con_freq_table2.iloc[: -1,i]/con_freq_table2.iloc[5,i]

con_freq_table2

#Quetelet relative index tables

quet_index_table1 = con_table1.copy(deep = True)
quet_index_table2 = con_table2.copy(deep = True)

for i in range(0,3):
    for j in range(0,5):
        quet_index_table1.iloc[j,i] = (quet_index_table1.iloc[j,i] *
        quet_index_table1.iloc[5,3] / (quet_index_table1.iloc[5,i] *
quet_index_table1.iloc[j,3])) - 1
        quet_index_table2.iloc[j,i] = (quet_index_table2.iloc[j,i] *
        quet_index_table2.iloc[5,3] / (quet_index_table2.iloc[5,i] *
quet_index_table2.iloc[j,3])) - 1

#table 1
quet_index_table1

#table 2
quet_index_table2

#3.3

# getting chi2 values

# for table1
chi2_sum_t1 = 0
for i in range(0,3):
    for j in range(0,5):
        chi2_sum_t1 += (quet_index_table1.iloc[j,i] * con_table1.iloc[j,i] /
con_table1.iloc[5,3])
print(chi2_sum_t1)

```

```

# for table2

chi2_sum_t2 = 0
for i in range(0,3):
    for j in range(0,5):
        chi2_sum_t2 += (quet_index_table2.iloc[j,i] * con_table2.iloc[j,i] /
con_table2.iloc[5,3])
print(chi2_sum_t2)

#3.4
# degrees of freedom
(5 - 1)*(3 - 1)
# it's time to go for the table and check values
# 95 % confidence - 15.507
# 99 % confidence - 20.090

#table 1

# 95 % confidence
print(" X^2 value : ",chi2_sum_t1)
print("NX^2 value, where N = 179 : ",179 * chi2_sum_t1) # it's more than 15.507, so we
reject our hypothesis
print("NX^2 value, where N = 177 : ",157 * chi2_sum_t1) # well, 157 is enough to say
that our h0 hypothesis is not rejected
# for 99 % confidence interval 17.65 < 20.090, so we are not rejecting the h0 hypothesis

#table 2

# 95 % confidence
print(" X^2 value : ",chi2_sum_t2)
print("NX^2 value, where N = 179 : ",179 * chi2_sum_t2) # it's more than 15.507, so we
reject our hypothesis
print("NX^2 value, where N = 71 : ",113 * chi2_sum_t2) # well, 113 is enough to say that
our h0 hypothesis is not rejected

# for 99 % confidence interval 24.44 > 20.090, so we are rejecting the h0 hypothesis,
we need to make n less
print(" X^2 value : ",chi2_sum_t2)
print("NX^2 value, where N = 93 : ",146 * chi2_sum_t2) # well, 146 is enough to say that
our h0 hypothesis is not rejected

```

A4 Source code

```

#importing libraries
import numpy as np
import pandas as pd
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from matplotlib import pyplot as plt

```

```

# reading data
df1 = pd.read_csv("countries_Mirkin.csv", header = 0, decimal = ",", sep = ',')

# dropping objects with missing values and unnecessary feature
df1.dropna(inplace = True)
df1.drop(['Phones (per 1000)'], axis = 1, inplace = True)

#4.1

# chosen features
features_subset1 = ['GDP ($ per capita)', 'Deathrate', 'Birthrate', 'Literacy (%)']

# creating new table with chosen features
df4 = df1[features_subset1]

#4.2

# Centering: only subtracting a column mean for each column
scaler = StandardScaler(with_std = False)
df4_scaled = pd.DataFrame(data = scaler.fit_transform(df4.values), columns =
features_subset1)
df4_scaled.head()

# scaling to (0 - 1) range
df5 = (df4_scaled - df4_scaled.min()) / (df4_scaled.max() - df4_scaled.min())
df5.head()

# Counting data scatter
ds = sum(sum(np.asarray(np.multiply(df5, df5))))
print("Data Scatter: %.4f" % ds)

# Making a SVD decomposition, using built-in numpy library method np.linalg.svd
Z, Mu, C = np.linalg.svd(df5)

# computing eigen values from singular values
eigen_values = [x**2 for x in Mu]
print(eigen_values)

# Let's count contribution of each component
Contribution_array = []
for x in eigen_values:
    contrib = x / ds
    Contribution_array.append((contrib, 100 * contrib))
# printing contributions of all principal components
for i in range(1, 5):
    print("Contribution of {} component: {}".format(i, round(Contribution_array[i-1][1], 3)))

#4.3

# getting indexes of pre-specified group (Western Europe countries)
west_eu_idx = np.where((df1['Region'].values == 'WESTERN EUROPE') == True)

```

```

other_country_idx = np.delete(np.arange(0,179,1),west_eu_idx[0])

# standardization with two versions (a) over std (b) over ranges

df4_sc_std = df4_scaled/df4.std(axis=0)
print(df4_sc_std.head())
df4_sc_rg = (df4_scaled - df4_scaled.min())/(df4_scaled.max() - df4_scaled.min())
print(df4_sc_rg.head())

# data scatter for std normalization
ds_std = sum(sum(df4_sc_std.values**2))
print("Data Scatter: ",ds_std)

# data scatter for range normalization
ds_rg = sum(sum(df4_sc_rg.values**2))
print("Data Scatter: ",ds_rg)

# (a) over std standardization

# Model-based approach

# Making a SVD decomposition
Z_a, Mu_a, C_a = np.linalg.svd(df4_sc_std)

# Finding our components
z1_a_m = Z_a[:,0]
z2_a_m = -Z_a[:,1]

#ploting on a 2d plane our objects

plt.plot(z1_a_m[other_country_idx] * np.sqrt(Mu_a[0]), z2_a_m[other_country_idx] *
np.sqrt(Mu_a[1]), 'go', label='Other Countries')
plt.plot(z1_a_m[west_eu_idx] * np.sqrt(Mu_a[0]), z2_a_m[west_eu_idx] *
np.sqrt(Mu_a[1]), 'ro', label='Western Europe')
plt.legend(loc = 0)

# contribution of the principal components
contrib_svd_a_1 = Mu_a[0]**2/ds_std
contrib_svd_a_2 = Mu_a[1]**2/ds_std
print('Contribution of 1st component: ',contrib_svd_a_1)
print('Contribution of 2nd component: ',contrib_svd_a_2)

# Conventional approach

# computing a covariation matrix divided by the numbe of objects
B_a = np.dot(df4_sc_std.T,df4_sc_std)/len(df4_sc_std)

# by using built-in method np.linalg.eig, we find eigen values and eigen vectors.
B_l_a , C_l_a = np.linalg.eig(B_a)
print(B_l_a)
print(C_l_a)

# we are taking the maximal eigen value, and corresponding eigen vector

```



```

l1_a = B_l_a[0]
c1_a = C_l_a[:,0]

# taking the second highest eigen value and corresponding eigen vector for the second
component stuff
l2_a = B_l_a[1]
c2_a = C_l_a[:,1]

# Getting our principal components z
z1_a_c = np.dot(df4_sc_std,c1_a)/np.sqrt(len(df4_sc_std)*l1_a)
z2_a_c = -np.dot(df4_sc_std,c2_a)/np.sqrt(len(df4_sc_std)*l2_a)

# contribution of the first principal component
contrib_std_1 = len(df4_sc_std)*l1_a/ds_std
print('Contribution of 1st component: ',contrib_std_1)

# contribution of the second principal component
contrib_std_2 = len(df4_sc_std)*l2_a/ds_std
print('Contribution of 2nd component: ',contrib_std_2)

# plotting on a 2d plane our objects

plt.plot(z1_a_c[other_country_idx] * np.sqrt(Mu_a[0]), z2_a_c[other_country_idx] *
np.sqrt(Mu_a[1]),
         'go', label='Other Countries')
plt.plot(z1_a_c[west_eu_idx] * np.sqrt(Mu_a[0]), z2_a_c[west_eu_idx] * np.sqrt(Mu_a[1]),
         'ro', label='Western Europe')
plt.legend(loc = 0)

# (b) Standardization over ranges

# Model based approach

# Making a SVD decomposition
Z_b, Mu_b, C_b = np.linalg.svd(df4_sc_rg)

# finding our components
z1_b = -Z_b[:,0]
z2_b = -Z_b[:,1]

# contribution of the principal components
contrib_con_b_1 = Mu_b[0]**2/ds_rg
contrib_con_b_2 = Mu_b[1]**2/ds_rg
print('Contribution of 1st component: ',contrib_con_b_1)
print('Contribution of 2nd component: ',contrib_con_b_2)

# plotting on a 2d plane our objects

plt.plot(z1_b[other_country_idx] * np.sqrt(Mu_b[0]), z2_b[other_country_idx] *
np.sqrt(Mu_b[1]), 'go', label='Other Countries')
plt.plot(z1_b[west_eu_idx] * np.sqrt(Mu_b[0]), z2_b[west_eu_idx] * np.sqrt(Mu_b[1]),
         'ro', label='Western Europe')

```

```

plt.legend(loc = 0)

# Convetional approach

# Computing a covariance matrix divided by number of objects
B_b = np.dot(df4_sc_rg.T,df4_sc_rg)/len(df4_sc_rg)

# getting our eigen vectors and eigen values
B_l_b , C_l_b = np.linalg.eig(B_b)
print(B_l_b)
print(C_l_b)
# we are taking the maximal eigen value, and corresponding eigen vector for the first
component
l1_b = B_l_b[0]
c1_b = C_l_b[:,0]

# I am taking the second highest eigen value and corresponding eigen vector for the
second component (as long as there are some misunderstanding about computing the second
component)
l2_b = B_l_b[1]
c2_b = C_l_b[:,1]

# Getting our principal components z
z1_b_c = np.dot(df4_sc_rg,c1_b)/np.sqrt(len(df4_sc_rg)*l1_b)
z2_b_c = np.dot(df4_sc_rg,c2_b)/np.sqrt(len(df4_sc_rg)*l2_b)

# contribution of the first principal component
contrib_rg_1 = len(df4_sc_rg)*l1_b/ds_rg
print('Contribution of 1st component: ',contrib_rg_1)

# contribution of the second principal component
contrib_rg_2 = len(df4_sc_rg)*l2_b/ds_rg
print('Contribution of 2nd component: ',contrib_rg_2)

# Plotting our objects on a 2d plane

plt.plot(z1_b_c[other_country_idx] * np.sqrt(Mu_b[0]), z2_b_c[other_country_idx] *
np.sqrt(Mu_b[1]),
        'go', label='Other Countries')
plt.plot(z1_b_c[west_eu_idx] * np.sqrt(Mu_b[0]), z2_b_c[west_eu_idx] * np.sqrt(Mu_b[1]),
        'ro', label='Western Europe')
plt.legend(loc = 0)

# 4.4

# Let's compute hidden factor. It should be in 0 - 100 rank scale (as well as features)
# so first we need to scale our data. So, in order to achieve 0 - 100 scale, we will
divide our centered
df5 = (df4 - df4.min())/(df4.max() - df4.min())

# Making a SVD decomposition
Z_t4, Mu_t4, C_t4 = np.linalg.svd(df5)

```

```

Z_t4, Mu_t4, C_t4

# We need to find first singular triplet (u, z, c).
# np.linalg.SVD return 3 matrices (U S V.T), so the 3rd one is already transposed
c_1 = - C_t4[0]
print(c_1)

# then we need to find 'a' in equation z = Xca, in order to rescale z to 0 - 100 scale
(Presentation n5)
alpha = 1/(np.sum(c_1))
print(alpha)
hf_vec = c_1 * alpha
print(hf_vec)

# let's check the final equation:
Z = hf_vec[0] * df5[features_subset1[0]] + hf_vec[1] * df5[features_subset1[1]] +
hf_vec[2] * df5[features_subset1[2]] + \
hf_vec[3] * df5[features_subset1[3]]

# all results are between 0 and 1 ( 0 - 100 if multiplied by 100)
np.array(Z)

np.array(Z)[west_eu_idx]
# hidden factor for EU, seems like its > 0.6 (so it's high, we can call it a factor of a
good life)

# the average is 0.53, so EU in common has more
np.array(Z).mean()

# the average is 0.65, so EU in common has a better factor than other countries
np.array(Z)[west_eu_idx].mean()

# checking a mean for a hidden factor without EU countries, it's even less
np.array(Z)[other_country_idx].mean()

# So the final equation will be like: Z = 0.155*GDP + 0.153 * Deathrate + 0.201 *
Birthrate + 0.489 * Literacy

```

A5 Source code

```

# importing libraries
import numpy as np
import pandas as pd
import seaborn as sns
import random
from sklearn.preprocessing import StandardScaler

# reading data
df1 = pd.read_csv("countries_Mirkin.csv", header = 0, decimal = ",", sep = ',')

```

```

# dropping objects with missing values and unnecessary feature
df1.dropna(inplace = True)
df1.drop(['Phones (per 1000)'], axis = 1, inplace = True)

# 5.1

# let's find a linear-like scatterplot
# we will look on pairplot graphic
cols = ['Population', 'Area (sq. mi.)', 'Pop. Density (per sq. mi.)',
        'Infant mortality (per 1000 births)', 'GDP ($ per capita)', 'Literacy (%)',
        'Birthrate', 'Deathrate']

# building a pairplot
sns_plot = sns.pairplot(df1[cols])

# By analysing all possible scatter plot, we can see some kind of linear stuff
# so, we chose Birthrate and literacy as features
y = df1['Birthrate'].values
x = df1['Literacy (%)'].values

#5.2
# building pairplot for chosen featues
sns_plot = sns.pairplot(df1[['Literacy (%)', 'Birthrate']], size = 10)

#5.3

# finding regression params, using the pseudo inverse matrix ( $x = A*y$ )
A = np.copy(x)
A = np.append(A, np.array([1]*179).reshape(179,1), axis = 1)
params = np.dot(np.linalg.pinv(A), y)

#printing params
print(params)

#Displaying the regression line on the scatter plot
sns.regplot(x="Birthrate", y="Literacy (%)", data=df1)

#5.4
# let's find correlation and determinancy coefficient
y_pred = np.dot(A, params)
u = ((y - y_pred)**2).sum()
v = ((y - y.mean())**2).sum()

R = 1 - u/v
print("Determinancy coefficient: %s" % R)

# for correlation coeff we gonna use Pearson coef
from scipy.stats.stats import pearsonr

cor_coef = pearsonr(x, y)[0]
print("Correlation coefficient: %s" % cor_coef)

```

```

#5.5
# building 3 random points.
rand_points = []
for i in range (0,3):
    rand_x = round(random.random()*100,2)
    y_r = params[0] * rand_x + params[1]
    rand_points.append((rand_x,y_r))
print(rand_points)

#5.6

# so here we compute a mean relative absolute error. in the top it's sum of modules 1/n
# * (y_pred -y)/y.
u_abs = (np.abs(y - y_pred)/y).sum()
rae = u_abs/len(y)
print("Mean Relative Absolute Error: %s" %rae)

```