PES University, Bangalore

(Established under Karnataka Act No. 16 of 2013)

MAY 2020: IN SEMESTER ASSESSMENT (ISA) B.TECH. IV SEMESTER

UE18MA251- LINEAR ALGEBRA

MINI PROJECT REPORT

ON

UNDERSTANDING PRINCIPLE COMPONENT ANALYSIS USING LINEAR ALGEBRA

ON THE EVOLUTION OF THE NOMINAL GDP PER CAPITA IN SPAIN

Submitted by

1.	Name: Aronya Baksy	SRN: PES1201800002
2.	Name: Ruchika Shashidhara	SRN: PES1201800046
3.	Name: Ansh Sarkar	SRN: PES1201800275

Branch & Section : CSE "D"

PROJECT EVALUATION

(For Official Use Only)

Sl.No.	Parameter	Max Marks	Marks Awarded
1	Background & Framing of the problem	4	
2	Approach and Solution	4	
3	References	4	
4	Clarity of the concepts & Creativity	4	
5	Choice of examples and understanding of the topic	4	
6	Presentation of the work	5	
	Total	25	

Name of the Course Instructor : P RAMA DEVI

Signature of the Course Instructor :

Principal Component Analysis

Ansh Sarkar Aronya Baksy Ruchika Shashidhara

May 2020

1 Abstract

Gross Domestic Product (GDP)[1] is the total monetary or market value of all the finished goods and services produced within a country's borders in a specific time period. As a broad measure of overall domestic production, it functions as a comprehensive scorecard of the country's economic health. Nominal GDP is the measurement of the raw data and is more useful comparing national economies on the international market. Some countries have a very divergent GDP per capita between its regions. A country's regions tend to converge over time, while in other cases the disparity between the poorer and the richer regions is kept over the decades. Here we take the example to provide an analysis of the evolution of the nominal GDP per capita across 19 regions in Spain. From the following analysis, we can answer a few questions in terms of Nominal GDP such as - Have the regions converged? Which is the spread between regions? Our aim of this is to answer whether we make a cluster analysis of the regions after applying PCA?

2 Keywords

Economics, GDP, Nominal GDP, Social Sciences, Correlation, Dimensionality Reduction, Eigenvectors, Eigenvalues, Feature extraction, Orthogonality, Patterns, PCA, SVD

3 Introduction

We have used a dataset that has been compiled by the INE (Spanish institute of statistics)[2] which has also been published on Kaggle - Nominal GDP per capita of Spain.[3] This dataset contains information on the Nominal GDP per capita across the 19 Spanish regions (autonomous communities and cities) from 2000 to 2016. To answer the questions that have been highlighted in the Abstract, to figure out whether there has been a convergence, we need to calculate whether the nominal GDP has been steady by looking at the spread between the maximum and the minimum values of the nominal GDP per capita across the Spanish regions. But our true goal is to perform a cluster analysis of the 19 nominal GDP per capita across the 19 Spanish regions. To perform cluster analysis we first need to select the appropriate features to visualize the different cluster classes and their effects on the Nominal GDP. Since our data set consists of 18 columns (here dimensions), we will have to perform PCA for feature extraction before the visualization.

PCA is a powerful statistical technique in Linear Algebra that reduces the dimensions of the data and helps us understand, plot the data with a lesser dimension compared to original data. Principal components are basically vectors that are linearly uncorrelated and have variance within data. From the principal components, the top p is picked which has the most variance. In PCA our goal is to separate the data by drawing a line or plane between data by figuring out the dimension which has the maximum variance.

4 PCA Module

4.1 Linear Algebra Background

Theorem 1. *If the matrix* $\mathbf{A}$ *is symmetric (meaning* $\mathbf{A}^T = \mathbf{A}$*), then* $\mathbf{A}$ *is orthogonally diagonalizable and has only real eigenvalues. In other words, there exist real numbers* $\lambda_1, \lambda_2, ..., \lambda_n$ *(the eigenvalues) and orthogonal, non-zero real vectors* $\vec{v_1}, \vec{v_2}, \vec{v_3}, ..., \vec{v_n}$*(the eigenvectors) such that for each* $i = 1, 2, ..., n$

$$\mathbf{A}v_i = \lambda_i \vec{v_i} \tag{1}$$

4.2 Implementation

We have implemented PCA on the basis of the minimization of the reconstruction error. Reconstruction error is defined as the Frobenius norm of the difference between the original and reconstructed matrix. Specifically, of all possible linear projections from n to m dimensions, taking the first k components of the PCA transformation of $\mathbf{X}$ minimizes the reconstruction error.

4.3 Proof

The input to the problem is m data points of the form $\mathbf{x}_i = (\mathbf{x}_1^i, \mathbf{x}_2^i, ..., \mathbf{x}_n^i), i = (1, 2, .., m)$ The aim of the PCA algorithm is to transform this n-dimensional feature vector to a k-dimensional space with an orthonormal basis $\mathbf{u}_1, \mathbf{u}_2, .., \mathbf{u}_k$

The new set of transformed co-ordinates in the k-dimensional space is represented as $\mathbf{z}_1, \mathbf{z}_2, .., \mathbf{z}_k$ where $\mathbf{z}_i = (\mathbf{x} - \overline{\mathbf{x}}).\mathbf{u}_i$ and $\overline{\mathbf{x}}$ is the mean of $\mathbf{x}$ over all the data points. From the new set of co-ordinate axes, we attempt to reconstruct the data $\mathbf{X}$. The reconstructed data example $\hat{\mathbf{x}}_i$ is given as

$$\hat{\mathbf{x}}^i = \overline{\mathbf{x}} + \sum_{j=1}^{n} z_j^i \, \mathbf{u_j} \tag{2}$$

The objective of PCA is to minimize the error caused by this reconstruction, given by

$$error_m = \sum_{i=1}^{m} (\mathbf{x}^i - \hat{\mathbf{x}}^i) \tag{3}$$

Substituting the above values in the error equation, we get the following result.

$$error_k = \sum_{j=k+1}^{n} \mathbf{u}_j^T \Sigma \, \mathbf{u}_j \tag{4}$$

Where

$$\Sigma = \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}^i - \bar{\mathbf{x}})(\mathbf{x}^i - \bar{\mathbf{x}})^T \tag{5}$$

5 Algorithm to Perform PCA

To implement PCA, we find the Singular Value Decomposition of a matrix A which almost immediately gives us the eigenvalues of and eigenvectors of $\mathbf{A}^T\mathbf{A}$

Part 1: Function to return the orthogonal vector

```
def householder_reflection(a, e):
    assert a.ndim == 1
    assert np.allclose(1, np.sum(e**2))

    u = a - np.sign(a[0]) * np.linalg.norm(a) * e
    v = u / np.linalg.norm(u)
    H = np.eye(len(a)) - 2 * np.outer(v, v)
return H
```

The Householder reflection is an alternate method of orthogonal transformation that transforms a vector x into a unit vector y parallel to x. The Householder reflection matrix $\mathbf{H}$ with a normal vector v takes the form:

$$\mathbf{H} = \mathbf{I} - 2v\,v^T \tag{6}$$

We must construct a matrix **H** such that $\mathbf{H}x = \alpha e_1$ for a constant α and $e_1 = \begin{bmatrix}1 & 0 & 0\end{bmatrix}^T$ Since the matrix **H** is orthogonal, $\|Hx\| = \|x\|$ and $\|\alpha e_1\| = \alpha$, and therefore $\alpha = \pm \|x\|$

The sign is so selected that it has the opposite sign of x_1, hence the vector u is

$$u = \begin{bmatrix} x_1 + sign(x_1)\,\|x_1\| \\ x_2 \\ . \\ . \\ . \\ x_n \end{bmatrix} \tag{7}$$

With the unit vector v defined as $u = \frac{v}{\|v\|}$, the corresponding Householder reflection matrix is

$$\mathbf{H} = \mathbf{I} - 2v\,v^T = \mathbf{I} - 2\frac{uu^T}{u^Tu} \tag{8}$$

Part 2: Function to return the QR Decomposition

```
def qr_decomposition(A):
    n, m = A.shape
    assert n >= m
    Q = np.eye(n)
    R = A.copy()
    for i in range(m - int(n==m)):
        r = R[i:, i]
        if np.allclose(r[1:], 0):
            continue
        # e is the i-th basis vector of the minor matrix.
        e = np.zeros(n-i)
        e[0] = 1
```

```
        H = np.eye(n)
        H[i:, i:] = householder_reflection(r, e)
        Q = Q @ H.T
        R = H @ R
        return Q, R
```

The Householder reflection method of QR decomposition works by finding appropriate H matrices and multiplying them from the left by the original matrix A to construct the upper triangular matrix **R**. As we saw earlier, unlike the Gram-Schmidt procedure, the Householder reflection approach does not explicitly form the matrix **Q**. However, the matrix **Q** can be found by taking the dot product of each successively formed a Householder matrix.

$$\mathbf{Q} = H_1 H_2 ... H_{m2} H_{m-1} \tag{9}$$

Part 3: Function to return Eigendecomposition of matrix **A**

```
def eigen_decomposition(A, max_iter=100):
    A_k = A
    Q_k = np.eye( A.shape[1] )
    for k in range(max_iter):
        Q, R = qr_decomposition(A_k)
        Q_k = Q_k @ Q
        A_k = R @ Q
    eigenvalues = np.diag(A_k)
    eigenvectors = Q_k
    return eigenvalues, eigenvectors
```

This algorithm is iterative: at each step, we calculate $\mathbf{A}_{k+1}$ by taking the QR decomposition of $\mathbf{A}_k$, reversing the order of Q and R, and multiplying the matrices together. Each time we do this, the off-diagonals get smaller and at the end, we return the respective eigenvalues and eigenvectors.

Part 4: Implementation of PCA Algorithm

```
class PCA:
    def __init__(self, n_components=None):
        self.n_components = n_components
    def fit(self, X):
        n, m = X.shape
        # subtract off the mean to center the data.
        self.mu = X.mean(axis=0)
        X = X - self.mu
        # Eigen Decomposition of the covariance matrix
        C = X.T @ X / (n-1)
        self.eigenvalues, self.eigenvectors = eigen_decomposition(C)
        # truncate the number of components
        if self.n_components is not None:
            self.eigenvalues = self.eigenvalues[0:self.n_components]
            self.eigenvectors = self.eigenvectors[:, 0:self.n_components]
        descending_order = np.flip(np.argsort(self.eigenvalues))
        self.eigenvalues = self.eigenvalues[descending_order]
        self.eigenvectors = self.eigenvectors[:, descending_order]
        return self

    def transform(self, X):
        X = X - self.mu
```

```
        return X @ self.eigenvectors

    def proportion_variance_explained(self):
        return self.eigenvalues / np.sum(self.eigenvalues)
```

The following steps are followed before we implement PCA in the idealized form in the fit() function:

1. Ensuring that the data is centered
2. Optionally “whiten” the data so that each feature has unit variance
3. Keeping Eigenvalues in descending order

This is done by subtracting off the mean to center the data and figuring out the Eigen covariance matrix. Since we are performing dimensionality reduction we also truncate the number of components and finally sort the eigenvalues in the descending order to return them.

6 Implementation

We first perform the PCA for feature extraction and reduce the dimensionality for cluster analysis. This part is implemented in the PCA module. Using the PCA module we transform the dataset to generate the first 3 components of PCA. Using this transformed data we perform the cluster analysis with the K-means clustering Algorithm.

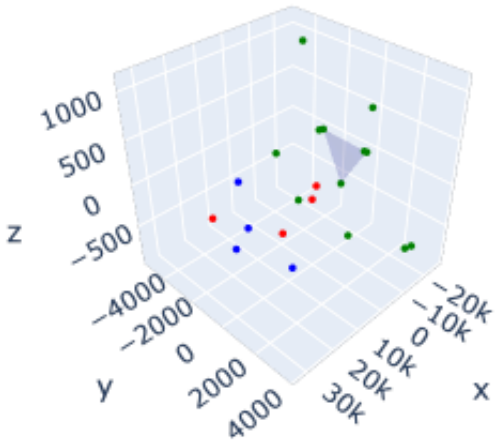
```
pca = PCA(n_components=3)
pca.fit(x)
x_prime = pca.transform(x)
x_prime = pd.DataFrame(x_prime)
```

Front-end implementation of PCA

K Means algorithm is an iterative algorithm that tries to partition the dataset into K pre-defined distinct non-overlapping subgroups (clusters) where each data point belongs to only one group. Here the value of k is 3. It tries to make the intra-cluster data points as similar as possible while also keeping the clusters as different (far) as possible. It assigns data points to a cluster such that the sum of the squared distance between the data points and the cluster's centroid (arithmetic mean of all the data points that belong to that cluster) is at the minimum. The less variation we have within clusters, the more homogeneous (similar) the data points are within the same cluster.

```
from sklearn.cluster import KMeans
kmeans = KMeans(n_clusters=3)
clusters = kmeans.fit(x_prime)
```

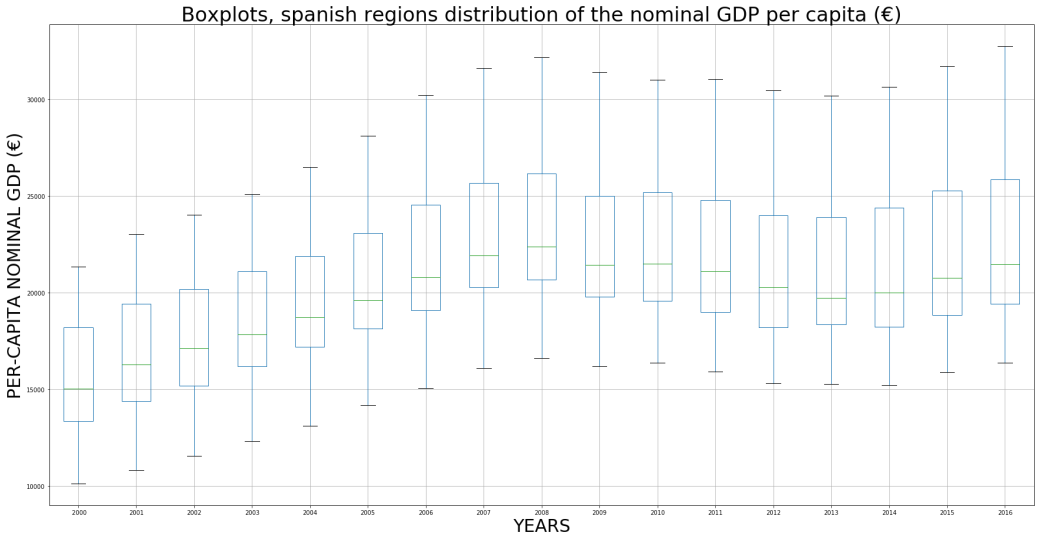
K-Means implementation



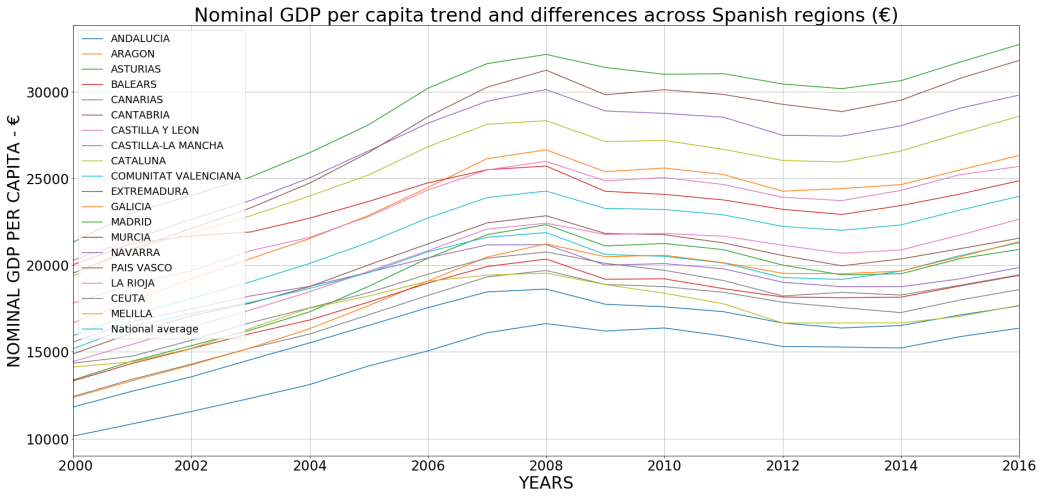
7 Visualizations

Our dataset contains the compilation of the nominal GDP per capita for each region/autonomous entity of Spain and the national average (in € / from 2000 to 2016). We first analyze the territorialized evolution of the nominal GDP per capita in Spain, identify the economic circles, the spread between regions, and examine if there has been a convergence (in GDP per capita) between regions from 2000 to 2016.

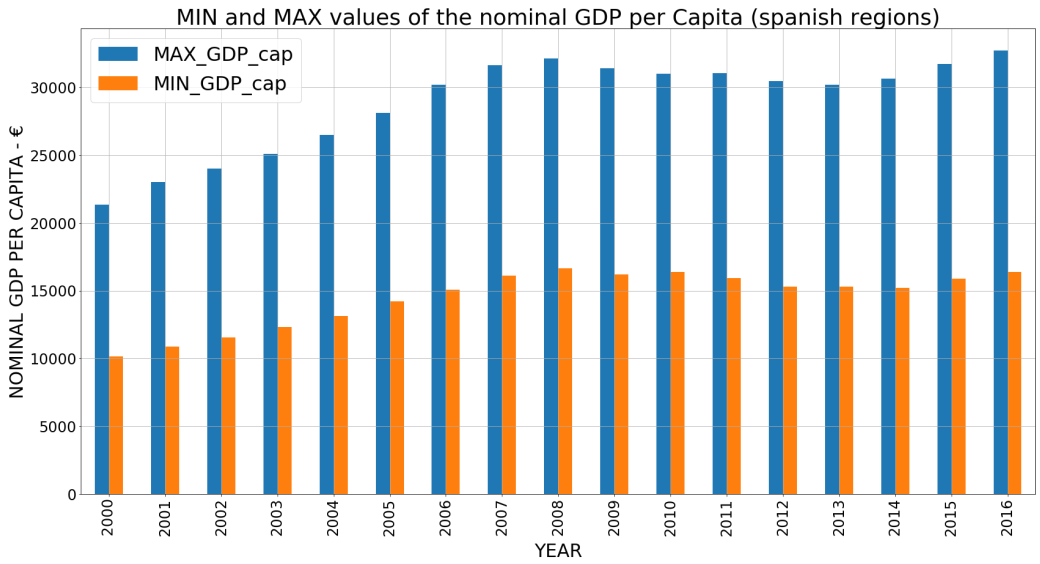
So we create a BOXPLOT to visualize the distribution of the nominal GDP per capita between the regions (from 2000 to 2016)



And we create a LINE PLOT to visualize the trend and spread of the regions :



Both plots emanate a relatively high spread of the nominal GDP per capita between regions. To generate a visualization/examination of the divergence between regions we can concatenate the series the percentage difference of both the maximum and the minimum values by creating a new data frame. We can infer from the plots that the series is regular. The plot that describes the MIN and MAX nominal GDP per Capita across Spanish regions :



8 Conclusions

During the last 16 years, the spread between the maximum and the minimum values of the nominal GDP per capita across Spanish regions has remained very steady. The differences in 2016 are almost the same as those of the year 2000. We can conclude that in spite of the efforts by the Spanish government to redistribute the country's wealth across its regions; there has not been a regional convergence in terms of nominal GDP per capita. From the Cluster Analysis and the Plots, we can distinguish 3 economic circles (3 classes from the clusters)

1. From 2000 to 2008: a period of continuous growth of the nominal GDP per capita in all regions.

2. From 2008 to 2013: the global recession started a strong downturn in all regions.

3. From 2013 to 2016: the start of a partial recovery of the nominal GDP per capita in all regions

9 Final Conclusion

- $$z = \frac{X - \mu}{\sigma} \tag{10}$$

  (Standardization of values)

- $$cov(Z) = \frac{1}{n}\sum_{i=1} m(X_i - \bar{X})(X_i - \bar{X})^T \tag{11}$$

  (computing the co-variance matrix)

- $$v(A - \lambda I) = 0 \tag{12}$$

  (Compute eigenvectors and the corresponding eigenvalues)

- Sort the eigenvectors by decreasing eigenvalues and choose k eigenvectors with the largest eigenvalues, these becoming the principal components.

- Final Data= Feature-Vector*Transpose(Scaled(Data)) is used to reorient the data according to the new principal component.

10 Scope of Future Work

Principal component analysis (PCA) is a statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables. Implementing neural networks with a layer of PCA would help us to optimize the model. There will be fewer chances of overfitting and the visualizations would be improved. The main advantage of PCA is to optimize the data and that would be helpful in any algorithm as it eradicates correlated features.

11 Summary

- We have taken the following example to provide an analysis of the evolution of the Nominal GDP per capita across 19 regions in Spain from 2000-2016. We have found out that there has not been a regional convergence in terms of nominal GDP per capita and we can distinguish 3 economic circles based on the steadiness of GDP per capita for all regions from the Cluster Analysis after applying Principal Component Analysis
- We have implemented the PCA module from scratch using the basis of the Minimization of the Reconstruction Error by using Householder Reflections and QR Decomposition to find the Eigenvalues and Eigenvectors.
- We have drawn various Boxplots, Line Plots and Bar Graphs to analyze the spread and distribution of the nominal GDP per capita between the regions.
- We have finally used the K-means Clustering Algorithm to perform the Cluster Analysis using the first 3 values after dimensionality reduction using Principal Component Analysis.

References

[1] Gross Domestic Product https://en.wikipedia.org/wiki/Gross_domestic_product

[2] Instituto Nacional de Estadistica

[3] Dataset - Nominal GDP per capita of Spain (by regions) - series from the years 2000 to 2016 (Kaggle)

[4] Principal Component Analysis with Linear Algebra by Jeff Jauregui, Aug 31, 2012

[5] Frobenius norm https://en.wikipedia.org/wiki/Matrix_norm#Frobenius_norm

[6] Handbook Series Linear Algebra, Singular Value Decomposition and Least Squares Solution by G.H. Golub and C. Reinsch, Numer. Math. 14, 403–420 (1970)

[7] Principal Component analysis by I.T.Jolliffe, Second Edition, Springer Verlag, New York, USA, Apr 2002 (Chapter 3 - Properties of Sample Principal Components, pg 29 - 59)

[8] Python Data Science Handbook by Jake VanderPlas, First Edition, O'Reilly, Sebastopol, USA, Dec 2016 (Chapter - Machine Learning - In Depth: Principal Component Analysis, pg 433 - 445)

[9] LIII. On lines and planes of closest fit to systems of points in space by Karl Pearson F.R.S