MATH 399 Report

Aidyn Kadyr

July 2021

1 Introduction

During the summer term I worked as a data scientist at the National Analytical Centre - a subsidiary of the Agency for Strategic Planning and Reforms. The company uses government generated data to make predictions and recommendations to different bodies in the government. During the period I have worked with two case studies. First, analyse, find outliers and forecast prices for socially significant food products. Second, predicting key macroeconomic indicators using the quarterly macroeconomic model. Both case studies required to know math and apply it using Python. Math included topics such as Principal Component Analysis (PCA) and Multiple Linear Regression. The challenging part was cleaning, preprocessing and fitting the data under Linear Regression assumptions.

2 Principal Component Analysis

PCA is a feature extraction technique that uses Principal Components to retain as much information from the original data as possible, i.e. maximum amount of variance, reducing the number of variables. Principal Components are new variables which are linear combinations of all the initial features. They are sorted in an order of importance and least important ones can be dropped to reduce the dimensionality of the data. Hence, the final data retains information of all the features in contrast to Feature Elimination when we just drop initial features. The drawback of the PCA is that independent variables become non interpretable while using regression analysis.

Derivation:

Suppose we have a data matrix $X_{m,n}$ where each row is a specific measurement from an experiment and each column represents a different feature.
1. Calculate the mean of each column:

$$\bar{\boldsymbol{x}} = \frac{1}{n}\sum_{i=1}^{n}\mathbf{X}_{i,j}$$

2. Let

$$\bar{\boldsymbol{X}} = \begin{bmatrix} 1 \\ 1 \\ ... \\ 1 \end{bmatrix} \begin{bmatrix} \bar{\boldsymbol{x}} & & ... & \end{bmatrix}$$

3. Then, let

$$\boldsymbol{B} = \boldsymbol{X} - \bar{\boldsymbol{X}}$$

such that each column of $\boldsymbol{B}$ has a mean of zero, i.e. mean centered matrix.
4. Compute the covariance matrix

$$\boldsymbol{B}^T\boldsymbol{B}$$

5. Find the eigenvectors and eigenvalues of the covariance matrix 6. Principal Components are the columns of a new matrix T:

$$\boldsymbol{T} = \boldsymbol{B}\boldsymbol{V}$$

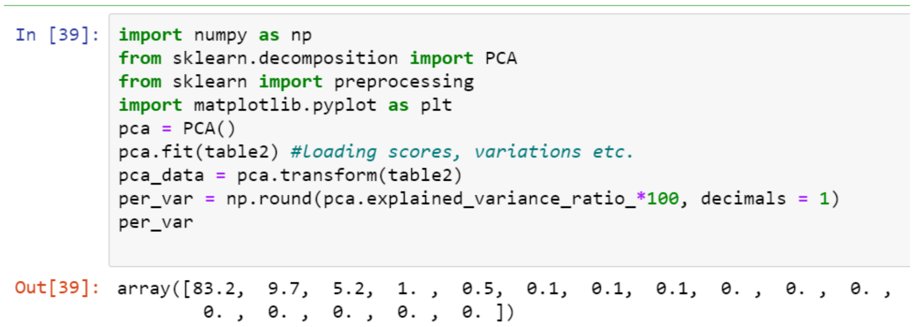
where V is the matrix of eigenvectors.
7. In order to reduce the dimensionality of the data, sort the eigenvectors $\lambda_1, \lambda_2, ..., \lambda_3$ in a decreasing order and corresponding eigenvectors in matrix $\boldsymbol{V}$ accordingly. Then drop an arbtirarily selected number of eigenvectors and eigenvalues to reduce the number of variables.
Note: Since eigenvectors are linearly independent and orthogonal, final variables are independent of one another.

Another more widely used method of dropping variables to reduce dimensionality is to calculate the proportion of variance that is explained by each variable. Since

$$\lambda = \sigma^2$$

Proportion of variance explained by the first k variables

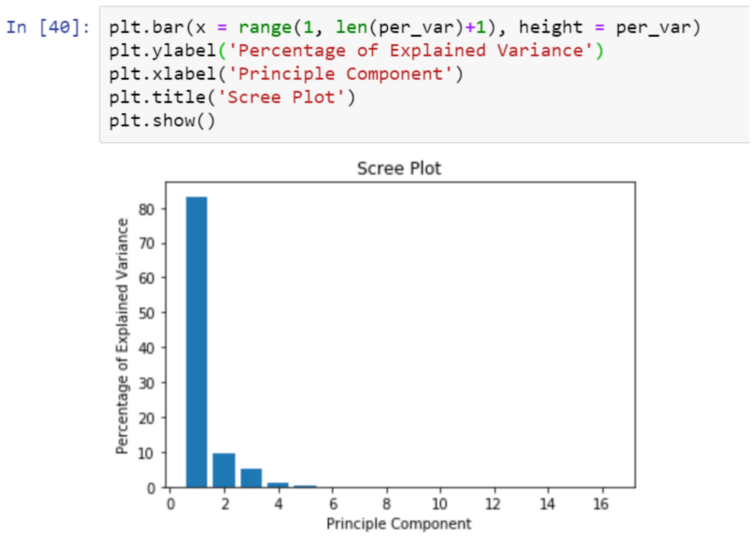
$$= \frac{\sum_{i=1}^{k} \lambda_i}{\sum_{i=1}^{n} \lambda_i}$$



```
In [39]: import numpy as np
         from sklearn.decomposition import PCA
         from sklearn import preprocessing
         import matplotlib.pyplot as plt
         pca = PCA()
         pca.fit(table2) #Loading scores, variations etc.
         pca_data = pca.transform(table2)
         per_var = np.round(pca.explained_variance_ratio_*100, decimals = 1)
         per_var

Out[39]: array([83.2,  9.7,  5.2,  1. ,  0.5,  0.1,  0.1,  0.1,  0. ,  0. ,  0. ,
                 0. ,  0. ,  0. ,  0. ,  0. ])
```

Figure 1: Variance ratio



```
In [40]: plt.bar(x = range(1, len(per_var)+1), height = per_var)
         plt.ylabel('Percentage of Explained Variance')
         plt.xlabel('Principle Component')
         plt.title('Scree Plot')
         plt.show()
```

Figure 2: Scree Plot

The Scree plot above helps to visualize how much information is captured by a specific number of Principal Components. Here, 83.2% of variance is captured by the First Principal Component and 9.7% by the Second Principal Component and so on. Thus, by taking the first, second and third Principal Components it possible to capture 98.1% of information out of the original data.

3 Multiple Linear Regression and Time-Series Data

Multiple Linear Regression was used in the scope of a Quarterly Macroeconomic Model. The Model was originally designed and published by the Economic Re-

search Institute of the Ministry of National Economy of the Republic of Kazakhstan in 2020. The model is divided into 6 blocks by key economic sectors, where each block is further divided to predict different macroeconomic indicators (GDP, inflation, amount of depoists, loans, etc.). The model itself was written in Stata statistical software. The goal of my team, as members of the National Analytical Centre was to rebuild and improve it in Python environment.
Challenging part was to read through the theory behind macroeconomics and statistics and understand its application and purpose to be able to make adjustments.

3.1 Stationarity - Augmented Dickey-Fuller Test

Since most of the variables are time dependent variables, i.e. time-series, they need to be stationary. Stationarity implies that summary statistics such as mean and variance should be constant over time. Thus time-series would not show seasonality or some kind of trend. Consider Kazakhstan GDP over 20 years

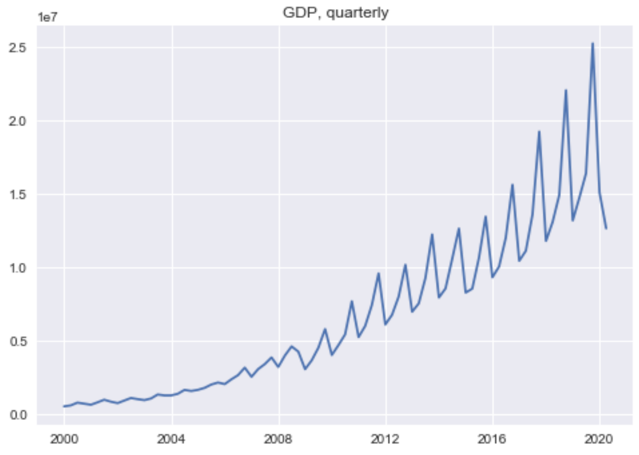


Figure 3: GDP, quarterly

An upward trend and increasing variance can be clearly seen affecting the model. Therefore, the annual GDP growth was applied instead:

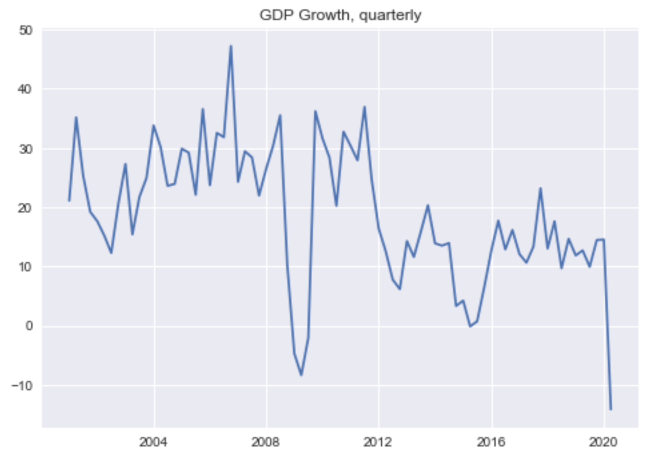
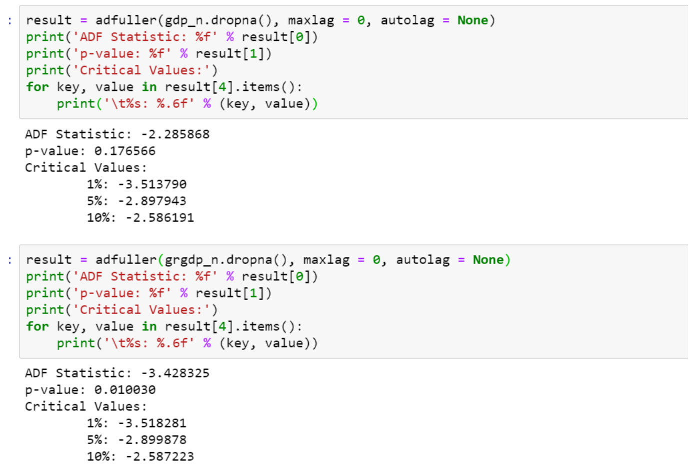


Figure 4: GDP growth, quarterly

Although the resulting graph is not perfectly stationary, it fits the model better than the original data. Despite some deeps in 2009 and 2020, remaining data shows relatively constant mean and variance.

Dickey-Fuller test

To quantify stationarity of the time-series, Augmented Dickey-Fuller test is applied to both GDP and GDP growth to compare.



```
result = adfuller(gdp_n.dropna(), maxlag = 0, autolag = None)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.6f' % (key, value))
```

```
ADF Statistic: -2.285868
p-value: 0.176566
Critical Values:
        1%: -3.513790
        5%: -2.897943
        10%: -2.586191
```

```
result = adfuller(grgdp_n.dropna(), maxlag = 0, autolag = None)
print('ADF Statistic: %f' % result[0])
print('p-value: %f' % result[1])
print('Critical Values:')
for key, value in result[4].items():
    print('\t%s: %.6f' % (key, value))
```

```
ADF Statistic: -3.428325
p-value: 0.010030
Critical Values:
        1%: -3.518281
        5%: -2.899878
        10%: -2.587223
```

Figure 5: Dickey-Fuller Test for GDP and GDP growth

The **statsmodels** Python library has the **adfuller** function which shows the ADF Statistics, p-value and critical values for a time-series variable. The ADF p-value for GDP is 0.1765 while for annual GDP growth is 0.01003 which is significant at the 95% confidence. Hence, the annual GDP growth was used in models and same trial and error procedure was applied to other variables.

3.2 Autocorrelation - Durbin Watson Test

Autocorrelation in time-series data refers to the correlation between a variable and its lagged version with some order i. In order to fit a model we assume that residuals of a variable are independent.
Consider an OLS model where the target variable has autocorrelated errors.

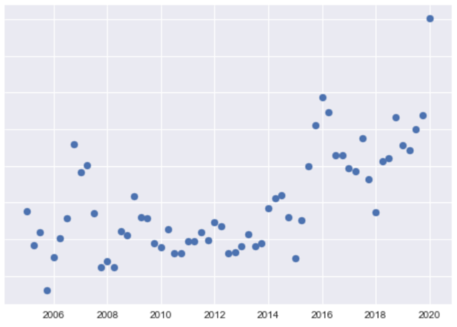


Figure 6: Autocorrelation

Now consider the case when the target variable growth was taken instead of the actual historical data.

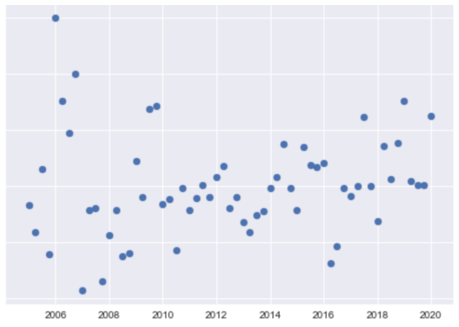


Figure 7: No Autocorrelation

While the first plot depicts a slight trend and correlation between the residuals, the second plot seems to have more independent residuals in a fixed range. When comparing the two OLS models there has been a significant improvement and the one with less autocorrelation showed better results.

Durbin-Watson Test
Numeric representation of the autocorrelation concept comes with the Durbin-Watson test. The function also comes with the **statsmodels** Python library and outputs a test statistic between 0 and 4, where 0 represents positive and 4 represents negative autocorrelation. Values closer to 2 are considered to show low autocorrelation.
For autocorrelated target variable the Durbin-Watson test statistic is 0.334 while in case of taking the growth of the variable it is 1.537.

Note: Durbin-Watson test is used to check for the first order autocorrelation, while there is also Breusch-Godfrey test which is designed for an arbitrary order autocorrelation.

4 Linear Regression Assumptions

4.1

Linear relationship between dependent and independent variables. Verified by the correlation matrix and linear plots.

4.2

Homoscedasticity - constant variance of independent variables residuals. Verified by the Breusch-Pagan test.

4.3

Low or no multicollinearity. Independent variables are independent of one another. Verified by the correlation matrix and linear plots.

4.4

Residulas are normally distributed. Verified by Quantile-Quantile plots.

5 Conclusion

During the period I have covered two blocks of the macroeconomic model: Monetary and GDP. Each block contains 9 target variables where for each target and independent variable my job was to make relevant adjustments to fit stationarity, no autocorrelation, homoscdedasticity, no multicollinearity and other assumptions for linear regression and time series analysis.