STA137 Final Project

The yearly Central African Republic Exports time series from 1960 to 2017

Fall 2024

Professor: Xiucai Ding

Group Members:

Bao Hu 919399420,

Shiman Zhang 920630796,

Ning Mu 919588943,

Huiwen Yang 919757347

I. Introduction

The Central African Republic (CAR) relies heavily on the export of natural resources such as timber, diamonds, cotton, and coffee. These exports play an important role in supporting foreign exchange reserves, economic stability, employment, and development, making them crucial for the country's overall prosperity and resilience. Given the country's vulnerability to global market fluctuations, political instability, and environmental challenges, understanding export trends is critical for sustainable economic development. This study aims to analyze CAR's export data from 1960 to 2017 using time series analysis with the ARIMA model. Since time series analysis will help uncover trends, patterns, and changes in the CAR's export data by analyzing how export values evolve over time, it provides insights into the factors influencing the export sector and helps forecast future trends. These findings can assist policymakers and stakeholders in making informed decisions to strengthen the country's economic resilience and stability.

II. Exploratory Data Analysis

The dataset represents yearly export values for the Central African Republic from 1960 to 2017, spanning 58 observations. The data is clean, with no missing or duplicate entries, ensuring its reliability for analysis. Descriptive statistics reveal that the average export value is approximately $20.66 million, with a median of $21.74 million, indicating balanced central tendencies. Export values range from a minimum of $10.68 million to a maximum of $34.31 million, with a standard deviation of $5.94 million, suggesting moderate variability in export performance over time. We decompose the original data into trend (Tt), seasonality (St), and residual (Rt), and yielded the following:

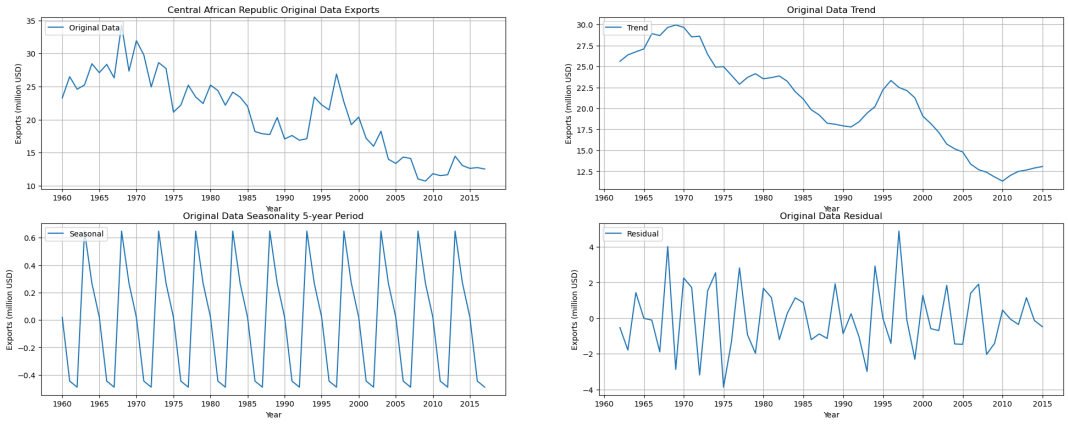

Figure 1

The first graph represents the time series of the Central African Republic's exports from 1960 to 2017. The second graph shows a clear downward trend after the early 1970s, with a peak in export values

around that time. The residual seems to be stationary, so we could have fit the residual to a model to do further analysis, but due to the project's rubric, we will use other methods such as transformation or differencing to obtain stationarity.

The series shows a declining mean and changing variance over time, violating stationarity assumptions. To confirm the non-stationary nature of the series, the Augmented Dickey-Fuller (ADF) test was performed. Given the p-value from the ADF test is greater than a significance level of 0.05, we fail to reject the null hypothesis, confirming that the export time series is non-stationary. Thus, further transformations or differencing would be necessary to ensure stationarity before applying time series models.

III. Transformation and ACF/PACF

In Figure 1, we can only see the trend, seasonal components, and residuals, which don't show normal distribution directly. Generally speaking, the original data of economic time series are mostly not normally distributed and normally need to be differenced, transformed, or fitted to the model before reviewing the normality of the residuals. In addition, in the original data, series that are not transformed exhibit large-scale changes and trend downward, with large differences in amplitude between peaks and troughs in the seasonal cycle. These features indicate that the variance varies over time and that there is variance instability, e.g., the amplitude of fluctuations in the early data is not the same as in the later period. However, it is only possible to assume that the original data has been variance-stabilized with the use of a specific variance-stabilizing transform.

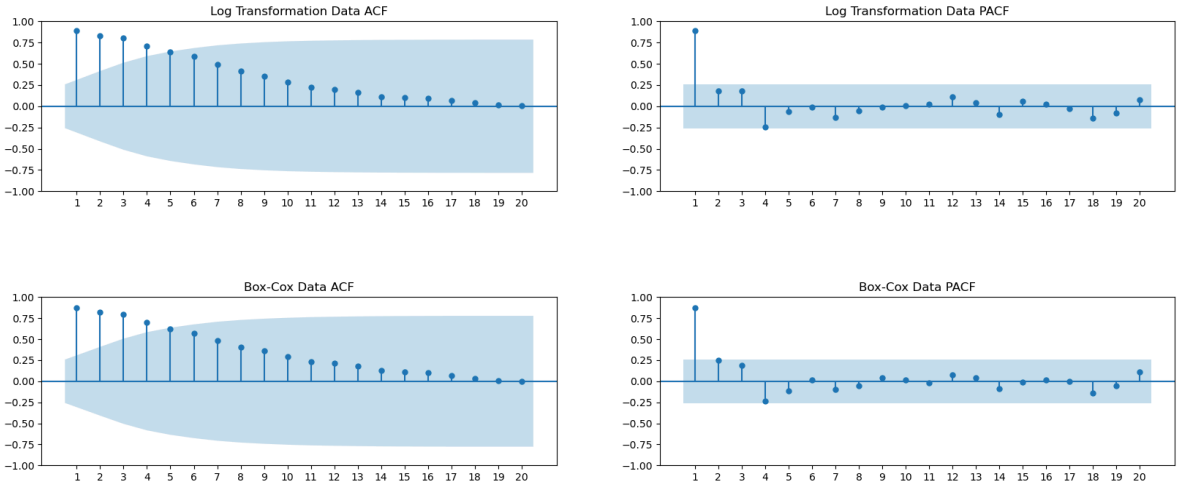

Figure 2

According to these backgrounds, we did the Log and Box-Cox Transformations to the original data to see if we could obtain a stationary series. Log-transformed ACF plots show high autocorrelation coefficients at lags 1 to 4, which decay slowly.PACF plots show significant biased autocorrelation at lag 1, but decay rapidly after lag 1. In the Box-Cox transformed ACF plots, the autocorrelation is significant at lower order lags (1 to 3), while the PACF plot class has significant biased autocorrelation at lag 1, which decays rapidly thereafter. These suggest that the Box-Cox transformed data outperforms the Log transform in terms of smoothness, but neither is completely smooth. In addition to that, we performed the ADF Test on each transformed data and failed to reject both null hypotheses. We found that in log transformation, the ADF check shows that the p-value is 0.837, and in Box-cox, the P-value is 0.823. Both of them are greater than 0.05, so we failed to reject H0 indicating none of them return stationary data. Therefore, we took the first difference in the original data to obtain stationarity.

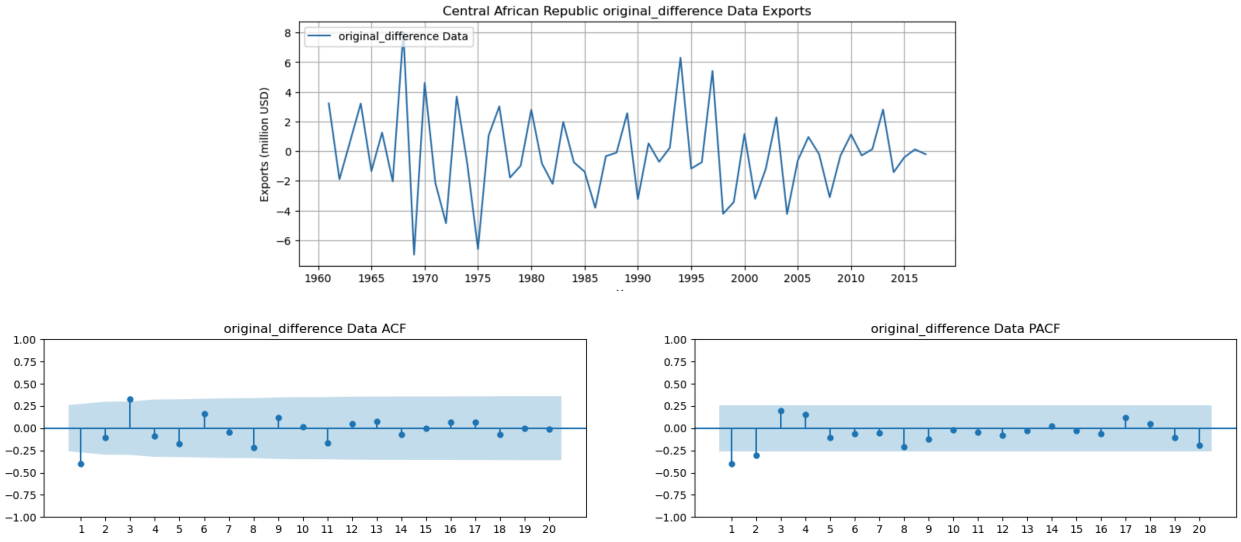

Figure 3

By performing first-order differencing on the raw data, we can clearly see that the long-term trend is successfully eliminated, and from the differenced time-series plot, we observe that the data fluctuates up and down around zero, which means it has stability. However, the variance still seems to be not exactly equal across time and there is some volatility. Furthermore, the ADF test shows p-value = 0.000263 ≤ 0.05, thus rejecting H0, indicating that the time series is stationary. Combining the ACF and PACF plots, we can also find that the ACF plot shows significant autocorrelation at lag order 1, but rapidly decays close to zero from lag order 2. This is typical of MA(1). The PACF plot shows significant biased

autocorrelation at lag order 2, which decays rapidly from lag order 3 which may have a small amount of lower order autoregressive components such as AR(2). These analyses provide a good basis for subsequent parameter estimation and prediction using the ARIMA method in Python and recommend MA(1) and AR(2) on the differenced data as candidate models.

IV. Finding Candidate Models

Hence, we obtained the MA(1) and AR(2) models with their coefficients (Table 1):

MA(1) Parameter/Equation	Value	P-value
$y_t = -0.417225 \cdot w_{t-1} + w_t$	-	-
θ (MA Coefficient)	-0.417225	5.81e-03
σ^2 (Variance)	6.889319	4.47e-10
AIC	275.96	-
BIC	280.04	-

AR(2) Parameter/Equation	Value	P-value
$y_t = -0.505033 \cdot y_{t-1} - 0.289666 \cdot y_{t-2} + w_t$	-	-
ϕ_1 (AR Coefficient 1)	-0.505033	6.42e-04
ϕ_2 (AR Coefficient 2)	-0.289666	5.28e-02
σ^2 (Variance)	6.470776	7.18e-09
AIC	274.54	-
BIC	280.67	-

Table 1

where $y_t = x_t - x_{t-1}$ is the first order difference of the original data.

We observed that the AR coefficient 2 of AR(2) is not significant since its p-value is slightly larger than 0.05, so we reduced the model to AR(1), the correlated values are in Table 2 below.

AR(1) Parameter/Equation	Value	P-value
$y_t = -0.391382 \cdot y_{t-1} + w_t$	-	-
ϕ (AR Coefficient)	-0.391382	0.00338
σ^2 (Variance)	7.094065	0.0
AIC	277.603456	-
BIC	281.689559	-

Table 2

Although the p-value for each coefficient for AR(1) is significant, the AIC (277.60) and BIC (281.69) of it became higher than AR(2)'s AIC (274.54) and BIC (280.67). So we will keep using AR(2) for further analysis and forecasting.

V. Diagnose

Through residual diagnostics, we find the model that best fits the export dynamics of the Central African Republic between AR(2) and MA(1).

The MA(1) model shows some issues in the residual diagnostics. The Standardized Residuals show residuals that are centered around zero and exhibit similar variability, with no strong patterns indicating obvious model misfit. The Q-Q plots align well with the 45-degree line, suggesting that the residuals are approximately normally distributed in both cases. The ACF plot shows a few lags with significant autocorrelations. The Ljung-Box suggests that some autocorrelations in the residuals are significant.

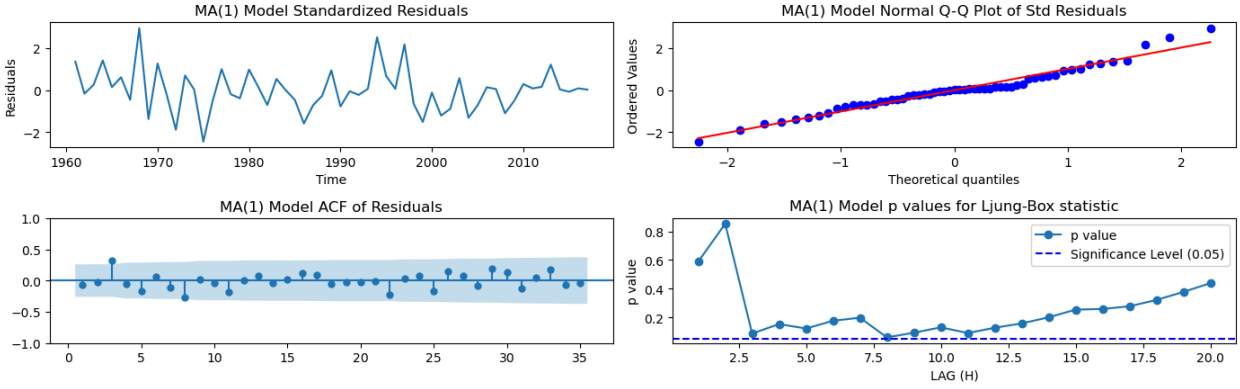

Figure 4

The standardized residuals and Normal Q-Q plots for AR(2) models are similar to MA(1), indicating no obvious patterns and approximate normality, but the AR(2) model performs better based on other diagnostics. In addition, there are no lags with significant autocorrelations in the ACF plot of AR(2). And the Ljung-Box of AR(2) model reveals its p-values remain above 0.05, indicating uncorrelated residuals.

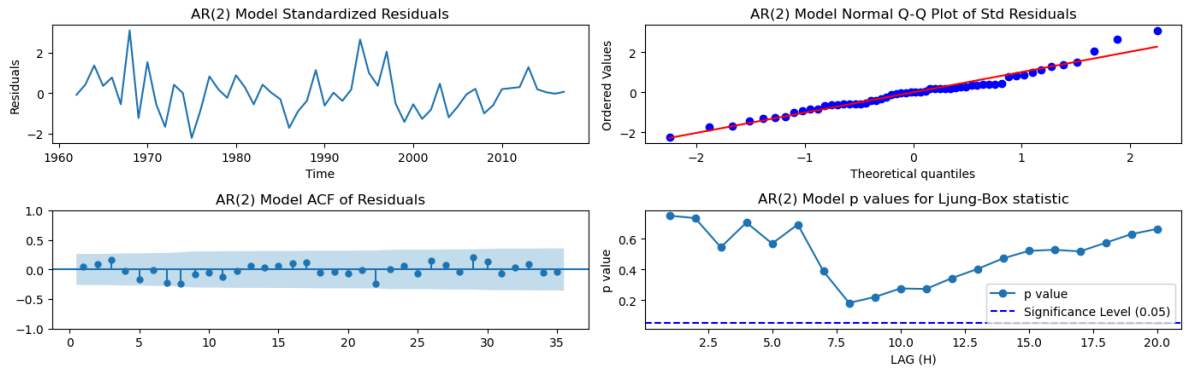

Figure 5

The AR(2) model demonstrates better diagnostic performance compared to the MA(1) model, with fewer significant autocorrelations in the residuals. Additionally, the AR(2) model has lower AIC and BIC values, so we use AR(2) for further analysis.

VI. Forecasting

After determining the model, we are now interested in using the model to predict future export data. We generated the forecast graph (Figure 6) and yielded 4 year predictions from the AR(2) model fitted to processed data from 2018 to 2021. In addition, we split the data into beginning and ending sets, allowing us to compare the predicted values with the actual historical data. This approach helps evaluate the model's performance by comparing the predicted line with the real historical data (Figure 7).

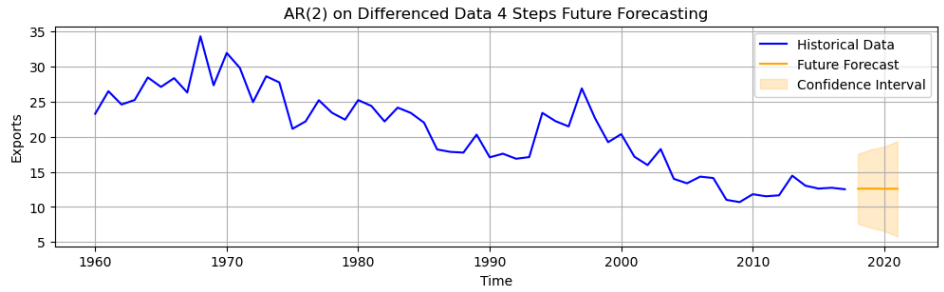

Figure 6

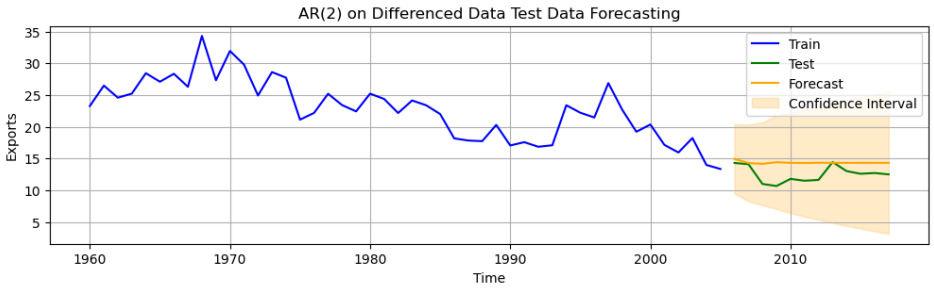

Figure 7

The forecasted values from the AR(2) model for the years 2018 to 2021 are in Table 3 below. The prediction results are approximate to 12.60, which would make the future forecast line appear to be horizontal. This is because the historical export data in the later years shows a tendency to stabilize, the AR(2) model will forecast a horizontal trend for future values.

Year	Prediction
2018	12.590701
2019	12.615136
2020	12.581762
2021	12.591539

Table 3

VII. Finding Model by MSE

We also split the data to train and test sets and use "finding the lowest MSE" as criteria to find the Best Model. After evaluating multiple model configurations, the ARIMA(1,1,2) model (on original data), or ARMA(1, 2) model (on differenced data), was selected due to its lowest MSE (1.595).

ARMA(1,2) Parameter/Equation	Value	P-value
$y_t = 0.337728 \cdot y_{t-1} - 0.883232 \cdot w_{t-1} + 0.441870 \cdot w_{t-2} + w_t$	-	-
ϕ_1 (AR Coefficient)	0.337728	4.12e-01
θ_1 (MA Coefficient 1)	-0.883232	1.86e-02
θ_2 (MA Coefficient 2)	0.441870	3.30e-03
σ^2 (Variance)	6.454815	2.19e-09
AIC	276.51	-
BIC	284.68	-

Table 4

We find out that the first AR coefficient is insignificant, so we drop this insignificant coefficient to the ARMA(0, 2) model which, however, gives a higher MSE. Since the criteria now is MSE, we kept using the ARMA(1, 2) model for forecasting, the value of it is in Table 4.

The standardized residual plot and the Q-Q plot (Figure 8) of the ARMA(1, 2) model show no obvious patterns and the residuals follow a normal distribution. The ACF plot of residuals (Figure 8) indicates no significant autocorrelation within the lags. The Ljung-Box (Figure 8) supports the hypothesis that the residuals are uncorrelated (p-values above the 0.05 significance level).

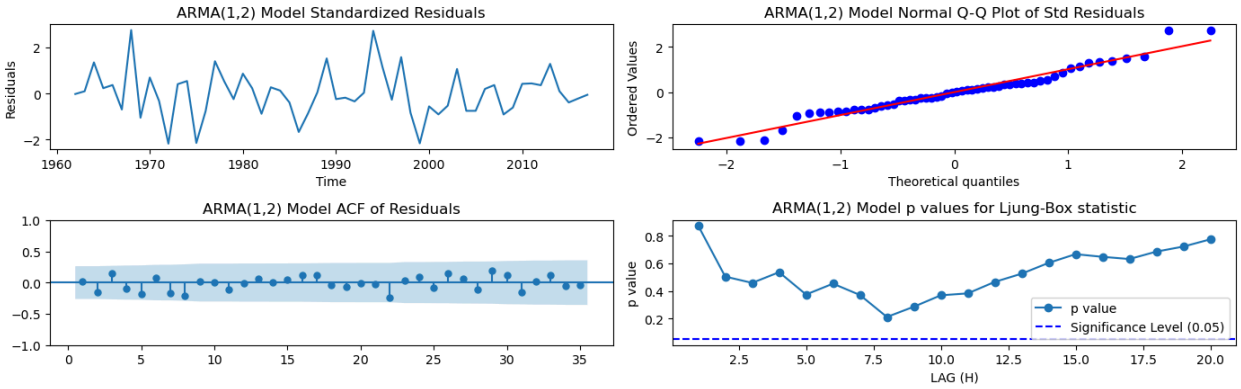

Figure 8

We will now apply the same procedure as with the AR(2) model to generate the forecast graph (Figure 9) and calculate four-year predictions (2018 to 2021) using an ARMA(1,2) model fitted to the processed data. We also generated predictions for the existing data and compared them with the actual historical data (Figure 10).

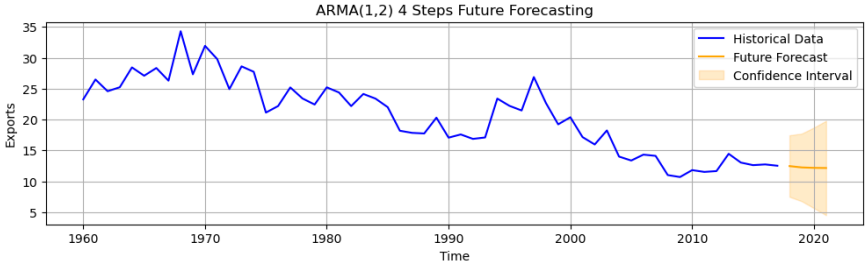

Figure 9

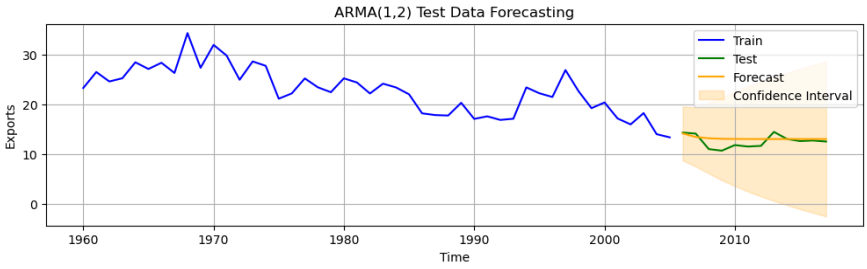

Figure 10

The forecasted values from the ARMA(1, 2) model for the years 2018 to 2021 are shown in table 5. The predictions are consistently around 12, causing the future forecast line to appear horizontal. This flattening effect occurs because of the lack of a discernible trend in the processed data.

Year	Prediction
2018	12.457999
2019	12.244672
2020	12.172626
2021	12.148294

Table 5

VIII. Conclusion

We analyzed the given time series data for stationarity using the plot, ACF, PACF, and the ADF test, confirming that the original data was non-stationary. To address this, we applied log transformation and Box-Cox transformation, but neither method achieved stationarity. Consequently, we performed the first-order differencing on the original data, which successfully yielded a stationary time series.

The ACF and PACF plots of the differenced data suggested MA(1) and AR(2) models as potential candidates. Using the ARIMA method from the statsmodels library in Python, we estimated the coefficients for these models and examined their p-values. After conducting diagnostic checks and comparing the results with the ACF and PACF plots, the AR(2) model:

$$y_t = -0.505033 \cdot y_{t-1} - 0.289666 \cdot y_{t-2} + w_t$$

emerged as the better fit.

We also explored a machine learning approach by splitting the data into Train and Test sets and identifying the model with the lowest Mean Squared Error (MSE) on the Test set. This method suggested the ARMA(1, 2) model as the optimal choice.

$$y_t = 0.337728 \cdot y_{t-1} - 0.883232 \cdot w_{t-1} + 0.441870 \cdot w_{t-2} + w_t$$

We conducted similar diagnostic checks on the ARMA(1, 2) model, as we had done for the AR(2) model, and found that it performed well. Finally, we used both the AR(2) and ARMA(1, 2) models to forecast future data and to predict values in the Test set, comparing their performance.

IX. Appendix

The code is provided in a Jupyter Notebook available at this ***Github Repo***.

To run the code, please download the accompanying .csv data file, which was converted from the original.Rdata file.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import skew
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from scipy.stats import boxcox
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error

from statsmodels.graphics.gofplots import qqplot
from statsmodels.stats.diagnostic import acorr_ljungbox
from scipy.stats import probplot
import warnings
warnings.filterwarnings("ignore")
df = pd.read_csv('Data_File_finalproject.csv')
df.head()
# set Year as index
df['Year'] = pd.to_datetime(df['Year'], format='%Y')
df.set_index('Year', inplace=True)

# we only focus on Year and Exports
df = df[['Exports']]
# Missing value
print('is there any missing value?', df['Exports'].isnull().any())
# Duplicates
print('is there any duplicated value?',df.duplicated().any())
stats = df['Exports'].describe()
stats['skew'] = skew(df['Exports'])
print(stats)
def visualize(data, name):
    # 5 years as a period
    seasonality = seasonal_decompose(data, model='additive', period=5)

    # plot
    plt.figure(figsize=(24, 9))

    # original
    plt.subplot(2, 2, 1)
    plt.plot(data, label=name)
    plt.legend(loc='upper left')
    plt.xticks(df.index[::5], df.index[::5].year)
    plt.title(f'Central African Republic {name} Exports')
    plt.xlabel('Year')
    plt.ylabel('Exports (million USD)')
    plt.grid(True)

    # trend
    plt.subplot(2, 2, 2)
    plt.plot(seasonality.trend, label='Trend')
    plt.legend(loc='upper left')
    plt.xticks(df.index[::5], df.index[::5].year)
    plt.title(f'{name} Trend')
    plt.xlabel('Year')
    plt.ylabel('Exports (million USD)')
    plt.grid(True)

    # seasonality
    plt.subplot(2, 2, 3)
    plt.plot(seasonality.seasonal, label='Seasonal')
    plt.legend(loc='upper left')
    plt.xticks(df.index[::5], df.index[::5].year)
```

```
    plt.title(f'{name} Seasonality 5-year Period')
    plt.xlabel('Year')
    plt.ylabel('Exports (million USD)')
    plt.grid(True)

    # residual
    plt.subplot(2, 2, 4)
    plt.plot(seasonality.resid, label='Residual')
    plt.legend(loc='upper left')
    plt.xticks(df.index[::5], df.index[::5].year)
    plt.title(f'{name} Residual')
    plt.xlabel('Year')
    plt.ylabel('Exports (million USD)')
    plt.grid(True)

    plt.show()
def check_is_stationary(data:float):
    # Augmented Dickey-Fuller
    # H0: non-stationary
    # H1: stationary
    adf_result = adfuller(data)
    adf_Statistic = adf_result[0]
    p = adf_result[1]
    # Printing the ADF test results
    print('ADF Statistic:', adf_Statistic)
    print('p-value:', p)
    if p > 0.05:
        print('Since p value > 0.05, failed to reject H0: the Time Series is non-stationary.')
    else:
        print('Since p value <= 0.05, rejected H0: the Time Series is stationary.')
def acf_pcf(data, name, lags=20):
    lags = 20
    plt.figure(figsize=(20, 3))
    plt.subplot(1, 2, 1)
    plot_acf(data, lags=20, ax=plt.gca(), zero=False)
    plt.xticks(range(1, lags + 1), labels=range(1, lags + 1))
    plt.title(f'{name} ACF')

    plt.subplot(1, 2, 2)
    plot_pacf(data, lags=20, ax=plt.gca(), zero=False)
    plt.xticks(range(1, lags + 1), labels=range(1, lags + 1))
    plt.title(f'{name} PACF')
    plt.show()
    check_is_stationary(data)
    visualize(df['Exports'], 'Original Data')
acf_pcf(df['Exports'], 'Original Data')
df['log_Exports'] = np.log(df['Exports'])
visualize(df['log_Exports'], 'Log Transformation Data')
acf_pcf(df['log_Exports'], 'Log Transformation Data')
df['BoxCox_Exports'], best_lambda = boxcox(df['Exports'])
visualize(df['BoxCox_Exports'], 'Box-Cox Data')
acf_pcf(df['BoxCox_Exports'], 'Box-Cox Data')
# take first order of difference
original_difference = df['Exports'].diff().dropna()
visualize(original_difference, 'original_difference Data')
acf_pcf(original_difference, 'original_difference Data')
def report_perf(fitted_model, name):
    params = fitted_model.params
    pvalues = fitted_model.pvalues
    aic = fitted_model.aic
    bic = fitted_model.bic

    # print results
    print(f"\n{name} Model Parameters:")
    print(params)
```

```
    print(f"\n{name} Model P-values:")
    print(pvalues)
    print(f"\n{name} AIC: {aic}")
    print(f"\n{name} BIC: {bic}")
# we use original difference data
# MA(1)
ma1_model = ARIMA(df['Exports'], order=(0, 1, 1))  # AR=0, I=1, MA=1
ma1_fit = ma1_model.fit()
report_perf(ma1_fit, 'MA(1) Model')

# AR(2)
ar2_model = ARIMA(df['Exports'], order=(2, 1, 0))  # AR=2, I=1, MA=0
ar2_fit = ar2_model.fit()
report_perf(ar2_fit, 'AR(2) Model')
# AR(1)
ar1_model = ARIMA(df['Exports'], order=(1, 1, 0))  # AR=1, I=1, MA=0
ar1_fit = ar1_model.fit()
report_perf(ar1_fit, 'AR(1) Model')
def diag(fitted_model, name, level=1):
    residuals = fitted_model.resid

    residuals = residuals[level:] # remove the first residual because we don't
                                  # have data to predict the first term
                              # as we need the previous term to do the prediction

    standardized_residuals = (residuals - np.mean(residuals)) / np.std(residuals)

    # Create subplots
    fig, axes = plt.subplots(2, 2, figsize=(14, 4.5))

    # Top-left: Time series plot of standardized residuals
    axes[0, 0].plot(standardized_residuals)
    axes[0, 0].set_title(f"{name} Standardized Residuals")
    axes[0, 0].set_xlabel("Time")
    axes[0, 0].set_ylabel("Residuals")

    # Bottom-left: ACF plot
    plot_acf(standardized_residuals, ax=axes[1, 0], lags=35, zero=False)
    axes[1, 0].set_title(f"{name} ACF of Residuals")

    # Top-right: Q-Q plot
    probplot(standardized_residuals, dist="norm", plot=axes[0, 1])
    axes[0, 1].set_title(f"{name} Normal Q-Q Plot of Std Residuals")

    # Bottom-right: Ljung-Box test p-values
    lags = 20  # Number of lags to check
    ljung_box = acorr_ljungbox(standardized_residuals, lags=lags, return_df=True)
    axes[1, 1].plot(ljung_box.index, ljung_box["lb_pvalue"], 'o-',
                    label="p value")
    axes[1, 1].axhline(y=0.05, color='blue', linestyle='--',
                       label="Significance Level (0.05)")
    axes[1, 1].set_title(f"{name} p values for Ljung-Box statistic")
    axes[1, 1].set_xlabel("LAG (H)")
    axes[1, 1].set_ylabel("p value")
    axes[1, 1].legend()

    # Adjust layout
    plt.tight_layout()
    plt.show()
diag(ma1_fit, 'MA(1) Model')
diag(ar2_fit, 'AR(2) Model', level=2) # we can only start at Residual3 because
                                      # we need previous 2 data points to
                                      # predict the third point
# Splitting the data into training and testing sets
```

```
train_size = int(len(df['Exports']) * 0.8)
train, test = df['Exports'][:train_size], df['Exports'][train_size:]
def calculate_MSE_Forecast_confi_int(data, ar_p, ar_i, ar_q):
    train_size = int(len(data) * 0.8)
    train, test = data[:train_size], data[train_size:]
    model = ARIMA(train, order=(ar_p, ar_i, ar_q))
    model_fit = model.fit()

    forecast_result = model_fit.get_forecast(steps=len(test))
    forecast = forecast_result.predicted_mean
    conf_interval = forecast_result.conf_int()

    mse = mean_squared_error(test, forecast)
    return mse, forecast, conf_interval
def forecast_future(data, ar_p, ar_i, ar_q, steps=4, alpha=0.05):
    model = ARIMA(data, order=(ar_p, ar_i, ar_q))
    model_fit = model.fit()

    forecast_result = model_fit.get_forecast(steps=steps, alpha=alpha)
    forecast = forecast_result.predicted_mean
    conf_interval = forecast_result.conf_int()

    return forecast, conf_interval
def plot_test_prediction(prediction, conf_int, name, future_prediction=False):
    plt.figure(figsize=(12, 3))
    if future_prediction:
        plt.plot(df.index, df['Exports'], label='Historical Data', color='blue')
        future_index = pd.date_range(
                    start=df.index[-1] + pd.DateOffset(1),
                    periods=len(prediction),
                    freq='AS'
                )
        print('prediction:', prediction)
        plt.plot(future_index, prediction, label='Future Forecast', color='orange')
        plt.fill_between(
            future_index,
            conf_int.iloc[:, 0],
            conf_int.iloc[:, 1],
            color='orange',
            alpha=0.2,
            label='Confidence Interval'
        )
        plt.title(f'{name} 4 Steps Future Forecasting')
    else:
        plt.plot(train.index, train, label='Train', color='blue')
        plt.plot(test.index, test, label='Test', color='green')
        plt.plot(test.index, prediction, label='Forecast', color='orange')
        plt.fill_between(
            test.index,
            conf_int.iloc[:, 0],
            conf_int.iloc[:, 1],
            color='orange',
            alpha=0.2,
            label='Confidence Interval'
        )
        plt.title(f'{name} Test Data Forecasting')
    plt.xlabel('Time')
    plt.ylabel('Exports')
    plt.legend()
    plt.grid(True)
    plt.show()
mse210, forecase210, confi_interal210 = calculate_MSE_Forecast_confi_int(df['Exports'],
                                                                         2, 1, 0)
plot_test_prediction(forecase210,confi_interal210, 'AR(2) on Differenced Data')
forecase_future210, confi_interal_future210 = forecast_future(df['Exports'], 2,
```

```
                                                                1, 0, steps=4)
plot_test_prediction(forecase_future210, confi_interal_future210,
                     'AR(2) on Differenced Data', future_prediction=True)
# Defining the range of values for p and q
p_values = range(0, 10)
q_values = range(0,10)
i_values = range(0, 3)


# Initializing the variables to store the best model and its performance
best_mse = float('inf')
best_order = None
best_forecast = None
best_conf_interval = None


# Brute Force iterate all the combinations to find the best set of parameter
# Parameter tuning to find the best values of p, i, and q
for p in p_values:
    for q in q_values:
        for i in i_values:
            try:
                mse, forecast, conf_interval = calculate_MSE_Forecast_confi_int
                 (df['Exports'], ar_p=p, ar_i=i, ar_q=q)

                # report MSE
                print(f'ARIMA{(p, i, q)} model: MSE={mse}')

                # Updating the best model based on MSE
                if mse < best_mse:
                    best_mse = mse
                    best_order = (p, i, q)
                    best_forecast = forecast
                    best_conf_interval = conf_interval

            except Exception as e:
                print(f"Skipping (p={p}, q={q}) due to error: {e}")

# Printing the best model parameters
print(f'Best ARIMA model: order={best_order} with MSE = {best_mse}')
arma112 = ARIMA(df['Exports'], order=(1, 1, 2))
arma112_fit = arma112.fit()
report_perf(arma112_fit, 'ARIMA(1,1,2) Model')
arma012 = ARIMA(df['Exports'], order=(0, 1, 2))
arma012_fit = arma012.fit()
report_perf(arma012_fit, 'ARIMA(0,1,2) Model')
mse012, _, _= calculate_MSE_Forecast_confi_int(df['Exports'], 0, 1, 2)
print(f'ARIMA{best_order}: MSE={best_mse}')
print(f'ARIMA(0, 1, 2): MSE={mse012}')
diag(arma112_fit, 'ARMA(1,2) Model', level=2)
plot_test_prediction(best_forecast, best_conf_interval, 'ARMA(1,2)')
forecase_future112, confi_interal_future112 = forecast_future(df['Exports'], 1,
                                                              1, 2, steps=4)
plot_test_prediction(forecase_future112, confi_interal_future112, 'ARMA(1,2)',
                     future_prediction=True)
```