Econometría II: Final Project

Time series modeling with ARIMA
Capturing time invariant corelations

Daniel Emanuel Albert Strauss
ID: 186021

Christian W. McDonald Ehrenberg
ID: 165650

November 30, 2024

Supervisor: Dr. Daniela Cortés Toto

Universidad de las Américas Puebla (UDLAP)

Abstract

Will the sun rise up tomorrow? Many people believe the sun will rise tomorrow since it rose every last day of their lives. Is that a logical conclusion? At least empirically seen, on every past day of their lives, it was more useful to predict the sun would do that thing the next morning than it did all days in their life before that past day. But is it logical to conclude that if a conclusion proved to be useful on every past day of one's life, it would be useful today? To not turn crazy, let's just make that conclusion, independent of whether it is logical or not. Carefully picking some things to assume to be time-invariant leaves us with a new ability: making predictions about the future from observations in the past. For example, one could pick, as such, the stationarity of a time series. If one wants to predict a time series W_t that takes algebraic values, such information about the future may lie in estimated autocorrelations and partial autocorrelations observed from the past. If this series is stationary, the correlations will repeat in the future. Empirically, we can not prove stationarity, but we can observe statistically significant stationarity. How can we use this knowledge to exploit autocorrelations and partial autocorrelations best for future predictions? In the course Econometria II, we learned about the theory and the practical usage of ARIMA models. ARIMA models are a family of models capable of expressing autocorrelation and partial autocorrelation functions (ACF and PACF) of a time series's stationary and invertible differentiation. This project aimed to model and validate a real-life time series with an ARIMA model.

1 Introduction

1.1 Problem to Investigate and Justification

Will the sun rise up tomorrow? Will it be colder in December than in July? At what time will it rain today? All of these phenomena have in common that they can be predicted more precisely from observations in the past. (At least so it seems.) Further examples are: Which stock should I buy? When should I sell my stock? Not only can these examples be predicted in the future from the past, but predicting them also yields practical use. There is a series of mathematical methods to do such. One of them are ARIMA models, at which we will look at this work.

1.2 Objective

The objective of this project is to fit an ARIMA model to a real-life time series we choose by ourselves and to validate the ARIMA model.

To fit the ARIMA models and predict future periods, we will follow this procedure:

- Optionally apply a transformation like BoxCox to ensure constant variance after differentiation
- Evaluate a sufficient amount of differentiations d to make the time-series stationary
- From the differentiated time series, we can extract the ACF and PACF to get an idea of the ARIMA model's necessary dimensions (p, q). (More on p and q later)
- Use the statsmodels library to fit ARIMA models with different suggestions of (p, d, q) to the undiffer-

entiated data

- Use the fitted models to make predictions of the optionally transformed series
- If a transformation was applied, apply the inverse transformation and correct the bias if needed.

To verify the ARIMA models, they have to fulfill 8 criterions.

1. Residuals have zero mean.
2. Resuduals have a constant variance.
3. Residuals are normally distributed.
4. Residuals not correlated.
5. Model parameters significantly non-zero.
6. Models are stationary and invertible
7. Model parameters have low correlation.
8. Atypical data points do not influence the result much.

The first four conditions mean that the residuals must be Gaussian noise. If they weren't, there would still be information left that was not captured by the model, even though the model could capture it, i.e., the residual criteria check if the model captured everything that the model can capture. The conditions on the parameters check the opposite; they check if the model does not contain redundancy. More on that later.

1.3 Research Hypothesis

We want to test the effectiveness of the AR and MA parts of ARIMA models. Therefore, we want to see if ARIMA models with chosen p and q perform better on the verification criteria as well as on the prediction than ARIMA models with poorly chosen p and q. Therefore, we will prosecute the above-described procedure both for well-chosen pairs of p and q as well as for bad-chosen dimensions and, in the end, derive a conclusion from our observations.

2 Methodology

2.1 Dataset

To choose an ARIMA-friendly data set, we used the WWWUsage dataset from [MWH08], which is often used in education examples. For testing and verification purposes, we also added code to create synthetic data, as shown in the code below.

```python
# Variable para alternar entre datos artificiales y datos reales
test_with_artificial = False

# Funcion para generar datos sintéticos siguiendo un modelo ARIMA
def artificial_arima(p=np.array([]), d=0, q=np.array([]), f=lambda x: x, n=100, m=0):
    """
    Genera datos sintéticos basados en un modelo ARIMA para validar el método en datos reales.

    Parámetros:
    - p: Coeficientes del modelo AR (Autoregresivo).
    - d: Numero de diferencias acumulativas para hacer la serie estacionaria.
    - q: Coeficientes del modelo MA (Media Movil).
    - f: Función de transformación aplicada a los datos generados.
    - n: Número de puntos en la serie temporal.
    - m: Media del ruido blanco agregado.

    Retorna:
    - Serie transformada basada en los parámetros proporcionados.
    """
    a = np.random.normal(0, 1, n)  # Generar ruido blanco con media 0 y varianza 1
    W = np.zeros(n)  # Inicializar la serie temporal

    for t in range(n):
        if t < len(p) or t < len(q):  # Manejar índices fuera de rango
            W[t] = 0
        else:
            # Aplicar componentes AR y MA usando productos escalares
            W[t] = -W[t-len(p):t] @ p[::-1] + a[t] + a[t-len(q):t] @ q[::-1]

    for d_c in range(d):  # Aplicar diferenciación acumulativa d veces
        W = np.cumsum(W)

    W += m  # Agregar media a la serie
    return f(W)  # Aplicar la transformación final

# Si se activa `test_with_artificial`, generar datos sintéticos
if test_with_artificial:
    n = 1000  # Número de puntos en la serie sintética
    points = artificial_arima(
        p=np.array([.5,-.4]),  # Coeficientes AR
        q=np.array([0]),  # Coeficientes MA
        d=1,  # Diferenciación
        f=lambda x: 1.01**x,  # Función de transformación exponencial
        n=n,  # Tamaño de la serie
        m=0  # Media
    )
    # Crear un índice de fechas mensual empezando desde 1800
    dates = pd.date_range(start='1800-01-01', periods=n, freq='MS')
    # Crear un DataFrame con los datos generados y el índice temporal
    time_series_df = pd.DataFrame({'points': 1.7**points}, index=dates)
```

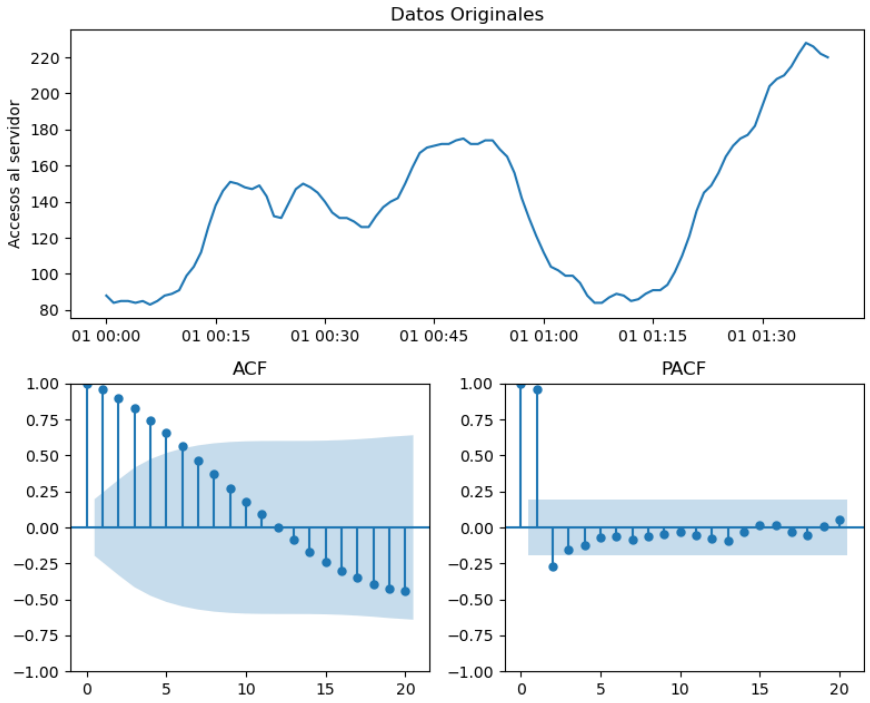

Figure 1: WWWUsage dataset

2.2 Identify a Box-Jenkins Model

2.2.1 Evaluate Stationariry

To evaluate stationarity, we used the statsmodels library. This library provides the Augmented Dickey–Fuller test, which has the null hypothesis, that a unit root is present, and as an alternative hypothesis, that the series is stationary. Since the p-value was roughly 12% we did not conclude stationarity.

```
import statsmodels.tsa.stattools as st  # Herramientas estadísticas para series temporales

# Verificar la estacionariedad de la serie temporal
# La hipótesis alternativa de la prueba ADF es que la serie es estacionaria
adf_result = st.adfuller(time_series_df['points'])  # Realizar la prueba ADF
```

2.2.2 Why we did not need a transformation to stabilize the variance

Since the data looked homoscedastic after two differentiations, it seemed like it was not necessary to apply a transformation to stabilize the variance. Furthermore, we tested fitting ARIMA models after applying the box Cox transformation with automated λ parameter and received not as good results when verifying the 8 criteria as without the transformation. Therefore we did not use any transformation. When we tested how the box-cox transformation would turn out, we used the Scipy stats library to automatically find λ.

```
from scipy import stats  # Importar herramientas estadísticas de SciPy

# Variable para controlar si se aplica la transformación Box-Cox
use_trafo = False

# Verificar si se aplica la transformación Box-Cox
```

```
if use_trafo:
    # Aplicar la transformación Box-Cox y estimar lambda automáticamente si no se especifica
    lmbda = None  # Valor inicial de lambda; si es None, se estima automáticamente
    if lmbda is None:
        # Aplicar Box-Cox y estimar lambda óptimo
        transformed_prices, lmbda = stats.boxcox(time_series_df["points"])
    else:
        # Aplicar Box-Cox con un valor específico de lambda
        transformed_prices = stats.boxcox(time_series_df["points"], lmbda=lmbda)

    # Almacenar los valores transformados en el DataFrame
    time_series_df['T_points'] = transformed_prices
    print("Lambda usado: ", lmbda)  # Mostrar el valor de lambda utilizado
else:
    # Si no se aplica la transformación, conservar la serie original
    lmbda = None
    time_series_df['T_points'] = time_series_df['points']

# Graficar la serie temporal (transformada o sin transformar)
plot_series(time_series_df['T_points'], f"Serie Temporal Transformada con Box-Cox (lambda={lmbda})
    ")
```

2.2.3 Differentiate the series

In order to make the series stationary we had to differentiate the series. Nicely, the Pandas' data frame provides a function 'diff()', which applies differentiation. To ensure the minimum necessary amount of differentiations to achieve stationarity, we differentiated the series until the Augmented Dickey–Fuller test significantly ($\alpha = 5\%$) rejected the null hypothesis.

```

# Nivel de significancia para la prueba ADF
alpha = 0.05

# Seleccionar la serie transformada sin valores nulos
current_series = time_series_df['T_points'].dropna()
d = 0  # Contador de diferenciaciones aplicadas

# Función para obtener el p-valor de la prueba ADF
def get_adf_p_value(series):
    """
    Realiza la prueba Dickey-Fuller Aumentada (ADF) y retorna el p-valor.

    Parámetros:
    - series: Serie temporal a analizar.

    Retorna:
    - p-valor de la prueba ADF.
    """
    adf_result = st.adfuller(series.dropna())  # Realizar la prueba ADF
    return adf_result[1]  # Retornar el p-valor

# Iterar hasta que la serie sea estacionaria o se alcance el máximo de diferenciaciones
while get_adf_p_value(current_series) >= alpha:
    d += 1  # Incrementar el contador de diferenciaciones
```

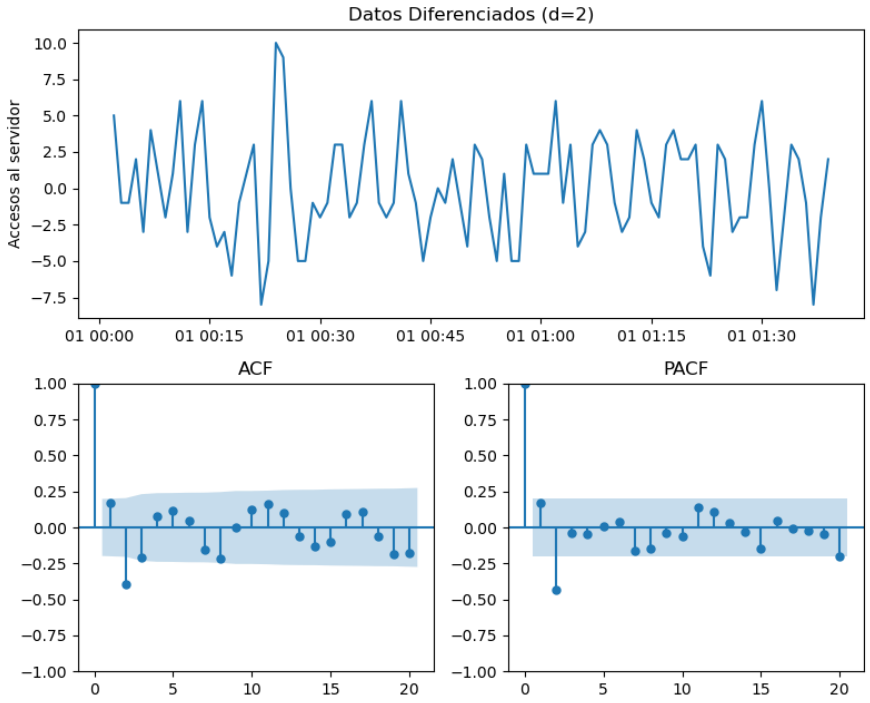

Figure 2: WWWUsage dataset after two differentiations

```
    # Aplicar diferenciación de primer orden
    current_series = current_series.diff()  # Calcula la diferencia entre valores consecutivos

    # Imprimir el progreso y el p-valor después de cada diferenciación
    print(f"Después de {d} diferenciación(es), el p-valor de ADF es: {get_adf_p_value(
    current_series)}")

# Agregar la serie diferenciada al DataFrame original
time_series_df['diff_points'] = current_series.copy()

# Verificar si la serie es estacionaria después de la diferenciación
if get_adf_p_value(current_series) < alpha:
    print(f"La serie es estacionaria después de {d} diferenciación(es).")
else:
    print("Se alcanzó el máximo de diferenciaciones sin lograr estacionariedad.")
```

We achieved for this dataset, significant stationarity after $d = 2$ differentiations.

2.2.4 Identify significant autocorrelations and partial autocorrelations

The statsmodels library provides methods to obtain the significant partial autocorrelations:

```
import statsmodels.tsa.stattools as st
# Calculate ACF with confidence intervals
acf_values, acf_confint = st.acf(data, alpha=alpha, fft=True, nlags=nlags, adjusted=True)
pacf_values, pacf_confint = st.pacf(data, alpha=alpha, nlags=nlags)
```

If the confidence intervall of a given lag did not include 0 this would mean, that the corresponsing correlation would be significantly non zero.

In the end, we obtained these significant ACF and PACF:

```
Significant lags at 5.0% significance level:
----------------------------------------
Lag | ACF Value | Confidence Interval
----------------------------------------
  0 |     1.000 | [ 1.000,  1.000]
  2 |    -0.399 | [-0.603, -0.195]
 24 |    -0.339 | [-0.640, -0.038]

Significant lags at 5.0% significance level:
----------------------------------------
Lag | PACF Value | Confidence Interval
----------------------------------------
  0 |      1.000 | [ 1.000,  1.000]
  2 |     -0.443 | [-0.641, -0.245]
 20 |     -0.266 | [-0.464, -0.068]
 23 |     -0.258 | [-0.456, -0.061]
 24 |     -0.263 | [-0.461, -0.065]
```

Listing 1: Significant Correlations

2.2.5 Proposal of Box-Jenkins Models

As seen in Table 1, the only significant autocorrelations and partial autocorrelations lie at lag 2. The correlations after lag 20 are at a very big lag and introduce a lot of parameters, which is why we choose not to model these correlations. Since the correlations at lag 1 are not significantly not zero, we are relatively sure that an ARIMA model with $p, q \leq 1$ would not be able to capture these autocorrelations. Thus, we choose, as reasonable models, the models ARIMA(0,2,2), ARIMA(2,2,0), and ARIMA(2,2,2). As unreasonable models (to control the reasonable models), we choose ARIMA(0,2,1), ARIMA(1,2,0), ARIMA(1,2,1), and the white noise model ARIMA(0,2,0).

To fit these models, we used the ARIMA class of statsmodels. Specifically, we used the method 'statsmodels.tsa.arima.model.ARIMA.fit()'.

```python
from statsmodels.tsa.arima.model import ARIMA  # Importar el modelo ARIMA

# Seleccionar la serie transformada y eliminar valores nulos
data = time_series_df['T_points'].dropna()

# Definir los modelos ARIMA sugeridos
suggested_models = np.array([
    [0, d, 0],  # Modelo ARIMA(0,d,0)
    [0, d, 1],  # Modelo ARIMA(0,d,1)
    [1, d, 0],  # Modelo ARIMA(1,d,0)
    [1, d, 1],  # Modelo ARIMA(1,d,1)
    [0, d, 2],  # Modelo ARIMA(0,d,2)
    [2, d, 0],  # Modelo ARIMA(2,d,0)
    [2, d, 2],  # Modelo ARIMA(2,d,2)
```

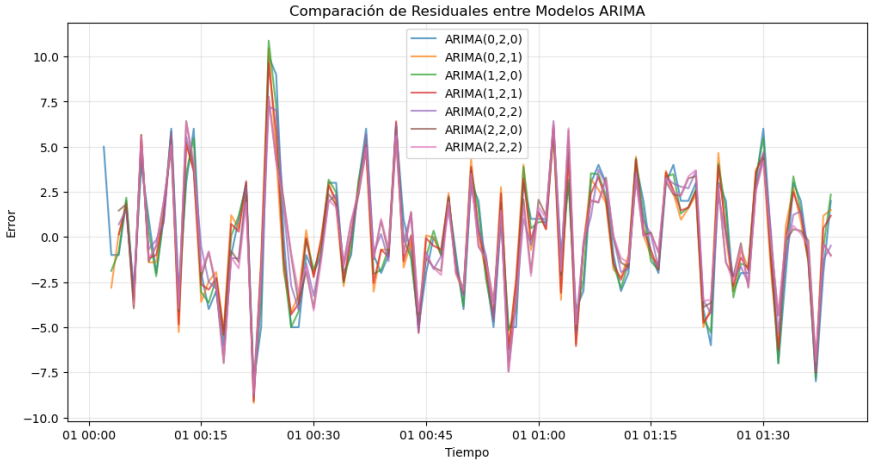

Figure 3: Residiuals of the different fitted ARIMA models

```
])


# Iterar sobre los modelos sugeridos y ajustar cada uno
for p, d, q in suggested_models:
    # Ajustar el modelo ARIMA con los parámetros actuales (p, d, q)
    model = ARIMA(data, order=(p, d, q))
    fitted = model.fit()
```

2.3 Verification of the Models

2.3.1 Model verification from the residuals

Four of the eight criteria can be extracted from the residuals. They are

1. Residuals have zero mean.

2. Resuduals have a constant variance.

3. Residuals are normally distributed.

4. Residuals not correlated.

If one of the criteria of 1,2,4 is violated, it may indicate that there is information that can be captured by box Jenkins models but has not been captured. ARIMA models with high enough dimensions, for example, should be able to capture all autocorrelations, which would have, as a consequence, that the residuals do not have any autocorrelation.

Note that for all of these criteria, there exists a hypothetical time series, that is, ϵ-close to fulfill this criterion, without actually fulfilling it. For example, the mean of the residuals could be $\epsilon > 0$ instead of 0, or the residuals follow a distribution that is almost a normal distribution but not the normal distribution itself.

That's why it is statistically impossible without further assumptions to observe that any of these criteria are fulfilled with a significance level smaller than 1. What we do instead is to try to reject the criteria with the standard significance level of 5%. Given that we have enough data and that the actual properties of the residual are close to the properties in the criteria, we hope that this does not lead to the rejection of the null hypothesis and that the opposite scenario does.

Mean of Zero: To evaluate whether the residuals have a zero mean, we apply a one-sample t-test, which is included in the Scipy stats library. Note that a one-sample t-test assumes normality and independence of residuals. Normality, we will test later, and for independence, we will test if they are not correlated.

```
# 1. Prueba de media cercana a 0 (t-test)
p_value = ttest_1samp(residuals, 0).pvalue
results['mean_close_to_0'] = p_value > alpha  # La media está cerca de 0 si p-valor > alpha
print(f"{model_name}: Media de residuales = {np.mean(residuals):.4f}, p-valor = {p_value:.4f}")
```

Constant Variance: To evaluate whether the residuals are homoscedastic, we use the Breusch-Pagan test, which is part of the statsmodels.stats.diagnostic package. Its null hypothesis is homoscedasticity, so we opt again for no-reject.

```
# 2. Prueba de varianza constante (homocedasticidad) usando Breusch-Pagan
_, pvalue, _, _ = het_breuschpagan(residuals, sm.add_constant(np.arange(len(residuals))))
results['constant_variance'] = pvalue > alpha  # Pasa si p-valor > alpha
print(f"{model_name}: Homocedasticidad (p-valor de Breusch-Pagan) = {pvalue:.4f}")
```

Normally Distributed: To evaluate whether the residuals are normally distributed, we consider two tests both implemented in the scipy.stats library. Namely, we use the Shapiro-Wilk and Jarque-Bera Tests, which both have a zero hypothesis for the normality of the residuals.

```
# 3. Pruebas de normalidad (Shapiro-Wilk y Jarque-Bera)
_, shapiro_pvalue = shapiro(residuals)  # Prueba Shapiro-Wilk
jb_stat, jb_pvalue = jarque_bera(residuals)  # Prueba Jarque-Bera
results['normal_distribution'] = shapiro_pvalue > alpha and jb_pvalue > alpha  # Ambas deben pasar
print(f"{model_name}: Normalidad (p-valor Shapiro-Wilk) = {shapiro_pvalue:.4f}")
print(f"{model_name}: Normalidad (p-valor Jarque-Bera) = {jb_pvalue:.4f}")
```

Uncorelated Residuals: To evaluate whether the residuals are not correlated, we use the Ljung-Box Test, implemented in the statsmodels.stats.diagnostic library. For n lags into the past, it can tell for each of these lags if the according to estimated autocorrelation is significantly not zero. If none of the autocorrelations are significantly not zero, we do not reject the null hypothesis that the residuals are uncorrelated.

```
# 4. Prueba de independencia usando Ljung-Box
lb_test = acorr_ljungbox(residuals, lags=[10], return_df=True)
pvalue_ljungbox = lb_test['lb_pvalue'].values[0]
results['independent_errors'] = pvalue_ljungbox > alpha  # Pasa si p-valor > alpha
print(f"{model_name}: Independencia (p-valor de Ljung-Box) = {pvalue_ljungbox:.4f}")
```

In the end, we obtained these results:

Model	Mean Close to 0	Constant Variance	Normal Distribution	Not Correlated Errors
ARIMA(0,2,0)	Pass	Pass	Pass	Fail
ARIMA(0,2,1)	Pass	Pass	Pass	Fail
ARIMA(1,2,0)	Pass	Pass	Pass	Fail
ARIMA(1,2,1)	Pass	Pass	Pass	Fail
ARIMA(0,2,2)	Pass	Pass	Pass	Pass
ARIMA(2,2,0)	Pass	Pass	Pass	Pass
ARIMA(2,2,2)	Pass	Pass	Pass	Pass

We see that for ARIMA models with $p, q \leq 1$, we see that the residuals have significant autocorrelations. This indicates they did not capture all autocorrelations from the original stationary data. Thus, this was to be expected as both the ACF and the PACF had a peak at 2. We see an example of why it is useful to check the 8 criteria.

2.3.2 Model verification from the parameter estimators

Significance of Parameters: One of the criteria was to check that the estimators for the parameters of the ARIMA model are significantly different from zero. If, for example, a model's highest MA degree parameter would not be significantly different from zero, this would mean that this parameter is redundant and that redundancy could be decreased by lowering the MA dimension. The statsmodels ARIMAResults class provides confidence intervals for the estimated parameters of the ARIMA models. If the confidence interval does not include 0, then the according parameter is significantly different, not zero.

We wrote some code to check for all proposed models if all parameters would be significantly non-zero and then printed which models passed this test and which ones did not:

```
1
2 Resumen de Significancia de Parámetros de Modelos ARIMA Basado en Intervalos de Confianza:
3 ==================================================
4 Modelos que Aprobaron (Intervalos de Confianza NO contienen cero):
5 ['ARIMA(0,2,0)', 'ARIMA(0,2,1)', 'ARIMA(2,2,0)']
6
7 Modelos que Fallaron (Intervalos de Confianza contienen cero):
8 ['ARIMA(1,2,0)', 'ARIMA(1,2,1)', 'ARIMA(0,2,2)', 'ARIMA(2,2,2)']
```

For the model ARIMA(0,2,2) it is important to note that it did not pass this test because the AR(1) parameter was not significantly not zero. The AR(2) parameter was on the other hand. Thus changing the ARIMA(0,2,2) model into an ARIMA(0,2,1) model would remove more than just redundancy as far as I can see it. Thus, one could argue that the ARIMA(0,2,2) model actually passed the test.

Admisible Models: Another criterion that the estimated models have to fulfill is stationarity and invertability. This was easy to check since the Numpy library provides a method "roots" to evaluate the roots of a polynomial. From the moving average and from the autoregressive parameters the according polynomials could be deducted and inserted into numpy.roots. If, for both polynomials, the roots would lie outside the unit circle, the model was admissible, i.e., stationarity and invertible.

In our case, we obtained all modules to be admissible.

```
Resumen de la Prueba de Admisibilidad Basada en Raíces de los Polinomios MA y AR:
==================================================
Modelos que Pasaron (Modelos Admisibles con Raíces Fuera del Círculo Unitario):
['ARIMA(0,2,0)', 'ARIMA(0,2,1)', 'ARIMA(1,2,0)', 'ARIMA(1,2,1)', 'ARIMA(0,2,2)', 'ARIMA(2,2,0)', '
    ARIMA(2,2,2)']

Modelos que Fallaron (Modelos No Admisibles con Raíces Dentro o en el Círculo Unitario):
[]
```

Correlations of the Parameters: It is better if the estimators for the model parameters have low correlations since then, I believe the model carries less redundancy.

The statsmodels ARIMAResults class offers the covariance matrix of the parameter estimators. Also the statsmodels library offers the function to turn a covariance matrix into a correlation matrix. Thus we calculated the correlation matrixes like this:

```
# Extraer la matriz de varianza-covarianza de los parámetros estimados
cov_matrix = fitted_model.cov_params()  # Matriz de covarianza de los parámetros

# Convertir la matriz de covarianza a una matriz de correlación
correlation_matrix = cov2corr(cov_matrix)
```

We obtained these correlation matrixes (I only print the matrixes for models with more than one parameter):

```
Matriz de Correlación de Parámetros para ARIMA(1,2,1):
           ar.L1     ma.L1    sigma2
ar.L1   1.000000 -0.882550  0.084942
ma.L1  -0.882550  1.000000 -0.043799
sigma2  0.084942 -0.043799  1.000000

Matriz de Correlación de Parámetros para ARIMA(0,2,2):
           ma.L1     ma.L2    sigma2
ma.L1   1.000000  0.318789  0.135840
ma.L2   0.318789  1.000000 -0.095333
sigma2  0.135840 -0.095333  1.000000

Matriz de Correlación de Parámetros para ARIMA(2,2,0):
           ar.L1     ar.L2    sigma2
ar.L1   1.000000 -0.154842  0.235739
ar.L2  -0.154842  1.000000 -0.038835
sigma2  0.235739 -0.038835  1.000000

Matriz de Correlación de Parámetros para ARIMA(2,2,2):
           ar.L1     ar.L2     ma.L1     ma.L2    sigma2
ar.L1   1.000000 -0.141192 -0.900061 -0.092983  0.230904
ar.L2  -0.141192  1.000000  0.125011 -0.870712  0.121231
ma.L1  -0.900061  0.125011  1.000000  0.073469 -0.144525
ma.L2  -0.092983 -0.870712  0.073469  1.000000 -0.216743
sigma2  0.230904  0.121231 -0.144525 -0.216743  1.000000
```

For the ARIMA(2,2,0) models, we can see a lower correlation of the parameters as for the ARIMA(0,2,2) model. In the ARIMA(2,2,2) model we see low correlations in general but high correlations between the AR(1) and MA(1) and the AR(2) and MA(2) part, indicating that both an autoregressive as well as a moving average part may not be necessary.

2.3.3 Criterion 8

As visible in Figure 1, our dataset does not contain any noticeable outliers. Therefore no further analysis is needed.

2.3.4 Which model to choose?

From our proposed models ARIMA(0,2,2), ARIMA(2,2,0), and ARIMA(2,2,2), the model ARIMA(2,2,0) looked the best with respect to the 8 criteria. Whilst all models passed all criteria for the residuals and all models are admissible, ARIMA(2,2,0) has the lowest correlations of the parameters and it was the only model, that just had parameters that were significantly not zero. Therefore we choose the ARIMA(2,2,0) model to presecute predictions.

2.4 Prediction

Here again, the statsmodels class ARIMAResults provides the function predictions to allow us to obtain the predictions of a fitted ARIMA model. If the requested times of the predictions lie within that time training data for the model stems from, then the model will predict one step into the future. For both prediction tasks (predict the last ten datapoints points and predict the next ten data points points), we also plotted the confidence intervals.

```
# Realizar pronósticos
forecast = fitted_model.get_prediction(start=start, end=end)
forecast_mean = forecast.predicted_mean  # Valores pronosticados
forecast_ci = forecast.conf_int(alpha=1 - confidence_level)  # Intervalos de confianza ajustados
```

In figure 4 we see the predictions of our chosen model ARIMA(2,2,0) and the control random model ARIMA(0,2,0) one step into the future. In table 2, we see the results of the forecast compared to the original data of the ARIMA(2,2,0) model, and in table 3, we see the results of the ARIMA(0,2,0) model.

In figure 5 we see the predictions of our chosen model ARIMA(2,2,0) and the control random model ARIMA(0,2,0) 10 steps into the future after the original data has ended. In table 4 we see the results of this forecastthe ARIMA(2,2,0) model and in table 5 we see the results of the ARIMA(0,2,0) model.

Time	Forecast	Original Data	Difference	Lower CI	Upper CI
01:30:00	188.66	193.00	4.34	182.42	194.89
01:31:00	204.23	204.00	-0.23	197.99	210.46
01:32:00	212.36	208.00	-4.36	206.12	218.59
01:33:00	210.19	210.00	-0.19	203.96	216.43
01:34:00	214.57	215.00	0.43	208.33	220.81
01:35:00	221.66	222.00	0.34	215.42	227.89

01:36:00	228.19	228.00	-0.19	221.96	234.43
01:37:00	232.86	226.00	-6.86	226.62	239.10
01:38:00	222.38	222.00	-0.38	216.14	228.61
01:39:00	221.01	220.00	-1.01	214.77	227.25

Table 2: Prodiction of ARIMA(2,2,0) with 95% Confidence Interval (CI) in the last 10 periods for the next period.

Time	**Forecast**	**Original Data**	**Difference**	**Lower CI**	**Upper CI**
01:30:00	187.00	193.00	6.00	179.93	194.07
01:31:00	204.00	204.00	-0.00	196.93	211.07
01:32:00	215.00	208.00	-7.00	207.93	222.07
01:33:00	212.00	210.00	-2.00	204.93	219.07
01:34:00	212.00	215.00	3.00	204.93	219.07
01:35:00	220.00	222.00	2.00	212.93	227.07
01:36:00	229.00	228.00	-1.00	221.93	236.07
01:37:00	234.00	226.00	-8.00	226.93	241.07
01:38:00	224.00	222.00	-2.00	216.93	231.07
01:39:00	218.00	220.00	2.00	210.93	225.07

Table 3: Prodiction of ARIMA(0,2,0) with 95% Confidence Interval (CI) in the last 10 periods for the next period.

Time	**Forecast**	**Lower CI**	**Upper CI**
01:40:00	219.40	213.16	225.63
01:41:00	218.27	202.87	233.68
01:42:00	216.40	191.48	241.32
01:43:00	214.56	180.10	249.02
01:44:00	213.06	168.23	257.90
01:45:00	211.64	155.16	268.11
01:46:00	210.08	140.95	279.21
01:47:00	208.46	125.94	290.97
01:48:00	206.87	110.30	303.45
01:49:00	205.33	93.97	316.70

Table 4: Forecast for ARIMA(2,2,0) with 95% Confidence Interval 10 periods into the future.

Time	**Forecast**	**Lower CI**	**Upper CI**
01:40:00	218.00	210.93	225.07

01:41:00	216.00	200.20	231.80
01:42:00	214.00	187.56	240.44
01:43:00	212.00	173.29	250.71
01:44:00	210.00	157.59	262.41
01:45:00	208.00	140.59	275.41
01:46:00	206.00	122.39	289.61
01:47:00	204.00	103.07	304.93
01:48:00	202.00	82.70	321.30
01:49:00	200.00	61.34	338.66

Table 5: Forecast for ARIMA(0,2,0) with 95% Confidence Interval 10 periods into the future.

3 Conclusion

In this work, we fitted ARIMA models to the WWWUsage dataset. First, we found out how many differentiations are necessary for stationarity. Then, from the obtained stationary series, we could deduct that either q or p must be at least 2. Then we fitted some ARIMA models with different dimensions. some ARIMA models, fullfiled the cretierion of having q or p at the value 2. Others had smaller values, and one was even the model that assumed noise after differentiation, e.g., it was an ARIMA(0,2,0) model. The models with badly chosen parameters indeed performed worse when it came to verifying the 8 criteria of the model. This was prominent when it was about verifying the criteria for the residuals, which indicated that the models with lower dimensions were indeed not able to capture the to-be-captured autocorrelations. This suggests that, on the other hand, the AR or MR components could indeed express relevant parts of the probability distribution of the dataset. However, when comparing the predictions of the ARIMA(0,2,2) model to the predictions of the ARIMA(0,2,0) model, the predicted data points were very close. This suggests that, in this case, the differentiation contributed the most to the prediction and that the MA component was only of secondary matter. It is important to note that the obtained results hold only for this dataset and that one should not be surprised when, from observing other datasets, different results are observed.

References

[MWH08] Spyros Makridakis, Steven C Wheelwright, and Rob J Hyndman. *Forecasting methods and applications*. John wiley & sons, 2008.

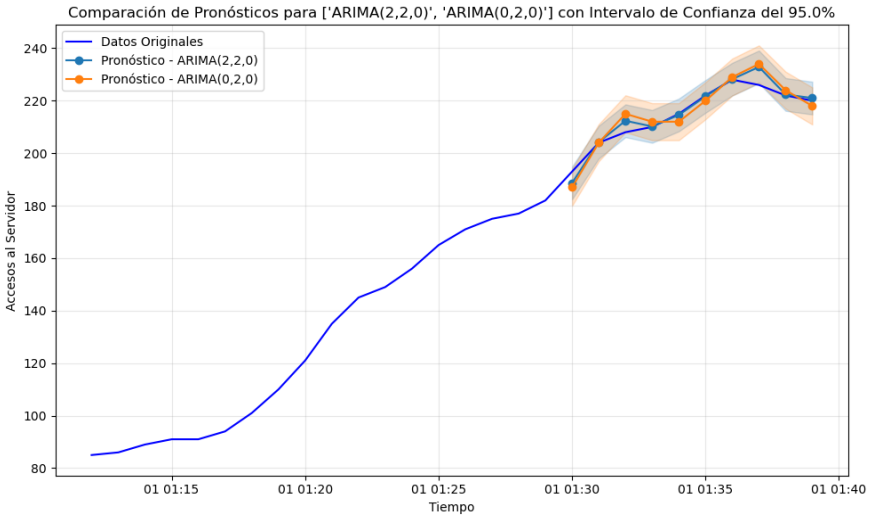

Figure 4: Prediction for ARIMA(2,2,0) and ARIMA(0,2,0) in the last 10 periods for the next period.

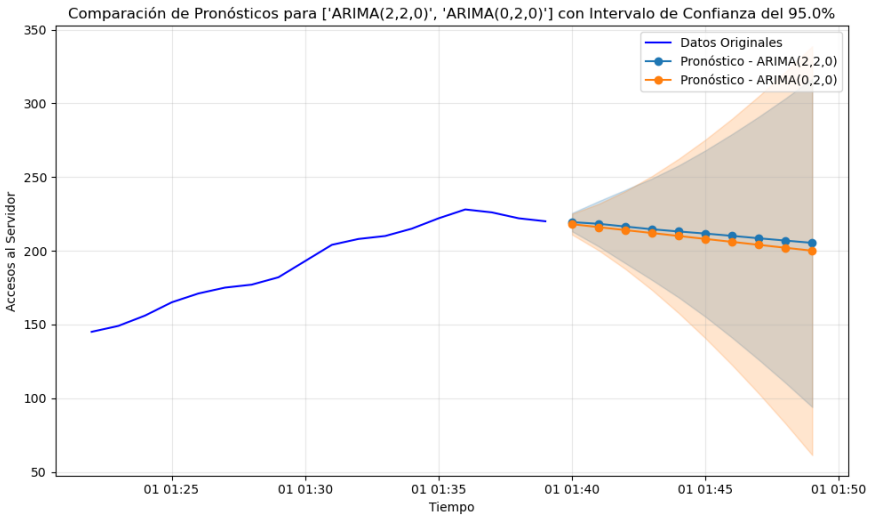

Figure 5: Prognosis ten periods into the future