Task 2

Alejandro Macías Pastor & Arturo Pérez Peralta

2024-03-18

1. Introduction and objectives

In our previous report we established the importance of the use of time series' techniques in the study of climate data. As a quick reminder, let us remember that weather scientists are particularly interested in models that are capable of accurately predicting the future evolution of temperature in order to better prepare for the secondary effects that this might have on the planet's climate. This is the focus of the present work, in which we will try to develop an appropriate model to forecast the future evolution of the average temperature. Once again we will apply tools from time series analyisis to study a dataset that tracks the monthly temperatures from a meteorological station at Boston's Logan Airport, Massachusetts, USA. This register started in 1936 and goes all the way to present times, and it can be found on the NOAA wepage. In particular we are interested in:

1. Study the ACF and PACF of the data in order to determine its structure.
2. Use the previous data to establish an appropriate ARIMA model.
3. Use bootstrap techniques to obtain confidence intervals for the model parameters.
4. Propose a vector model that uses the rest of the variables of the dataset to try and find new relationships in the data.

Now that we have a clear picture we can start by importing all the different packages and libraries that we are going to use, both in `R` and in `Python`:

```
library(reticulate) # to integrate Python within the Markdown
library(ggplot2)
library(ggfortify) # better plots
library(TSA)
```

```
##
## Attaching package: 'TSA'
```

```
## The following objects are masked from 'package:stats':
##
##     acf, arima
```

```
## The following object is masked from 'package:utils':
##
##     tar
```

```
library(tseries)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(forecast) # to deal with time series
```

```
## Registered S3 methods overwritten by 'forecast':
```

```
##   method                 from
##   autoplot.Arima         ggfortify
##   autoplot.acf           ggfortify
##   autoplot.ar            ggfortify
##   autoplot.bats          ggfortify
##   autoplot.decomposed.ts ggfortify
##   autoplot.ets           ggfortify
##   autoplot.forecast      ggfortify
##   autoplot.stl           ggfortify
##   autoplot.ts            ggfortify
##   fitted.Arima           TSA
##   fitted.ar              ggfortify
##   fortify.ts             ggfortify
##   plot.Arima             TSA
##   residuals.ar           ggfortify
```

```
library(Metrics) # to compare predictions with MSE
```

```
##
## Attaching package: 'Metrics'
```

```
## The following object is masked from 'package:forecast':
##
##     accuracy
```

```
library(blocklength) # to make bootstrap simulation
library(latticeExtra)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'latticeExtra'
```

```
## The following object is masked from 'package:ggplot2':
##
##     layer
```

```
set.seed(12345) # set seed for reproducibility
```

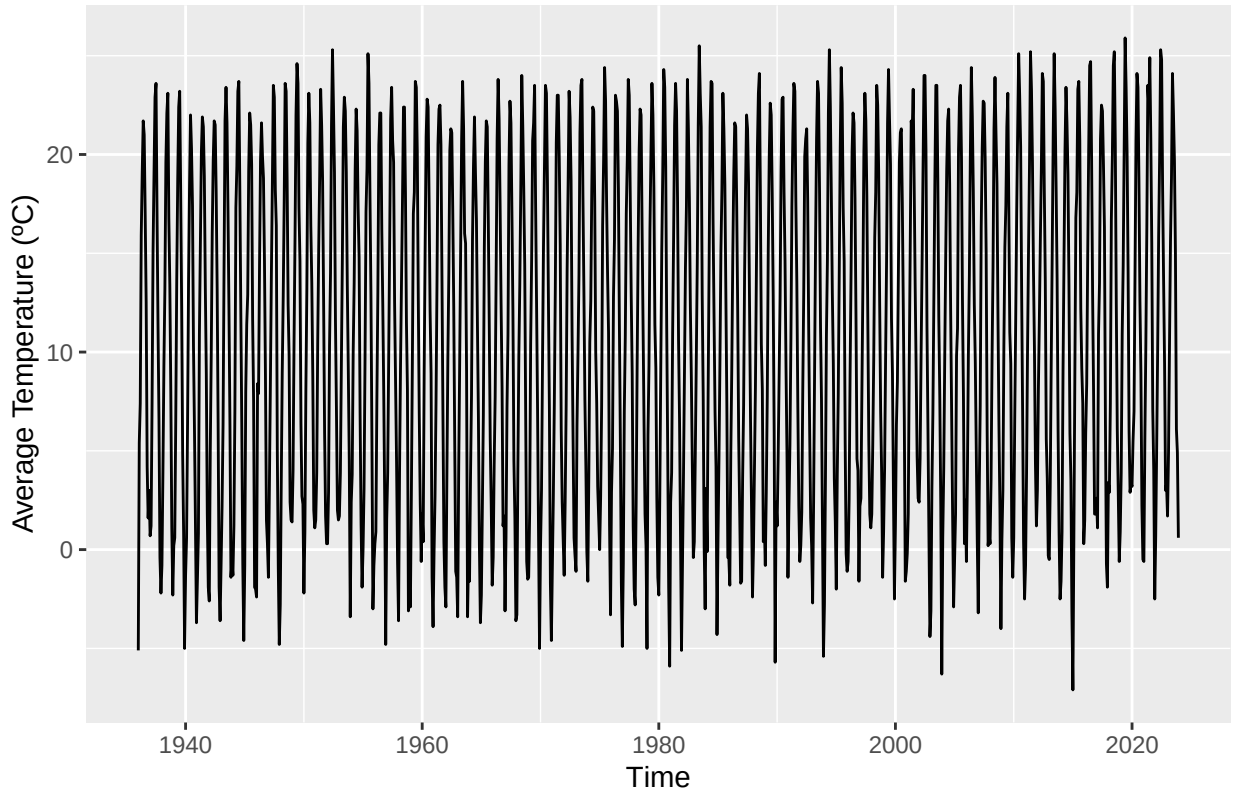
```
# Basic imports for any Python data science project
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings # supress unnecessary warnings in Python that clutter the screen
warnings.filterwarnings('ignore')

#The following two imports are used to make models based on
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.holtwinters import ExponentialSmoothing

#The following three imports are used to make models based on neural networks
from keras.models import Sequential
```

```
from keras.layers import LSTM, Dense
from sklearn.preprocessing import MinMaxScaler
```

```
#We import all the packages for the ARIMA analysis
import statsmodels.api as sm
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
from pmdarima.arima import auto_arima
```

We can now load our data:

```
#We load the dataset
data = read.csv("3610412.csv")

#We will only consider average temperature
data = data[,c("DATE","TAVG")]
#We remove the last observation, which is the only missing value
data = data[c(1:1056),]

#We transform the dataset into a
series = ts(data$TAVG, start=c(1936,1,1), freq=12) #Monthly frequency
autoplot(series, ylab='Average Temperature (ºC)') #Let ggfortify plot the series
```



2. Partition of the data and Exploratory Data Analysis

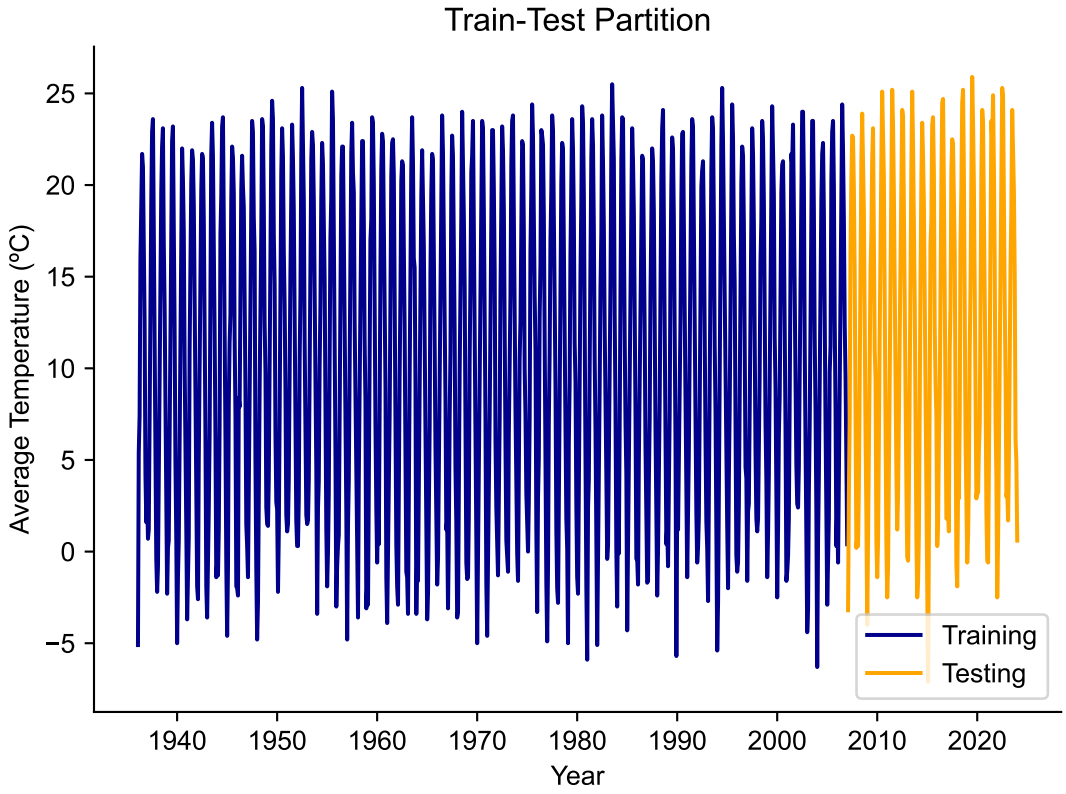
In order to better estimate the real performance of the models we are going to study, we will use the method of 1-fold cross-validation. Therefore, we will divide the dataset into two parts: one that will be used only for

training the models and another one that will be used only for testing their performance. The partition will be done in such a way that the first 70 years of the time series will be used for the training set and the last 17 years for testing, which translates into a roughly 80-20 train-test partition.

```
#First 70 years are used for training
data_train = data[c(1:852),]
series_train = ts(data_train$TAVG, start=c(1936,1,1), freq=12)

#Last 17 years are used for testing
data_test = data[c(853:1056),]
series_test = ts(data_test$TAVG, start=c(2001,1), freq=12)
```

We can visualize this split using `Python`, which requires that we import our data to the `Python` interpreter:

```
#We import the data from the R console
data = r.data
#We transform the 'DATE' column to a datetime format
data["DATE"] = pd.to_datetime(data["DATE"])
#We use the 'DATE' column as an index
data = data.set_index("DATE")
#We indicate that we are working with monthly data
data.index.freq = 'MS'

# We create a train-test partition like we did in R
# We create a train-test partition like we did in R
train = data[:-204] # First 70 years are used in the training set
test = data[-204:] # Last 17 years are used in the testing set

plt.clf() # Clear all previous plots
plt.plot(train,label="Training", color = 'darkblue') # Plot training set
plt.plot(test,label="Testing", color = 'orange') # Plot testing set
plt.xlabel("Year") #Title and axis levels
plt.ylabel("Average Temperature (ºC)")
plt.title("Train-Test Partition")
plt.legend() #legend
sns.set_theme(style = 'white') #Fancy looks for the graphic
sns.despine()
plt.show()
```



We are now ready to begin our analysis.

3. ARIMA analysis

We are going to fit a seasonal ARIMA model, that is we assume that the data follows the following relationship:

$$\Phi_P(B^s)\phi_p(B)\nabla_s^D\nabla^d z_t = \Theta_Q(B^s)\theta_q(B)a_t$$

In order to do this we need to:

1. Stabilize the mean and the variance of our model.
2. Determine the order of the seasonality, s.
3. Determine the order of the seasonal ARIMA, (P, D, Q).
4. Determine the order of the regular ARIMA, (p, d, q).

The first step we should take is to stabilize the mean and the variance of our model. We are going to start with the variance. In order to do this we should check whether there is a linear relationship between the mean and variance and, if so, apply the proper box-cox transformation:

```
#We plot yearly mean and std
years = 70

#We initialize the vector of means and stds
set_mean = np.zeros(years)
set_std = np.zeros(years)

#We iterate over all years and compute the mean and std
```

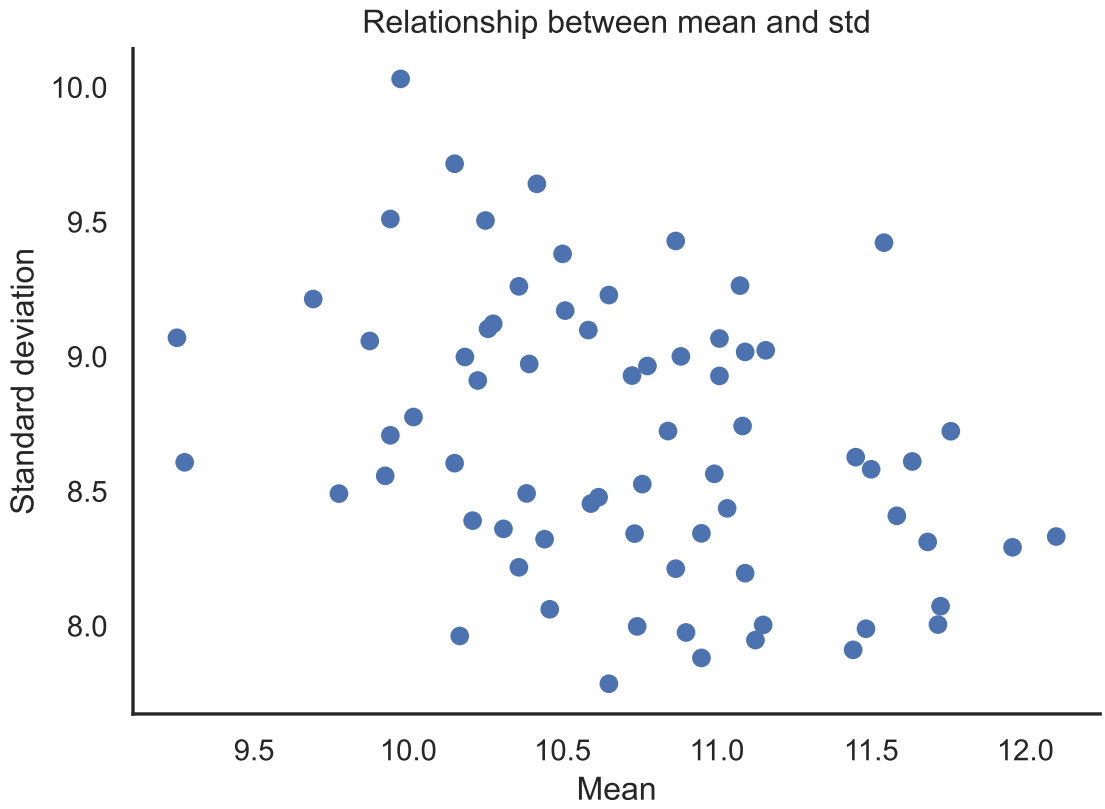
```python
for i in range(years):
    set_mean[i] = np.mean(train.iloc[12*i:12*(i+1)])
    set_std[i] = np.std(train.iloc[12*i:12*(i+1)])
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:3462: FutureWarning: In a future
##   return mean(axis=axis, dtype=dtype, out=out, **kwargs)
```

```python
#We plot our results and check whether there is a linear relationship
plt.clf()
plt.scatter(set_mean, set_std)
plt.title('Relationship between mean and std')
plt.xlabel('Mean')
plt.ylabel('Standard deviation')
sns.set_theme(style = 'white')
sns.despine()
plt.show()
```



There is no clear relationship between the mean and the standard deviation. However, if one would insist on making a Box-Cox transformation one could use the following code:

```
#We define the boxcox transformation
def boxcox(x, alfa):
  if alfa == 1:
    return(np.log(x))
  else:
    return (x**(1-alfa) - 1)/(1 - alfa)

#We import the linear model and make our estimation for our data
from sklearn.linear_model import LinearRegression
model = LinearRegression().fit(np.log(set_mean).reshape(-1,1), np.log(set_std))
model.coef_
```

```
## array([-0.36392248])
```

```
#We could transform our data
new_dat = boxcox(train + np.abs(np.min(train)) + 0.01, model.coef_)
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\numpy\core\fromnumeric.py:84: FutureWarning: In a future v
##   return reduction(axis=axis, out=out, **passkwargs)
```

However, we have found that this transformation has no impact in our analysis.

We now need to stabilize the mean. In the previous report we saw that the average temperature goes up as a result of climate change. Therefore, it stands to reason that we do need to differentiate the series, probably just one time will be enough. We can check whether this differentiation is necessary by means of the Dickey-Fuller test:

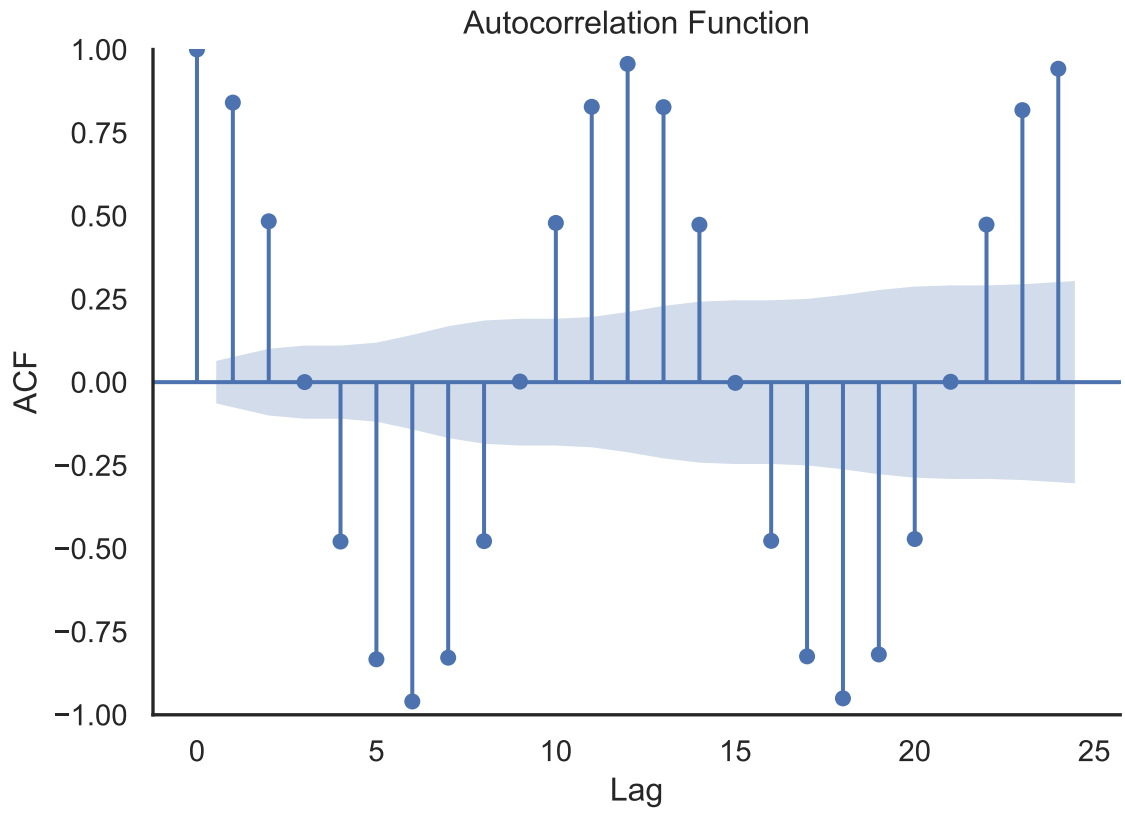
```
adf.test(series_train, k=0)
```

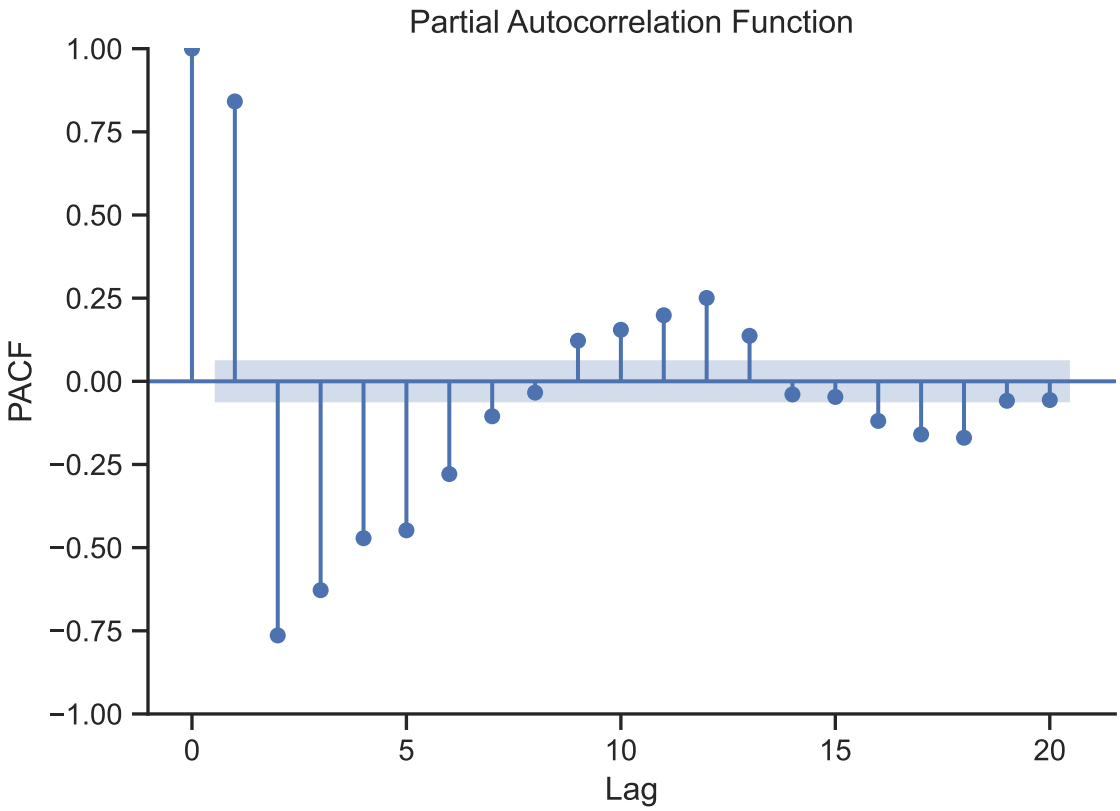
```
##
##  Augmented Dickey-Fuller Test
##
## data:  series_train
## Dickey-Fuller = -8.5641, Lag order = 0, p-value = 0.01
## alternative hypothesis: stationary
```

The p-value of the test indicates that the null hypothesis should be rejected, and so the series should actually be considered stationary. Although this contradicts the results of the Holt-Winters model developed in the previous report, in order to take into account the Dickey-Fuller test we are going to consider models with both $d = 1$ and $d = 0$ and choose the best among them.

In any case, we can now proceed with our analysis. Let us now take a look at the autocorrelation and partial autocorrelation functions:

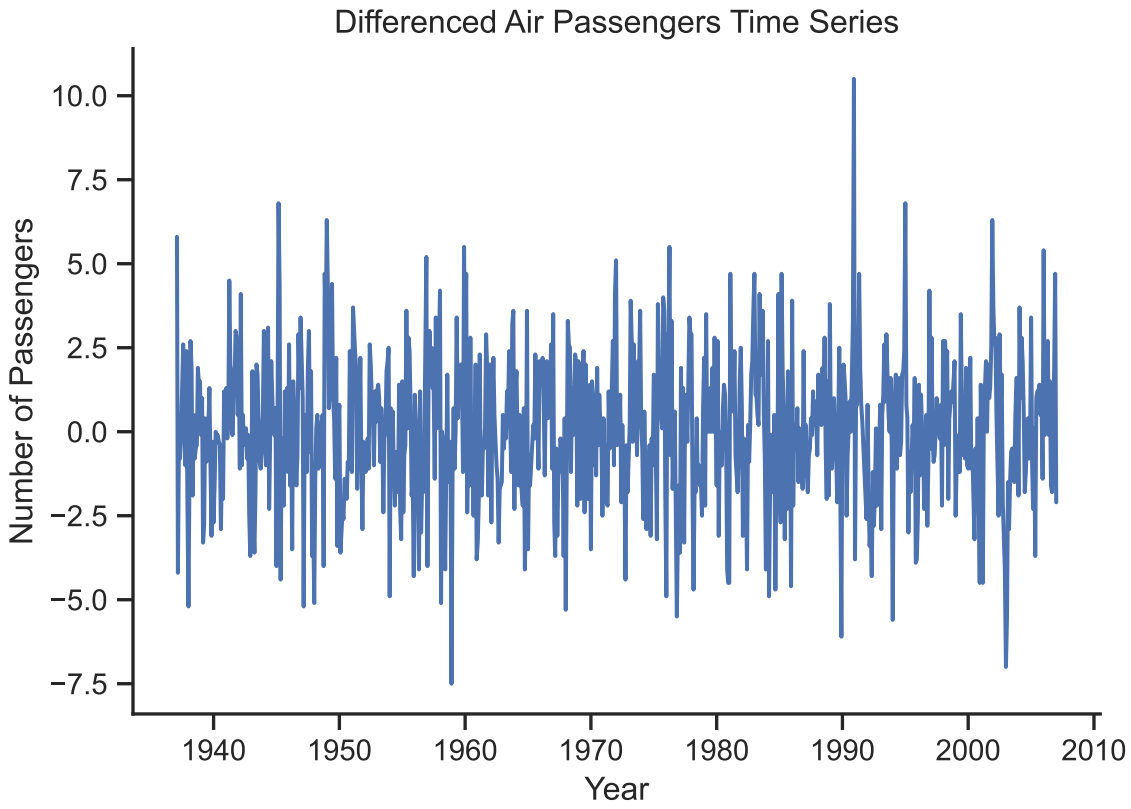
```
plt.clf()
plot_acf(train, lags=24) #Plot ACF
plt.title('Autocorrelation Function') #Title and axis labels
plt.xlabel('Lag')
plt.ylabel('ACF')
sns.set_theme(style = 'ticks') #Fancy graphics
sns.despine()
plt.show()
```




Partial Autocorrelation Function
PACF
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
Lag

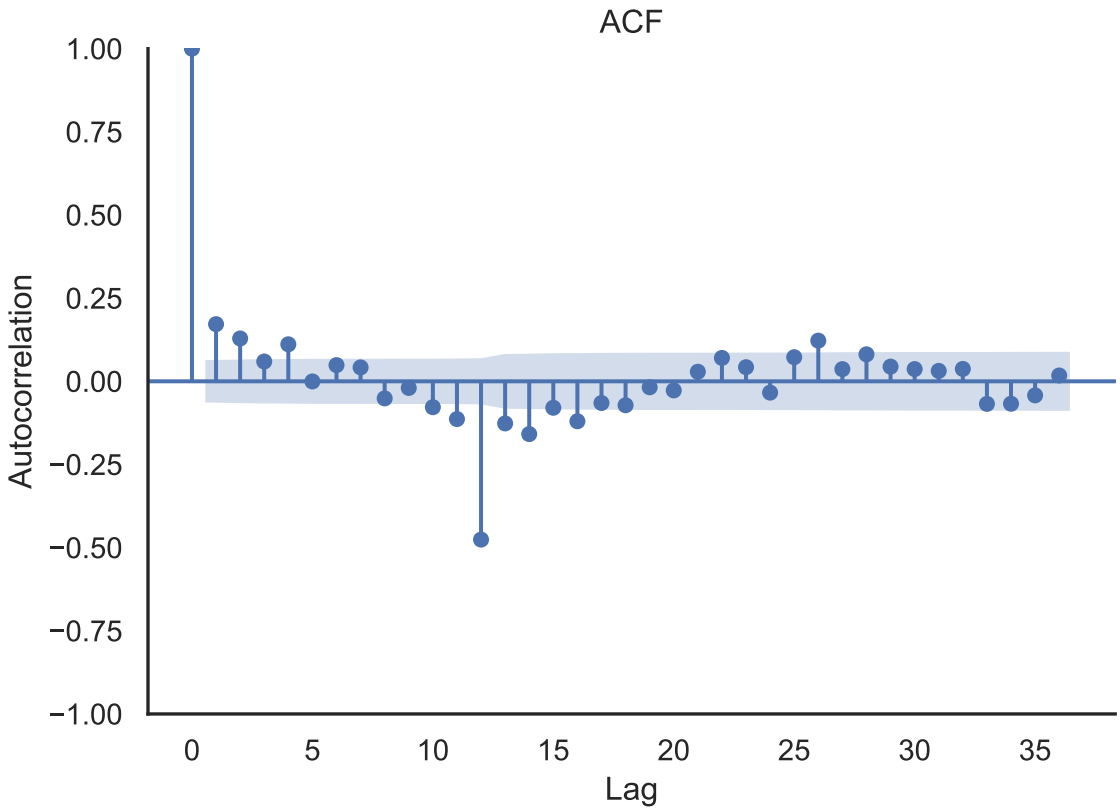
We observe seasonality with a period of $s = 12$, which is to be expected due to the monthly nature of our data. Therefore, it is reasonable to differentiate with respect to 12 lags:

```
#We differentiate the seasonal period
diff_ts = train.diff(12).dropna()
#We plot our data
plt.clf()
plt.plot(diff_ts)
plt.xlabel('Year')
plt.ylabel('Number of Passengers')
plt.title('Differenced Air Passengers Time Series')
sns.set_theme(style = 'white')
sns.despine()
plt.show()
```

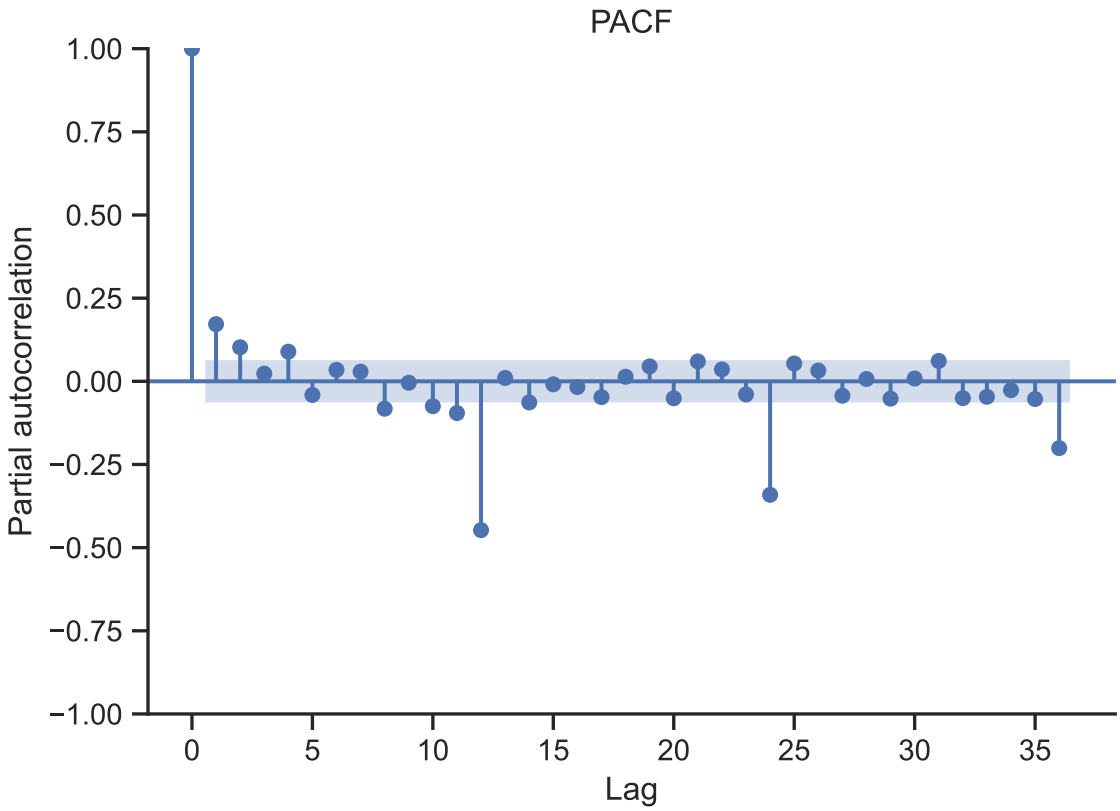


We should remark that we have also tried differentiating the data by a set of multiples of 12 (24, 36,...). However, we could not improve the results we obtained from simply differentiating the data once. The resulting ACF and PACF structure is the following:

```
plot_acf(diff_ts, lags=36) #We plot the acf
plt.xlabel('Lag')
plt.ylabel('Autocorrelation')
plt.title('ACF')
sns.set_theme(style = 'ticks')

sns.despine()
plt.show()
```




```
plot_pacf(diff_ts, lags=36, method='ywm') #We plot the pacf
plt.xlabel('Lag')
plt.ylabel('Partial autocorrelation')
plt.title('PACF')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```



From these graphics we can see that:

1. The seasonal MA structure seems to be of order 4 or 5 (the 5×12 lag is really close to the band so it may be zero).
2. The seasonal AR structure seems to be of order 1 (the 2×12 lag ism again, too close to the band so as to merit much consideration).
3. The AR structure seems to be of order 3 or 4.
4. The MA structure seems to be of order 3, 4 or 5.

These conclusions seem to be confirmed by the `auto_arima` model which outputs similar orders for the seasonal and ARIMA structures:

```
#We run the auto arima process
model = auto_arima(train, seasonal=True,  start_p=0, d=1, start_q=0, max_p=5,
max_d=5, max_q=5, start_P=0, D=1, start_Q=0, max_P=5, max_D=5,max_Q=5, m=12, suppress_warnings=True, st

#We print the model summary
print(model.summary())
```

```
##                                      SARIMAX Results
## ==========================================================================================
## Dep. Variable:                                   y   No. Observations:                  852
## Model:             SARIMAX(5, 1, 0)x(5, 1, 0, 12)   Log Likelihood               -1658.186
## Date:                            lu., 18 mar. 2024   AIC                            3338.373
## Time:                                    18:59:32   BIC                            3390.427
## Sample:                                02-01-1936   HQIC                           3358.325
##                                      - 01-01-2007
## Covariance Type:                              opg
```

```
## ==============================================================================
##                  coef    std err          z      P>|z|      [0.025      0.975]
## ------------------------------------------------------------------------------
## ar.L1         -0.7062      0.030    -23.895      0.000      -0.764      -0.648
## ar.L2         -0.5136      0.039    -13.220      0.000      -0.590      -0.437
## ar.L3         -0.4279      0.042    -10.296      0.000      -0.509      -0.346
## ar.L4         -0.2748      0.043     -6.395      0.000      -0.359      -0.191
## ar.L5         -0.1854      0.038     -4.911      0.000      -0.259      -0.111
## ar.S.L12      -0.8224      0.031    -26.412      0.000      -0.883      -0.761
## ar.S.L24      -0.7177      0.041    -17.500      0.000      -0.798      -0.637
## ar.S.L36      -0.5229      0.040    -13.029      0.000      -0.602      -0.444
## ar.S.L48      -0.3262      0.040     -8.215      0.000      -0.404      -0.248
## ar.S.L60      -0.1599      0.032     -5.031      0.000      -0.222      -0.098
## sigma2         3.0010      0.128     23.522      0.000       2.751       3.251
## ==============================================================================
## Ljung-Box (L1) (Q):                   0.72   Jarque-Bera (JB):                34.17
## Prob(Q):                              0.40   Prob(JB):                         0.00
## Heteroskedasticity (H):               0.89   Skew:                            -0.14
## Prob(H) (two-sided):                  0.35   Kurtosis:                         3.95
## ==============================================================================
##
## Warnings:
## [1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

Therefore, we are going to try many different models changing the orders. We will start with the simplest model and work our way up. We will try to have residuals without structure (with null ACF and PACF, that is, they remain within the bands) and among those models with the appropriate residuals we will select the one that minimizes the AIC and MSE. Therefore, we are going to try the following models:

1. ARIMA: $(3, 1, 2)$, seasonal: $(4, 1, 1)$, $s = 12$.
2. ARIMA: $(4, 1, 2)$, seasonal: $(4, 1, 1)$, $s = 12$.
3. ARIMA: $(5, 1, 2)$, seasonal: $(4, 1, 1)$, $s = 12$.
4. ARIMA: $(5, 1, 2)$, seasonal: $(5, 1, 1)$, $s = 12$.
5. ARIMA: $(5, 1, 3)$, seasonal: $(4, 1, 1)$, $s = 12$.
6. ARIMA: $(4, 0, 2)$, seasonal: $(4, 1, 1)$, $s = 12$.
7. ARIMA: $(5, 0, 2)$, seasonal: $(4, 1, 1)$, $s = 12$.
8. ARIMA: $(5, 0, 2)$, seasonal: $(5, 1, 1)$, $s = 12$.
9. ARIMA: $(5, 0, 3)$, seasonal: $(4, 1, 1)$, $s = 12$.

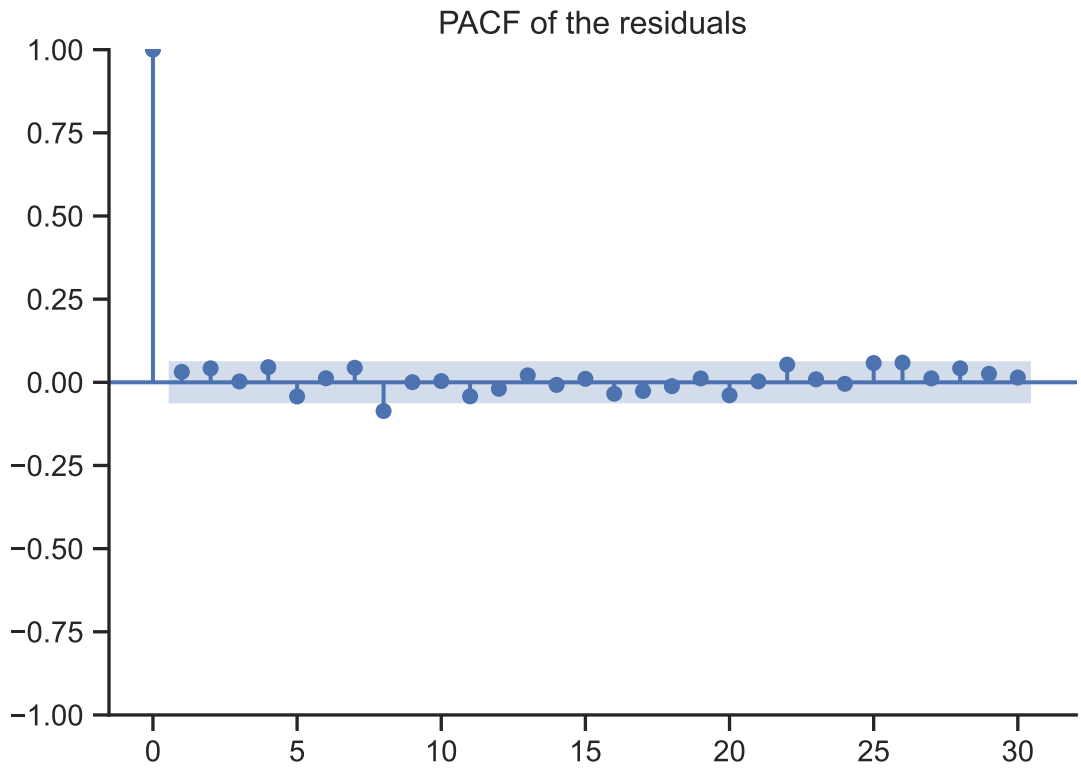
We start with the first model and build up from there:

```
#We define our model
model1 = ARIMA(train, order=(3, 1, 2), seasonal_order=(4, 1, 1, 12))
#We fit our model
model1_fit = model1.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non
##   warn('Non-invertible starting MA parameters found.'
```

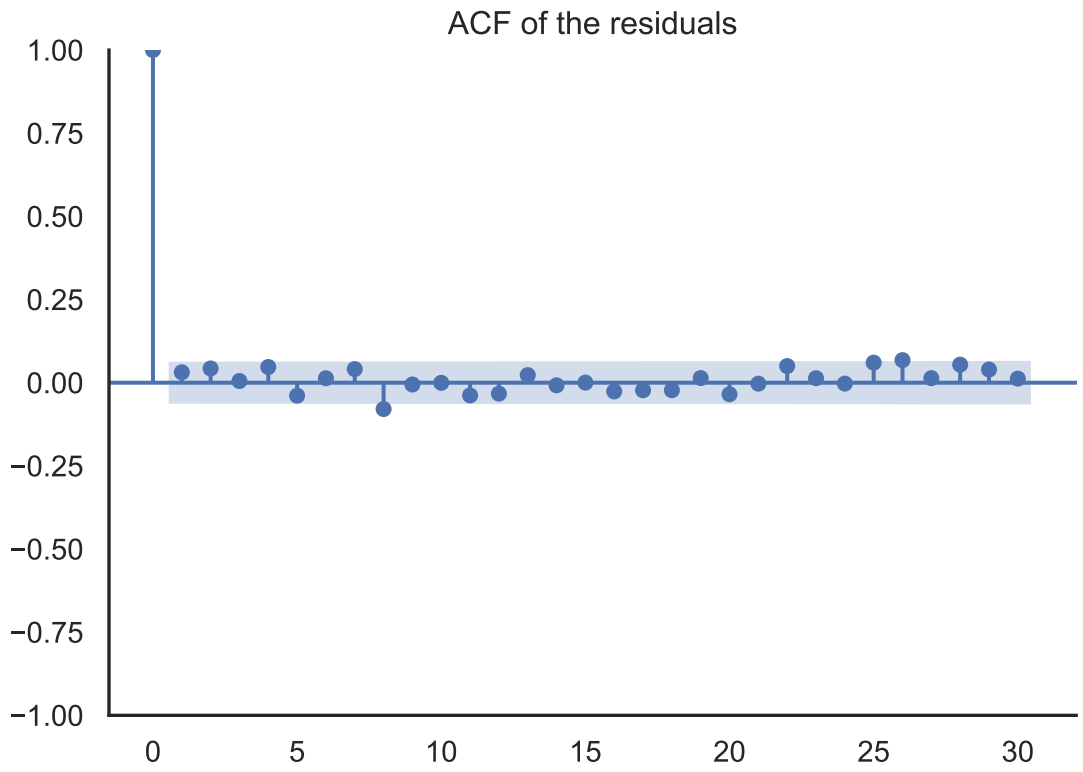
```
#We make our predictions
predictions1 = model1_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
#We compute the residuals
residuals1 = pd.DataFrame(model1_fit.resid)
```

```
#We plot the residuals
residuals1.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

#We plot the density of the residuals
residuals1.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

#We plot the ACF
plot_acf(residuals1)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

#We plot the PACF
plot_pacf(residuals1)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```



```
print(residuals1.describe())
```

```
##                0
## count  852.000000
## mean    -0.069930
## std      1.710668
## min     -9.999961
## 25%     -1.026066
## 50%     -0.057948
## 75%      0.934939
## max     10.499991
```

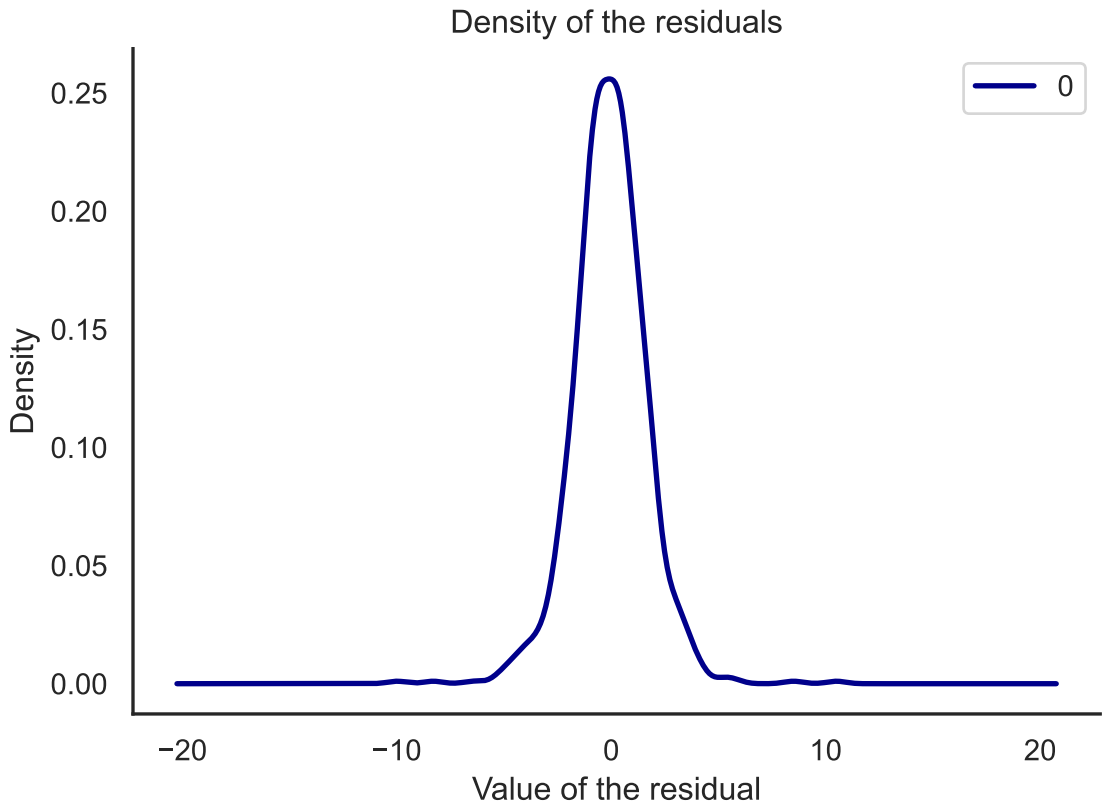


We can already see the residual structure we were looking for: all of the autocorrelations and partial autocorrelations of the residuals lie within the band and, although this result does not mean that they *really* are zero (specially for those with lags closer to zero) it is a promising result. We are going to try with the rest of the models and check whether this structure is preserved:

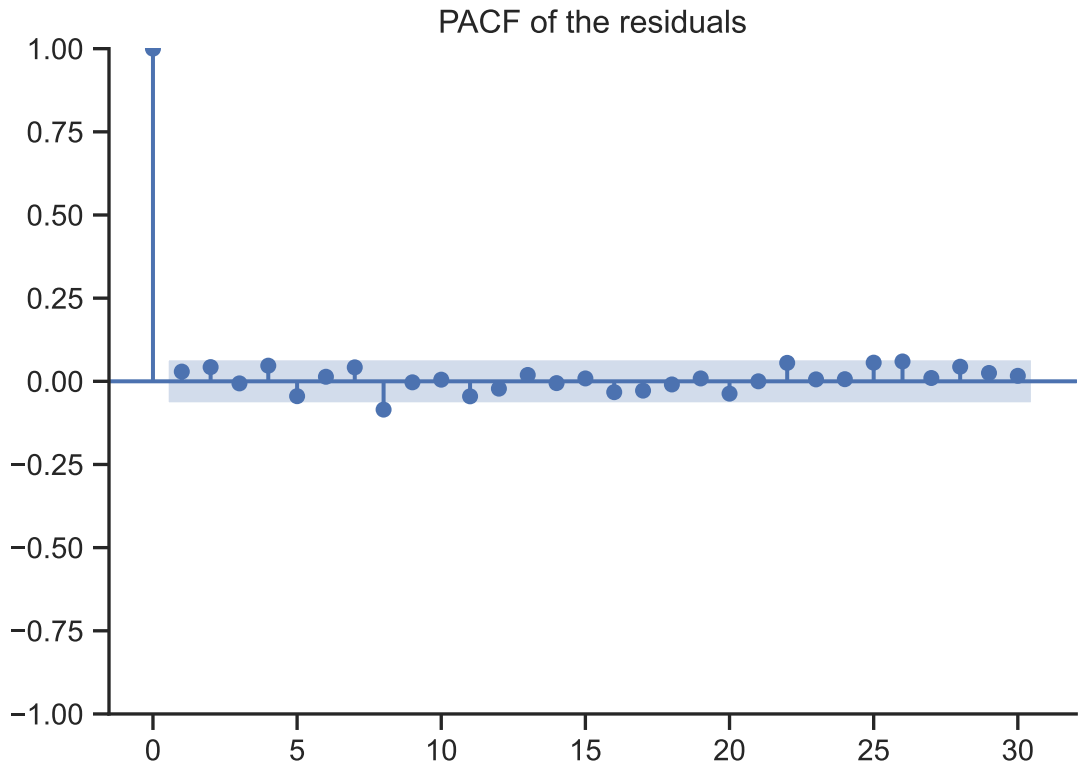
```
#We do not further comment because it is identical to model1
model2 = ARIMA(train, order=(4, 1, 2), seasonal_order=(4, 1, 1, 12))
model2_fit = model2.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Non
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Non
##   warn('Non-invertible starting MA parameters found.'
```

```
predictions2 = model2_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals2 = pd.DataFrame(model2_fit.resid)
```

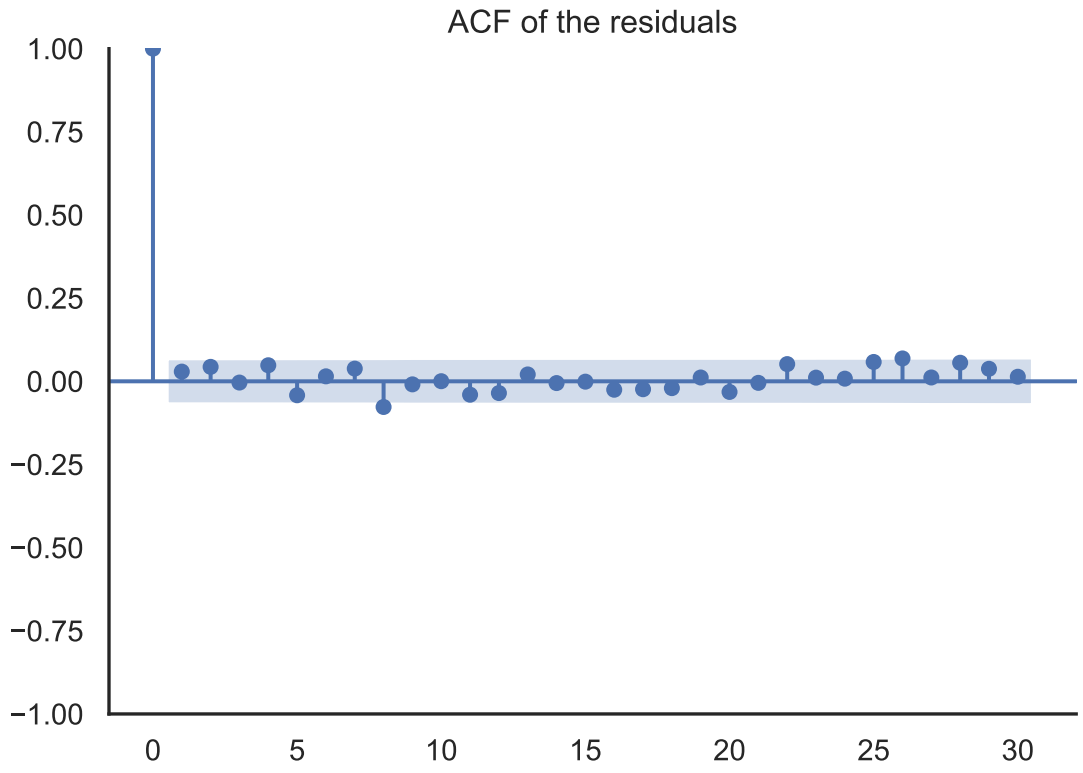



```
residuals2.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals2.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals2)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals2)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals2.describe())
```

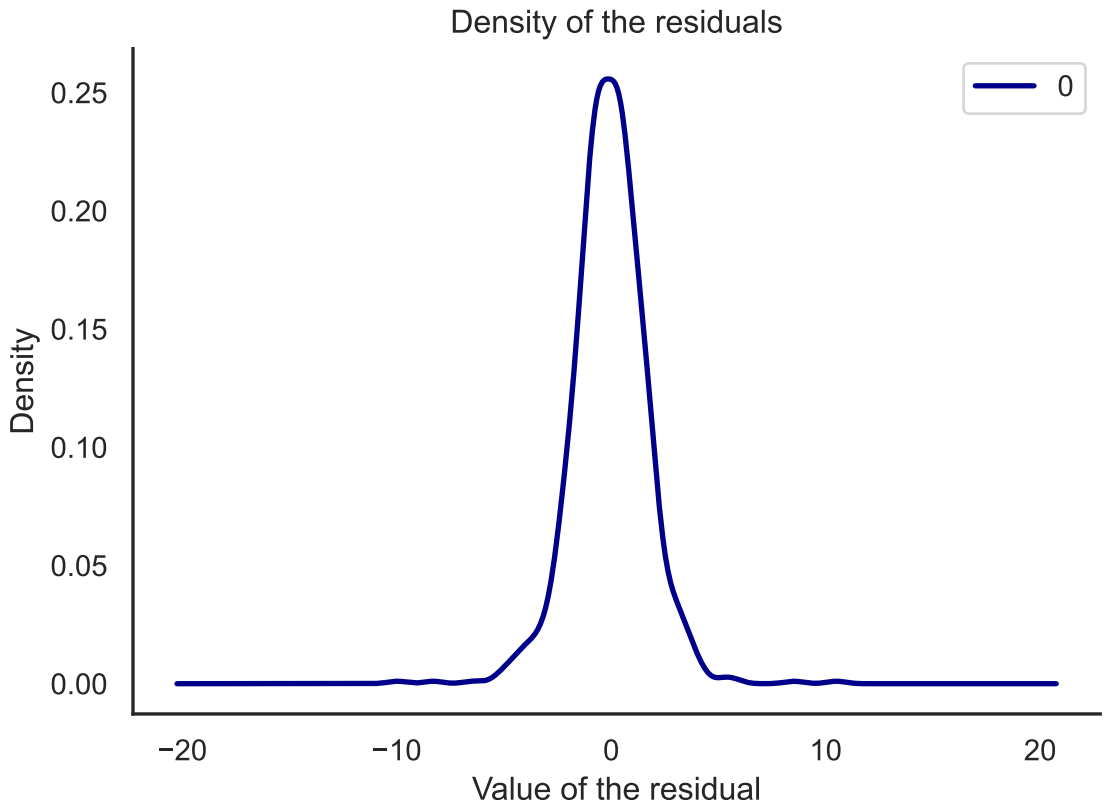
```
##                0
## count  852.000000
## mean    -0.069068
## std      1.710232
## min     -9.999962
## 25%     -1.018141
## 50%     -0.070422
## 75%      0.938569
## max     10.499991
```



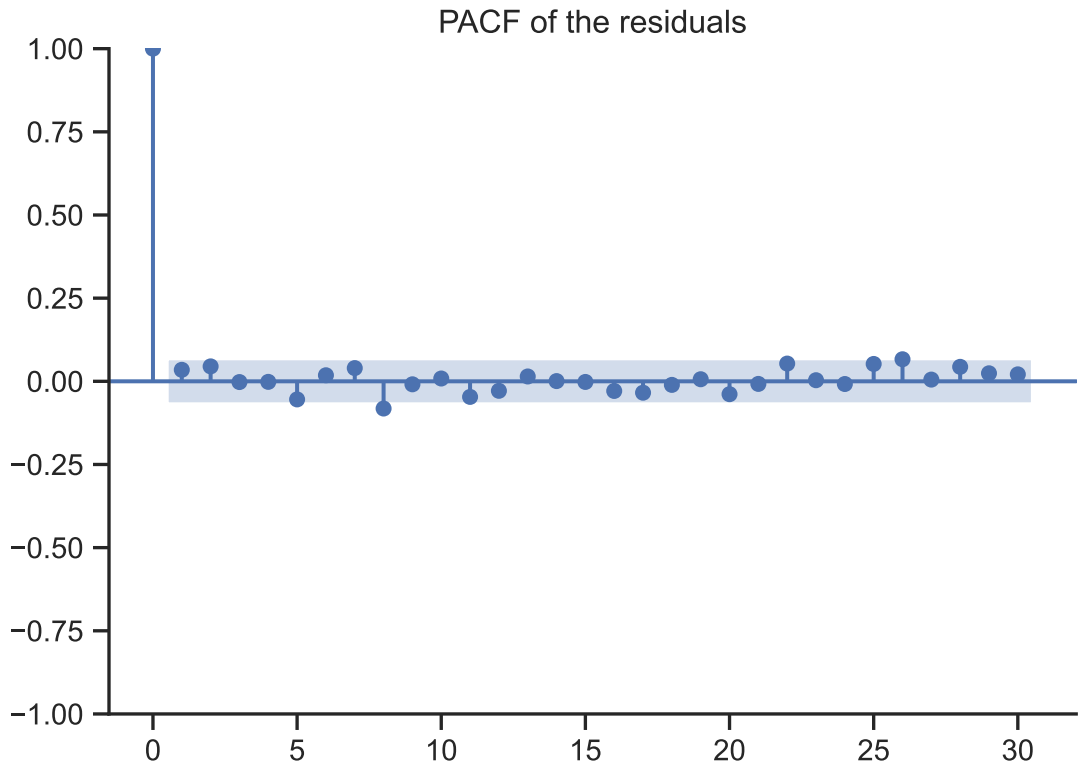

```
model3 = ARIMA(train, order=(5, 1, 2), seasonal_order=(4, 1, 1, 12))
model3_fit = model3.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Nor
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Nor
##   warn('Non-invertible starting MA parameters found.'
```

```
predictions3 = model3_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals3 = pd.DataFrame(model3_fit.resid)
```

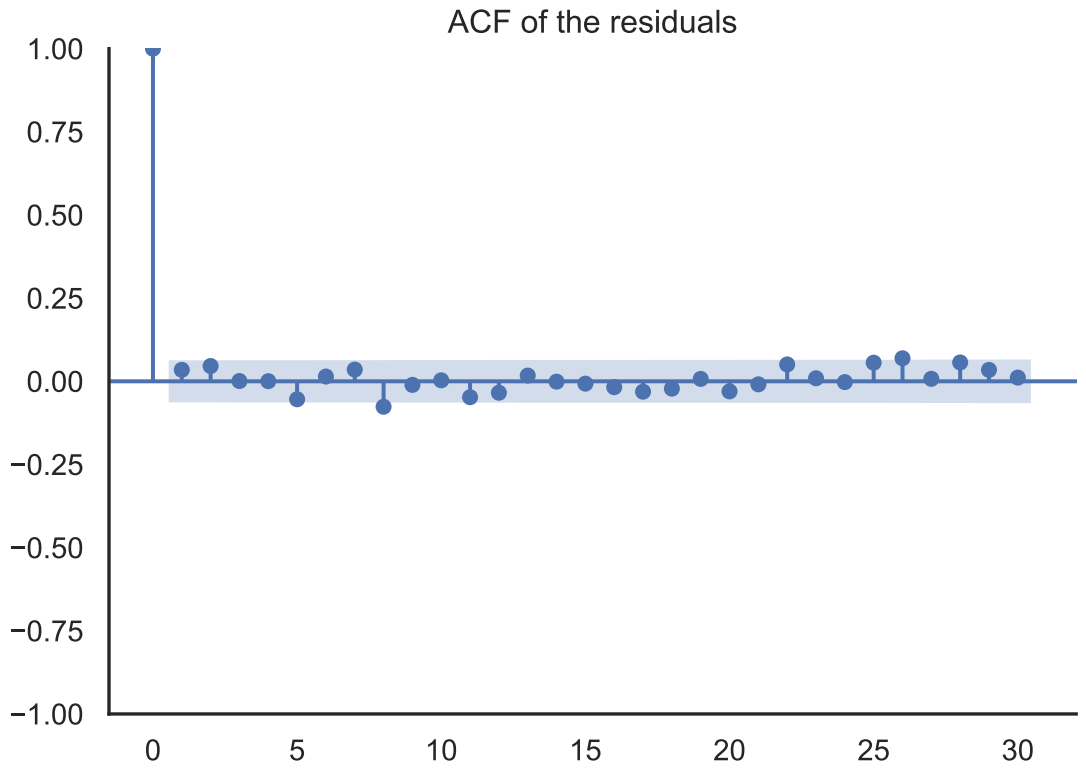



```
residuals3.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals3.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals3)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals3)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```

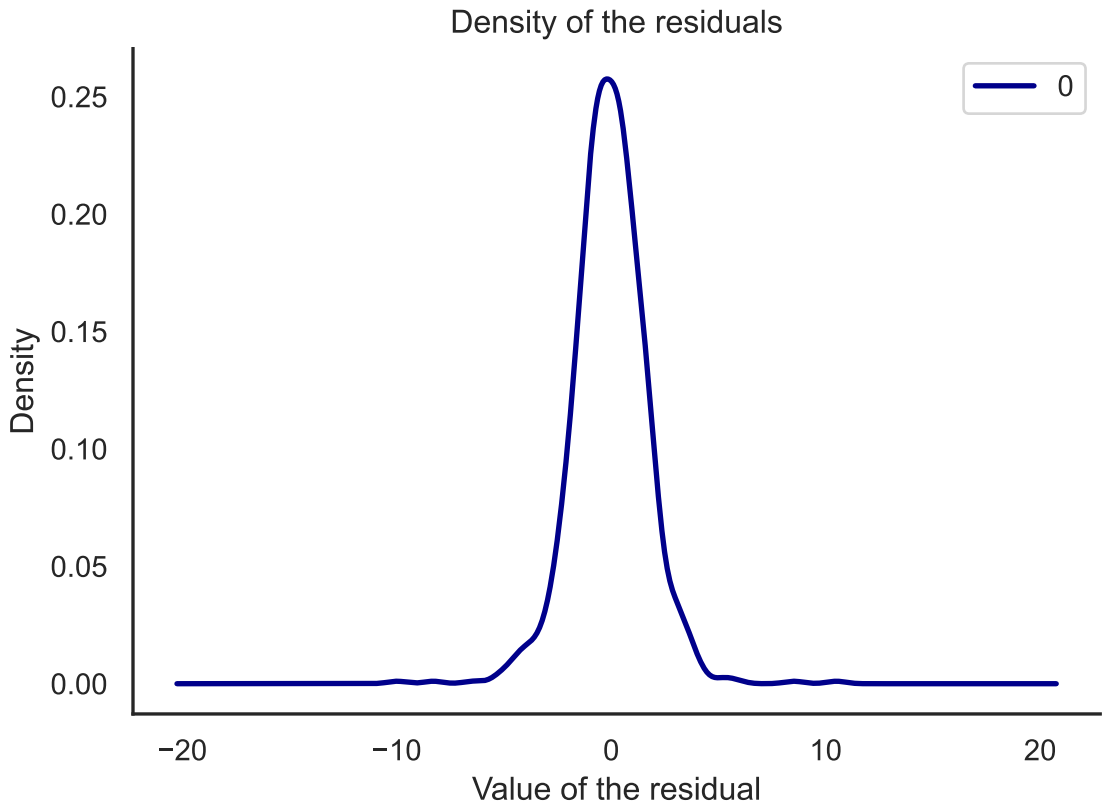



```
print(residuals3.describe())
```

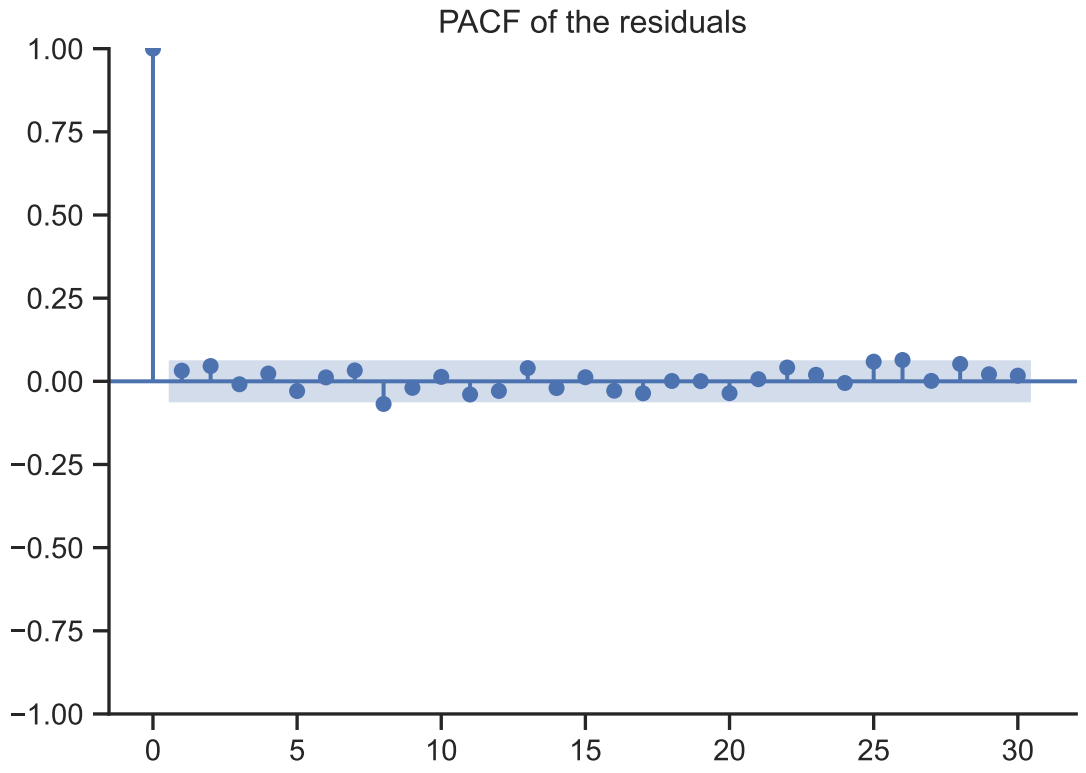
```
##                 0
## count  852.000000
## mean    -0.063977
## std      1.706612
## min     -9.999957
## 25%     -1.007930
## 50%     -0.061349
## 75%      0.947227
## max     10.499991
```




```
model4 = ARIMA(train, order=(5, 1, 3), seasonal_order=(4, 1, 1, 12))
model4_fit = model4.fit()
predictions4 = model4_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals4 = pd.DataFrame(model4_fit.resid)
```

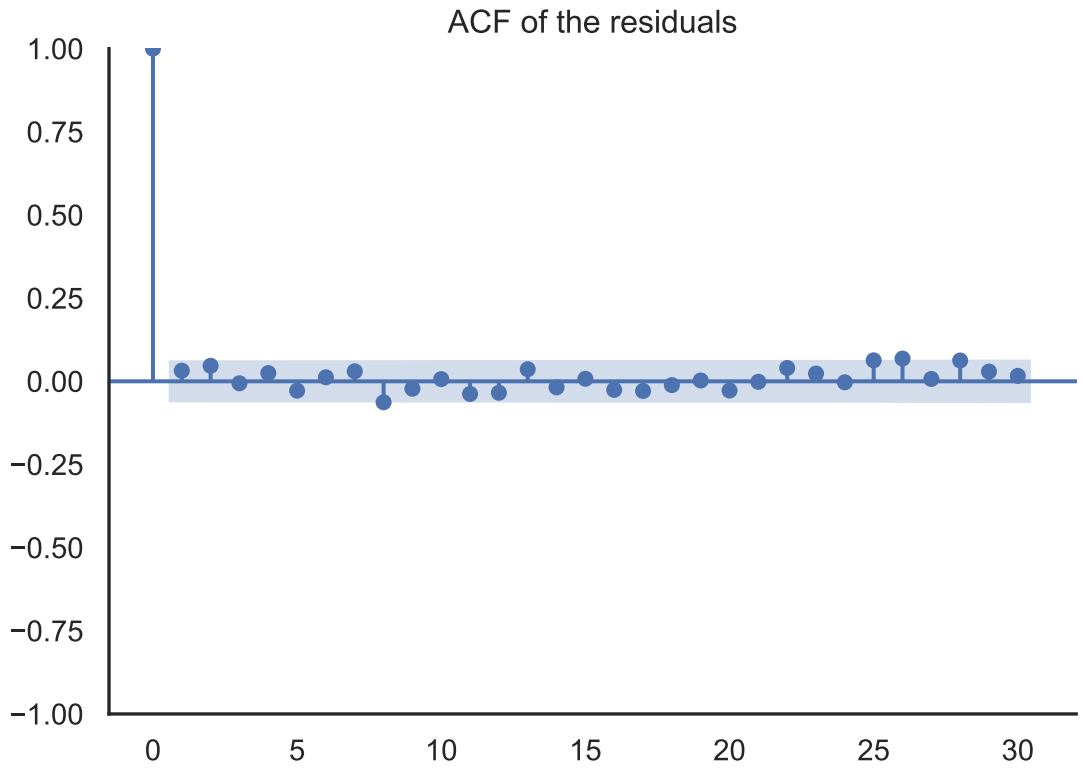



```
residuals4.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals4.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals4)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals4)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals4.describe())
```

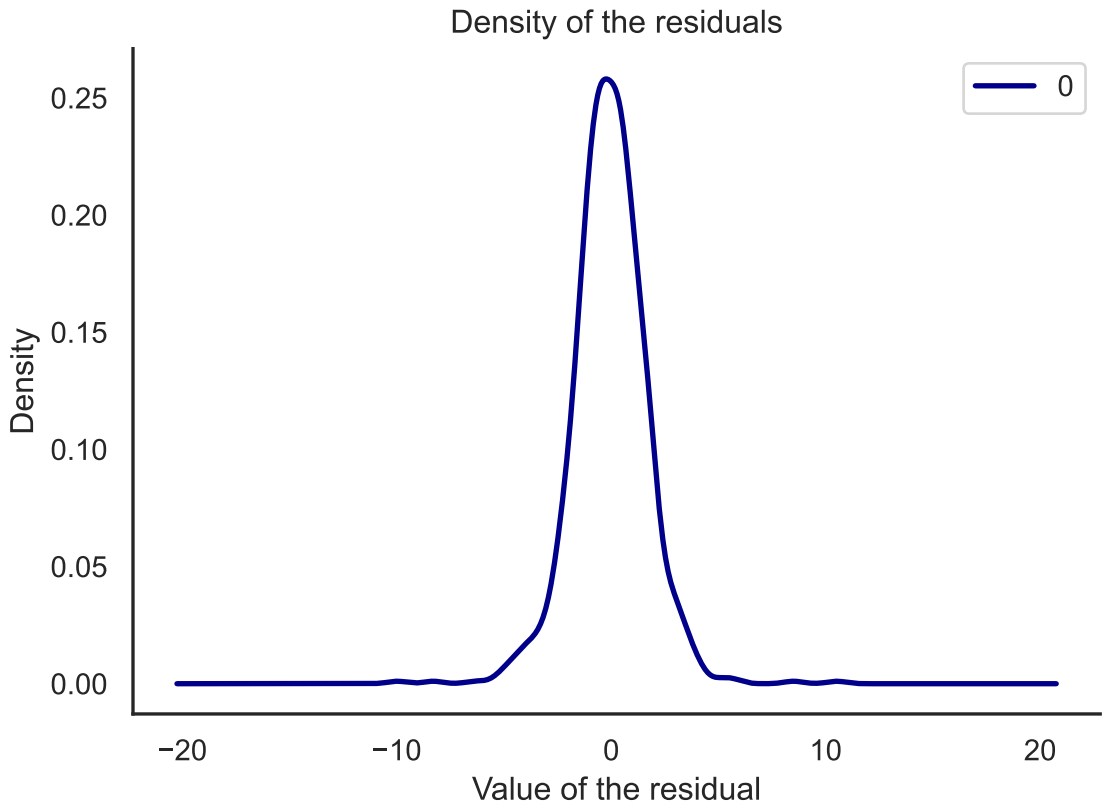
```
##                0
## count  852.000000
## mean    -0.068622
## std      1.706584
## min     -9.999958
## 25%     -1.018466
## 50%     -0.080526
## 75%      0.922685
## max     10.499991
```



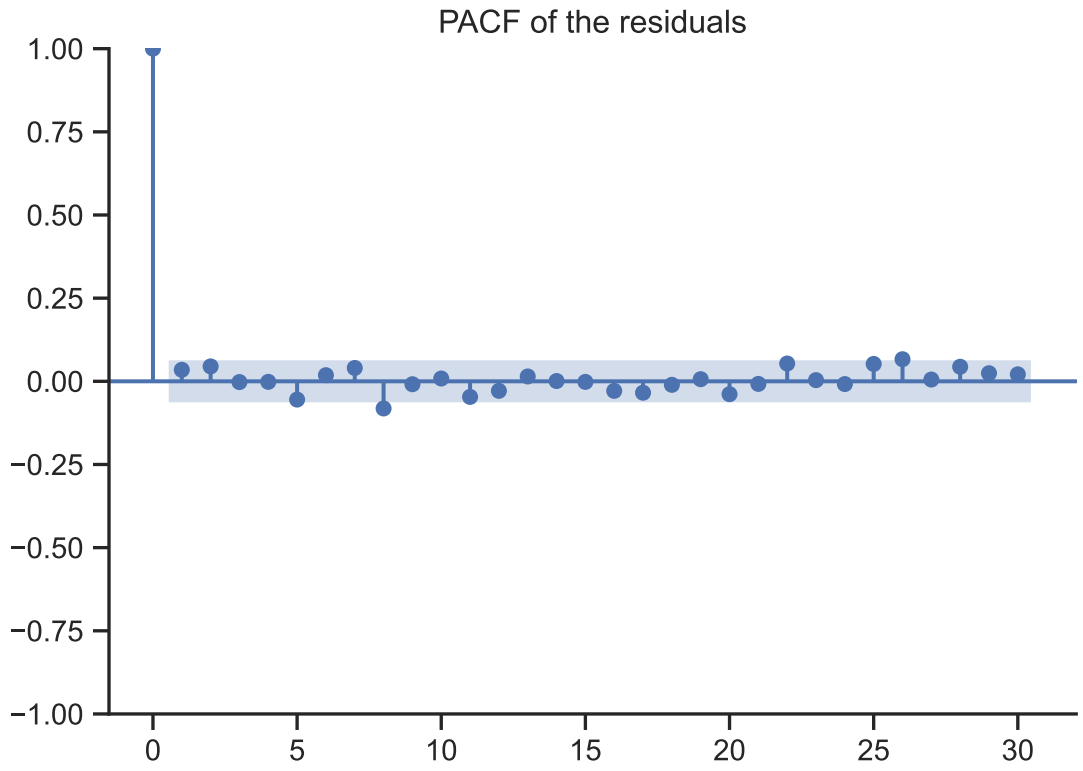

```
model5 = ARIMA(train, order=(5, 1, 2), seasonal_order=(5, 1, 1, 12))
model5_fit = model5.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Nor
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Nor
##   warn('Non-invertible starting MA parameters found.'
```

```
predictions5 = model5_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals5 = pd.DataFrame(model5_fit.resid)
```

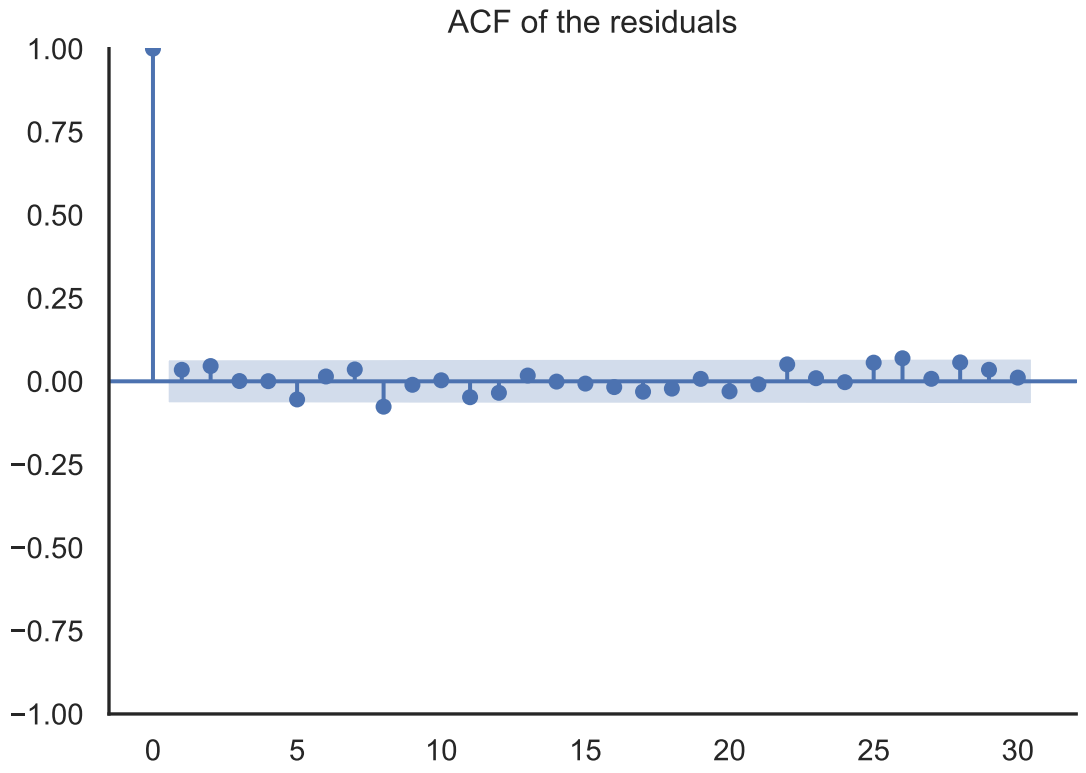



```
residuals5.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals5.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals5)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals5)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals5.describe())
```

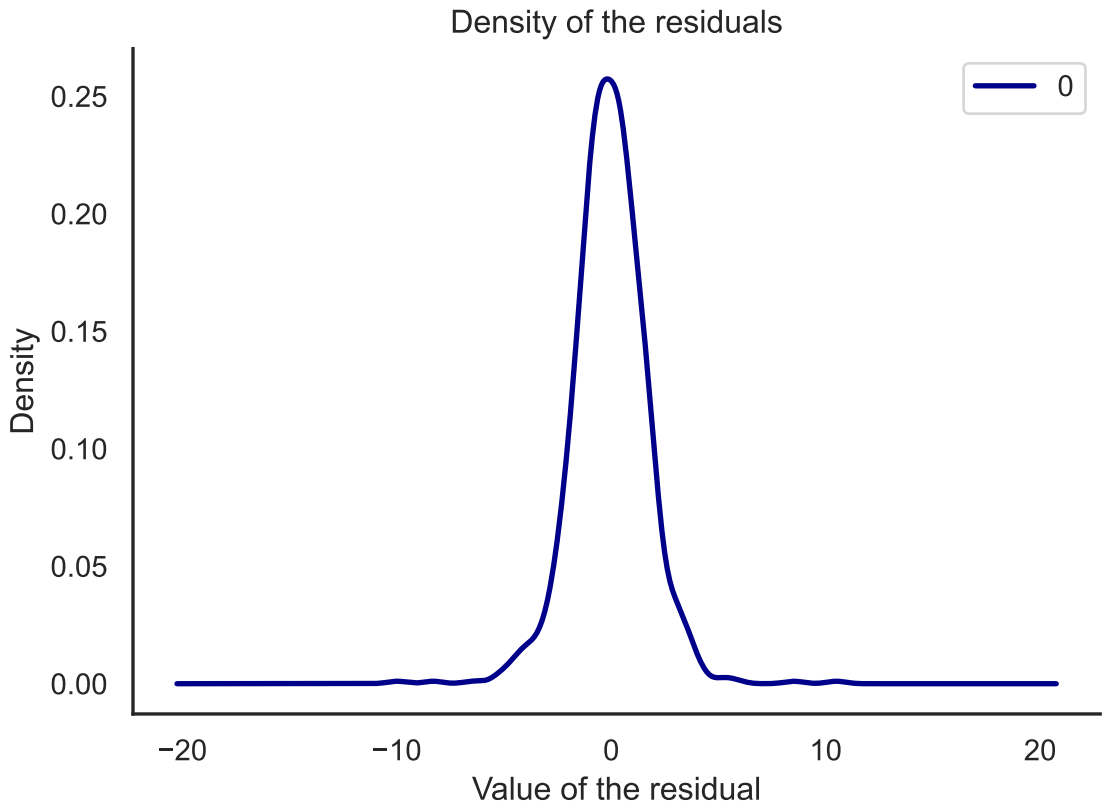
```
##                 0
## count  852.000000
## mean    -0.064021
## std      1.706566
## min     -9.999957
## 25%     -1.000829
## 50%     -0.066985
## 75%      0.944855
## max     10.499991
```



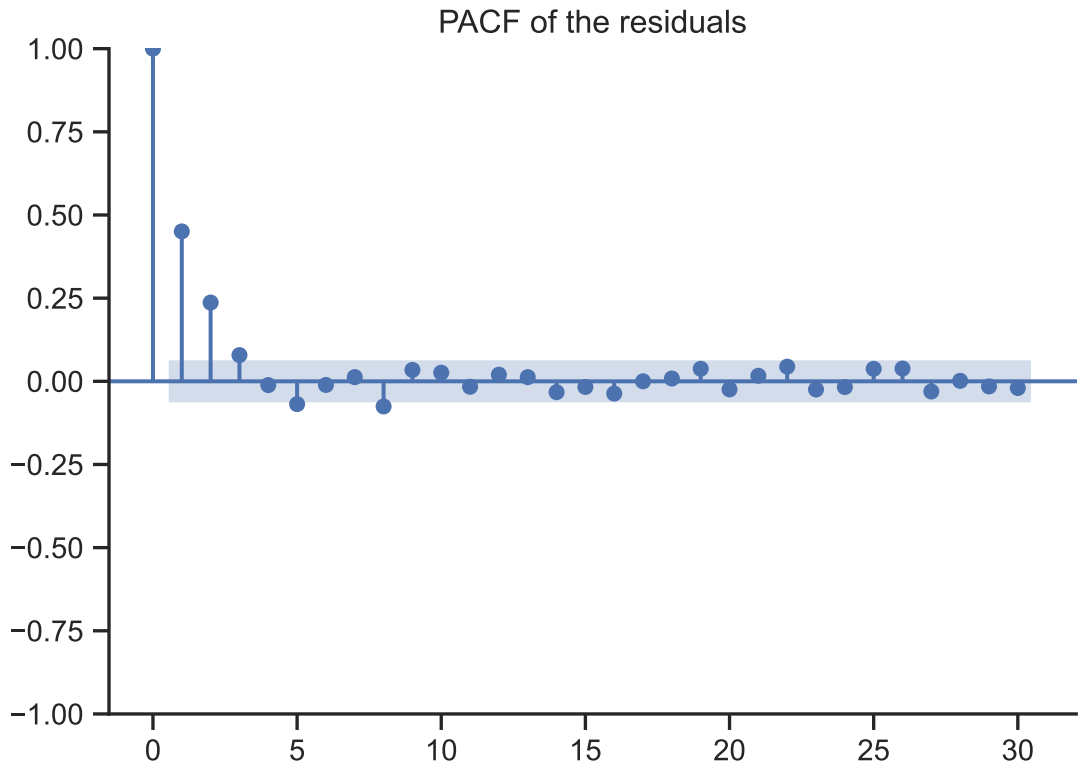

```
model6 = ARIMA(train, order=(4, 0, 2), seasonal_order=(4, 1, 1, 12))
model6_fit = model6.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Nor
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Nor
##   warn('Non-invertible starting MA parameters found.'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maximum
##   warnings.warn("Maximum Likelihood optimization failed to "
```

```
predictions6 = model6_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals6 = pd.DataFrame(model6_fit.resid)
```

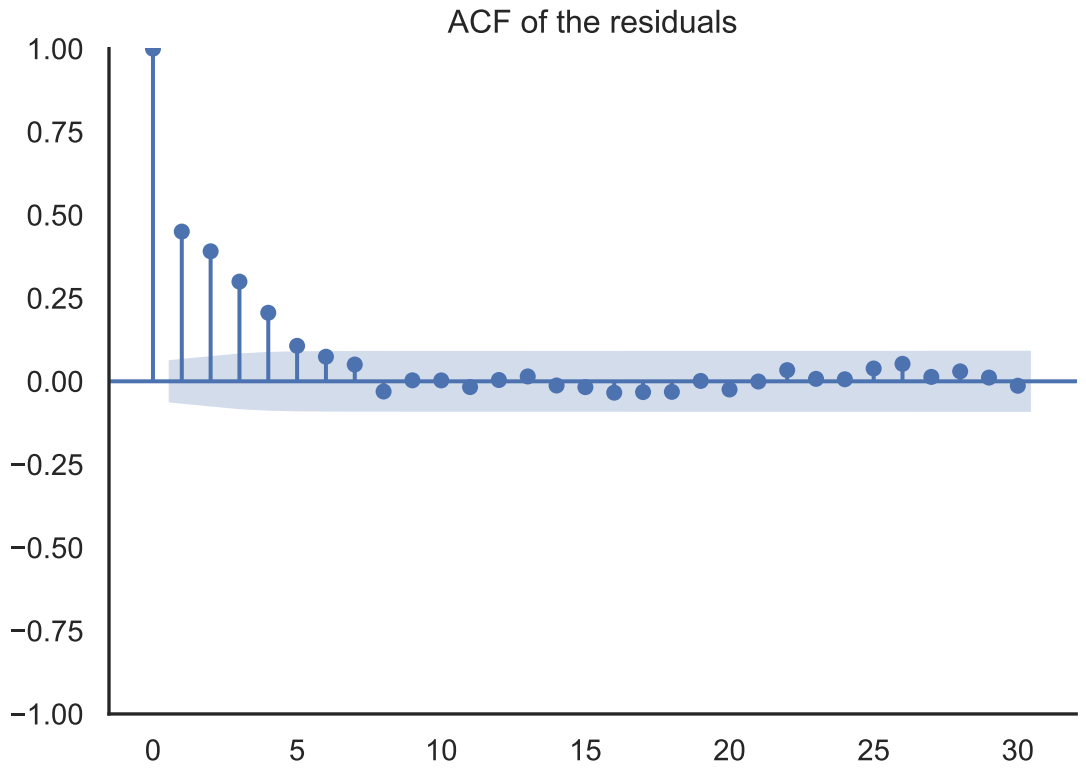



```
residuals6.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals6.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals6)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals6)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals6.describe())
```

```
##                0
## count  852.000000
## mean     0.254779
## std      2.204440
## min     -6.320871
## 25%     -0.838487
## 50%      0.118979
## 75%      1.133488
## max     21.699973
```

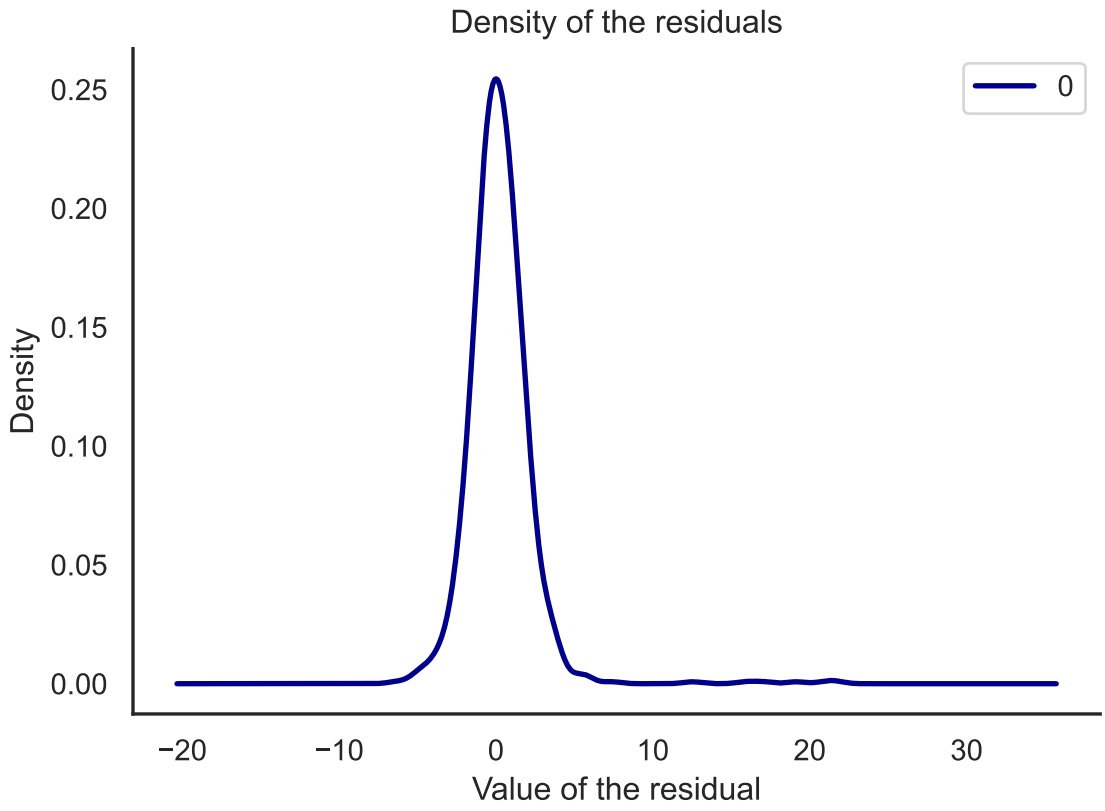


We can see that the residuals do not have the white noise structure when $d = 0$. We are going to try some other models in order to discard that this is not due to happenstance but we will see that this is a common ocurrence:

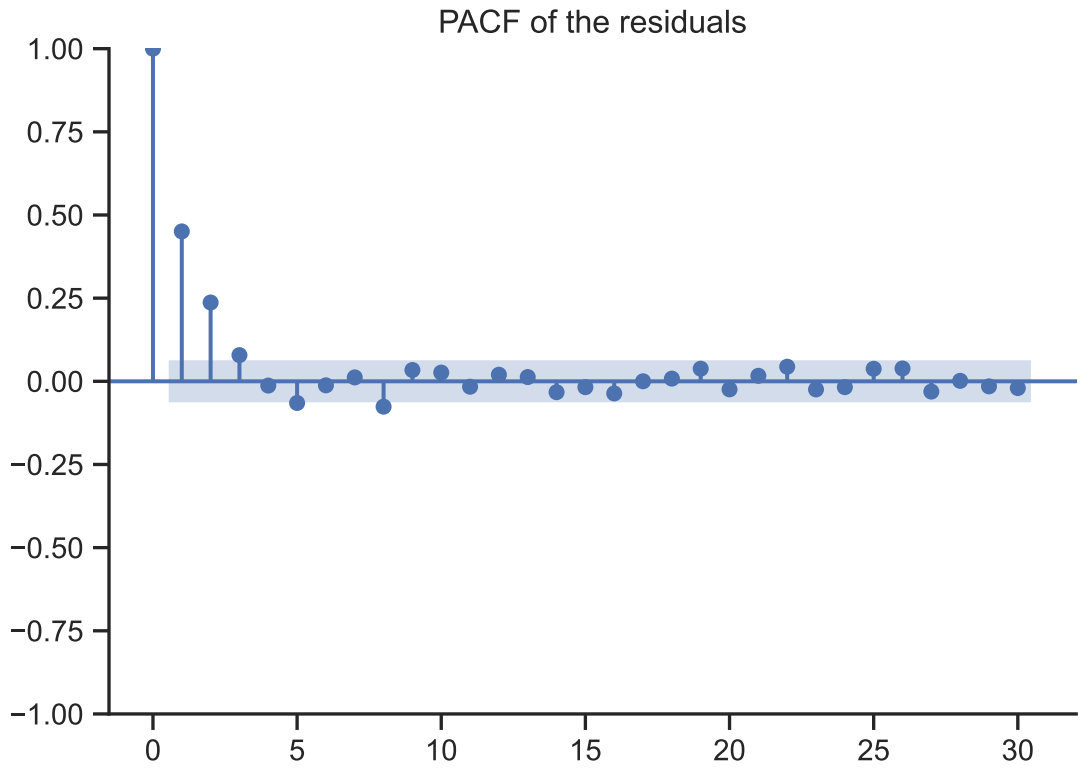
```
model7 = ARIMA(train, order=(5, 0, 2), seasonal_order=(4, 1, 1, 12))
model7_fit = model7.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maximum
##   warnings.warn("Maximum Likelihood optimization failed to "
```

```
predictions7 = model7_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals7 = pd.DataFrame(model7_fit.resid)
```

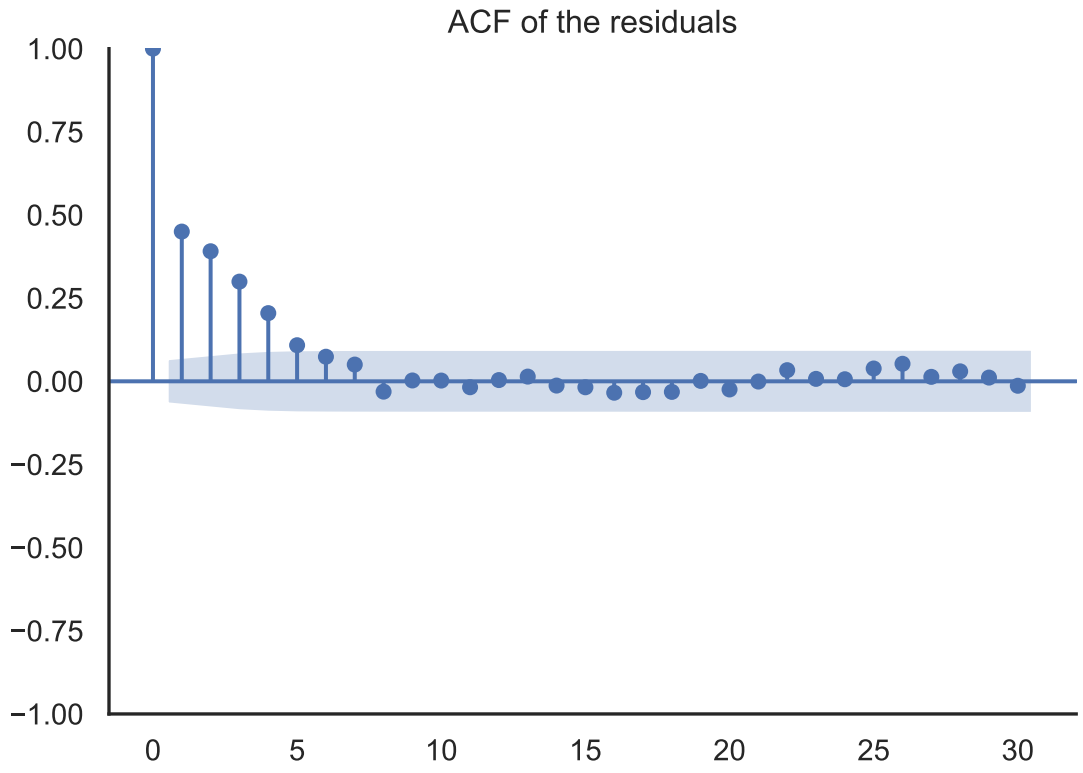



```
residuals7.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals7.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals7)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals7)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals7.describe())
```

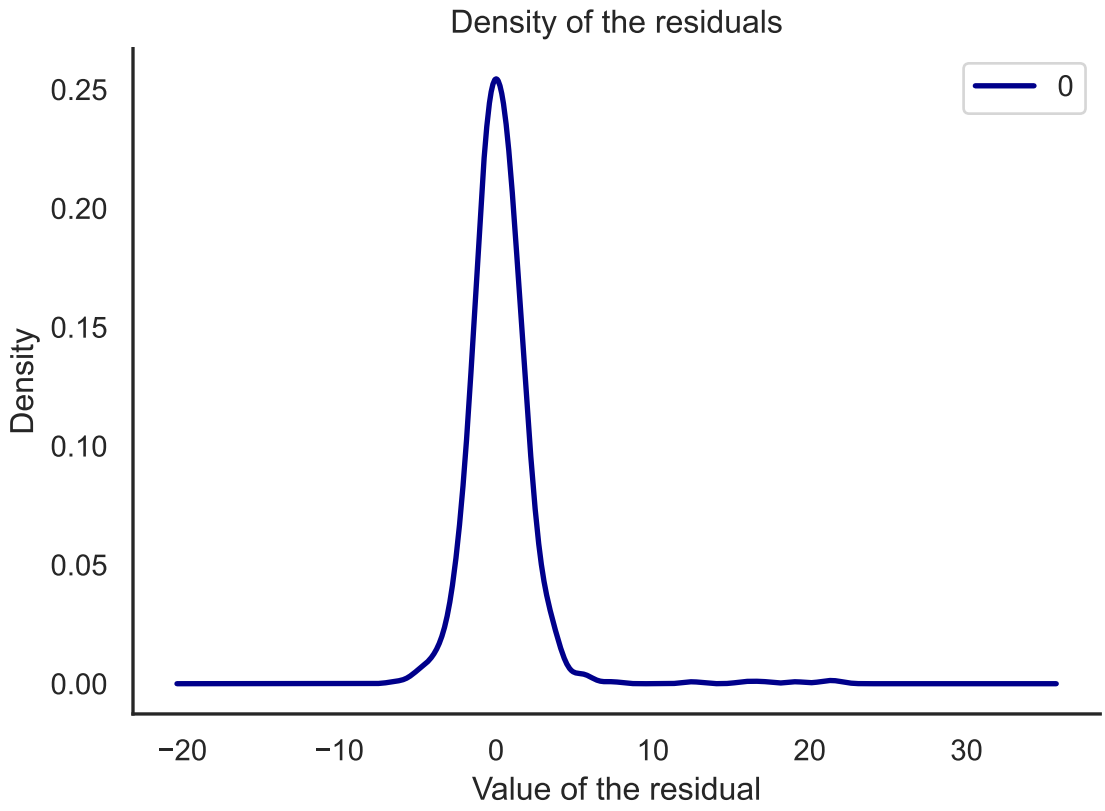
```
##                 0
## count  852.000000
## mean     0.253650
## std      2.204495
## min     -6.328468
## 25%     -0.838305
## 50%      0.118911
## 75%      1.134429
## max     21.699973
```



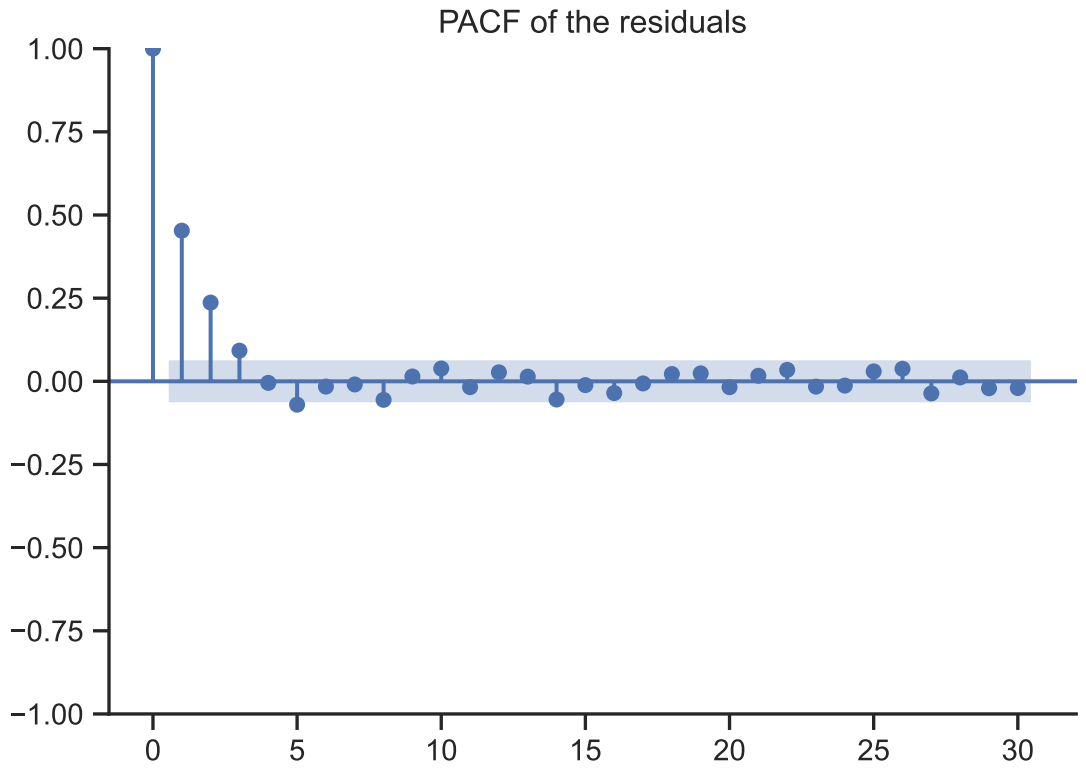

```
model8 = ARIMA(train, order=(5, 0, 3), seasonal_order=(4, 1, 1, 12))
model8_fit = model8.fit()
```

```
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:966: UserWarning: Nor
##   warn('Non-stationary starting autoregressive parameters'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\tsa\statespace\sarimax.py:978: UserWarning: Nor
##   warn('Non-invertible starting MA parameters found.'
## C:\Users\alema\ANACON~1\Lib\site-packages\statsmodels\base\model.py:604: ConvergenceWarning: Maximum
##   warnings.warn("Maximum Likelihood optimization failed to "
```

```
predictions8 = model8_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals8 = pd.DataFrame(model8_fit.resid)
```

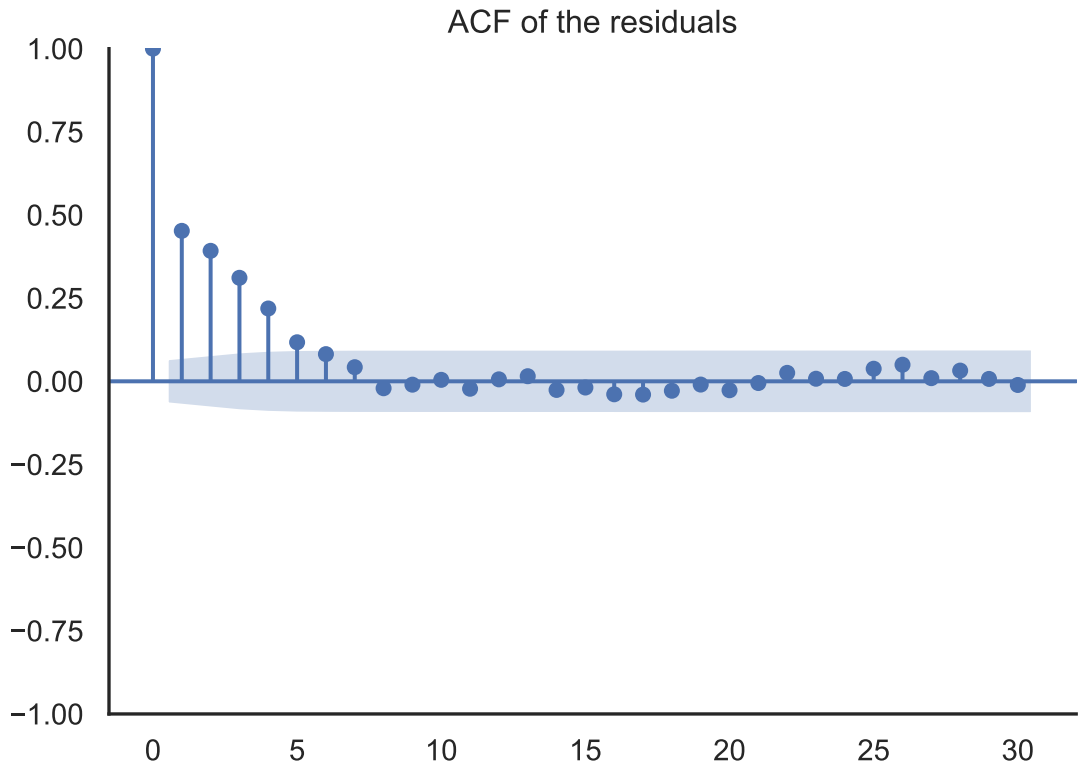



```
residuals8.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals8.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals8)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals8)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```

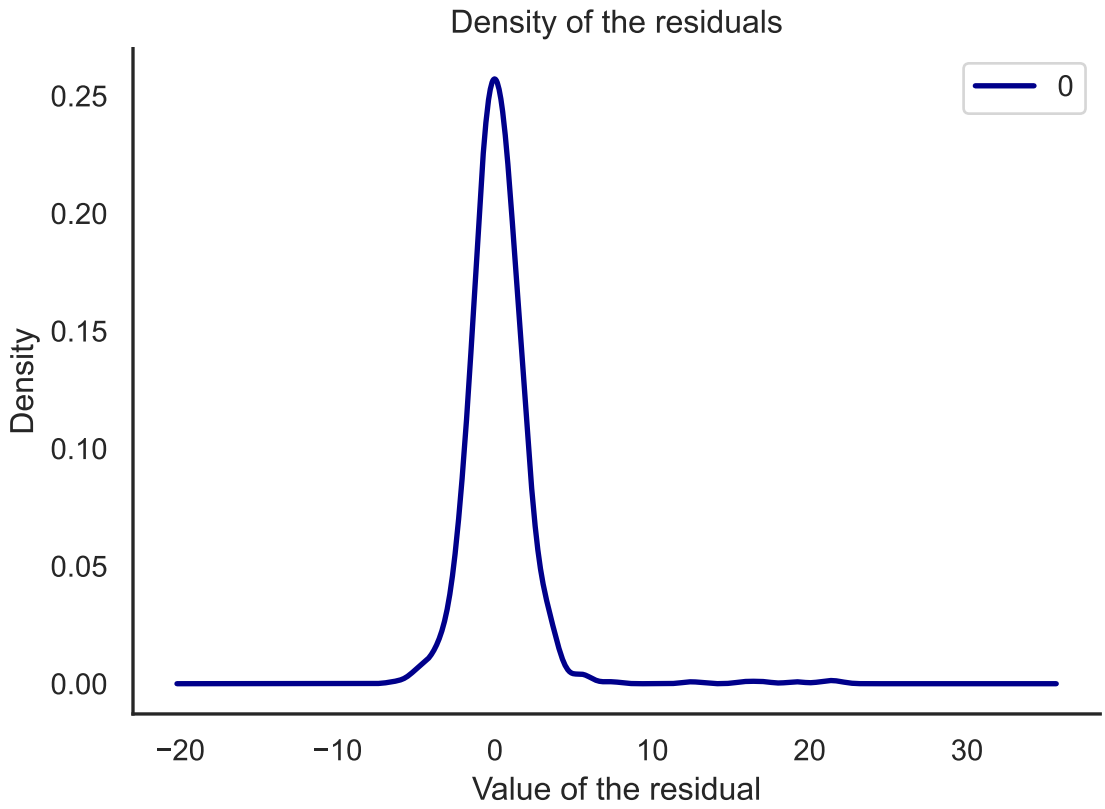



```
print(residuals8.describe())
```

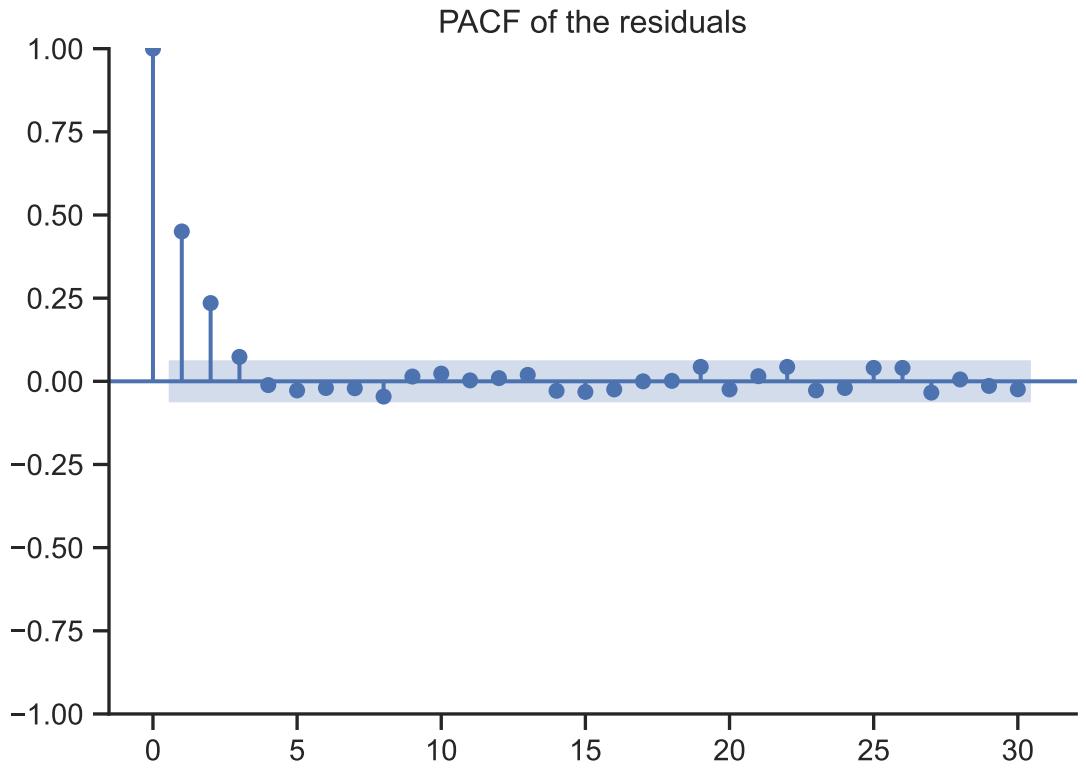
```
##                0
## count  852.000000
## mean     0.221988
## std      2.204777
## min     -6.236824
## 25%     -0.863416
## 50%      0.062409
## 75%      1.057657
## max     21.699977
```




```
model9 = ARIMA(train, order=(5, 0, 2), seasonal_order=(5, 1, 1, 12))
model9_fit = model9.fit()
predictions9 = model9_fit.predict(start=len(train), end=len(train)+len(test)-1, typ='levels')
residuals9 = pd.DataFrame(model9_fit.resid)
```

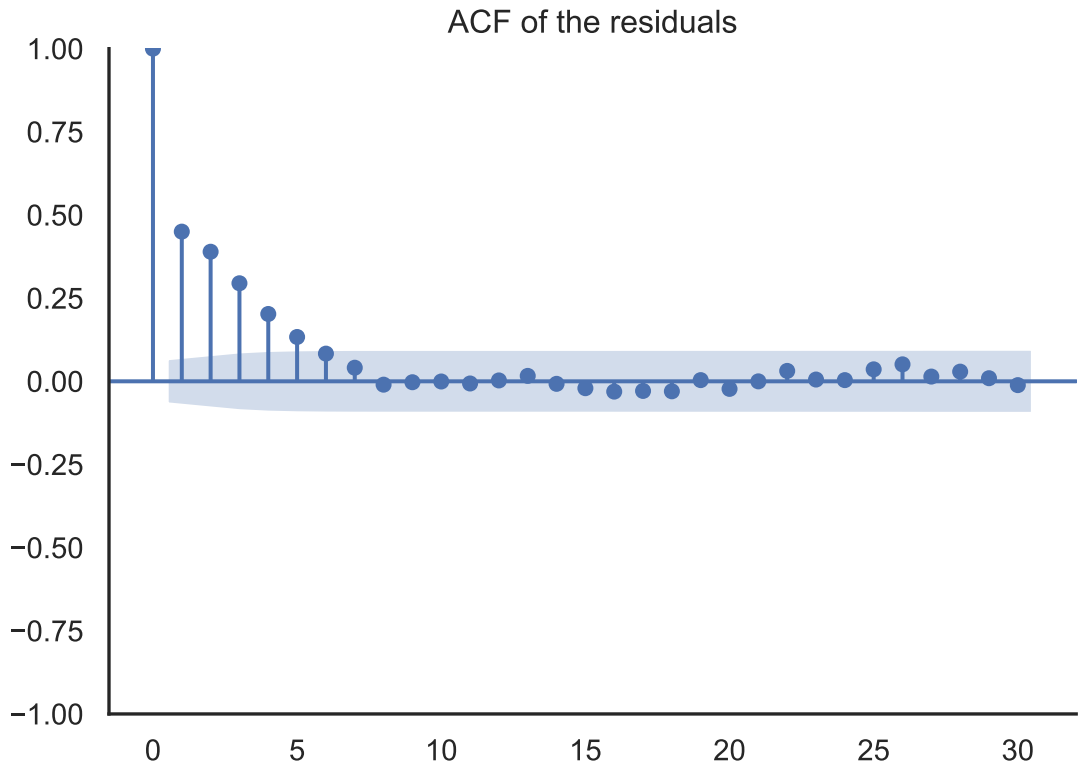



```
residuals9.plot(linewidth = 1, color = 'darkblue', linestyle = '-', title = 'Residuals')
plt.xlabel('Date')
plt.ylabel('Residuals')
sns.set_theme(style = 'white')
sns.despine()

residuals9.plot(kind='kde', linewidth = 2, color = 'darkblue', title = 'Density of the residuals', xlab
plt.xlabel('Value of the residual')
plt.ylabel('Density')
sns.set_theme(style = 'white')
sns.despine()

plot_acf(residuals9)
plt.title('ACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()

plot_pacf(residuals9)
plt.title('PACF of the residuals')
sns.set_theme(style = 'ticks')
sns.despine()
plt.show()
```




```
print(residuals9.describe())
```

```
##                0
## count  852.000000
## mean     0.270881
## std      2.200757
## min     -6.234438
## 25%     -0.801542
## 50%      0.149855
## 75%      1.104486
## max     21.699969
```



We are not going to consider models with $d = 0$ because their residuals are not white noise. However, we can see that all models with $d = 1$ have the appropriate residuals structure. Therefore, the only way to choose between them is through numerical criteria like the AIC and the MSE:

```
#We print the AIC
print(model1_fit.aic)
```

```
## 3162.5539173268476
```

```
print(model2_fit.aic)
```

```
## 3163.861639460554
```

```
print(model3_fit.aic)
```

```
## 3161.1454161182746
```

```
print(model4_fit.aic)
```

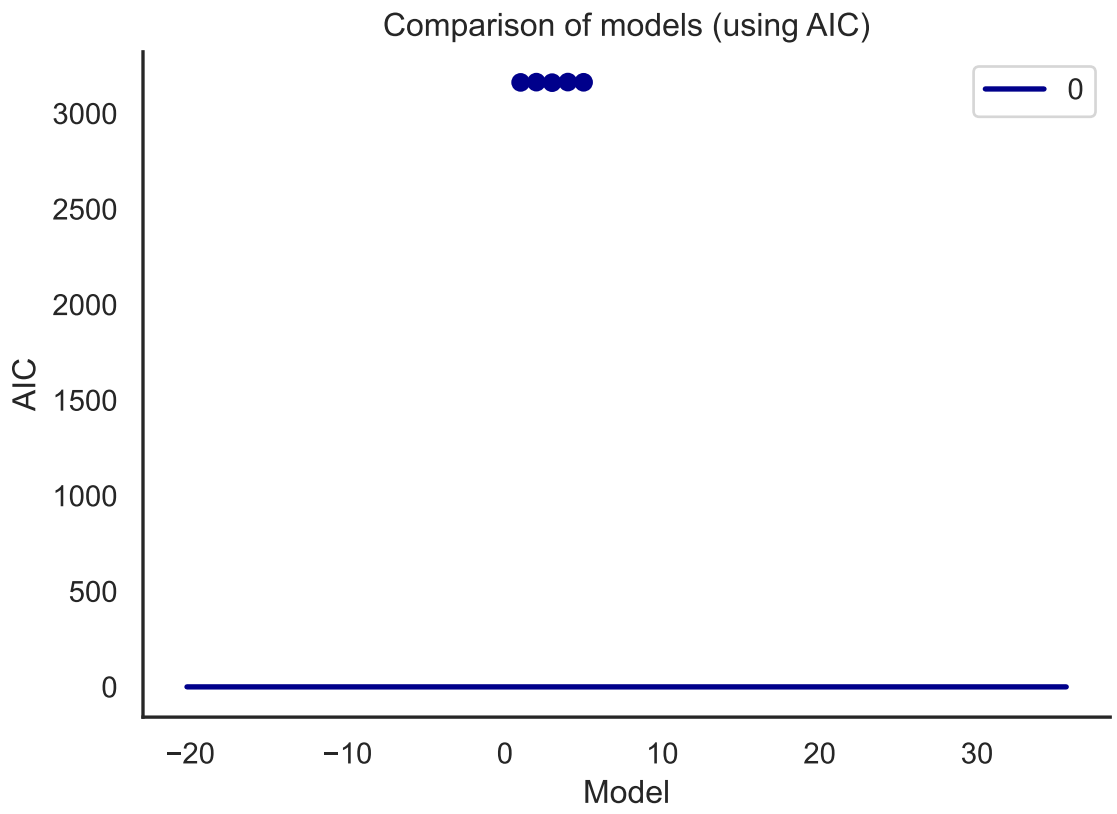
```
## 3164.2096155223617
```

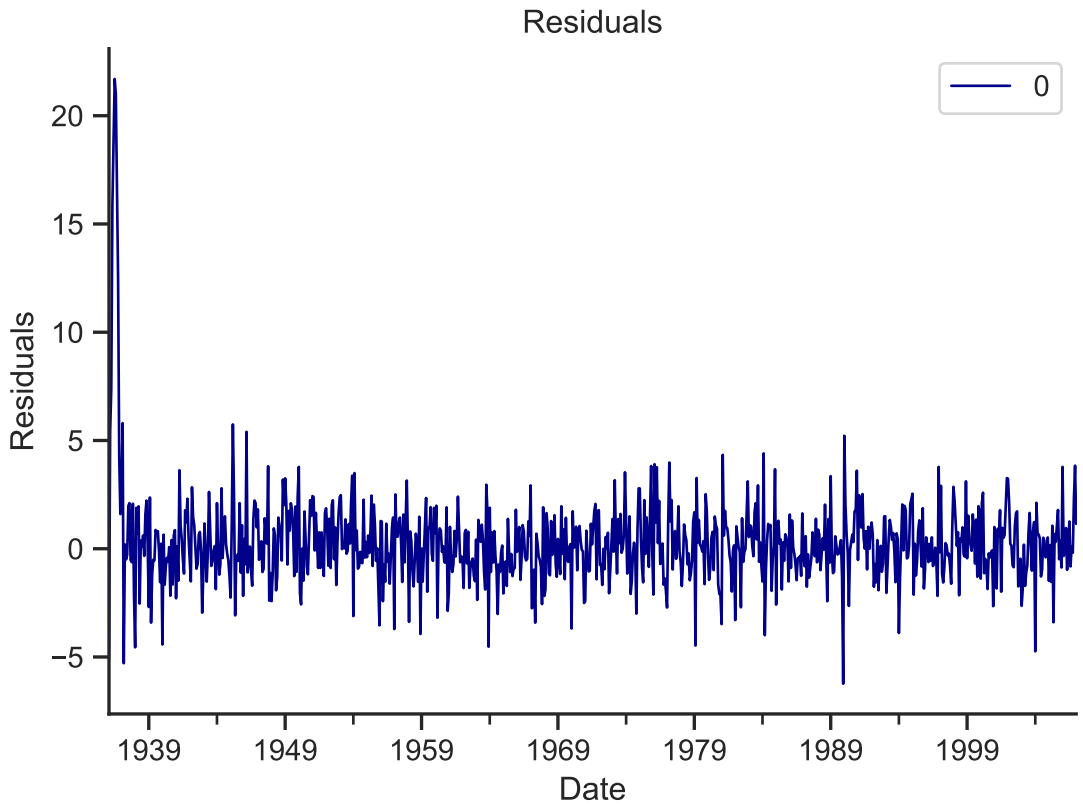
```
print(model5_fit.aic)
```

```
## 3163.1020302605357
```

```
#We plot our results
AIC = [model1_fit.aic, model2_fit.aic, model3_fit.aic, model4_fit.aic, model5_fit.aic]
plt.plot([1,2,3,4,5], AIC, linewidth = 2, marker = 'o', color = 'darkblue', linestyle = ':')
plt.xlabel('Model')
plt.ylabel('AIC')
```

```
plt.title('Comparison of models (using AIC)')
sns.set_theme(style = 'white')
sns.despine()
plt.show()
```






```
#We print the MSE
print(model1_fit.mse)
```

```
## 2.9278403820518046
```

```
print(model2_fit.mse)
```

```
## 2.9262306564853753
```

```
print(model3_fit.mse)
```

```
## 2.913198727554039
```

```
print(model4_fit.mse)
```

```
## 2.9137184430077814
```

```
print(model5_fit.mse)
```

```
## 2.9130483243600014
```

```
print(model6_fit.mse)
```

```
## 4.918763069733953
```

```
print(model7_fit.mse)
```

```
## 4.918432982204188
```

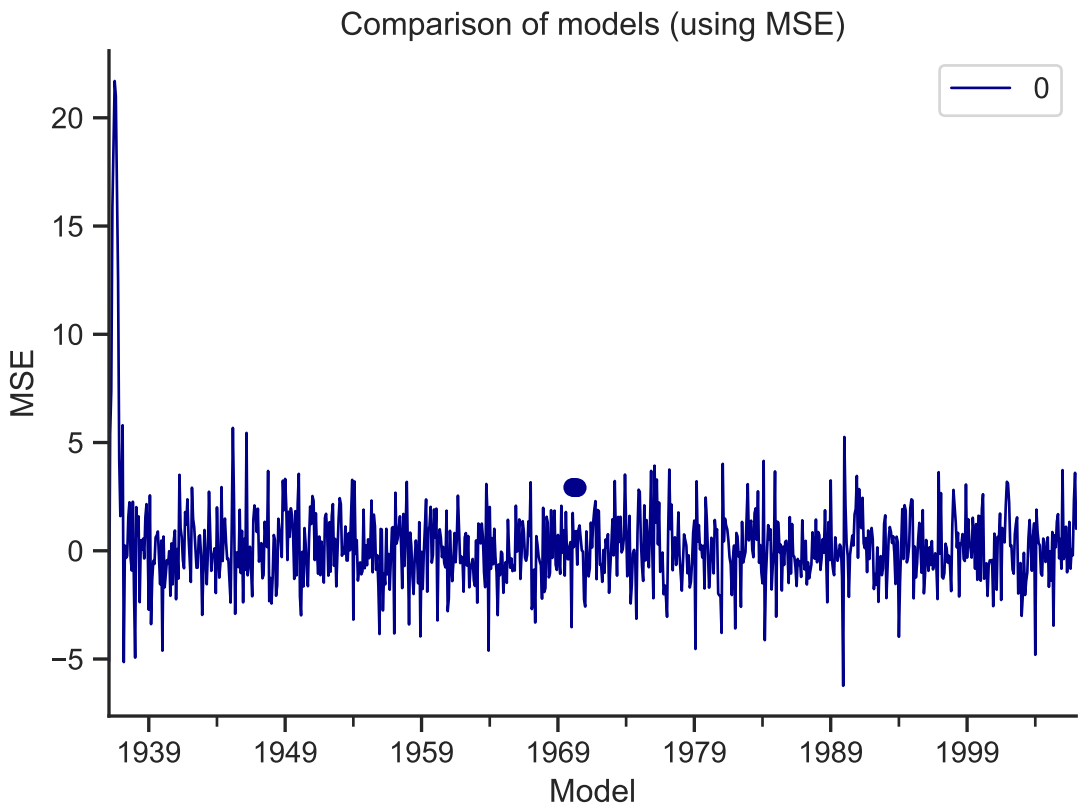
```
print(model8_fit.mse)
```

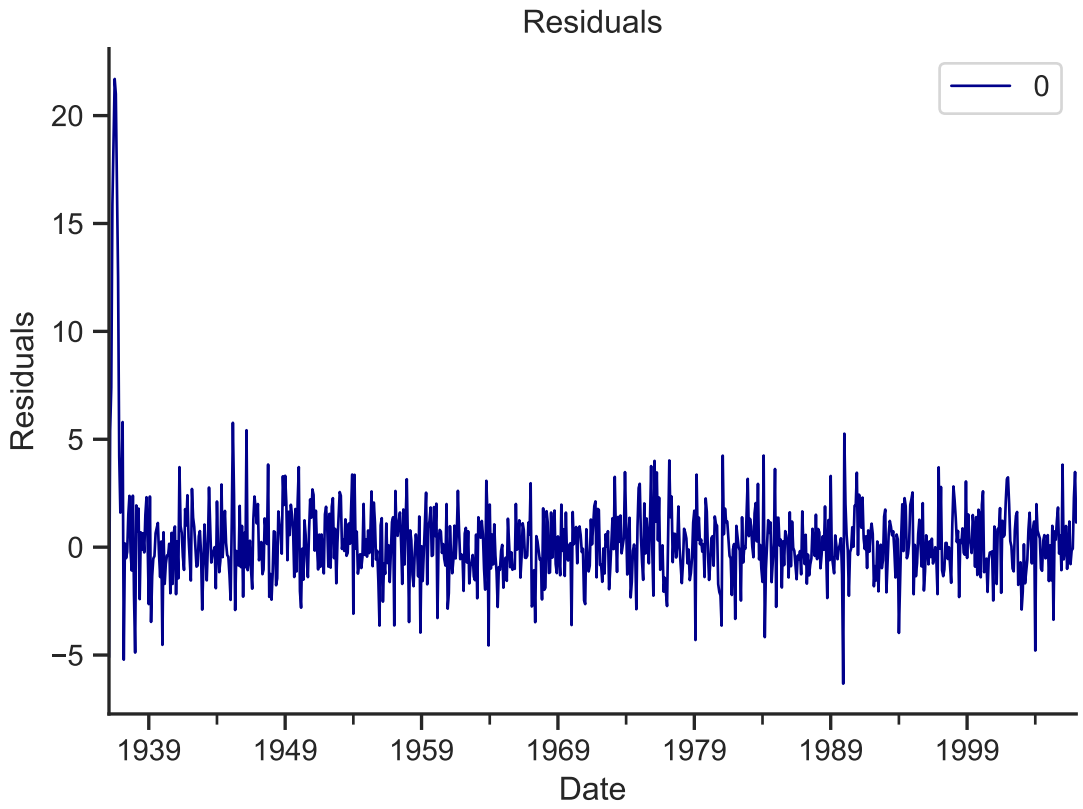
```
## 4.904616767470473
```

```
print(model9_fit.mse)
```

```
## 4.911025379461177
```

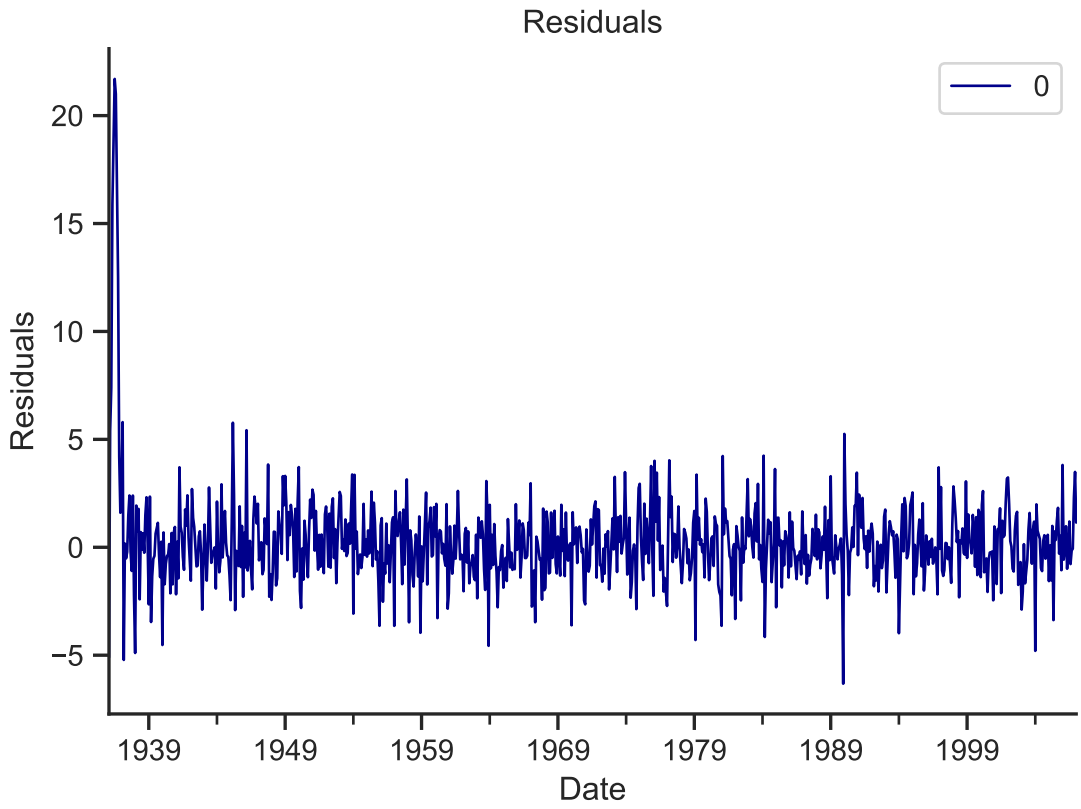
```
#We plot the MSE
MSE = [model1_fit.mse, model2_fit.mse, model3_fit.mse, model4_fit.mse, model5_fit.mse]
plt.plot([1, 2, 3, 4, 5], MSE, linewidth = 2, marker = 'o', color = 'darkblue', linestyle = ':')
plt.xlabel('Model')
plt.ylabel('MSE')
plt.title('Comparison of models (using MSE)')
sns.set_theme(style = 'white')
sns.despine()
plt.show()
```





We can see that the model with the lowest `AIC` is model 3 (the one with $(5, 1, 2)$, seasonal: $(4, 1, 1)$, $s = 12$). Moreover, although it is not the one with the lowest MSE (that would be model 5), its MSE is close to the lowest. Therefore, this is the model that we are going to use for our predictions. Moreover we already have its performance metrics (`AIC` and `MSE`). Therefore, let us store this information:

```
#We store the final model and its predictions
model = model3_fit
model_mse = model.mse
model_aic = model.aic
predictions = predictions3
```



4. Bootstrap

In order to determine standard error and confidence intervals for the coefficients we can make the following procedure:

1. Obtain a bootstrap sample for the coefficients.
2. Estimate the standard error using the standard deviation of the sample.
3. Estimate the confidence intervals using the quantile intervals.

In order to obtain a bootstrap sample we have decided to implement two different procedures, the *residuals method* and *block method*. We start by explaining the *residuals method*:

1. We start by estimating the parameters of the original series.
2. Then we compute the residuals of the coefficients.
3. Then we sample n elements from the residuals with replacement (n being the length of the series) and we simulate a replicate of the time series with the estimated parameters and add the sampled residuals.
4. We estimate the coefficients of the new series.

This method is implemented through the following code:

```
#We first store important variables:

#Number of bootstrap samples
B <- 1000

#Parameters of the ARIMA model
order_arima = c(5, 1, 2)
order_seas = c(4, 1, 1)
```

```
s = 2

#Training data
y = data_train[,2]
len_train = length(data_train[,2])

#We now fit the model
est.arima = arima(y, order = order_arima, seasonal = list(order = order_seas, period = s), method = 'CS
#We store its coefficients
coefficients <- coef(est.arima)
#And residuals
residuals <- est.arima$residuals

#We implement the residuals method
boot.residuals <- function(){
  #We sample a list of indices
  index <- sample(1:len_train, size = len_train, replace = TRUE)
  #We extract the residual associated to this indices
  resid_boot <- residuals[index]
  #We simulate an ARIMA model with residuals
  model <- Arima(ts(resid_boot,freq=12), order=order_arima,
                 seasonal=list(order = order_seas, period = s),
                 fixed=coefficients)
  simul <- simulate(model, nsim = len_train)
  #We estimate the coefficients from this model
  est.arima.boot <- arima(simul, order = order_arima,
                          seasonal = list(order = order_seas, period = s),
                          include.mean = FALSE,method='CSS')
  return(coef(est.arima.boot))
}

#We simulate B samples
srs.residuals <- replicate(B, boot.residuals())
```

On the other hand, we have the *block procedure* which consists of the following:

1. We divide our data into a series of blocks of a fixed length.
2. We obtain a sample with replacement of these blocks and put them together.
3. We estimate the coefficients from this sample.

This method is implemented in the following code:

```
#We define the block length and the number of blocks (in our case we choose yearly data)
blockLen = 12
N = length(y)
blockNum = round(N/blockLen)

#Block procedure:
betaBlock = replicate(B, {
  #Start: we sample the starting positions of the blocks
  start = sample(1:(N - blockLen + 1), size = blockNum, replace = TRUE)
  #We obtain the indices of the rest of the elements of the blocks
  blockedIndices = c(sapply(start, function(x) seq(x, x + blockLen - 1)))
  #We obtain the sample of the blocks
  eso = y[blockedIndices]
```

```
  #We estimate the coefficients
  coef(arima(eso, order = order_arima, seasonal = list(order = order_seas, period = 12), method = 'CSS'
})
```

We are now going to obtain make computations using these samples. In particular:

1. We are going to plot the density of each bootstrap sample (to get an idea of the distribution of the coefficients).
2. We are going to compute its standard deviation.
3. We are going to compute its mean value.
4. We are going to compute its confidence intervals.

This is done automatically with the following code:

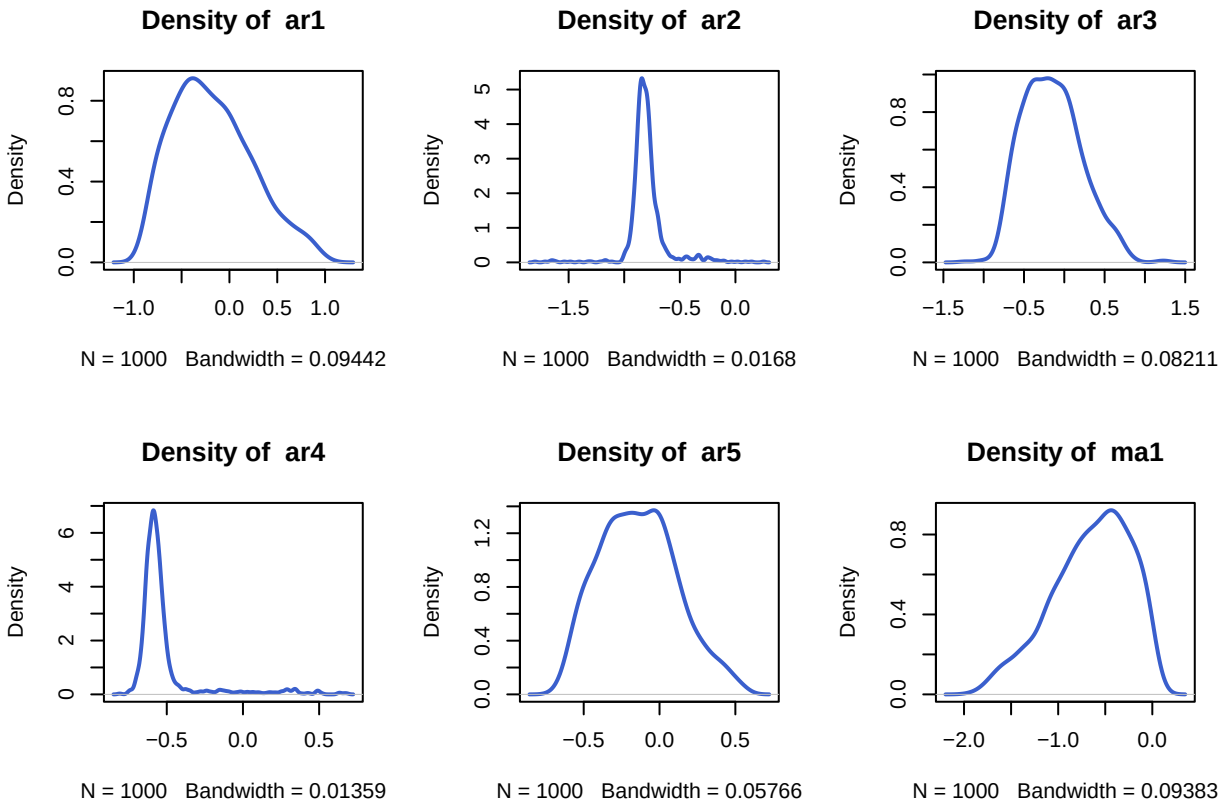
```
results <- function(beta.boot, alfa = 0.05){
  #We store the names of the coefficients
  names <- c('ar1', 'ar2', 'ar3', 'ar4', 'ar5',
             'ma1', 'ma2', 'sar1', 'sar2', 'sar3', 'sar4', 'sma')
  #Vector with the standard deviations
  sds <- rep(0, 12)
  #Matrix with the confidence intervals
  CI <- matrix(0, nrow = 12, ncol = 2)
  #For each coefficient:
  par(mfrow=c(2,3))
  for(i in 1:12){
    print(paste('Coefficient: ', names[i]))
    print('')
    #We plot its density
    plot(density(beta.boot[i,]), col = 'royalblue3', lwd = 2, main = paste('Density of ', names[i]))
    print('Left: variance of the coefficient, right: estimated bootstrap sd.')
    #We compute its standard deviation
    sds[i] <- sd(beta.boot[i,])
    #We compare it with the estimated variance
    print(c(sqrt(vcov(est.arima))[i,i], sds[i]))
    print('Confidence interval for the coefficient')
    #We compute the confidence interval
    CI[i,] <- quantile(beta.boot[i,], probs = c(alfa/2, 1-alfa/2))
    print(CI[i,])
    print('---------------')
    print('')
  }
  #We store our results in a list
  return(list(coef = colMeans(t(beta.boot)), intervals = CI, sds = sds))
}

#We obtain the results for both samples
results.residuals <- results(srs.residuals)
```

```
## [1] "Coefficient:  ar1"
## [1] ""
```



```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.00100096 0.41766251
## [1] "Confidence interval for the coefficient"
## [1] -0.8167675  0.7539867
## [1] "---------------"
```

```
## [1] ""
## [1] "Coefficient:  ar2"
## [1] ""
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0005329577 0.1659245294
## [1] "Confidence interval for the coefficient"
## [1] -0.9791047 -0.3255384
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar3"
## [1] ""
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003539351 0.363202702
## [1] "Confidence interval for the coefficient"
## [1] -0.7273841  0.6226518
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar4"
## [1] ""
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0002225952 0.2149104270
## [1] "Confidence interval for the coefficient"
## [1] -0.6873407  0.2918609
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar5"
## [1] ""
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003401383 0.255075650
## [1] "Confidence interval for the coefficient"
## [1] -0.5710276  0.4086283
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ma1"
## [1] ""
```

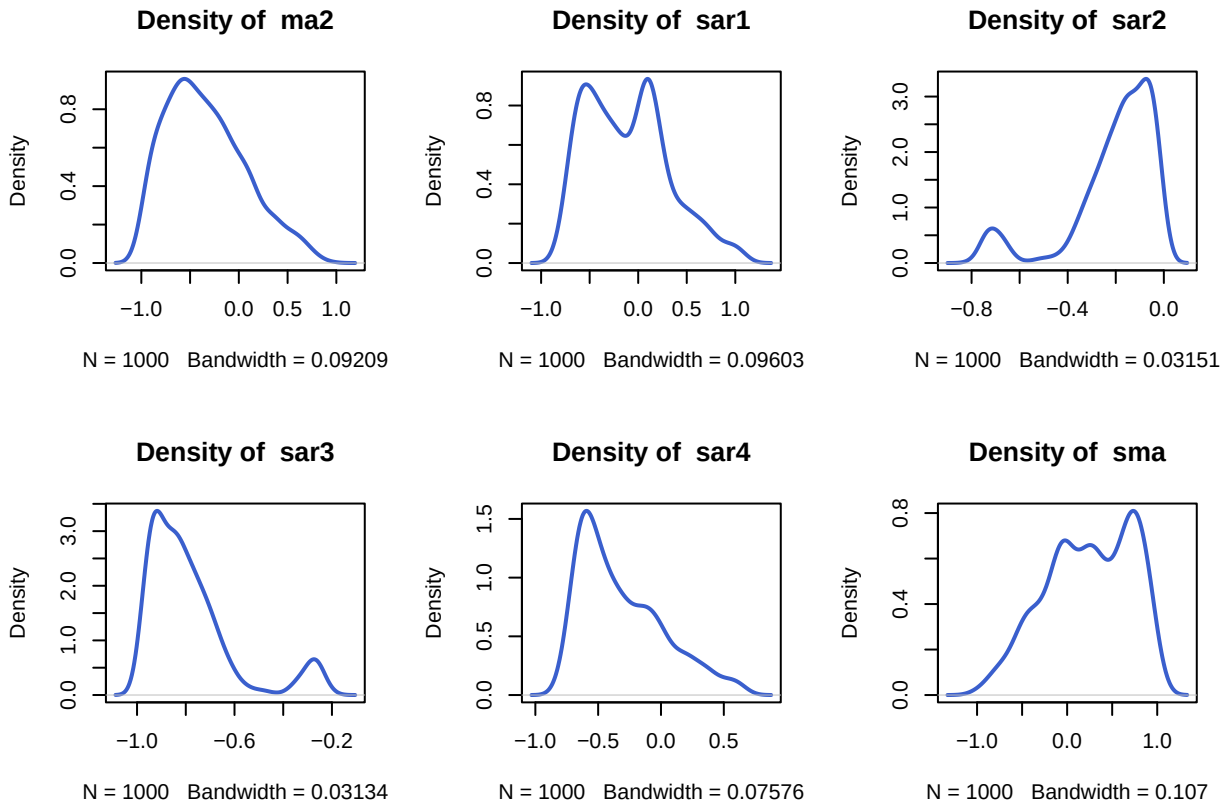



```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003542619 0.415046938
## [1] "Confidence interval for the coefficient"
## [1] -1.57278403 -0.02510625
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  ma2"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.004012425 0.407339463
## [1] "Confidence interval for the coefficient"
## [1] -0.9468385  0.5766277
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  sar1"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0001641681 0.4247882148
## [1] "Confidence interval for the coefficient"
## [1] -0.7249246  0.8388577
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  sar2"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
```

```
## [1] 0.0001108414 0.1801113060
## [1] "Confidence interval for the coefficient"
## [1] -0.72888060 -0.01185992
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sar3"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0001927951 0.1800558532
## [1] "Confidence interval for the coefficient"
## [1] -0.9765746 -0.2616597
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sar4"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0005972357 0.3351344472
## [1] "Confidence interval for the coefficient"
## [1] -0.7300498  0.4980051
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sma"
## [1] ""
```




```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.03306227 0.47346567
```

```
## [1] "Confidence interval for the coefficient"
## [1] -0.7301373  0.9467801
## [1] "--------------"
## [1] ""
```

```
results.block <- results(betaBlock)
```

```
## [1] "Coefficient:  ar1"
## [1] ""
```

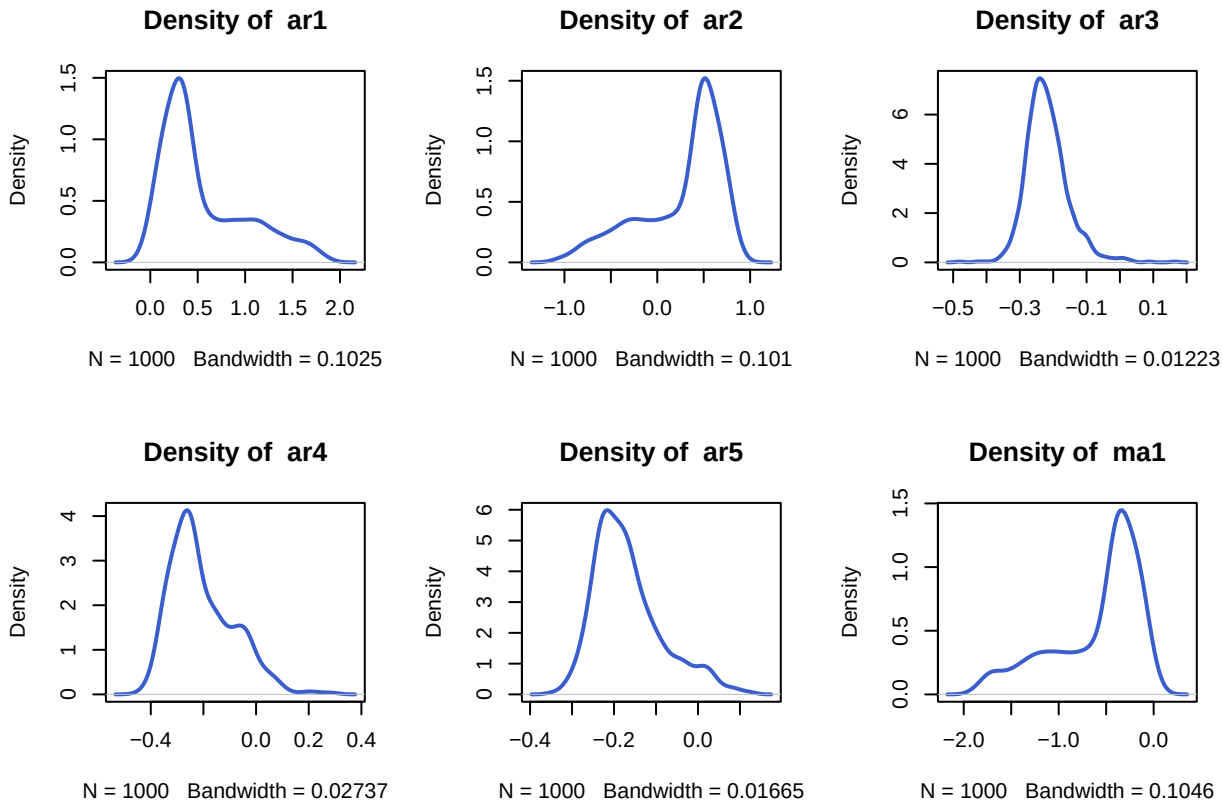
```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.00100096 0.45359535
## [1] "Confidence interval for the coefficient"
## [1] 0.04159232 1.65024584
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar2"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0005329577 0.4469845559
## [1] "Confidence interval for the coefficient"
## [1] -0.8069635  0.7788722
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar3"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003539351 0.063339223
## [1] "Confidence interval for the coefficient"
## [1] -0.31859233 -0.07827162
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar4"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0002225952 0.1210688170
## [1] "Confidence interval for the coefficient"
## [1] -0.37607516  0.06160312
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ar5"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003401383 0.082323820
## [1] "Confidence interval for the coefficient"
## [1] -0.28581038  0.03227474
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  ma1"
## [1] ""
```

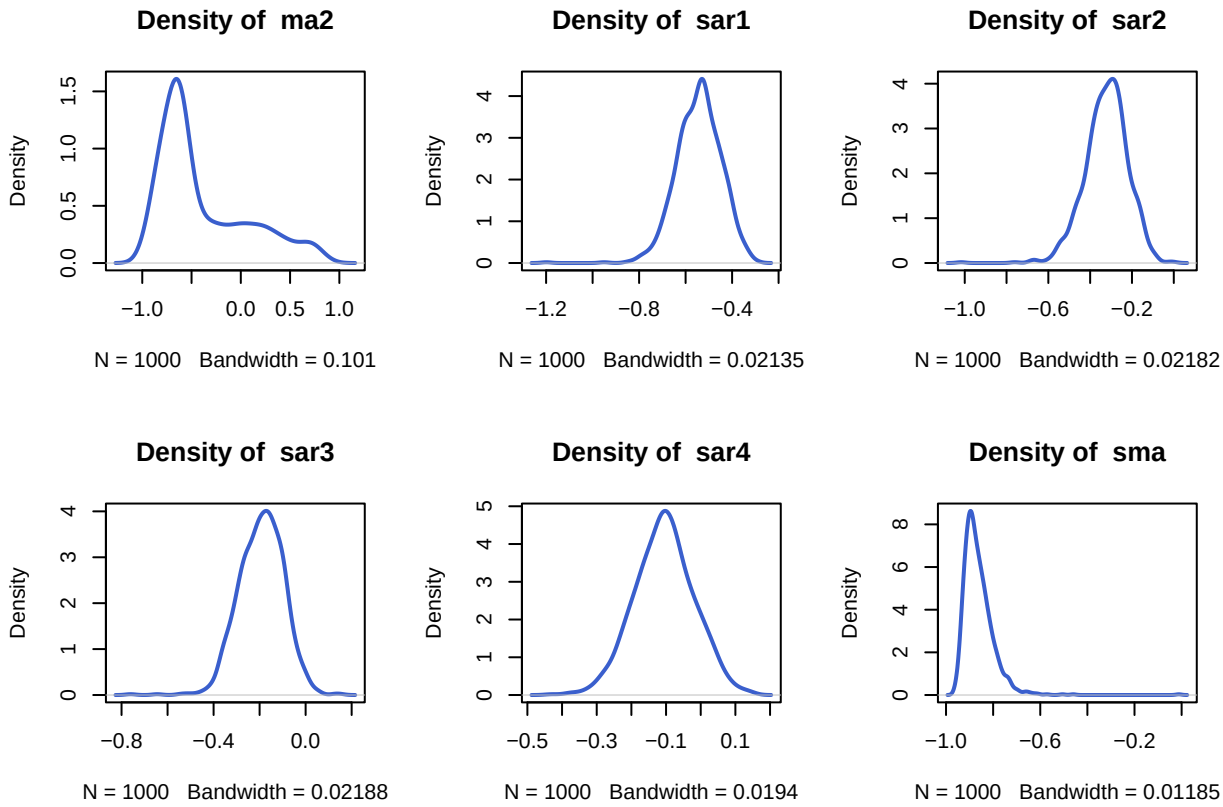



```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.003542619 0.462763640
## [1] "Confidence interval for the coefficient"
## [1] -1.71179967 -0.06832736
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  ma2"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.004012425 0.453015998
## [1] "Confidence interval for the coefficient"
## [1] -0.8954937  0.7110665
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  sar1"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0001641681 0.0951104109
## [1] "Confidence interval for the coefficient"
## [1] -0.7319119 -0.3710080
## [1] "---------------"
## [1] ""
## [1] "Coefficient:  sar2"
## [1] ""

## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
```

```
## [1] 0.0001108414 0.1023453598
## [1] "Confidence interval for the coefficient"
## [1] -0.5390723 -0.1434435
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sar3"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0001927951 0.0968047771
## [1] "Confidence interval for the coefficient"
## [1] -0.36960842 -0.01542676
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sar4"
## [1] ""
```

```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.0005972357 0.0862548120
## [1] "Confidence interval for the coefficient"
## [1] -0.27918400  0.06191689
## [1] "--------------"
## [1] ""
## [1] "Coefficient:  sma"
## [1] ""
```




```
## [1] "Left: variance of the coefficient, right: estimated bootstrap sd."
## [1] 0.03306227 0.06434492
```

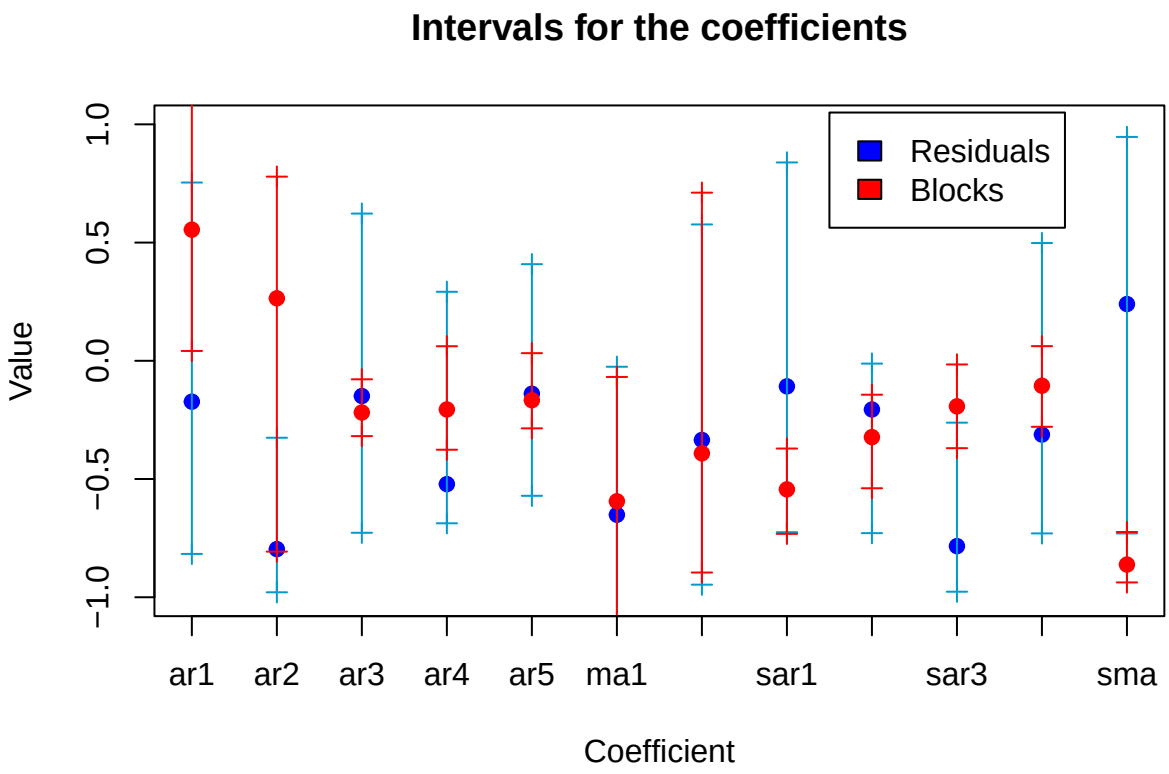
```
## [1] "Confidence interval for the coefficient"
## [1] -0.9372960 -0.7240353
## [1] "--------------"
## [1] ""
```

```
#And store each variable
coef.residuals <- results.residuals$coef
CI1 <- results.residuals$intervals
sds1 <- results.residuals$sds
coef.block <- results.block$coef
CI2 <- results.block$intervals
sds2 <- results.block$sds
```

The densities obtained through the two different methods can be compared using the densities plots showed above. At first sight, the main takeaway is the general difference in variance that the two methods show: Moving blocks bootstrap seems to have a much smaller variation around the mode than residuals-based bootstrap. Although there are some slight exceptions, more densities obtained through residuals-based bootstrap seems to be bimodal than those obtained through moving blocks bootstrap, especially when focusing on the coefficeints of the seasonal ARIMA. Both of these feature seem to indicate that, for the same number of bootstrap simulations, moving blocks boostrap shows a better convergence than residuals-based bootstrap.

We can visualize our results with the following plot:

```
plot(1:12, coef.residuals, main = 'Intervals for the coefficients',
     xlab = 'Coefficient', ylab = 'Value', ylim = c(-1,1), col = 'blue', bg = 'blue', pch = 21,xaxt='n')
for(i in 1:12){
  lines(c(i,i), CI1[i,], col = 'deepskyblue3')
  points(c(i,i), CI1[i,], pch = 3, col = 'deepskyblue3')
}

points(1:12, coef.block,xlab = 'Coefficient', ylab = 'Value', ylim = c(-1,1), col = 'red', bg = 'red', p
for(i in 1:12){
  lines(c(i,i), CI2[i,], col = 'red')
  points(c(i,i), CI2[i,], pch = 3, col = 'red')
}

legend(8.5, 1.05, legend=c("Residuals", "Blocks"), fill=c("blue","red"))
names.coef <- c('ar1', 'ar2', 'ar3', 'ar4', 'ar5',
             'ma1', 'ma2', 'sar1', 'sar2', 'sar3', 'sar4', 'sma')
axis(side=1,at=1:12,labels=names.coef)
```

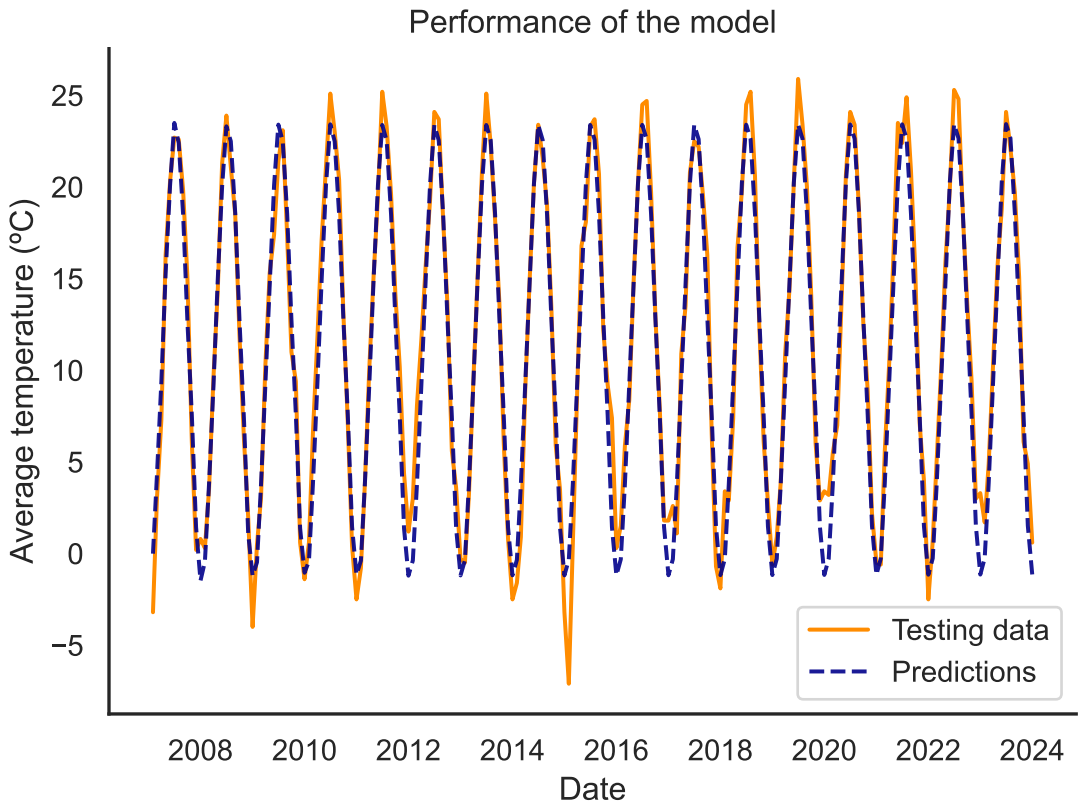


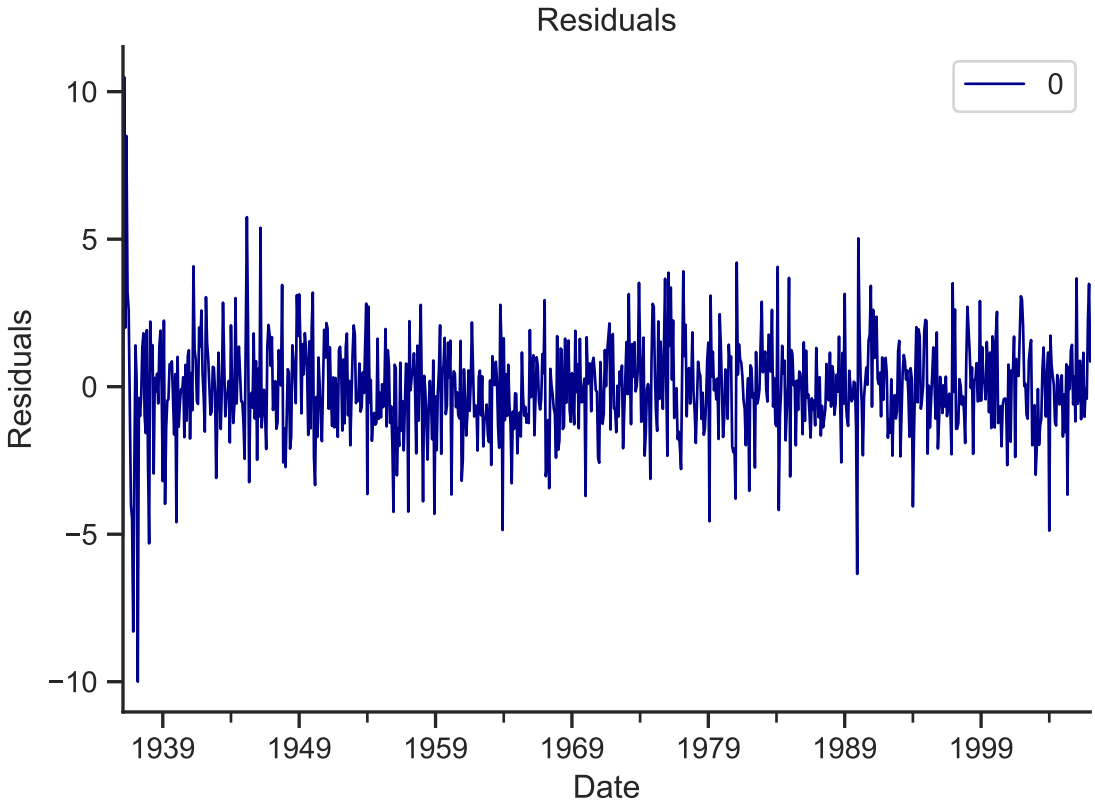
As we can see, the coefficients obtained through the two approaches range from essentially taking the same value to being radically different. Both bootstrap estimates can be compared with the coefficients estimated through the Python `arima` implementation, after which we find that most of moving blocks estimates are compatible with them, whereas several residuals-based estimates are quite far from it. Furthermore, the confidence intervals for some coefficients include 0 within their range. This could indicate that said coefficients are actually 0 and thus the model considered would not be the most appropriate. However, the previous results from the comparison of several different ARIMA models suggest that this model is the correct one. Even still, this same process could be carried out with the rest of the model to check whether the confidence intervals of their coefficients would include 0.

5. Predictions

We are now ready to discuss our predictions. In reality they have already been computed. However, we can visualize them along with the testing set:

```
plt.clf()
plt.plot(test.index, test.values, label='Testing data',color="darkorange")
plt.plot(test.index, predictions, label='Predictions',color="darkblue", alpha = 0.9, linestyle = '--')
plt.title('Performance of the model')
plt.ylabel('Average temperature (ºC)')
plt.xlabel('Date')
sns.set_theme(style = 'white')
sns.despine()
plt.legend()
plt.show()
```


Performance of the model
Average temperature (°C)
25
20
15
10
5
0
−5
2008
2010
2012
2014
2016
2018
2020
2022
2024
Date
Testing data
Predictions



We can see that our fit is quite good. We can see that it fails to capture the most extreme behavior we can see in the highest peaks and the lowest valleys, but in general it fits the data quite well. This is confirmed by both its `MSE` and `AIC`.

6. Vector Autoregressive Models (VAR)

6.1. A brief introduction to the topic

In this course we have extensively studied time series models in which we observed the evolution of one single variable through time. However, there are many scenarios for which it is interesting to check the evolution of many variables at the same time. For example, in a finance context one could be interested in price changes for multiple stocks in a portfolio, or when we deal with epidemics it is useful to track different groups of the population (say, sick, infected and recovered) in order to see the advancement of the disease. However, we do not need to look elsewhere in order to find an application: the dataset we are using tracks many variables other than temperature, many of which are related to each other. This is why there is a clear need to study such systems. Fortunately there is a whole subfield of time series modelling that deals with this type of data.

The simplest multivariate time series model one could come up with is the so-called *VAR(1)* model:

$$\mathbf{y}_t = \mu + \mathbf{A}_1 \mathbf{y}_{t-1} + \mathbf{a}_t$$

Where we have used bold to indicate the fact that we are dealing with vectors and matrices. $\mathbf{a}_t$ is a vector of innovations which follows a mutivariate normal distribution with a null mean vector and a diagonal covariance matrix. Let us assume for simplicity that $\mu = 0$.

This model allows for much of the manipulations we saw for the *AR(1)* model, but special care needs to be

put in order to deal with the matrix operators. For example, let us achieve the $MA(\infty)$ representation of the process. In order to do this we can write it using the lag operator in the following way:

$$(\mathbf{I} - \mathbf{A}_1 B)\mathbf{y}_t = \mathbf{a}_t$$

The matrix operator we can see on the left-hand side of the equation is invertible (in which case we will say the proccess is *invertible*) if and only if the roots of the equation:

$$det(\mathbf{I} - \mathbf{A}_1 z) = 0$$

(Where we are taking z as a complex variable) lie outside of the unit circle. In that case we can invert the operator and make the substitution:

$$\mathbf{y}_t = (\mathbf{I} - \mathbf{A}_1 B)^{-1}\mathbf{a}_t$$

If $\mathbf{A}_1$ has matrix norm strictly less than one (in which case we say that the proccess is *stable*) then we $(\mathbf{I} - \mathbf{A}_1\mathbf{B})^{-1}$ admits an expression as a series on $\mathbf{A}_1$, namely:

$$(\mathbf{I} - \mathbf{A}_1\mathbf{B})^{-1} = \sum_{i=0}^{\infty} \mathbf{A}_1^i B^i$$

Which implies that we can express the *VAR(1)* model as:

$$\mathbf{y}_t = \left(\sum_{i=0}^{\infty} \mathbf{A}_1^i B^i\right)\mathbf{a}_t = \sum_{i=0}^{\infty} \mathbf{A}_1^i \mathbf{a}_{t-i}$$

With this we have arrived at the $MA(\infty)$ representation of the *VAR(1)* process. However, the previous discussion can result somewhat dry. Perhaps a more natural way of arriving to this by recursive substitution in the first equation:

$$\mathbf{y}_t = \mathbf{A}_1^{j+1}\mathbf{y}_{y-(j+1)} + \sum_{i=0}^{j} A_1^i \mathbf{u}_{t-i}$$

Assuming that the process is stable we can take the limit $j \longrightarrow \infty$ and write:

$$\mathbf{y}_t = \sum_{i=0}^{\infty} \mathbf{A}_1^i \mathbf{u}_{t-i}$$

This discussion also applies to the general *VAR(p)* model, which has the following form:

$$\mathbf{y}_t = \sum_{i=1}^{p} \mathbf{A}_i \mathbf{y}_{t-i} + \mathbf{a}_t$$

We could express it using a matrix operator that takes the form of a polynomial on the lag operator:

$$\mathbf{y}_t = \sum_{i=1}^{p} A_i \mathbf{y}_{t-i} + \mathbf{u}_t = \sum_{i=1}^{p} A_i B^i \mathbf{y}_t + \mathbf{u}_t = \mathbf{A}(B)\mathbf{y}_t + \mathbf{u}_t$$

Where

$$A(B) = \sum_{i=1}^{p} A_i B^i$$

The inversion of the operator $(I - \mathbf{A}(B))$ would allow us to get conditions on the matrices $\mathbf{A}_i$ as well as the *MA(∞)* representation, just like we saw in the one dimensional case. However, in the vector case, all theoretical complications are restricted to the *VAR(1)* case because we can express any *VAR(p)* model as a *VAR(1)* of vectores of higher dimension. The way to do this is the following:

$$\begin{pmatrix} \mathbf{y}_t \\ \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \\ \dots \\ \mathbf{y}_{t-(p-1)} \\ \mathbf{y}_{t-p} \end{pmatrix} = \begin{pmatrix} \mathbf{A}_1 & \mathbf{A}_2 & \dots & \mathbf{A}_{p-1} & \mathbf{A}_p \\ \mathbf{I} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{I} & \dots & \mathbf{0} & \mathbf{0} \\ \dots & \dots & \dots & \dots & \dots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{I} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} & \mathbf{I} \end{pmatrix} \cdot \begin{pmatrix} \mathbf{y}_{t-1} \\ \mathbf{y}_{t-2} \\ \mathbf{y}_{t-3} \\ \dots \\ \mathbf{y}_{t-(p-2)} \\ \mathbf{y}_{t-(p-1)} \end{pmatrix} + \begin{pmatrix} \mathbf{a}_t \\ \mathbf{0} \\ \mathbf{0} \\ \dots \\ \mathbf{0} \\ \mathbf{0} \end{pmatrix}$$

So the study of the *VAR(1)* model is more than enough for vector autoregression (at least from the theoretical point of view).

In a more practical setting, many concrete aspects of the implementation of the *VAR* models are very similar to those that we have already seen in the 1D case (parameter estimation, bootstrap procedures,...). For example, if we have computed the parameters of the model we can make forecasts using the equations:

$$\tilde{\mathbf{y}}_{t+1|t} = \sum_{i=1}^{p} \mathbf{A}_i \mathbf{y}_{t+1-i}$$

$$\tilde{\mathbf{y}}_{t+2|t} = \mathbf{A}_1 \tilde{\mathbf{y}}_{t+1|t} + \sum_{i=2}^{p} \mathbf{A}_i \mathbf{y}_{t+2-i}$$

If we proceed until we have estimated p instances of the series:

$$\tilde{\mathbf{y}}_{t+p+2|t} = \sum_{i=1}^{p} \mathbf{A}_i \tilde{\mathbf{y}}_{t+p+1-i}$$

That is, the procedure is identical to that of the one dimensional case: we substitute the available data to make the first forecast and the we start using our forecasts as if it was from the original series. Eventually we end up using our forecasts for all the terms of the sum (provided we need to predict that far into the future).

More sophisticated methods (like bootstrap simulation of the predictions) are also available. However, they are either way beyond the scope of this report or almost identical to the one dimensional case. We are satisfied with just showcasing a brief introduction to the topic.

6.2. Application to our dataset.

Our original dataset recorded more variables other than the average temperature. This is why it is interesting to check if we can build a model that relates two different variables and explains their evolution and interactions. In particular, we are going to propose a *VAR* model for the average temperature and the `"EMNT"` (Which represents the extreme minimum temperature for the month). Therefore, we are looking for a relationship of the form:

$$M_t = a_{11}^1 M_{t-1} + a_{12}^1 T_{t-1} + ... + a_{11}^p M_{t-p} + a_{12}^p T_{t-p} + a_{t,1} T_t = a_{21}^1 M_{t-1} + a_{22}^1 T_{t-1} + ... + a_{11}^p M_{t-p} + a_{12}^p T_{t-p} + a_{t,2}$$

Where M_t represents the `"EMNT"` variable and T_t `"TAVG"`. For simplicity we have not written an explicit relationship between contemporary M and T.

In order to explore this model we are going to use the `vars` package:

```r
#We load the required libraries
library(vars)
```

```
## Loading required package: MASS
```

```
## Loading required package: strucchange
```

```
## Loading required package: zoo
```

```
##
## Attaching package: 'zoo'
```

```
## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```
## Loading required package: sandwich
```

```
## Loading required package: urca
```

```
## Loading required package: lmtest
```

```r
library(astsa)
```

```
##
## Attaching package: 'astsa'
```

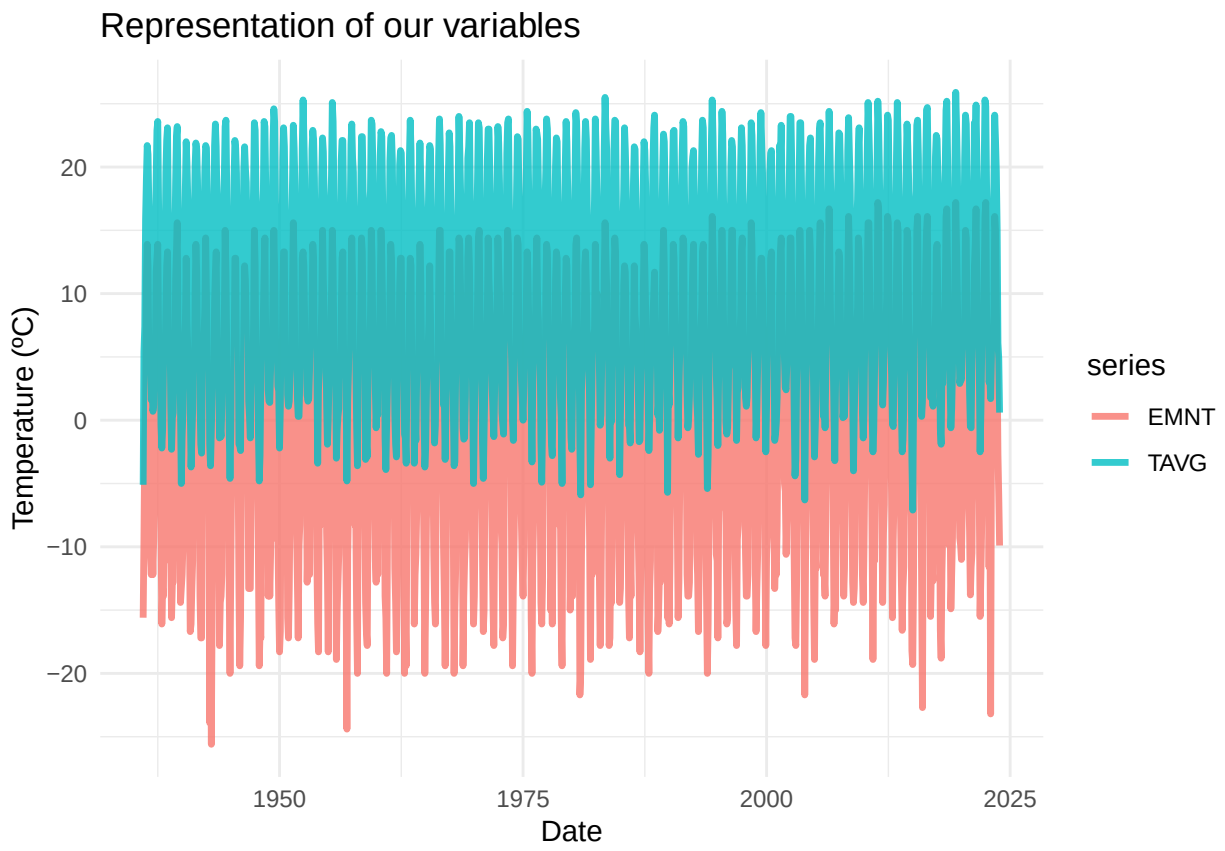
```
## The following object is masked from 'package:forecast':
##
##     gas
```

We load our data and store the relevant variables:

```r
data.var = read.csv("3610412.csv")
data.var = data.var[-c(1,2)]
data.var = data.var[,c('EMNT','TAVG')]
series.var = ts(data.var, start=c(1936,1,1), freq=12)
```

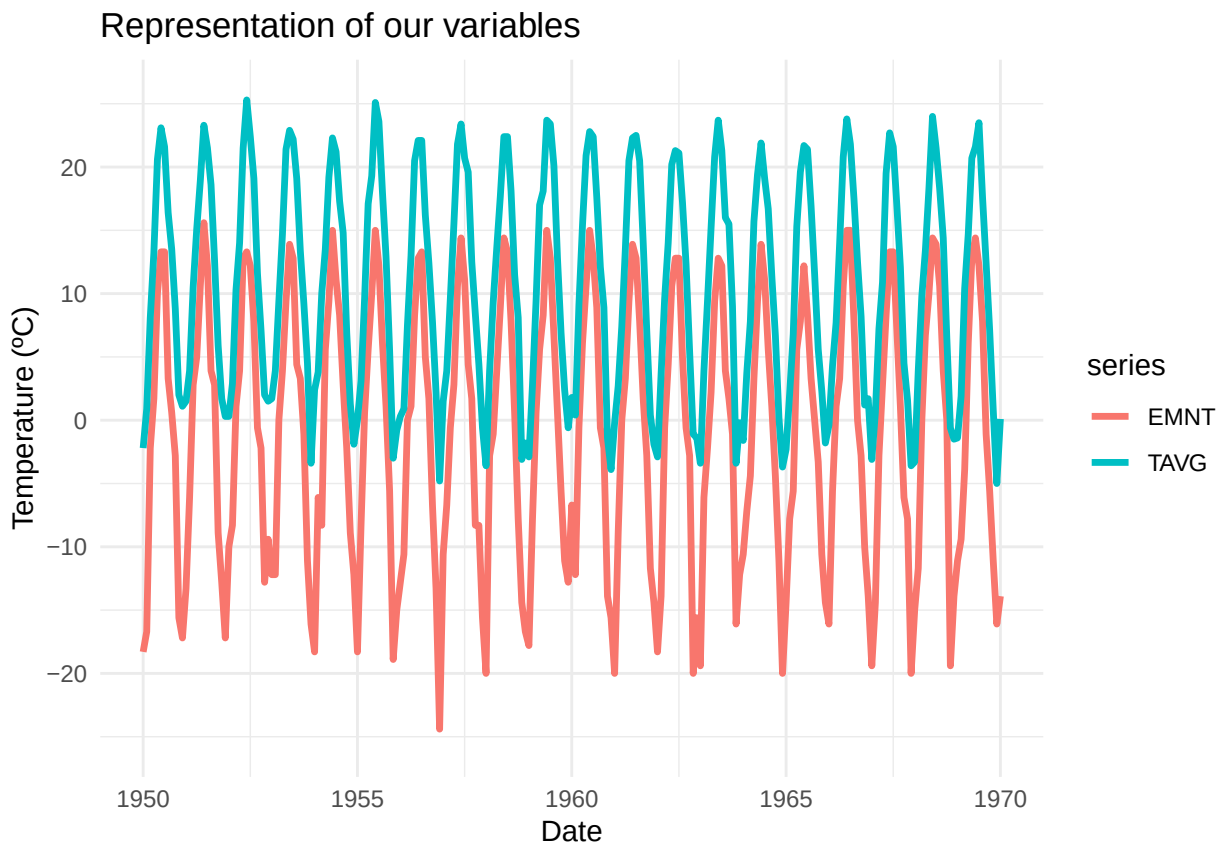
We can plot our variables together to see if they are related:

```r
autoplot(series.var, alpha = 0.8, lwd = 1.2) +
  ggtitle('Representation of our variables') +
  xlab('Date') +
  ylab('Temperature (ºC)') +
  theme_minimal()
```



This graphic is somewhat cluttered. In order to better visualize it we can restrict ourselves to a smaller time frame:

```
autoplot(series.var, lwd = 1.2) +
  xlim(1950, 1970) +
  ggtitle('Representation of our variables') +
  xlab('Date') +
  ylab('Temperature (ºC)') +
  theme_minimal()
```



We can see that there is a strong correlation between the two variables: their peaks and valleys are somewhat aligned. This suggests that our analysis will be fruitful. Let us fit a *VAR(2)* model. First, we should carry out another train-test split:

```
#First 70 years are used for training
var_train = data.var[c(1:852),]
var.series_train = ts(var_train, start=c(1936,1,1), freq=12)

#Last 17 years are used for testing
var_test = data.var[c(853:1056),]
var.series_test = ts(var_test, start=c(2001,1), freq=12)
```

```
fitvar = VAR(na.omit(var_train),p=2)
summary(fitvar)
```

```
##
## VAR Estimation Results:
## ========================
## Endogenous variables: EMNT, TAVG
## Deterministic variables: const
## Sample size: 850
## Log Likelihood: -4013.835
## Roots of the characteristic polynomial:
## 0.8877 0.8877 0.1647 0.1647
## Call:
## VAR(y = na.omit(var_train), p = 2)
##
```

```
##
## Estimation results for equation EMNT:
## ====================================
## EMNT = EMNT.l1 + TAVG.l1 + EMNT.l2 + TAVG.l2 + const
##
##         Estimate Std. Error t value Pr(>|t|)
## EMNT.l1  0.28307    0.05890   4.806 1.82e-06 ***
## TAVG.l1  1.46005    0.07681  19.008  < 2e-16 ***
## EMNT.l2 -0.11300    0.05976  -1.891    0.059 .
## TAVG.l2 -0.84073    0.07058 -11.912  < 2e-16 ***
## const   -7.28224    1.13838  -6.397 2.62e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 3.967 on 845 degrees of freedom
## Multiple R-Squared: 0.8547,  Adjusted R-squared: 0.854
## F-statistic:  1242 on 4 and 845 DF,  p-value: < 2.2e-16
##
##
## Estimation results for equation TAVG:
## ====================================
## TAVG = EMNT.l1 + TAVG.l1 + EMNT.l2 + TAVG.l2 + const
##
##         Estimate Std. Error t value Pr(>|t|)
## EMNT.l1  0.24755    0.04281   5.782 1.04e-08 ***
## TAVG.l1  1.20858    0.05584  21.645  < 2e-16 ***
## EMNT.l2 -0.06758    0.04344  -1.556     0.12
## TAVG.l2 -0.69200    0.05130 -13.488  < 2e-16 ***
## const    5.31637    0.82751   6.425 2.21e-10 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
##
## Residual standard error: 2.884 on 845 degrees of freedom
## Multiple R-Squared: 0.8909,  Adjusted R-squared: 0.8904
## F-statistic:  1726 on 4 and 845 DF,  p-value: < 2.2e-16
##
##
##
## Covariance matrix of residuals:
##        EMNT  TAVG
## EMNT 15.737 9.329
## TAVG  9.329 8.316
##
## Correlation matrix of residuals:
##        EMNT   TAVG
## EMNT 1.0000 0.8155
## TAVG 0.8155 1.0000
```

```
print(mean(fitvar$varresult$EMNT$residuals^2))
```

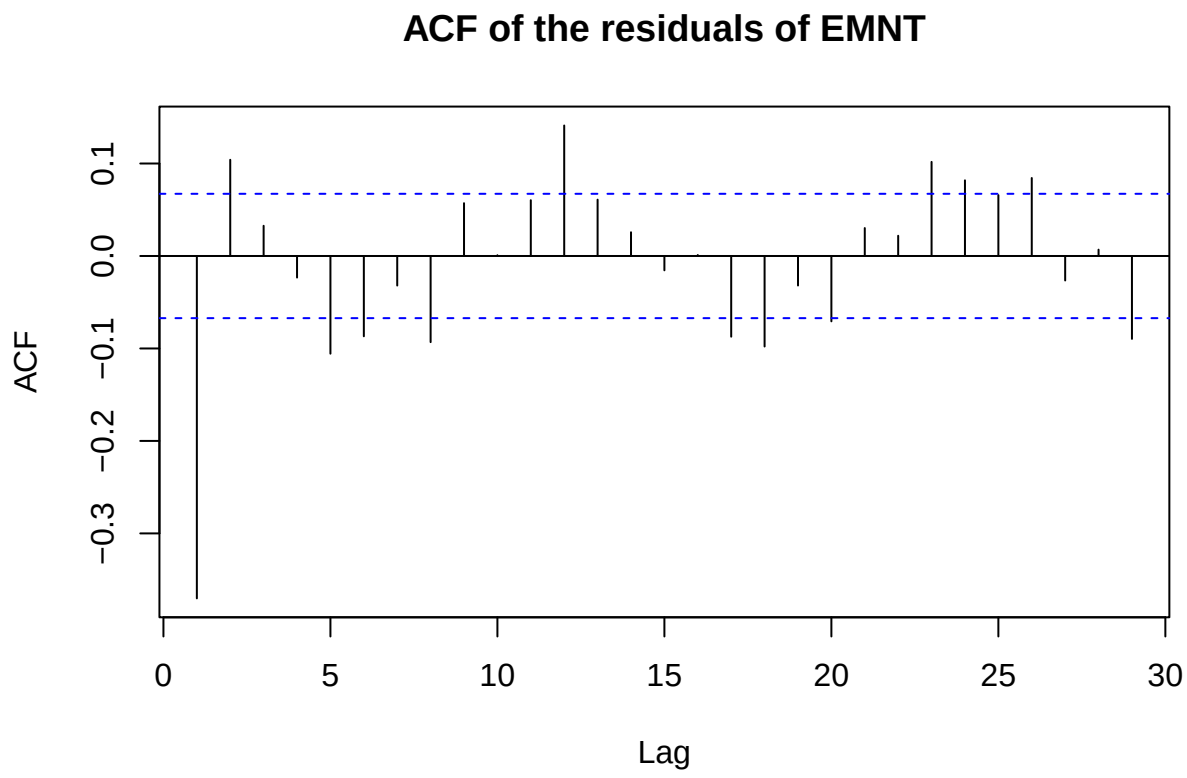
```
## [1] 15.64418
```

```
print(mean(fitvar$varresult$TAVG$residuals^2))
```

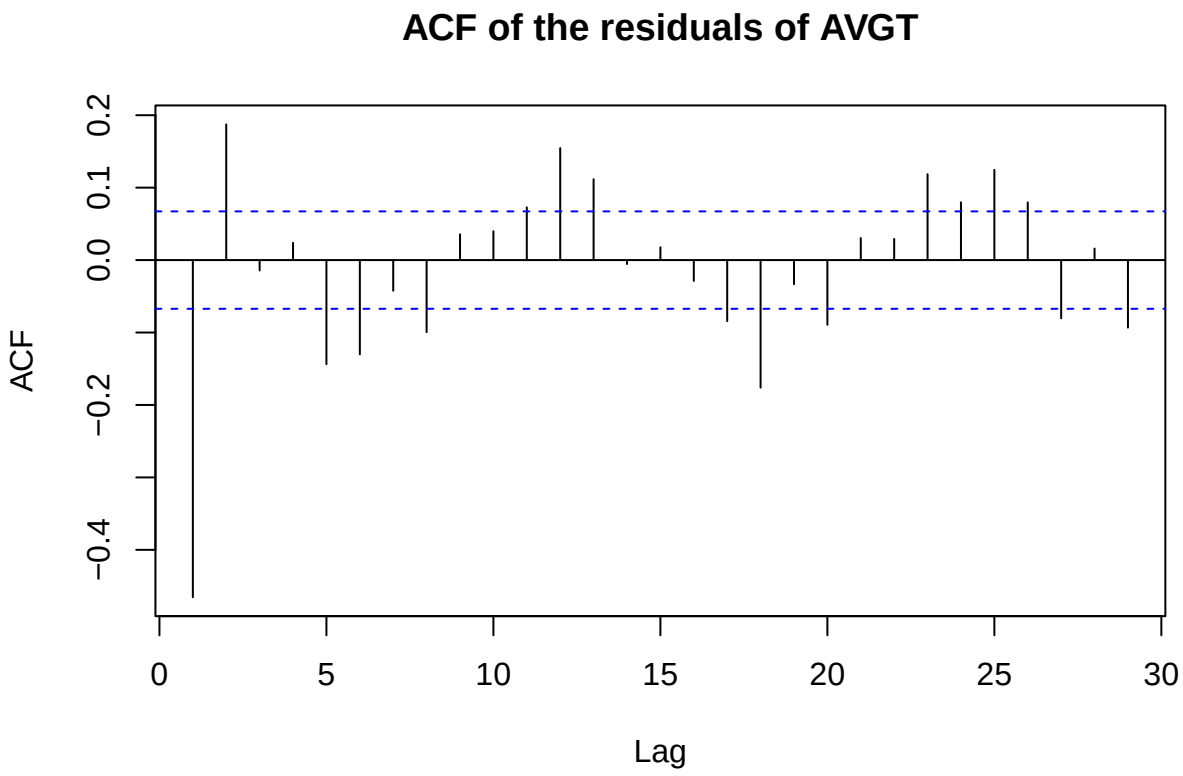
```
## [1] 8.266596
```

We can see a strong correlation between the two variables which suggests that there is a relationship to study between these two variables. We already saw this correlation in the plot, but we are now formalizing this notion. We can further investigate our data by plotting the autocorrelation function for both residuals:

```
#We plot the acfs:
acf(residuals(fitvar)[,1], main = 'ACF of the residuals of EMNT')
```




```
acf(residuals(fitvar)[,2], main = 'ACF of the residuals of AVGT')
```



We can see the striking seasonal behavior of our data: in lags which are multiples of 12 we can see non-trivial contributions to the ACF. In order to mitigate this effect we should fit a VARIMA model. However, this kind of process is well beyond the scope of this report.

Nonetheless, it is interesting to try to find the best *VAR* process possible. In our case let us try to fit the model that will minimize the sum of squared residuals trying different values for p:

```
#We initialize our results
MSE <- 1000
opt_p <- 0
#We find the optimal p
for(p in 1:20){
  #We fit the model
  model <- VAR(na.omit(var_train), p = p)
  # perform forecast
  pred = predict(model,n.ahead=204)
  # Compute the MSE
  #MSE1 <- mean(model$varresult$EMNT$residuals^2)
  MSE1 <- mse(var_test$EMNT, pred$fcst$EMNT[1,])
  #MSE2 <- mean(model$varresult$TAVG$residuals^2)
  MSE2 <- mse(var_test$TAVG, pred$fcst$TAVG[11])
  # If we reduce the MSE we update our parameters
  if(MSE1 + MSE2 <= MSE){
    opt_p <- p
    MSE <- MSE1 + MSE2
    best.model = model
  }
```

```
}
#We print the optimal parameters
print(opt_p)
```

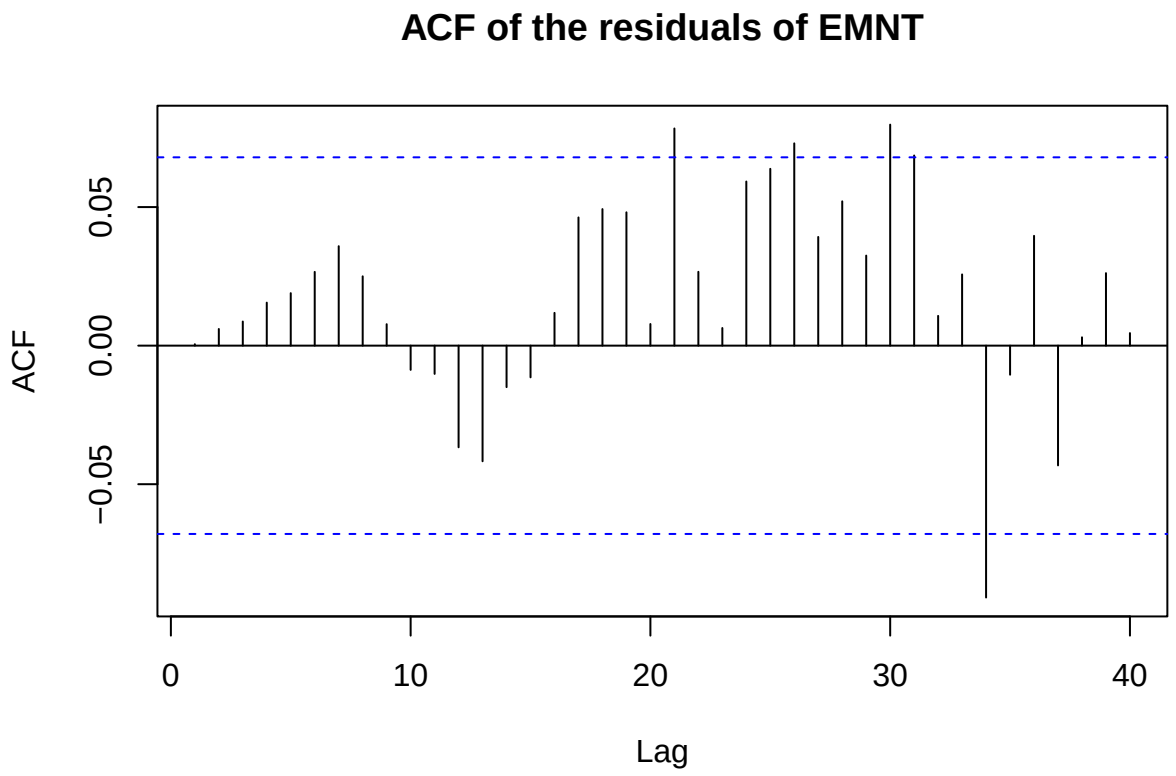
```
## [1] 4
```

```
print(MSE)
```

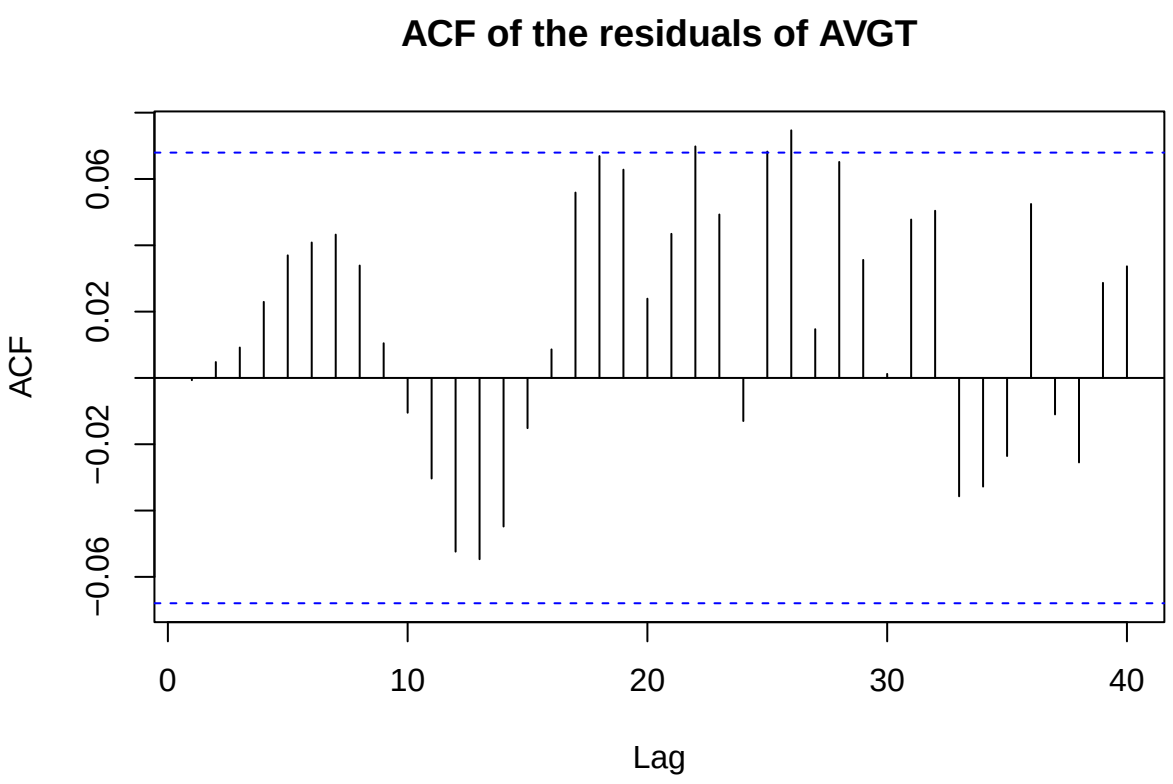
```
## [1] 377.9493
```

We can see that the optimal model is the one which $p = 20$. We can try to plot its ACF structure and see if we have made any improvement:

```
#We plot the acfs:
acf(residuals(model)[,1], lag.max = 40, main = 'ACF of the residuals of EMNT')
```




```
acf(residuals(model)[,2], lag.max = 40, main = 'ACF of the residuals of AVGT')
```



We have improved somewhat our fit. However, this is only true up to the twentieth lag, and this apparently fails for the "EMNT" variable. We can still see lags at higher multiples of 12, so we have still failed to explain our process. Moreover, this suggest that we only reduced the *MSE* due to overfitting our data. Let us check if we could further reduce it by simply increasing the number of variables:

```
new_fit <- VAR(na.omit(var_train), p = 50)
# produce new forecasts
pred = predict(new_fit,n.ahead=204)
#Compute the MSE
MSE1 <- mse(var_test$EMNT, pred$fcst$EMNT[1,])
MSE2 <- mse(var_test$TAVG, pred$fcst$TAVG[11])

print(MSE)
```

```
## [1] 377.9493
```

```
print(MSE1 + MSE2)
```

```
## [1] 442.2961
```

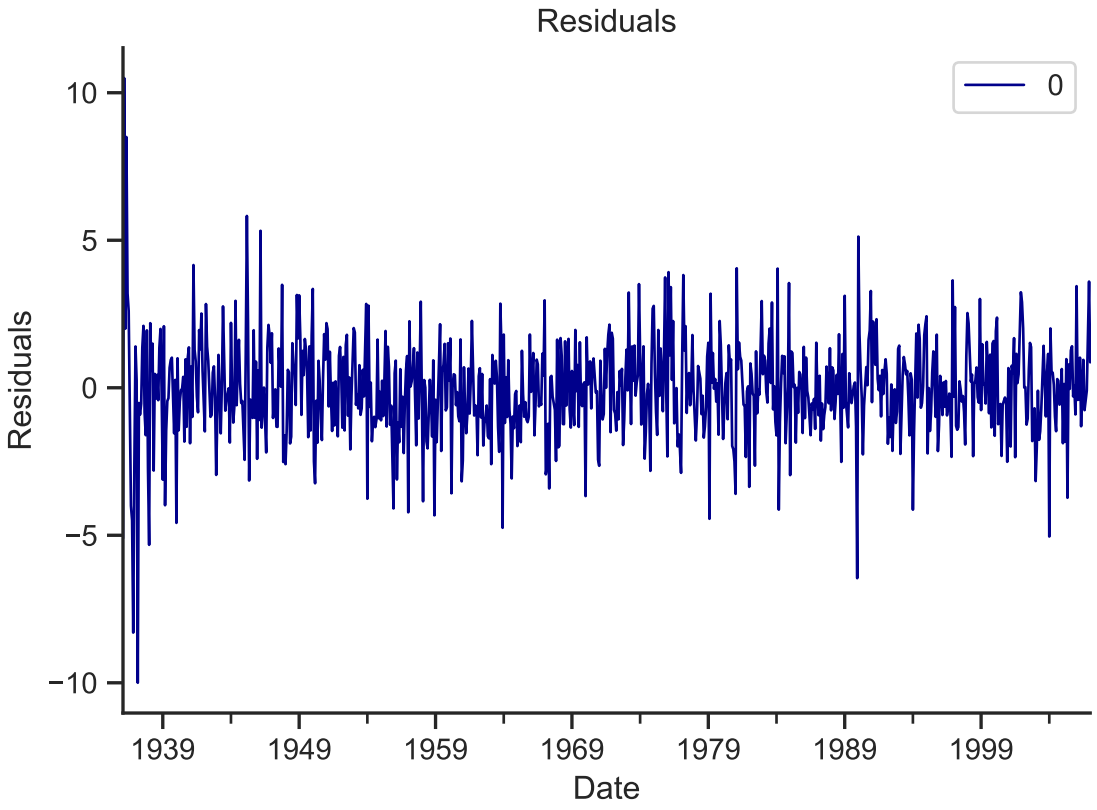
As we can see increasing the number of lags considered does not decrease the *MSE*. In any case, the model will always fail to explain the true nature of the data due to the lack of a way of accounting for seasonality. In conclusion, while we have been able to formalize the relationship between two variables of our dataset, we have been incapable of fully explaining our data with the *VAR* model. Nonetheless, it is apparent that a seasonal vector ARIMA model (seasonal VARIMA) would be more appropriate to deal with this kind of data and we have reason to believe that it would outperform our ARIMA model (at least in principle). Nonetheless, this topic is way beyond the scope of the present report and remains an interesting direction of further research.

7. Conclusions

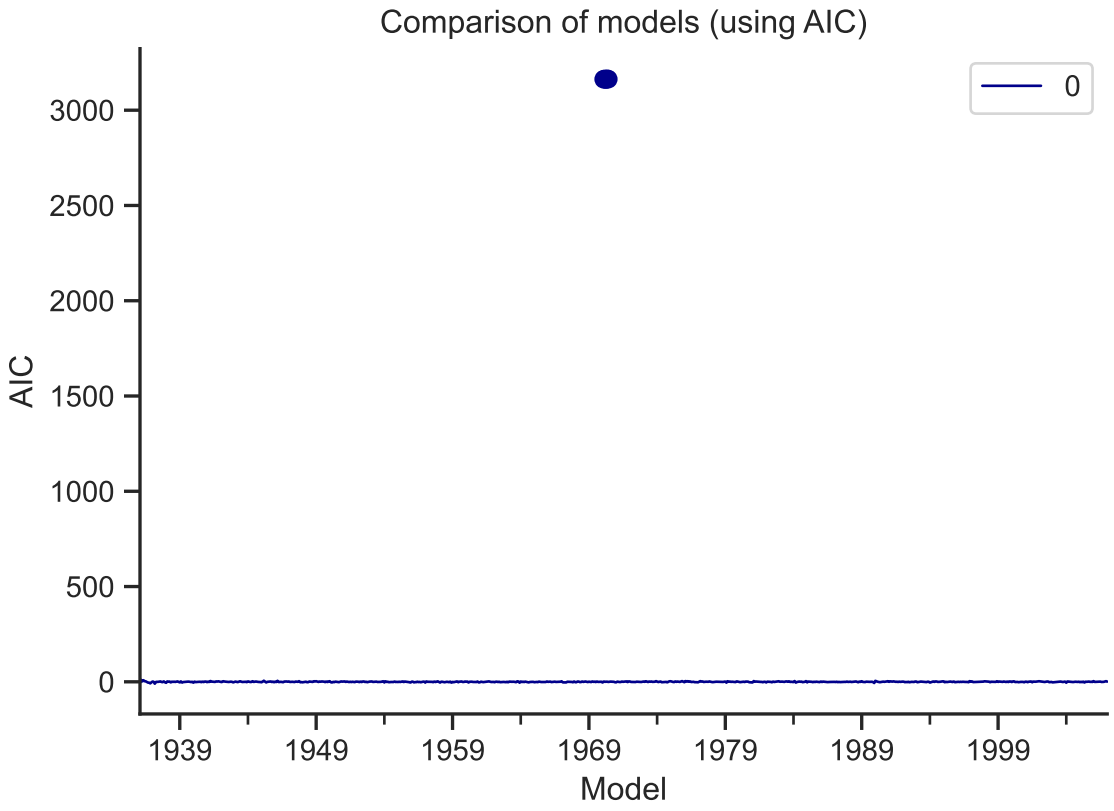
In this report we have continued our analysis of the temperature data at a certain location. In particular, we have successfully built a seasonal ARIMA with the seasonal parameter $s = 12$, seasonal ARIMA order $(4, 1, 1)$ and ARIMA components $(5, 1, 2)$. Furthermore, we can check the parameter estimation:

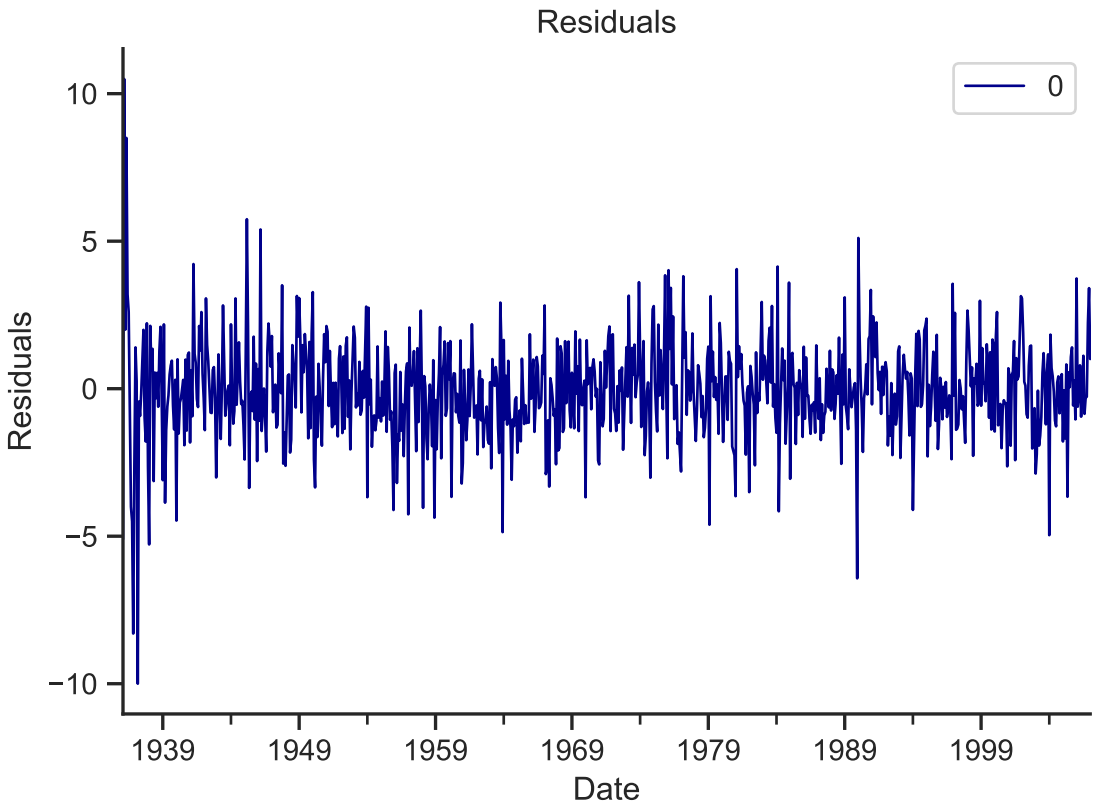
```
print(model5_fit)
```

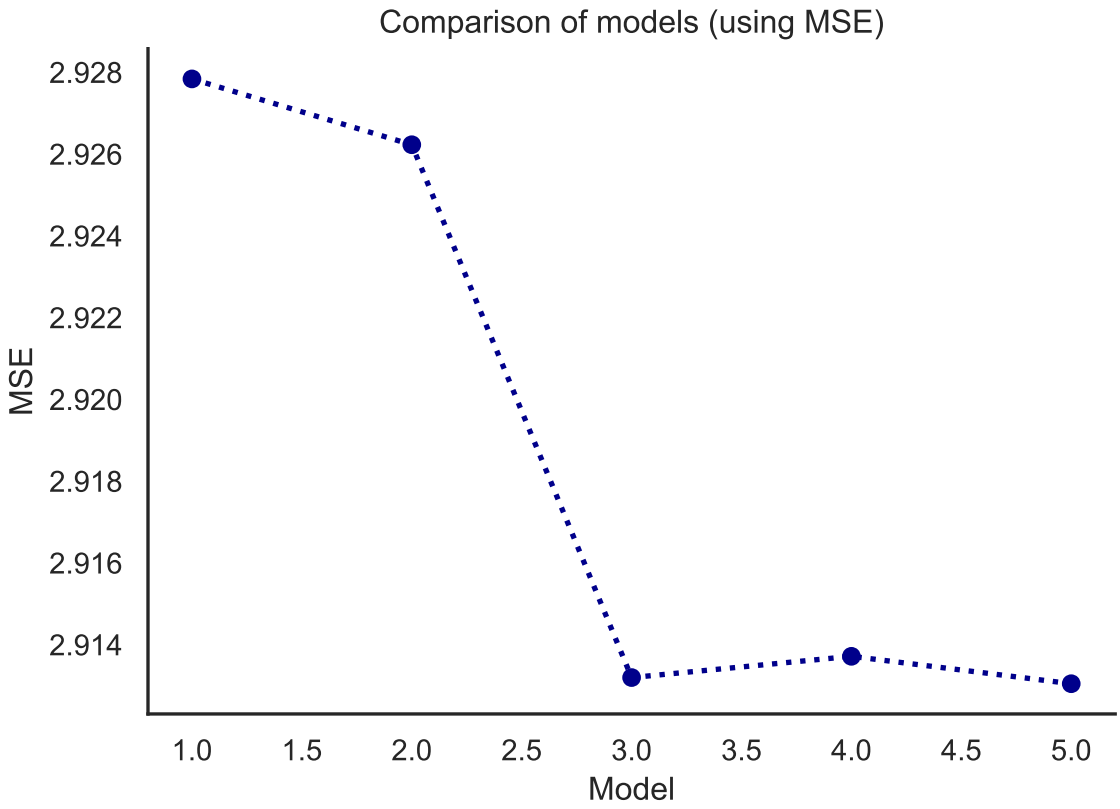
```
## <statsmodels.tsa.arima.model.ARIMAResultsWrapper object at 0x000001FB9A215F10>
```



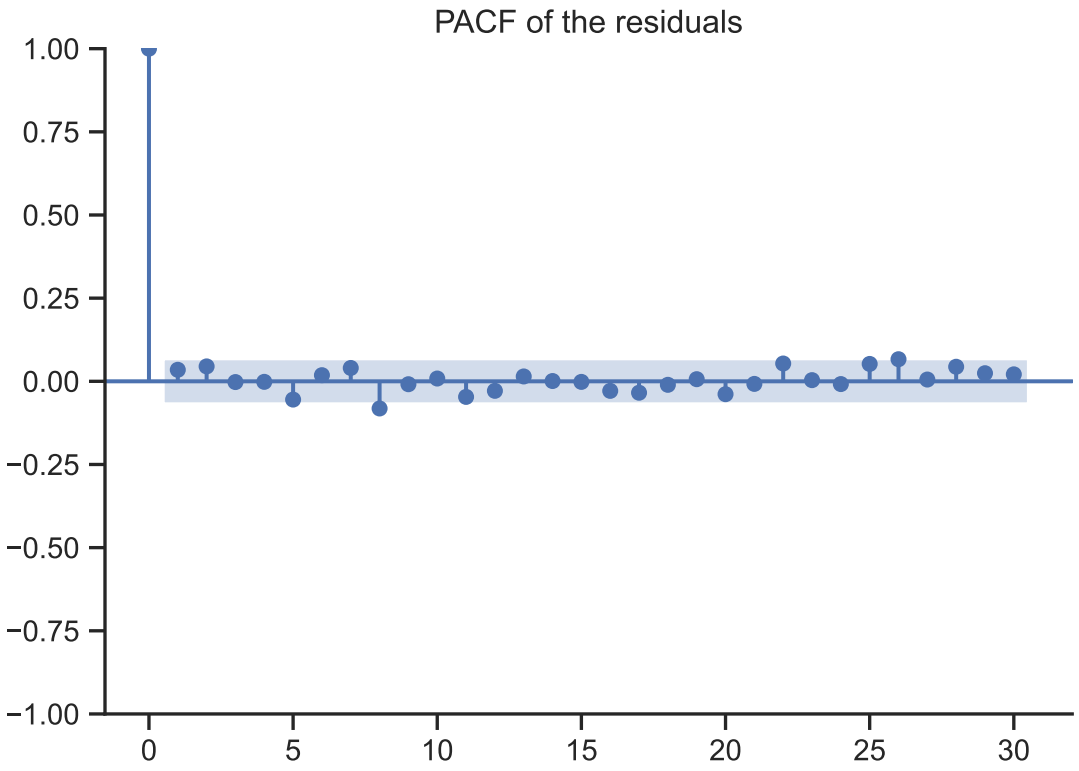
In order to select this model we used the AIC and MSE as criterion to choose among a series of models. The performance of each model is summarized in the following figures:

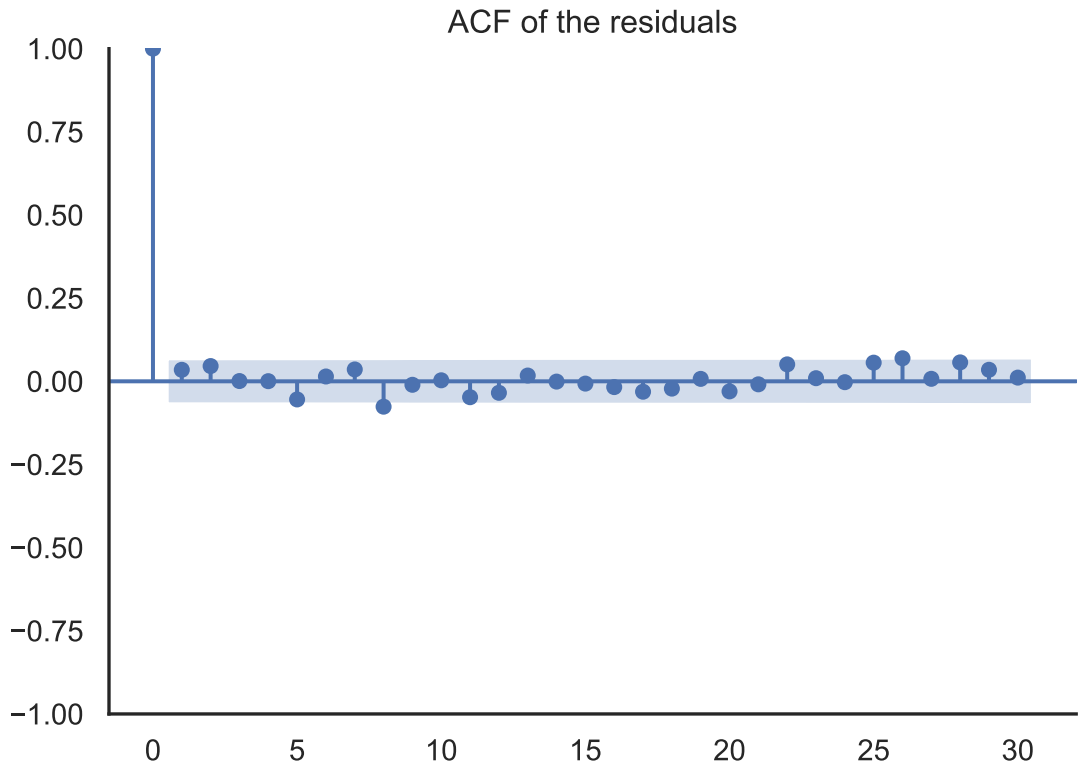

Comparison of models (using AIC)
0
3000
2500
2000
1500
1000
500
0
AIC
1939
1949
1959
1969
1979
1989
1999
Model


Residuals
0
10
5
0
−5
−10
Residuals
1939
1949
1959
1969
1979
1989
1999
Date



Moreover, we checked the structure of its residuals to see that they truly are white noise:


PACF of the residuals
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30

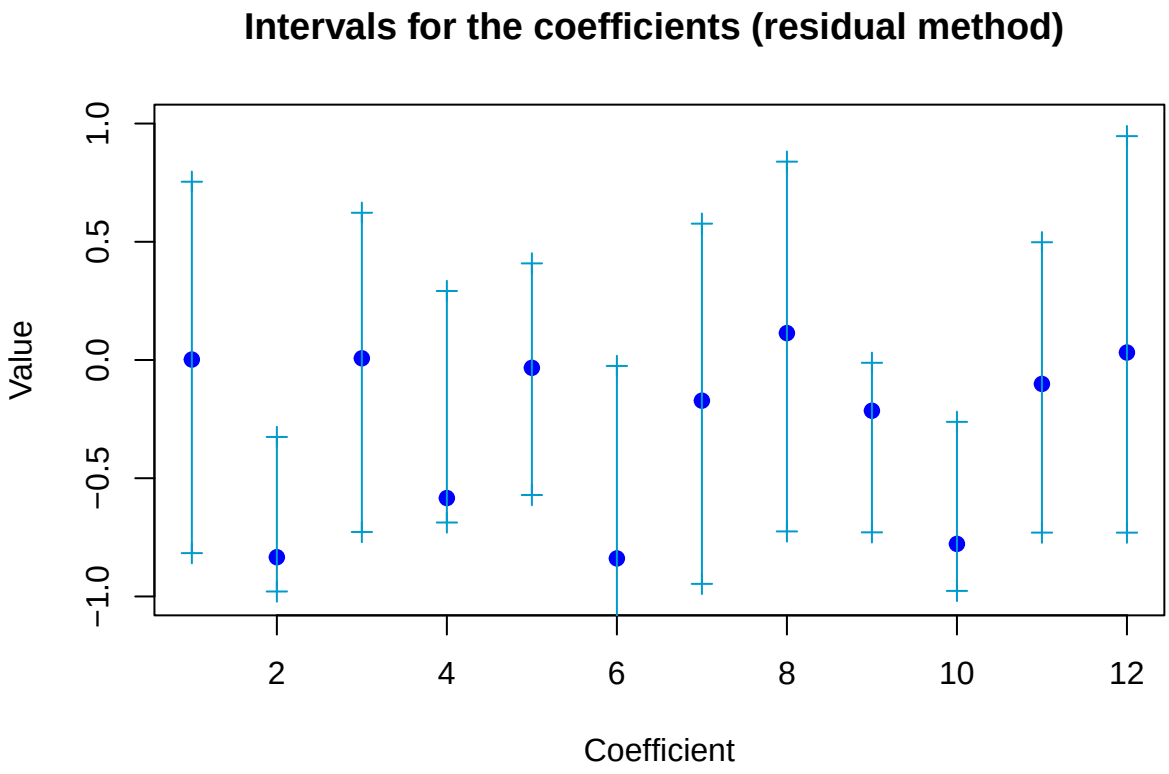


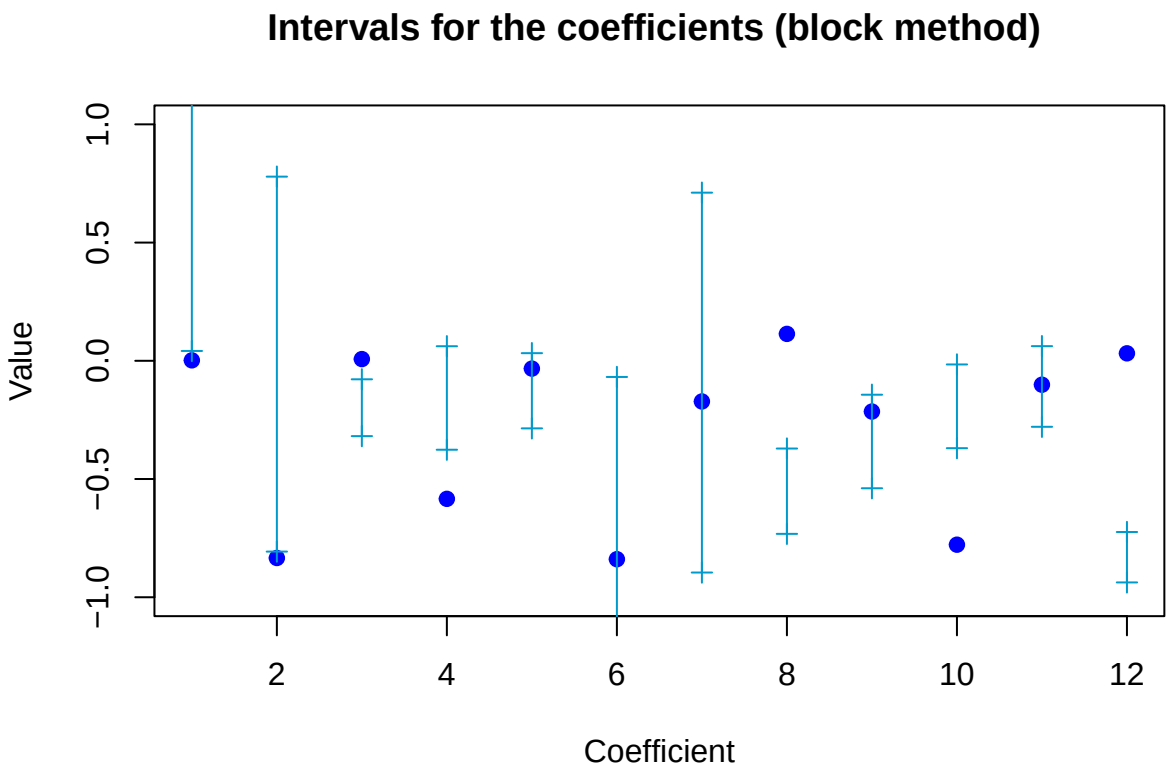
Furthermore, we have used bootstrap methods to compute the coefficients and 95% confidence intervals for them:

Coefficient	Estimation	Confidence interval (residuals)	Standard error (residuals)	Confidence interval (blocks)	Standard error (blocks)
ar1	0.0019262	-0.8167675, 0.7539867	0.4176625	0.0415923, 1.6502458	0.4535953
ar2	-0.8340711	-0.9791047, -0.3255384	0.1659245	-0.8069635, 0.7788722	0.4469846
ar3	0.0072737	-0.7273841, 0.6226518	0.3632027	-0.3185923, -0.0782716	0.0633392
ar4	-0.583838	-0.6873407, 0.2918609	0.2149104	-0.3760752, 0.0616031	0.1210688
ar5	-0.0330444	-0.5710276, 0.4086283	0.2550757	-0.2858104, 0.0322747	0.0823238
ma1	-0.8391194	-1.572784, -0.0251062	0.4150469	-1.7117997, -0.0683274	0.4627636
ma2	-0.1720092	-0.9468385, 0.5766277	0.4073395	-0.8954937, 0.7110665	0.453016
sar1	0.1138083	-0.7249246, 0.8388577	0.4247882	-0.7319119, -0.371008	0.0951104
sar2	-0.2146008	-0.7288806, -0.0118599	0.1801113	-0.5390723, -0.1434435	0.1023454
sar3	-0.7777908	-0.9765746, -0.2616597	0.1800559	-0.3696084, -0.0154268	0.0968048

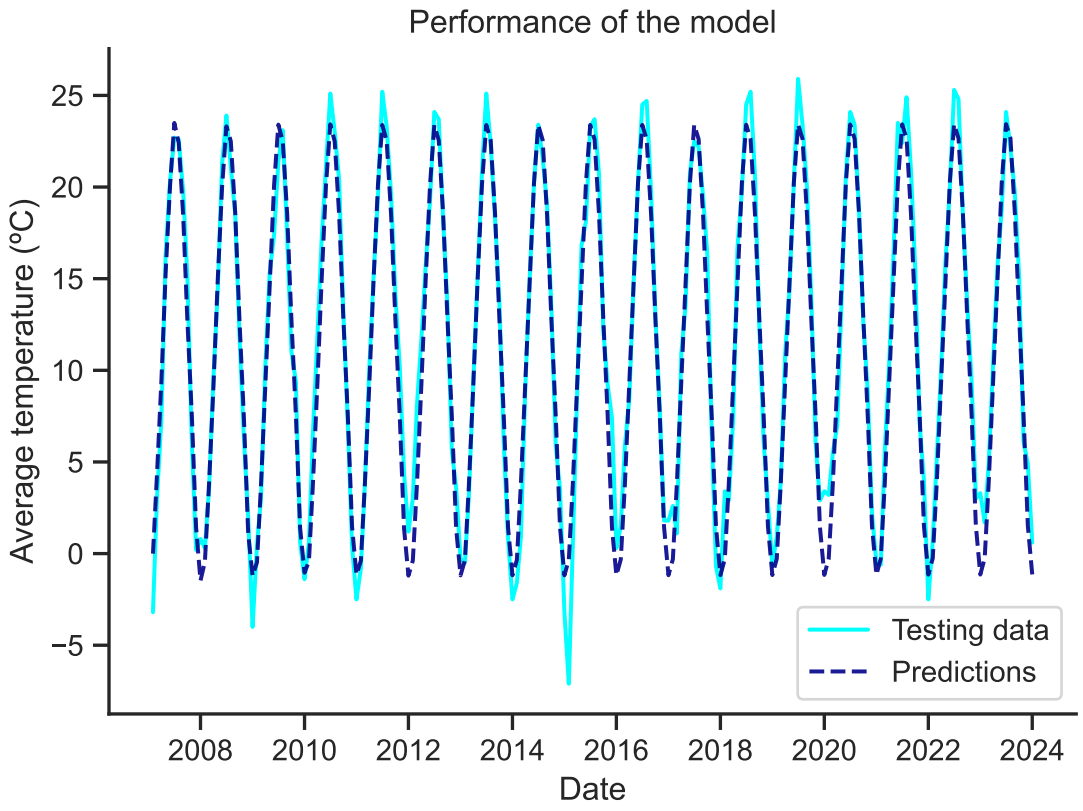
Coefficient	Estimation	Confidence interval (residuals)	Standard error (residuals)	Confidence interval (blocks)	Standard error (blocks)
sar4	-0.101208	-0.7300498, 0.4980051	0.3351344	-0.279184, 0.0619169	0.0862548
sma	0.0315341	-0.7301373, 0.9467801	0.4734657	-0.937296, -0.7240353	0.0643449

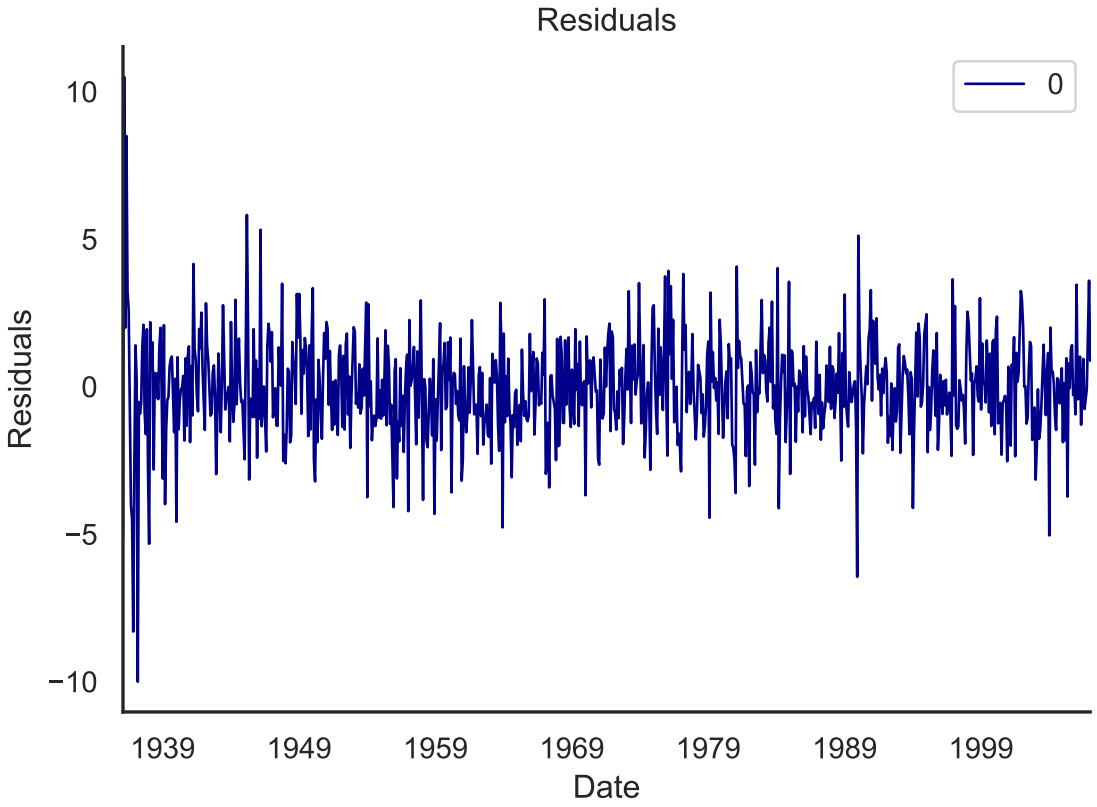
This table can be easily visualized with the following figures:





With this estimations we were able to make predictions which can be visualized in the following figure:


Performance of the model
Average temperature (°C)
25
20
15
10
5
0
−5
2008
2010
2012
2014
2016
2018
2020
2022
2024
Date
Testing data
Predictions



Finally, we have explained the main ideas behind *Vector Autoregressive Models* and applied them to our data. We have succesfully established a model that relates the average temperature with the minimum, but we have failed to fully explain the behavior of the process. The reason behind this is that *VAR* models do not account for seasonality. This is why we have proposed the application of a seasonal VARIMA model as a way of building upon our results. Nonetheless, we have managed to formalize the r elationship between the two variables, which we could not have done with the regular ARIMA model alone.