Task 1

Alejandro Macías Pastor & Arturo Pérez Peralta

2024-02-26

1. Introduction and objectives

The output of man-made greenhouse emissions into the atmosphere has increased meteorically in the last 150 years as a consequence of the Industrial Revolution. This has led to, among other things, a gradual increase in the global average temperature. This warming, along with its potential effects on the global climate, is usually known as Anthropogenic Climate Change. It is feared that this phenomenon will have terrible long term consequences for the global economy and the wellfare of humankind.

This is why there is a big incentive to study meteorological variables through the tools of time series. In particular, weather scientists are 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.

In this report we will use standard time series' techniques 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. Our objectives with this dataset are the following:

1. Perform a basic Exploratory Data Analysis.
2. Use three different models to explain the data. In particular we will fit a Holt-Winters model, a harmonic analysis and a neural network.
3. Compare between the models to see which fits our data the best.

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

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

```
## 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
```

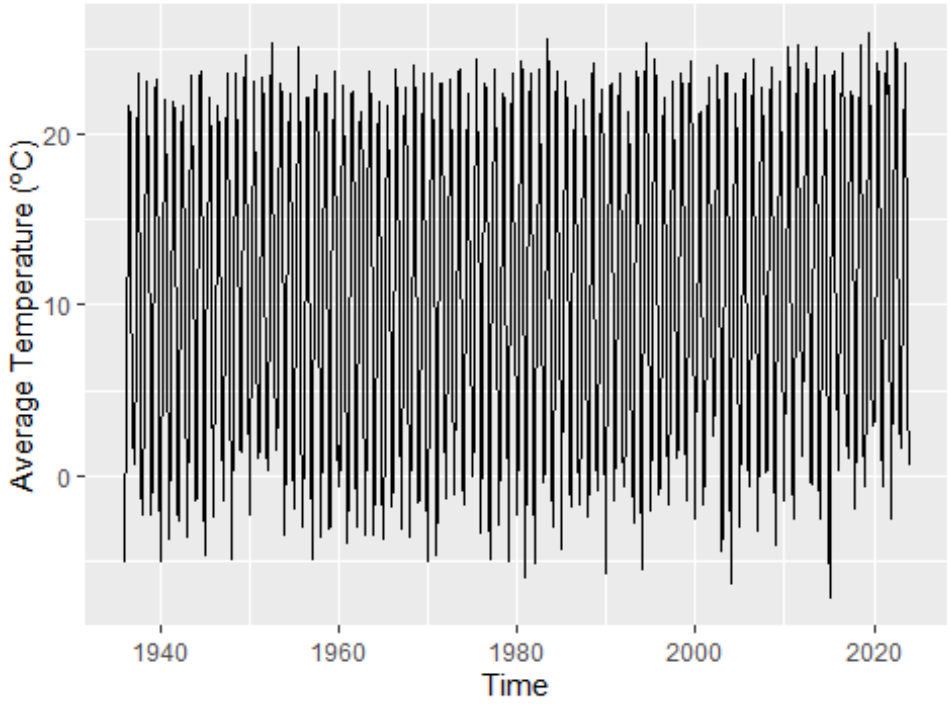
```
set.seed(12345) # set seed for reproducibility
```

```
# 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

#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
```

And we can start by loading or dataset and transforming it into a `ts` format and simply plotting the time series to get a feel for 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
```



At a glance, we observe a cyclical behavior that falls in line with what one would expect from temperature (which varies in an expectable pattern according to the seasons). From looking to the `Y` axis we can see that temperature oscillates between sub zero temperatures in winter (the lowest ones being around $-5°C$) and $20°C$ in summer, which is to be expected from the location we are studying. We are now ready to begin the proper study of our data.

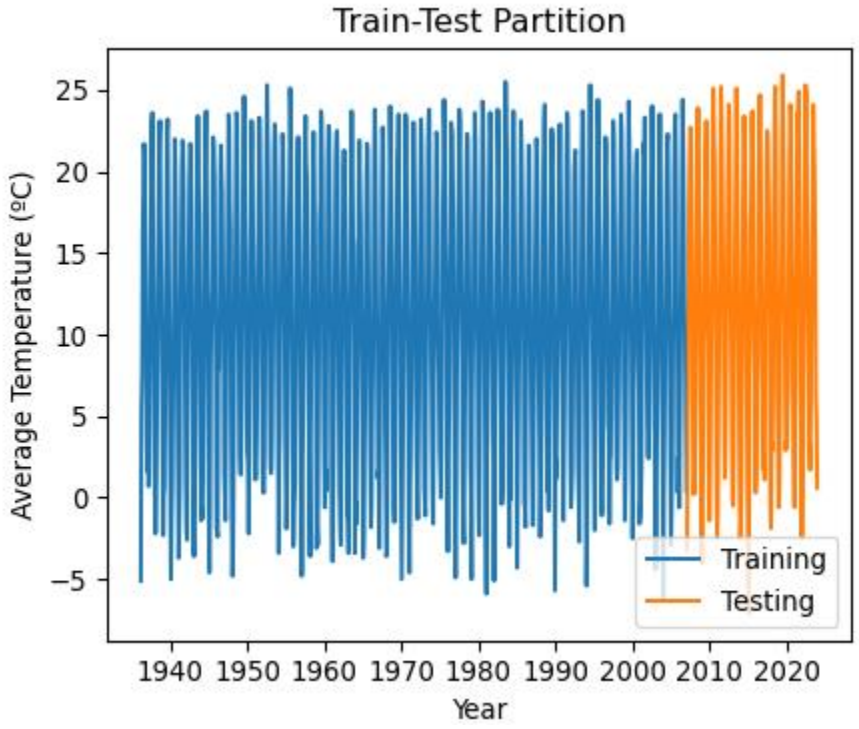
2. Partition of the data and Exploratory Data Analysis

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
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") #Plot training set
plt.plot(test,label="Testing") #Plot testing set
plt.xlabel("Year")
plt.ylabel("Average Temperature (ºC)")
plt.title("Train-Test Partition")
plt.legend()
plt.show()
```

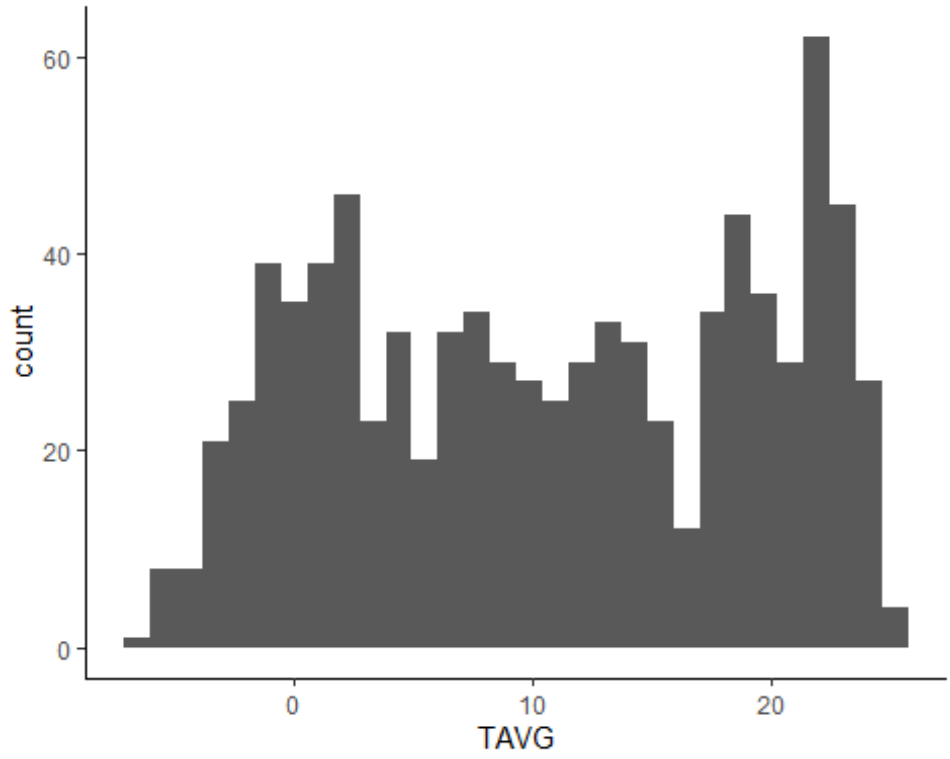


We can now perform a really basic Exploratory Data Analysis as a sanity check before we proceed. For starters we can check the proportion of missing values:

```
#We see how many missing values there are in our data
print(colSums(is.na(data_train)))

## DATE TAVG
##    0    0
```

As we can see our data has no missing values, which means that we do not need to delve into imputation methods of any kind. We now plot a histogram of the average temperature in order to get a feel for its distribution:

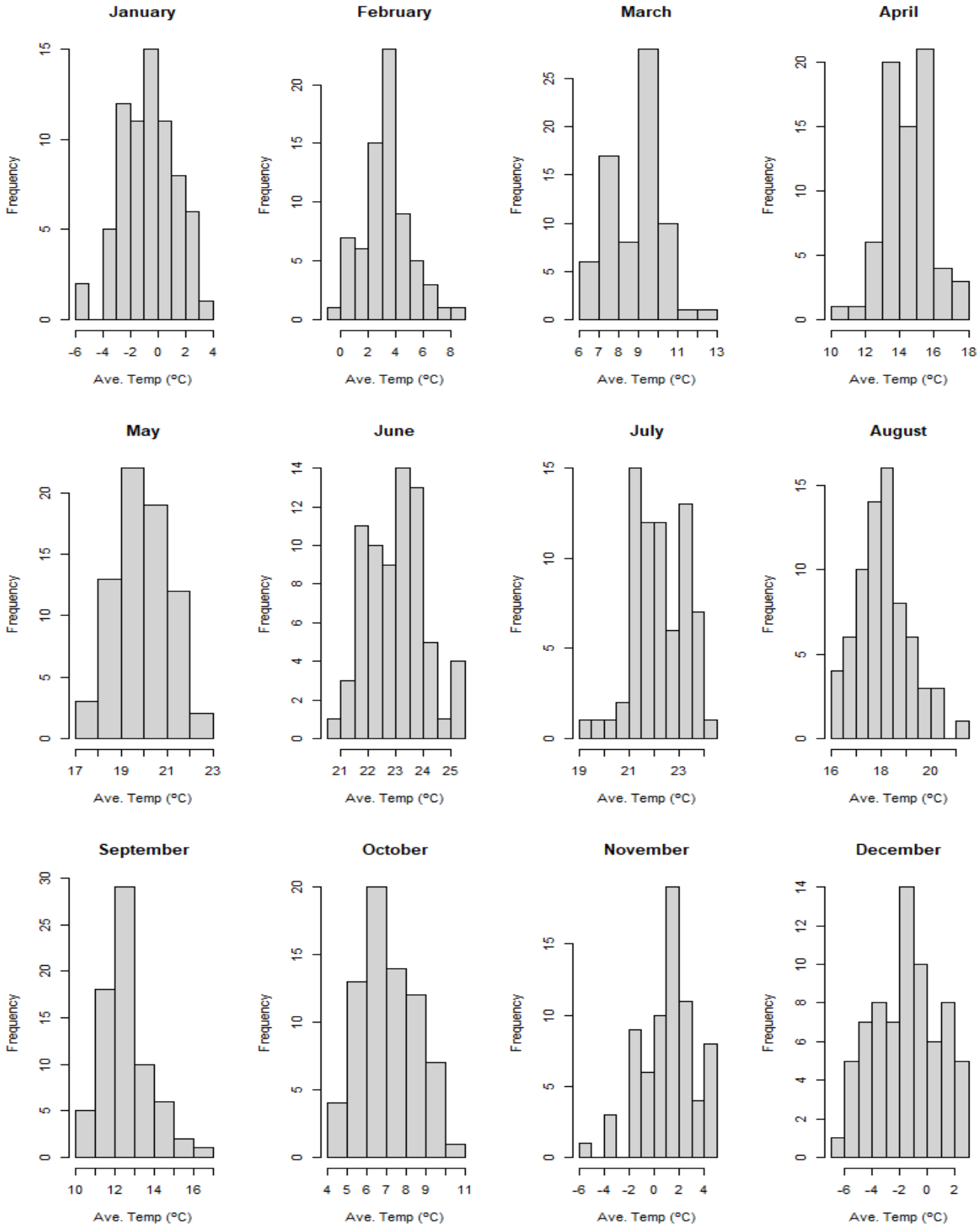
```
#We plot a histogram of the average temperature
ggplot(data = data_train, mapping = aes(x = TAVG)) +
  geom_histogram() +
  theme_classic()

## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



At a glance we observe that temperatures are kind of uniformly distributed which falls in line with what one could expect from the cyclical nature of the weather. However, there seems to be a bias towards higher temperatures, although it remains to be seen if such bias can be corroborated by the tools from time series.

However, it is interesting to check the distribution of temperatures among all months, which yields the following graphics:

```
months <- c('January', 'February', 'March', 'April', 'May', 'June',
            'July', 'August', 'September', 'October', 'November',
'December')
par(mfrow=c(3,4))

for (i in 1:12){
  hist(data_train$TAVG[seq(i,length(data_train$TAVG),12)],
main=months[i], xlab='Ave. Temp (ºC)')
}
```


January
Frequency
Ave. Temp (°C)
February
Frequency
Ave. Temp (°C)
March
Frequency
Ave. Temp (°C)
April
Frequency
Ave. Temp (°C)
May
Frequency
Ave. Temp (°C)
June
Frequency
Ave. Temp (°C)
July
Frequency
Ave. Temp (°C)
August
Frequency
Ave. Temp (°C)
September
Frequency
Ave. Temp (°C)
October
Frequency
Ave. Temp (°C)
November
Frequency
Ave. Temp (°C)
December
Frequency
Ave. Temp (°C)

As we can see, the temperature distribution of each month is more “normal like”, with extreme temperatures being rarer than central values.

3. Descriptive techniques

We are now ready to begin applying descriptive techniques to our dataset. These kind of tools aim to explain the past evolution and predict the future of a given series by identifying simple patterns such as trends, cycles or seasonal variation. We will use three descriptive techniques to study our temperature time series: Holt-Winters, harmonic analysis and neural networks.

3.1 Holt-Winters model

The Holt-Winters model is a method proposed for time series that show both trends and seasonality. It is made up of three smoothing equations: one for level, one for trend and one for seasonality. Due to the characteristics of our time series, we will use the additive formulation of model, which is defined by the following equations:

$$\hat{z}_{t+h} = a_t + hb_t + S_{t-p+(h-1) \bmod p}$$

where a_t is the level, b_t the trend, and S_t the season. These three terms are updated following the equations shown below:

$$a_t = \alpha\left(z_t - S_{t-p}\right) + (1-\alpha)(a_{t-1} + b_{t-1})$$

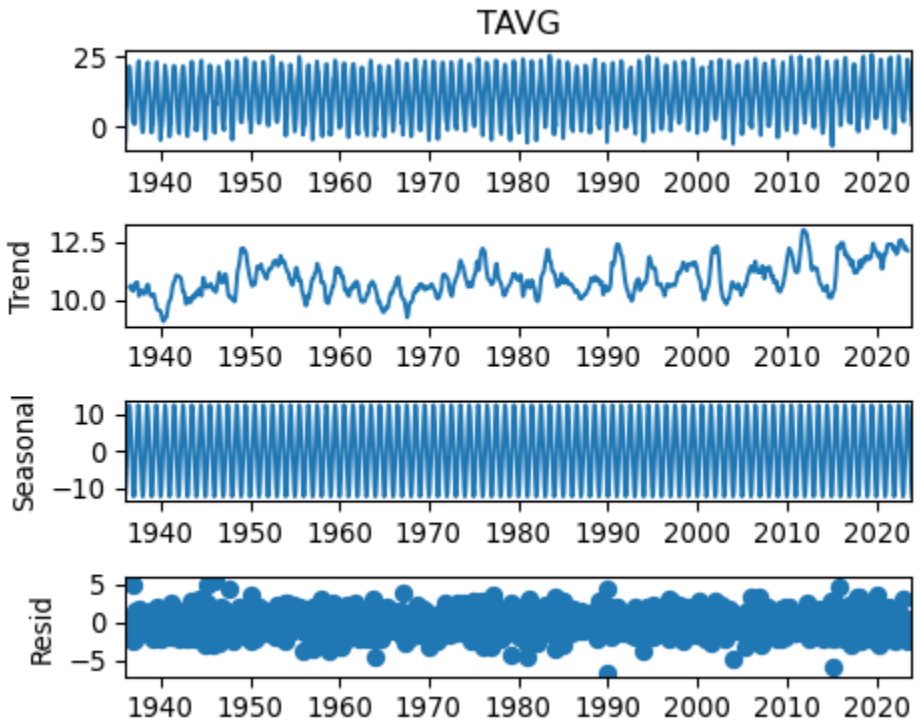
$$b_t = \beta(a_t - a_{t-1}) + (1-\beta)b_{t-1}$$

$$S_t = \gamma(z_t - a_t) + (1-\gamma)S_{t-p}$$

where α, β and γ are the three smoothing parameters.

The `statsmodel` library in Python contains functions that greatly facilitate the computation of both the decomposition and the parameters of the Holt-Winters model. For the time series at hand:

```
#We compute the decomposition and plot it
holtwin = seasonal_decompose(data["TAVG"], model='additive')
plt.clf()
holtwin.plot()
plt.show()
```



From this decomposition we can see how there is a clear seasonal component to the series, which might correspond to the yearly cycles of climate seasons. The trend on the other hand is less clear, showing a somewhat erratic behavior. Nonetheless, it seems to be slightly positive, especially when comparing the data from the 70s to present readings.

Let us now compute the specific parameters we previously mentioned:

```
#We define the model and use the fit method
model = ExponentialSmoothing(train["TAVG"], seasonal_periods=12,
trend='add',
seasonal='add')
fit = model.fit()

#We print our results
print("Model parameters: \n", fit.params)
```

```
## Model parameters:
##  {'smoothing_level': 0.0306840191910907, 'smoothing_trend': 0.0,
'smoothing_seasonal': 0.0, 'damping_trend': nan, 'initial_level':
10.201664079709708, 'initial_trend': 0.001102727052983609,
'initial_seasons': array([-11.40641816,  -7.33569425,  -1.80581499,
3.78745665,
##          9.13423738,  12.26975167,  11.51091826,   7.37881936,
##          1.80165701,  -3.60931572,  -9.64137827, -12.30304552]),
'use_boxcox': False, 'lamda': None, 'remove_bias': False}
```

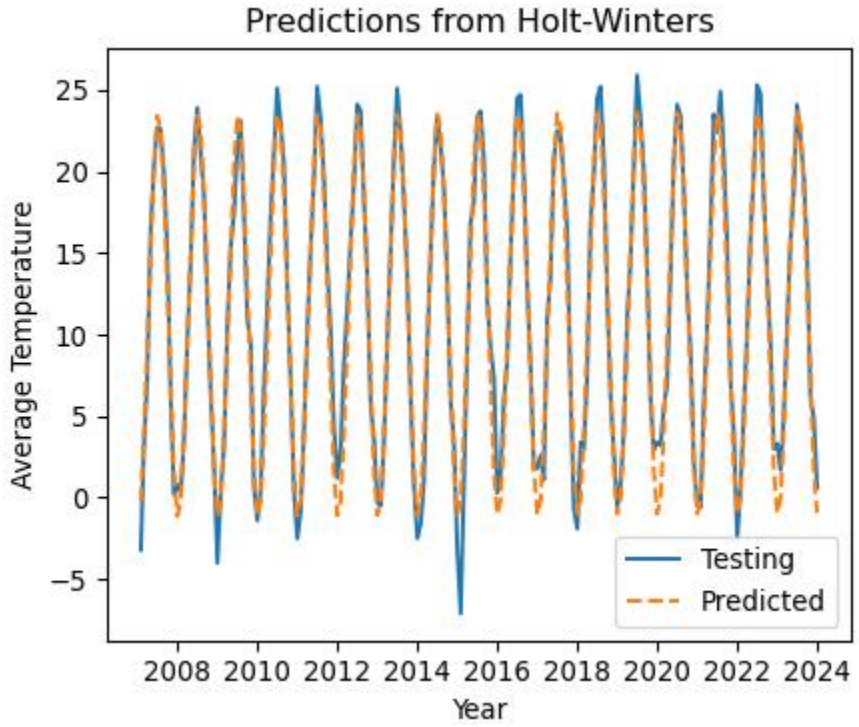
```
#We specifically print the parameters we previously defined
print(f"The estimated alpha is: {fit.params['smoothing_level']:.3f}")

## The estimated alpha is: 0.031

print(f"The estimated beta is: {fit.params['smoothing_trend']:.3f}")

## The estimated beta is: 0.000

print(f"The estimated gamma is: {fit.params['smoothing_seasonal']:.3f}")

## The estimated gamma is: 0.000
```

We can now use these estimated parameters to predict the future evolution of the time series:

```
#We make our predictions
pred_hw = fit.predict(start=test.index[0], end=test.index[-1])
#We store them in a data frame
pred_hw = pd.DataFrame(pred_hw, index=test.index,
columns=["PredictedTAVG"])
```

In order to visually test the performance of our model we can plot its predictions along with the testing series:

```
plt.clf()
#We plot the testing set
plt.plot(test['TAVG'],label='Testing')
#We plot our predictions
plt.plot(pred_hw, label='Predicted',linestyle='--')
plt.xlabel("Year")
plt.ylabel("Average Temperature")
plt.title("Predictions from Holt-Winters")
plt.legend()
plt.show()
```



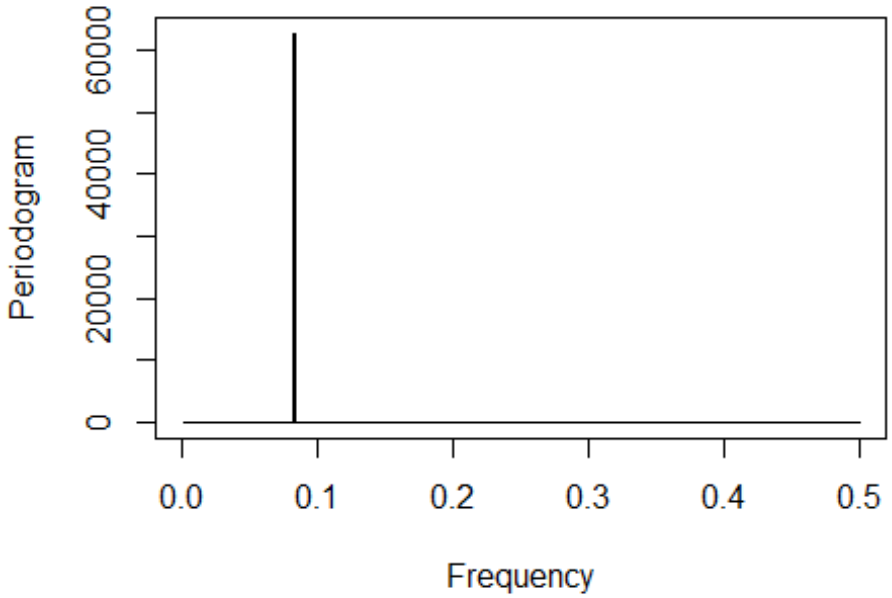
We can see in the plot that the predictions given by the Holt-Winters model are able to successfully capture the seasonal behavior of the time series, as well as its level and trend. However it is not perfect as it fails to predict the specific shapes of the peaks, which seem to follow a more chaotic pattern. Nonetheless, this is to be expected and we will later measure how much the model differs from reality.

3.2 Harmonic analysis

Harmonic analysis is based on Fourier's ideas that any periodic function can be decomposed as a sum of pure sine waves. In our case we will perform a Fourier transform on our training data to get this decomposition, which will serve as a model to predict the testing set.

For starters, we plot the periodogram of the training set, which allows us to visualize the contribution of each frequency:

```
#We plot the preiodogram
peri = periodogram(series_train)
```



We can see that the main contribution to the spectrum of the data is that of a single frequency that falls shy of 0.1 months^{-1}. We will use this frequency as a reference to determine the order of the Fourier decomposition, which we are now ready to perform:

```
#We transform the maximum contribution to period units
period.max = 1/peri$freq[which.max(peri$spec)]
#We compute the Fourier transform
taylor.lm = tslm(series_train ~ fourier(series_train, K=period.max/2))
#We make our prediction
taylor.fcast = forecast(taylor.lm,
                        data.frame(fourier(series_train, K=period.max/2,
h=204)))
prediction_har = taylor.fcast$mean
```

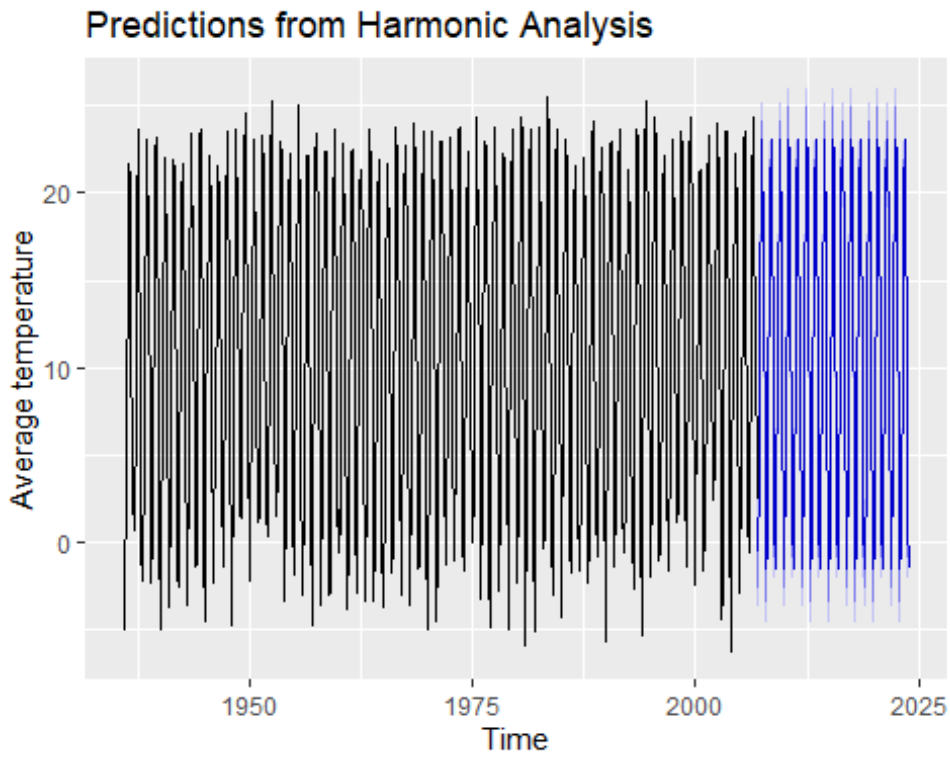
Since the `tslm` function is nothing but a wrapper of the `lm` function designed specifically for time series, we can check the parameters estimated by the model:

```
summary(taylor.fcast$model)

##
## Call:
## tslm(formula = series_train ~ fourier(series_train, K = period.max/2))
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -6.8099 -0.9377 -0.0268  0.9211  4.9944
```

```
##
## Coefficients:
##                                              Estimate Std. Error  t
value
## (Intercept)                                 10.728052   0.053138
201.890
## fourier(series_train, K = period.max/2)S1-12  -2.006728   0.075148  -
26.704
## fourier(series_train, K = period.max/2)C1-12 -11.954832   0.075148 -
159.083
## fourier(series_train, K = period.max/2)S2-12   0.066883   0.075148
0.890
## fourier(series_train, K = period.max/2)C2-12  -0.055751   0.075148   -
0.742
## fourier(series_train, K = period.max/2)S3-12  -0.090141   0.075148   -
1.200
## fourier(series_train, K = period.max/2)C3-12  -0.409155   0.075148   -
5.445
## fourier(series_train, K = period.max/2)S4-12   0.107135   0.075148
1.426
## fourier(series_train, K = period.max/2)C4-12   0.009977   0.075148
0.133
## fourier(series_train, K = period.max/2)S5-12   0.109545   0.075148
1.458
## fourier(series_train, K = period.max/2)C5-12   0.080888   0.075148
1.076
## fourier(series_train, K = period.max/2)C6-12   0.050117   0.053138
0.943
##                                                Pr(>|t|)
## (Intercept)                                     < 2e-16 ***
## fourier(series_train, K = period.max/2)S1-12    < 2e-16 ***
## fourier(series_train, K = period.max/2)C1-12    < 2e-16 ***
## fourier(series_train, K = period.max/2)S2-12      0.374
## fourier(series_train, K = period.max/2)C2-12      0.458
## fourier(series_train, K = period.max/2)S3-12      0.231
## fourier(series_train, K = period.max/2)C3-12 6.82e-08 ***
## fourier(series_train, K = period.max/2)S4-12      0.154
## fourier(series_train, K = period.max/2)C4-12      0.894
## fourier(series_train, K = period.max/2)S5-12      0.145
## fourier(series_train, K = period.max/2)C5-12      0.282
## fourier(series_train, K = period.max/2)C6-12      0.346
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.551 on 840 degrees of freedom
## Multiple R-squared:  0.9688, Adjusted R-squared:  0.9684
## F-statistic:  2369 on 11 and 840 DF,  p-value: < 2.2e-16
```

It is easier to visualize our predictions by plotting them just like we did before:

```
#We plot our results
autoplot(taylor.fcast, ylab='Average temperature',
         main='Predictions from Harmonic Analysis')
```



Once more, we can see that our model has successfully replicated the periodic behavior of the data, although its precision will be tested once we compute its mean square error.

3.3 Neural networks

The third and last approach we will use to predict the behavior of the series is one based on neural networks. Specifically, the model considered here is the one known as Long Short-Term Memory (LSTM) network. The main idea behind it is to predict an instance of the series by studying a certain number of previous values. In our particular case every prediction will be made by considering the previous 12 months.

This model's architecture consists of a sequential stack of layers made up of two LSTM layers with 50 neurons each and a Dense layer to finish it off. During training, the model will use mean square error as a loss function and the adam optimizer. Like in the above sections, the model will be trained on the first 70 years of the time series, after which it will be used to predict the last 17 years.

Before training the model, it is convenient to rescale the values of the time series to be in the [0,1] range due to the characteristics of the neural network. For this purpose we will use a MinMax scaler. Furthermore, since each prediction will be made by looking

at the previous 12 values in the series, the train-test partition has to be modified accordingly:

```
#We define the scaler we are going to use
scaler = MinMaxScaler(feature_range=(0,1))
#We fit the scaler with our data and transform it accordingly
train_scaled = scaler.fit_transform(train)
test_scaled = scaler.transform(test)
#We define a new dataset:
#In X_train we store observations in groups of twelve
#In y_train we store the thirteenth observation, which will predicted
#using the previous twelve
X_train, y_train = [], []
for i in range(12, len(train)):
  X_train.append(train_scaled[i-12:i,0])
  y_train.append(train_scaled[i,0])
#We transform our lists into numpy arrays
X_train, y_train = np.array(X_train), np.array(y_train)
#We give our arrays the correct shape for the neural network
X_train = np.reshape(X_train, X_train.shape + (1,))
```

We are now ready to create and fit the model. The data will be processed for 100 epochs in batches of size 32 in order to adjust the model's parameters.

```
#We define the model
model = Sequential([
  LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)),
  LSTM(50),
  Dense(1)
])

#We define the loss function and the optimizer to be used
model.compile(loss='mean_squared_error', optimizer='adam')

#We fit the model with the training data
model.fit(X_train, y_train, epochs=100, batch_size=32, verbose=0)
```

We can now make predictions on the testing set. However, we also need to carefully manipulate it like we did with the training set:

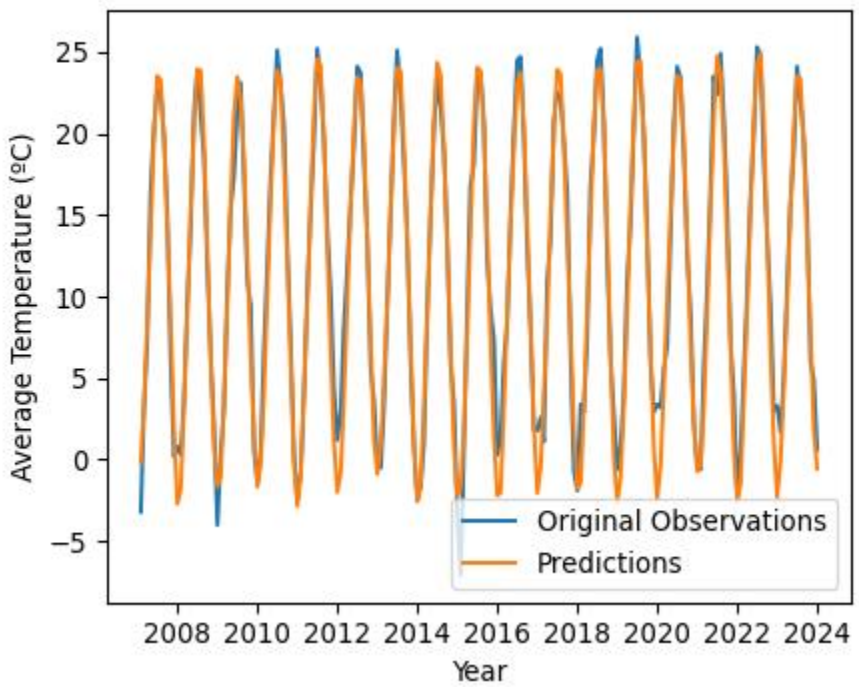
```
#We get the data we are going to use
inputs = data[-216:]
#We transform it using the previously defined scaler
inputs = scaler.transform(inputs)
#We reshape it to the correct shape
inputs = inputs.reshape(-1,1)

#We store the observations of the testing set into groups of twelve like
we did before
X_test = []
for i in range(12,len(inputs)):
```

```
  X_test.append(inputs[i-12:i, 0])
#We store our new list as a numpy array and reshape it accordingly
X_test = np.array(X_test)
X_test = np.reshape(X_test, X_test.shape + (1,))

#We make our predictions
predictions  = model.predict(X_test, verbose=0)
#We invert the transformation we did at the beginning
pred_nn = scaler.inverse_transform(predictions)
#We store the results into a data frame
result_df = pd.DataFrame({'a': test.values[:, 0], 'b': pred_nn[:, 0]})
pred_nn = pd.DataFrame(pred_nn, index=test.index,
columns=['PredictedTAVG'])
print(result_df)
```

```
##         a          b
## 0    -3.2  -0.168099
## 1     3.0   3.206445
## 2     7.3   8.396635
## 3    16.3  13.781871
## 4    20.3  19.771425
## ..    ...        ...
## 199  19.7  19.027700
## 200  15.1  13.692228
## 201   6.1   7.920852
## 202   4.9   1.873790
## 203   0.6  -0.550362
##
## [204 rows x 2 columns]
```

Let us now visualize our predictions:

```
plt.clf()
#plt.plot(train)
plt.plot(test, label="Original Observations")
plt.plot(pred_nn, label="Predictions")
plt.legend(loc='best')
plt.xlabel('Year')
plt.ylabel('Average Temperature (ºC)')
plt.show()
```



Like the previous two models, the LSTM network seems to also be capable of capturing the seasonal tendencies of the series. Nonetheless it has troubles predicting the concrete shape of most peaks and valleys, which might be a result of simple happenstance due to the random nature of weather rather than a fault within the model.

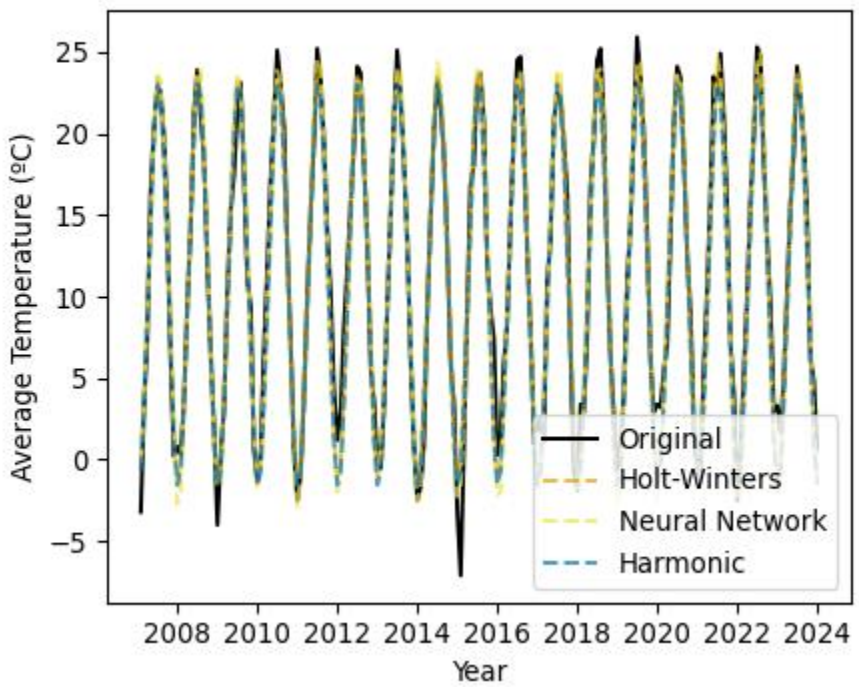
4. Comparison between models

Finally we are going to compare the predictions produced by each model, both visually and quantitatively. For the latter we need to set a performance metric, which in our case will be the mean squared error (which is widely used in regression problems).

We start with a simple visual comparison:

```
# retrieve the harmonic analysis predictions
pred_har = pd.DataFrame(r.prediction_har, index=test.index,
columns=["PredictedTAVG"])
plt.clf()
plt.plot(test['TAVG'], 'k', label='Original')
plt.plot(pred_hw['PredictedTAVG'],'#E69F00', linestyle='dashed',
alpha=0.7,
label='Holt-Winters')
plt.plot(pred_nn['PredictedTAVG'],'#F0E442', linestyle='dashed',
alpha=0.7,
label='Neural Network')
```

```
plt.plot(pred_har['PredictedTAVG'],'#0072B2', linestyle='dashed',
alpha=0.7,
label='Harmonic')
plt.ylabel('Average Temperature (ºC)')
plt.xlabel('Year')
plt.legend(loc='lower right')
plt.show()
```



Although all three models seem to fit the data quite well, the figure is kind of cluttered and it is hard to see which one is best. Therefore, in order to compare our predictions we should use a quantitative criteria, which in our case will be the mean squared error with respecto to the original observations:

```
prediction.hw = py$pred_hw
prediction.nn = py$pred_nn
prediction.har = data.frame(taylor.fcast$mean,
row.names=row.names(data_test))

mse(prediction.hw$PredictedTAVG, data_test$TAVG)

## [1] 2.810815

mse(prediction.nn$PredictedTAVG, data_test$TAVG)

## [1] 3.343529

mse(prediction.har$taylor.fcast.mean, data_test$TAVG)
```

```
## [1] 3.458528
```

As we can see, the Holt-Winters models is the best of the three, which may be surprising considering its simplicity. However, perhaps the neural network could be improved through hyper-parameter tuning (changing, for example, the number of observations we considered from 12 to some other number which may improve our results) or through changing its architecture (adding more layers or neurons). Finally, the harmonic model is the worst one, but perhaps it is a consequence of the simplicity of the periodogram: if the original data had a more complex spectrum perhaps this model would have fitted the data better than the other two.

5. Conclusions

In this report we have successfully analyzed the temperature data at a certain location, trying to describe and predict some of the patterns that govern the weather. With this goal in mind we have implemented three distinct models: a Holt-Winters model, a harmonic model and a neural network. Moreover, we have measured the performance of each of them through the use of a training-test split, yielding the following results:

Model	Holt-Winters	Harmonic model	Neural Network
MSE	2.81	3.46	3.34

As we can see, the model with the lowest mean squared error is the Holt-Winters model, which furthermore allowed to see an upwards trend in the level of the temperature. This latter fact is of particular interest when considering the context of the problem. However, the results of the Neural Network could be further improved through hyper-parameter tuning or modifying its architecture, and the harmonic model could be more useful if a more complex periodic pattern emerged. In any case, all three of them fitted our data in a satisfactory manner, which was the original purpose of this report.