Stat 443: Time Series and Forecasting

Assignment 3: Time Series Models

Caden Hewlett

March 21, 2024

Question 1: El Niño Forecasting

The file `NINO34.csv` contains the monthly El Niño 3.4 index from 1870 to 2023. The El Niño 3.4 index represents the average equatorial sea surface temperature (in degrees Celsius) from around the international dateline to the coast of South America.

Part a

Perform exploratory data analysis.

Part a.1

Import the data into R and create a time-series object for the El Niño 3.4 index.

```r
# import data
# setwd('C:/Users/caden/OneDrive/Desktop/STAT_443/STAT443')
df <- read.csv("NINO34.csv")

# raw data is in a bad format, need to pivot it without years
dflong <- df %>%
  pivot_longer(cols = -Year, names_to = "Month", values_to = "Value")
# re-translate into dates with month abbreviations
dflong <- dflong %>%
  mutate(Date = make_date(Year, match(Month, month.abb), 1))
# then, format into a time series!
nino_ts <- ts(dflong$Value,
              start = c(1870, 1), frequency = 12)

# time series length = data frame length = 154 years * 12 months per year
stopifnot(all.equal(length(nino_ts), nrow(df)*12, 154*12))
```

Break the time series object into a training and test set. You can use the function `window()` on a ts object to split the data. Let the training set be from January 1870 to December 2021, and let the test set start in January 2022 and end in November 2023.

```r
# split data into train and test
nino_train = window(nino_ts, start = c(1870, 1), end = c(2021, 12))
nino_test = window(nino_ts, start = c(2022, 1), end = c(2023, 11))
# verify split
stopifnot(all.equal(
          length(nino_test)+length(nino_train), length(na.remove(nino_ts))
      )
)
```

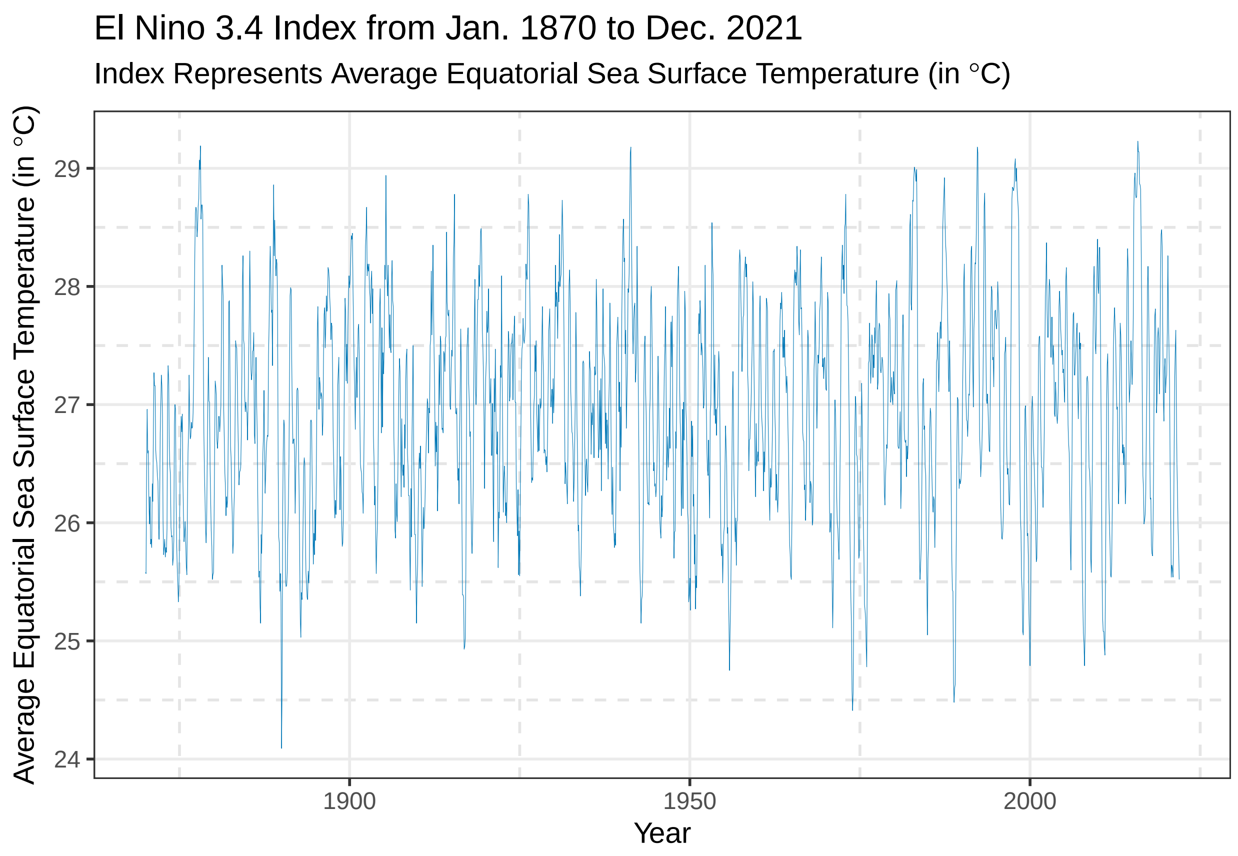
Part a.2

Plot the training data as well as its acf and pacf.

Training Data

```r
p1data = fortify.zoo(nino_train)
```

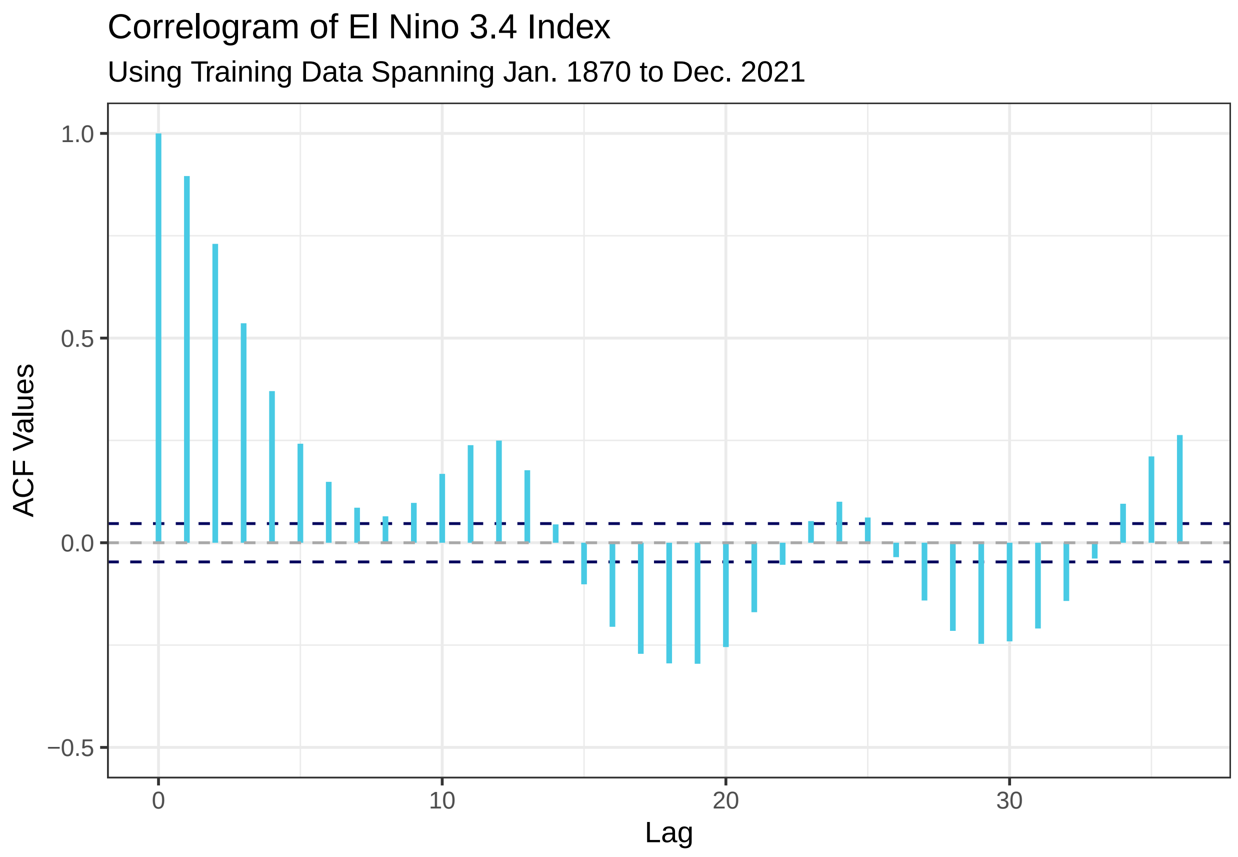
```r
print(p1)
```



Autocorrelation Function

```r
p2data = data.frame(
  h = 0:36,
  rh = acf(nino_train, plot = FALSE, lag.max = 36)$acf
)
```

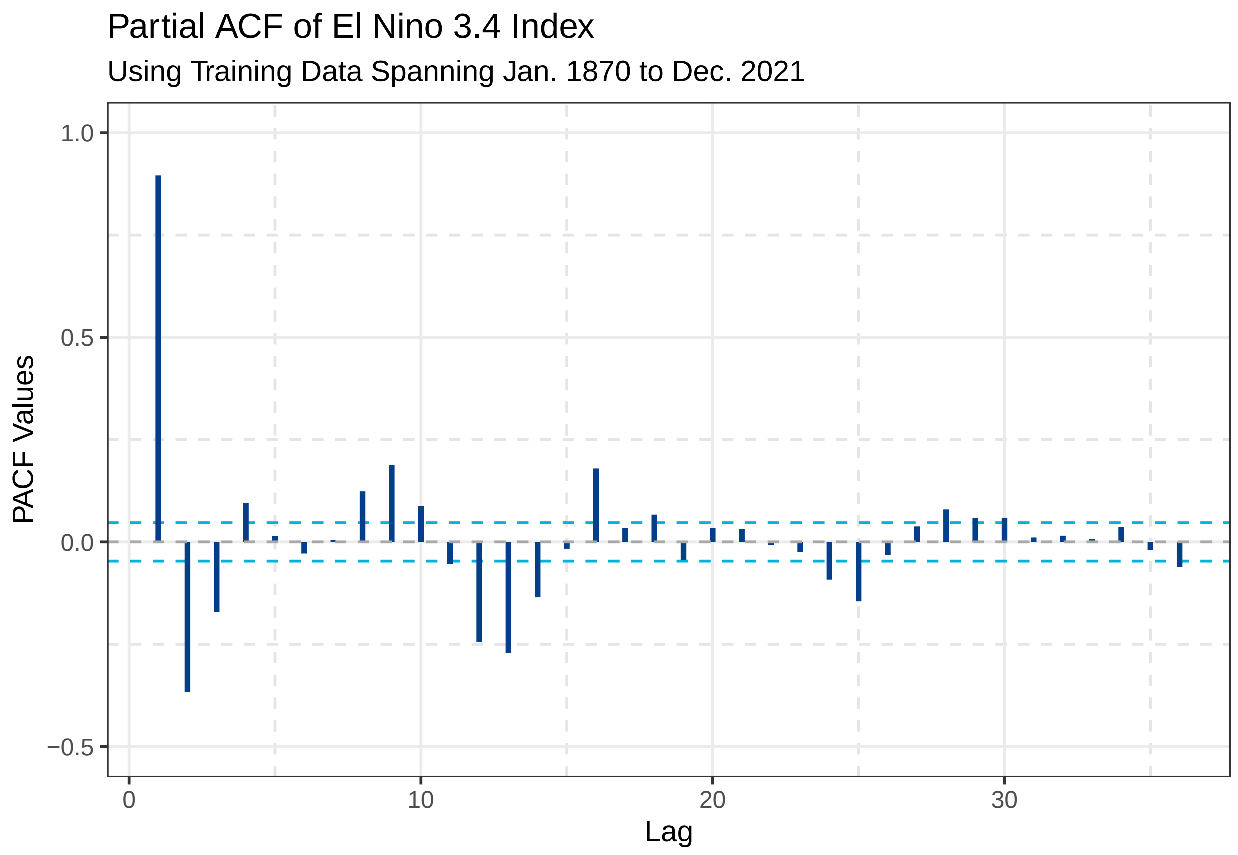
```r
print(p2)
```



Partial Autocorrelation Function

```r
p3data = data.frame(
  h = 1:36,
  rhh = pacf(nino_train, plot = FALSE, lag.max = 36)$acf
)
```

```r
print(p3)
```

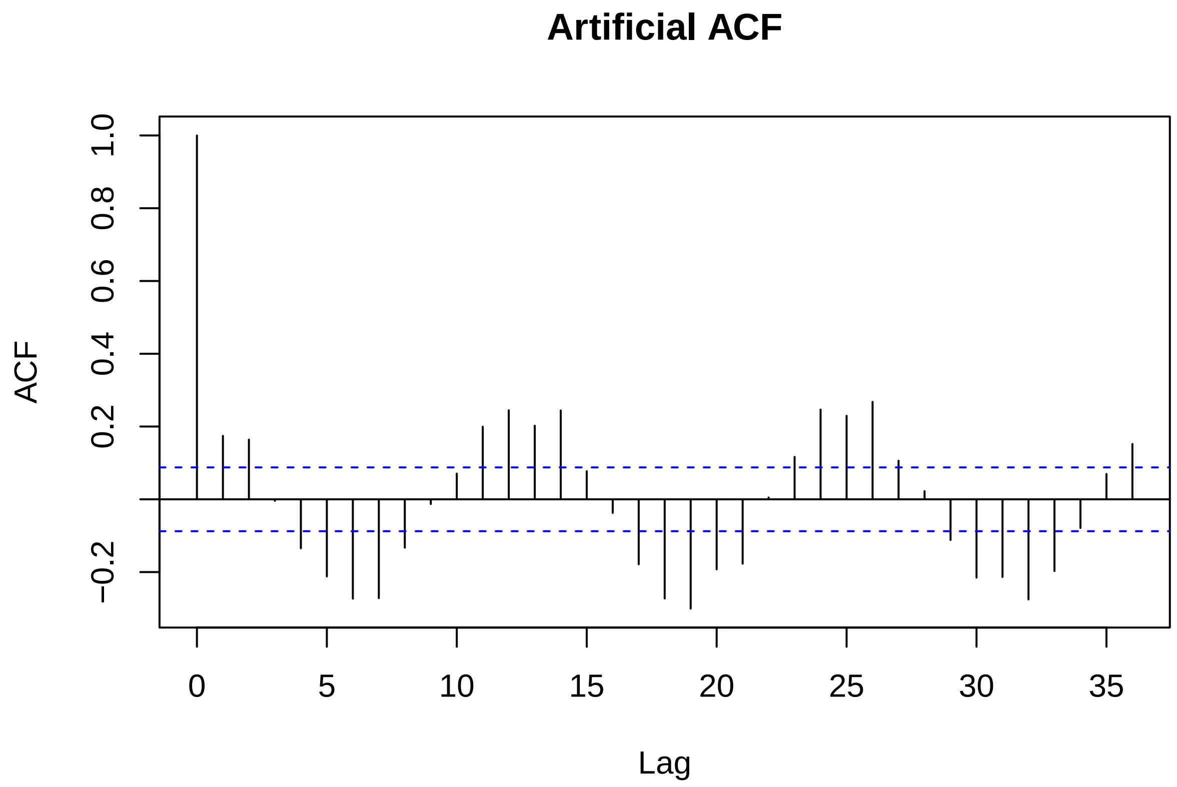


Comments

From the ACF, it is very likely that the original series exhibits a trend. This is due to the fact that there is a clear sinusoidal component of the ACF that is not dampening at a high rate (as we could potentially expect in models with AR components with $\alpha_i < 0$ for $i \in [1, p]$).

In fact, we can replicate the pattern in the sample ACF quite closely by simply using a sine function alongside random noise. Let $\{Z_t\}_{t=1}^{500} \overset{\text{iid}}{\sim} N(0,1)$ and let $X_t = Z_t + \sin(t/2)$, for $t \in \{1, 2, \dots, 500\} \subset \mathbb{N}$. A plot of this artificial additive seasonal model is below:

```r
set.seed(443); Z = rnorm(500); t = seq(1:500)
acf(Z + sin(t/2), main = "Artificial ACF", lag.max = 36)
```



As you can see, this artificial ACF closely matches our observed sample ACF from earlier. As such, it is very likely that there exists some seasonal term s_t in the time series of sea surface temperature data. We would intuitively anticipate a seasonal component to these data, as it is very likely that sea temperatures vary over the course of the year due to seasons as measurements are coming from the same area (the international dateline to the coast of South America.) Moreover, from a more informal observational perspective, a plot of the original series data doesn't seem to support a non-constant seasonal amplitude - as in there's no changing peak heights over time - which may indicate an additive seasonality rather than multiplicative.

In addition, neither the plot of the data nor the ACF seem to indicate a significant trend in the data. If there were a trend component m_t, we would see some consistent change over time in the overall direction of the series beyond simple seasonality. It doesn't seem like there is anything like that in this series; **however**, a purely visual analysis isn't comprehensive so this doesn't mean that a trend component doesn't exist.

Finally, to determine whether or not the series is stationary, we recall the definition of a weakly stationary stochastic process. Specifically, we will consider the first property of weak stationarity - that the mean is constant. We defined this formally in Assignment 1 previously, and was given similar to the following:

$$\text{Weak Stationarity Property One: } \exists \mu \in \mathbb{R} \text{ s.t. } \forall t \in \mathbb{Z},\ \mathbb{E}(X_t) = \mu$$

The presence of *either* a seasonal component or a trend causes a contradiction, as the expected value of the series becomes some function of t (and hence cannot be a constant.)

Thus, for our particular series, we have the following implication, letting s_t be the seasonal component, $c \in \mathbb{R}$ be some real constant and $f(t)$ be a non-constant real function (as in, it changes with t.) Then,

$$\exists s_t \implies \forall t\ \mathbb{E}(X_t) = f(t) + c \implies \nexists \mu \in \mathbb{R} \text{ s.t. } \forall t\ \mathbb{E}(X_t) = \mu \ \therefore\ X_t \text{ is not stationary. } \square$$

In short, due to the presence of a trend, the time series for the training data is not stationary. It should be noted that not much can be concluded with results from the PACF, as the series is non-stationary.

Part b

Forecast sea surface temperature for 2022 and 2023 using the Box-Jenkins method and the data from 1870-2021.

Part b.1.

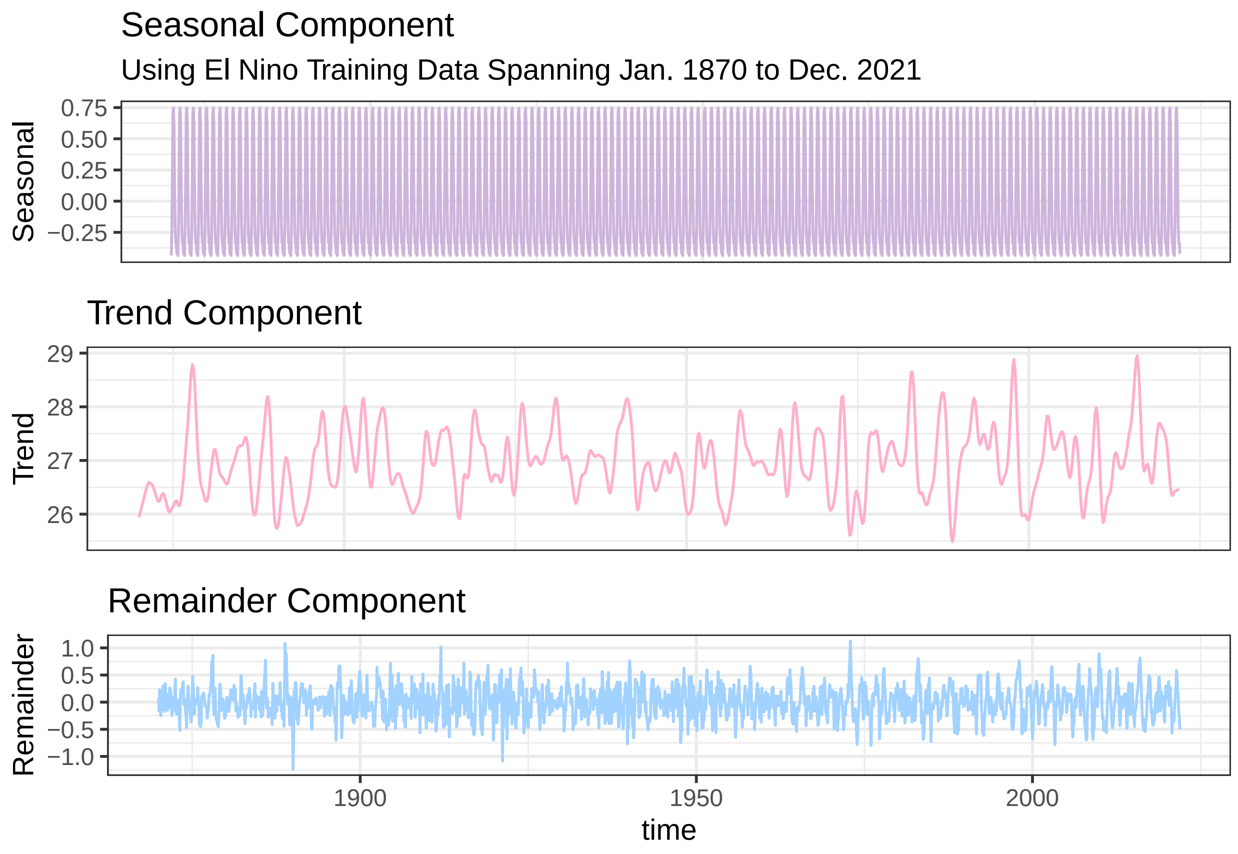
Remove any seasonal variation and trend from the training data, if there is any, using the `stl` function in R.

We define an object `nino_stl` using `loess` decomposition, and plot each of S, T and L (seasonal, trend and irregular) of the original series below:

```r
# perform loess decomposition
nino_stl = stl(nino_train, s.window = "periodic")
# cast to data frame
decomposition = as.data.frame(nino_stl$time.series)
decomposition$time = as.numeric(time(nino_train))
```

Note that the code to plot the decomposition with ggplot became quite long, so I made the code cell hidden. The complete plots are provided below:

```r
grid.arrange(pb1, pb2, pb3, ncol = 1)
```

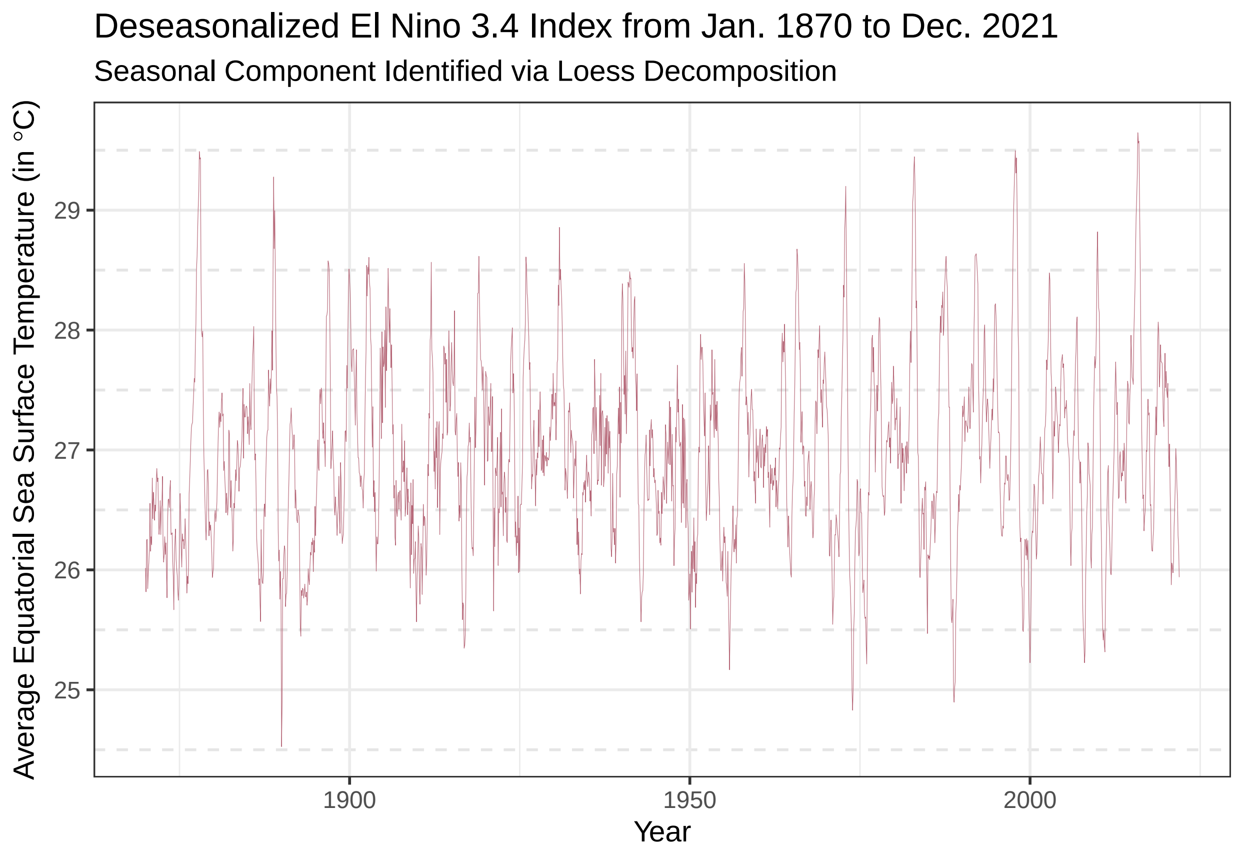


Now, we plot the filtered data set, as well as its acf and pacf. It doesn't seem like there is a significant trend component, as our original observations also showed, so we merely de-seasonalize the data. The removal of s_t and the plot of the filtered data is below:

Filtered Data

```r
# remove the season
nino_filtered =  nino_train - decomposition$seasonal
# prepare the filtered series for plotting
pb4data = fortify.zoo(nino_filtered)
```

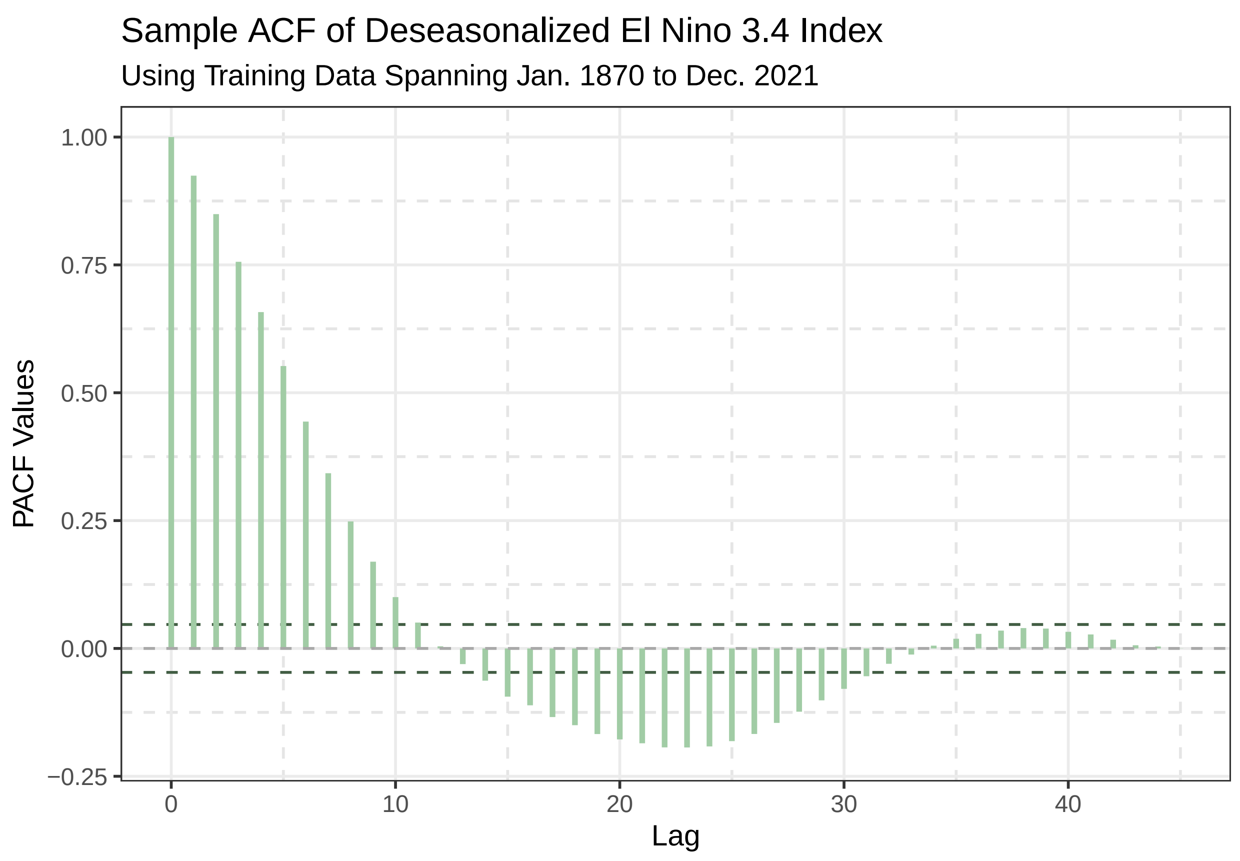
```r
print(pb4)
```



Autocorrelation

```r
# compute acf values for lags
p5bdata = data.frame(
  h = 0:45,
  rh = acf(nino_filtered, plot = FALSE, lag.max = 45)$acf
)
```

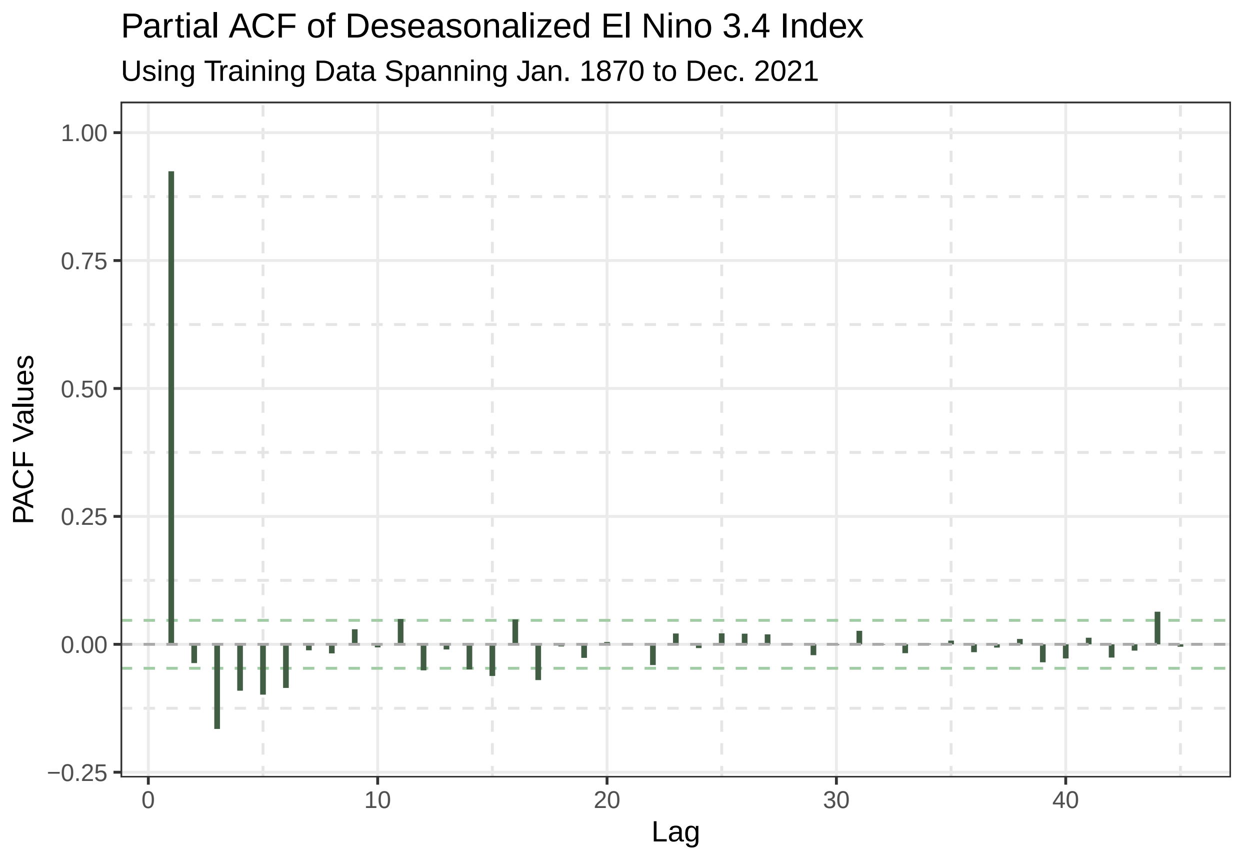
```r
print(pb5)
```



<u>Partial Autocorrelation</u>

```
p6bdata = data.frame(
  h = 1:45,
  rhh = pacf(nino_filtered, plot = FALSE, lag.max = 45)$acf
)
```

```
print(pb6)
```



Part b.2.

In the work to follow, we assume the de-seasonalized time series is stationary, such that the ACF and PACF can be used for model identification.

Now, using the standard graphical tools, let's select two models from the pure MA and pure AR family of models for the filtered data set.

Since the ACF follows a (slightly complicated) exponentially decaying sinusoidal pattern, it seems quite likely that the best candidate models will be from the AR family. If this were a pure MA process for some order q, we would see the ACF sharply cut off after the value of q is reached. Instead, the ACF slowly decays over time; this is indicative of an AR component to the series. Hence, we'll consider models form the AR family of models.

Now, given that we are attempting to find p under the presumption that the series follows some pure AR process, we can utilize the PACF to attempt to find a suitable p. Unfortunately, it isn't immediately evident which term p in the PACF satisfies the condition we commonly use that $\forall p > k,\ |\hat{\alpha}_{kk}| < 2/\sqrt{n}$.

We will consider two potential candidates, the first being AR(1) process. The justification for deciding upon $p = 1$ is based on the fact that $\hat{\alpha}_{11}$ is significantly larger than all other PACF values. While it is true that $\exists k > 1$ s. t. $|\hat{\alpha}_{kk}| \geq 2/\sqrt{n}$, there are no other values in the PACF greater than $\hat{\alpha}_{11}$. In other words, the PACF may not decline into noise; however, $\forall k \in [2, n] \subset \mathbb{Z}, \hat{\alpha}_{11} > \hat{\alpha}_{kk}$. So, while the PACF doesn't cut off into white noise after $p = 1$, it does cut off to significantly smaller values.

The second candidate is an AR(6) process. The reasoning is due to the fact that after $\hat{\alpha}_{66}$, the vast majority of observed $\hat{\alpha}_{kk}$ values for $k > 6$ seem to be below the white noise threshold. Critically, there isn't an easily distinguishable pattern to those $\hat{\alpha}_{kk}$ values greater than $\pm\sqrt{2}/n$ after PACF lag 6. This could mean that for $k > 6$ there is simply noise (as we want), however the data could have higher noise values than those commonly assumed by the W.N. threshold. It's important to recall that $\pm\sqrt{2}/n$ is more of a "rule of thumb" rather than a strict rule.

With all of this mind, we fit the models below:

```
# ar(1) vs. ar(6)
ar1 = arima(nino_filtered, order = c(1, 0, 0))
ar6 = arima(nino_filtered, order = c(6, 0, 0))
```

Then, we extract the coefficients and show them in the table below, rounding each to three decimal places:

```
ar_fits = t(round(data.frame(
  AR1 = c(ar1$coef[-1], head(ar1$coef, -1), rep(NA, times = 5), ar1$sigma2),
  AR6 = c(tail(ar6$coef, 1), head(ar6$coef, -1), ar6$sigma2)), 3))
# report with nice latex column names
kable(ar_fits, "latex", escape = FALSE,  col.names = var_col_names,
      caption = "Fitted AR Coefficients")  %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 1: Fitted AR Coefficients

	$\hat{\mu}$	$\hat{\alpha}_1$	$\hat{\alpha}_2$	$\hat{\alpha}_3$	$\hat{\alpha}_4$	$\hat{\alpha}_5$	$\hat{\alpha}_6$	$\hat{\sigma}^2$
AR1	26.955	0.926	NA	NA	NA	NA	NA	0.085
AR6	26.954	0.920	0.129	-0.074	0.009	-0.016	-0.087	0.080

Part b.3.

Compare the AIC values of your two models. Which would you pick based on AIC?

First, we extract the AIC values of each fit.

```
kable(
  t(data.frame(
    AR1 = round(ar1$aic, 3),
    AR6 = round(ar6$aic, 3))),
  col.names = c("Candidate Model", "AIC"),
  caption = "Akaike Information Criterion"
)  %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 2: Akaike Information Criterion

Candidate Model	AIC
AR1	687.816
AR6	596.045

Let AIC(m) be the AIC for models $m \in M$, where $M = \{\text{AR}(1), \text{AR}(6)\}$. Generally, models with smaller AIC values are preferred.

So, we can select the best model m_{best} with the following logic:

$$m_{\text{best}} = \underset{m \in M}{\text{argmin}} \big\{ \text{AIC}(m) \big\}, \text{ where } M = \big\{ \text{AR}(1), \text{AR}(6) \big\}$$

Then, using the reported values above, we can select m_{best} as follows:

$$\begin{aligned} m_{\text{best}} &= \underset{m \in M}{\text{argmin}} \big\{ \text{AIC}(m) \big\} \\ m_{\text{best}} &= \underset{m \in M}{\text{argmin}} \Big\{ \text{AIC}\big(\text{AR}(1)\big), \text{AIC}\big(\text{AR}(6)\big) \Big\} \\ m_{\text{best}} &= \underset{m \in M}{\text{argmin}} \Big\{ 687.816, 596.045 \Big\} \\ m_{\text{best}} &= \text{AR}(6) \end{aligned}$$

So, by AIC selection, we prefer the AR(6) model.

Part b.4.

We recall that the third plot of `tsdiag` are the p-values of the Ljung-Box Version of the Portmanteau Lack-of-Fit test, where the following hypotheses are being compared:

$$H_0: \text{ The Model is Well-Specified. } \text{ v.s. } H_A: \text{ The Model is Not Well-Specified.}$$

Where the p-value is found via the χ^2 distribution and the test statistic, given by the following for a pure AR model (i.e. $q = 0$)

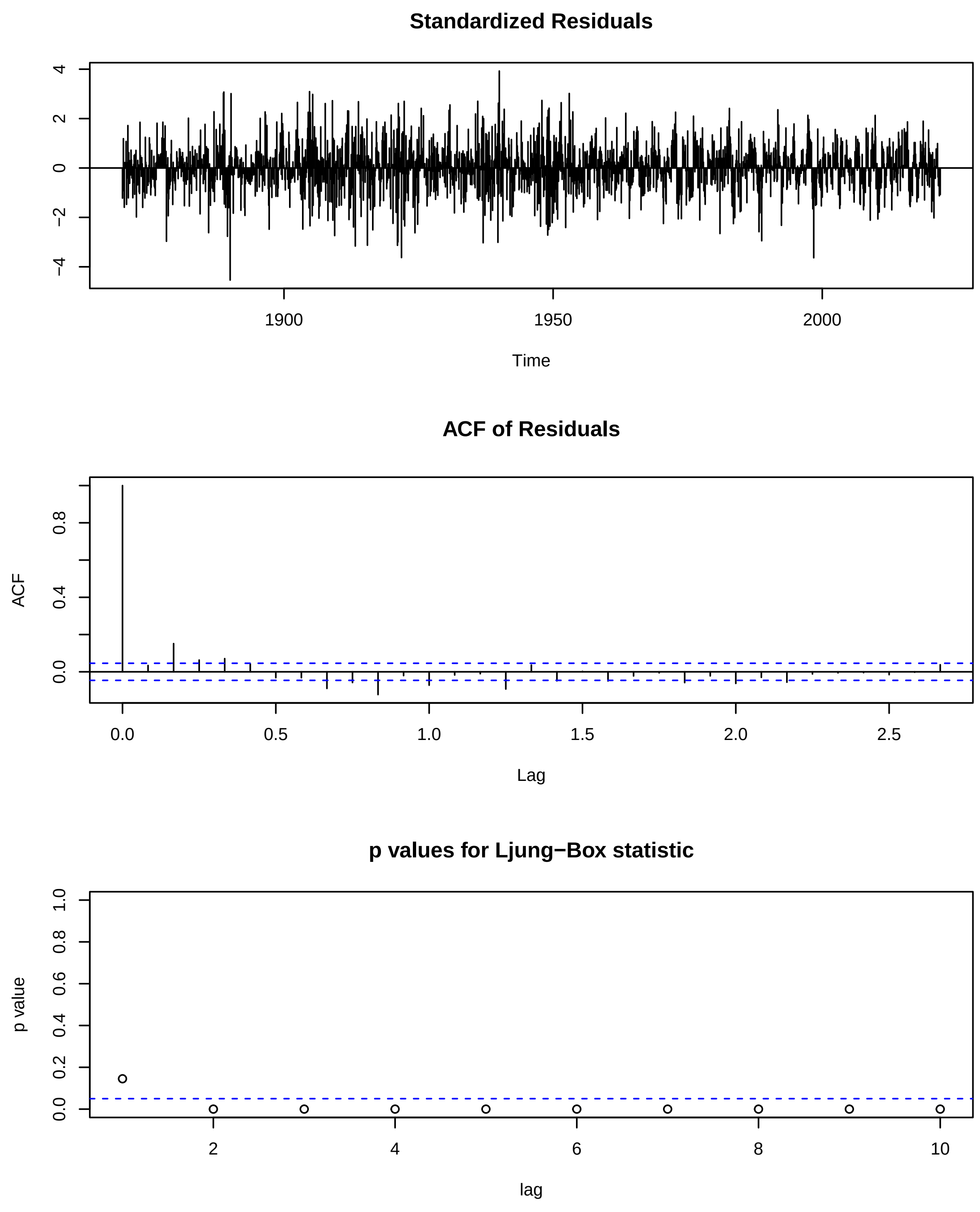
$$Q_2 = n(n+2) \sum_{i=1}^{h} \frac{\hat{\rho}_z(h)^2}{n-h} \sim \chi^2_{h-p}$$

To get a better understanding of the plots to follow, we compute the Q_2 values for $h \in [1, 10] \subset \mathbb{Z}$ below.

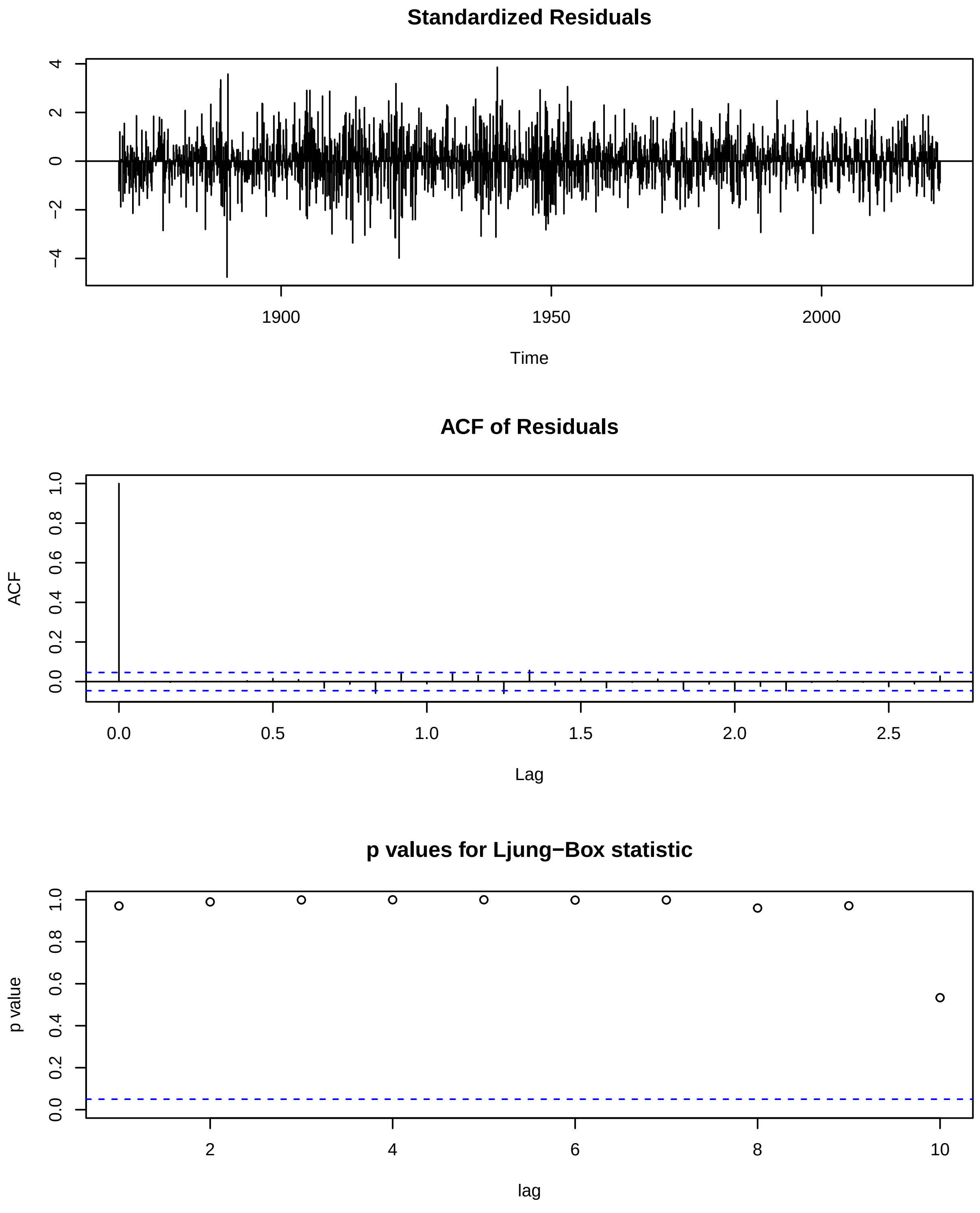
```r
n = length(nino_filtered)
lb_stats = matrix(0, ncol = 2, nrow = 10)
M = list(ar1, ar6)
for(i in 1:length(M)){
  # get i-th model
  m = M[[i]]
  # get rho(h) values of residuals
  rh_m  = acf(m$residuals, lag.max = 10, plot = F)$acf
  # compute Ljung-Box stat
  lb_ar =  sapply(2:11,
         function(h){
           Q = n*(n+2)*sum( rh_m[2:h]^2 / (n - h))
           p = pchisq(Q, df = h - 1, lower.tail = F) # p = 1 for AR(1)
           p
         })
  lb_stats[, i] = lb_ar
}
```

Now, we the `tsdiag` function in R to plot diagnostics for our two models.

```r
tsdiag(ar1)
```


Standardized Residuals
4
2
0
−2
−4
1900
1950
2000
Time
ACF of Residuals
ACF
0.0
0.4
0.8
0.0
0.5
1.0
1.5
2.0
2.5
Lag
p values for Ljung−Box statistic
p value
0.0
0.2
0.4
0.6
0.8
1.0
2
4
6
8
10
lag

```r
tsdiag(ar6)
```



What do you observe?

We see that the AR(1) model has Ljung-Box p-values below the $\alpha = 0.05$ significance threshold $\forall h \in [2, 10]$, which is confirmed by out `lb_stats` object, as you can see below:

```r
which(lb_stats[, 1] < 0.05)
```

```
## [1]  2  3  4  5  6  7  8  9 10
```

This tell us that we reject H_0 in favour of the alternative that the model is ill-fitting for the AR(1) case in the majority of scenarios. This implies that that the AR(1) model may not be well-specified for these data. This theory is supported by the ACF of the residuals, which appears to have a vague sinusoidal pattern to it, as well as having values above $\pm 2/\sqrt{n}$ for quite a few of the lags. All in all, the `tsdiag` results for the AR(1) model shed doubt on our confidence of this candidate being a good fit for these data.

To contrast, the Ljung-Box statistic values for the AR(6) model are consistently well above $\alpha = 0.05$, as shown in the table below (for the first five lags to conserve space):

```r
kable(t(data.frame(
          AR1 = format(lb_stats[1:5, 1], scientific = TRUE, digits = 5),
          AR6 = format(lb_stats[1:5, 2], scientific = TRUE, digits = 5))),
     caption = "Ljung-Box Portmanteau Test $p$-Values for Varying Lags",
     col.names = paste("Lag", 1:5))  %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 3: Ljung-Box Portmanteau Test $p$-Values for Varying Lags

	Lag 1	Lag 2	Lag 3	Lag 4	Lag 5
AR1	1.4506e-01	2.6950e-10	4.0126e-11	2.1668e-12	1.8243e-12
AR6	9.7066e-01	9.8999e-01	9.9923e-01	9.9995e-01	9.9998e-01

This shows that the probability of observing the computed Q_2 values given H_0 that the model is well-specified is much higher for the AR(6) model than the AR(1) alternative. Further, the ACF of the residuals of the AR(6) model seem to quickly decay towards white noise, and shows far less extremity and patterning than the AR(1) ACF. So, if we were deciding the best candidate m_{best} purely from these results, we would again prefer the AR(6) model.

Part b.5.

Predict the sea surface temperature for Jan 2022 through Nov 2023 using both candidate models. Use this to calculate the mean squared prediction error (MSPE) for both models.

We again let $m \in M$ be the candidate models, for $M = \{\text{AR}(1), \text{AR}(6)\}$. We let $\{y_t\}_{t \in \mathbb{Z}}$ be the observed values from the test set. We let $\hat{y}_t^{(m)}(\ell)$ be the ℓ-step ahead Box-Jenkins forecast (for $\ell \in \mathbb{Z}$) of model m (with the seasonal component added back in.) Let N_ℓ be the total number of steps ahead. In our case, this is 23 for both.

The set of MSPE value for the candidate models is given by:

$$\text{MSPE} : \left\{ m \in M : \frac{1}{N_\ell} \sum_{\ell=1}^{N_\ell} \left(y_{t+\ell} - \hat{y}_t^{(m)}(\ell) \right)^2 \right\}$$

We compute these values in `R` below:

```r
N_L = 23
# make predictions for held-out test data
ar1_forecast = predict(ar1, n.ahead = 23)
ar6_forecast = predict(ar6, n.ahead = 23)
# re-seasonalize the forecasts
s_test = (decomposition$seasonal)[1:23]
ar1_fit = ar1_forecast$pred + s_test
ar6_fit = ar6_forecast$pred + s_test
# define the models
mods = c("AR(1)", "AR(6)")
# compute MSPE for each
mspes = sapply(list(ar1_fit, ar6_fit),
               function(m_fit){
                 mean( (nino_test - m_fit )^2 )
               })
# report results
kable(t(round(mspes, 5)), col.names = mods,
      caption = "Observed MSPE of Candidates") %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 4: Observed MSPE of Candidates

AR(1)	AR(6)
0.99617	0.79398

Then, to determine which method performs better, we select the argmin of the MSPEs set.

$$m_{\text{best}} = \underset{m \in M}{\text{argmin}} \Big\{ \text{MSPE}(m) \Big\} = \underset{m \in M}{\text{argmin}} \Big\{ 0.996, 0.794 \Big\} = \text{AR}(6)$$

In our case, this is the AR(6) model.

```r
mods[which.min(mspes)]
```

```
## [1] "AR(6)"
```

Hence, by MSPE, the best model of the candidate models m_{best} remains the AR(6) model.

Part b.6.

On a single plot, display the test set of sea surface temperature, the predictions for both models, and their approximate 95% prediction intervals.

NOTE: We will include the plotting code for this question, as it specifically asks for it.

First, we compute the 95% prediction intervals using the following method: As before, we let $\hat{y}_t^{(m)}(\ell)$ be the $\ell \in \mathbb{Z}$-step ahead Box-Jenkins forecast for model $m \in M$. We let $\text{Z} \sim N(0,1)$ and "PrI" indicate the prediction interval at step ℓ, given significance level α and model m. We then define the prediction intervals as follows:

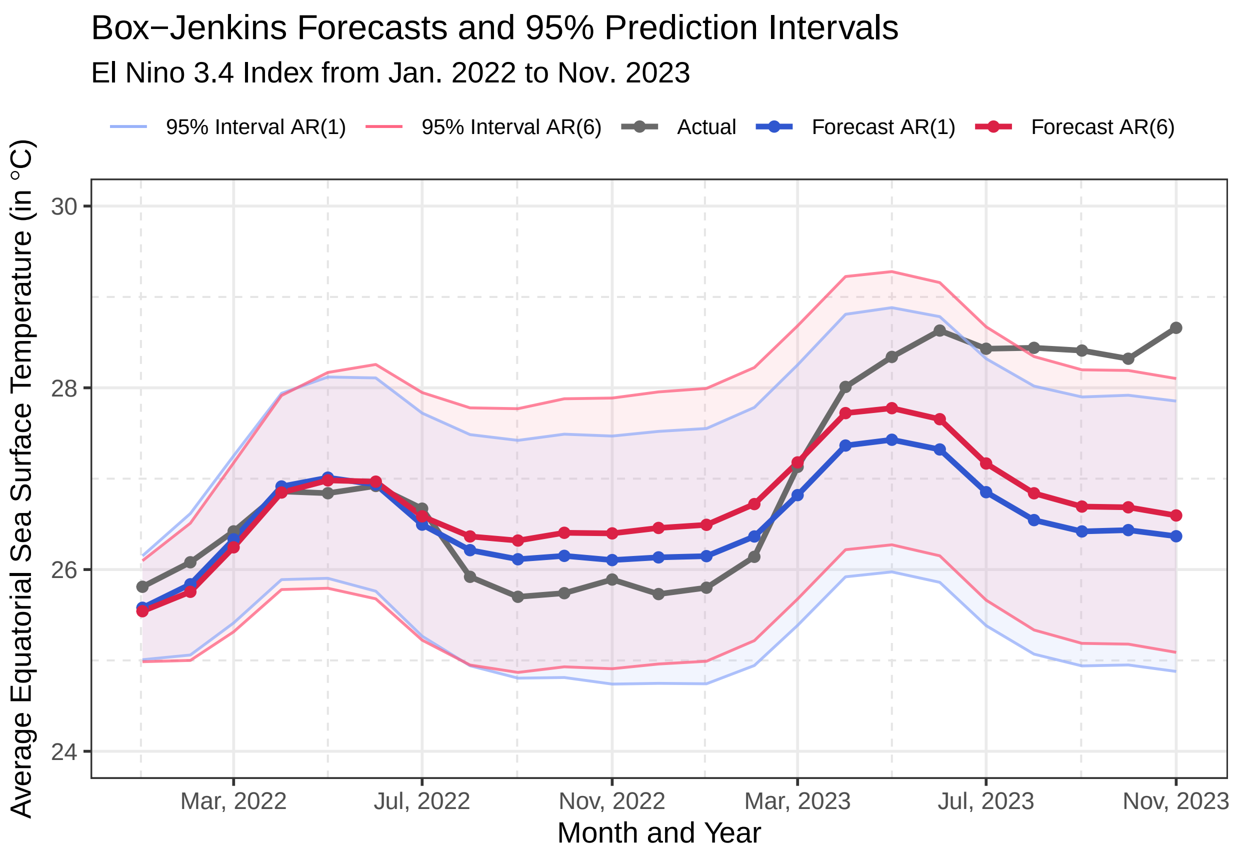
$$\text{PrI}(\ell \mid \{\alpha, m\}) = \hat{y}_t^{(m)}(\ell) \pm \text{Z}_{\alpha/2}\, \text{se}\Big(\hat{y}_t^{(m)}(\ell)\Big)$$

For this question, we employ $\alpha = 0.05$ and $m \in M$ to compute the intervals. The `R` code to do this is below. We define the lower and upper intervals separately so that all of the values can be stored in a data frame for plotting.

```r
alpha = 0.05
Z = qnorm(1-alpha/2)
# prediction intervals for AR(1)
ar1_Low = ar1_fit - Z*ar1_forecast$se
ar1_High = ar1_fit + Z*ar1_forecast$se
# prediction intervals for AR(6)
ar6_Low = ar6_fit- Z*ar6_forecast$se
ar6_High = ar6_fit + Z*ar6_forecast$se
# construct the data frame
bigplot_DF = data.frame(
  Time = as.Date(time(nino_test)),
  Observed = as.numeric(nino_test),
  Forecast_AR1 = as.numeric(ar1_fit),
  Lower_AR1 = as.numeric(ar1_Low),
  Upper_AR1 = as.numeric(ar1_High),
  Forecast_AR6 = (ar6_fit),
  Lower_AR6 = as.numeric(ar6_Low),
  Upper_AR6 = as.numeric(ar6_High)
)
```

Now, we make the combined plot (with labels and a legend) below:

```r
pfancy = ggplot(data = bigplot_DF, aes(x = Time)) +
  # light blue and red fill area between prediction intervals
  geom_ribbon(aes(ymin = Lower_AR1, ymax = Upper_AR1), fill = "#a7bdfa", alpha = 0.15) +
  geom_ribbon(aes(ymin = Lower_AR6, ymax = Upper_AR6), fill = "#fa98ac", alpha = 0.15) +
  # lines for observed
  geom_line(aes(y = Observed, color = "Actual"),  lwd = 1) +
  # lines and prediction interval for AR(1)
  geom_line(aes(y = Forecast_AR1, color = "Forecast AR(1)"),lwd = 1) +
  geom_line(aes(y = Lower_AR1, color = "95% Interval AR(1)"), alpha = 0.75) +
  geom_line(aes(y = Upper_AR1, color = "95% Interval AR(1)"), alpha = 0.75) +
  # lines and prediction interval for AR(6)
  geom_line(aes(y = Forecast_AR6, color = "Forecast AR(6)"),lwd = 1) +
  geom_line(aes(y = Lower_AR6, color = "95% Interval AR(6)"), alpha = 0.75) +
  geom_line(aes(y = Upper_AR6, color = "95% Interval AR(6)"), alpha = 0.75) +
  # add points too
  geom_point(aes(y = Observed, color = "Actual")) +
  # points for AR(1)
  geom_point(aes(y = Forecast_AR1, color = "Forecast AR(1)")) +
  # points for AR(6)
  geom_point(aes(y = Forecast_AR6, color = "Forecast AR(6)")) +
  # legend and colour assignment
  scale_color_manual(name = "",
    values = c(
    "Actual" = "#696969",
    "Forecast AR(1)" = "#3057cf",
    "95% Interval AR(1)" = "#93adfa",
    "Forecast AR(6)" = "#db2146",
    "95% Interval AR(6)" = "#ff5c7c"
  )) +
  # customize x-axis for nice dates
  scale_x_date(date_breaks = "4 month",  date_labels = "%b, %Y") +
  # add titles with units
  labs(
    title = "Box-Jenkins Forecasts and 95% Prediction Intervals",
    subtitle = "El Nino 3.4 Index from Jan. 2022 to Nov. 2023",
    x = "Month and Year",
    y = expression(
      paste("Average Equatorial Sea Surface Temperature (in ", degree, "C)")
    )
  ) + ylim(24, 30) +
  # customize theme: gridlines, margins, etc.
  theme_bw() +
  theme(
    panel.grid.minor =  element_line(
      color = "grey90",
      linewidth  = 0.35,
      linetype = "dashed"
    ),
    legend.position = "top", legend.justification = "left",
    legend.margin = margin(0,0,0,0),  legend.text = element_text(size = 8))
print(pfancy)
```



Comments: The AR(6) model seems to do a slightly better job than the AR(1) alternative in terms of closeness to the true Average Equatorial Sea Surface Temperature for the test set from Jan. 2022 to

Nov. 2023, however, for larger lead times neither model seems to perform extremely well. It is interesting to note that the 95% prediction interval for the AR(1) is lower than that of the AR(6) model; however, there is quite a bit of overlap between the intervals and a distinct similarity to the overall pattern of forecast values. Summarizing all of our analyses thusfar, it seems that the AR(6) model is best from the pure AR/MA candidates at our disposal. This being said, it still doesn't seem to precisely capture the overall pattern of the data, suggesting that some more complicated method (such as SARIMA) may better in future investigations.

Part c

Forecast sea surface temperature for 2022 and 2023 using the Holt-Winters method and the data from 1870-2021.

Part c.i.1.

First, we use the `HoltWinters` function in R to fit an appropriate model to the training data.

As in our original series (and with the Box-Jenkins method) we did not observe a significant trend, hence we will construct a Holt-Winters model with $\beta = 0$.

We will use an additive seasonal effect, which inherits the `Frequency` of the time series as the period. In this case, $p = 12$.

We then have an additive seasonal effect I_t for t given by the following:

$$I_t = \gamma(x_t - L_t) + (1 - \gamma)I_{t-p}$$

Where L_t is the level component that defines Holt-Withers smoothing/forecasting methods. Since $\beta = 0$, it is constructed without a T component, and is given by:

$$L_t = \alpha(x_t - I_{t-p}) + (1 - \alpha)L_{t-1}$$

For $\alpha, \gamma \in [0, 1]$. Further, for $\ell \in \mathbb{Z}$, the ℓ-step ahead forecast at time t is given by the following.

$$\hat{x}_t(\ell) = L_t + I_{t-p+\ell}$$

We report the coefficients from `HoltWinters` in the table below.

```r
hw_no_trend = HoltWinters(nino_train, beta = FALSE, seasonal = "additive") # no trend
# data frame for nice formatting
hw_results_coefs = (data.frame(
  Alpha = round(hw_no_trend$alpha, 3),
  Beta = paste(0, ".000", sep = ""),
  Gamma = paste(hw_no_trend$gamma, ".000", sep = "")
))
# show results in a table
rownames(hw_results_coefs) = NULL
kable(hw_results_coefs, "latex", escape = FALSE,
      col.names = c("$\\hat{\\alpha}$", "$\\hat{\\beta}$", "$\\hat{\\gamma}$"),
      caption = "Fitted Holt-Winters Coefficients")  %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Our fitted $\hat{\gamma} = 1$, implying that the model places maximum smoothing weight on the current state of the season (i.e. the $(1 - \gamma)I_{t-p}$ component has zero weight according to the fitting process.) Further, our fitted $\hat{\alpha} \approx 0.96$ implies the model is placing a lot of significance on the current level (and period) and much less on levels in the past when fitting.

Table 5: Fitted Holt-Winters Coefficients

$\hat{\alpha}$	$\hat{\beta}$	$\hat{\gamma}$
0.96	0.000	1.000

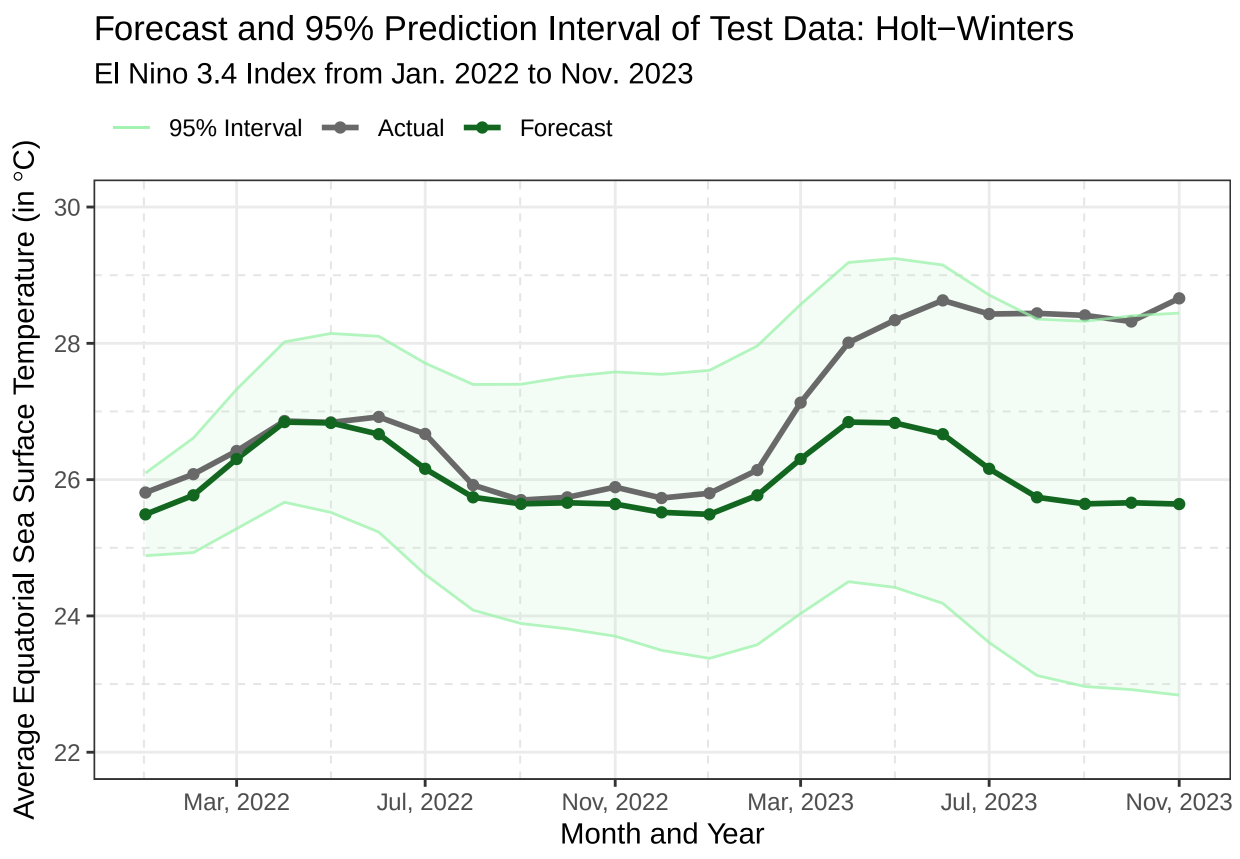
Part c.i.2.

Use this model to predict sea surface temperature from Jan 2022 through Nov 2023.

```r
# make predictions and format as data frame
hw_predictions = data.frame(
  predict(hw_no_trend, n.ahead = 23, prediction.interval = TRUE, level = 0.95))

# prepare the hw forecast data
p4data = data.frame(
  Time = as.Date(time(nino_test)),
  Observed = as.numeric(nino_test),
  Forecast = as.numeric(hw_predictions$fit),
  Lower = as.numeric(hw_predictions$lwr),
  Upper = as.numeric(hw_predictions$upr)
)
```

```r
print(p4)
```



Part c.i.3.

Calculate the mean squared prediction error.

The mean squared prediction error for our $N_\ell = 23$ total ℓ step ahead forecast is given by:

$$\text{MSE}_{\text{pred}} = \frac{1}{N} \sum_{\text{Holdout}} \Big(\text{Truth} - \text{Forecast}\Big)^2 = \frac{1}{N_\ell} \sum_{\ell=1}^{N_\ell} \Big(x_{t+\ell} - \hat{x}_t(\ell)\Big)^2$$

```r
N = length(nino_test)
# verify equality in set cardinalities
stopifnot(all.equal(N, length(hw_predictions$fit)))
# calculate ms prediction error
ms_pre_hw = sum ( (nino_test - hw_predictions$fit )^2 ) / N
ms_pre_hw
```

```
## [1] 1.972297
```

The observed Mean Square prediction error for the Holt-Withers forecast is approximately 1.972.

Part c.i.4.

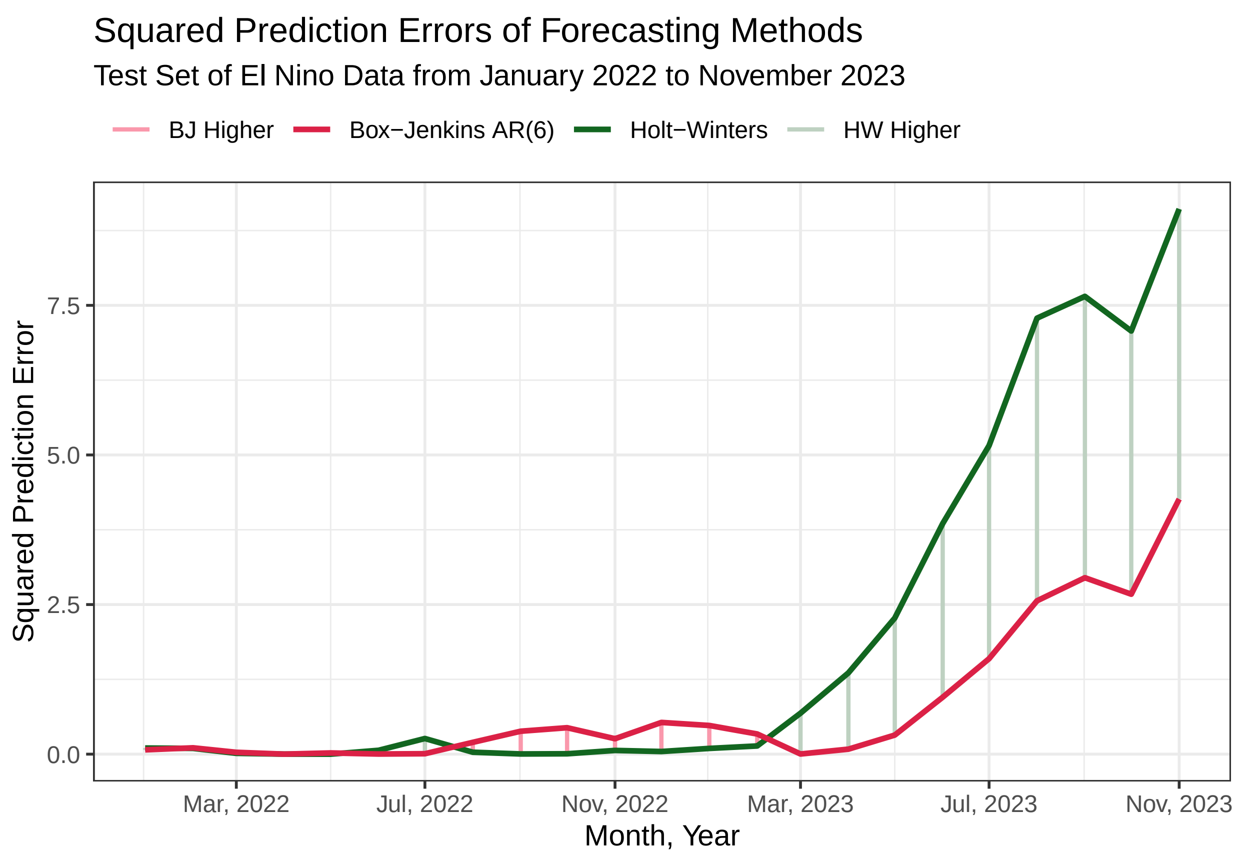
How does it compare to the Box-Jenkins models above?

The Holt-Winters forecast appears quite close to the true values small values of ℓ small, but becomes much worse at prediction as the time predicted into the future grows. Further, the prediction interval of the Holt-Withers method becomes much wider over time, when compared to the two Box-Jenkins forecasts. While the Box-Jenkins methods also exhibit this increase in uncertainty, it is notably less significant (noting that the lower end of the interval for H.W. is near 22, which is completely outside of the range of the B.J. approach.)

When comparing the Holt-Winters squared prediction errors to the better performing of the two Box-Jenkins models, we see that for smaller vallues of ℓ there are cases in which the H.W. Forecast has a smaller MSPE than AR(6) Box-Jenkins. However, as the plot below demonstrates, the squared error skyrockets in the Holt-Winters method as ℓ grows.

```r
squared_errors = data.frame(
  Time =  as.Date(time(nino_test)),
  err_HW = as.numeric((nino_test - hw_predictions$fit )^2),
  err_BJ = as.numeric((nino_test - ar6_fit)^2)
  )
```

```r
print(p13)
```



Comments: As the plot above shows, while the Box-Jenkins model also has a sharp squared error increase, it is much smaller than that of the Holt-Winters model. Quantitatively, this is why the MSPE is much larger for Holt-Winters (MSPE ≈ 1.972) than either of the AR(1) and AR(6) Box-Jenkins methods (MSPE ≈ 0.996 and 0.794 respectively.) So, for these particular data, some variation on the Box-Jenkins approach may be preferred.

Part c.ii.

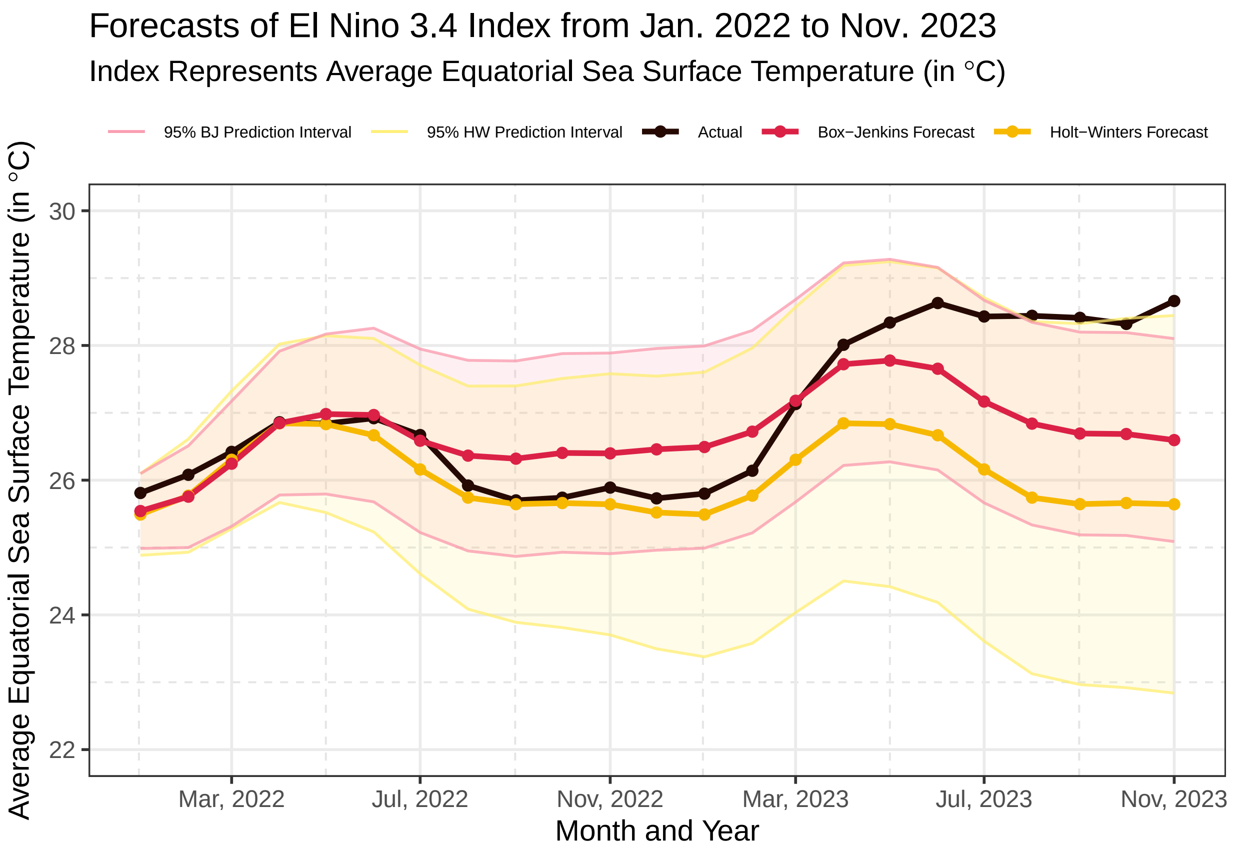
On a single plot, display the test set of sea surface temperature, the predictions from your preferred Box-Jenkins model, the predictions from your Holt-Winters model, and relevant 95% prediction intervals. Which method performs better?

We will select the AR(6) model and compare it to the Holt-Winters model.

```r
# format all plotting data
p1c2data = data.frame(
  Time = as.Date(time(nino_test)),
  Observed = as.numeric(nino_test),
  Forecast_HW = as.numeric(hw_predictions$fit),
  Lower_HW = as.numeric(hw_predictions$lwr),
  Upper_HW = as.numeric(hw_predictions$upr),
  Forecast_AR6 = (ar6_fit),
  Lower_AR6 = as.numeric(ar6_Low),
  Upper_AR6 = as.numeric(ar6_High)
)
```

We omit the code used to make this plot, as it is quite long. It is a nearly-identical process to the other plots in this assignment.

```r
print(p1c2)
```



Comments: In addition to our previous computations and analyses, the above plot emphasizes the better performance of the Box-Jenkins model when compared to the Holt-Winters model at predicting the test set of sea surface temperatures. The Holt-Winters fit severely underestimates the truth, and the prediction interval is far wider than the Box-Jenkins interval and descends far lower than anywhere near the true average equatorial temperature. In addition, the Box-Jenkins prediction interval is far narrower and precise (only occasionally creeping above the Holt-Winters interval.) With this plot alongside the comparison of MSPE and visual Squared-Error plots, I would recommend the Box-Jenkins model over the Holt-Winters model for forecasting the El Nino 3.4 index.

Question 2 : Hours Worked Forecasting

In this question we will predict the time series of monthly average values of the usual hours worked across all industries in Canada for the period from January 1987 until December 2023, which was explored in Assignment 1, using the file `usual_hours_worked_ca.csv`.

We'll use the Box-Jenkins method and Holt-Winters method.

Part 1: Data Preparation

Read in the data and create a time-series object for the mean monthly working hours:

```r
hDF = read.csv("usual_hours_worked_ca.csv")
hours_series = ts(hDF$Hours, start = c(1987, 1), frequency = 12)
```

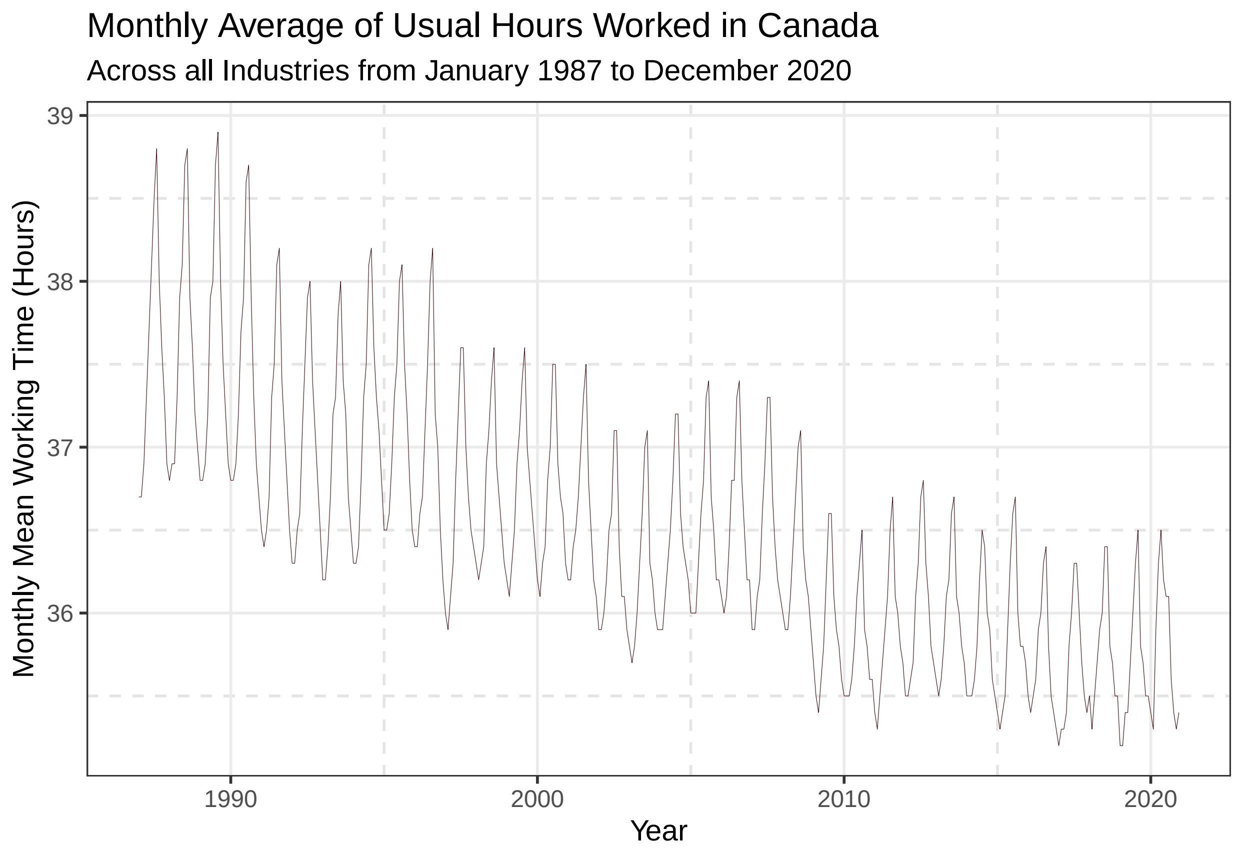
Separate the data into train and test. The training dataset should include all observations up to and including December 2020; The test dataset should include all observations from January 2021 to December 2023

```r
hours_train = window(hours_series, end = c(2020, 12))
hours_test = window(hours_series, start = c(2021, 1), end = c(2023, 12))
# verify split is valid
stopifnot(all.equal(length(hours_test)+length(hours_train),
                    length(na.remove(hours_series))))
```

Prepare and plot the training data.

```r
p5data = fortify.zoo(hours_train)
```

```r
print(p5)
```



Part 2: Box-Jenkins Method

In this part, we select and fit a SARIMA$(p, d, q) \times (P, D, Q)_s$ model and make forecasts using the fitted model.

Part A: Differencing

Difference the training set time series at lag 1. Let $\{X_t\}_{t\in\mathbb{Z}}$ be our original series. We define below $Y_t = \nabla^1 X_t = X_t - X_{t-1}$

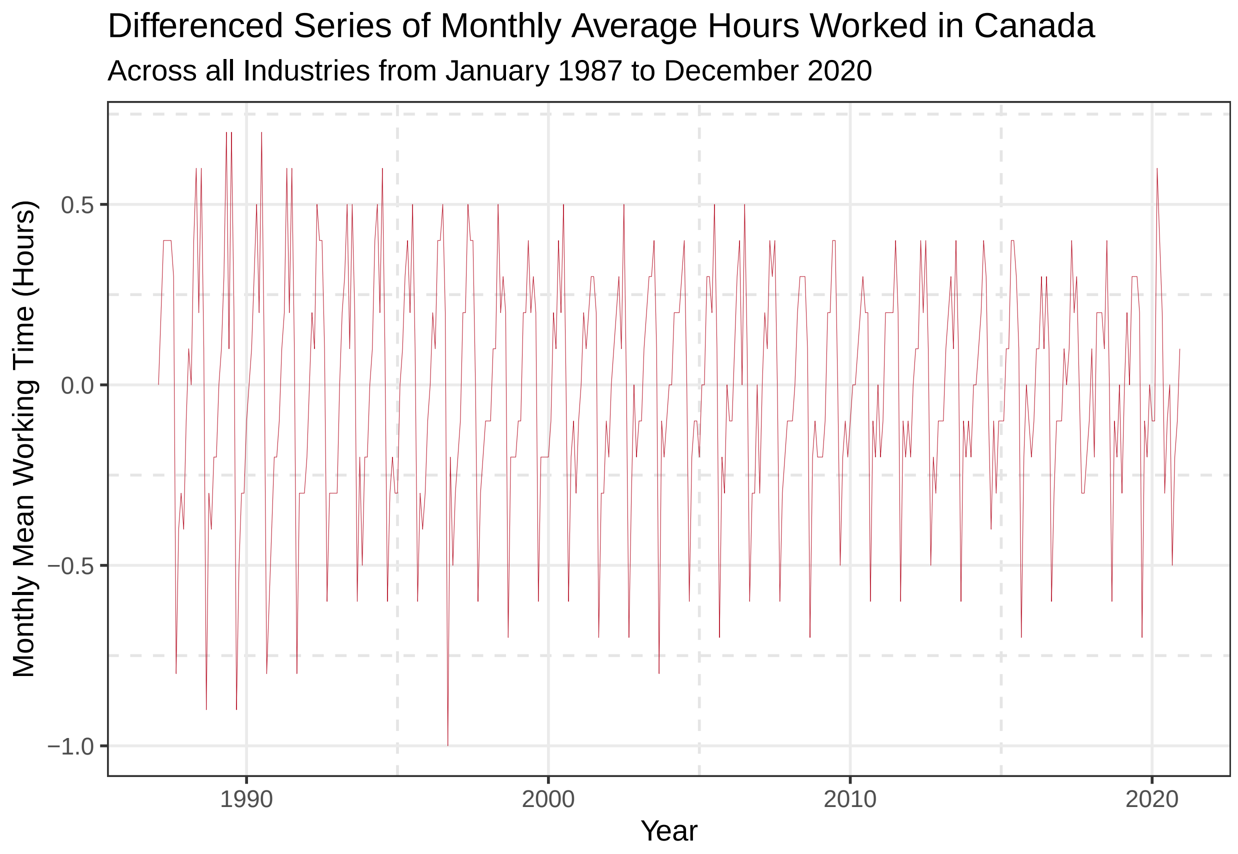
```r
hours_train_diff = diff(hours_train, lag = 1, differences = 1)
```

Plot the new time series and its correlogram.

New Series

```r
p6data = fortify.zoo(hours_train_diff)
```

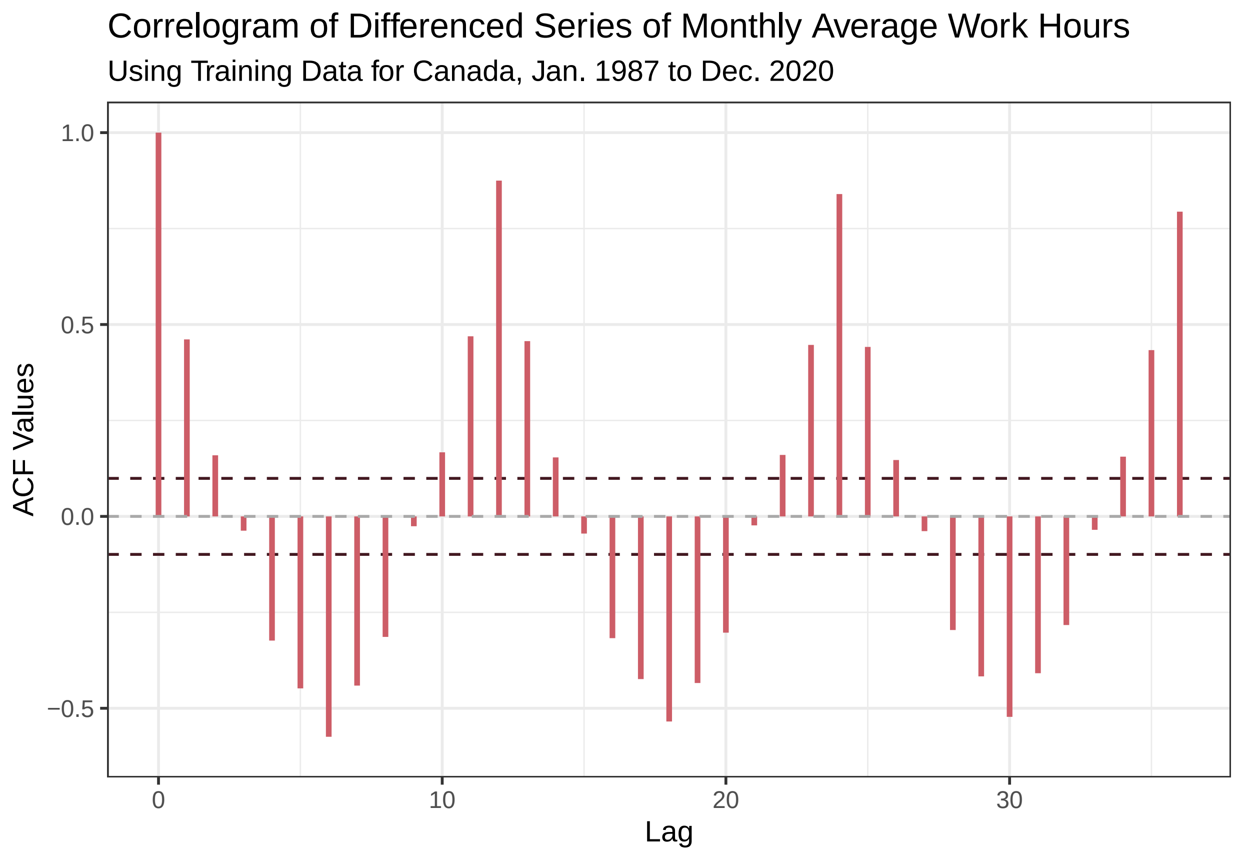
```r
print(p6)
```



Correlogram

```r
p7data = data.frame(
  h = 0:36,
  rh = acf(hours_train_diff, plot = FALSE, lag.max = 36)$acf
)
```

```r
print(p7)
```



Comment on what you observe.

While the single-iteration ($d = 1$) of sequential differencing at lag 1 appears to have helped to mitigate the trend in the original training data, as seen in the plot of the differenced training time series, there still appears to be a notable seasonal component in the data. We know that this is likely the case due to the sinusoidal component in the correlogram, which is not really decreasing as a function of lag as we would like to see. This periodicity is indicative of a lingering seasonal component that we still need to difference out or remove via other methods.

Part B: Seasonal Differencing

Apply seasonal differencing to remove seasonal variation. Since we have monthly data, $s = 12$ seems sensible.

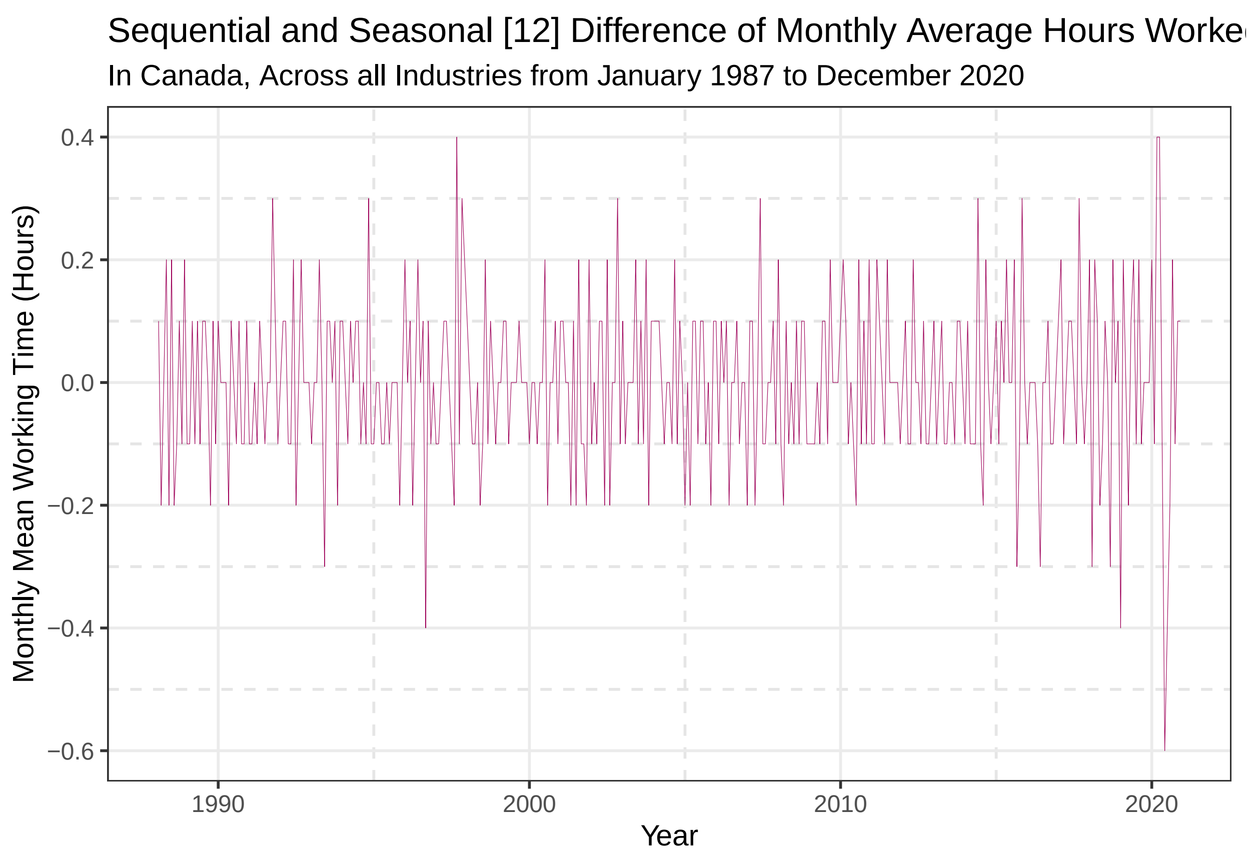
```
hours_train_diff_s = diff(hours_train_diff, lag = 12, differences = 1)
```

Plot the resulting differenced time series along with its sample acf and pacf.

Series

```
p8data = fortify.zoo(hours_train_diff_s)
```

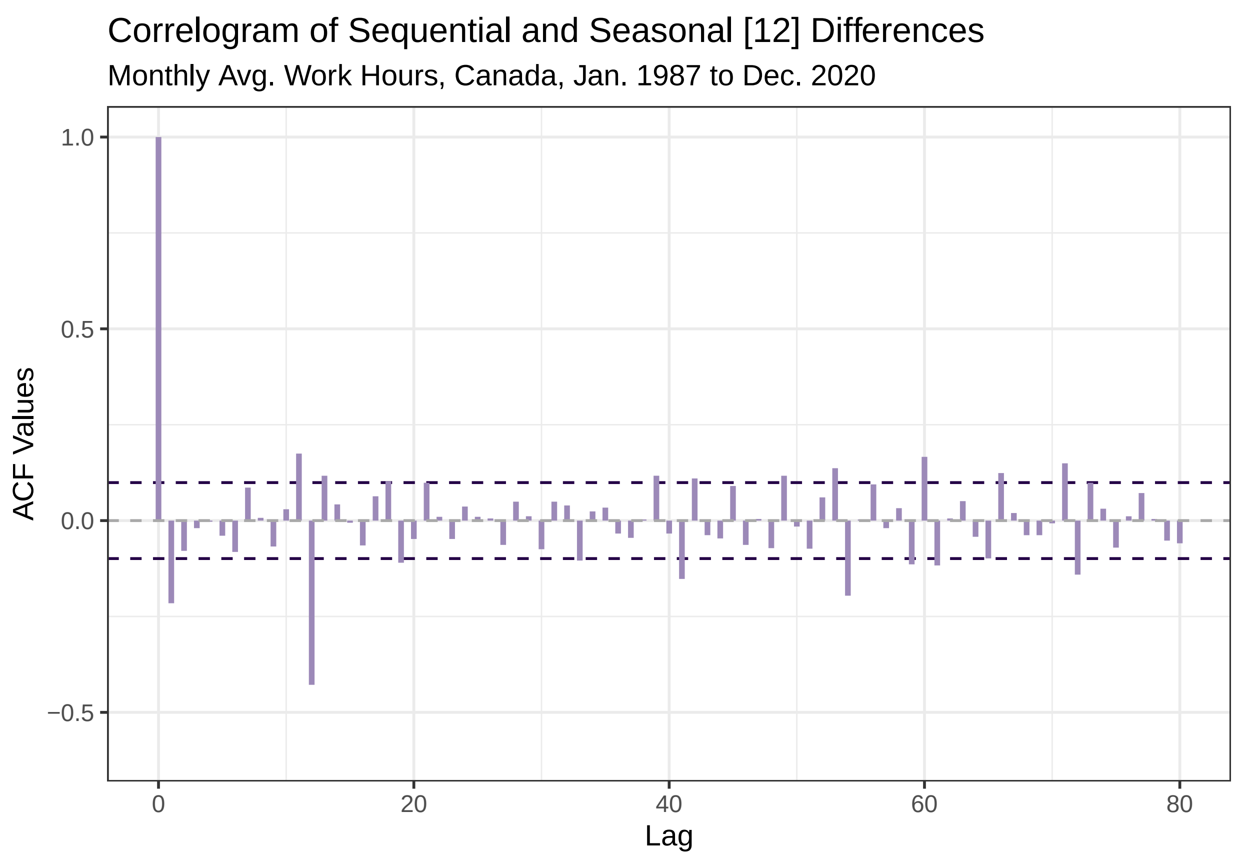
```
print(p8)
```



Sample ACF

```r
p9data = data.frame(
  h = 0:80,
  rh = acf(hours_train_diff_s, plot = FALSE, lag.max = 80)$acf
)
```

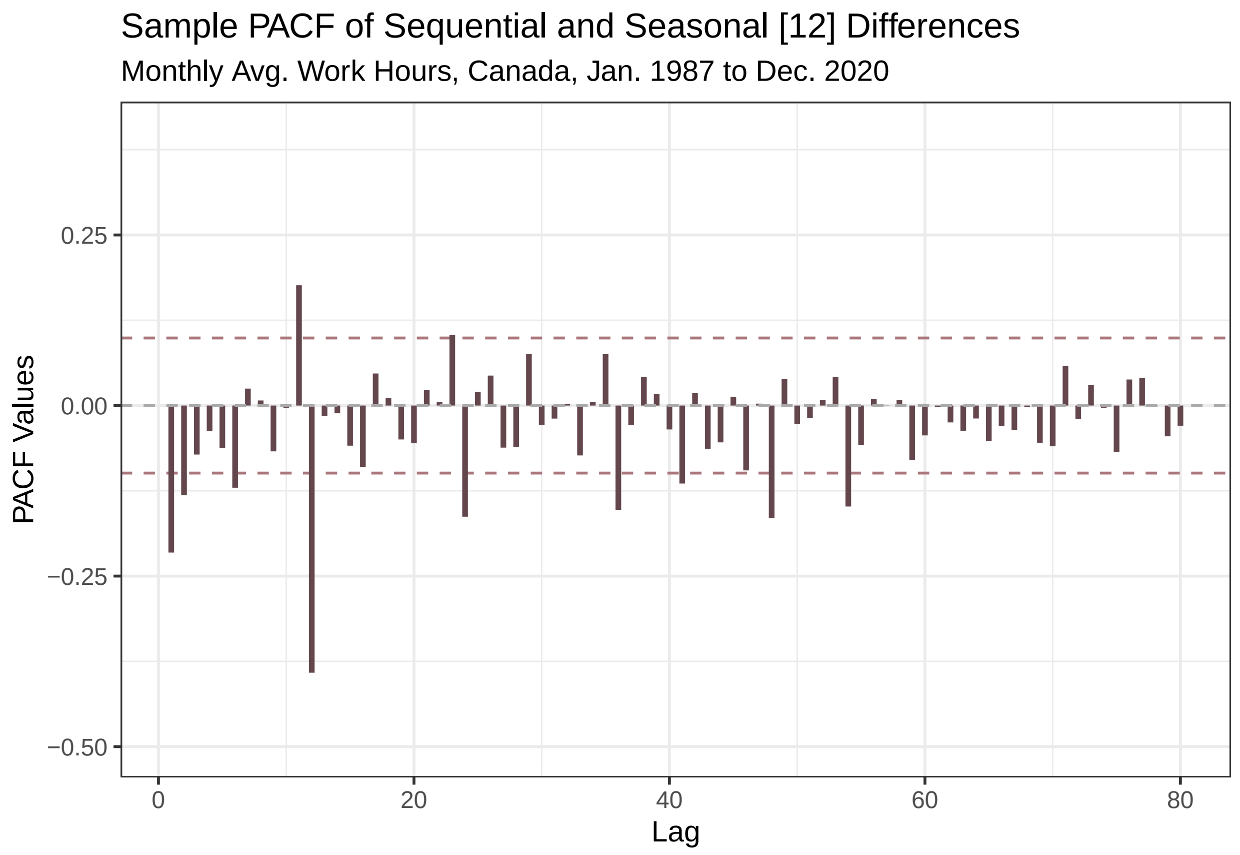
```r
print(p9)
```



Sample PACF

```
p10data = data.frame(
  h = 1:80,
  rh = pacf(hours_train_diff_s, plot = FALSE, lag.max = 80)$acf
)
```

```
print(p10)
```



<u>Comment on what you observe:</u>

The seasonal differencing paired with the sequential differencing seems to have done a good job in making the series roughly stationary. We see from the plotted data as well as the correlogram that there isn't evidence of a significant season or trend. Further, the visual complexity of both the ACF and the PACF hints at the fact that there may be a blended model (i.e. a mixture of AR and MA) underlying this process. We will discuss in the next sections exactly what the parameters of the model equation are in this case.

Part C: Difference Parameters.

Based on the results of Part II (a) and (b), specify the values of d, D, and s.

From Part II (a), we applied one sequential difference using `diff`, so we know $d = 1$. Then, in Part II (b), we applied one seasonal difference using `lag = 12`, hence $D = 1$ with $s = 12$.

Part D: SARIMA Parameters

Based on the plots in Part II (b), suggest possible values of justifying your choices.

NOTE: It is very difficult to accurately determine these coefficients precisely, but I will justify my decisions to the best of my ability.

Firstly, let's discuss the presence of an MA component. We henceforth assume stationarity.

In the ACF, we see a fairly large r_h value at $h = 12$, that significantly cuts off for $h > 12$. Unlike a pure MA plot where we would see an expect a very sharp cutoff, in a blended model there will still be other components at work after the MA component.

All of the lags after $h = 12$ for $h \in [13, 80] \subset \mathbb{Z}$ are notably smaller, as the Sample ACF shows.

This would imply that either we have a sequential MA with $q = 12$ or a seasonal MA with $Q = 1$. By the prinicple of parsimony, we prefer the latter. Also, this conclusion would imply that it is likely that there is a q component.

To explain, we recall the right-hand side of a SARIMA model, assuming $Q = 1$.

$$\begin{aligned} \text{RHS} &= \Theta(B^s)\theta(B)Z_t \\ \text{RHS} &= (1 + \tilde{\beta}B^{12})\theta(B)Z_t \\ \text{RHS} &= \theta(B)Z_t + \theta(B)\tilde{\beta}Z_{t-12} \end{aligned}$$

This would help explain the decently large "spike" at lag 12 (also perhaps suggesting that $\tilde{\beta} < 0$.)

However, is there a sequential component $q > 0$? By the above, under the assumption that $Q = 1$, if $\theta(B)$ was of an order greater than zero, there would be additional components to the model associated with Z_{t-1} and Z_{t-s-1}, which we would expect to see reflected in the lags of the ACF.

The most likely visually is that $q = 1$. An explanation of why is facilitated by the equation below:

$$\begin{aligned} \text{RHS} &= (1 + \beta B)Z_t + (1 + \beta B)\tilde{\beta}Z_{t-12} \\ \text{RHS} &= Z_t + \beta Z_{t-1} + \tilde{\beta}Z_{t-12} + \beta\tilde{\beta}Z_{t-13} \end{aligned}$$

In other words, from our understanding of MA models, we would see large in magnitude r_1 and (slightly less large) r_{13}; this isn't explainable by additional Q components, as they can only increase Z_{t-h} by a factor of 12. To support this theory we see in the sample ACF a "spike" at lag 1 less than $-2/\sqrt{n}$ and a shorter spike at lag 13, slightly above $2/\sqrt{n}$, supporting our idea of a sequential MA component. For this reason, $q = 1$ seems plausible (noting that other $q > 0$ are also plausible.)

But what about AR components? It would be a bit challenging to use the PACF to identify the order p of any AR components, as in this course we only know graphical methods for this interpretation from pure AR models. Regardless of if the PACF can be used for blended models, it is very difficult to tell from the ACF whether or not an AR component *actually exists.* Importantly, the PACF is a tool to quantify the parameters of an AR model once we've identified one using the ACF of the stationary time series. There doesn't seem to be any decaying exponential pattern to the ACF, nor does there seem to be a dampened sine curve - these are the two telltale signs of an AR component. While it is tempting to examine spikes in the PACF, they can be utilized do identify p or P if and only if we know from the ACF that AR component(s) exist. From the plots above, such a scenario doesn't seem plausible as it exhibits none of the characteristic patterns described previously. Hence, we can reason it's likely that $P = p = 0$.

Part E: Iterative AIC

Now, we see the Akaike's Information Criterion (AIC) to select the model based on the training dataset in Part I. We fix the values of p and P to 0 as suggested in Part II (d),

We will consider $q \in [0, 5]$ and $Q \in [0, 5]$ and compute all AIC values and then select the values of q and Q according to these computations.

First, we declare our fixed p, P, s, d and D values, then iterate through the different 36 models that can be developed and report the AIC for each.

```r
p = 0; P = 0; s = 12; d = 1; D = 1; s = 12
# declare empty matrix
results = matrix(0, nrow = 6, ncol = 6)
# with legible names
rownames(results) = c("q=0", "q=1", "q=2","q=3","q=4","q=5")
```

```
colnames(results) = c("Q=0", "Q=1", "Q=2","Q=3","Q=4","Q=5")
# iterate through all candidates
for (q in 0:5){
  for (Q in 0:5){
    results[q+1, Q+1] = arima(hours_train,
                              order = c(p, d, q),
                              seasonal = c(P, D, Q))$aic
  }
}
```

The AIC results are summarized in the table below.

```
kable(round(results, 3), caption = "AIC Values for Varying $(q, Q)$")
```

Table 6: AIC Values for Varying $(q, Q)$

	Q=0	Q=1	Q=2	Q=3	Q=4	Q=5
q=0	-464.461	-613.121	-611.440	-609.462	-612.427	-612.305
q=1	-487.115	-616.650	-614.907	-612.958	-616.029	-616.225
q=2	-489.810	-618.001	-616.272	-614.388	-617.448	-618.428
q=3	-488.932	-622.962	-621.387	-619.411	-623.477	-623.785
q=4	-487.747	-624.882	-623.135	-621.160	-625.432	-625.629
q=5	-488.378	-623.821	-621.895	-619.917	-624.506	-624.633

Now, we fit the model the model chosen according to the AIC Table and print the values of the estimated parameters and report the AIC value for the model.

According to this table, the best model corresponds to:

```
as.numeric(which(results == min(results), arr.ind = TRUE))-1 ; q = 4; Q = 5
```

```
## [1] 4 5
```

Which corresponds to $q = 4$ and $Q = 5$. It has an AIC of approximately -625.63.

Hence, we will fit a model of the form:

$$\text{SARIMA}(\underset{p}{0}, \underset{d}{1}, \underset{q}{4}) \times (\underset{P}{0}, \underset{D}{1}, \underset{Q}{5})_{\underset{s}{12}}$$

```
sarima_fit = arima(hours_train,
                   order = c(p, d, q),
                   seasonal = list(order = c(P, D, Q), period = s))
```

Finally, we print the values of the estimated parameters along with the AIC value for the model. We let $\hat{\beta}_i$ be the i-th estimated standard MA coefficient for $i \in [1, 4]$ and $\breve{\beta}_j$ be the j-th estimated seasonal MA component for $j \in [1, 5]$. We let $\hat{\sigma}^2$ be the estimated variance of the process.

For conciseness, we round each estimate to three decimal places.

```r
# get all the desired coefficients into a data frame
sarima_coefs = t(data.frame(Est = round(
  c( sarima_fit$coef[1:4], sarima_fit$coef[5:9],
     sarima_fit$sigma2, sarima_fit$aic), 3 )))
# produce a nice table
kable(sarima_coefs, "latex", escape = FALSE,
      col.names = fancycolnames, caption = "Fitted SARIMA Coefficients")  %>%
      kable_styling(latex_options = "hold_position") %>%
      kable_styling(position = "center")
```

Table 7: Fitted SARIMA Coefficients

	$\hat{\beta}_1$	$\hat{\beta}_2$	$\hat{\beta}_3$	$\hat{\beta}_4$	$\breve{\beta}_1$	$\breve{\beta}_2$	$\breve{\beta}_3$	$\breve{\beta}_4$	$\breve{\beta}_5$	$\hat{\sigma}^2$	AIC
Est	-0.164	-0.126	-0.13	-0.101	-0.758	0.048	-0.107	0.067	0.093	0.011	-625.629

Part F: Model Diagnostics:

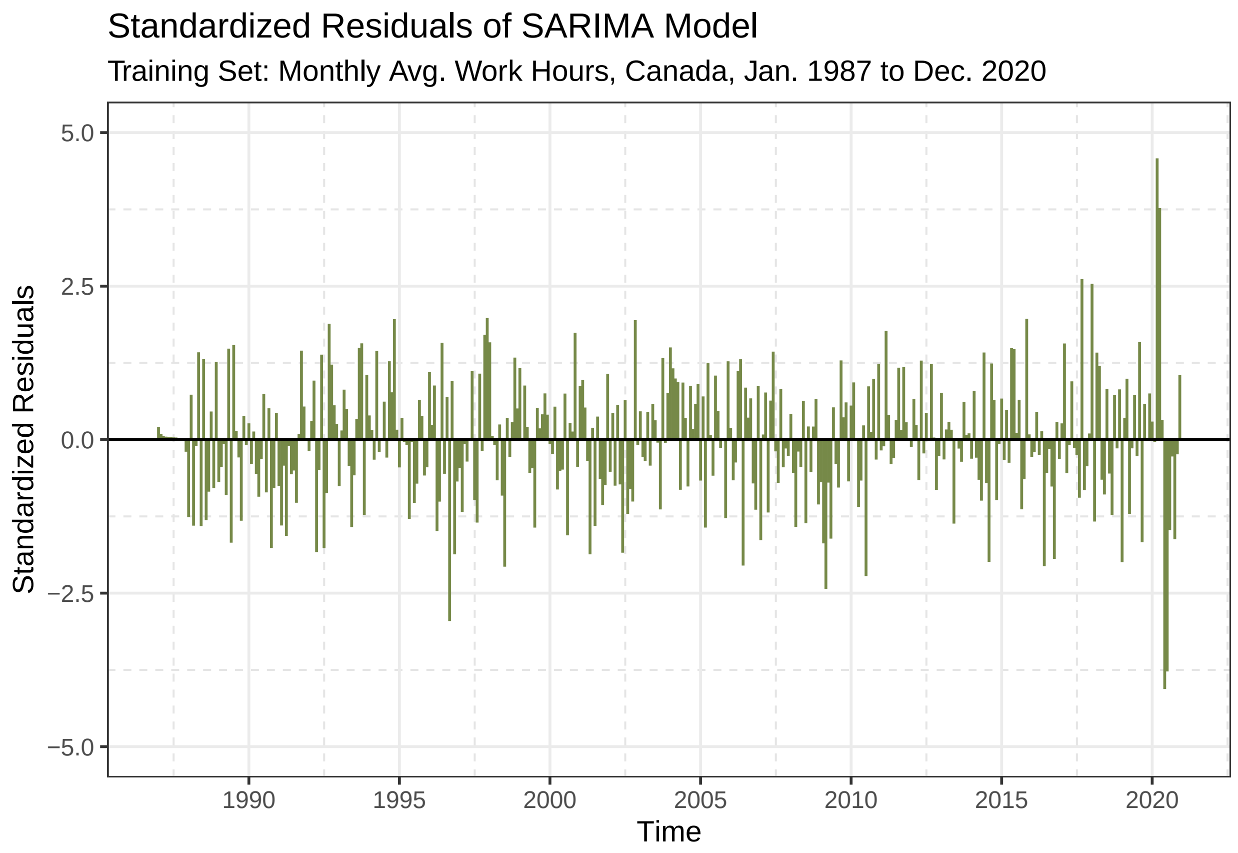
Perform the model diagnostics for the model above and comment on the goodness of fit for your chosen model.

Again, we compute the Ljung-Box p-values by the process detailed in the previous question. We manually replicate the results of `tsdiag` by providing plots of the standardized residuals, residual acf and Ljung-Box statistics. As we create these plots, comments on them will be provided.

Standardized Residuals

```r
# standardized residuals
p2fdata = data.frame(
  Time = as.Date(time(hours_train)),
  resids = sarima_fit$residuals / sd(sarima_fit$residuals))
```

```r
print(p2f)
```

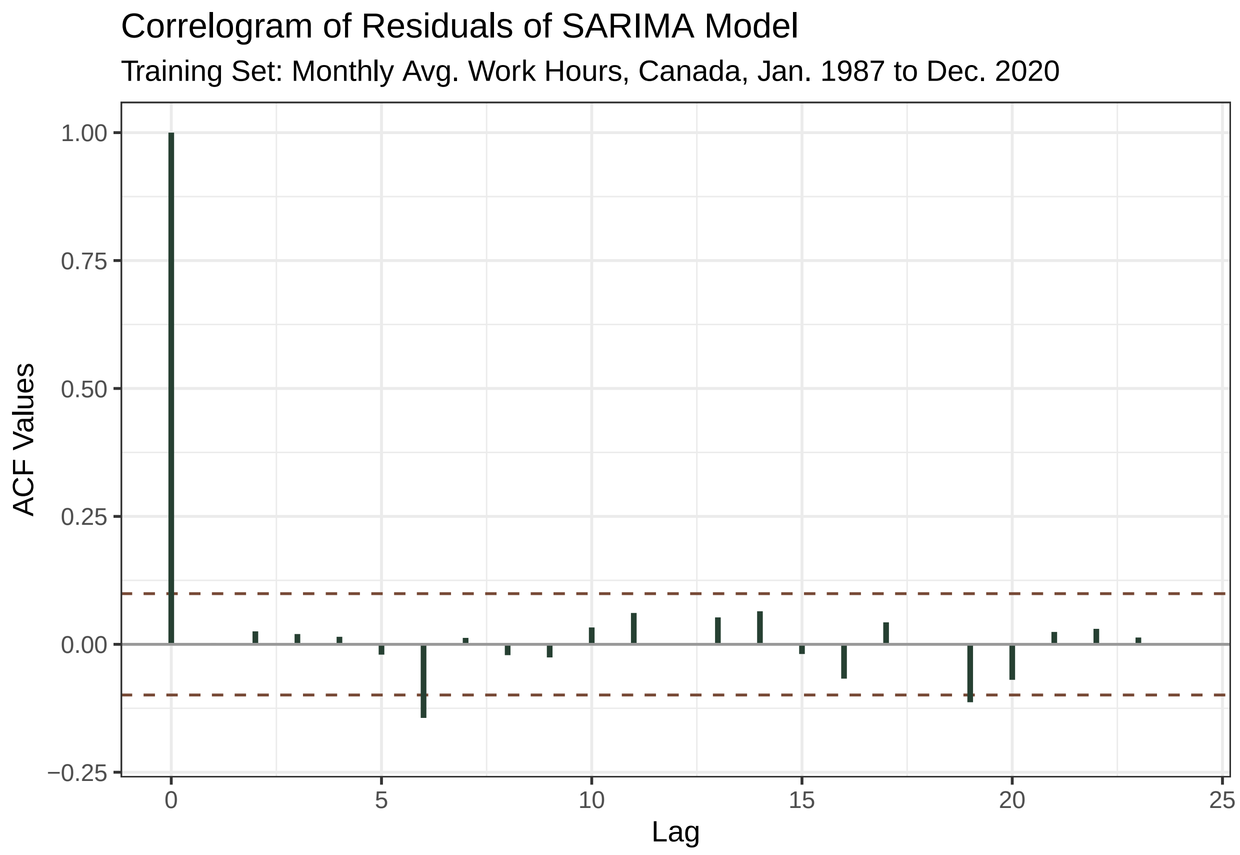


Comments: There's no noticeable long-term pattern to the residuals - supporting the theory that the model is well fitting - there is a sharp increase in the residual values from around 2020 onwards. This could be due to the large fluctuations in working hours caused by the COVID-19 pandemic. However, beyond these extreme values, the residuals appear to be randomly distributed about zero, as we would anticipate for a well-fitting model.

Residual ACF

```
# residual acf
p2f2data = data.frame(
  h = 0:24,
  rh = acf(sarima_fit$residuals, lag.max = 24, plot = F)$acf
)
```

```
print(p2f2)
```

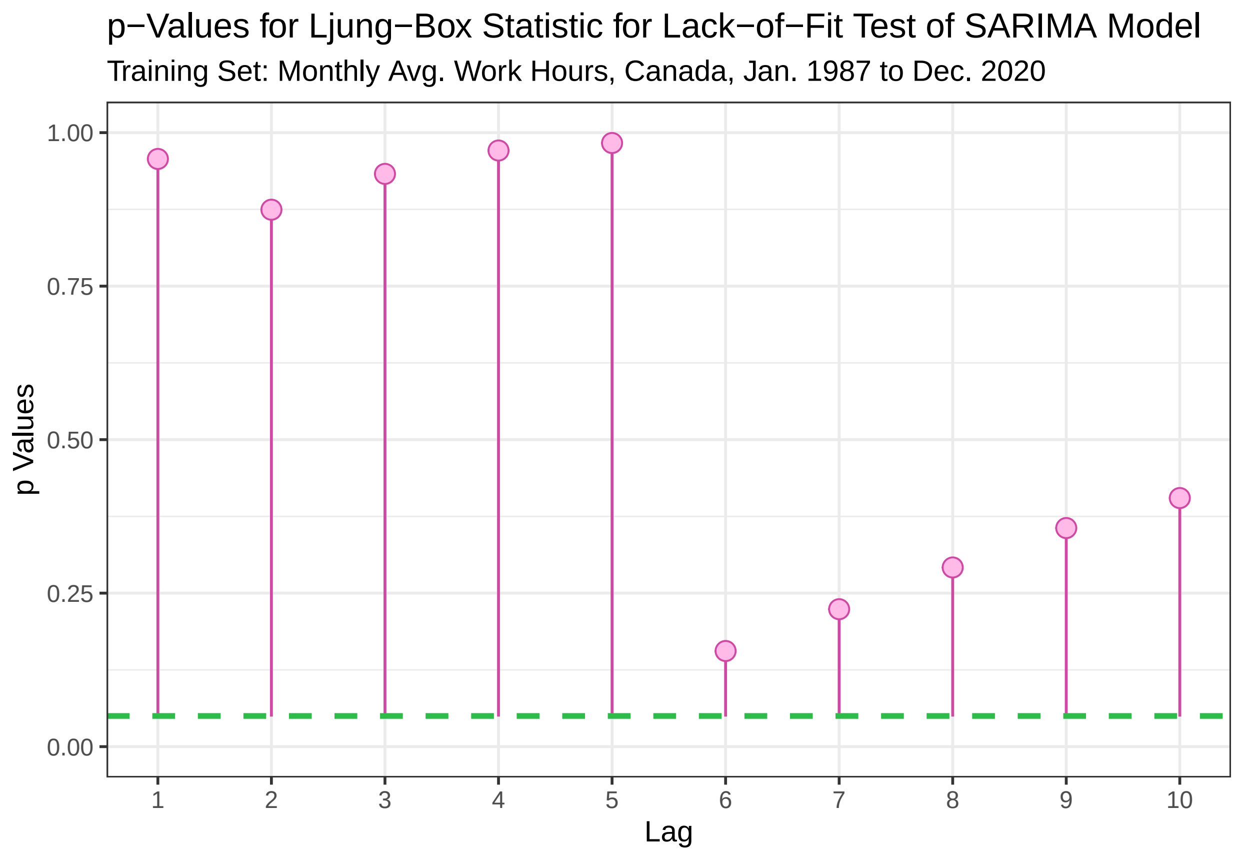


Comments: The ACF of the residuals seems to be mostly white noise, as most observations are below the $\pm 2/\sqrt{n}$ boundaries. This is what we would expect to see for a well-fitting model. The fact that there are ACF values slightly beyond the white noise boundaries for lags 6 and 19 may be a cause for some concern; however, there isn't a consistent enough pattern to suggest lack-of-fit.

Ljung-Box P-Values

```
n = length(hours_train); lagmax = 10
# get rho(h) values of residuals
rh_s  = acf(sarima_fit$residuals, lag.max = lagmax, plot = F)$acf
# compute Ljung-Box statistic
lb_s =  sapply(2:(lagmax+1),
         function(h){
           Q = n*(n+2)*sum( rh_s[2:h]^2 / (n - h))
           pchisq(Q, df = h - 1, lower.tail = F)})
# prep data for plotting
p2f3data = data.frame(
  h = 1:lagmax,
  lb_stats = lb_s
)
```

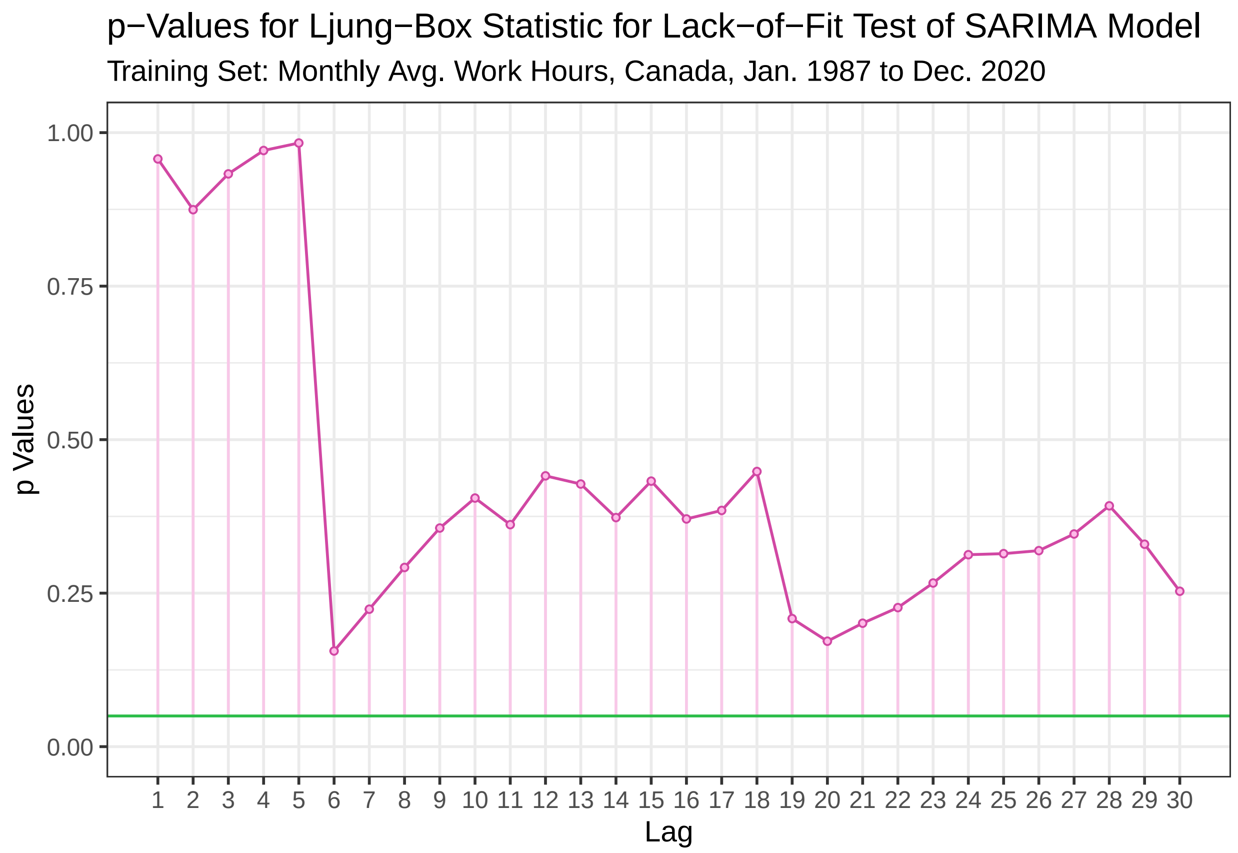
```
print(p2f3)
```



Comments and General Summary: From the above plot of Ljung-Box Statistic p-values, we see a sudden dip of confidence, i.e. $\mathbb{P}(\text{Model} \mid H_0 : \text{Well-Fitting})$, around lag $h = 6$, which is quite concerning. This could be indicative of some phenomenon in our data beginning at the first half-season (recalling $s = 12$) that our current SARIMA fit is not capturing. An extended view of the Ljung-Box statistic p-values up to Lag 30 shown below demonstrates that this "dip" in the p-values isn't just a one time occurrence:

```r
# increase lag and run code again
lagmax = 30
```

```r
plot(p2f4)
```



As the above shows, there may be some additional unidentified periodic component causing these occasional "shocks" to the confidence of our model. However, the observed p-values for all of the lags between 15 and 30 (common values for these tests according to the slides) are still comfortably above $\alpha = 0.05$ implying that there is not a significant lack-of-fit to the SARIMA model. This pattern of sudden decreases in p-values is concerning, as well as the overall decline in the p-values over time. Notably, the Ljung-Box portmanteau test is robust for different lag lengths, and hence under H_0 the p-values should be greater than α regardless of the lag, so this gradual decline may indicate that there is something about our current SARIMA model is missing and would warrant further investigation.

To summarize, the diagnostics shown across the three plots equivalent to the standard model diagnostics of `tsdiag` show that while the SARIMA model selected seems to not have a significant lack-of-fit, there are some concerning patterns in the diagnostic that may shed doubt on whether this is the best model *overall* for these data. To me, the diagnostics indicate that there are likely to be better-performing models out there; however, the chosen one isn't completely mis-specified.

Part G: Plot and Results

Now, we predict the mean monthly working hours records for the period from January 2021 to December 2023 based on the model fitted in Part II (f).

NOTE: Again because we are tasked to create a detailed plot, the plotting code will not be hidden.

We construct the 95% prediction intervals and forecasts by the same method discussed in detail in the previous question. In this case, our largest value ahead is 36, so we make forecasts $\forall \ell \in \{1, 2, \dots, 36\} \subset \mathbb{Z}$.

```
# declare constants
alpha = 0.05
Z = qnorm(1-alpha/2)
```

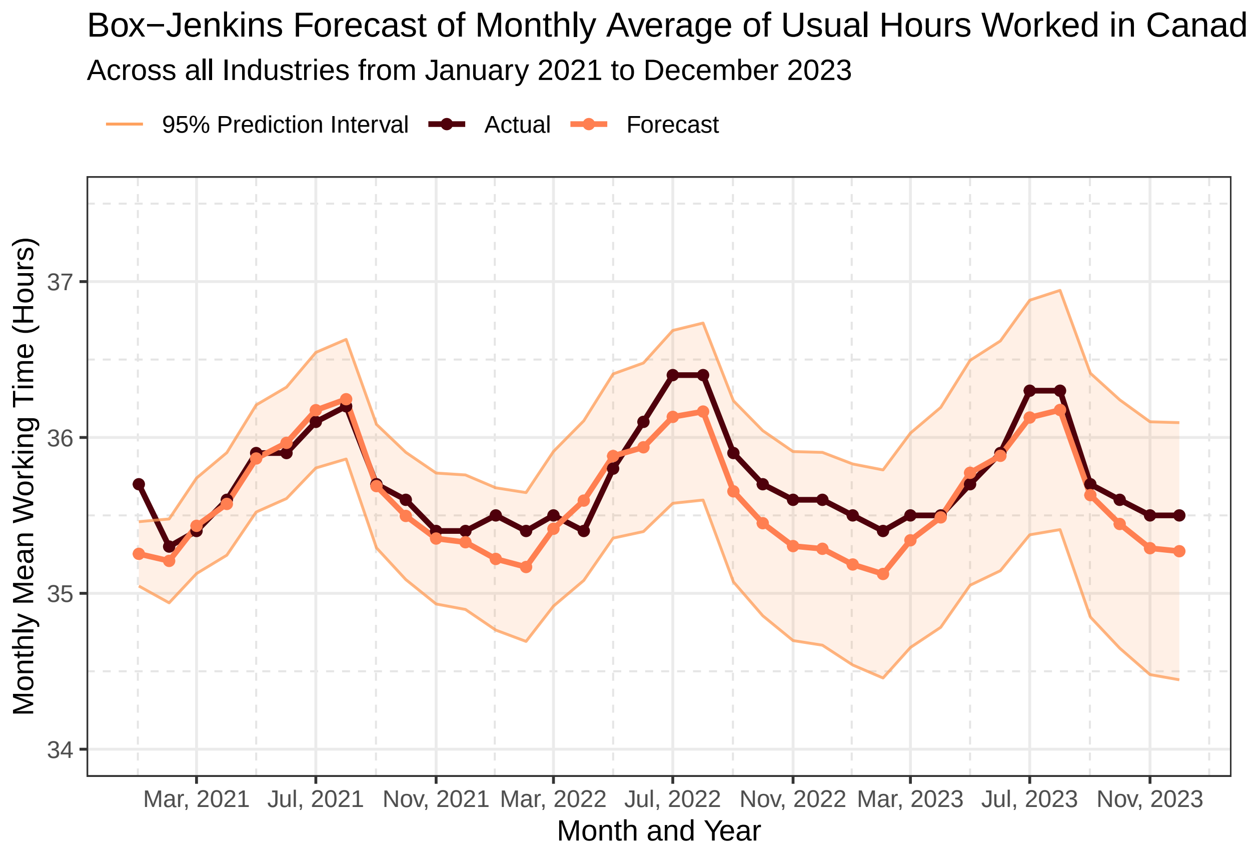
```r
# make forecast object
s_forecast = predict(sarima_fit, n.ahead = length(hours_test))
# get point forecasts
s_point = s_forecast$pred
# prediction intervals
s_Low = s_point - Z*s_forecast$se
s_High = s_point + Z*s_forecast$se

# put everything into a data frame for plotting
p2edata =  data.frame(
  Time = as.Date(time(hours_test)),
  Observed = as.numeric(hours_test),
  Forecast = as.numeric(s_point),
  Lower = as.numeric(s_Low),
  Upper = as.numeric(s_High)
)
```

Now that the data is formatted nicely, we can create the plot. Note that we have consistency in our naming conventions such that the code is easily translatable from previous forecasting plots.

```r
p2e = ggplot(data = p2edata, aes(x = Time)) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper), fill = "#FF9B54", alpha = 0.15) +
  # lines for obsv, pred and interval bounds
  geom_line(aes(y = Observed, color = "Actual"),  lwd = 1) +
  geom_line(aes(y = Forecast, color = "Forecast"),lwd = 1) +
  geom_point(aes(y = Observed, color = "Actual")) +
  geom_point(aes(y = Forecast, color = "Forecast")) +
  geom_line(aes(y = Lower, color = "95% Prediction Interval"), alpha = 0.75) +
  geom_line(aes(y = Upper, color = "95% Prediction Interval"), alpha = 0.75) +
  # legend and colour assignment
  scale_color_manual(name = "",
    values = c(
    "Actual" = "#4F000B",
    "Forecast" = "#FF7F51",
    "95% Prediction Interval" = "#FF9B54"
  )) +
  # customize x-axis for nice dates
  scale_x_date(date_breaks = "4 month",  date_labels = "%b, %Y") +
  # add titles with units
  labs(
    title = "Box-Jenkins Forecast of Monthly Average of Usual Hours Worked in Canada",
    subtitle = "Across all Industries from January 2021 to December 2023",
    x = "Month and Year",
    y = "Monthly Mean Working Time (Hours)"
  ) +
  theme_bw() + ylim(34, 37.5)+
  theme(panel.grid.minor =  element_line(
    color = "grey90",
    linewidth  = 0.35,
    linetype = "dashed"
  ), legend.position = "top", legend.justification = "left",
  legend.margin = margin(0,0,0,0))
```

```
# show results
print(p2e)
```



Comments: Overall, the forecasting procedure seems to perform very well! For the vast majority of the test data, the forecasted values are very near the actual values and the actual values are almost always within the 95% prediction interval. While there are the occasional instances in which the Box-Jenkins method over-estimates the mean monthly working time, it seems to slightly underestimate more often. The only instances in which the forecast and the interval do not capture the truth is near the beginning of the forecast (early 2021.) As we discussed in the plot of standardized residuals, the overall time series was disrupted from late 2019 to early 2021 as the patterns of the global workforce were changed significantly by the COVID-19 pandemic. So, it's not surprising that the forecast misses the true values for early 2021. It is encouraging to see that as time progresses, and many people returned to work after the pandemic, that the Box-Jenkins forecast continues to perform quite well. Notably, when compared to our "naive" forecasting procedure from Assignment 1, the Box-Jenkins forecast is far superior.

Part 3: Holt-Winters Method

Part A: Model Specification

When we determining the parameters of the model in the Box-Jenkins method, observed removed both trend *and* seasonality (by sequential and seasonal differencing), hence we will use a nonzero β in the Holt-Winters model to keep the comparisons in the next section consistent.

Further, we will again use an additive seasonal effect, which inherits the `Frequency` of the time series as the period. In this case, $p = 12$.

Since we now have a trend component to the model, we define the following set of equations for our Holt-Winters model given $\{x_t\}_{t=1}^{N}$ for discrete (integer-valued) t:

$$\begin{aligned}
\text{Trend: } & T_t = \beta(L_t - L_{t-1}) + (1-\beta)T_{t-1} \\
\text{Seasonal: } & I_t = \gamma(x_t - L_t) + (1-\gamma)I_{t-p} \\
\text{Level: } & L_t = \alpha(x_t - I_{t-p}) + (1-\alpha)(L_{t-1} + T_{t-1}) \\
\text{Forecast: } & \hat{x}_t(\ell) = L_t + \ell\, T_t + I_{t-p+\ell} \\
\text{Where: } & \{\beta, \alpha, \gamma\} \in [0,1] \subseteq \mathbb{R}
\end{aligned}$$

We use the `HoltWinters()` function to estimate $\{\beta, \alpha, \gamma\}$ and report them below. While not explicitly specified in the table title (for brevity) this fit is for the Canada Mean Hours Worked Dataset.

```r
hw_hours = HoltWinters(hours_train, seasonal = "additive")
# data frame for nice formatting
hw_hours_results_coefs = (data.frame(
  Alpha = round(hw_hours$alpha, 4),
  Beta  = round(hw_hours$beta, 4),
  Gamma = round(hw_hours$gamma, 4)
))
# show results in a table
rownames(hw_hours_results_coefs) = NULL
kable(hw_hours_results_coefs, "latex", escape = FALSE,
      col.names = c("$\\hat{\\alpha}$", "$\\hat{\\beta}$", "$\\hat{\\gamma}$"),
      caption = "Fitted Holt-Winters Coefficients")  %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 8: Fitted Holt-Winters Coefficients

$\hat{\alpha}$	$\hat{\beta}$	$\hat{\gamma}$
0.6596	0.0017	0.891

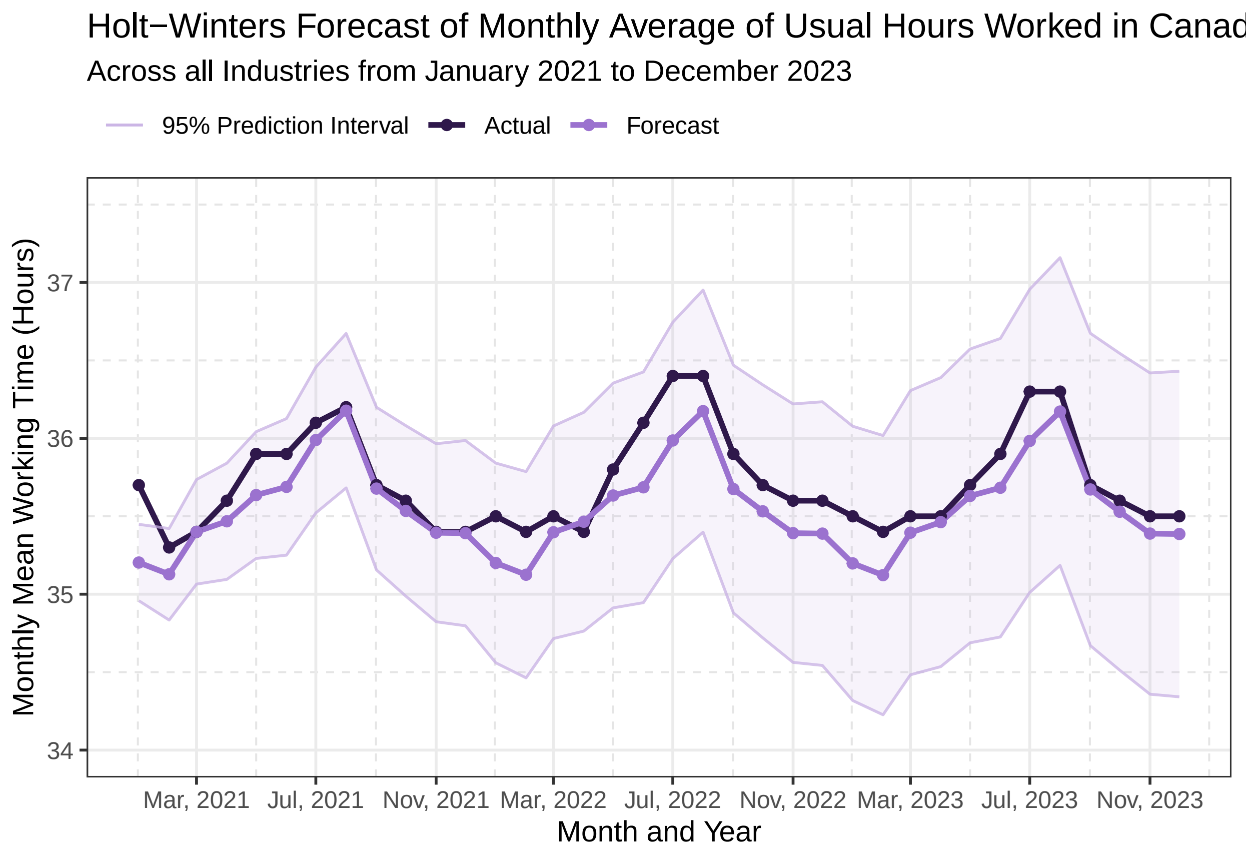
Part B: Prediction and Plotting

Now, we the predict mean monthly working hours for the period from January 2021 to December 2023 based on the Holt-Winters filtering.

```r
# make forecast
hw_f_hours = data.frame(predict(hw_hours, n.ahead = 36,
                        prediction.interval = TRUE, level = 0.95))
# format into data frame
p3bdata = data.frame(
  Time = as.Date(time(hours_test)),
  Observed = as.numeric(hours_test),
  Forecast = as.numeric(hw_f_hours$fit),
  Lower = as.numeric(hw_f_hours$lwr),
  Upper = as.numeric(hw_f_hours$upr)
)
```

In one figure, we will plot the test dataset along with the forecasts and corresponding 95% prediction intervals. We use the same general plotting structure we've been using throughout the assignment.

```r
# make fancy as heck plot
p3b = ggplot(data = p3bdata, aes(x = Time)) +
  geom_ribbon(aes(ymin = Lower, ymax = Upper), fill = "#C8B1E4", alpha = 0.15) +
  # lines for obsv, pred and interval bounds
  geom_line(aes(y = Observed, color = "Actual"),  lwd = 1) +
  geom_line(aes(y = Forecast, color = "Forecast"),lwd = 1) +
  geom_point(aes(y = Observed, color = "Actual")) +
  geom_point(aes(y = Forecast, color = "Forecast")) +
  geom_line(aes(y = Lower, color = "95% Prediction Interval"), alpha = 0.75) +
  geom_line(aes(y = Upper, color = "95% Prediction Interval"), alpha = 0.75) +
  # legend and colour assignment
  scale_color_manual(name = "",
    values = c(
    "Actual" = "#2F184B",
    "Forecast" = "#9B72CF",
    "95% Prediction Interval" = "#C8B1E4"
  )) +
  # customize x-axis for nice dates
  scale_x_date(date_breaks = "4 month",  date_labels = "%b, %Y") +
  # add titles with units
  labs(
    title = "Holt-Winters Forecast of Monthly Average of Usual Hours Worked in Canada",
    subtitle = "Across all Industries from January 2021 to December 2023",
    x = "Month and Year",
    y = "Monthly Mean Working Time (Hours)") +
  theme_bw() + ylim(34, 37.5)+
  theme(panel.grid.minor =  element_line(
    color = "grey90",
    linewidth  = 0.35,
    linetype = "dashed"
  ), legend.position = "top", legend.justification = "left",
  legend.margin = margin(0,0,0,0),
  plot.title = element_text(size = 13))
print(p3b)
```



Comments: The Holt-Winters forecasting procedure also seems to perform well! Visually, it sticks quite close to the true results throughout the test set, although it seems to be under-estimating the monthly mean working time slightly. Unlike the Box-Jenkins forecast which occasionally over-estimated the truth, the fitted Holt-Winters forecast doesn't seem to noticeably over-estimate at all, which is interesting. In addition, the 95% prediction interval for the Holt-Winters method is wider than that of the Box-Jenkins method, especially as ℓ increases. In all, however, it seems that this model is capturing the truth well. Except for at the beginning of the forecast (an anomalous time discussed in the previous question) the true average of usual hours worked in Canada between January 2021 and December 2023 seems to be captured by the interval.

Part C: Model Comparison

Let x_t be the true value of the test time series at time t, for $t \in \{(t+1),(t+2),\dots,(t+\varkappa)\}$ (assuming the training set ends at time t and is contiguous with the test set), where $\varkappa \in \mathbb{N}$ is the lead at the end of the test set, i.e. $\ell \in [1,\varkappa]$ for the ℓ-step ahead forecast methods. In our case, $\varkappa = 36 = |\text{Test Set}|$ as we have 36 predictions.

Similarly, we let $\hat{x}_t^{\mathrm{H}}(\ell)$ be the ℓ-step ahead Holt-Winters forecast and $\hat{x}_t^{\mathrm{B}}(\ell)$ be the Box-Jenkins forecasted value at time $t+\ell$.

Then, the MSPE of the two models are given by the set:

$$\text{MSPE} = \left\{ \mathrm{M} \in \{\mathrm{H},\mathrm{B}\} : \frac{1}{\varkappa}\sum_{\ell=1}^{\varkappa}\left(x_{t+\ell} - \hat{x}_t^{\mathrm{M}}(\ell)\right)^2 \right\}$$

```r
mspe_H = mean( (as.numeric(hw_f_hours$fit) -  as.numeric(hours_test))^2 )
mspe_B = mean( (as.numeric(s_point) -  as.numeric(hours_test))^2 )
```

Then, we report these findings in the table below:

```r
# define the models
mods_p2 = c("Holt-Winters", "Box-Jenkins")
# compute MSPE for each
mspes_p2 = c(mspe_H, mspe_B)
# report results
kable(t(round(mspes_p2, 5)), col.names = mods_p2,
      caption = "Observed MSPE for Forecasts, Hours Worked Data") %>%
  kable_styling(latex_options = "hold_position") %>%
  kable_styling(position = "center")
```

Table 9: Observed MSPE for Forecasts, Hours Worked Data

Holt-Winters	Box-Jenkins
0.04373	0.03514

Since the Box-Jenkins method as the smaller MSPE, i.e. for the set of MSPEs defined earlier:

$$m_{\text{best}} = \underset{\text{M}\in\{\text{H,B}\}}{\text{argmin}} \left\{\text{MSPE(M)}\right\} = \underset{\text{M}\in\{\text{H,B}\}}{\text{argmin}} \left\{0.0437, 0.0351\right\} = \text{B}$$

In more informal terms, the Box-Jenkins has the smaller MSPE. Since it has the smaller mean-squared prediction error, I would recommend Box-Jenkins. Further, we saw from the plots earlier that the Box-Jenkins method has a narrower prediction interval, which is helpful if we want to be more confident in our forecast.

In addition, the Holt-Winters method consistently underestimates the truth, while the Box-Jenkins method underestimates a bit less frequently and has a balance of under and over estimation around the true values. However, it should be noted that both methods perform very well, and so I would be comfortable recommending either.

It should be noted that if the target audience (i.e. who we are presenting the forecasts to) is not extremely well-versed in the technical side of time series, the Holt-Winters method could be preferable as an understanding of model equations isn't required to understand the source of the forecast. However, this is a purely informal selection method. Purely from MSPE and the plots, the Box-Jenkins model seems better.