Time Series Analysis in Health Research

Forecasting of Weekly Mortality Rate in Canada

Erfan Hoque
Dept. of Community Health and Epidemiology
University of Saskatchewan.

SEA 24th Annual Symposium Workshop, September 26, 2025

We will work in *fpp3* package. Load the library *fpp3*. If *fpp3* package is not installed, then we need to install it first and then load the library.

```
library(fpp3)
library(dplyr)
library(fable.prophet) # needed for Prophet model
```

This document is intended to serve as a rough template for what we covered in the last lectures on time series analysis in R.

Weekly mortality Data

Data Discription:

This dataset (weekly mortality Data) provides weekly age-standardized mortality rates across Canadian provinces, disaggregated by sex, beginning in 2019. The rates are standardized using the 2011 Canadian population, allowing for consistent comparisons across regions and time periods regardless of demographic shifts.

Publisher: Statistics Canada Release (https://www150.statcan.gc.ca/t1/tbl1/en/tv.action?pid=1310087901)

Frequency: Weekly

Geography: Canada and Provinces (excluding Territories)

Set Directory

To check your current working directory in R, use: `getwd()`

This will return the path of the directory where R is currently reading and writing files like this

```
getwd()
```

Read Data

To read a different data you need to change the name of the data based on the data file name.

```
mortality <- read.csv("mortality_model.csv")
```

tisbble objects

We need to create the data in *tisbble* object to use the functions from *fpp3* package.

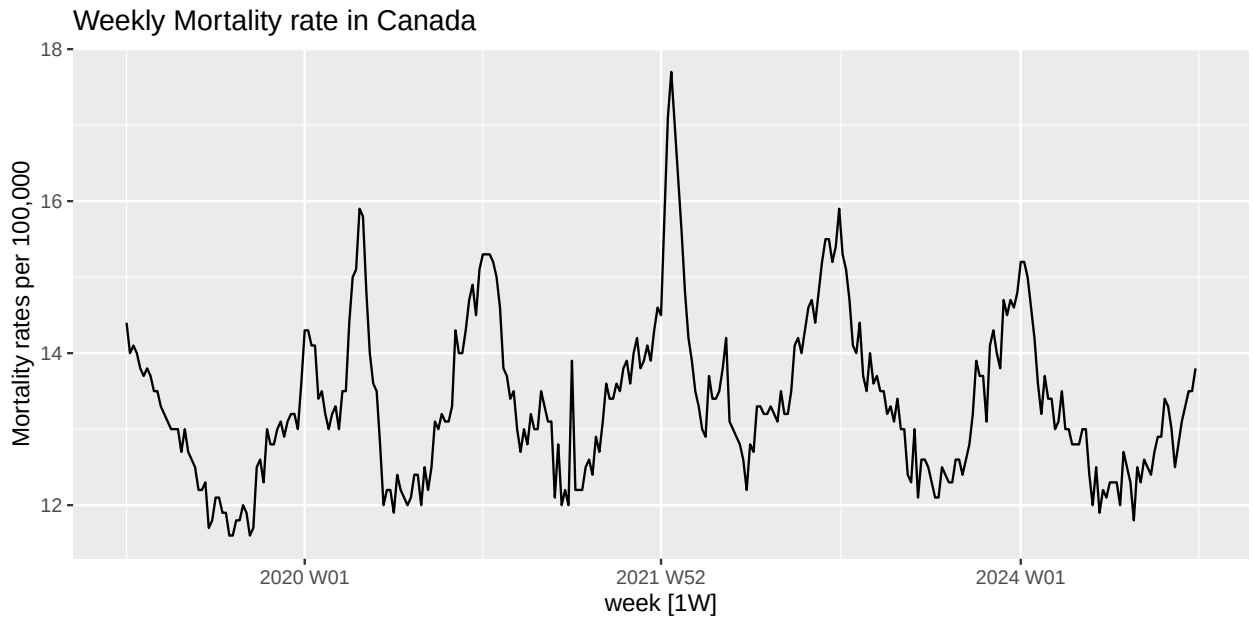
```
mortality_ts <- mortality |>
  mutate(week=yearweek(week),
         Date=as.Date(Date)) |>
  as_tsibble(index = week)
mortality_ts |>
  head(10) # check first few rows of the data
```

```
## # A tsibble: 10 x 5 [1W]
##    Date           week  Rate pre_CA tem_CA
##    <date>       <week> <dbl>  <dbl>  <dbl>
##  1 2019-01-05 2019 W01  14.4 0.0573  -19.4
##  2 2019-01-12 2019 W02  14   0.0557  -22.2
##  3 2019-01-19 2019 W03  14.1 0.0336  -25.4
##  4 2019-01-26 2019 W04  14   0.0533  -23.8
##  5 2019-02-02 2019 W05  13.8 0.0493  -26.2
##  6 2019-02-09 2019 W06  13.7 0.0458  -25.0
##  7 2019-02-16 2019 W07  13.8 0.0377  -22.2
##  8 2019-02-23 2019 W08  13.7 0.0309  -21.7
##  9 2019-03-02 2019 W09  13.5 0.0249  -20.8
## 10 2019-03-09 2019 W10  13.5 0.0235  -18.0
```

Time series plot, patterns and decomposition

Time plot

```
mortality_ts %>% autoplot(Rate) +
  labs(title = "Weekly Mortality rate in Canada",
         y="Mortality rates per 100,000")
```

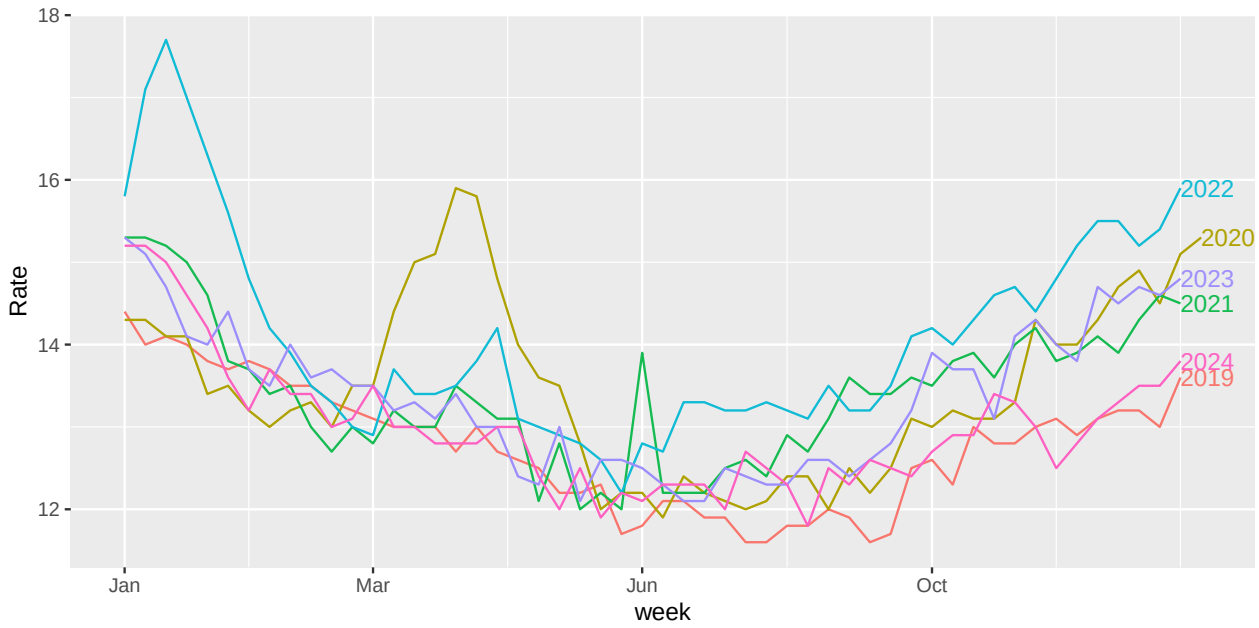


We can see some short trend and seasonality in the mortality rate.

Seasonal plots

A seasonal plot is similar to a time plot except that the data are plotted against the individual "seasons" in which the data were observed. This enables the underlying seasonal pattern to be seen more clearly, and also allows any substantial departures from the seasonal pattern to be easily identified.

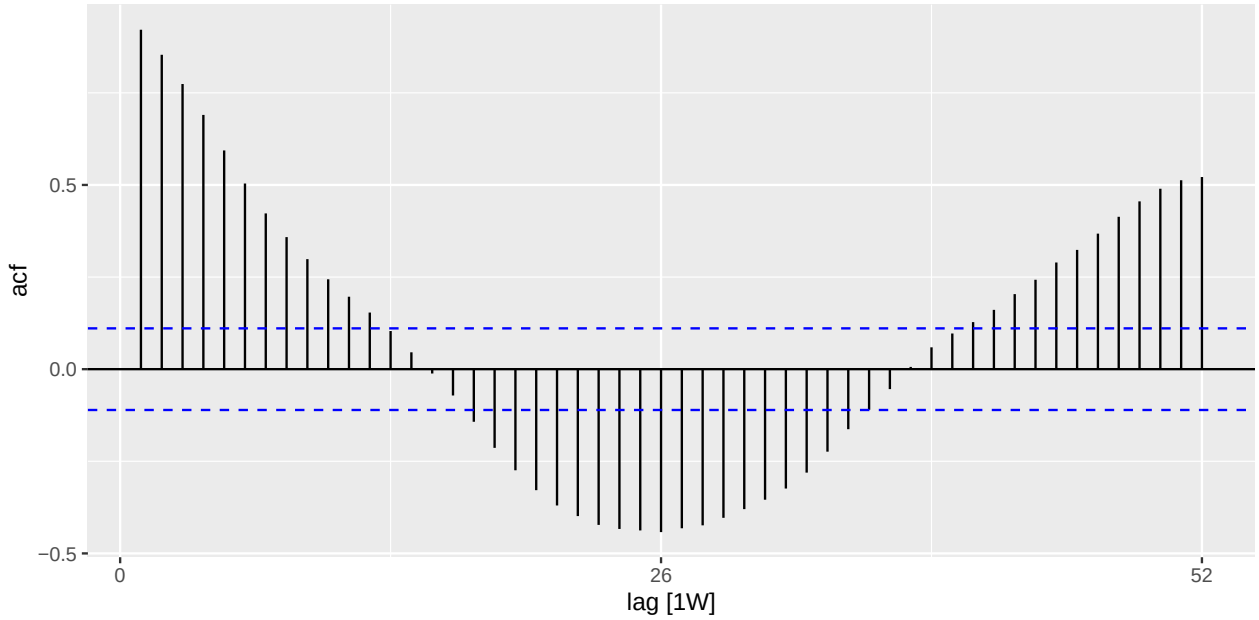
```
mortality_ts %>% gg_season(Rate, labels="right")
```



Autocorrelation

When data have a trend, the autocorrelations for small lags tend to be large and positive. When data are seasonal, the autocorrelations will be larger at the seasonal lags (i.e., at multiples of the seasonal frequency). When data are trended and seasonal, we see a combination of these effects.

```
mortality_ts %>%
  ACF(Rate, lag_max = 52) %>% autoplot()
```



Time series decomposition

STL (Seasonal and Trend decomposition using Loess) is a versatile and robust method for decomposing time series.

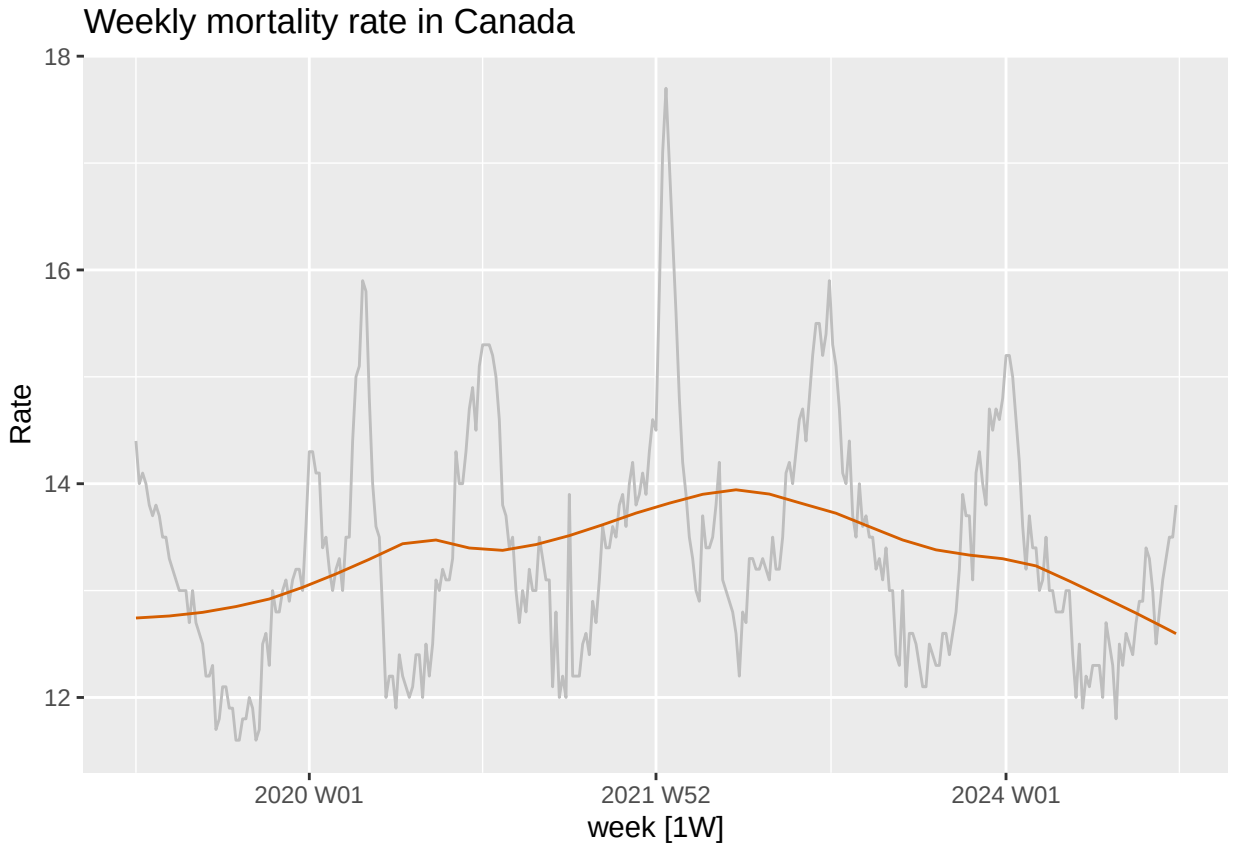
```
dcmp <- mortality_ts %>%
  model(stl = STL(Rate))
components(dcmp)
```

```
## # A dable: 313 x 7 [1W]
## # Key:     .model [1]
## # :        Rate = trend + season_year + remainder
##    .model     week  Rate trend season_year remainder season_adjust
##    <chr>     <week> <dbl> <dbl>       <dbl>     <dbl>         <dbl>
##  1 stl    2019 W01  14.4  12.7       1.49      0.167          12.9
##  2 stl    2019 W02  14    12.7       1.58     -0.324          12.4
##  3 stl    2019 W03  14.1  12.7       1.76     -0.404          12.3
##  4 stl    2019 W04  14    12.7       1.76     -0.506          12.2
##  5 stl    2019 W05  13.8  12.8       1.30     -0.249          12.5
##  6 stl    2019 W06  13.7  12.8       1.06     -0.109          12.6
##  7 stl    2019 W07  13.8  12.8       0.737     0.309          13.1
##  8 stl    2019 W08  13.7  12.8       0.400     0.543          13.3
##  9 stl    2019 W09  13.5  12.8       0.234     0.507          13.3
```

```
## 10 stl    2019 W10  13.5  12.8       0.264     0.476          13.2
## # i 303 more rows
```

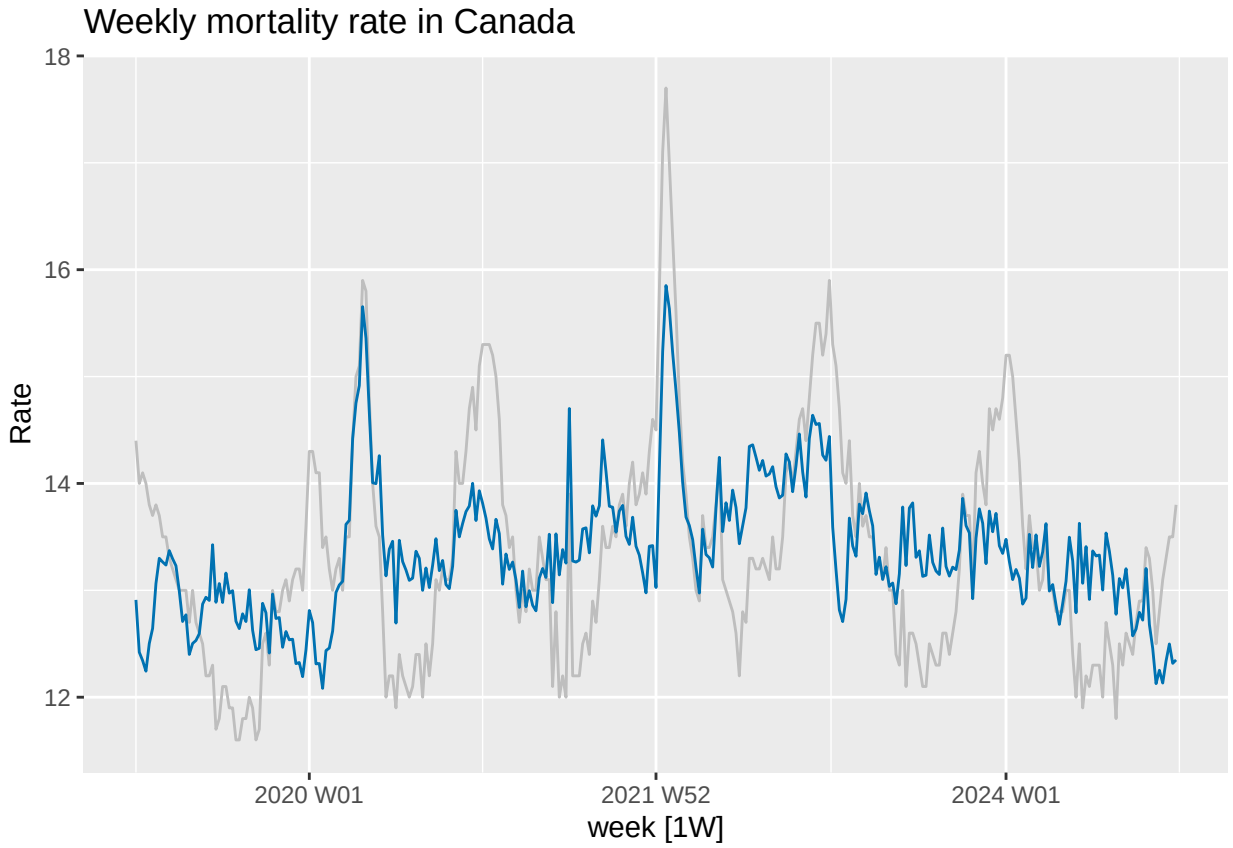
Trend-adjustment

```
mortality_ts %>%
  autoplot(Rate, color='gray') +
  autolayer(components(dcmp), trend, color='#D55E00') +
  labs(y = "Rate", title = "Weekly mortality rate in Canada")
```



Seasonal-adjustment

```
mortality_ts %>%
  autoplot(Rate, color='gray') +
  autolayer(components(dcmp), season_adjust, color='#0072B2') +
  labs(y = "Rate", title = "Weekly mortality rate in Canada")
```



Classical Decomposition

The traditional way to do time series decomposition is called `Classical decomposition`. The simplest estimate of the trend-cycle uses *moving averages* which is an average of nearby points.

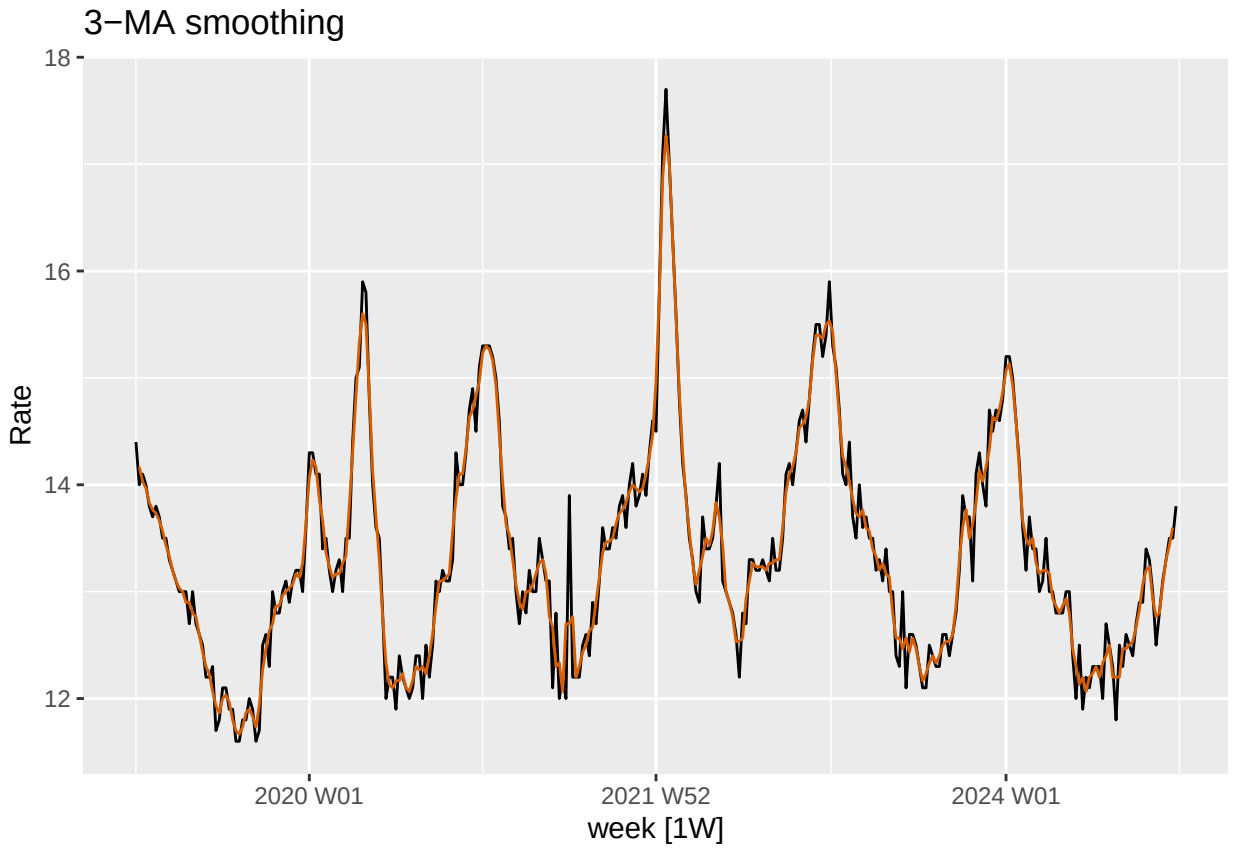
```
mortality_ts_decom <- mortality_ts |>
  mutate(
    `3-MA` = slider::slide_dbl(Rate, mean,
                .before = 1, .after = 1, .complete = TRUE),
    `5-MA` = slider::slide_dbl(Rate, mean,
                .before = 2, .after = 2, .complete = TRUE)
  )

mortality_ts_decom  |>
  head(10)
```

```
## # A tsibble: 10 x 7 [1W]
##    Date           week  Rate pre_CA tem_CA `3-MA` `5-MA`
##    <date>       <week> <dbl>  <dbl>  <dbl>  <dbl>  <dbl>
##  1 2019-01-05 2019 W01  14.4 0.0573  -19.4   NA     NA
##  2 2019-01-12 2019 W02  14   0.0557  -22.2   14.2   NA
##  3 2019-01-19 2019 W03  14.1 0.0336  -25.4   14.0   14.1
##  4 2019-01-26 2019 W04  14   0.0533  -23.8   14.0   13.9
##  5 2019-02-02 2019 W05  13.8 0.0493  -26.2   13.8   13.9
##  6 2019-02-09 2019 W06  13.7 0.0458  -25.0   13.8   13.8
```

```
##  7 2019-02-16 2019 W07  13.8 0.0377  -22.2   13.7   13.7
##  8 2019-02-23 2019 W08  13.7 0.0309  -21.7   13.7   13.6
##  9 2019-03-02 2019 W09  13.5 0.0249  -20.8   13.6   13.6
## 10 2019-03-09 2019 W10  13.5 0.0235  -18.0   13.4   13.4
```

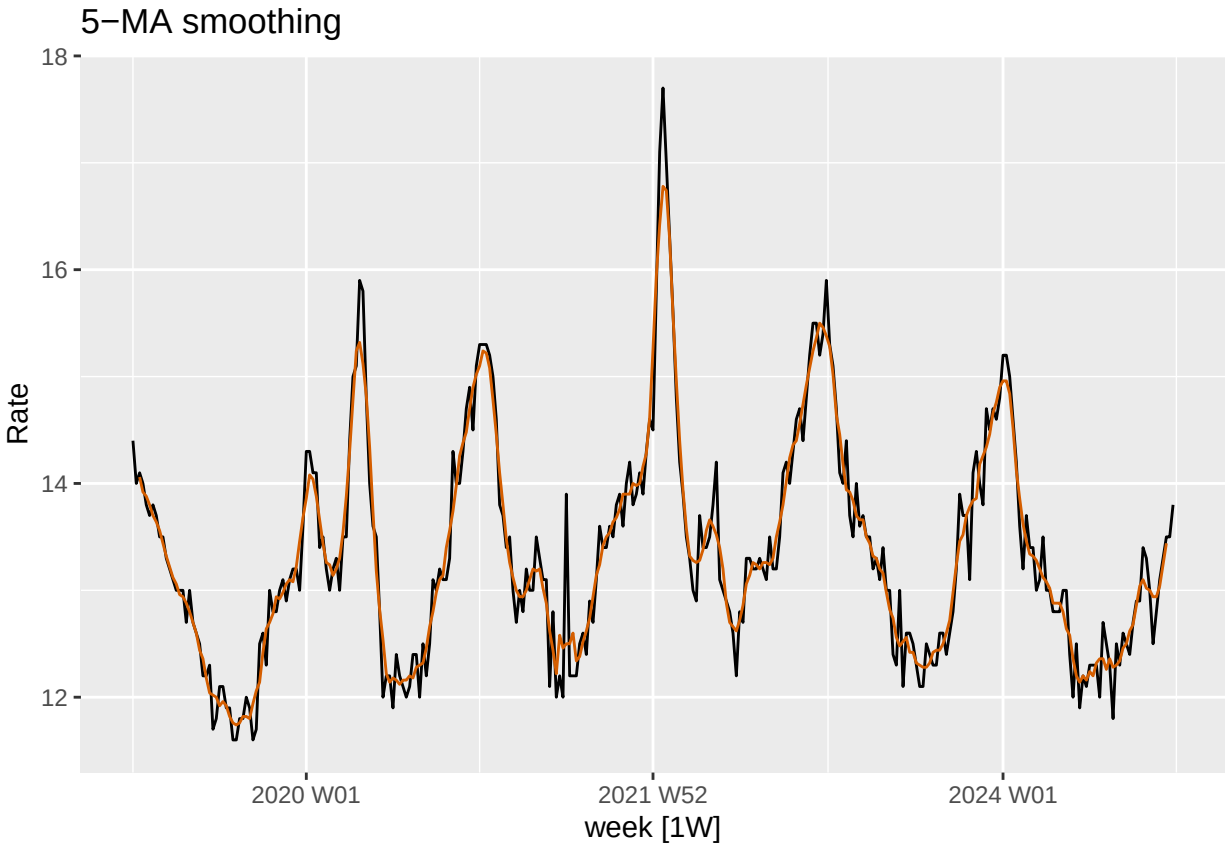
```r
# plot of 3-MA
mortality_ts_decom |>
  autoplot(Rate) +
  geom_line(aes(y = `3-MA`), colour = "#D55E00")+
  labs(title = "3-MA smoothing")
```

```
## Warning: Removed 2 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```




```r
# plot of 5-MA
mortality_ts_decom |>
  autoplot(Rate) +
  geom_line(aes(y = `5-MA`), colour = "#D55E00")+
  labs(title = "5-MA smoothing")
```

```
## Warning: Removed 4 rows containing missing values or values outside the scale range
## (‘geom_line()‘).
```



Forecasting Models

Simple forecasting methods

The `model()` function trains models to data. We are going to forecast using Mean, Naive and Seasonal Naive method.

```
mortality_fit <-  mortality_ts %>%
  model(
    Seasonal_naive = SNAIVE(Rate),
    Naive = NAIVE(Rate),
    Mean = MEAN(Rate)
  )
```

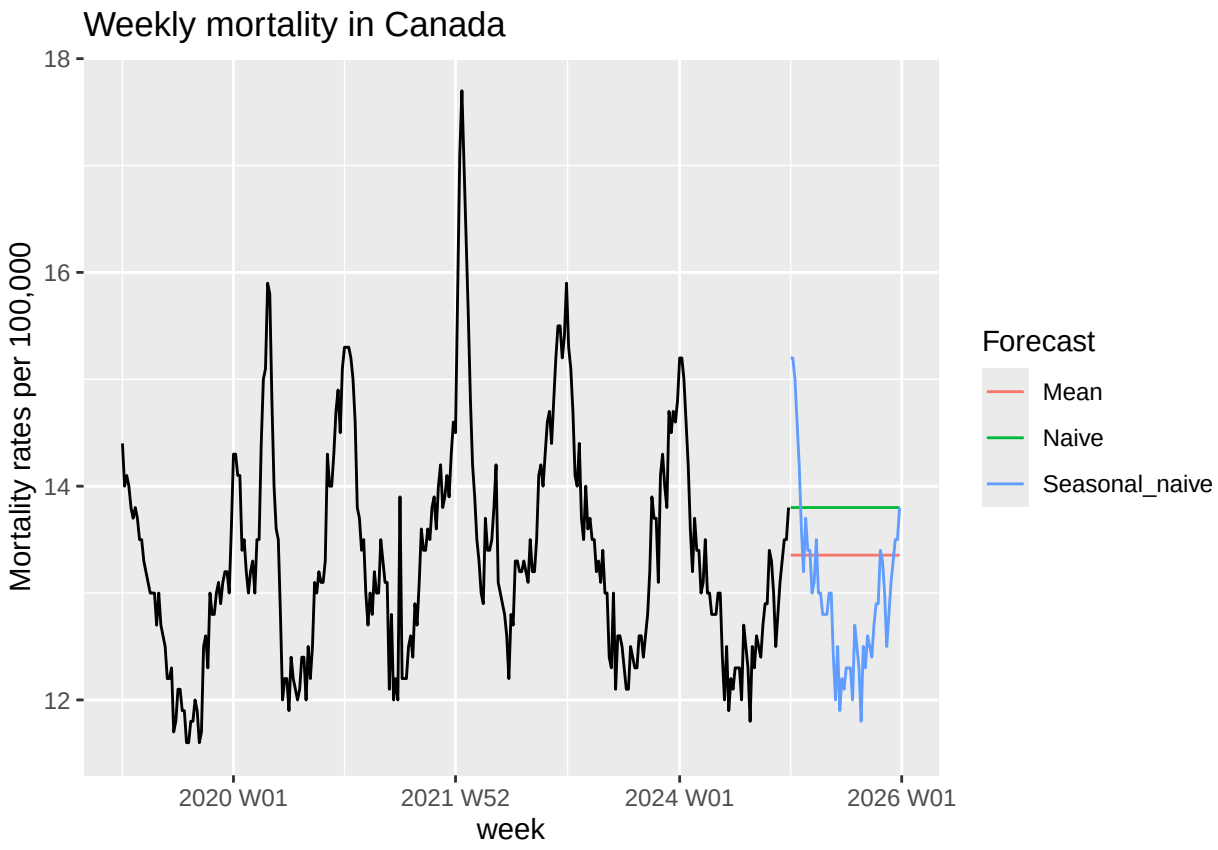
We can now produce forecasts using the fitted models.

```
mortality_fc <- mortality_fit %>%
  forecast(h = "1 years")
print(mortality_fc, n = 4)
```

```
## # A fable: 156 x 4 [1W]
## # Key:     .model [3]
##   .model             week
##   <chr>             <week>
```

```
## 1 Seasonal_naive 2025 W01
## 2 Seasonal_naive 2025 W02
## 3 Seasonal_naive 2025 W03
## 4 Seasonal_naive 2025 W04
## # i 152 more rows
## # i 2 more variables: Rate <dist>, .mean <dbl>
```

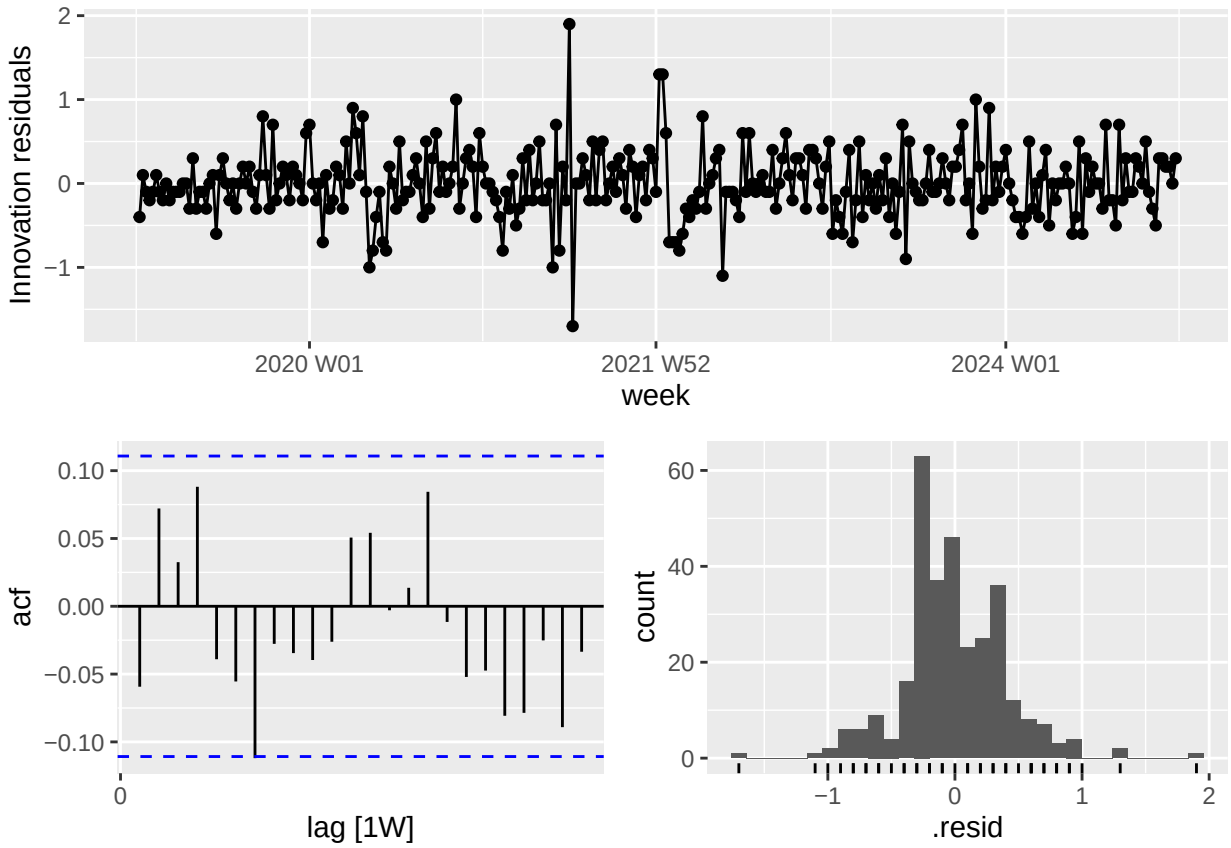
```
mortality_fc %>%
  autoplot(mortality_ts, level = NULL) +
  labs(title = "Weekly mortality in Canada",
       y = "Mortality rates per 100,000") +
  guides(colour = guide_legend(title = "Forecast"))
```



Residual diagnostics

It is very important to do the residual diagnostic after fitting any model to check whether the residual assumptions have been satisfied or not.

```
fit.naive <- mortality_ts %>% model(NAIVE(Rate))
gg_tsresiduals(fit.naive)
```



Here, based on the plots, we can say the residual assumptions (uncorrelated, mean zero, constant variance) have been satisfied for Naive model.

Tests for autocorrelation

Moreover, we can do Box-Pierce or Ljung-Box test to see whether the residuals are significantly different from a zero set.

H_0: the series is white noise vs H_1: the series is not white noise.

```
augment(fit.naive) %>%
  features(.resid, ljung_box, lag=10, dof=0)
```

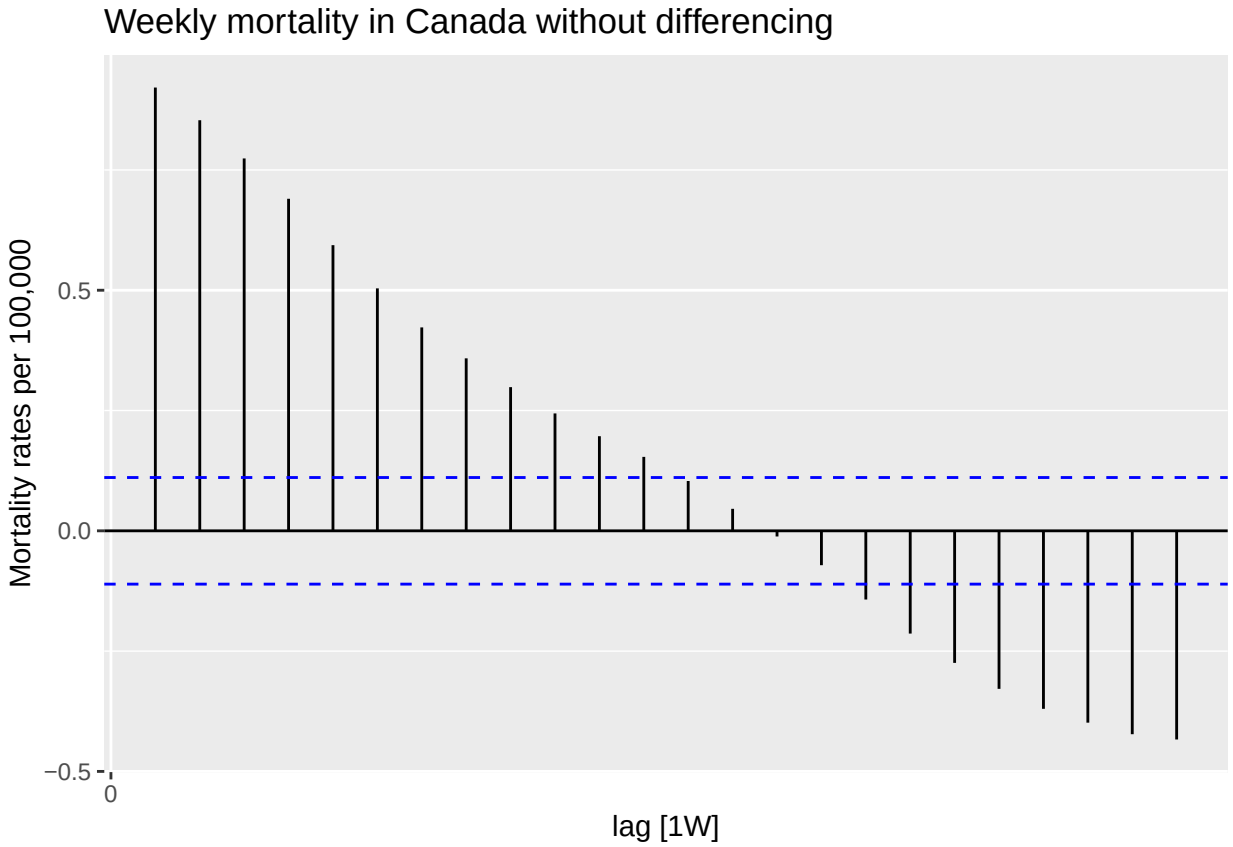
```
## # A tibble: 1 x 3
##   .model      lb_stat lb_pvalue
##   <chr>         <dbl>     <dbl>
## 1 NAIVE(Rate)    12.2     0.272
```

The results are not significant (i.e., the p-values are relatively large). Thus, we can conclude that the residuals are not distinguishable from a white noise series.

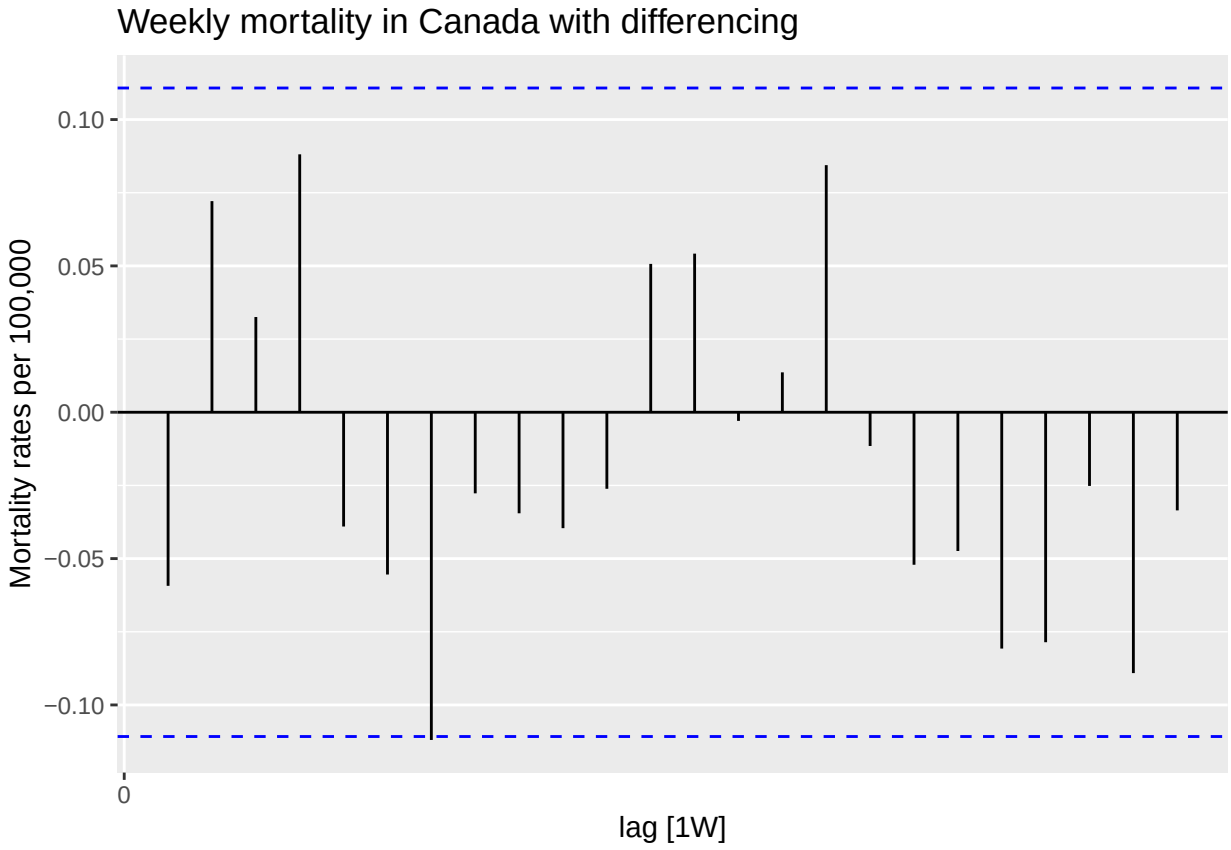
ARIMA / SARIMA models

Before fitting the ARIMA model, it is important to check the stationary of the data. If the data is non-stationary, then we need to make it stationary using the idea of differencing. Differencing helps to stabilize the mean (one way to make a non-stationary time series stationary).

```
mortality_ts %>% ACF(Rate) %>% autoplot() +
    labs(title = "Weekly mortality in Canada without differencing",
       y = "Mortality rates per 100,000")
```




```
mortality_ts %>% ACF(difference(Rate)) %>% autoplot() +
    labs(title = "Weekly mortality in Canada with differencing",
       y = "Mortality rates per 100,000")
```



First order differencing seems make the mortality data stationary. But still there are some pattern left over.

Now let's fit the ARIMA/SARIMA model. The *ARIMA* frunction form *fpp3* package does everything together (checking for stationarity of the data, doing any differencing if needed and then fitting the model using ARIMA or SARIMA based on data)

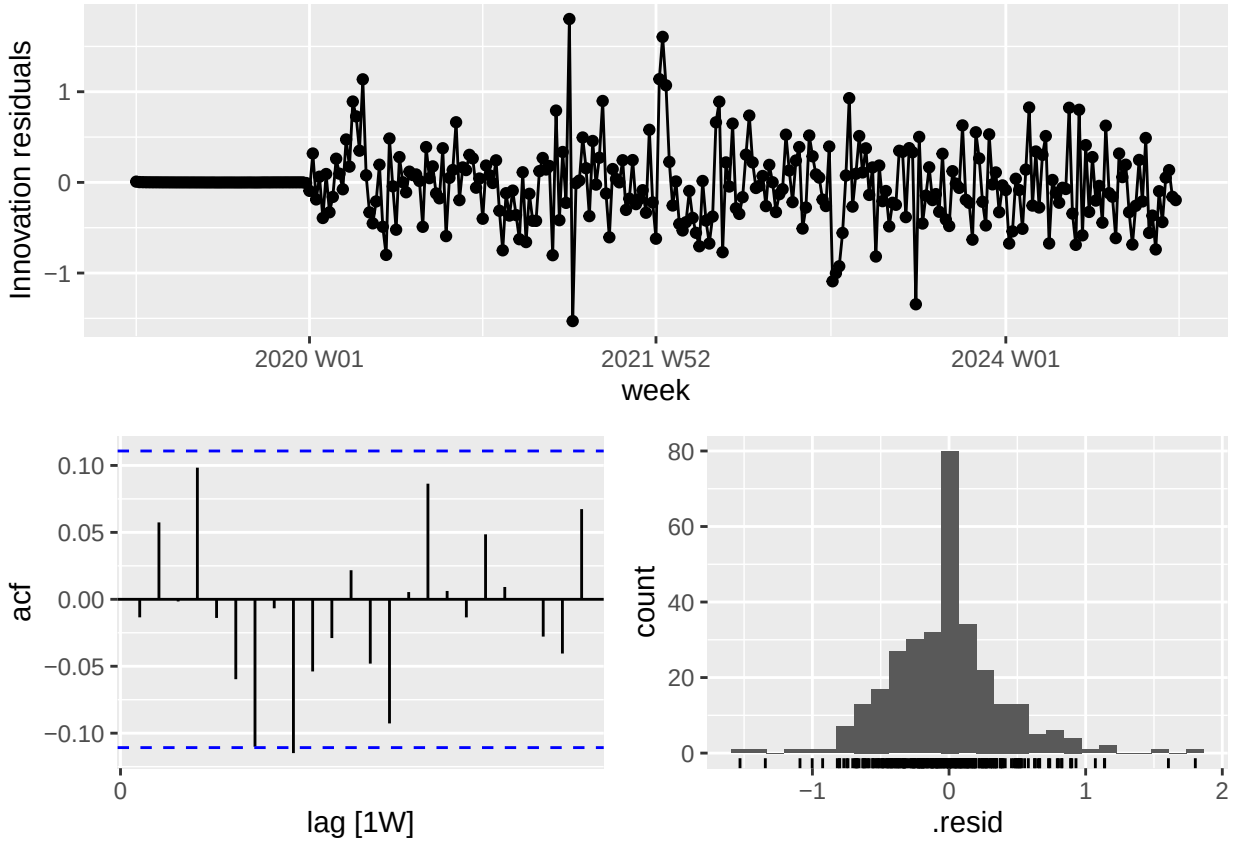
```
fit.arima <- mortality_ts |>
  model(ARIMA(Rate))
report(fit.arima)
```

```
## Series: Rate
## Model: ARIMA(1,1,2)(1,1,0)[52]
##
## Coefficients:
##          ar1      ma1     ma2     sar1
##       0.8391  -1.0746  0.1003  -0.5548
## s.e.  0.0486   0.0748  0.0690   0.0502
##
## sigma^2 estimated as 0.2067:  log likelihood=-171.98
## AIC=353.96   AICc=354.2   BIC=371.77
```

We have SARIMA(1,1,2)(1,1,0)[52] model for this data. We have $d = 1$ means there was first order differencing on non-seasonal part and $D = 1$ means first order differencing on seasonal part.

Let's check the residual.

```
gg_tsresiduals(fit.arima)
```



Here, based on the plots, we can say the residual assumptions (uncorrelated, mean zero, constant variance) have been satisfied for ARIMA model.

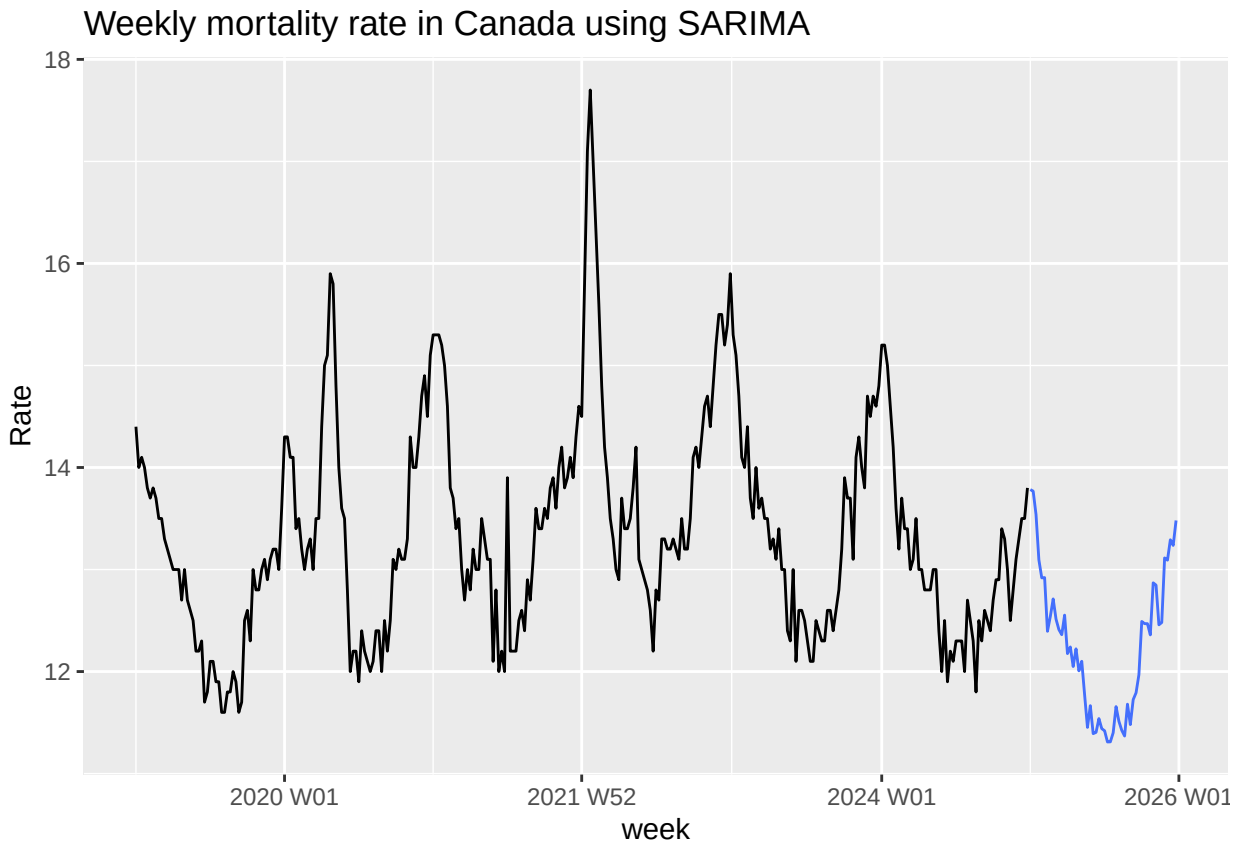
```
augment(fit.arima) %>%
  features(.innov, ljung_box, lag = 36, dof=4)
```

```
## # A tibble: 1 x 3
##   .model      lb_stat lb_pvalue
##   <chr>         <dbl>     <dbl>
## 1 ARIMA(Rate)    34.2     0.363
```

The results are not significant (i.e., the p-values are relatively large). Thus, we can conclude that the residuals are not distinguishable from a white noise series.

Let's forecast mortality rate for next one year (52 weeks).

```
fit.arima %>% forecast(h = 52) %>%  # h = "1 years"
  autoplot(mortality_ts, level=NULL) +
  labs(y = "Rate", title = "Weekly mortality rate in Canada using SARIMA")
```

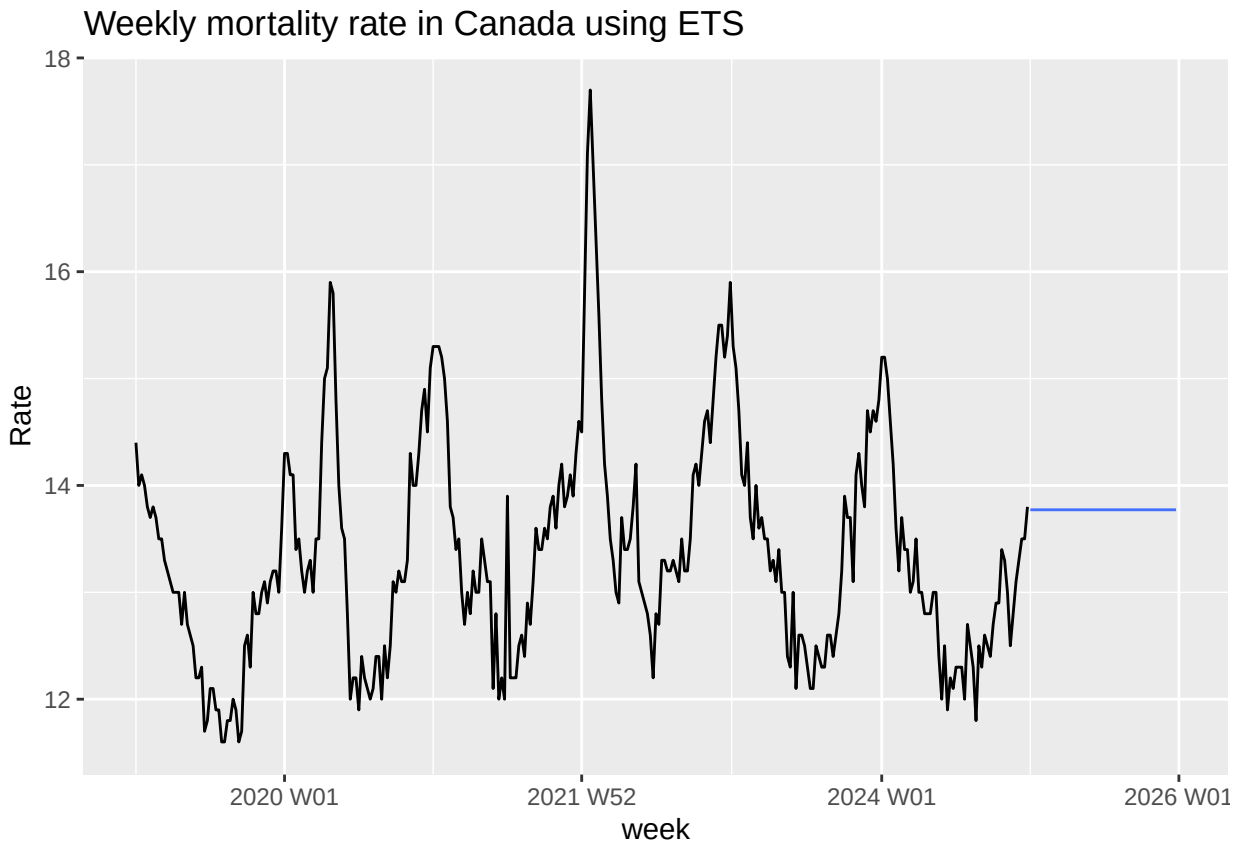


Simple Exponential Smoothing

```
fit.ets <- mortality_ts %>% model(ETS(Rate))
report(fit.ets)
```

```
## Series: Rate
## Model: ETS(M,N,N)
##   Smoothing parameters:
##     alpha = 0.9080476
##
##   Initial states:
##      l[0]
##  14.35072
##
##   sigma^2:  9e-04
##
##      AIC     AICc      BIC
## 1229.487 1229.564 1240.725
```

```
fit.ets %>%
  forecast(h = "1 years") %>%
  autoplot(mortality_ts, level=NULL)+
  labs(title="Weekly mortality rate in Canada using ETS", y="Rate")
```



Neural Netwrok Model

Now, we will use Neural network autoregression model to forecast mortality rate using `NNETAR()` function that fits an NNAR$(p, P, k)_m$ model. If p and P are not specified, they are automatically selected.

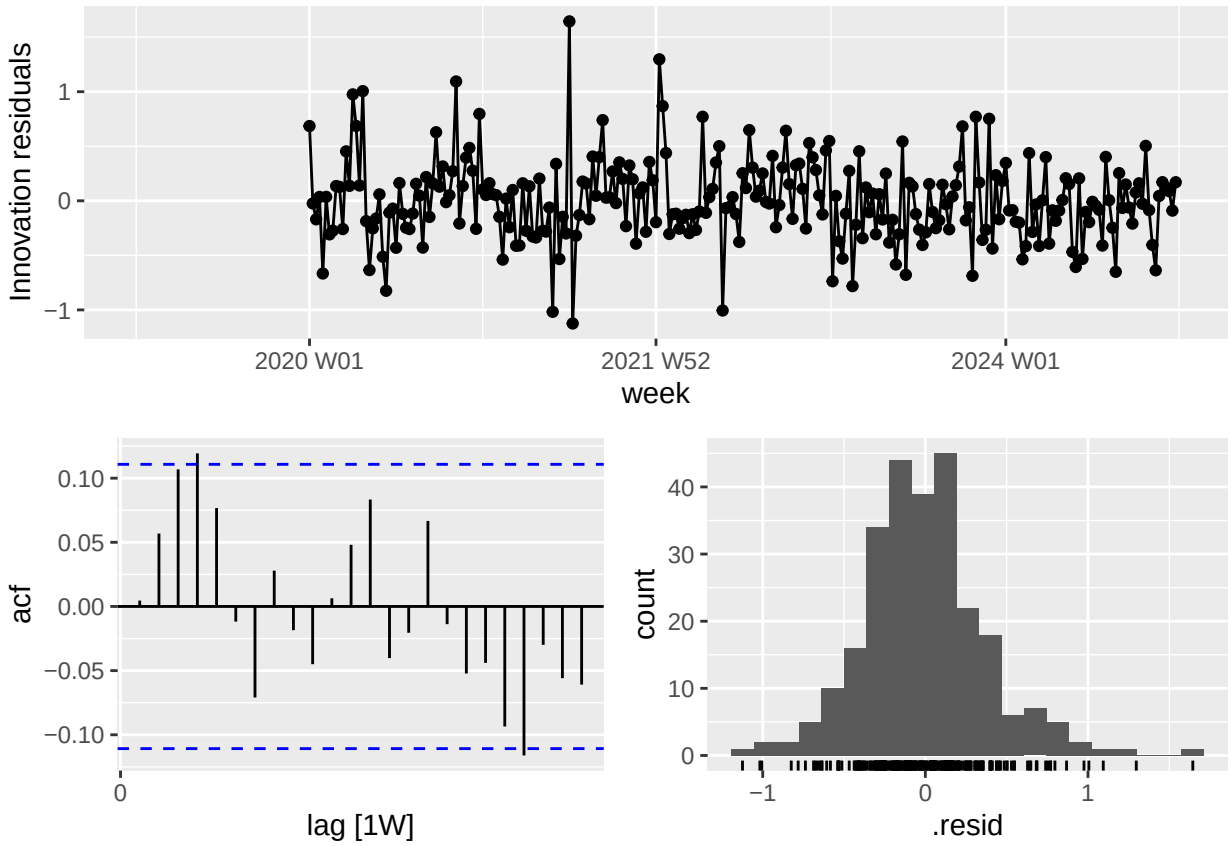
```
fit.nn <- mortality_ts %>% model(NNETAR(Rate))
fit.nn
```

```
## # A mable: 1 x 1
##      'NNETAR(Rate)'
##              <model>
## 1 <NNAR(2,1,2)[52]>
```

The result provides a NNAR(2,1,2)[52] model for mortality rate. Here, the last 2 observations are used as predictors, and there are 2 neurons in the hidden layer.

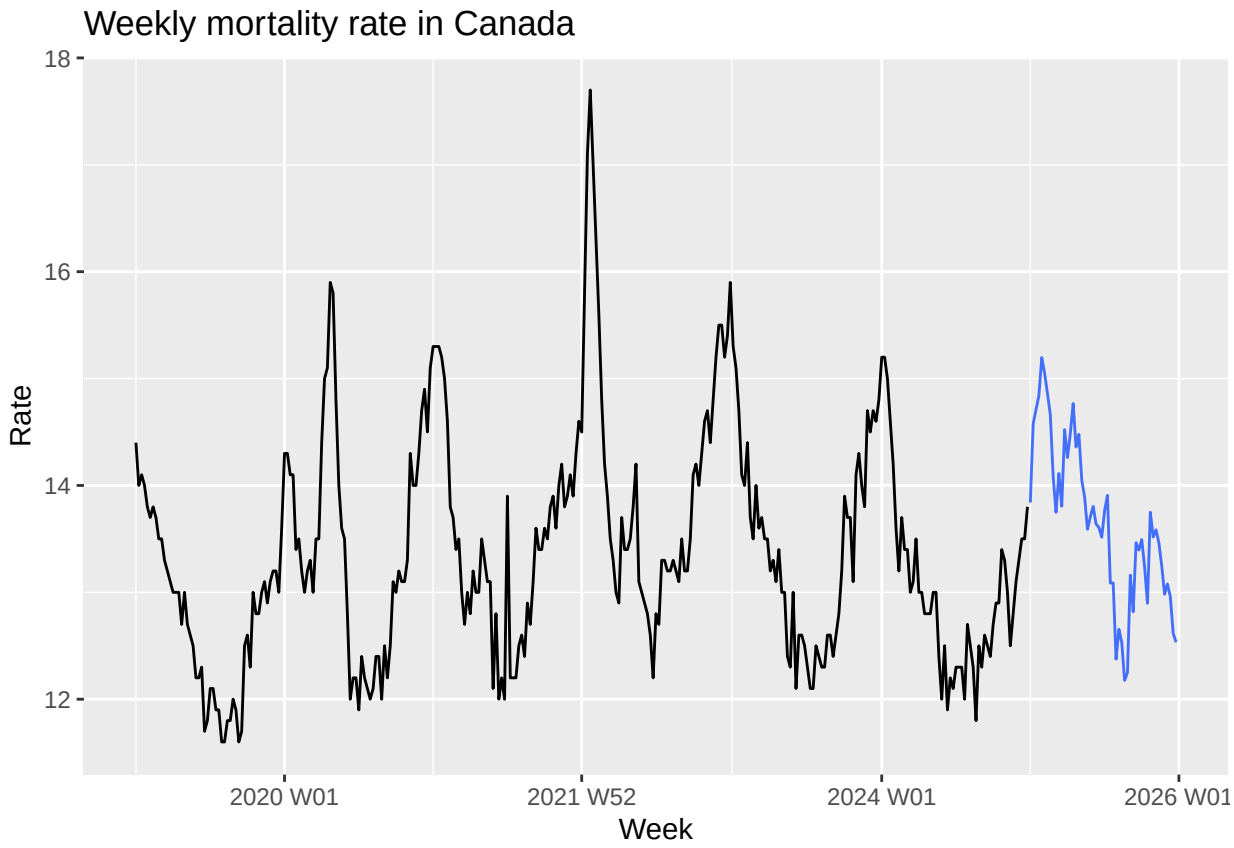
Let's check the residual.

```
gg_tsresiduals(fit.nn)
```



Let's forecast mortality rate for next one year (52 weeks).

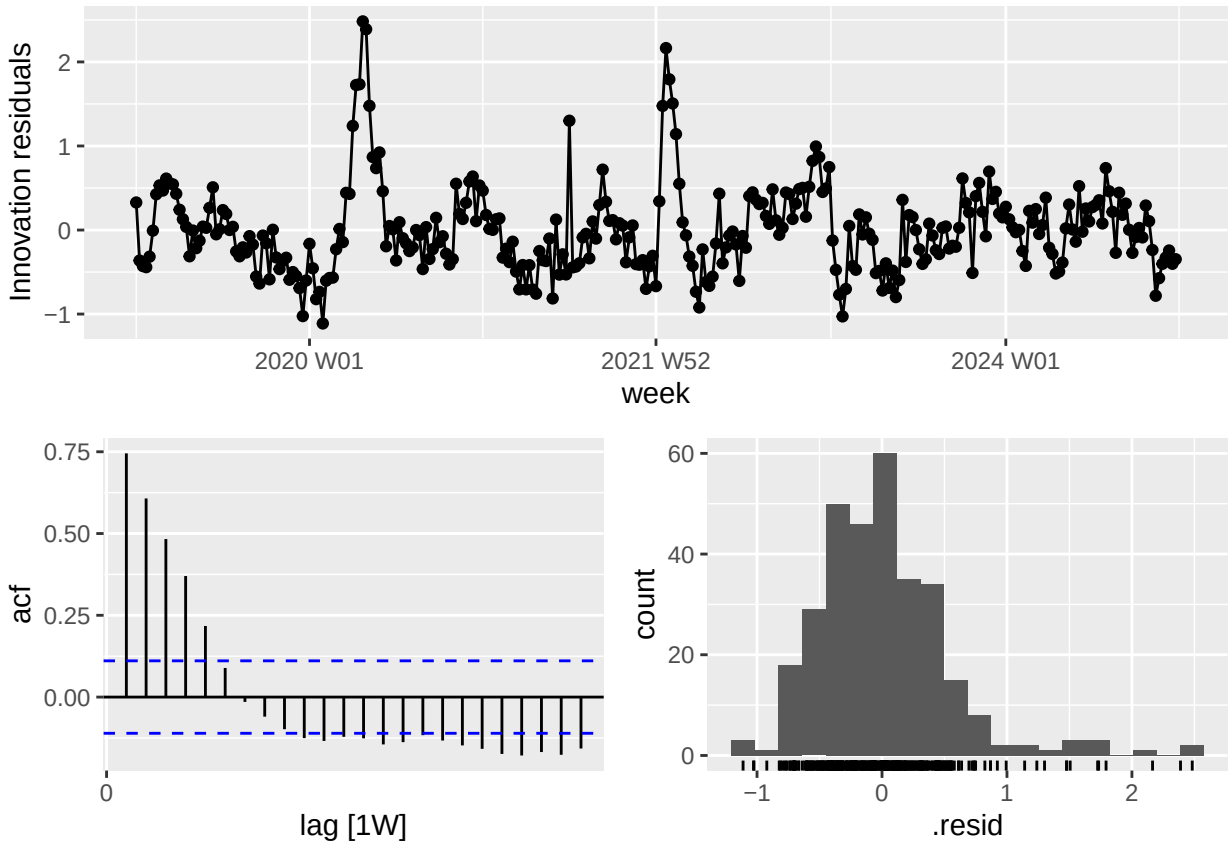
```
fit.nn %>% forecast(h="1 years", times=1) %>%
  autoplot(mortality_ts, level=NULL) +
  labs(x = "Week", y = "Rate", title = "Weekly mortality rate in Canada")
```



Prophet Model

Prophet is an open-source forecasting tool developed by Facebook for time series forecasting. Prophet model is available via the `fable.prophet` package.

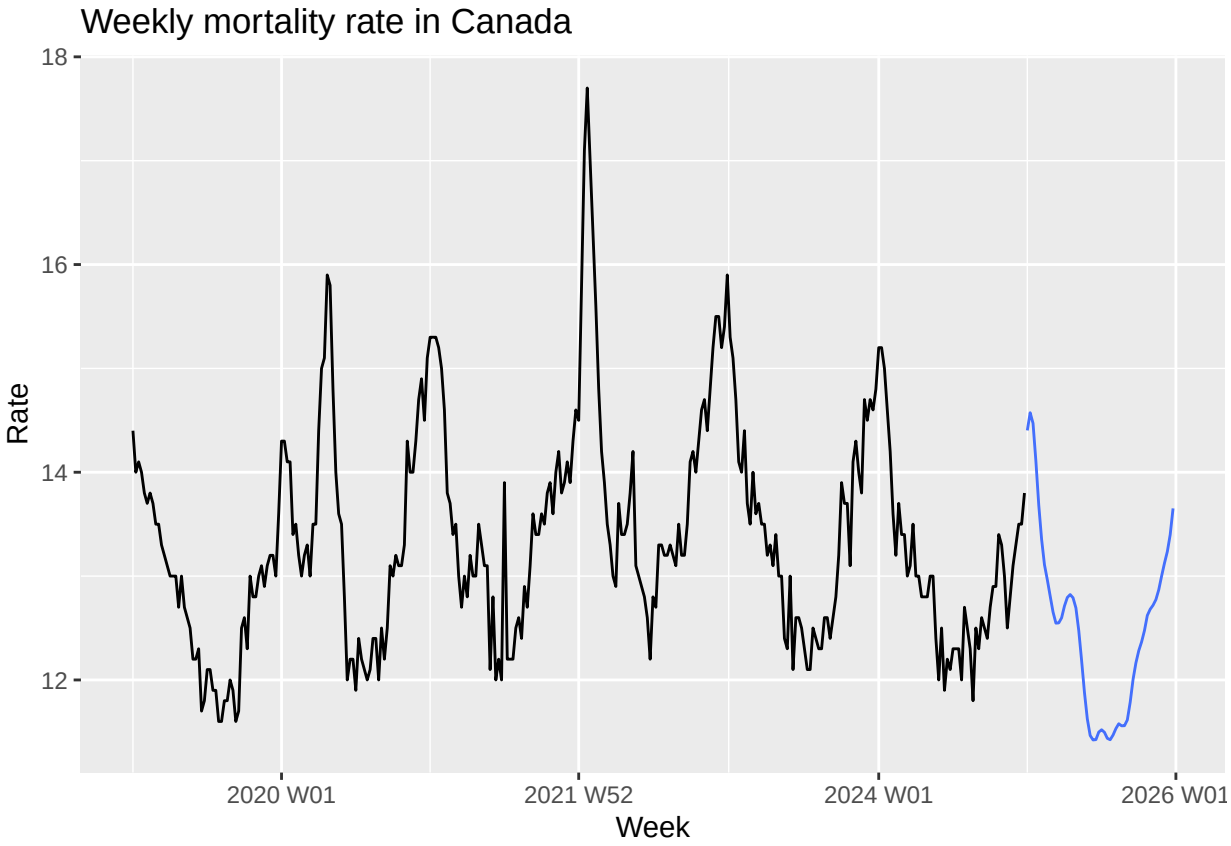
```
fit.ph <- mortality_ts %>%
  model(prophet(Rate ~ season(period = 52, order = 10)))
fit.ph |> gg_tsresiduals() # check residual
```




```
#components(fit.ph) %>% autoplot()
```

Let's forecast mortality rate for next one year (52 weeks).

```
fit.ph %>% forecast(h = "1 years") %>%
  autoplot(mortality_ts, level=NULL) +
  labs(x = "Week", y = "Rate", title = "Weekly mortality rate in Canada")
```



Forecast accuracy: Comparing all models

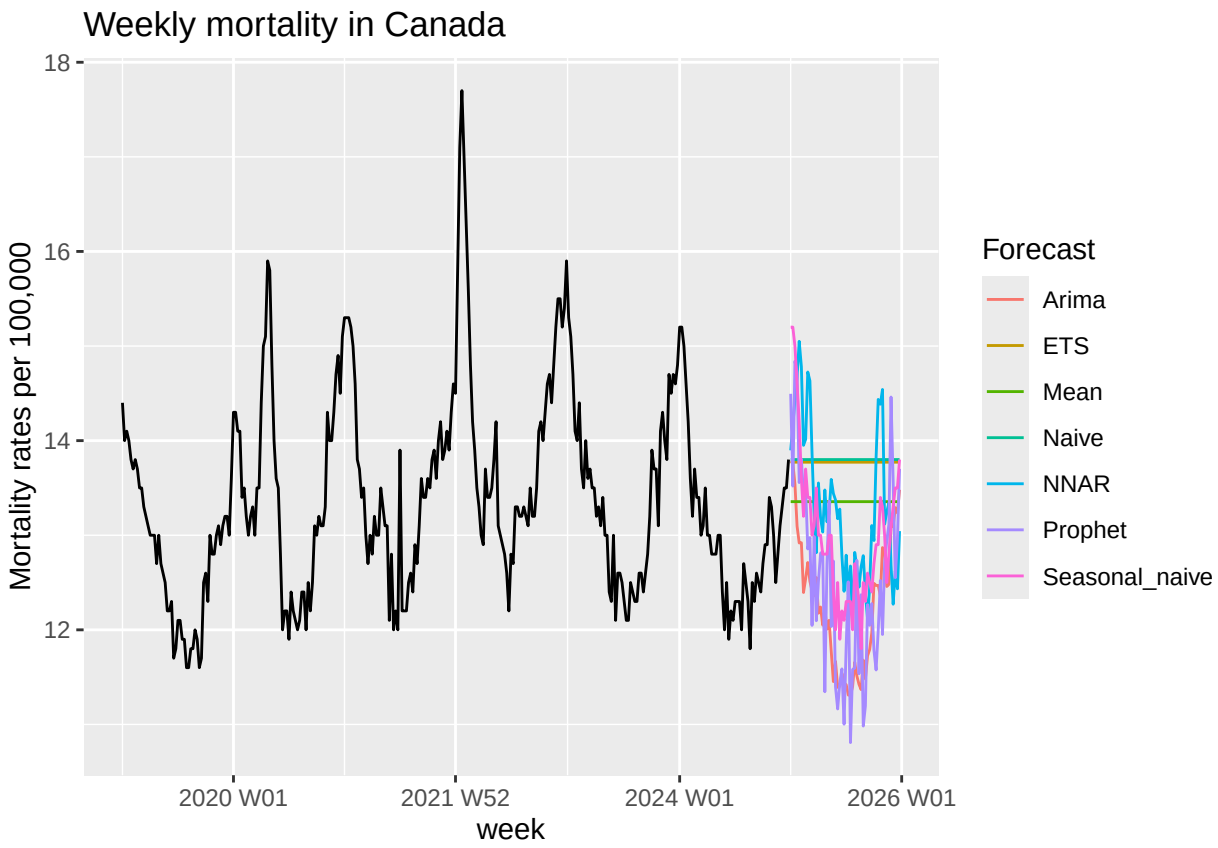
We can fit all model together in one function and compare them based on some accuracy measures. Then choose the best one and forecast based on the best model.

```
mortality_fit <-  mortality_ts %>%
  model(
    Seasonal_naive = SNAIVE(Rate),
    Naive = NAIVE(Rate),
    Mean = MEAN(Rate),
    Arima = ARIMA(Rate),
    ETS = ETS(Rate),
    NNAR = NNETAR(Rate),
    #Prophet = prophet(Rate)
    Prophet = prophet(Rate ~ season(period = 52, order = 10))
  )
```

We can now produce forecasts for next 1 years (52 weeks) using the fitted models.

```
mortality_fc <- mortality_fit %>%
  forecast(h = "1 years", times=1)

mortality_fc %>%
  autoplot(mortality_ts, level = NULL) +
  labs(title = "Weekly mortality in Canada",
```

```
      y = "Mortality rates per 100,000") +
  guides(colour = guide_legend(title = "Forecast"))
```



Let's compare all models.

```
accuracy(mortality_fit)
```

```
## # A tibble: 7 x 10
##   .model         .type         ME  RMSE   MAE     MPE  MAPE  MASE RMSSE     ACF1
##   <chr>          <chr>      <dbl> <dbl> <dbl>   <dbl> <dbl> <dbl> <dbl>    <dbl>
## 1 Seasonal_naive Train~  4.56e- 2 0.947 0.725  0.127   5.27 1     1      0.811
## 2 Naive          Train~ -1.92e- 3 0.405 0.296 -0.0586  2.20 0.408 0.428 -0.0593
## 3 Mean           Train~ -9.65e-17 1.03  0.803 -0.569   5.94 1.11  1.09   0.921
## 4 Arima          Train~ -2.58e- 2 0.411 0.287 -0.242   2.13 0.395 0.434 -0.0135
## 5 ETS            Train~ -2.04e- 3 0.405 0.297 -0.0628  2.21 0.410 0.427  0.0390
## 6 NNAR           Train~ -1.33e- 6 0.376 0.283 -0.0792  2.09 0.390 0.397  0.00156
## 7 Prophet        Train~ -4.29e- 5 0.525 0.381 -0.141   2.78 0.525 0.554  0.745
```

Based on the accuracy measures (say, RMSE, MAE, MAPE, MASE), it is found that Neural net and ARIMA models provide better forecasting performance compare to other models.

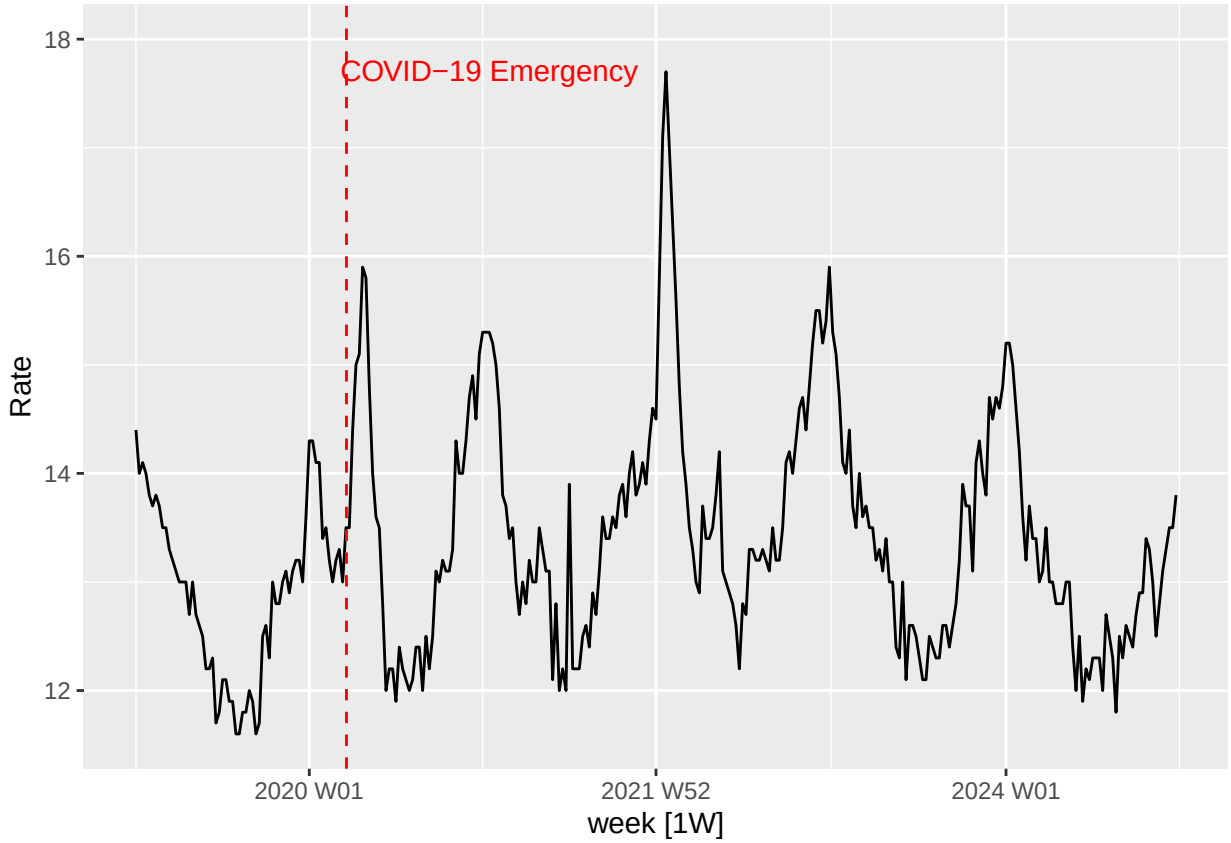
Time Series Regression

Now, we are going to do the time series regression models. We will forecast the weekly mortality rate based on some other predictor, say mean precipitation and temperature of week, trend of the data, any intervention

effect (e.g. Covid-19 effect) and so on. We need to assume that morality rate has a linear relationship with these predictors.

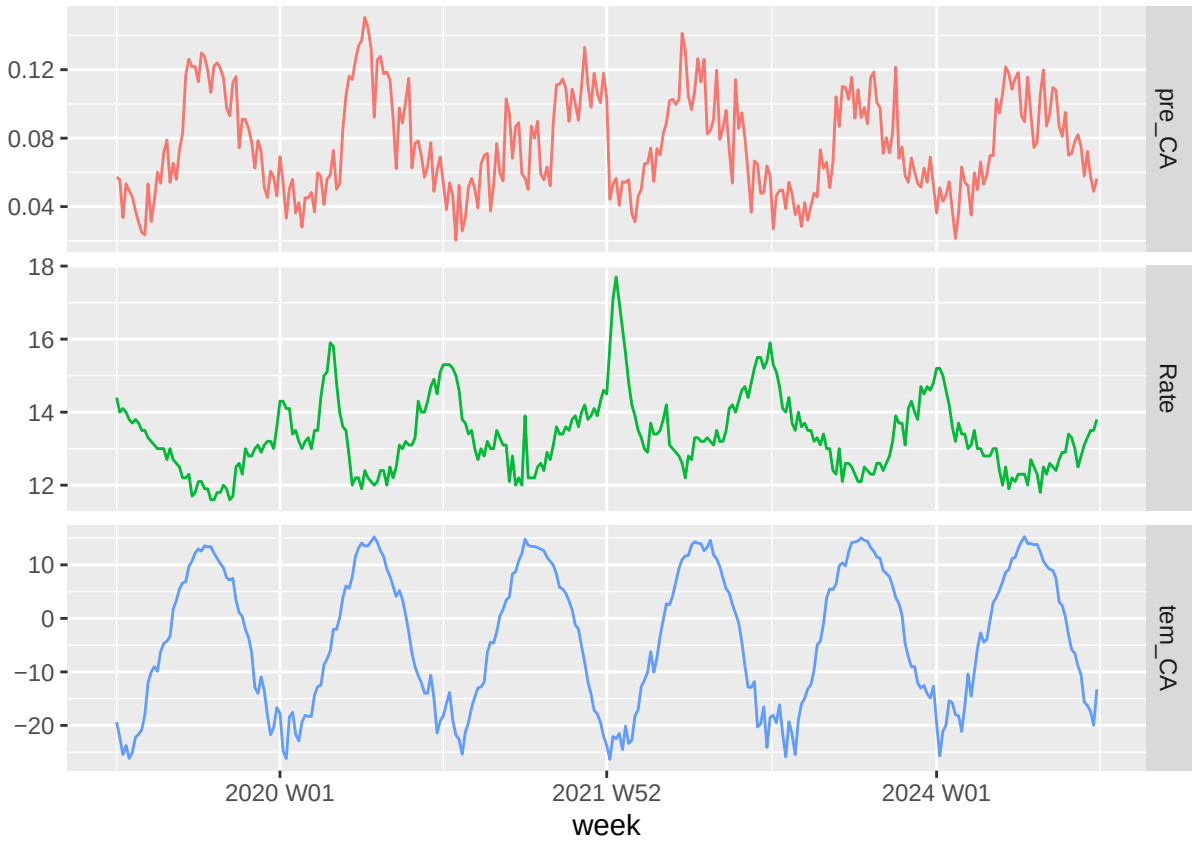
First let's look at the time plot again.

```
peak_time <- as.Date("2020-03-17")
mortality_ts %>% autoplot(Rate) +
  geom_vline(xintercept = peak_time, linetype = "dashed", color = "red")+
  annotate("text", x = peak_time + 300, y = 18, label = "COVID-19 Emergency",
           vjust = 2, color = "red", angle = 0)
```



Now, let's look at the relationship among variables.

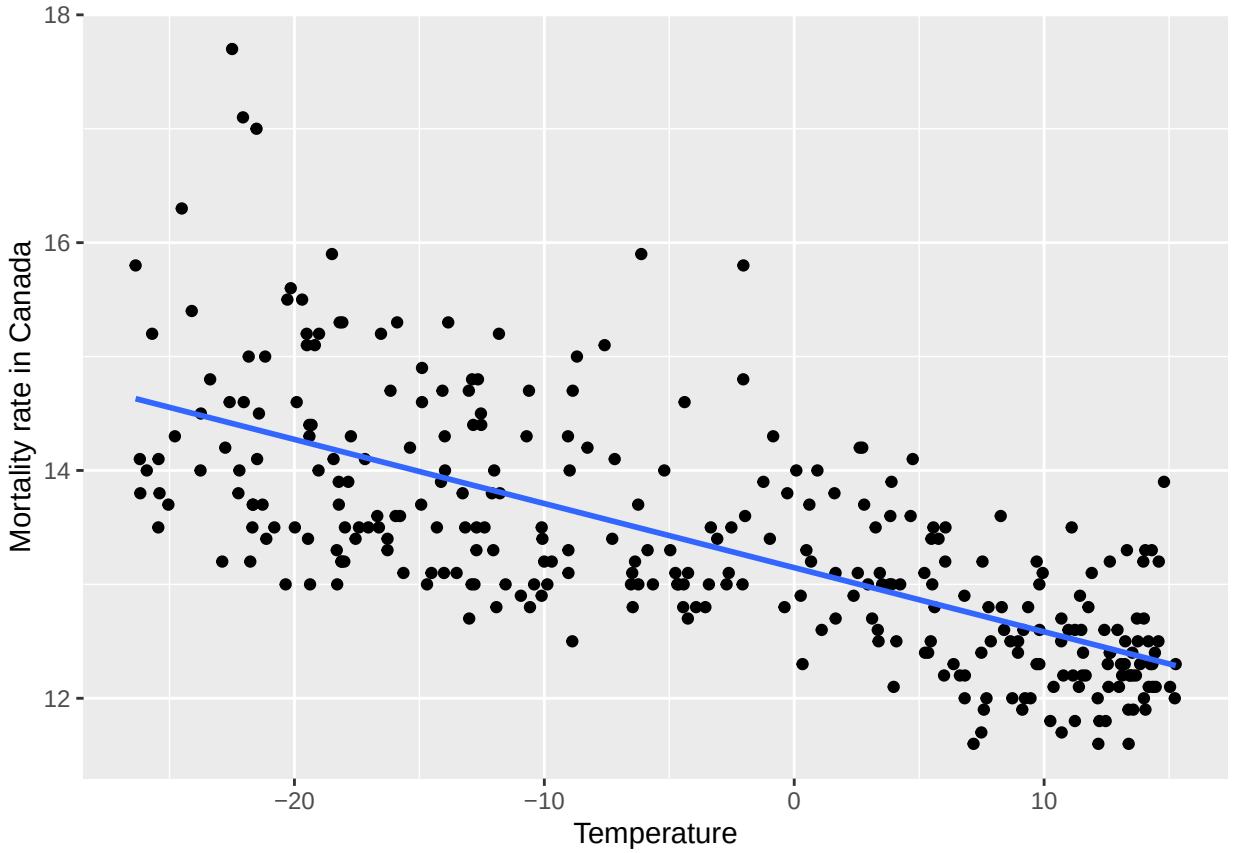
```
mortality_ts %>%
  select(Rate, tem_CA, pre_CA, week) %>% pivot_longer(-week) |>
  ggplot(aes(week, value, colour = name)) +
  geom_line() + facet_grid(name ~ ., scales = "free_y") +
  guides(colour = "none") + labs(y=" ")
```



Let's see the relation between weekly mortality rate and weekly mean temperature.

```
mortality_ts %>%
  ggplot(aes(y = Rate, x = tem_CA)) +
  labs(x = "Temperature", y = "Mortality rate in Canada") +
  geom_point() + geom_smooth(method = "lm", se = FALSE)
```

```
## `geom_smooth()` using formula = 'y ~ x'
```



Let's fit the time series regression model.

```
fit_MR <- mortality_ts %>%
  model(lm = TSLM(Rate ~ tem_CA + pre_CA))
report(fit_MR)
```

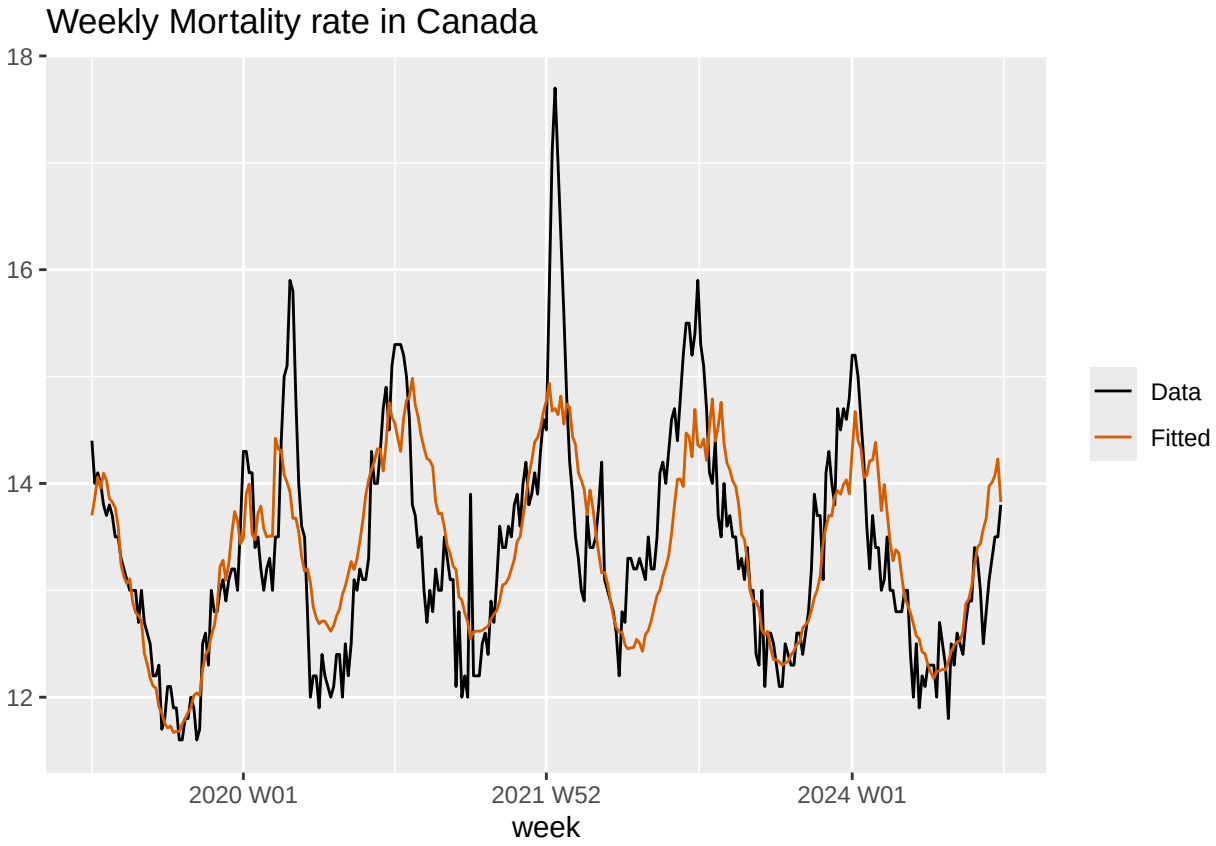
```
## Series: Rate
## Model: TSLM
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -1.2905 -0.5014 -0.1395  0.4034  3.2895
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept) 13.163523   0.178717  73.656   <2e-16 ***
## tem_CA      -0.055956   0.004751 -11.777   <2e-16 ***
## pre_CA      -0.207068   2.108964  -0.098    0.922
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.7382 on 310 degrees of freedom
## Multiple R-squared: 0.494,   Adjusted R-squared: 0.4907
## F-statistic: 151.3 on 2 and 310 DF, p-value: < 2.22e-16
```

Let's fit the model using other covariates/predictors, say Covid-19 intervention effect, trend variable (already built in *fpp3* package as *trend()*)

```r
# create dummy for Covid-19 intervention effect
mortality_ts <- mortality_ts |>
  mutate(covid19=ifelse(Date<as.Date("2020-03-17"),0,1))

fit_mortality <- mortality_ts |>
  model(TSLM(Rate ~ tem_CA + pre_CA + covid19 + trend()))
report(fit_mortality)
```

```
## Series: Rate
## Model: TSLM
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.53407 -0.42401 -0.06095  0.35059  2.99934
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.5620911  0.1796296  69.933  < 2e-16 ***
## tem_CA      -0.0593914  0.0042171 -14.083  < 2e-16 ***
## pre_CA      -0.1915244  1.8620008  -0.103 0.918141
## covid19      1.1539455  0.1277332   9.034  < 2e-16 ***
## trend()     -0.0021284  0.0005663  -3.759 0.000204 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.6488 on 308 degrees of freedom
## Multiple R-squared: 0.6117,  Adjusted R-squared: 0.6066
## F-statistic: 121.3 on 4 and 308 DF, p-value: < 2.22e-16
```

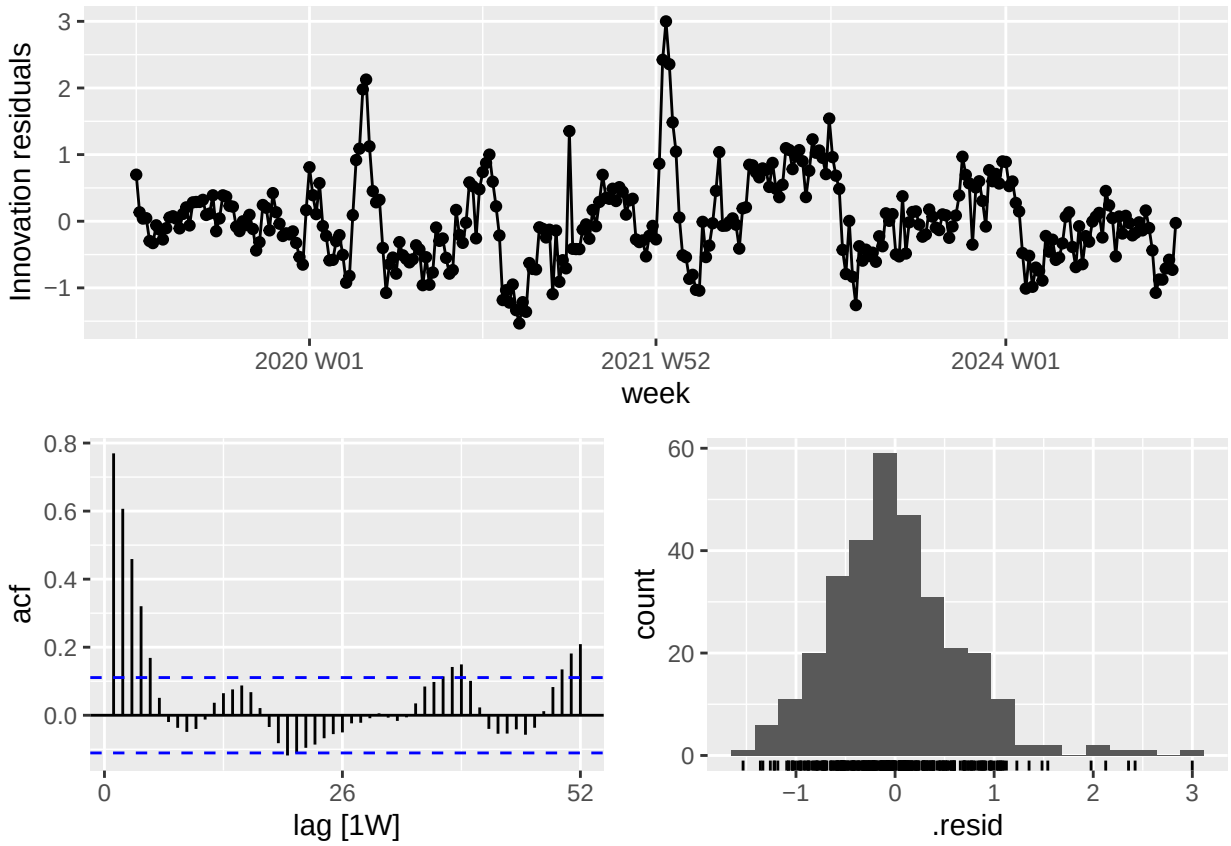
```r
# Check the fitted values
augment(fit_mortality) |>
  ggplot(aes(x = week)) +
  geom_line(aes(y = Rate, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Fitted")) +
  labs(y = NULL,
    title = "Weekly Mortality rate in Canada"
  ) +
  scale_colour_manual(values=c(Data="black",Fitted="#D55E00")) +
  guides(colour = guide_legend(title = NULL))
```



Residual diagnostics

It is very important to do residual diagnostic after fitting the regression model so that we can verify the assumptions (residual's are uncorrelated, zero mean, constant variance, uncorrelated with other predictor).

```
fit_mortality |> gg_tsresiduals(lag=52)
```



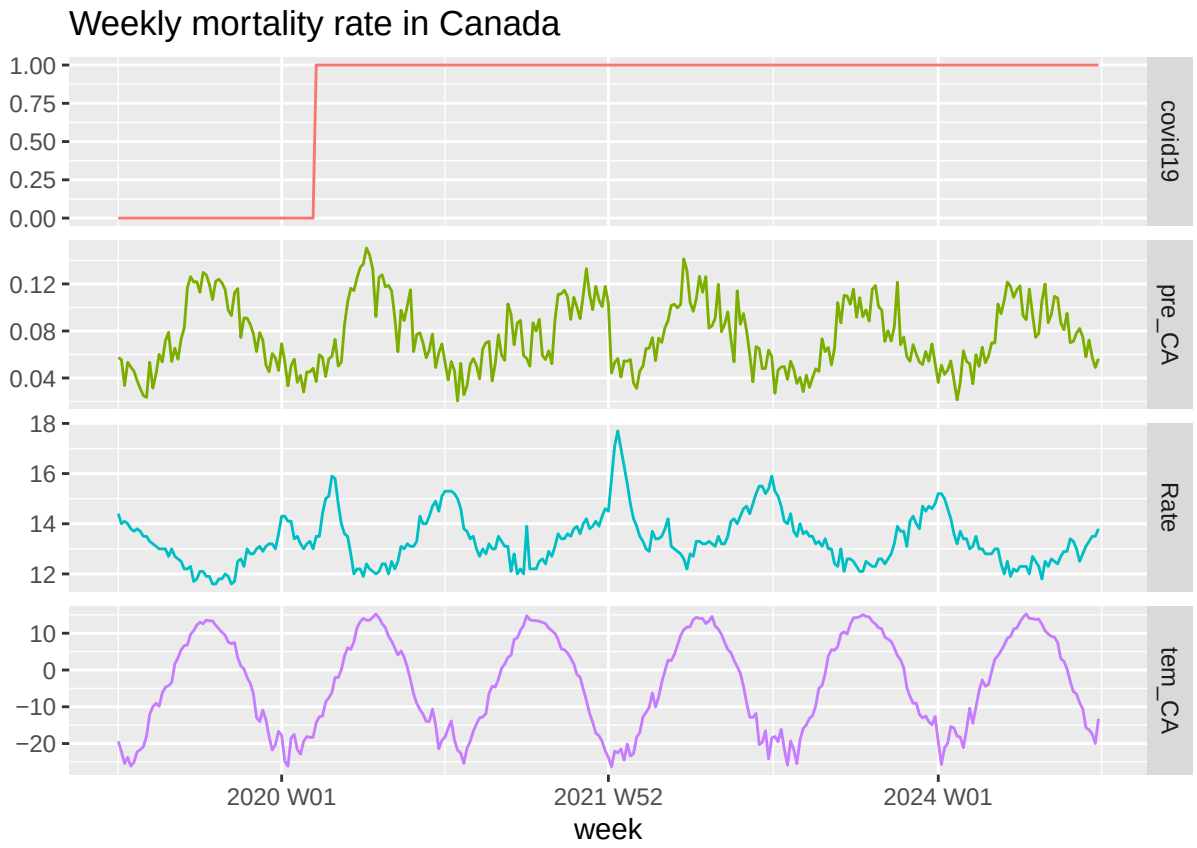
By looking at the auto correlation function of residuals, we can see still some correlation left in the residuals!

What to do then??

Solution: we can use Dynamic regression model.

Dynamic regression models

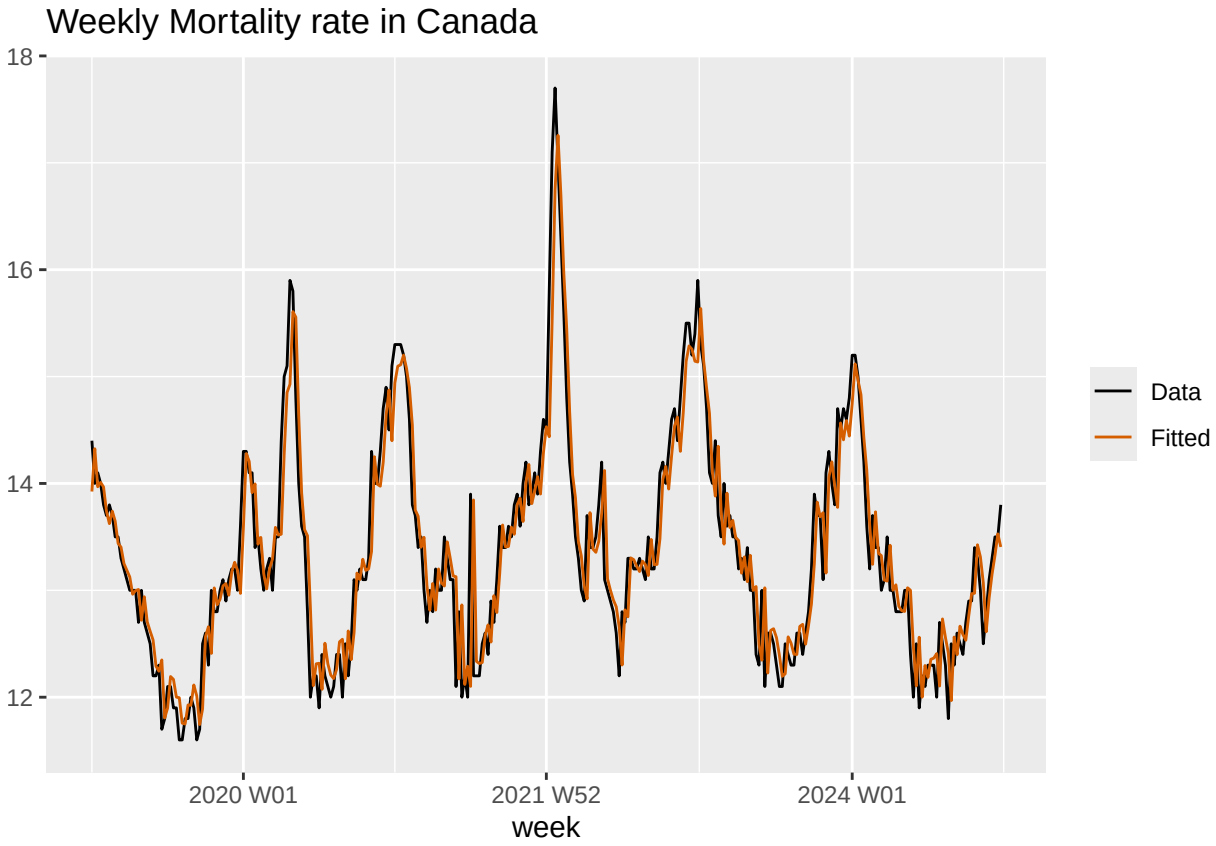
```
mortality_ts %>%
  select(-Date) %>%
  gather(key='variable', value='value') %>%
  ggplot(aes(y=value, x=week, group=variable, colour=variable)) +
  geom_line() + facet_grid(variable ~ ., scales='free_y') +
  labs(y="",title ="Weekly mortality rate in Canada") +
  guides(colour="none")
```



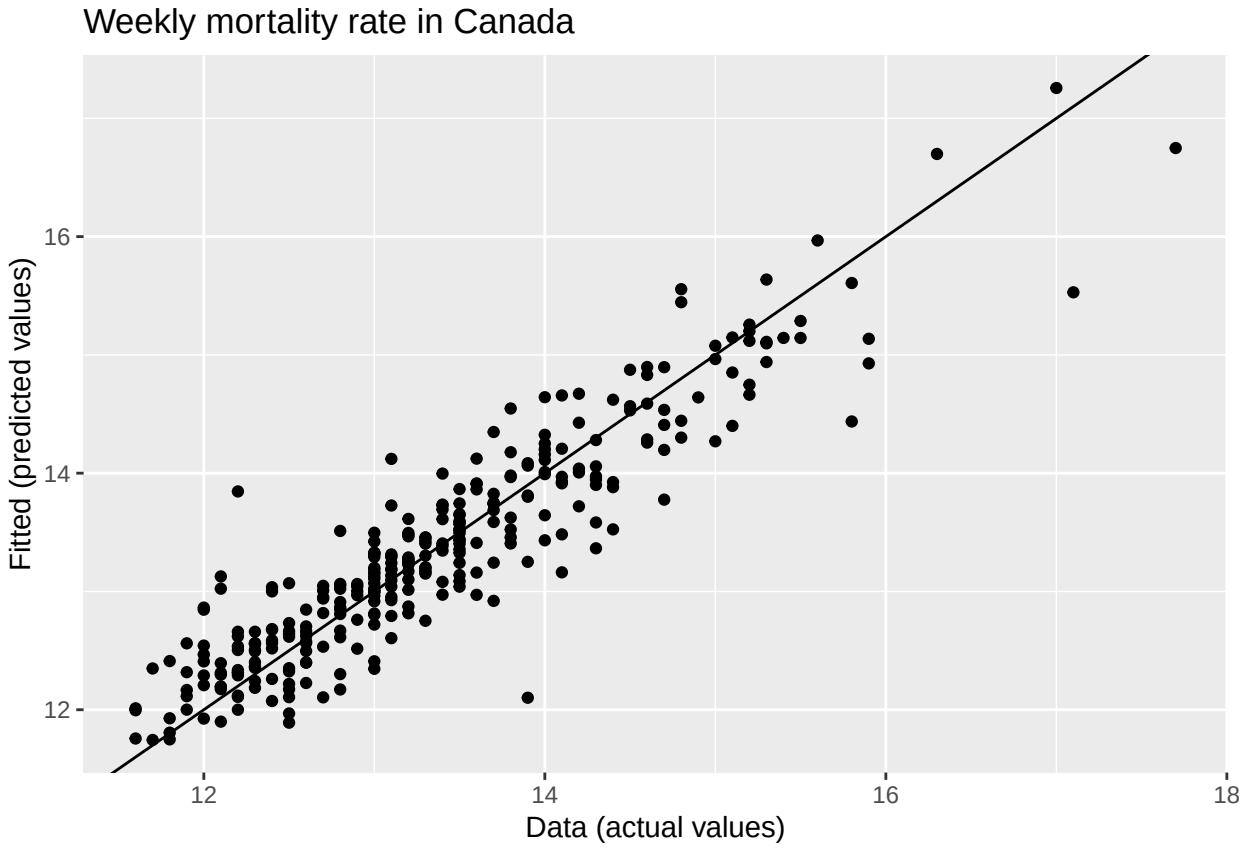

```
fit.dyn <- mortality_ts %>% model(ARIMA(Rate ~ tem_CA + pre_CA+ covid19+ trend()))
report(fit.dyn)
```

```
## Series: Rate
## Model: LM w/ ARIMA(1,0,0) errors
##
## Coefficients:
##          ar1   tem_CA  pre_CA  covid19  trend()  intercept
##       0.8927  -0.0144  0.4343   0.6282  -0.0015    13.0433
## s.e.  0.0352   0.0109  1.3558   0.3702   0.0024     0.4373
##
## sigma^2 estimated as 0.1591:  log likelihood=-154.22
## AIC=322.44   AICc=322.81   BIC=348.66
```

```
# Check the fitted values
augment(fit.dyn) |>
  ggplot(aes(x = week)) +
  geom_line(aes(y = Rate, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Fitted")) +
  labs(y = NULL,
    title = "Weekly Mortality rate in Canada"
  ) +
  scale_colour_manual(values=c(Data="black",Fitted="#D55E00")) +
  guides(colour = guide_legend(title = NULL))
```




```
augment(fit.dyn) |>
  ggplot(aes(x = Rate, y = .fitted)) +
  geom_point() +
  labs(
    y = "Fitted (predicted values)",
    x = "Data (actual values)",
    title = "Weekly mortality rate in Canada"
  ) +
  geom_abline(intercept = 0, slope = 1)
```

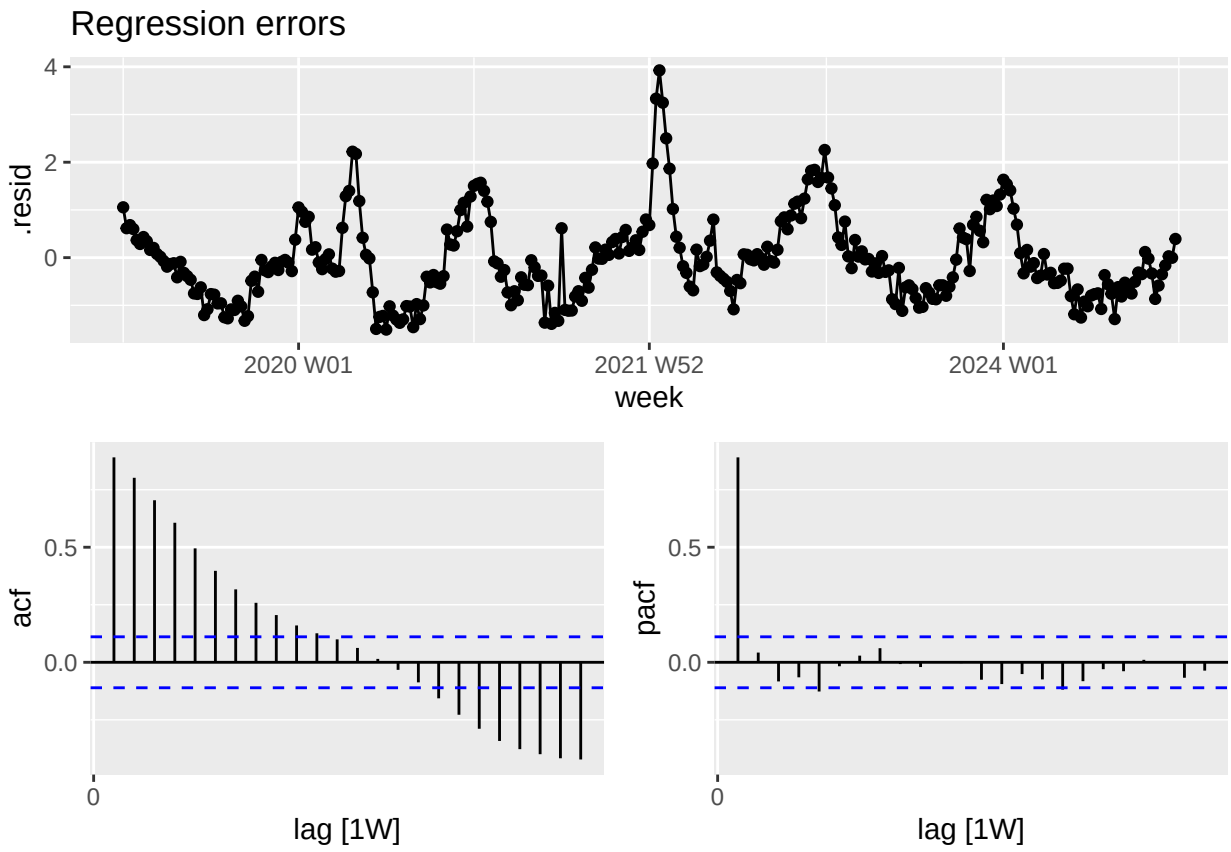


Residual diagnostics

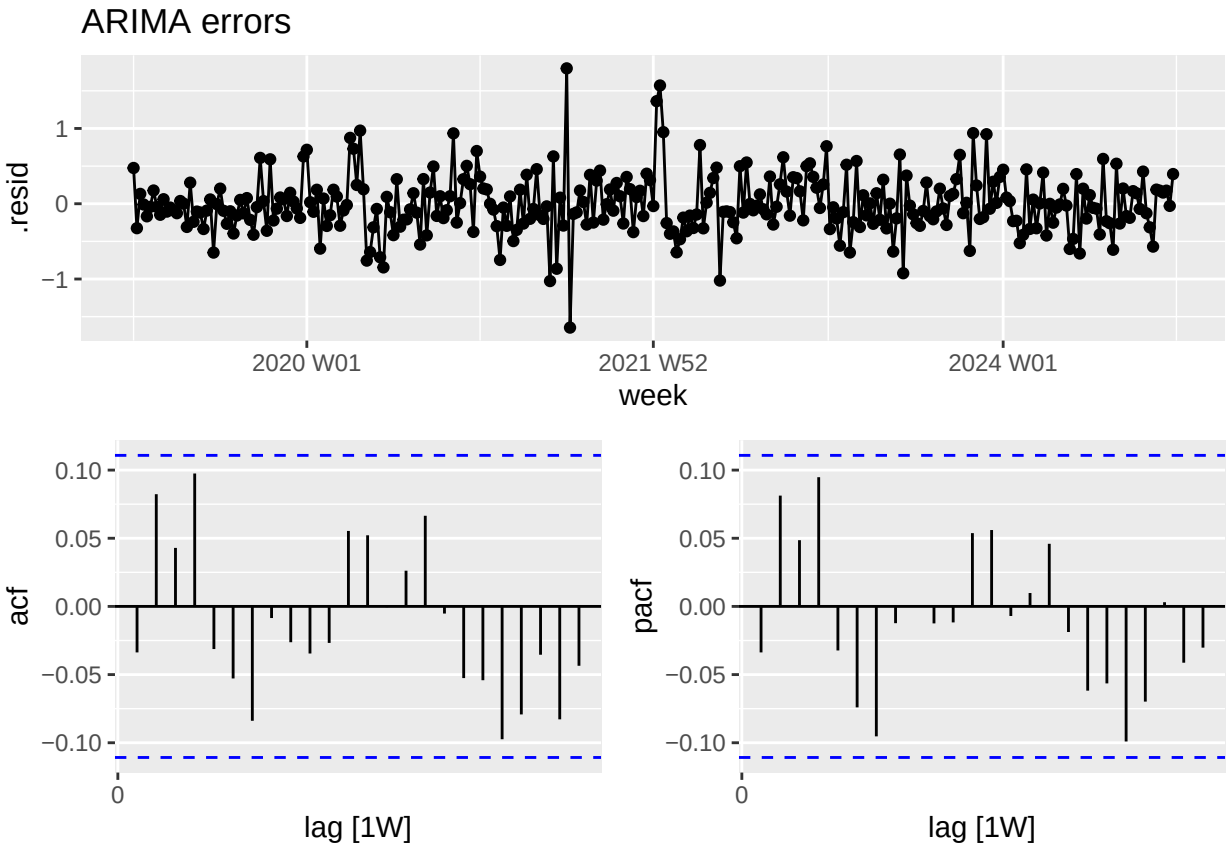
Recall that, in dynamic regression we have two errors: regression error (η_t, will show some correlation) and Arima error (ϵ_t, will show no significant correlation).

```
# regression error
residuals(fit.dyn, type='regression') %>%
  gg_tsdisplay(.resid, plot_type = 'partial') +
  labs(title = "Regression errors")
```

```
## Warning: `gg_tsdisplay()` was deprecated in feasts 0.4.2.
## i Please use `ggtime::gg_tsdisplay()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```




```
# Arima error
residuals(fit.dyn, type='innovation') %>%
  gg_tsdisplay(.resid, plot_type = 'partial') +
  labs(title = "ARIMA errors")
```



Residuals seems white noise, means no significant correlation left!

```
augment(fit.dyn) |>
  features(.innov, ljung_box,lag = 52)
```

```
## # A tibble: 1 x 3
##   .model                                                lb_stat lb_pvalue
##   <chr>                                                   <dbl>     <dbl>
## 1 ARIMA(Rate ~ tem_CA + pre_CA + covid19 + trend())    61.5     0.173
```

The results are not significant (i.e., the p-values are relatively large). Thus, we can conclude that the residuals are not distinguishable from a white noise series.

Comment: Overall, it shows that the dynamic regression perform better than the time series linear regression.

Interactive graphics for Time Series data:

We are trying to developed interactive shiny graphics to address these issues for several of the most common time series data analyses. The interactive shiny graphics is used to generate an interactive visualization environment that contains several distinct graphics, many of which are updated in response to user input. These visualizations reduce the burden of exploratory analyses and can serve as a useful tool for the communication of results to non-statisticians.

Link (still under working)

<https://syedrizvi05.shinyapps.io/TimeSeries/>

Acknowledgement:

This document was prepared with the help of my research student, *Syed Jafar Rizvi*, MSc Student, Dept. of Community Health and Epidemiology, University of Saskatchewan. Fell free to knock Rizbi at *jafar.rizvi@usask.ca* if you have any question.