Forecasting Twitch Stream Time - Time Series Analysis and Practice

Yuya Ogawa

2024-07-05

Introduction

In the first section, we are interested in running Seasonal Autoregressive Integrated Moving Average model (SARIMA), Exponential Smoothing model (ETS), and Feed Forward Neural Net model (NNET). Then we will compare the forecasting accuracy of above models. In the second section, I will introduce two advanced methods. First one concerns with combining the Dynamic Regression models and SARIMA model to conduct a very simple contingency analysis. Second model concerns with the decomposing the series using STL decomposition and run the SARIMA on the de-seasoned component of the series. The seasonal portion is modeled using seasonal naive model. I assume some background knowledge in time series analysis on the part of readers, and my intention is to show how to run those analysis in R.

Data and Model Estimation: SARIMA, ETS, and Neural Net

First, download the library.

Data

```
library(dplyr)
library(lubridate)
library(fpp3)
library(urca)
```

Then, download the data for Twitch. The Twitch data was obtained from https://sullygnome.com/ using API and the full code to obtain the data is in my Github

The twitch data contains language streamed, games, watch time, etc. Here we will simply focus on the aggregate.

```
# Make sure to set the working directory
temp = readr::read_csv('twitch.csv')
filtered_temp <- temp[temp$language == "English", ]
```

We will sum up all of the stream time and watch time for each game to obtain the aggregate.

```
# Group by year and month, then summarize
twitch <- temp %>%
  group_by(year, month) %>%
  summarize(
            watch_time_min = sum(watch_time_min),
            stream_time_min = sum(stream_time_min),
            .groups = 'drop'
            )
```

Now, we will convert our data into tsibble for time series analysis.

```
# Define month levels in the correct order
month_levels <- tolower(month.name)
# Convert month column to a factor with the correct levels
twitch$month <- factor(twitch$month, levels = month_levels)
# Create a date column
twitch <- twitch %>%
  mutate(date = make_date(year, match(month, month_levels), 1))
# Tell R that date column is a time series date index by converting it to a tsibble
twitch <- twitch %>%
              mutate(date = yearmonth(ymd(date))) %>%
              as_tsibble(index=date)
```

Now that we have our Twitch data cleaned, we will add two more types of data that we might think is helpful in predicting the Twitch stream time: Unemployment rate and producer price index for telecommunication services such as cable, internet user services, etc. Data can be obtained from FRED

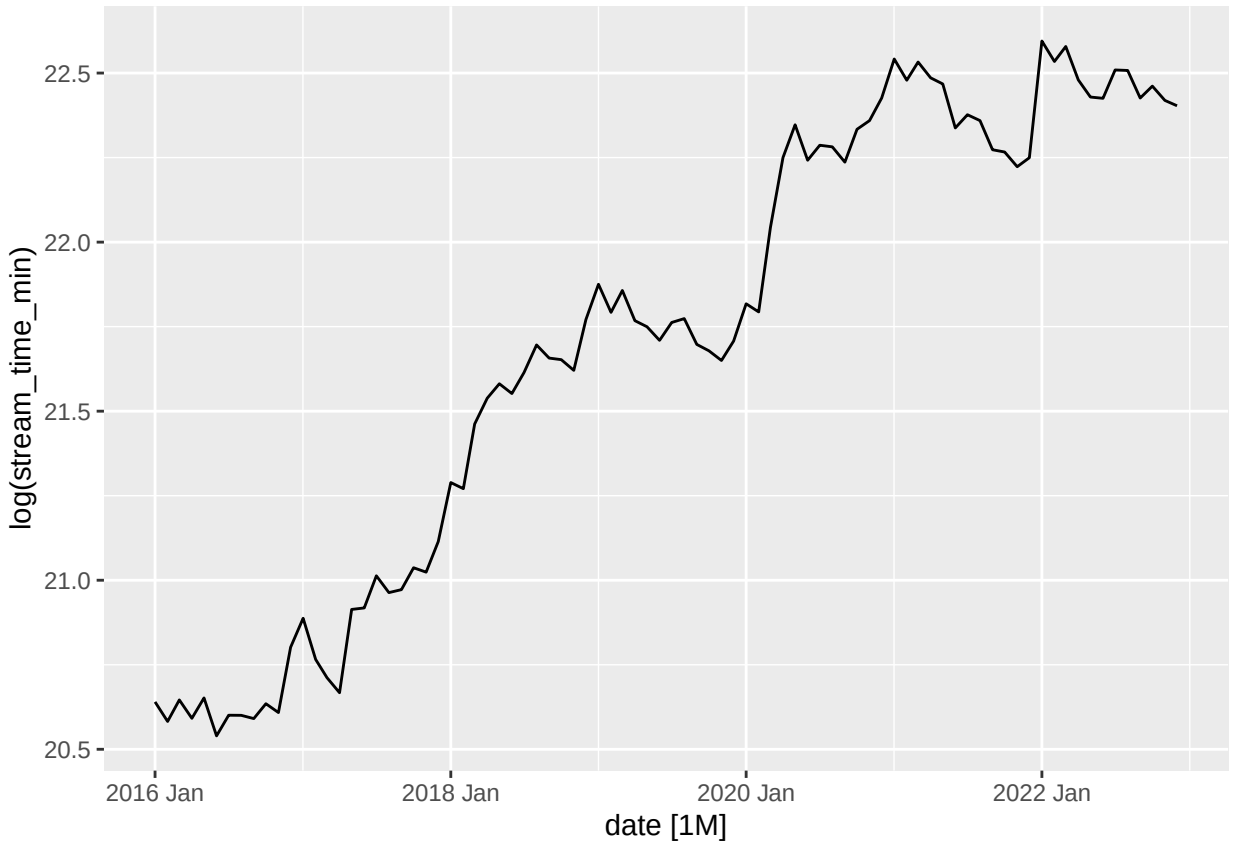
```
# Download the file.
temp2 = readr::read_csv('fredgraph.csv')

temp2 = temp2 %>%
              mutate(date = yearmonth(ymd(DATE))) %>%
              as_tsibble(index=date) %>%
              filter_index("2016 Jan" ~ "2023 Dec")

twitch$PPIintnet = temp2$WPS3721
twitch$unemp = temp2$UNRATE
twitch$ffrate = temp2$FEDFUNDS
twitch$mltrvl = temp2$TRFVOLUSM227NFWA
```

Now for the sake of calculating the out-sample accuracy. Create a Hold-out period. We will take the last 12 months of our data.

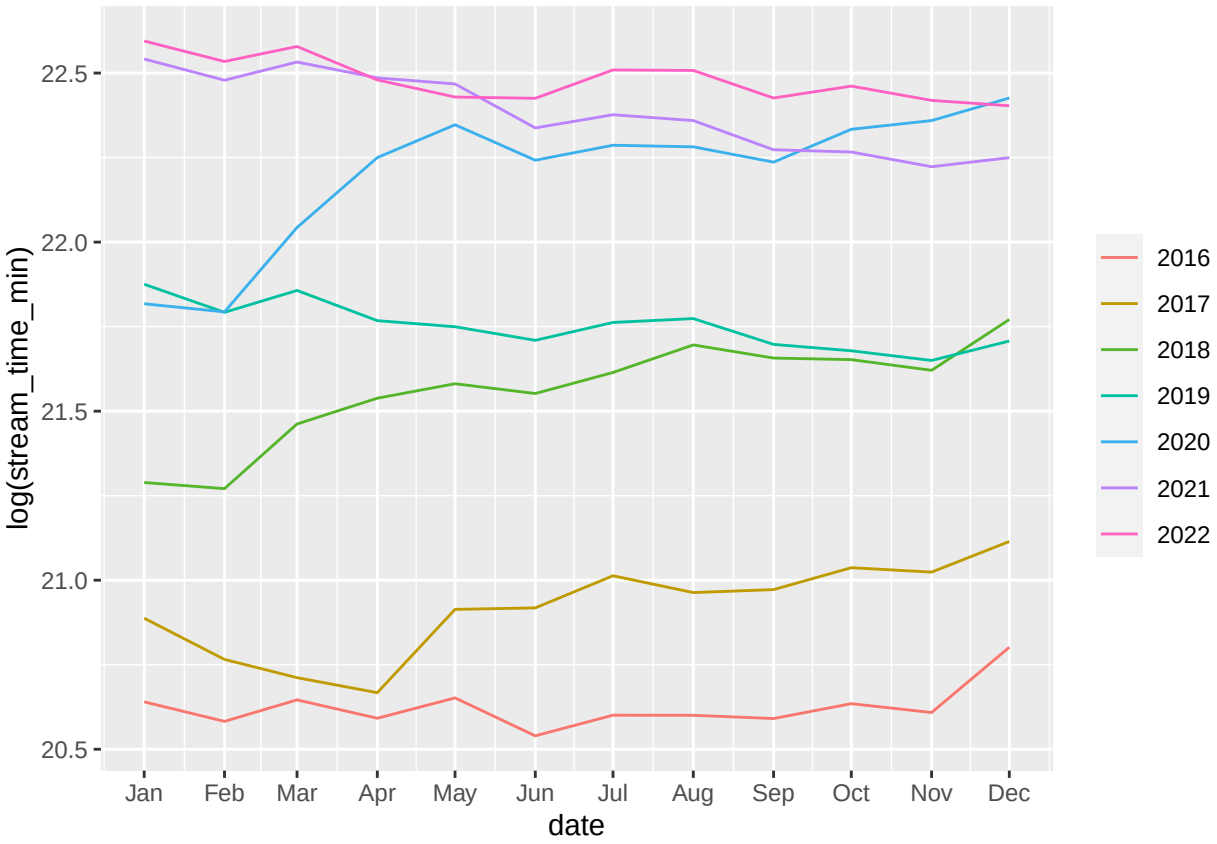
```
final_data = twitch

final_data <- final_data %>%
  mutate(unemp = as.numeric(unemp),
         PPIintnet = as.numeric(PPIintnet),
         ffrate = as.numeric(ffrate),
         mltrvl = as.numeric(mltrvl))

# create a hold out period for model training and out-sample forecasting comparison.
finalHOLD = final_data %>% filter_index('2016 Jan' ~ '2022 Dec')
```

Visually, our logged value of our stream time data looks like this.

```
finalHOLD %>% autoplot(log(stream_time_min))
```




```
finalHOLD %>% gg_season(log(stream_time_min))
```



Model Estimation

Continuing from the last section, we will use the above data to forecast Twitch stream time using SARIMA, ETS, and Neural Net.

SARIMA model

As we have seen in the above graph, we suspect that data is not stationary and perhaps requires a differencing to obtain stationarity. To formally test this, we will use the KPSS to test for stationarity.

```
# Test stationarity without differencing.
finalHOLD %>%features(log(stream_time_min),unitroot_kpss)
```

```
## # A tibble: 1 x 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1      2.07        0.01
```

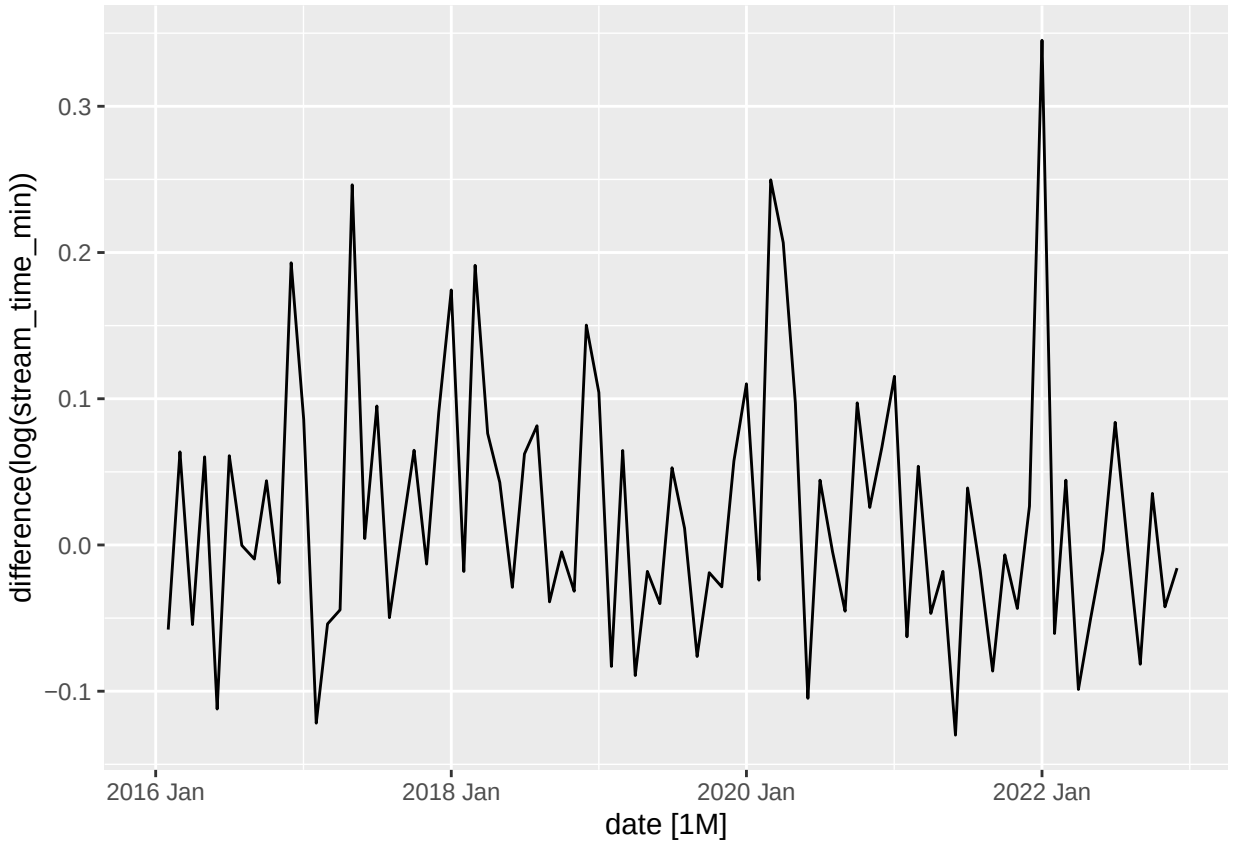
```
# Test stationarity with first differencing.
finalHOLD %>%features(difference(log(stream_time_min)),unitroot_kpss)
```

```
## # A tibble: 1 x 2
##   kpss_stat kpss_pvalue
##       <dbl>       <dbl>
## 1     0.140         0.1
```

As you can see, the KPSS test for our logged stream time series shows that we reject the null hypothesis that series is stationary at 1% level. However, the second test shows that perhaps first differencing is enough to obtain a stationary series, as we fail to reject that series is stationary at 10% level. Thus, we will proceed

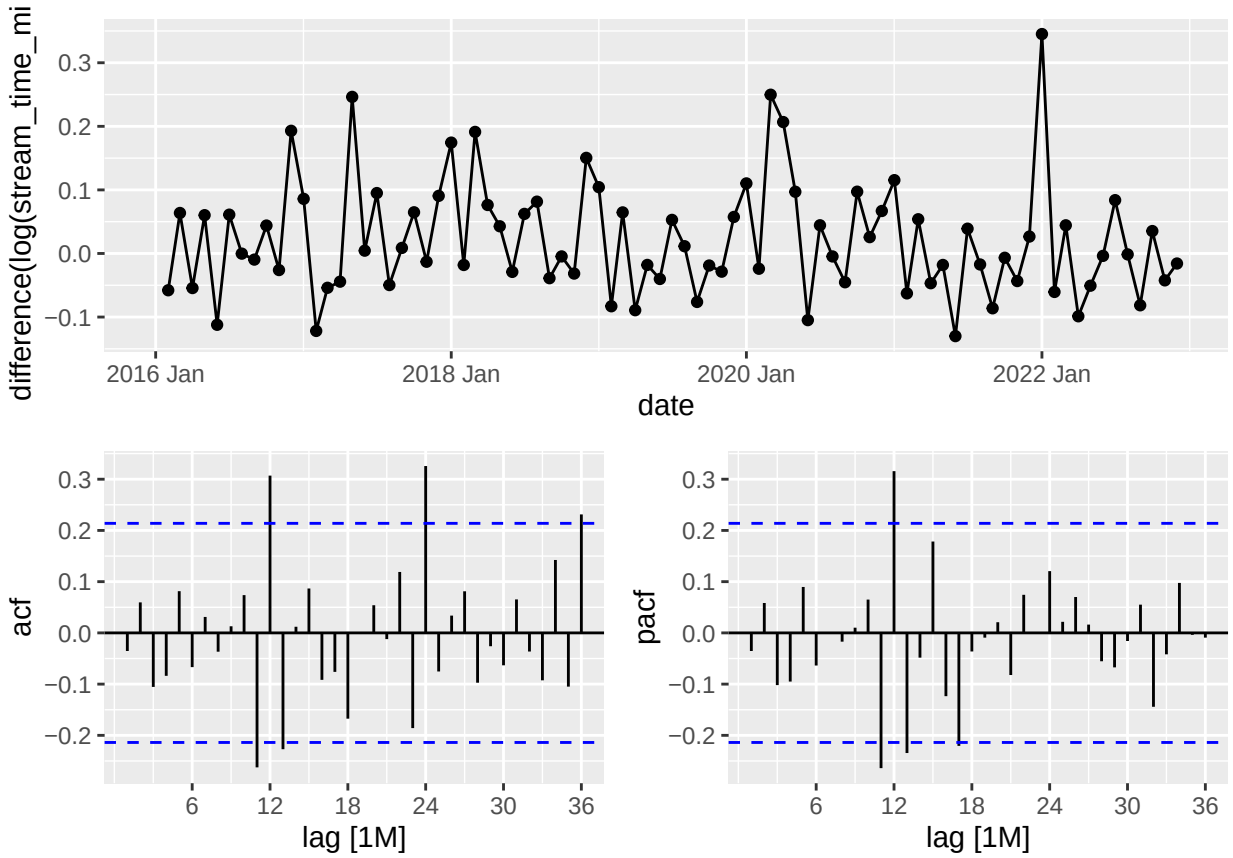
with the first differencing. Our differenced stream time looks like this:

```
finalHOLD %>% autoplot(difference(log(stream_time_min)))
```



To find which SARIMA model is most likely to have produced the data we observed, we will plot the autocorrelation function (ACF) and partial autocorrelation function (PACF).

```
finalHOLD %>%
  gg_tsdisplay(difference(log(stream_time_min)), lag_max=36, plot_type = 'partial')
```



There are two notable significant spikes at 12, 24, and 36th lags in ACF that are showing a sign of decay. We also see one signficant spike at 12 lag in PACF but not on 24. Other than that, we see significant lags near 12 for both acf and pacf. The corresponding BIC and AIC for all of my model guesses is as follows:

```
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(0,1,0) + PDQ(1,0,0))) %>% report()
```

```
## Series: stream_time_min
## Model: ARIMA(0,1,0)(1,0,0)[12]
## Transformation: log(stream_time_min)
##
## Coefficients:
##         sar1
##       0.4166
## s.e.  0.1075
##
## sigma^2 estimated as 0.007129:  log likelihood=86.78
## AIC=-169.55   AICc=-169.4   BIC=-164.72
```

```
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,0))) %>% report()
```

```
## Series: stream_time_min
## Model: ARIMA(1,1,0)(1,0,0)[12]
## Transformation: log(stream_time_min)
##
## Coefficients:
##          ar1    sar1
```

```
##       0.2771  0.5503
## s.e.  0.1180  0.1109
##
## sigma^2 estimated as 0.006617:  log likelihood=89.32
## AIC=-172.64   AICc=-172.34   BIC=-165.39
```

```
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(0,1,0) + PDQ(1,0,1))) %>% report()
```

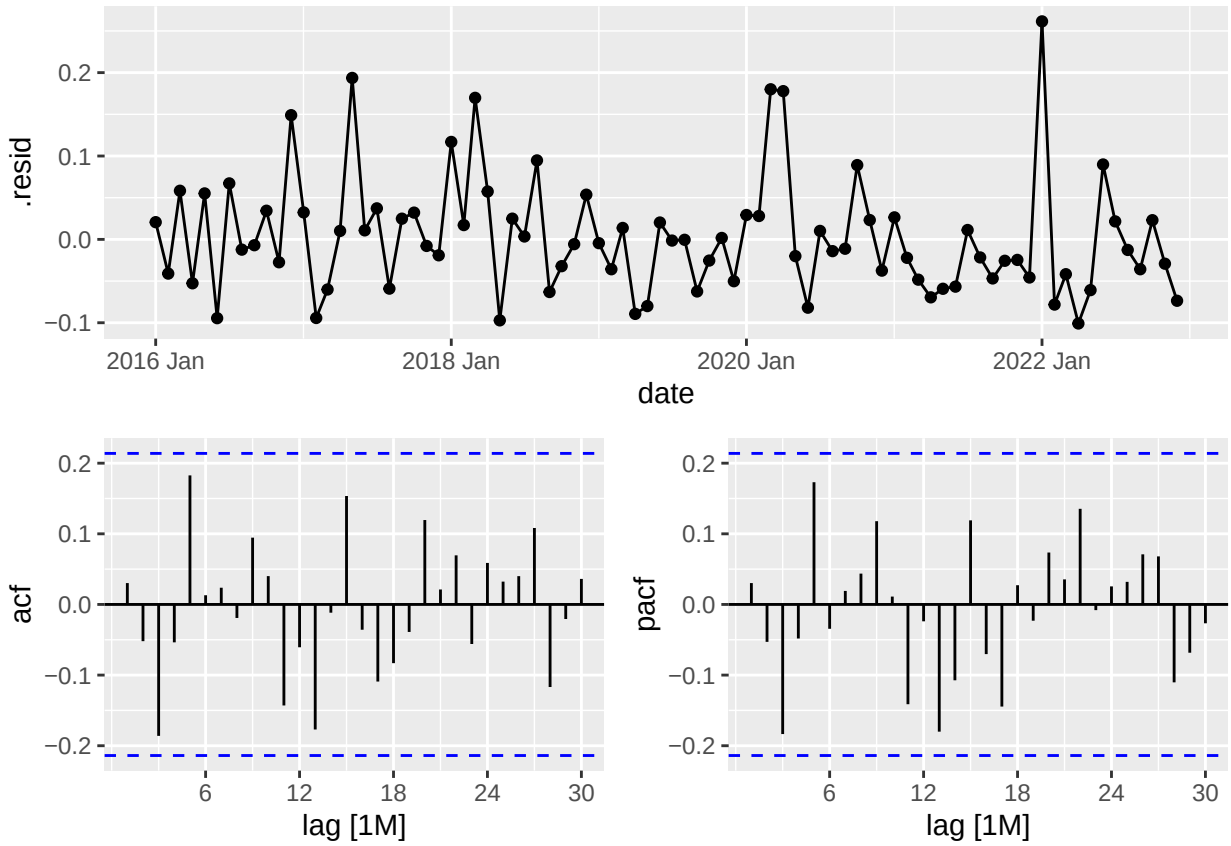
```
## Series: stream_time_min
## Model: ARIMA(0,1,0)(1,0,1)[12]
## Transformation: log(stream_time_min)
##
## Coefficients:
##         sar1     sma1
##       0.978  -0.8145
## s.e.  0.058   0.2447
##
## sigma^2 estimated as 0.005481:  log likelihood=93.35
## AIC=-180.71   AICc=-180.4   BIC=-173.45
```

```
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1))) %>% report()
```

```
## Series: stream_time_min
## Model: ARIMA(1,1,0)(1,0,1)[12]
## Transformation: log(stream_time_min)
##
## Coefficients:
##          ar1    sar1     sma1
##       0.2690  0.9630  -0.7153
## s.e.  0.1111  0.0472   0.1836
##
## sigma^2 estimated as 0.005186:  log likelihood=96.13
## AIC=-184.26   AICc=-183.75   BIC=-174.59
```

As you can see, the last model with p=1, P=1, Q=1, d=1 has the highest AIC and BIC, and we will choose that as our candidate model. As a final test, we will see if there is any other lags that are correlated with the error term of the candidate model. We will plot the ACF of residual and formally test the residual correlation using Ljung-Box test.

```
candidate = finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1)))

candidate %>% residuals() %>% gg_tsdisplay(.resid, lag_max = 30, plot_type = 'partial')
```

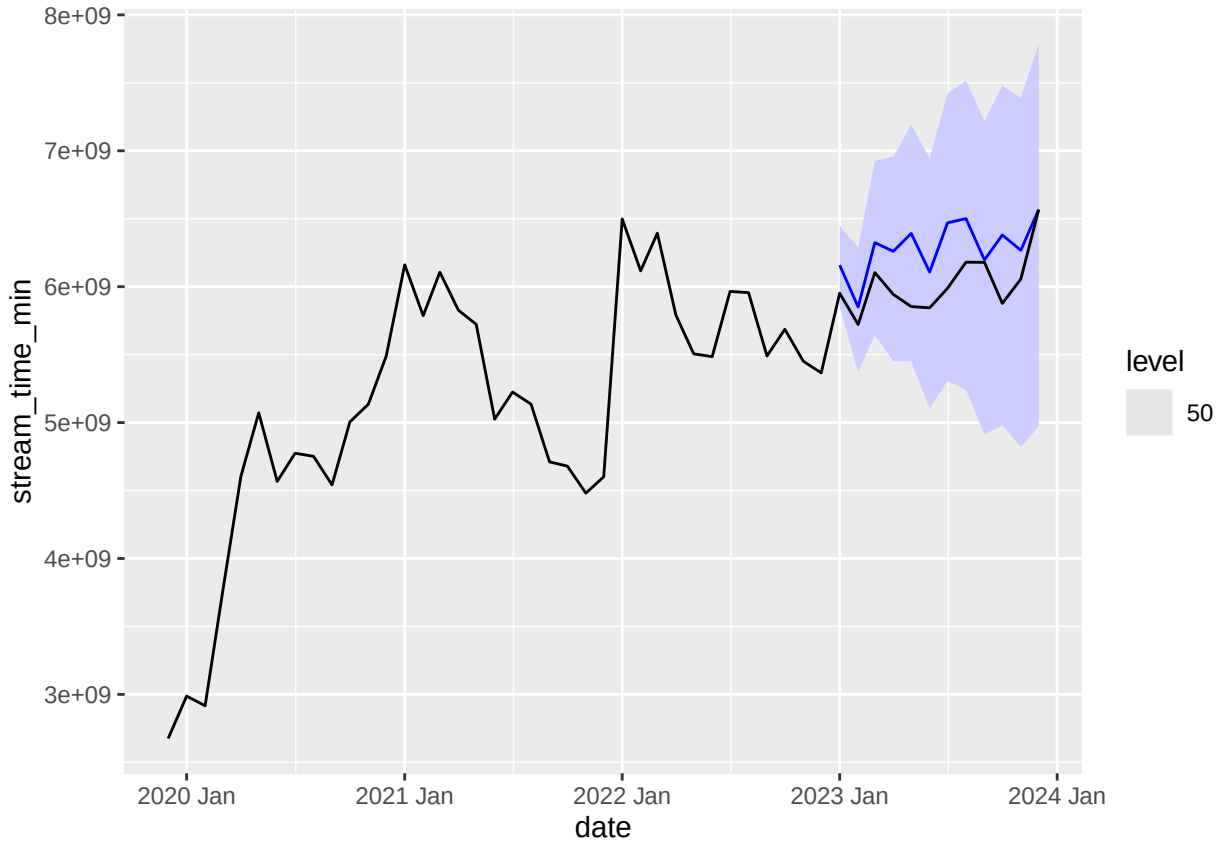



```
candidate %>% augment() %>% features(.innov,ljung_box,lag=60,dof=6)
```

```
## # A tibble: 1 x 3
##   .model                                                  lb_stat lb_pvalue
##   <chr>                                                     <dbl>     <dbl>
## 1 ARIMA(log(stream_time_min) ~ 0 + pdq(1, 1, 0) + PDQ(1, 0, 1~    57.5     0.346
```

As our data shows, we fail to reject that there is a significant residual correlation; thus, we will proceed with our candidate model. The following plot shows the estimated SARIMA model with a 50% confidence interval.

```
candidate %>% forecast(h=12) %>%
  autoplot(filter_index(final_data,"2019 Dec"~ "2023 Dec"), level = 50)
```

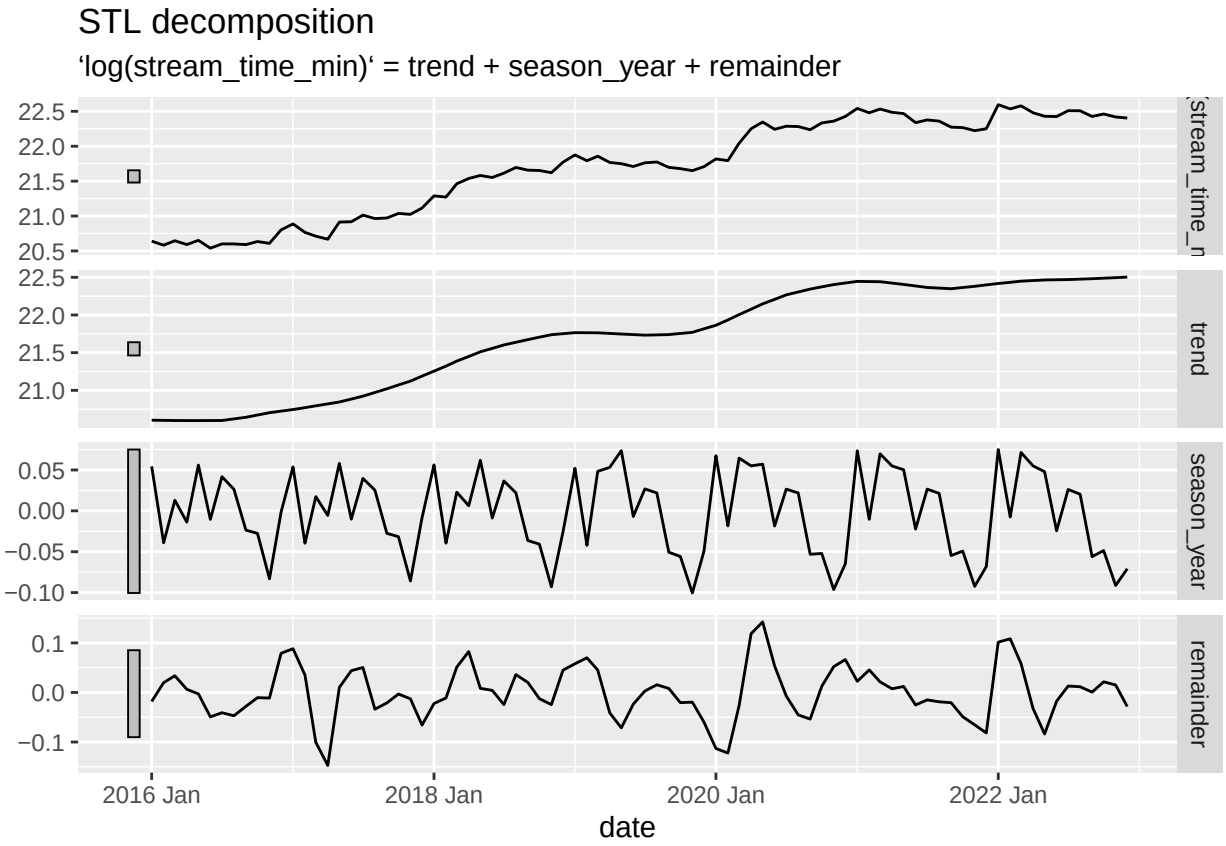


As you can see our SARIMA model has more or less closely forecasted the actual value of the stream time minute. However, we are now interested in different estimation methods.

ETS model

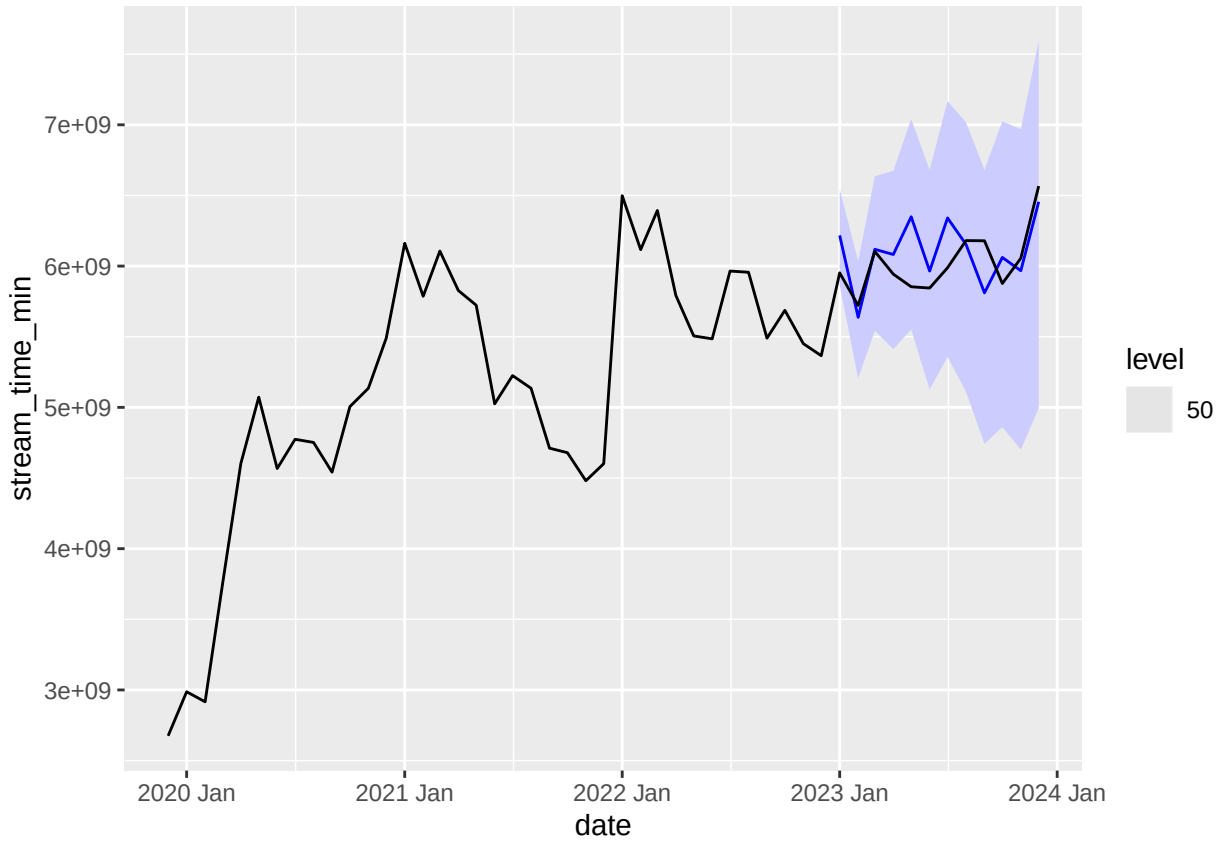
To run the ETS model we will need to decompose the series into season, trend, and level equations to find whether or not the series has a multiplicative or linear trend, season, and error. The following shows the STL decomposition of the series.

```
finalHOLD %>% model(STL(log(stream_time_min)~season(window=7)+
                          trend(window = 13))) %>% components() %>% autoplot()
```



The STL decomposition of setting seasonal window and trend window equal to 7 and 13 respectively shows that it is likely that we have a Additive trend, seasonality, and levels. We thus consider that model as our second candidate. The forecast generated by our model is displayed below.

```
candidate2 = finalHOLD %>% model(ETS(log(stream_time_min)~error("A")+trend("A")+
                                   season("A")))
candidate2%>%forecast(h=12)%>%
  autoplot(filter_index(final_data,"2019 Dec"~ "2023 Dec"), level = 50)
```

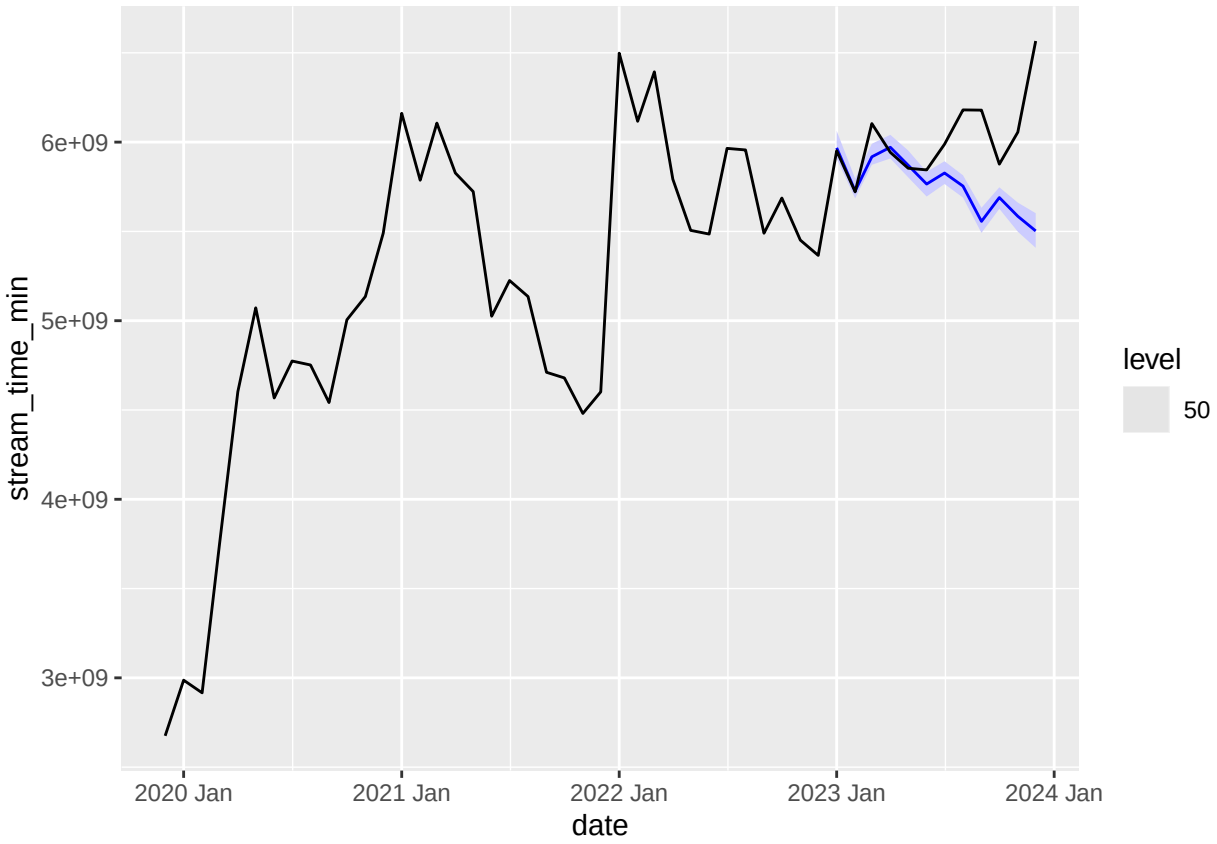


As you can observe, seems like ETS method has produced a prediction that is a bit more closer to actual than the SARIMA model.

Feed Forward Neural Net Model

Now, we will consider the Feed Forward Neural Net model (NNET) to forecast the twitch stream time. To do this, refer to the ACF and PACF plot in SARIMA model to decide the number of inputs in the neural network. As we have seen, we had significant lags at 12 multiples and other significant lags at 11 and 13 ACF and PACF. Moreover, both ACF and PACF shows complicated decay. For that reason, we will let p = 11 and P = 1.

```
candidate3 = finalHOLD %>% model(NNETAR(log(stream_time_min)~AR(P=1,p=11),n_networks=100))

candidate3 %>% forecast(h=12,times=100,bootstrap=TRUE)%>%
  autoplot(filter_index(final_data,"2019 Dec"~ "2023 Dec"), level = 50)
```

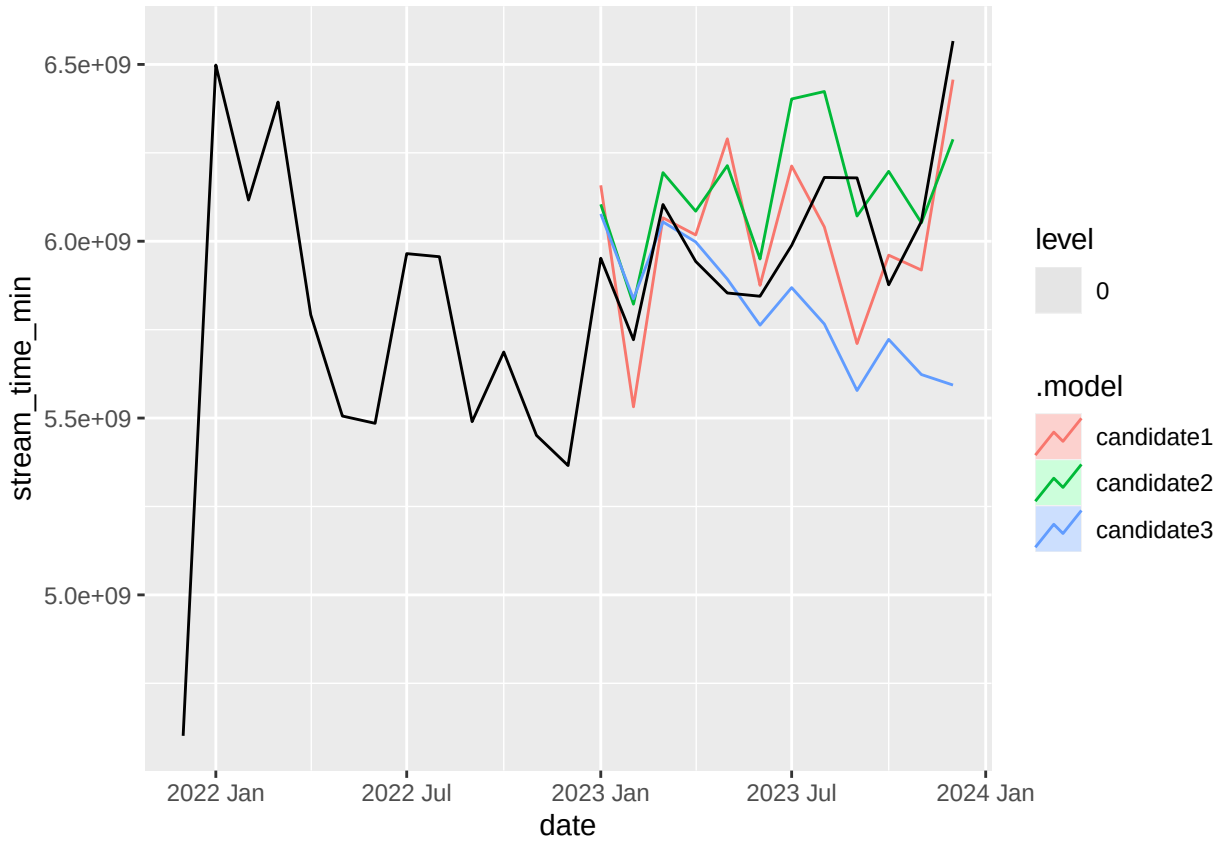


The neural net model has performed relatively better for a short-term forecast, but the accuracy declines as the forecast horizon increases.

Model Comparison

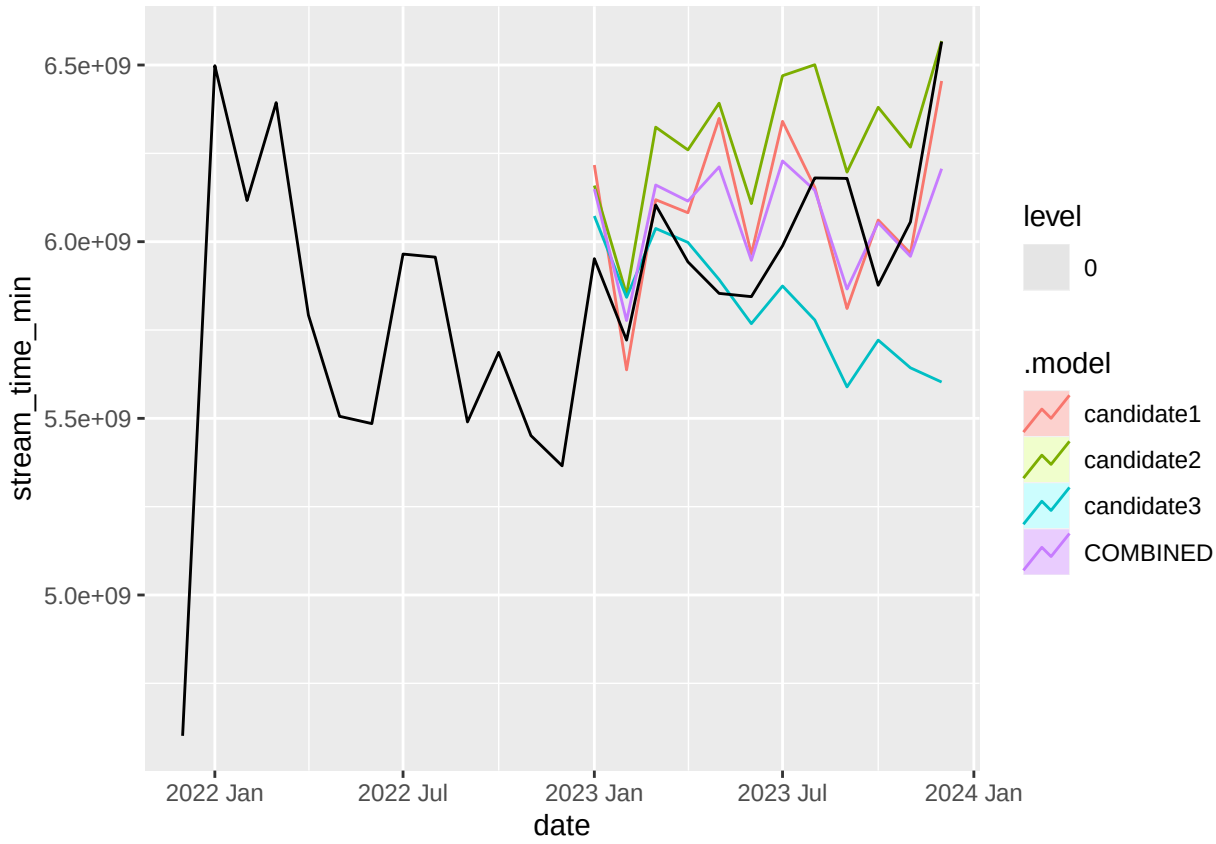
We now would like to compare three different models that we have treated in our analysis and take the average of these models to create a synthetic model that will help forecast the actual value. The following code produces a plot that compares three models we have chosen: ETS (candidate1), SARIMA(candidate2), and NNET (candidate3).

```
lastmodel = finalHOLD %>%
  model(candidate1 = ETS(log(stream_time_min)~error("A")+trend("A")+season("A")),
        candidate2 = ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1)),
        candidate3 = NNETAR(log(stream_time_min)~AR(P=1,p=11),n_networks=100))

fcst_12mnt = lastmodel %>% forecast(h=12, times = 100, bootstrap = TRUE)

fcst_12mnt %>%
  autoplot(filter_index(final_data,"2021 Dec"~ "2023 Dec"), level = 0)
```



It appears that candidate1 (ETS model) has the highest accuracy. However, as we can see, candidate1 tend to overestimate the forecast and candidate3 tends to underestimate the forecast with candidate2 somewhere in between. Thus, it would be a good idea to take the average of three methods to obtain a synthetic model.

```
comb = lastmodel %>% mutate(COMBINED = (candidate1+candidate2+candidate3)/3)

fcst_12mnt = comb %>% forecast(h=12)

fcst_12mnt %>% autoplot(filter_index(final_data,"2021 Dec"~ "2023 Dec"), level = 0)
```




```
residuals <- fcst_12mnt %>%
  as_tibble() %>%
  left_join(final_data%>%filter_index("2023 Jan" ~ "2023 Dec"), by = "date") %>%
  mutate(
    .resid = (.mean - stream_time_min.y)^2
  )

# Calculate MSE
mse <- residuals %>%
  group_by(.model) %>%
  summarise(MSE = mean(.resid, na.rm = TRUE))

print(mse)
```

```
## # A tibble: 4 x 2
##   .model         MSE
##   <chr>        <dbl>
## 1 COMBINED   4.50e16
## 2 candidate1 5.60e16
## 3 candidate2 1.00e17
## 4 candidate3 1.41e17
```

The MSE of all of the models are printed above. Combined model has the highest model accuracy, and candidate1 (ETS) comes in second. Thus, we conclude that even though the SLT model does the best job of predicting the actual values, the combined model further improves the forecasting accuracy even more.

Advanced Methods: Dynamic Regression and ETS with SARIMA error.

There is a reason to believe that total stream time of a Twitch is influenced by the price of telecommunication services such as cable and internet user services. Moreover, the unemployment level in the economy could potentially be related to the Twitch stream time as well. In this section, we will also consider adding the ETS model and SARIMA model synthetically. For instance, if you strip out the trend and seasonal component using ETS, we can use the SARIMA model on the remainder of the series to improve the forecast accuracy. In this section we will cover those topics.

Dynamic Regression with SARIMA error

Here, we will run the SARIMA model with exogenous variables: unemployment rate and the producer price index for internet/telecom services. Note that we have the same ACF and PACF as in the previous section. Thus, we will choose the above specified model for our SARIMA model. Now we must check if our exogenous variables are significant and whether or not we need any lags.

```
# Log of unemp
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1) +
                unemp + log(PPIintnet))
    ) %>% report()
```

```
## Series: stream_time_min
## Model: LM w/ ARIMA(1,1,0)(1,0,1)[12] errors
## Transformation: log(stream_time_min)
##
## Coefficients:
##          ar1    sar1     sma1   unemp  log(PPIintnet)
##       0.1761  0.9550  -0.6665  0.0207         -2.2420
## s.e.  0.1192  0.0458   0.1694  0.0059          1.4275
##
## sigma^2 estimated as 0.004564:  log likelihood=102.62
## AIC=-193.24   AICc=-192.13   BIC=-178.72
```

```
finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1) +
              unemp + log(PPIintnet) + lag(log(PPIintnet)) + lag(unemp)
              + log(mltrvl))
        ) %>% report()
```

```
## Series: stream_time_min
## Model: LM w/ ARIMA(1,1,0)(1,0,1)[12] errors
## Transformation: log(stream_time_min)
##
## Coefficients:
##          ar1    sar1     sma1   unemp  log(PPIintnet)  lag(log(PPIintnet))
##       0.1789  0.9629  -0.6336  0.0123         -1.4047              -0.0640
## s.e.  0.1191  0.0330   0.1494  0.0077          1.6115               1.4026
##       lag(unemp)  log(mltrvl)
##           0.0098      -0.3072
## s.e.      0.0072       0.1975
##
## sigma^2 estimated as 0.004402:  log likelihood=102.37
## AIC=-186.73   AICc=-184.27   BIC=-164.96
```

As you can see the first model has lower AIC and BIC and moreover the lag of unemployment and Producer price index are both insignificant. For that reason we will simply choose the first model as our choice.

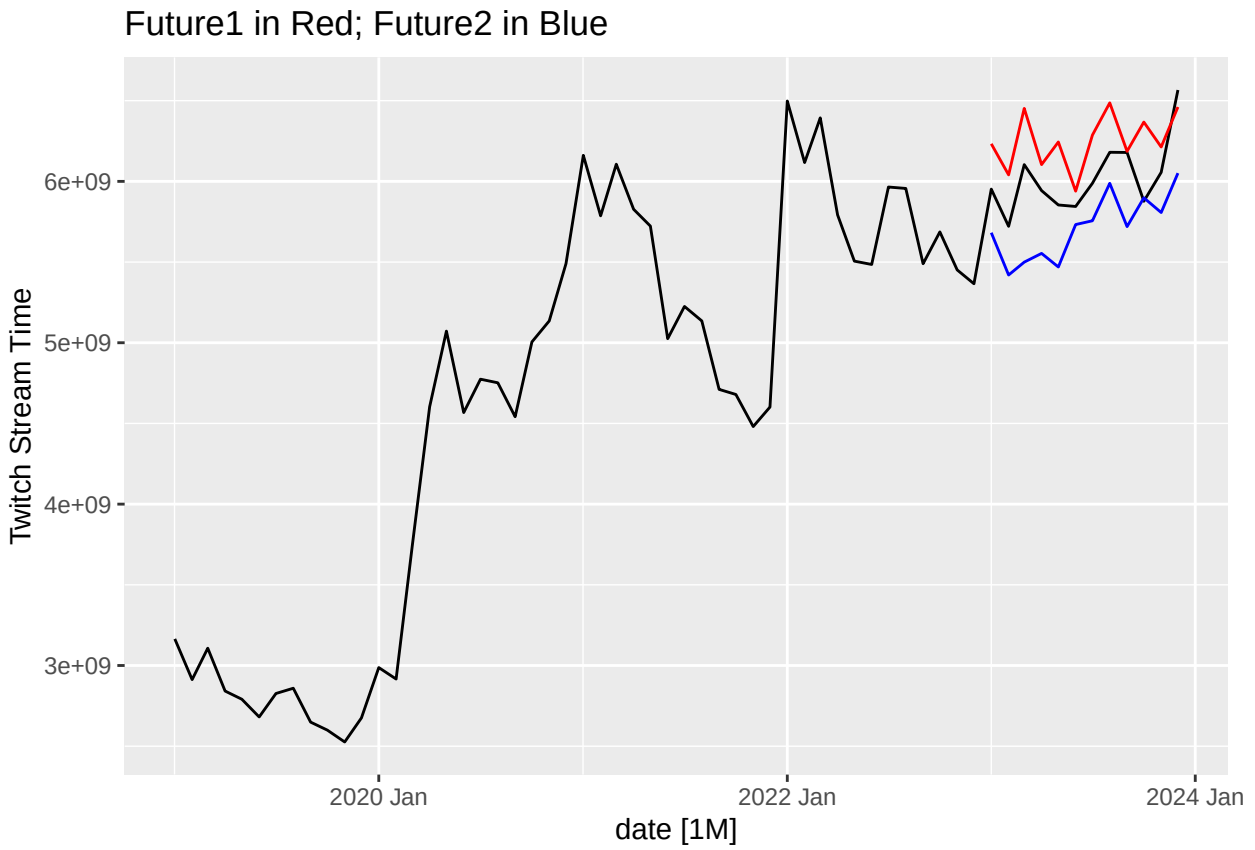
```
dynmc_arima1 <- finalHOLD %>%
  model(ARIMA(log(stream_time_min) ~ 0 + pdq(1,1,0) + PDQ(1,0,1) +
            log(unemp) + log(PPIintnet)))

dynmc_arima1 %>% augment() %>% features(.innov,ljung_box,lag=60,dof=6)
```

```
## # A tibble: 1 x 3
##   .model                                                    lb_stat lb_pvalue
##   <chr>                                                       <dbl>     <dbl>
## 1 "ARIMA(log(stream_time_min) ~ 0 + pdq(1, 1, 0) + PDQ(1, 0, ~    57.9     0.333
```

Moreover, we fail to reject the Ljung box test for residual correlation. Thus, we can safely proceed with this model. \ To forecast the future, we need the future values for our exogenous variables. Although, in real situations, we do not observe the future values, we rely on our best estimates of our exogenous variables in that case. In a different case, if we are interested in finding the sensitivity of our main variable with respect to the change in our exogenous variable, this method accomplishes exactly that.

Let us now generate the forecast using two future values and see if we can compare two different states of the world. Future1 uses all of the future values of PPI and employment. Future2 uses a very simple assumption based guesses of the future value of PPI and employment. Then we compare the result.

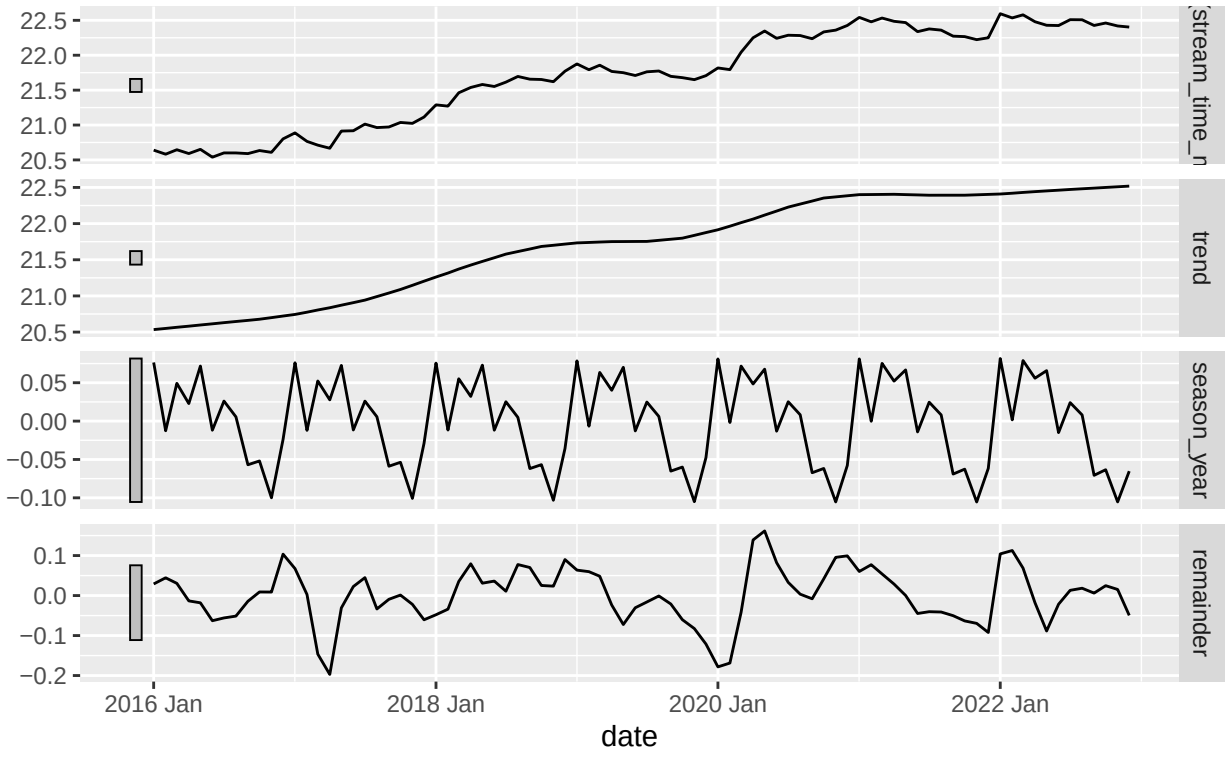
```
ftr_unemp =
  final_data %>% filter_index("2023 Jan" ~ "2023 Dec") %>% pull(unemp)
ftr_PPIintnet =
  final_data %>% filter_index("2023 Jan" ~ "2023 Dec") %>% pull(PPIintnet)
ftr_ffrate =
  final_data %>% filter_index("2023 Jan" ~ "2023 Dec") %>% pull(ffrate)

future1 = new_data(finalHOLD, 12) %>%
  mutate(unemp = ftr_unemp, PPIintnet = ftr_PPIintnet, ffrate = ftr_ffrate)

future2 = new_data(finalHOLD, 12) %>%
  mutate(unemp = c(2.3,3,3,3,3,3,3,3,3,3,3,3), PPIintnet = c(65,67,69,67,68,65,67,66,66,66,66,66),
         ffrate = c(4.5,4.5,4.5,4.5,4,4,4,4,4,4,4,4))

final_data%>%filter_index("2019 Jan"~"2023 Dec")%>%autoplot(stream_time_min)+
  autolayer(dynmc_arima1%>%forecast(future1),colour="RED",level=NULL)+
  autolayer(dynmc_arima1%>%forecast(future2),colour="BLUE",level=NULL)+
  ggtitle("Future1 in Red; Future2 in Blue")+
  labs(y="Twitch Stream Time")
```



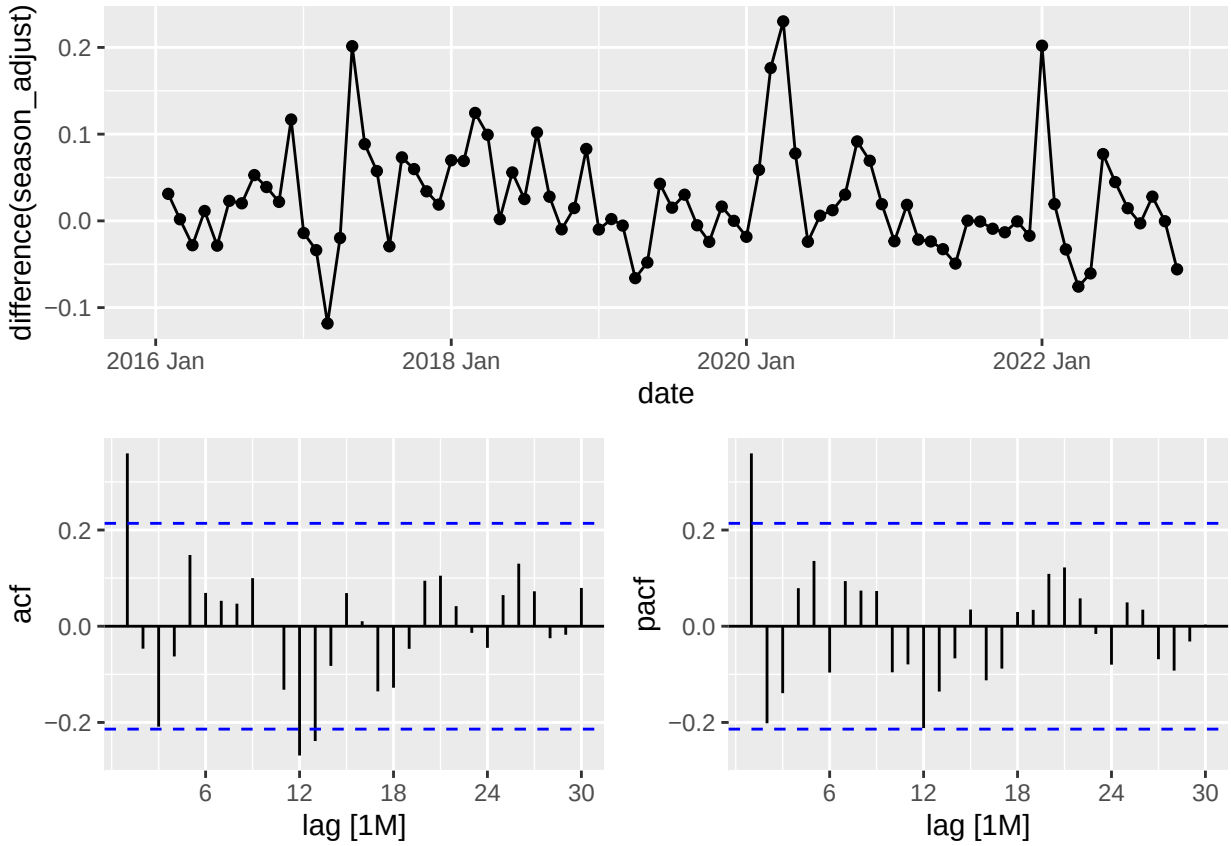
STL method with SARIMA

In this method, we will simply de-season the trend using STL decomposition and apply SARIMA in that component. Final model combines the STL decomposition and SARIMA model.

```
finalHOLD %>% model(STL(log(stream_time_min))) %>% components() %>% autoplot()
```




```
deSEASON = finalHOLD %>% model(STL(log(stream_time_min))) %>% components() %>%
  select(season_adjust)

deSEASON %>% gg_tsdisplay(difference(season_adjust), lag_max=30, plot_type = 'partial')
```

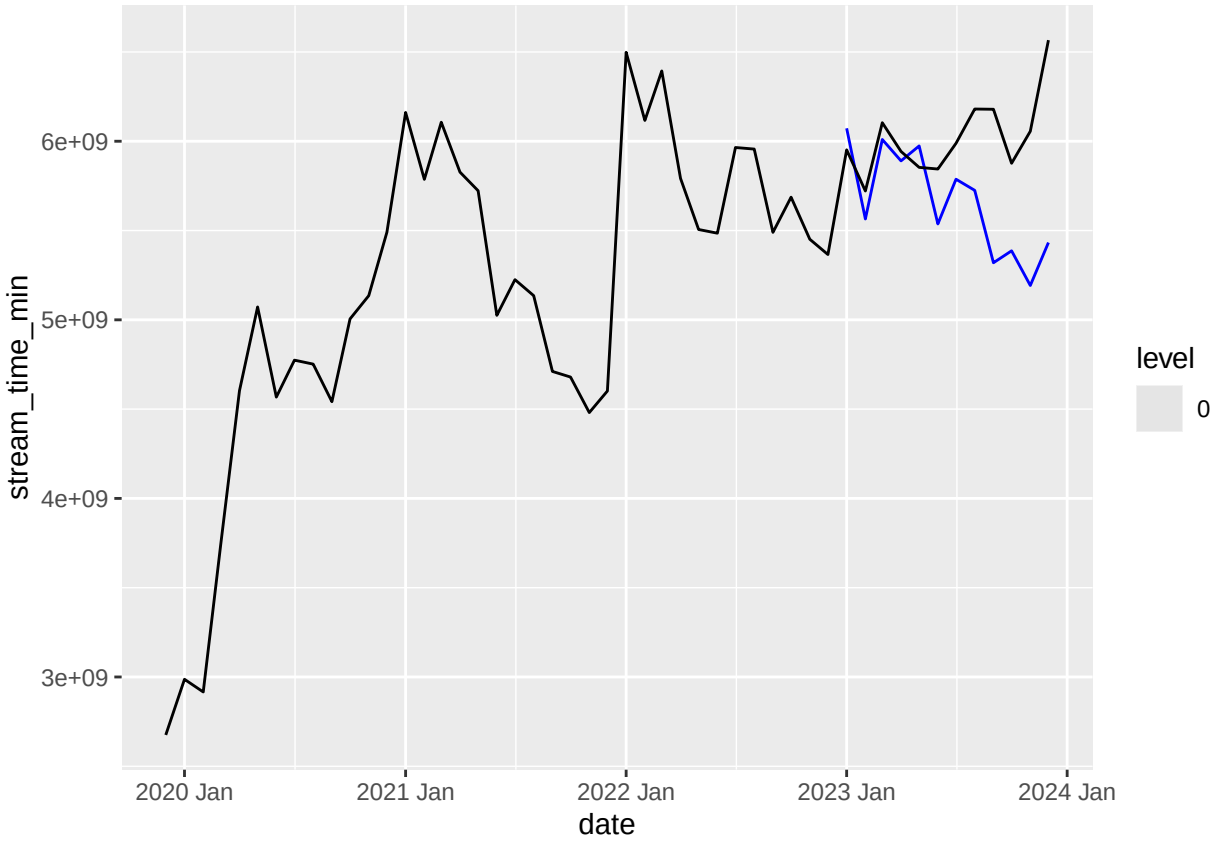



```
STL_SARIMA = decomposition_model(
          STL(log(stream_time_min)),
          ARIMA(season_adjust~0+pdq(1,1,0)+PDQ(0,0,0)))

finalHOLD %>% model(STL_SARIMA) %>% report()
```

```
## Series: stream_time_min
## Model: STL decomposition model
## Transformation: log(stream_time_min)
## Combination: season_adjust + season_year
##
## ========================================
##
## Series: season_adjust
## Model: ARIMA(1,1,0)
##
## Coefficients:
##          ar1
##       0.4429
## s.e.  0.0979
##
## sigma^2 estimated as 0.0034:  log likelihood=118.57
## AIC=-233.15   AICc=-233   BIC=-228.31
##
## Series: season_year
## Model: SNAIVE
##
```

```
## sigma^2: 0
```

```
finalHOLD %>% model(STL_SARIMA) %>% forecast(h=12) %>%
  autoplot(filter_index(final_data,"2019 Dec"~ "2023 Dec"), level = 0)
```



Unfortunately, the STL with SARIMA method did not do very well in predicting the actual values as much as our previous method did.

In practice, the dynamic regression model with SARIMA error offers a great benefit that allows the contingency analysis. With this, we conclude the time series analysis of twitch stream time.