Trabalhando com o pacote Modeltime

Ivanildo Batista da Silva Júnior

9 de abril de 2021

Um grande desafio para organizações, empresas e governos e estimar um horizonte previsível do futuro, pois sabe-se que a ocorrência de eventos são dotados de incertezas. Um tipo de dado bastante conhecido é a série temporal, que como o nome já explicita, é uma sequência de valores ao longo de um período de tempo (exemplo: preço de commodities durante 10 anos, precipitação de chuva anual, velocidade do vento anual, número mensal de homicídios, consumo de energia elétrica semestral, etc.).Esses dados são analisados/estudados por especialistas com o objetivo de identificar padrões e extrair *insights*; e dentro dessa análise há a etapa de previsão, ou seja, tentar estimar como será o comportamento dessa série de tempo em um futuro desconhecido.

Para auxiliar nesse desafio de criar cenários, gerar previsões e na tomada de decisão do analista, são usados modelos matemáticos/estatísticos que podem ser determinísticos ou estocásticos. Existem vários tipos de modelos que, dos mais simples aos mais complexos, possuem uma variedade de parâmetros.

O *R* possui vários pacotes para analisar e modelar séries temporais, porém um pacote bastante interessante que possui uma variedade de modelos é a *modeltimes*. Esse pacote usa a estrutura de previsão de séries temporais para uso com o ecossistema *tidymodels*, um outro pacote do *R*.

Os modelos incluem ARIMA, Suavização exponencial *Holt-Winter* e modelos de série temporal adicionais dos pacotes de *forecast* e *Prophet Facebook*. Assim sendo, posso treinar vários modelos de séries temporais, tornando esse pacote uma ferramenta de *Auto Time Series*, assim como funciona pacote de *Auto Machine Learning*.

Dados

Os dados foram extraídos do *site* do FRED (*Federal Reserve Economic Data*) e trata da vendas no varejo de lojas de cerveja, vinho e licores. Os dados possuem periodicidade mensal de Janeiro de 1992 até Janeiro de 2021. Os dados podem ser obtidos em aqui (https://fred.stlouisfed.org/series/MRTSSM4453USN).

Vamos iniciar importando os pacotes.

Instalando os pacote

```
library(xgboost)
```

```
## Warning: package 'xgboost' was built under R version 4.0.5
```

```
library(tidymodels)
```

```
## Warning: package 'tidymodels' was built under R version 4.0.5
```

```
## -- Attaching packages --------------------------------------- tidymodels 0.1.2 --
```

```
## v broom     0.7.4      v recipes   0.1.15
## v dials     0.0.9      v rsample   0.0.8
## v dplyr     1.0.5      v tibble    3.0.4
## v ggplot2   3.3.3      v tidyr     1.1.2
## v infer     0.5.4      v tune      0.1.2
## v modeldata 0.1.0      v workflows 0.2.1
## v parsnip   0.1.5      v yardstick 0.0.8
## v purrr     0.3.4
```

```
## Warning: package 'dplyr' was built under R version 4.0.4
```

```
## Warning: package 'ggplot2' was built under R version 4.0.4
```

```
## Warning: package 'yardstick' was built under R version 4.0.4
```

```
## -- Conflicts ---------------------------------------- tidymodels_conflicts() --
## x purrr::discard() masks scales::discard()
## x dplyr::filter()  masks stats::filter()
## x dplyr::lag()     masks stats::lag()
## x dplyr::slice()   masks xgboost::slice()
## x recipes::step()  masks stats::step()
```

```
library(modeltime)
```

```
## Warning: package 'modeltime' was built under R version 4.0.5
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.0.5
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.0 --
```

```
## v readr   1.4.0     v forcats 0.5.1
## v stringr 1.4.0
```

```
## Warning: package 'stringr' was built under R version 4.0.4
```

```
## -- Conflicts ---------------------------------------- tidyverse_conflicts() --
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x dplyr::filter()     masks stats::filter()
## x stringr::fixed()    masks recipes::fixed()
## x dplyr::lag()        masks stats::lag()
## x dplyr::slice()      masks xgboost::slice()
## x readr::spec()       masks yardstick::spec()
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.0.5
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(timetk)
```

```
## Warning: package 'timetk' was built under R version 4.0.5
```

E em seguida importando os dados.

Importando os dados

```
beer = read_delim('MRTSSM4453USN.csv', delim = ',')
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   DATE = col_date(format = ""),
##   MRTSSM4453USN = col_double()
## )
```

Transformando a base de dados em uma série temporal

```
beer_time_series <- ts(beer[,2],start=c(1992,1),
                       end=c(2021,1),frequency=12)
```

Análise exploratória dos dados

Agora irei fazer uma breve análise exploratória dos dados.

Primeiras e últimas observações

```
head(beer_time_series)
```

```
##      MRTSSM4453USN
## [1,]          1509
## [2,]          1541
## [3,]          1597
## [4,]          1675
## [5,]          1822
## [6,]          1775
```

```
tail(beer_time_series)
```

```
##      MRTSSM4453USN
## [1,]          6099
## [2,]          5855
## [3,]          6050
## [4,]          6086
## [5,]          7739
## [6,]          5320
```

Dimensão da base de dados

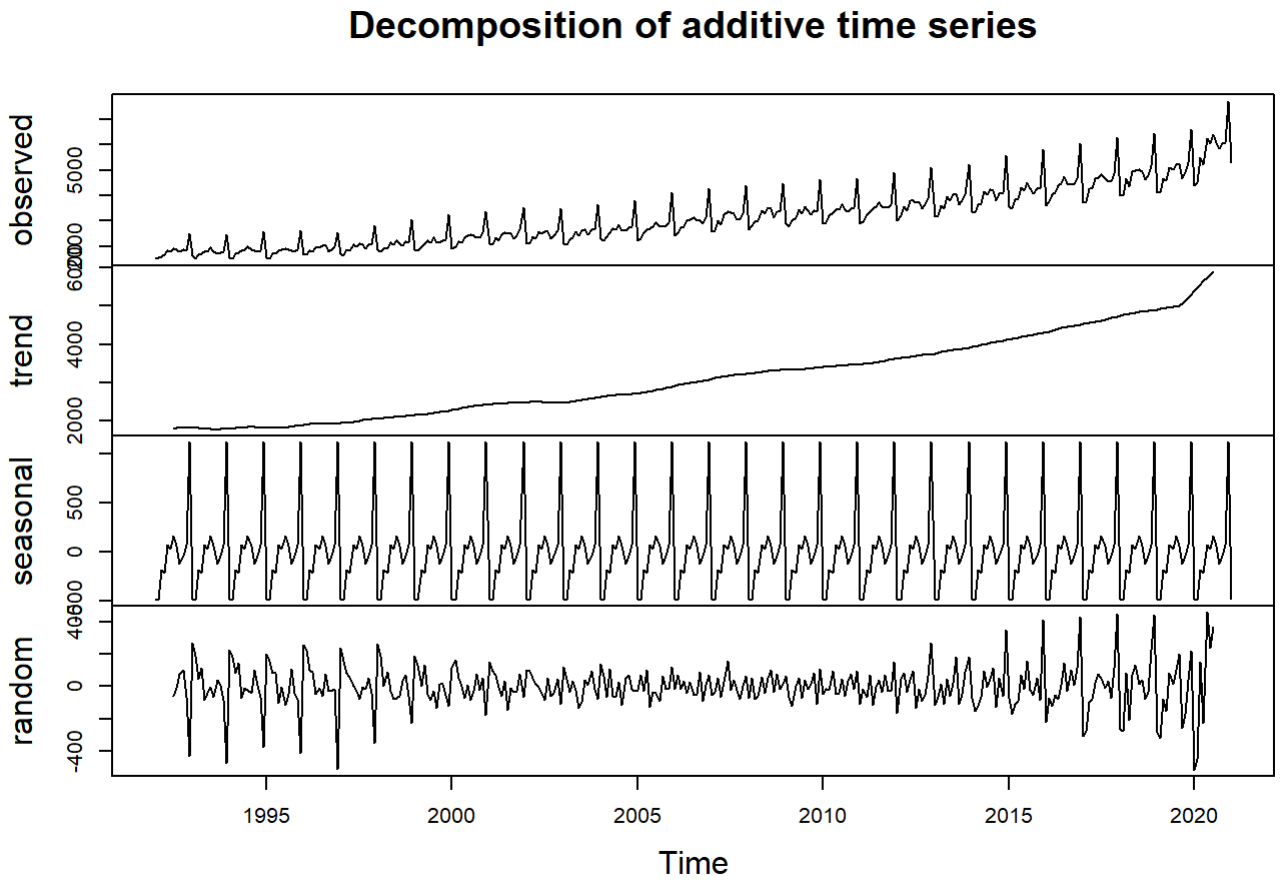
```
dim(beer_time_series)
```

```
## [1] 349   1
```

Decomposição da série

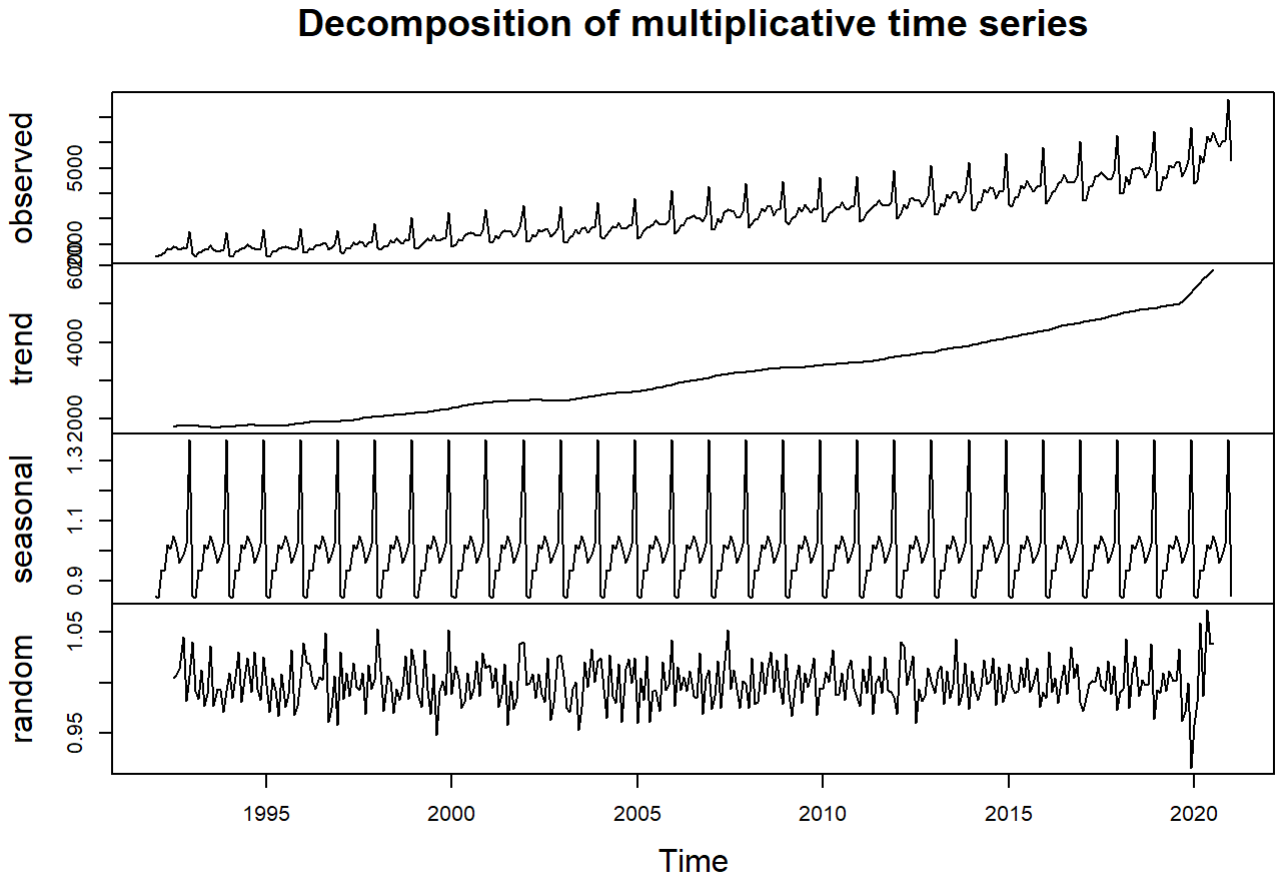
Realizei a decomposição da série, primeiramente na forma aditiva, onde vemos a série observada, em seguida a tendência (percebemos no final da tendência uma mudança de inclinação da mesma), depois a sazonalidade e, por fim, os resíduos (onde ainda consta a presença de algum resquícios de sazonalidade).

```
#decomposição aditiva
plot(decompose(beer_time_series))
```



N decomposição multiplicativa observa-se uma decomposição semelhante, mas com a diferença nos resíduos que assemelham-se com um ruído branco.

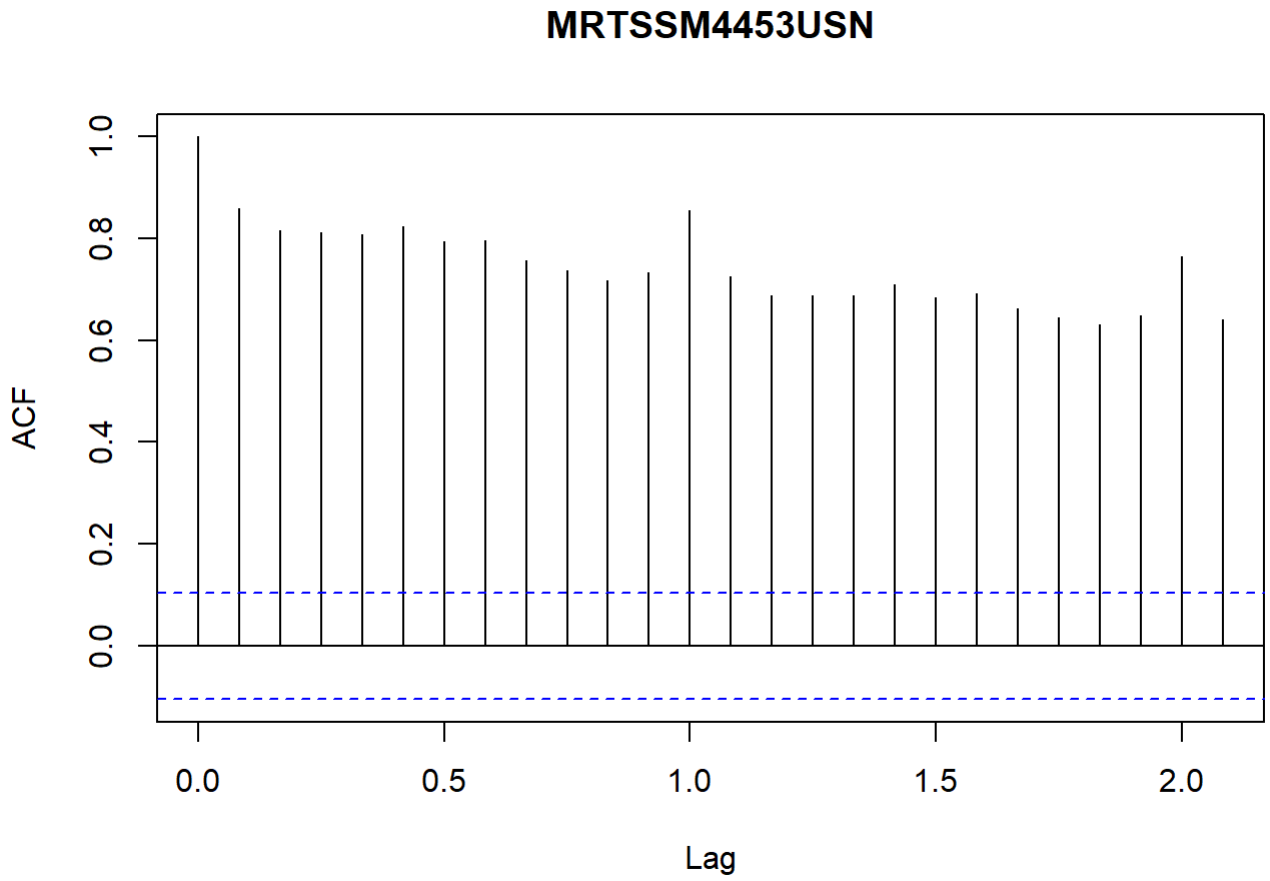
```
#decomposição multiplicativa
plot(decompose(beer_time_series, type="mult"))
```



Gráficos de autocorrelação e autocorrelação parcial

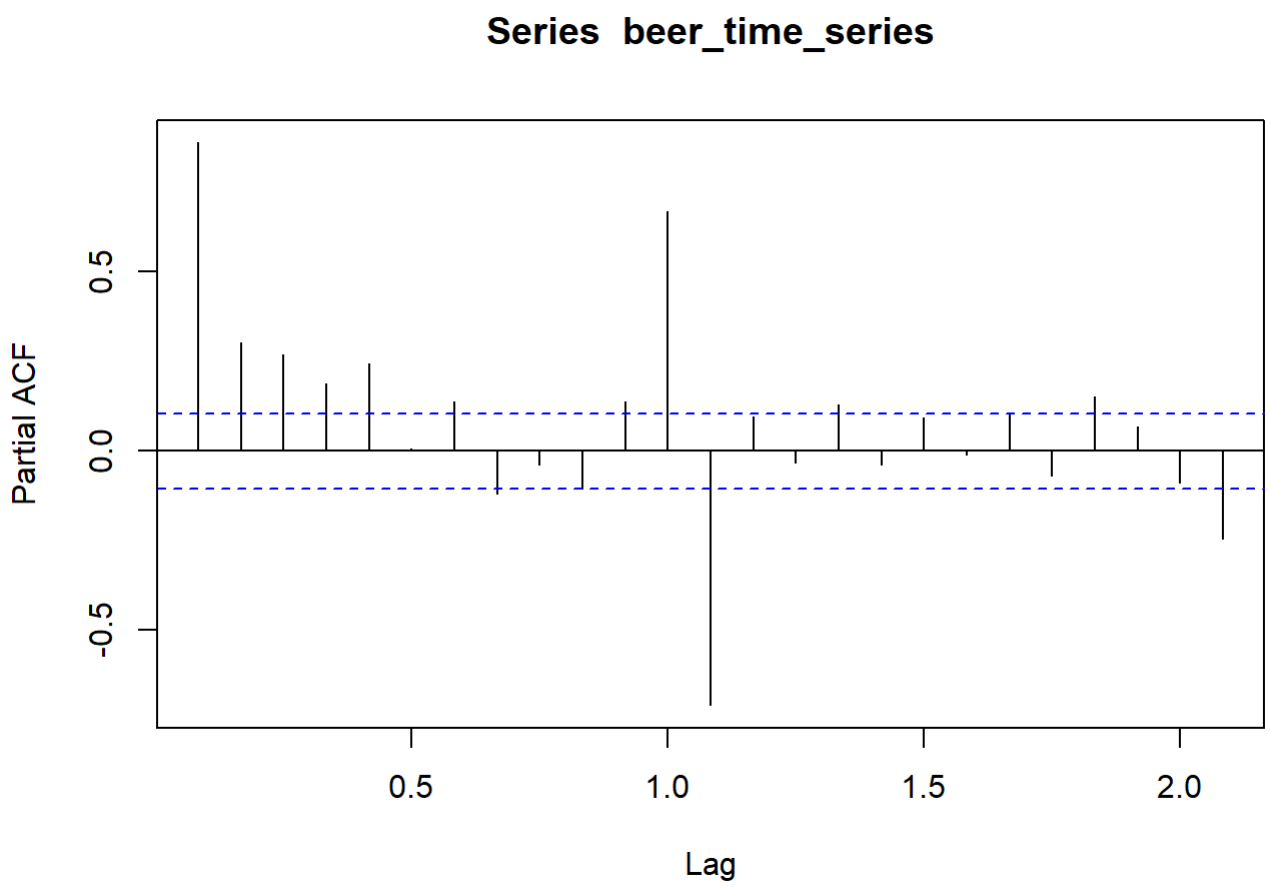
Pelos gráficos abaixo podemos oberservar que a série é não estacionária, pois há um decaimento demorado do gráfico de autocorrelação.

```
acf(beer_time_series)
```



No gráfico de autocorrelação parcial, mesmo com o rápido decaimento, vemos defasagens que são significativas.

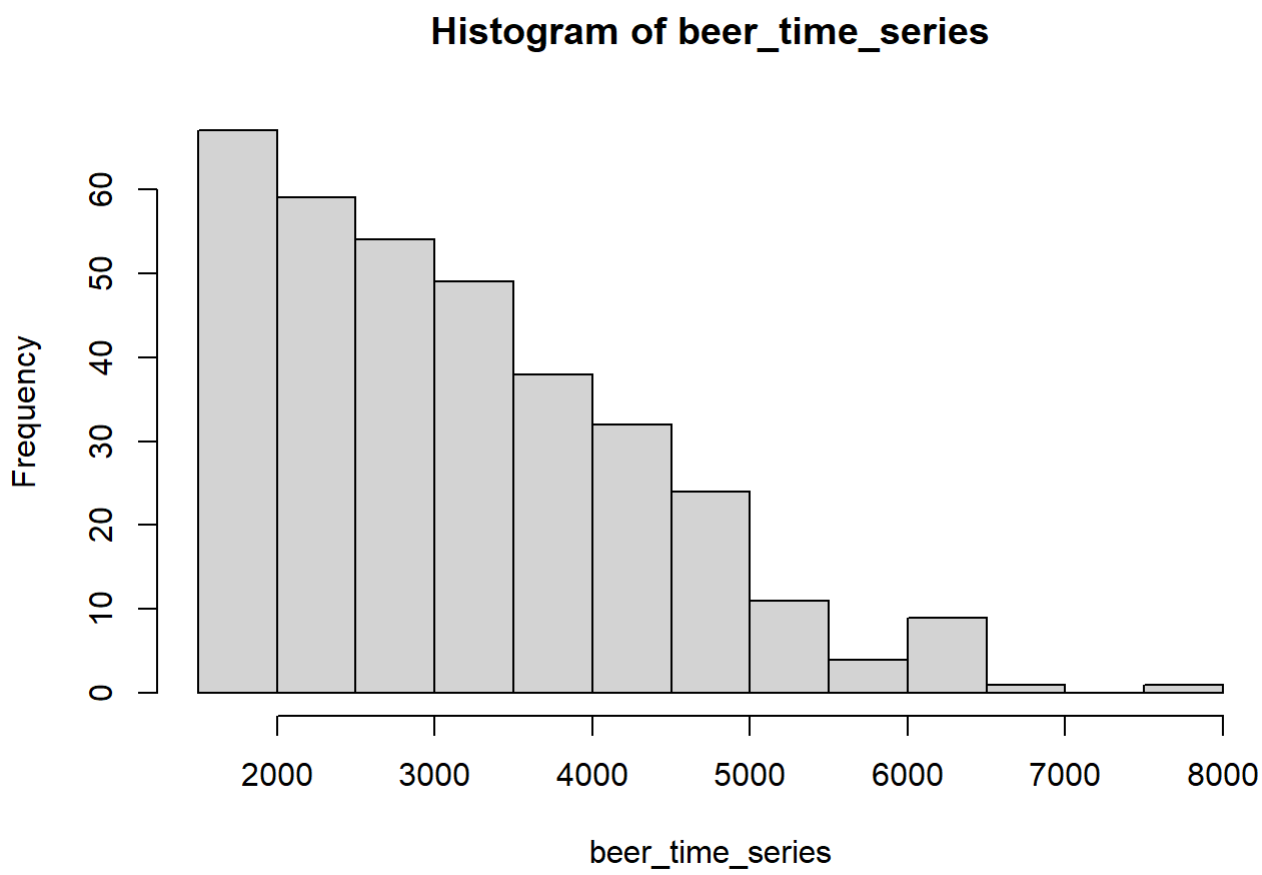
```
pacf(beer_time_series)
```



Histograma da série temporal

O comportamento dos dados da série não é normal (formato de sino).

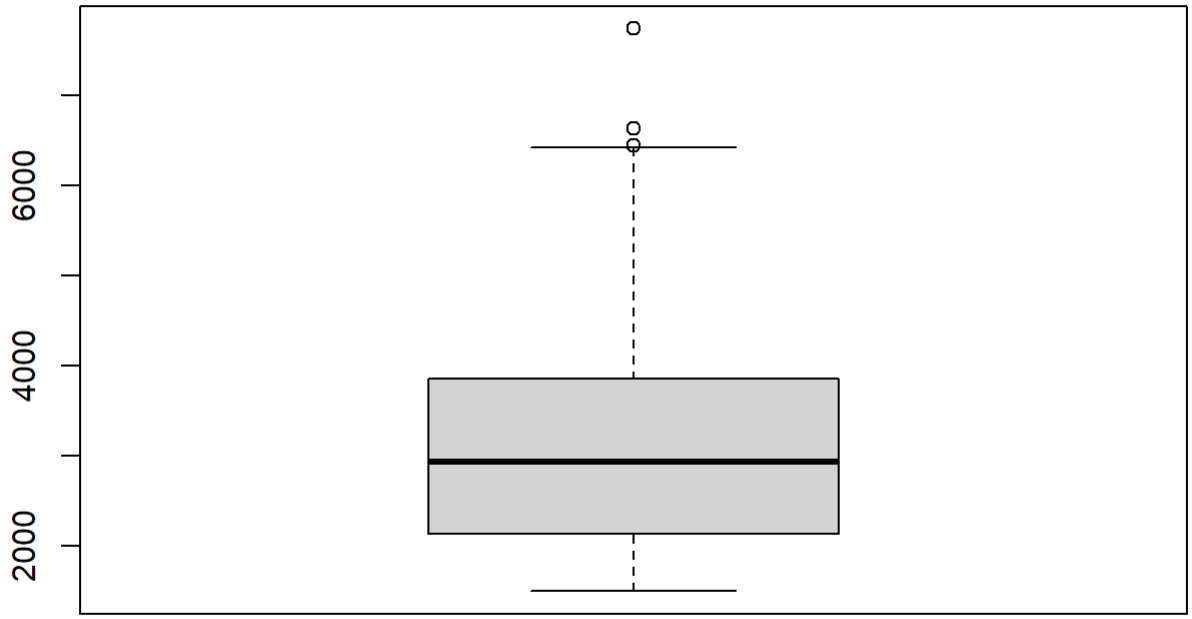
```
hist(beer_time_series)
```



Boxplot da série temporal

Há apenas 3 valores extremos.

```
boxplot(beer_time_series)
```



Modelagem da série temporal

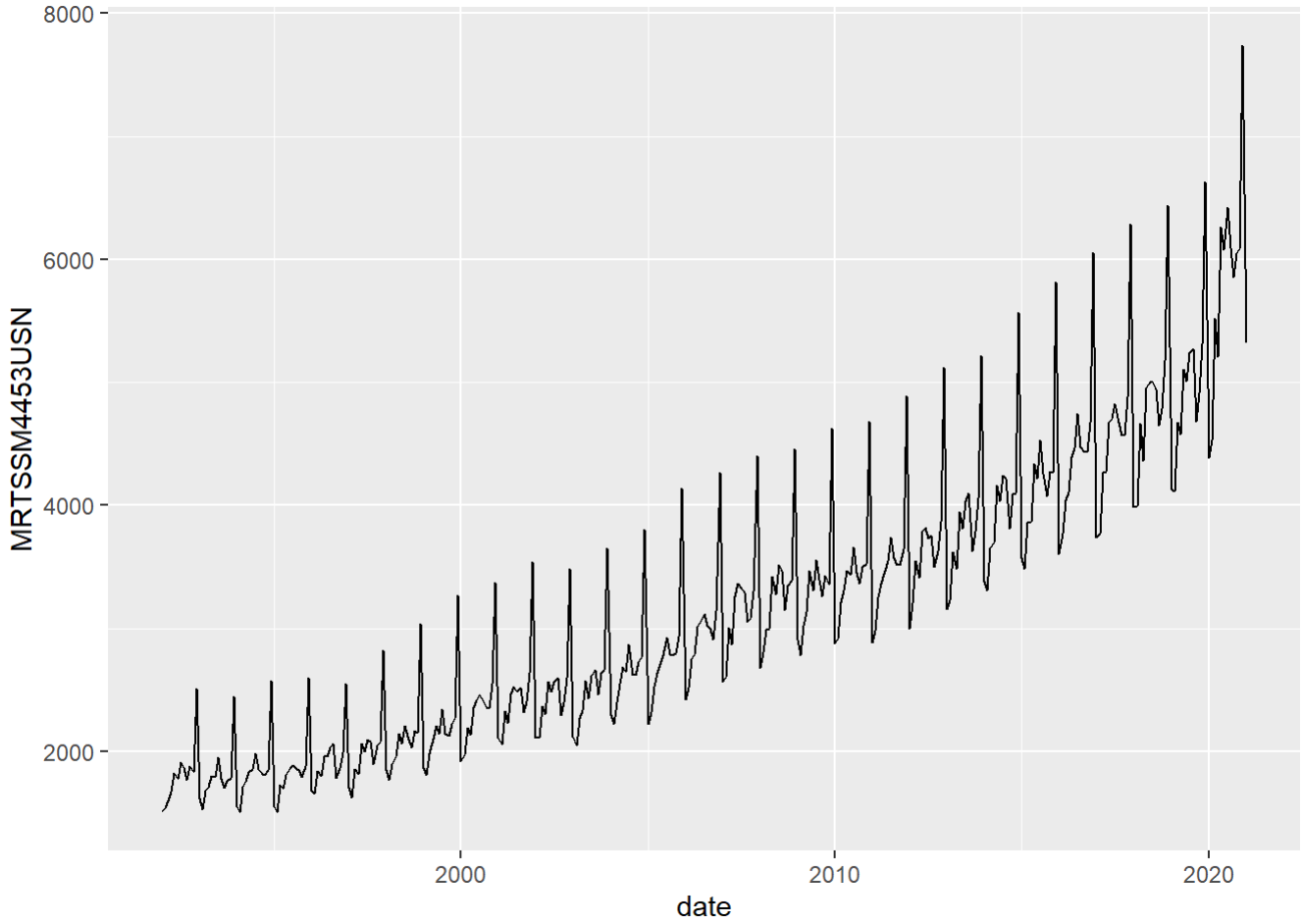
Transformando a base de dados para o formato adequado.

```
beer <- beer %>% select(date = DATE, MRTSSM4453USN)
```

Plotando a série temporal

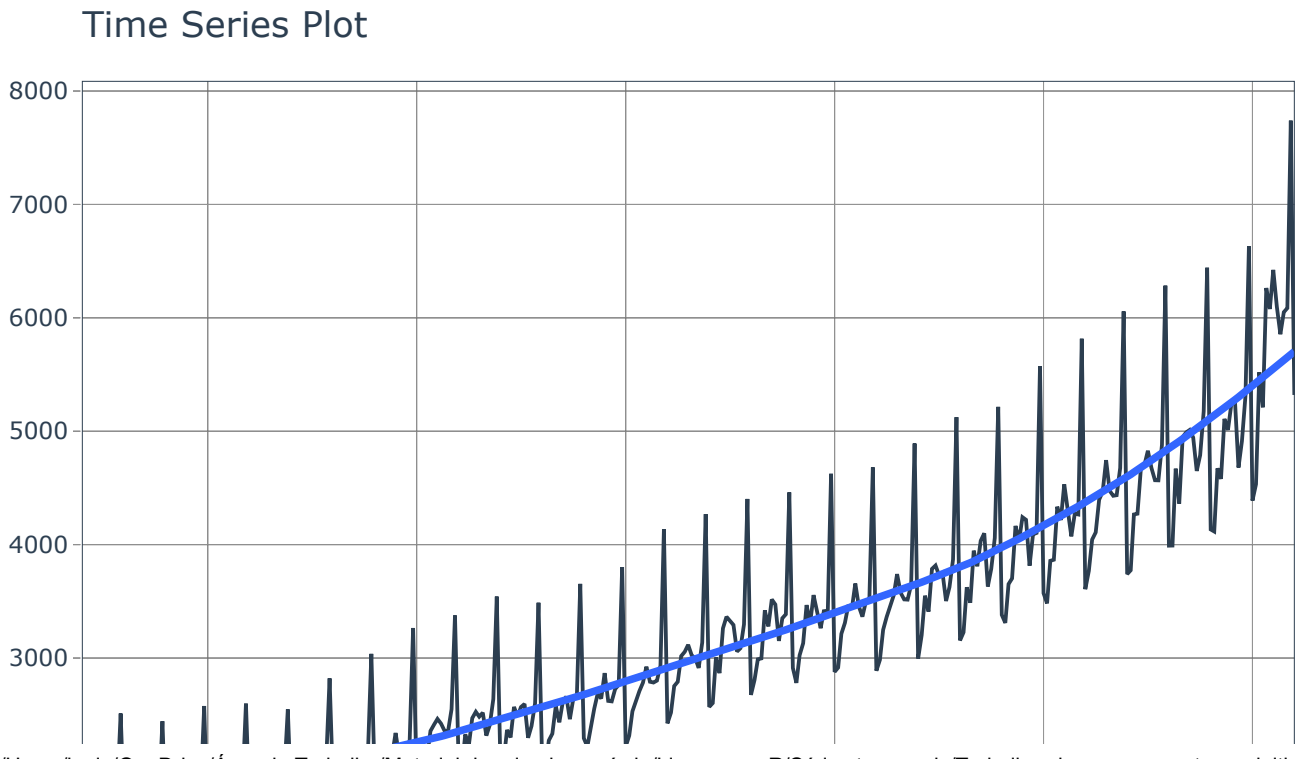
Nos gráfico abaixo podemos ver que a série possui forte presença de tendência global e de sazonalidade.

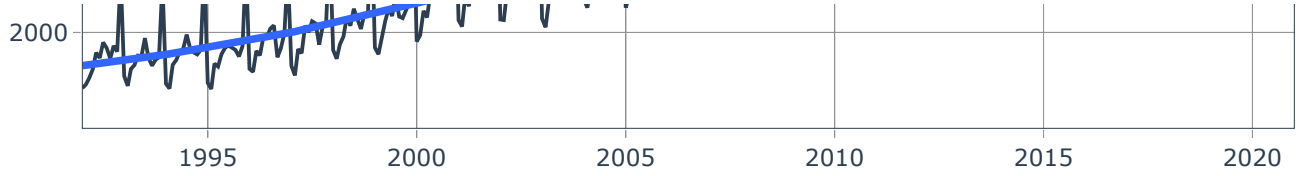
```
beer %>%
  ggplot(aes(x = date, y = MRTSSM4453USN)) +
  geom_line()
```



com a função do pacote modeltime

```
#cria uma linha com a tendÃªncia da sÃ©rie.
beer %>%
  plot_time_series(date, MRTSSM4453USN)
```





Separando os dados para modelo

85% da série será usada para treino dos modelos e os outros 15% para realizaçãoo da validação dos modelos.

```
splits <- initial_time_split(beer, prop = 0.8)
splits
```

```
## <Analysis/Assess/Total>
## <279/70/349>
```

Treinamento os modelos

Nessa etapa irei treinar 8 modelos diferentes utilizando os parâmetros padrão de cada modelo.

```
#modelo 1: MODELO AUTO ARIMA
modelo_1 <- arima_boost(min_n = 2,learn_rate = 0.015) %>%
  set_engine(engine = "auto_arima_xgboost") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#modelo 2 :
modelo_2 <- arima_reg() %>%
  set_engine(engine = "auto_arima") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#modelo 3 : PROPHET
modelo_3 <- prophet_reg() %>%
  set_engine(engine = "prophet") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
#modelo 4 : PROPHET XGBOOST
#model_4 <- prophet_reg() %>%
 # set_engine(engine = "prophet_xgboost") %>%
  #fit(MRTSSM4453USN ~ date, data = training(splits))

#modelo 5
modelo_5 <- exp_smoothing() %>%
  set_engine(engine = "ets") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#model 6
modelo_6 <- seasonal_reg() %>%
  set_engine(engine = "stlm_arima") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#model 7
modelo_7 <- seasonal_reg() %>%
  set_engine(engine = "stlm_ets") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#modelo 8
modelo_8 <- prophet_boost() %>%
  set_engine(engine = "prophet_xgboost") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
#modelo 9
modelo_9 <- nnetar_reg() %>%
  set_engine(engine = "nnetar") %>%
  fit(MRTSSM4453USN ~ date, data = training(splits))
```

```
## frequency = 12 observations per 1 year
```

```
#modelo 10
#modelo_10 <- naive_reg() %>%
 # set_engine(engine = "naive") %>%
#  fit(MRTSSM4453USN ~ date, data = training(splits))
```

Criando a tabela com os modelos

Inserindo os modelos em uma tabela para realizar as próximas etapas com todos eles simultaneamente.

```
tabela_de_modelos <- modeltime_table(
  modelo_1,
  modelo_2,
  modelo_3,
  #model_4,
  modelo_5,
  modelo_6,
  modelo_7,
  modelo_8,
  modelo_9
 # model_10
)
```

Tabela com os modelos

Vemos que os parâmetros de cada modelos foram escolhidos de forma automática pelo pacote no momento do treinamento. Esse parâmetros podem ser alterados a critério de quem está manipulando o pacote.

```
tabela_de_modelos
```

```
## # Modeltime Table
## # A tibble: 8 x 3
##   .model_id .model   .model_desc
##       <int> <list>   <chr>
## 1         1 <fit[+]> ARIMA(3,1,2)(1,1,1)[12]
## 2         2 <fit[+]> ARIMA(3,1,2)(0,1,2)[12]
## 3         3 <fit[+]> PROPHET
## 4         4 <fit[+]> ETS(M,A,M)
## 5         5 <fit[+]> SEASONAL DECOMP: ARIMA(2,1,2) WITH DRIFT
## 6         6 <fit[+]> SEASONAL DECOMP: ETS(M,A,N)
## 7         7 <fit[+]> PROPHET
## 8         8 <fit[+]> NNAR(1,1,10)[12]
```

Gerando as previsões

Dado que temos os modelos treinados

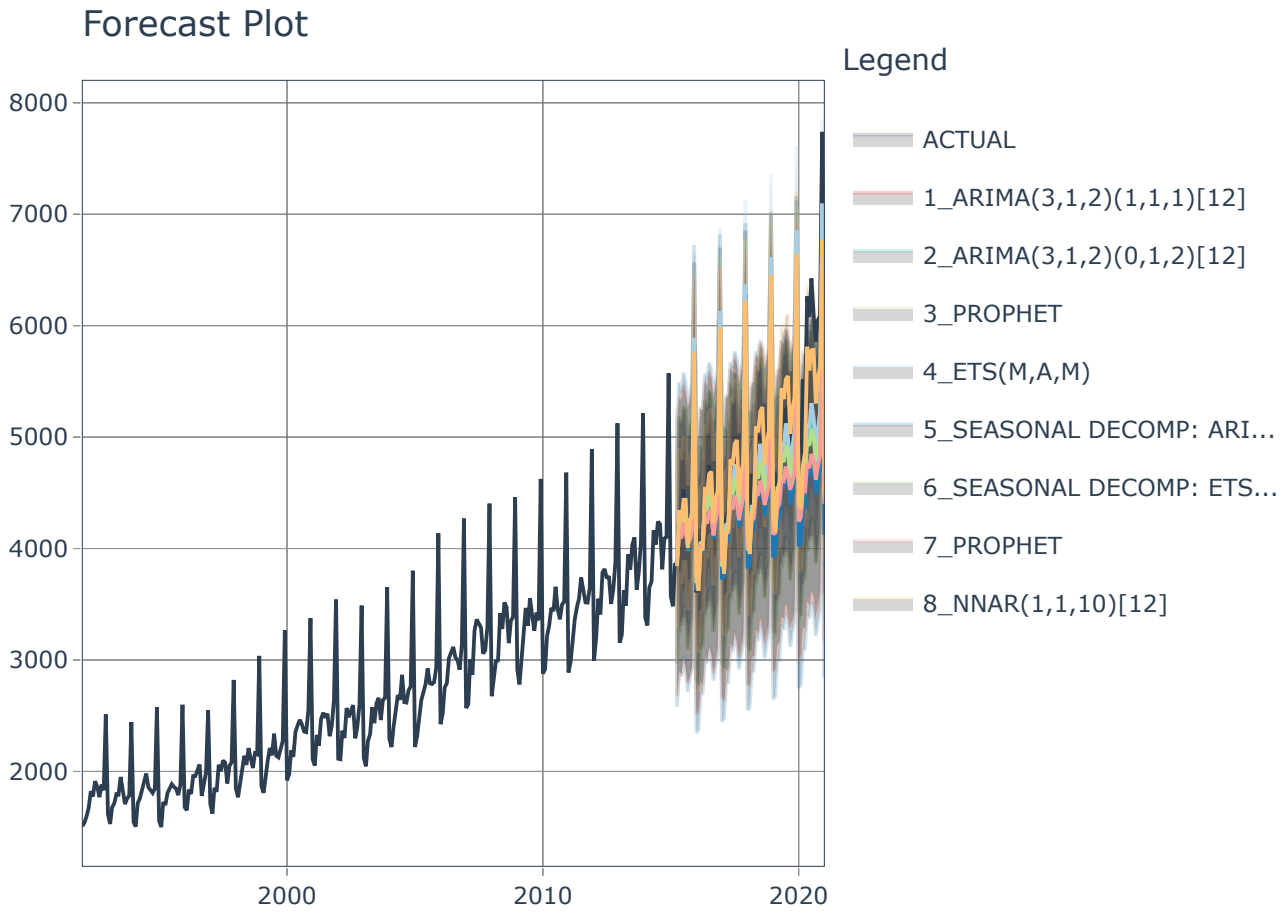
```
previsoes <- tabela_de_modelos %>%
  modeltime_calibrate(new_data = testing(splits))
```

Validação dos modelos

Previsões *vs* Valores reais

Abaixo irei comparar graficamente as previsões geradas com os valores reais.

```
previsoes %>%
  modeltime_forecast(
    new_data    = testing(splits),
    actual_data = beer
  ) %>%
  plot_modeltime_forecast(
    .legend_max_width = 25, # For mobile screens
  )
```



Métricas de avaliação Não é muito recomendado a análise meramente visual, por isso é necessário uma análise mais objetiva, por isso cada modelos será analisado utilizando as seguintes métricas de avaliação:

1. **mae** (erro absoluto médio);
2. **mape** (erro médio absoluto percentual)
3. **mase** (Erro médio absoluto escalado)
4. **smape** (erro médio absoluto percentual simétrico)
5. **rmse** (raiz do erro médio quadrado)
6. **rsq** ou R^2 (R quadrado - coeficiente de determinação).

Com exceção do R^2 (que deve ser o maior possível), todas as outras métricas devem estar o mais próximo possível de 0 (zero) .

Os resultado da tabela abaixo mostram que os melhores modelos foram o 4, 8, 1 e 2.

```
previsoes %>%
  modeltime_accuracy() %>%
  table_modeltime_accuracy()
```

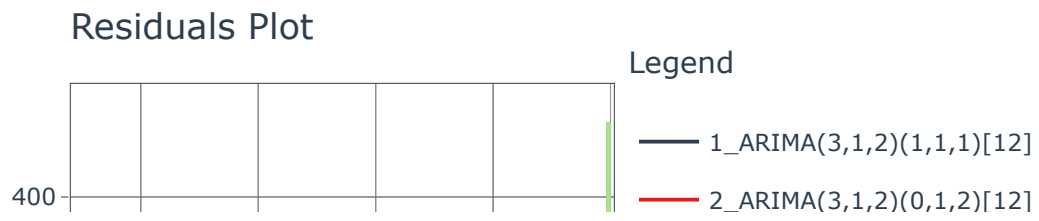
Search

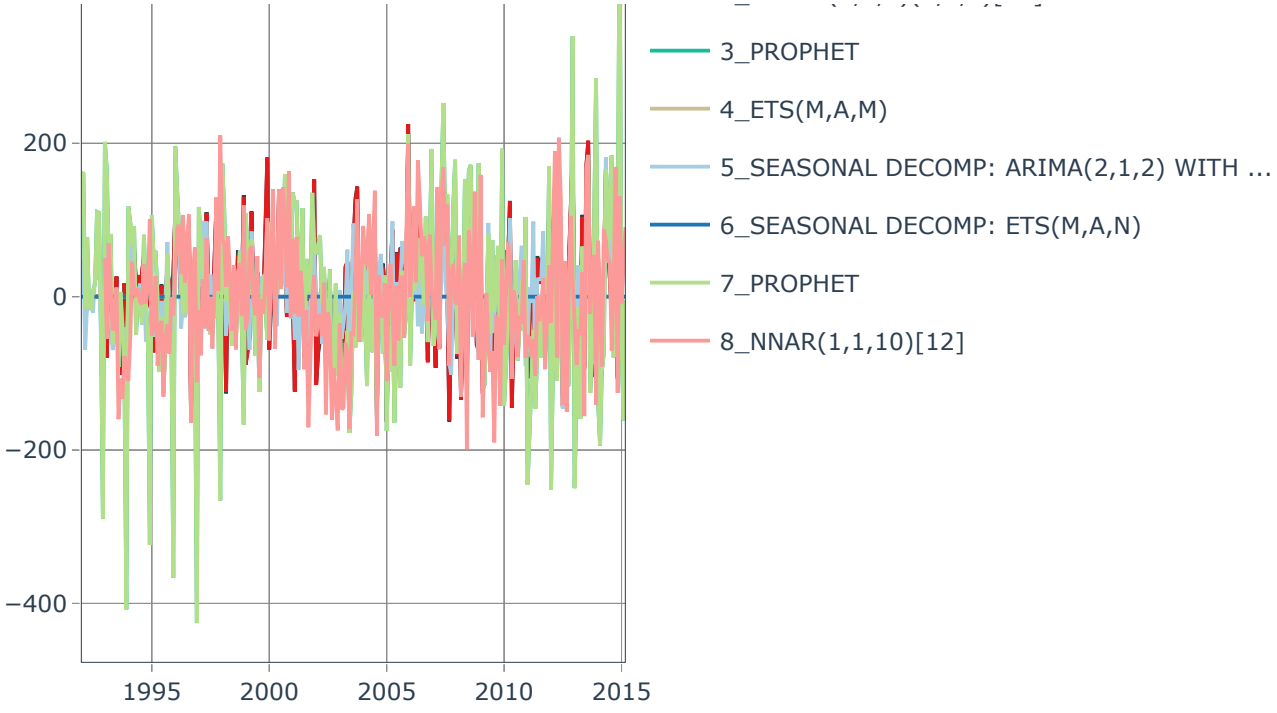
.model_id	.model_desc	.type	mae	mape	mase	smape
1	ARIMA(3,1,2)(1,1,1)[12]	Test	288.02	5.24	0.55	5.54
2	ARIMA(3,1,2)(0,1,2)[12]	Test	288.74	5.25	0.55	5.56
3	PROPHET	Test	450.92	8.25	0.85	8.86
4	ETS(M,A,M)	Test	229.01	4.18	0.43	4.4
5	SEASONAL DECOMP: ARIMA(2,1,2) WITH DRIFT	Test	477.61	8.87	0.91	9.55
6	SEASONAL DECOMP: ETS(M,A,N)	Test	320.55	5.79	0.61	6.15
7	PROPHET	Test	450.92	8.25	0.85	8.86
8	NNAR(1,1,10)[12]	Test	196.38	3.78	0.37	3.84

Análise dos resíduos de cada modelo na amostra

Aqui eu irei analisar o comportamento dos resíduos dos dados usados para treinar cada modelo. O ideal é que que o comportamento dos resíduos sejam bem comportados, ou seja, com uma distribuição normal (média em torno de zero e variância constante).

```
tabela_de_modelos %>%
  modeltime_calibrate(new_data = training(splits)) %>%
  modeltime_residuals() %>%
  plot_modeltime_residuals()
```

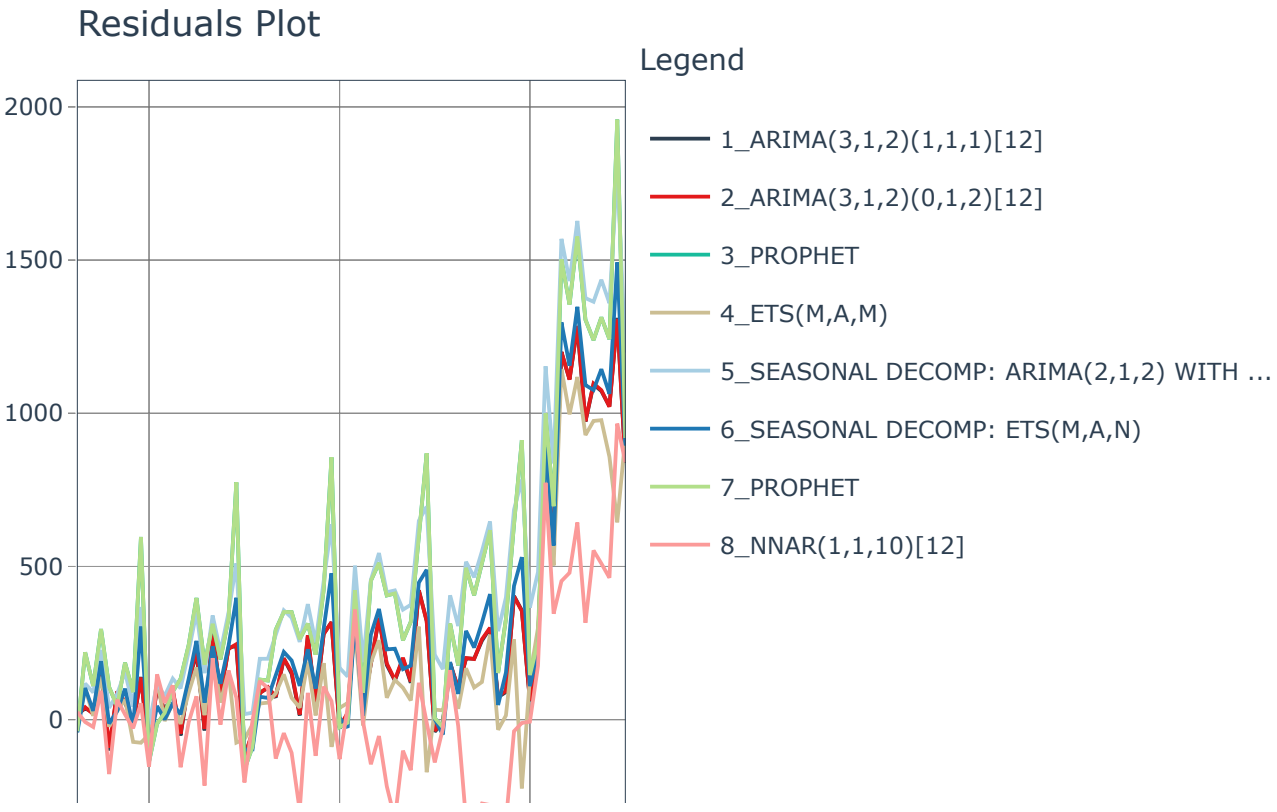




Análise dos resíduos de cada modelo fora da amostra

Aqui serão analisado os resíduos dos dados reais não usado na modelagem com as previsões geradas dos modelos.

```
tabela_de_modelos %>%
  modeltime_calibrate(new_data = testing(splits)) %>%
  modeltime_residuals() %>%
  plot_modeltime_residuals()
```



Teste nos resíduos de dentro e fora da amostra

O teste aplicado aos resíduos é o Teste *Shapiro-Wilk* que tem como hipóteses :

1. Hipótese Nula (H_0):A amostra provém de uma população normal;
2. Hipótese alternativa (H_1) : A amostra não provém de uma população normal.

São gerados outros dois teste de normalidade na saída desse comando, que são o *Box - Pierce* e o *Ljung-Box* que apresentam as mesmas hipóteses nulas:

1. Hipótese Nula (H_0): Os resíduos são *i.i.d.* (independentes e idênticamente distribuídos);
2. Hipótese alternativa (H_1) : Os resíduos não são *i.i.d.* (independentes e idênticamente distribuídos)

Também é gerado teste de *Durbin-Watson* que analisa se há autocorrelação serial nos resíduos. Basicamente, é ideal que o modelo gerado não tenha resíduos autocorrelacionados, sendo que pode existir autocorrelação positiva ou negativa. Para esse teste queremos que o seu resultado esteja o mais próximo possível do valor 2 (ausência de autocorrelação serial).

Para os dados de dentro da amostra tivemos como resultado que os modelos 1, 4, 5, 6 e 8 apresentaram comportamento normal (pelo teste *Shapiro-Wilk*). Os resultados dos testes *Box - Pierce* e o *Ljung-Box* desses modelos apresentaram resultados em que aceitamos a hipótese nula de *i.i.d.* dos resíduos.

E, por fim, todos eles apresentaram valores do teste *Durbin-Watson* próximos de 2, evidenciando ausência de autocorrelação seria (desse o mais próximo do valor 2 foi o modelo 1).

```
#para os dados de treino (dentro da amostra)
tabela_de_modelos %>%
  modeltime_calibrate(new_data = training(splits)) %>%
  modeltime_residuals() %>%
  modeltime_residuals_test()
```

```
## # A tibble: 8 x 6
##   .model_id .model_desc          shapiro_wilk box_pierce ljung_box durbin_watson
##       <int> <chr>                       <dbl>      <dbl>     <dbl>         <dbl>
## 1         1 ARIMA(3,1,2)(1,1,1)~   0.0594         0.964     0.964          1.99
## 2         2 ARIMA(3,1,2)(0,1,2)~   0.0487         0.977     0.977          1.99
## 3         3 PROPHET                0.00000515     0.328     0.325          1.88
## 4         4 ETS(M,A,M)             0.265          0.535     0.533          2.07
## 5         5 SEASONAL DECOMP: AR~   0.0941         0.990     0.990          1.98
## 6         6 SEASONAL DECOMP: ET~   0.110          0.0148    0.0143         2.28
## 7         7 PROPHET                0.00000515     0.328     0.325          1.88
## 8         8 NNAR(1,1,10)[12]       0.439          0.156     0.154          1.82
```

Agora para os resíduos de fora da amostra todos os resultados dos testes foram insatisfatórios. Uma coisa que pode ser percebida nessa série é o impacto da pandemia do coronavírus e isso provavelmente refletiu na aqui nesses resultados.

```
#para os dados de teste (fora da amostra)
tabela_de_modelos %>%
  modeltime_calibrate(new_data = testing(splits)) %>%
  modeltime_residuals() %>%
  modeltime_residuals_test()
```

```
## # A tibble: 8 x 6
##   .model_id .model_desc          shapiro_wilk box_pierce ljung_box durbin_watson
##       <int> <chr>                       <dbl>      <dbl>     <dbl>         <dbl>
## 1         1 ARIMA(3,1,2)(1,1,1)~     4.21e-9   2.24e-11  8.31e-12         0.228
## 2         2 ARIMA(3,1,2)(0,1,2)~     3.81e-9   2.04e-11  7.51e-12         0.224
## 3         3 PROPHET                  4.01e-6   2.98e- 8  1.50e- 8         0.330
## 4         4 ETS(M,A,M)               1.52e-9   6.42e-11  2.48e-11         0.261
## 5         5 SEASONAL DECOMP: AR~     8.54e-8   6.68e-12  2.35e-12         0.139
## 6         6 SEASONAL DECOMP: ET~     1.89e-8   9.81e-11  3.87e-11         0.246
## 7         7 PROPHET                  4.01e-6   2.98e- 8  1.50e- 8         0.330
## 8         8 NNAR(1,1,10)[12]         5.41e-5   3.59e- 8  1.82e- 8         0.548
```

Retreino do modelo

Irei pegar todos os modelos e irei trená-los novamente com toda a série temporal.

```
retreino <- previsoes %>%
  modeltime_refit(data = beer)
```

```
## frequency = 12 observations per 1 year
## frequency = 12 observations per 1 year
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
## frequency = 12 observations per 1 year
## frequency = 12 observations per 1 year
## frequency = 12 observations per 1 year
```

```
## Disabling weekly seasonality. Run prophet with weekly.seasonality=TRUE to override this.
```

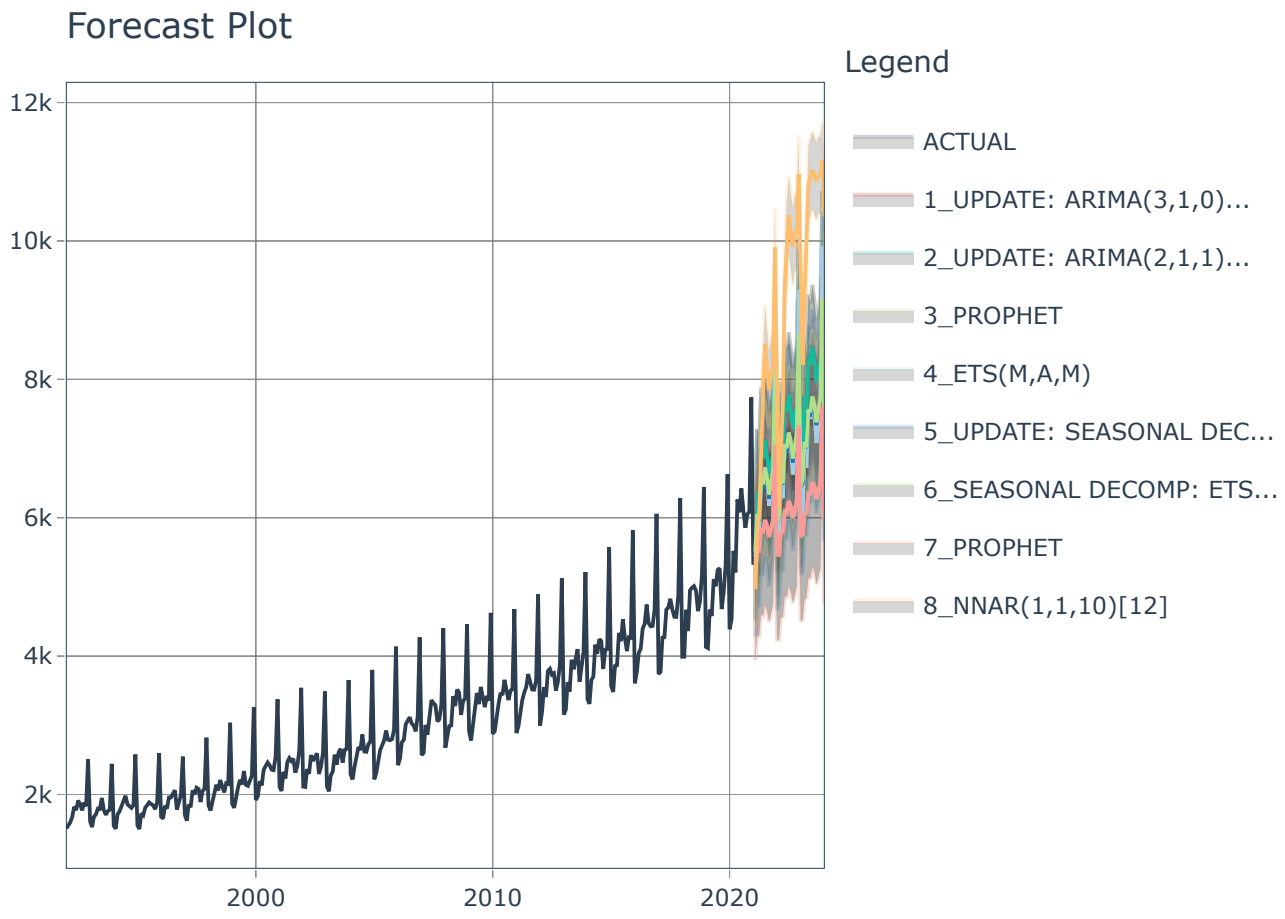
```
## Disabling daily seasonality. Run prophet with daily.seasonality=TRUE to override this.
```

```
## frequency = 12 observations per 1 year
```

Previsão com os modelos retreinados

Agora a última etapa desse projeto é gerar previsões para um horizonte de tempo desconhecido, usando os modelos para gerar cenários futuros de como a nossa variável irá comportar-se.

```
retreino %>%
  modeltime_forecast(h = "3 years", actual_data = beer) %>%
  plot_modeltime_forecast(
    .legend_max_width = 25, # For mobile screens
  )
```



Conclusão

Nesse projeto realizei uma pequena exploração desse pacote que mostrou-se muito útil para modelagem de séries temporais. Infelizmente os modelos treinado não obtiveram um bom desempenho quando realizada a etapa de validação, entretanto vale salientar (conforme gráfico dos resíduos) que a maior proporção da diferença entre estimado e realizado foi no ano de 2020, ano que ocorreu a pandemia do coronavírus. Em seguida, em outros projetos, utilizarei outros módulos desse pacote, que utiliza modelos de *machine learning* e *redes neurais*.