Eddie Aguilar

Quiz - Forecasting and beyond

Try to answer each question or exercise by yourself and without any external help. If neded, you can check your class notes, exercises, documentation online, or even your peers.

Let's see what you've learned regarding the forecasting workflow so far. Answer the following questions to the best of your abilities.

What are the type of components that a time series can exhibit?

✓ A residual component.

✓ A trend-cycle component.

✗ A fluctuating component.

✗ Autocorrelation component.

✓ Seasonal component(s).

Correct!

What is the purpose of performing calendar, population, or inflation adjustments to a time series?

✗ To remove the trend from it.

✓ Reduce/remove variation that is not relevant to the forecast/analysis done.

✗ Correct possible heteroscedasticity in the ts.

✗ Make the time series look prettier.

Correct!

R Code Start Over ▶ Run Code

```
# This is here just in case you need to run any R code.


```

Next Topic

Forecasting - Prerequisites

Let's do some exercises and practice what we've seen so far. We will download some data and go through all the forecasting workflow. To start, load the necessary packages into the environment.

R Code Start Over Hint Run Code

```
library(tidyverse)
library(fpp3)

```

✓ Importing data

To simplify the data importing step, we will import a time series directly from the FRED website using the `tq_get()` function from the **tidyquant** package. In this case we will work with the Advance Retail Sales, specifically sales from Food and Beverages stores (`RSDBSN`). Go ahead and load it and save it to a variable called `ex1_tbl`. Some code is already provided for you.

R Code Start Over Hints Run Code

```
ex1_tbl <- tidyquant::tq_get(
  x    = "RSDBSN",
  get  = "economic.data",
  from = "1992-01-01"
)
```

✓ Wrangling the data

Inspect the data frame you just downloaded using function `class()`.

R Code Start Over Hint Run Code

```
class(ex1_tbl)


```

```
[1] "tbl_df"     "tbl"        "data.frame"
```

You should see something like:

```
## [1] "tbl_df"     "tbl"        "data.frame"
```

The first argument `tbl_df` indicates we have a `tibble`. We must convert it to a `tsibble`. But first, we need to make sure our **date** variable is in the correct format for your time series. Change it if needed and overwrite `ex1_tbl`, otherwise leave it as is.

R Code Start Over Hint Run Code

```

ex1_tbl <- ex1_tbl |>
  as_tsibble(index = date)
  ex1_tbl

```

symbol	**date**	**price**
<chr>	<date>	<int>
RSDBSN	1995-05-01	33178
RSDBSN	1995-06-01	33060
RSDBSN	1995-07-01	33520

Now you can convert your `tibble` into a `tsibble`. Do so using `as_tsibble()` and store your result now in a variable named `ex1_tsbl`.

Any given `tsibble` needs an *index*, which is the date variable. When do you need to specify the `key` argument for the `tsibble` to work?

✓ When you data frame is in a *long* format, and you have **more than one time series** in the data frame.

X Always.

Correct!

R Code Start Over Hint Run Code

```r
ex1_tsbl <- ex1_tbl |>
  as_tsibble(index = date)
  ex1_tbl

```

symbol <chr>	date <mth>	price <int>
RSDBSN	1992 Jan	29589
RSDBSN	1992 Feb	28570
RSDBSN	1992 Mar	29682
RSDBSN	1992 Apr	30228
RSDBSN	1992 May	31677
RSDBSN	1992 Jun	30769
RSDBSN	1992 Jul	32402
RSDBSN	1992 Aug	31469
RSDBSN	1992 Sep	30162
RSDBSN	1992 Oct	31407

1-10 of 375 rows Previous **1** 2 3 4 5 6 ... 38 Next

Check now the class of this new variable `ex1_tsbl`:

R Code Start Over Solution Run Code

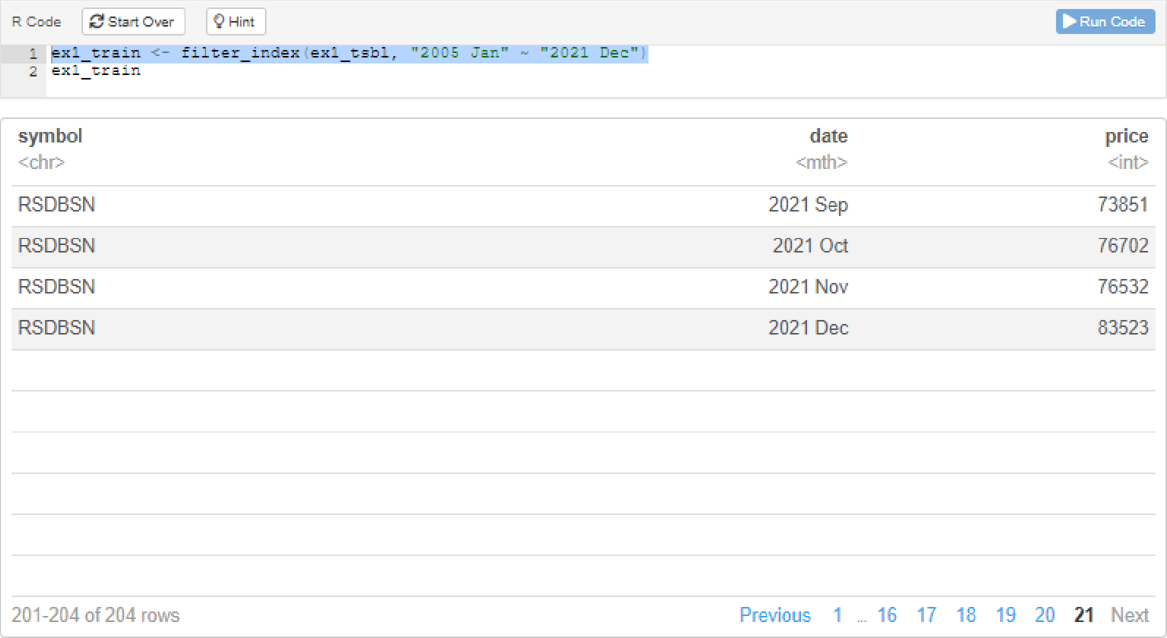
```r
class(ex1_tsbl)

```

```
[1] "tbl_ts"     "tbl_df"     "tbl"        "data.frame"
```

It should now display as its first argument `tbl_ts`, which indicates the object is a `tsibble`.

✓ Train/test splits

Split your data set so that you have the last **2 years** as test data. Save your training set to a variable called `ex1_train`.



```r
ex1_train <- filter_index(ex1_tsbl, "2005 Jan" ~ "2021 Dec")
ex1_train
```

symbol <chr>	date <mth>	price <int>
RSDBSN	2021 Sep	73851
RSDBSN	2021 Oct	76702
RSDBSN	2021 Nov	76532
RSDBSN	2021 Dec	83523

201-204 of 204 rows

Exploratory Data Analysis

✓ Visualization

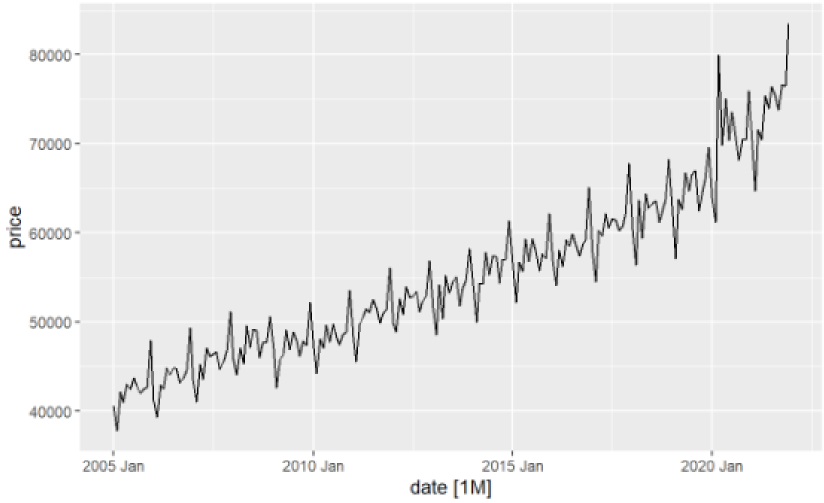
Time plot

Plot your training set using `autoplot()`.

R Code   Start Over   ▶ Run Code

```r
ex1_train <- filter_index(ex1_tsbl, "2005 Jan" ~ "2021 Dec")
autoplot(ex1_train)

```

```
Plot variable not specified, automatically selected `.vars = price`
```



Which patterns do you see on it?

Seasonality

✓ Yes

✗ No

Correct!

Trend

✓ Yes

✗ No

Correct!

Constant variance (Homoscedasticity)

✗ Yes

✓ No

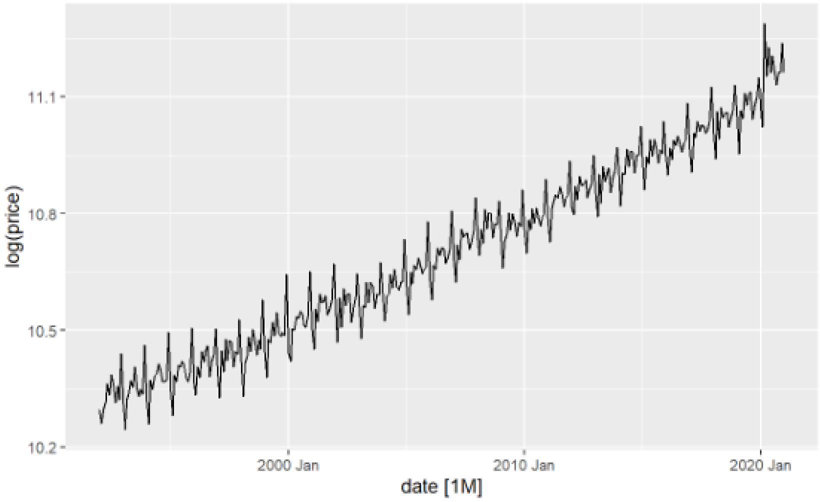
Correct!

Mathematical Transformations

It seems that we may need a transformation to stabilize the variance of the ts. Try using a log transformation and plot it to verify if it works:

R Code Start Over ▶ Run Code

```r
autoplot(ex1_train, log(price))


```



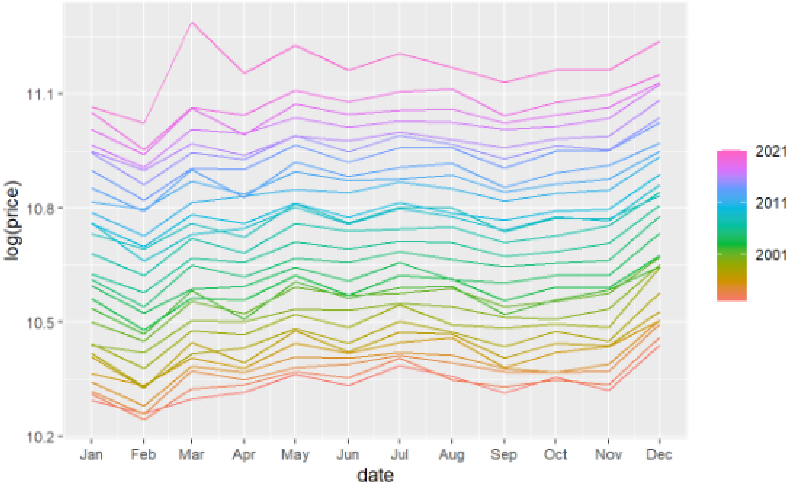
If you decided a transformation was in order, remember to use it from here on over.

Season plot

Now let's check a **season plot**, to get a closer look about the seasonality.

R Code Start Over Hint Run Code

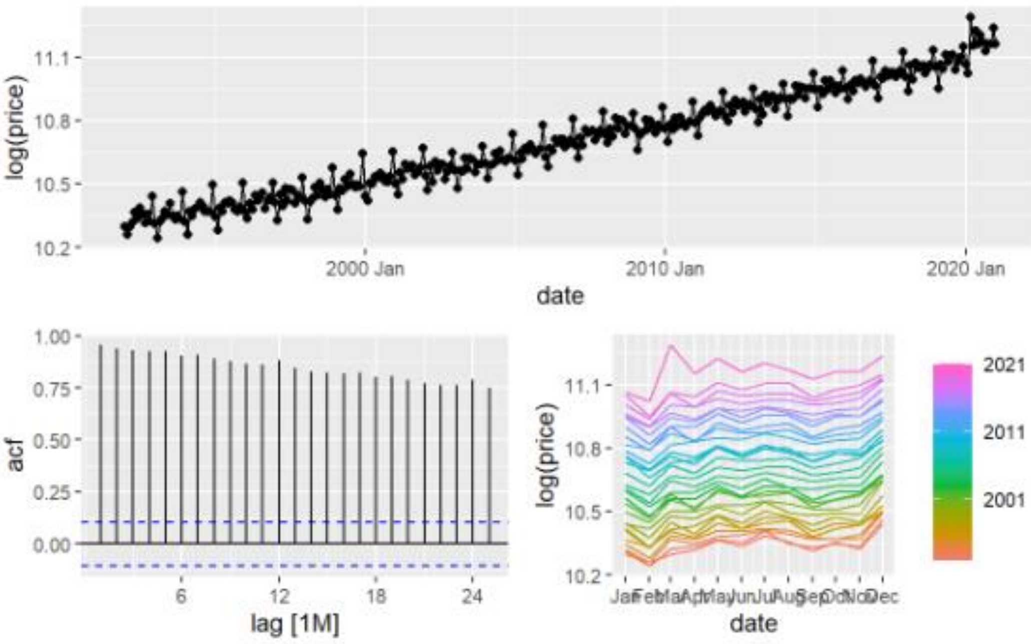
```
gg_season(ex1_train, log(price))


```



There are many other types of plots that could aid you in getting more insights from the data. Feel free to try them here:

R Code Start Over Run Code

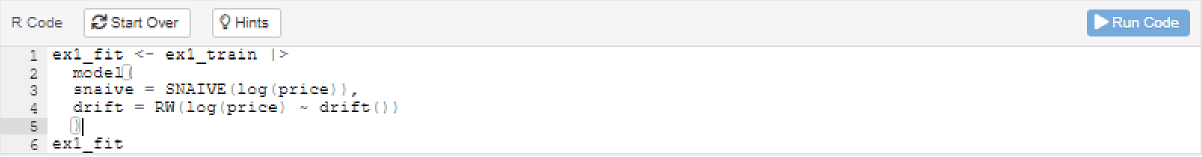
```

gg_tsdisplay(ex1_train, log(price))

```


log(price)
11.1
10.8
10.5
10.2
2000 Jan
2010 Jan
2020 Jan
date
acf
1.00
0.75
0.50
0.25
0.00
6
12
18
24
lag [1M]
log(price)
11.1
10.8
10.5
10.2
Jan
Feb
Mar
Apr
May
Jun
Jul
Aug
Sep
Oct
Nov
Dec
date
2021
2011
2001

Modeling the time series

So far, we only know the basic benchmark models (*Mean, Naïve, Seasonal Naïve, and Drift* methods). Based upon what you've discovered from this time series so far, create a `mable` (table of models) estimating those which you consider will fit better to the data. Store the `mable` in `ex1_fit`.

If you chose to use a transformation, remember to apply it inside the model specification, so that R can detect it.



```r
ex1_fit <- ex1_train |>
  model(
  snaive = SNAIVE(log(price)),
  drift = RW(log(price) ~ drift())
  )
ex1_fit
```

symbol <chr>	snaive <lst_mdl>	drift <lst_mdl>
RSDBSN	<lst_mdl>	<lst_mdl>

1 row

✓ Model evaluation

How can you tell if your model(s) fit the data correctly? We can do so in several ways:

- Checking the residual diagnostics
  - Visually
  - Using statistical tests, such as the Ljung-Box test for autocorrelation.
- Verify the error metrics on the training set.

Quiz

We expect the residuals of the model to be entirely white noise. A variable can be considered white noise if:

✗ Its variance is 1.

✓ It has no seasonality

✓ It has no significant autocorrelation.

✓ It exhibits no trend.

Correct!

When referring to the residuals of a model, besides what it was said on the previous question, we also check that:

✓ The mean is close to zero.

✓ They are normally distributed.

✓ They present homoscedasticity (time-invariant variance).

✗ The variance is close to one.

Correct!

✓ Residual diagnostics

Do the residual diagnostics with the help of the `gg_tsresiduals()` function.

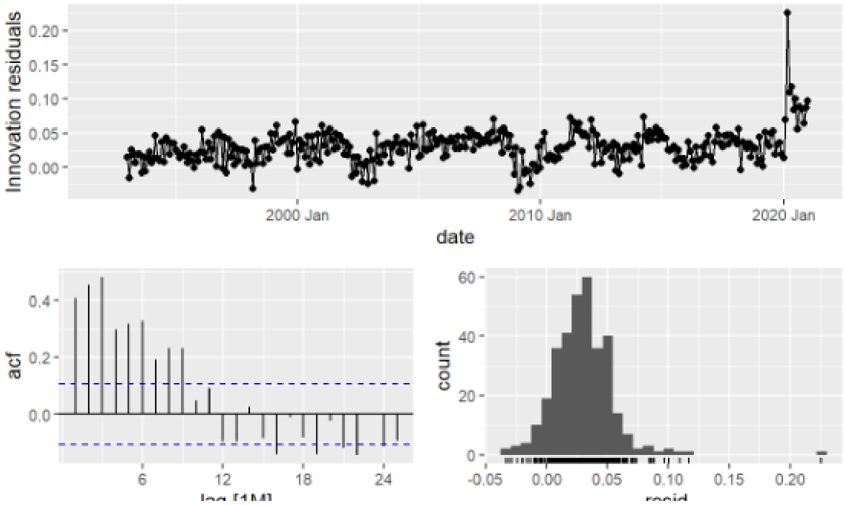
R Code | Start Over | Hint | Run Code

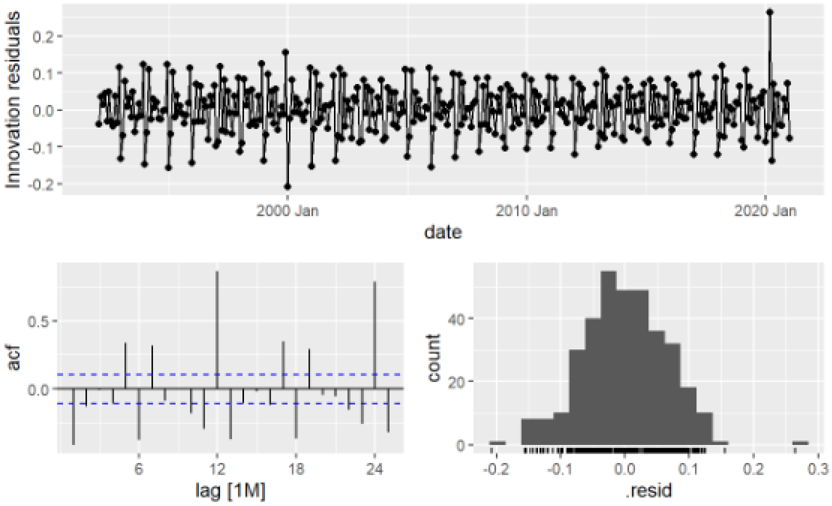
```
ex1_fit <- ex1_train |>
  model(
  snaive = SNAIVE(log(price)),
  drift = RW(log(price) ~ drift())
  )

gg_tsresiduals(select(ex1_fit, snaive))
gg_tsresiduals(select(ex1_fit, drift))

```

```
Warning: Removed 12 rows containing missing values (`geom_line()`).
```

```
Warning: Removed 12 rows containing missing values (`geom_point()`).
```

```
Warning: Removed 12 rows containing non-finite values (`stat_bin()`).
```





✓ Portmanteau tests

To further assess whether the model(s) provided are producing white noise residuals, test it out using Ljun-Box.

R Code · Start Over · Hints · Run Code

```
ex1_fit |>
  augment() |>
  features(.resid, ljung_box, lag = 24, dof = 0)
```

symbol <chr>	.model <chr>	lb_stat <dbl>	lb_pvalue <dbl>
RSDBSN	drift	885.0620	0
RSDBSN	seasonal_naive	630.6089	0

2 rows

✓ Training Accuracy

Use the `accuracy()` function together with the `mable` to see the different error metrics calculated on the training set.

R Code · Start Over · Hints · Run Code

```
accuracy(ex1_fit)
```

symbol <chr>	.model <chr>	.type <chr>	ME <dbl>	RMSE <dbl>
RSDBSN	seasonal_naive	Training	1441.94362	2082.457
RSDBSN	drift	Training	3.69474	3053.115

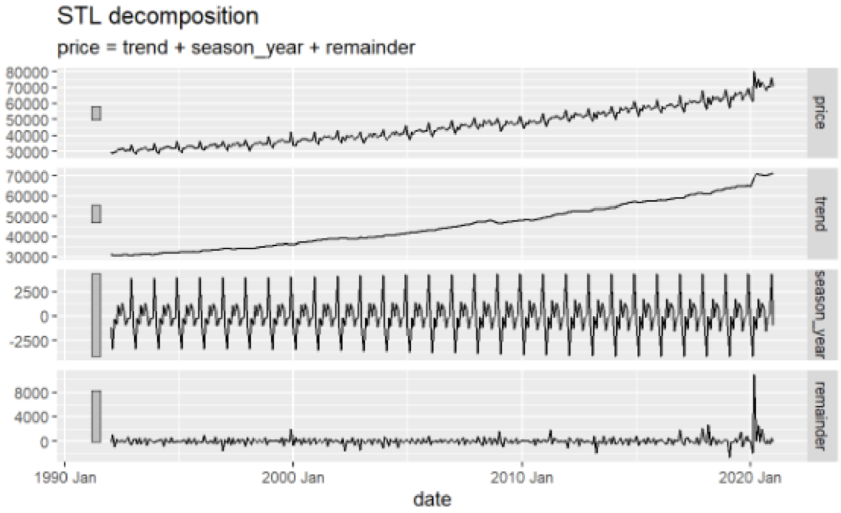
2 rows | 1-5 of 11 columns

✓ Model with decomposition

Let's add another model to our analysis, using a STL decomposition and model each component independently. Save this new model in `ex1_fit_dcmp`:

R Code Start Over Hints ▶ Run Code

```r
ex1_fit_dcmp <- ex1_train %>%
  model(STL(price ~ trend(window = 7) + season(window=21), robust=TRUE)) %>%
  components()

autoplot(ex1_fit_dcmp)


```



Now join this new model with our original `mable` using `left_join()`:

R Code Start Over ▶ Run Code

```r
(ex1_fit <- left_join(x = ex1_fit, y = ex1_fit_dcmp))



```

```
Joining with `by = join_by(symbol)`
```

symbol	seasonal_naive	drift	stlf
<chr>	<lst_mdl>	<lst_mdl>	<lst_mdl>
RSDBSN	<lst_mdl>	<lst_mdl>	<lst_mdl>

1 row

Previous Topic Next Topic

Forecasting on the test set

✓ Getting the `fable`

Produce forecasts for the test set. Store the `fable` (*forecast table*) in `ex1_fc`.

R Code   Start Over   Hints   ▶ Run Code

```r
(ex1_fc <- forecast(ex1_fit, h = 16))
```

symbol <chr>	.model <chr>	date <mth>	price <dist>	.mean <dbl>
RSDBSN	stlf	2021 Oct	<dist>	73260.16
RSDBSN	stlf	2021 Nov	<dist>	73997.00
RSDBSN	stlf	2021 Dec	<dist>	79738.74
RSDBSN	stlf	2022 Jan	<dist>	72788.18
RSDBSN	stlf	2022 Feb	<dist>	67893.19
RSDBSN	stlf	2022 Mar	<dist>	74677.21
RSDBSN	stlf	2022 Apr	<dist>	72818.65
RSDBSN	stlf	2022 May	<dist>	77448.55

41-48 of 48 rows   Previous 1 2 3 4 **5** Next

✓ Forecast plots

Plot the forecasts produced by each model, along with the training and test set. **Tip:** you'll need the `fable`, and the full `tsibble`.

R Code   Start Over   Solution   ▶ Run Code

```r
ex1_fc |>
  autoplot(filter_index(ex1_tsbl, "2005" ~ .), level = FALSE) +
  ggtitle("Forecast")
```

```
Warning: There was 1 warning in `filter()`.
ℹ In argument: `time_in(date, ...)`.
Caused by warning:
! `yearmonth()` may yield unexpected results.
ℹ Please use arg `format` to supply formats.
```

```
Warning: Plot argument `level` should be a numeric vector of levels to display.
Setting `level = NULL` will remove the intervals from the plot.
```

✓ Forecast plots

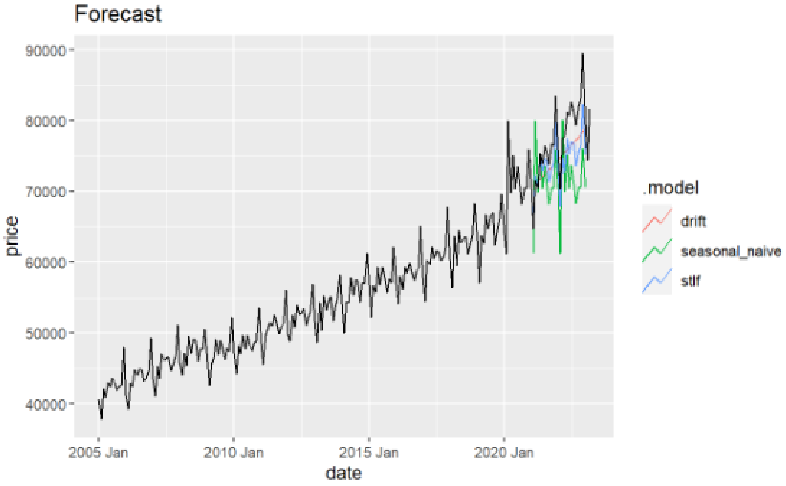
Plot the forecasts produced by each model, along with the training and test set. **Tip:** you'll need the `fable`, and the full `tsibble`.

R Code | Start Over | Solution | ▶ Run Code

```
ex1_fc |>
  autoplot(filter_index(ex1_tsbl, "2005" ~ .), level = FALSE) +
  ggtitle("Forecast")


```

```
Warning: There was 1 warning in `filter()`.
ℹ In argument: `time_in(date, ...)`.
Caused by warning:
! `yearmonth()` may yield unexpected results.
ℹ Please use arg `format` to supply formats.
```

```
Warning: Plot argument `level` should be a numeric vector of levels to display.
Setting `level = NULL` will remove the intervals from the plot.
```



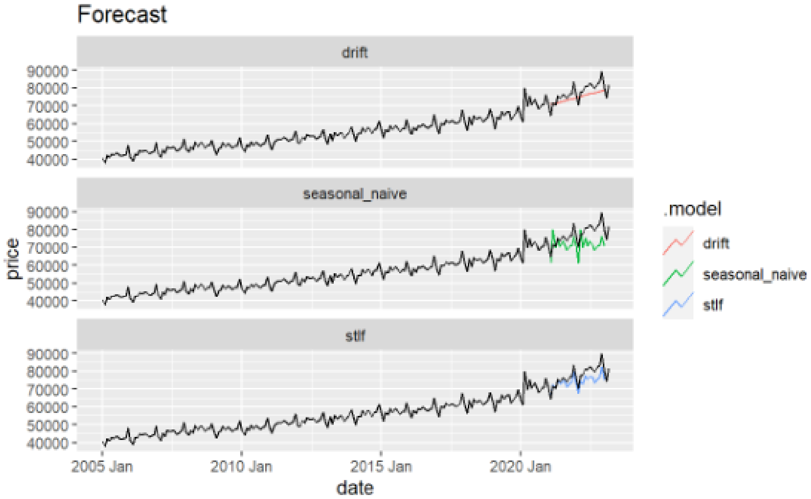
Try splitting each forecast in its own subplot, defining free scales for the y axis:

R Code | Start Over | Hints | ▶ Run Code

```
ex1_fc |>
  autoplot(filter_index(ex1_tsbl, "2005" ~ .), level = FALSE) +
  ggtitle("Forecast") +
  facet_wrap(~ .model, scales = "free_y", ncol = 1)
```

```
Warning: There was 1 warning in `filter()`.
ℹ In argument: `time_in(date, ...)`.
Caused by warning:
! `yearmonth()` may yield unexpected results.
ℹ Please use arg `format` to supply formats.
```

```
Warning: Plot argument `level` should be a numeric vector of levels to display.
Setting `level = NULL` will remove the intervals from the plot.
```



✓ Forecasting errors

Estimate the forecast errors on the test set using the `accuracy()` function. You now need to provide the `fable`, and the full `tsibble`.

R Code | Start Over | Hints | Run Code

```
ex1_fc |>
  accuracy(ex1_tsbl)

```

.model <chr>	symbol <chr>	.type <chr>	ME <dbl>	RMSE <dbl>	MAE <dbl>
drift	RSDBSN	Test	2796.643	4543.413	3751.480
seasonal_naive	RSDBSN	Test	6034.057	7817.156	6953.326
stlf	RSDBSN	Test	3249.102	3967.249	3394.971

3 rows | 1-6 of 11 columns

Previous Topic   Next Topic

Forecasting the future

✓ Refitting the best model

It seems that the model that produces the best forecasts is the one with decomposition (`stlf`). Refit this model using the whole data (not just the training set). Store your newly fit model into `ex1_fit_fut`

R Code Start Over Hint Run Code

```r


ex1_fit_fut <-   model(ex1_tsbl, STL(price ~ trend(window = 7) + season(window=21), robust=TRUE)) %>%
  components()


```

✓ Forecast the future

Produce a 2-year forecast into the future. Save it to `ex1_fc_fut`

R Code Start Over Run Code

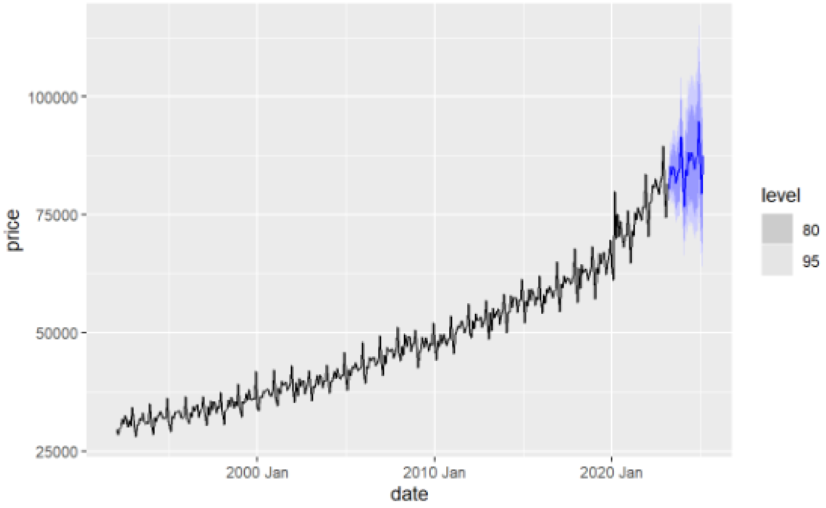
```r
ex1_fc_fut <- ex1_fit_fut |>
  forecast(h = "2 years")

```

✓ Forecast plot

Plot the final forecast.

R Code Start Over Solution Run Code

```r
autoplot(ex1_fc_fut, ex1_tsbl)


```


price
2000 Jan
2010 Jan
2020 Jan
date
level