Chapter 7 Review

Anthony Tricarico

Table of contents

The main topic of this chapter is regression models. In particular, we start the discussion with simple linear regression models and how those are useful in estimating the relationship between two variables: a *dependent variable* (aka *forecast variable*) and an *independent variable* (also called *predictor*). We expand the discussion to cover multiple linear regression which allows us to include more than one predictor in our model and allows us to model more complex real-world phenomena which might depend on more than one independent variable.

> **Tip**
>
> Make sure that you go over all the previous chapters as in this one we will only be discussing the new concepts and it will be very likely that the functions encountered in previous chapters will not be discussed in-depth anymore. Whenever lines of code are skipped you can refer back to the original version of the script (`chap7.R`). Also, whenever you have doubts, have a look at the links that are scattered throughout the review. There you can find different explanations with simpler examples that might help you.

Let's load the `fpp3` package before we do anything else. This ensures that we will have (most) of the functions we will need throughout the chapter.

```
library(fpp3)
```

```
Registered S3 method overwritten by 'tsibble':
  method               from
  as_tibble.grouped_df dplyr
```

```
-- Attaching packages --------------------------------------- fpp3 1.0.1 --
```

```
v tibble      3.2.1     v tsibble     1.1.5
v dplyr       1.1.4     v tsibbledata 0.4.1
v tidyr       1.3.1     v feasts      0.4.1
v lubridate   1.9.3     v fable       0.4.1
v ggplot2     3.5.1

-- Conflicts ------------------------------------------------ fpp3_conflicts --
x lubridate::date()    masks base::date()
x dplyr::filter()      masks stats::filter()
x tsibble::intersect() masks base::intersect()
x tsibble::interval()  masks lubridate::interval()
x dplyr::lag()         masks stats::lag()
x tsibble::setdiff()   masks base::setdiff()
x tsibble::union()     masks base::union()
```

1 Time series linear regression

To estimate the relationship between two (or more) time series we use regression methods.

1.1 Simple linear regression

In a simple linear regression you have exactly one prediction variable and one predictor. The equation that determines the form of such regression models is of the following type:

$$y_t = \beta_0 + \beta_1 * x_t + \epsilon_t$$

where β_0 is the intercept and β_1 is the coefficient associated to our predictor variable x. Since this can be viewed as the equation of a line the β_1 coefficient is also referred to as the *slope* of the regression line. The ϵ refers to the *error* which captures the variation in y_t that x_t does not explain.

The code to be used to set up a linear model in R was already encountered in chapter 5. We review it here:

```
# A mable: 1 x 1
  `TSLM(Consumption ~ Income)`
                        <model>
1                        <TSLM>
```

1. use `TSLM()` inside the `model()` function whenever you want to set up a time series linear model (spoiler: you'll be doing a lot of that so remember the syntax)

Here, `Consumption` is our prediction variable y and the only predictor used x_1 is `Income`. We can wrap our fitted model inside the report() function to get a summary of the hypothesis tests on the estimated coefficients $\hat{\beta}_0$ and $\hat{\beta}_1$ in our model and of the overall significance of the model.

> **!** Important
>
> The $\hat{\beta}_1$ notation means that we are now referring to the estimated coefficient and not to the true (unknown) coefficient.

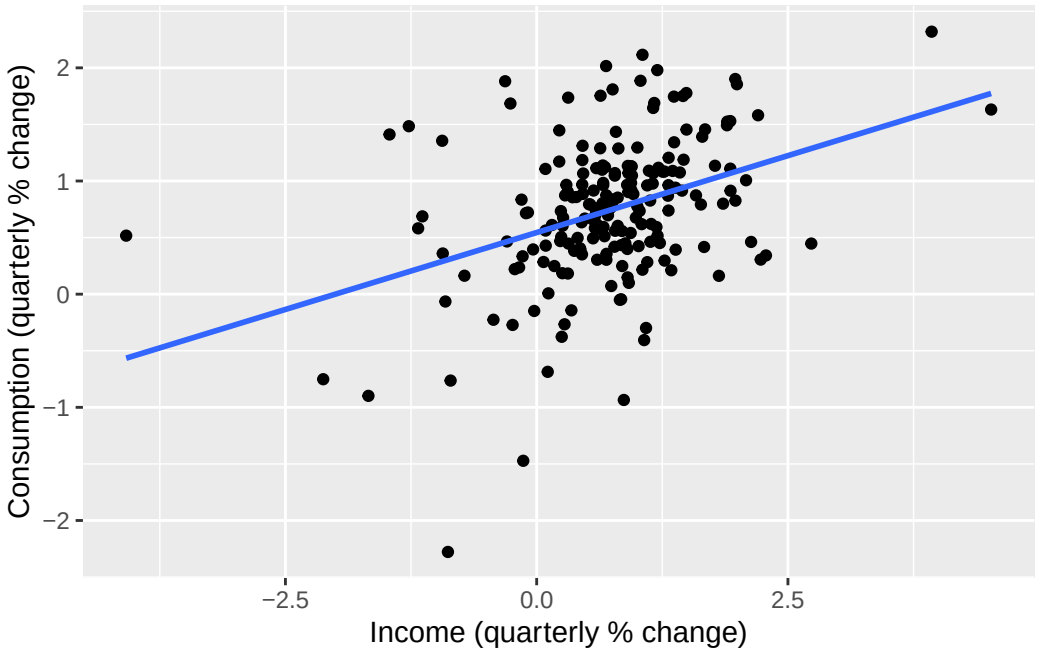
```
report(model(us_change, TSLM(Consumption ~ Income)))
```

```
Series: Consumption
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-2.58236 -0.27777  0.01862  0.32330  1.42229

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.54454    0.05403  10.079  < 2e-16 ***
Income       0.27183    0.04673   5.817  2.4e-08 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.5905 on 196 degrees of freedom
Multiple R-squared: 0.1472, Adjusted R-squared: 0.1429
F-statistic: 33.84 on 1 and 196 DF, p-value: 2.4022e-08
```

You can check the textbook (chapter 7.1) to review how to interpret the coefficients. We can also add the regression line to the scatterplot by adding the `geom_smooth()` layer to a ggplot object and setting the method to `"lm"` (i.e., linear model) and specifying that we do not want the standard error (`se`) bars to show (`se = FALSE`)

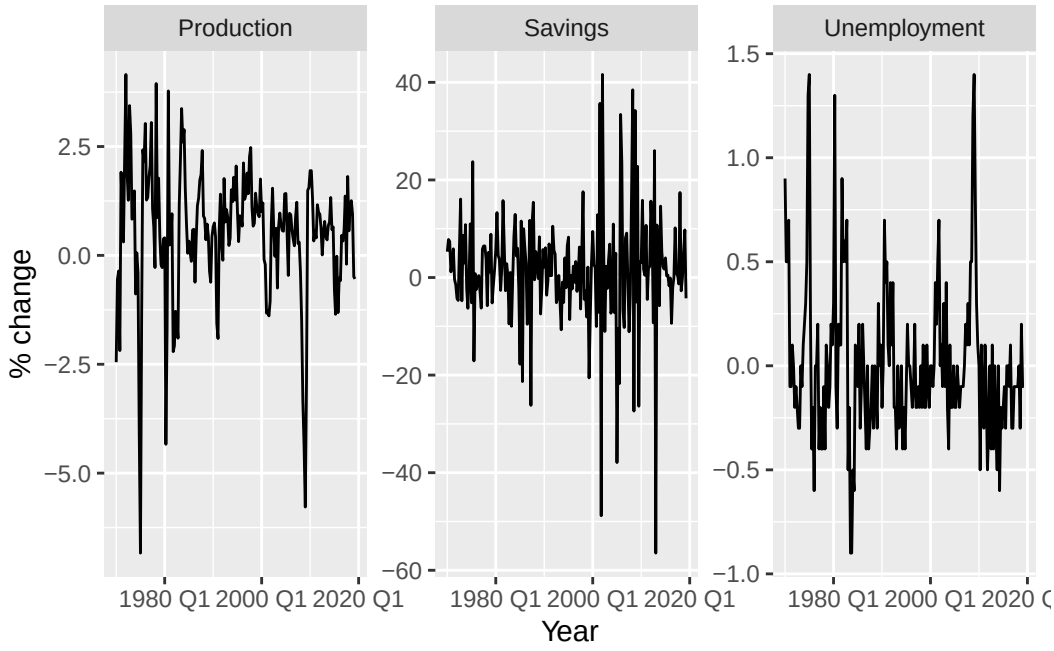
```
ggplot(us_change, aes(x = Income, y = Consumption)) +
  labs(y = "Consumption (quarterly % change)",
       x = "Income (quarterly % change)") +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```

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



Also note that in the following call to `autoplot` the `vars` function is used to put together different variables and plot them all together.

```
autoplot(us_change,vars(Production,Savings,Unemployment)) +
  ylab("% change") +
  xlab("Year")
```



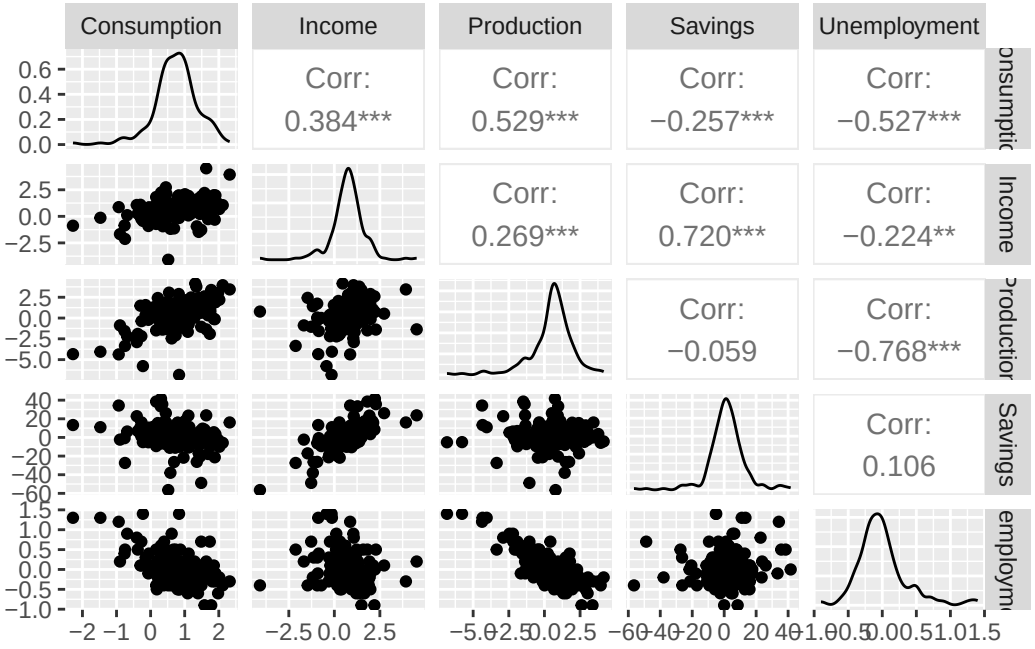
1.2 Multiple linear regression

When we want to model more complex phenomena, we resort to using multiple linear regression to specify that the prediction variable y depends on more than one predictor (i.e., different x's).

Before specifying the model, we might want to check the correlation between the different variables. Specifically, we want to see how each potential predictor x_i correlates to the prediction variable y. With `ggpairs()` from the `GGally` package we can do that while simultaneously checking also the distribution of the different variables.

```
GGally::ggpairs(us_change, columns = 2:6)
```

```
Registered S3 method overwritten by 'GGally':
  method from
  +.gg   ggplot2
```



Then we can fit the multiple regression model to the data. This is done simply by specifying other predictors inside the call to TSLM each separated by a + sign as shown below.

```
fit_consMR <- model(us_change,
tslm = TSLM(Consumption ~ Income + Production + Unemployment + Savings))
```

```
report(fit_consMR)
```

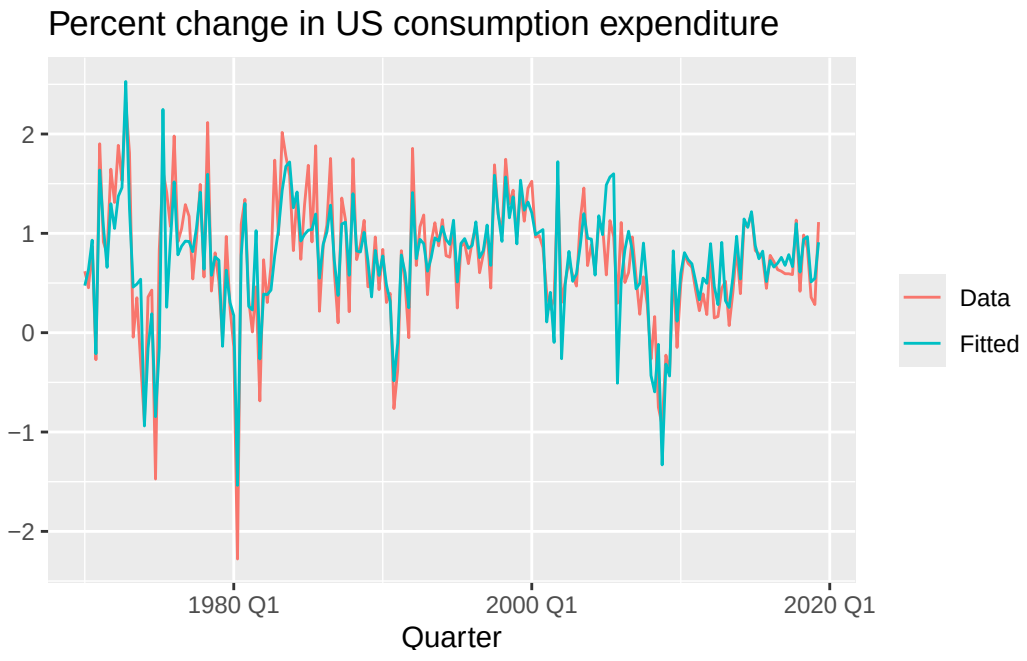
```
Series: Consumption
Model: TSLM

Residuals:
```

```
     Min       1Q   Median       3Q      Max
-0.90555 -0.15821 -0.03608  0.13618  1.15471

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept)   0.253105   0.034470   7.343 5.71e-12 ***
Income        0.740583   0.040115  18.461  < 2e-16 ***
Production    0.047173   0.023142   2.038   0.0429 *
Unemployment -0.174685   0.095511  -1.829   0.0689 .
Savings      -0.052890   0.002924 -18.088  < 2e-16 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 0.3102 on 193 degrees of freedom
Multiple R-squared: 0.7683, Adjusted R-squared: 0.7635
F-statistic:   160 on 4 and 193 DF, p-value: < 2.22e-16
```

Now also the report will include the different predictors along with their estimated coefficients (in the `Estimate` column). For the significance of each individual predictor you can refer to the last column which reports the p-value (smaller is better!).

Using augment() we can now retrieve the fitted values (i.e., what the model estimates for y_t at each time t) and compare it to actual data to visualize how well our model fits the data.

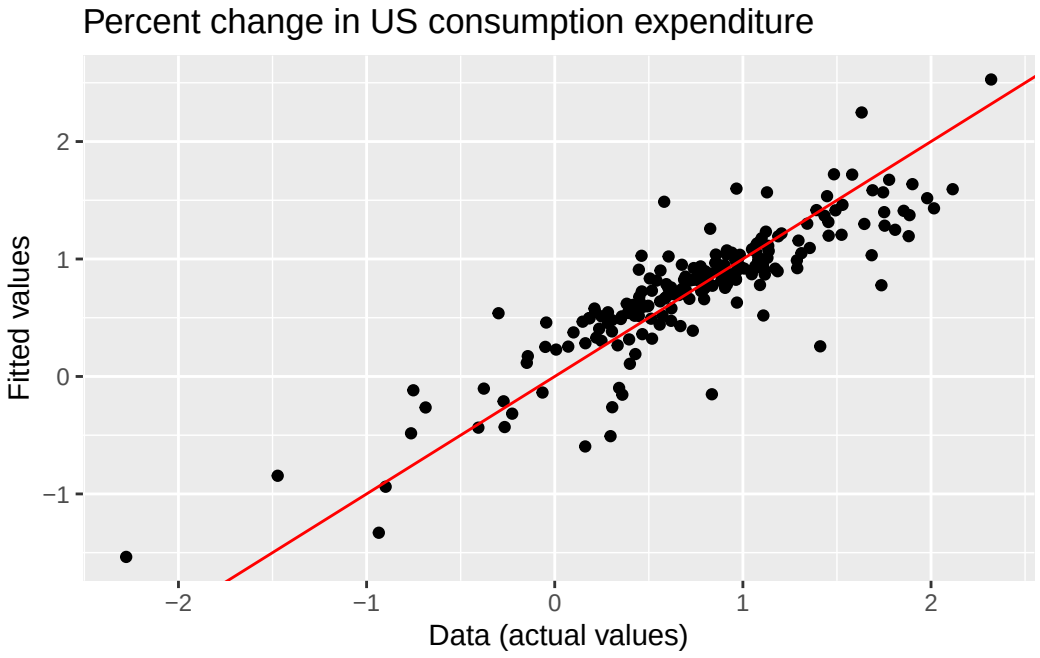
```
ggplot(augment(fit_consMR), aes(x = Quarter)) +
  geom_line(aes(y = Consumption, colour = "Data")) +
  geom_line(aes(y = .fitted, colour = "Fitted")) +
  labs(y = NULL, title = "Percent change in US consumption expenditure") +
  guides(colour = guide_legend(title = NULL))
```



From the plot we see our model fits the data pretty well.

An alternative way of visualizing how well our model fits the data is by plotting the fitted values to the actual values and then adding an identity line (a line of equation $y = x$) to the plot and check how close to the line the points are (closer to the line implying a better fit).

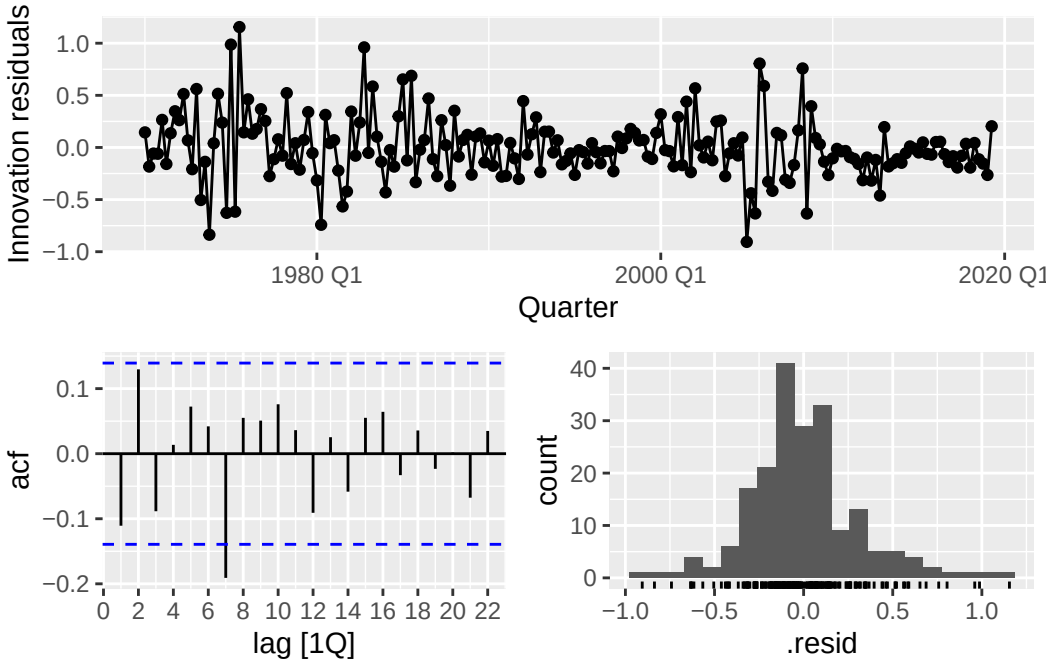
```
ggplot(augment(fit_consMR), aes(x = Consumption, y = .fitted)) +
  geom_point() +
  labs(y = "Fitted values",
    x = "Data (actual values)",
    title = "Percent change in US consumption expenditure") +
  geom_abline(intercept = 0, slope = 1, color = "red")
```



1.3 Diagnostics of residuals

As usual, this is done by using the `gg_tsresiduals()` function passing as argument inside the function the fitted model.

```
gg_tsresiduals(fit_consMR)
```



Then we can compute the Ljung-Box statistic to determine the presence of autocorrelation within the residuals of the model.

```
features(augment(fit_consMR),.innov, ljung_box, lag = 10, dof = 0)
```

```
# A tibble: 1 x 3
  .model lb_stat lb_pvalue
  <chr>    <dbl>     <dbl>
1 tslm      18.9    0.0420
```

A low enough p-value (even below .1) implies that there is a systematic pattern in the distribution of the residuals, and that therefore the residuals are not randomly distributed (i.e., not resulting from a white noise process). This is usually a problem that we want to address in our model.

Remember that having mean equal to 0 is also a property that is assumed for residuals in a model to respect along with other assumptions which you can check at the end of chapter 7.1.

2 New applications to chapter 7

In this section we gather some common lines of code that you can find useful to find solution to exercises.

2.1 Checking relevance of predictors

When fitting a model to data we might want to understand which predictors are more likely to be related to the prediction variable. One visual approach to do this has been outlined at the beginning of Section 1.2. Now we have a look at how to use glance to access model-specific statistics that can be used to compare one model to others and be sure to check the model that performs better on the given indicators.

```
select(glance(fit_consMR), adj_r_squared, CV, AIC, AICc, BIC)
```

```
# A tibble: 1 x 5
  adj_r_squared    CV   AIC  AICc   BIC
          <dbl> <dbl> <dbl> <dbl> <dbl>
1         0.763 0.104 -457. -456. -437.
```

By using the code above, we check 5 different measurements of fitness to the data. Each has a different interpretation. For instance a higher value for the adjusted R squared means that a model fits well the data, but a lower AIC also implies a better fit to the data! Check section 7.5 of the textbook to learn more about this.

2.2 Producing forecasts

Now we move on to using our model to produce forecasts. One way to do that is by using the `new_data()` function which appends to the end of a dataset the number of rows specified in the second argument passed inside it. Here we are adding 4 rows (equivalent to 4 quarters) at the end of the `us_change` dataset.

```
fit_consBest <- model(us_change,
    lm = TSLM(Consumption ~ Income + Savings + Unemployment))
(NewData <- new_data(us_change, 4))
```

```
# A tsibble: 4 x 1 [1Q]
  Quarter
    <qtr>
1 2019 Q3
2 2019 Q4
3 2020 Q1
4 2020 Q2
```

With `scenarios()` we can also specify different scenarios with specific values for the predictors employed in our model. New scenarios are specified by starting from our NewData and forecasting based on the values set for each individual scenario by `mutate()`.

```
(future_scenarios <- scenarios(Increase = mutate(NewData,
  Income=1, Savings=0.5, Unemployment=0),
  Decrease = mutate(NewData, Income=-1, Savings=-0.5,
  Unemployment=0),
  names_to = "Scenario"))
```

```
$Increase
# A tsibble: 4 x 4 [1Q]
  Quarter Income Savings Unemployment
    <qtr>  <dbl>   <dbl>        <dbl>
1 2019 Q3      1     0.5            0
```

```
2 2019 Q4      1     0.5            0
3 2020 Q1      1     0.5            0
4 2020 Q2      1     0.5            0

$Decrease
# A tsibble: 4 x 4 [1Q]
  Quarter Income Savings Unemployment
    <qtr>  <dbl>   <dbl>        <dbl>
1 2019 Q3     -1    -0.5            0
2 2019 Q4     -1    -0.5            0
3 2020 Q1     -1    -0.5            0
4 2020 Q2     -1    -0.5            0

attr(,"names_to")
[1] "Scenario"
```

Now we make forecasts for each scenario using the forecast function. Notice the `new_data` argument which is used to specify which dataset (or subset thereof) contains the values of the predictors that we want to use to make predictions about our y (`consumption` in our case).

```
(fc <- forecast(fit_consBest, new_data = future_scenarios))
```

```
# A fable: 8 x 8 [1Q]
# Key:     Scenario, .model [2]
  Scenario .model Quarter
  <chr>    <chr>    <qtr>
1 Increase lm     2019 Q3
2 Increase lm     2019 Q4
3 Increase lm     2020 Q1
4 Increase lm     2020 Q2
5 Decrease lm     2019 Q3
6 Decrease lm     2019 Q4
7 Decrease lm     2020 Q1
8 Decrease lm     2020 Q2
# i 5 more variables: Consumption <dist>, .mean <dbl>, Income <dbl>,
#   Savings <dbl>, Unemployment <dbl>
```

2.3 Nonlinear regression

Sometimes, it might be the case that the relationship between our prediction variables and the predictors is not linear. It is in cases like these that we can transform our prediction variable to get back to a linear relationship and use the methods discussed so far.

Tip

Remember the relationship between polynomials and linear functions. For instance, a polynomial of degree 2 can be returned to its linear form by taking the square root (i.e. it reduces the degree back to 1). Practically, if I take the following quadratic relationship given by:

$$y = x^2$$

this can be translated back into a linear relationship by taking the square root of both sides.

$$\sqrt{y} = x$$

Similarly, we can apply this procedure of applying the inverse function to exponential (using logarithmic functions) and other non-linear relationships to get back to a linear form (at least in the estimated parameters of the model!).

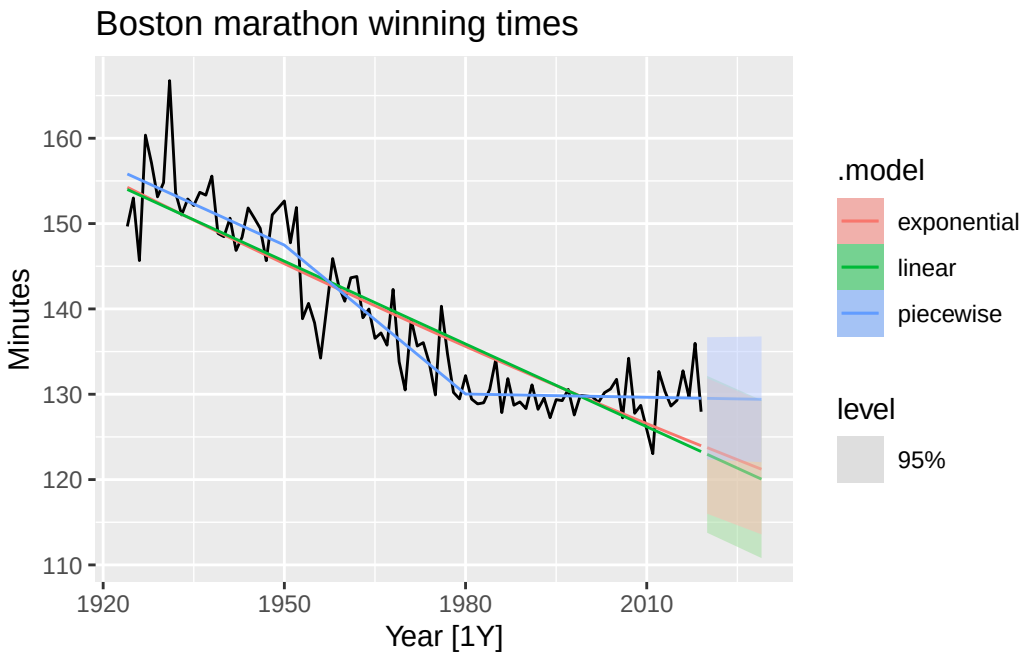
In the following code, we specify three models:

1. a model using no transformations and only `trend()` as a predictor.
2. a model applying a log transformation to the prediction variable y.
3. a `piecewise` model which estimates different coefficients for each time period specified in the `knots` argument inside of `trend`.

```
fit_trends <- model(boston_men, linear = TSLM(Minutes ~ trend()),
    exponential = TSLM(log(Minutes) ~ trend()),
    piecewise = TSLM(Minutes ~ trend(knots = c(1950, 1980))))
fc_trends <- forecast(fit_trends, h = 10)
```

Now in this plot we see how well each model fits the data.

```
autoplot(boston_men, Minutes) +
  geom_line(data = augment(fit_trends), aes(y = .fitted,
  colour = .model)) +
  autolayer(fc_trends, alpha = 0.5, level = 95) +
  labs(y = "Minutes", title = "Boston marathon winning times")
```



2.4 Piecewise method

As you might have noticed, we encountered most of the methods outlined in Section 2.3 in previous sections. However, one method we did not see before is the `piecewise` method. Using this method we can specify different trends to be estimated for each different time period. For instance in the third model specified in Section 2.3 the three different trends are specified in the model. The first trend goes from the beginning up to 1950, another is estimated from 1950 to 1980 and the last goes from 1980 to the end of our time series. You can go back to Section 2.3 and check how this is translated into code.

```
fit_trends %>%
  select(piecewise) %>%
  report()
```

```
Series: Minutes
Model: TSLM

Residuals:
    Min      1Q  Median      3Q     Max
-9.7548 -1.8547 -0.1071  1.9111 13.1794

Coefficients:
                                    Estimate Std. Error t value Pr(>|t|)
(Intercept)                        156.13547    1.28793 121.230  < 2e-16 ***
trend(knots = c(1950, 1980))trend   -0.32061    0.06721  -4.770 6.88e-06 ***
trend(knots = c(1950, 1980))trend_27 -0.26121    0.10165  -2.570   0.0118 *
trend(knots = c(1950, 1980))trend_57  0.56919    0.07620   7.469 4.51e-11 ***
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
Residual standard error: 3.496 on 92 degrees of freedom
Multiple R-squared: 0.884,  Adjusted R-squared: 0.8802
F-statistic: 233.6 on 3 and 92 DF, p-value: < 2.22e-16
```

Here I also included a summary of the model that shows the three estimated coefficients for each trend period which shows that each one is statistically significant.