FORECASTING AND PREDICTIVE MODELING

LECTURE 9

Lect. PhD. Oneț-Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2024 - 2025

In Lecture 8

- The forecasting workflow
  - Prediction intervals
  - Forecasting with transformations
  - Forecasting with decomposition
  - Evaluating the accuracy of forecasting
  - Cross validation

Today

- Time series regression models

Time series regression models

- The main idea of time series regression models is to forecast the value of a time series y assuming that it has a linear (or non-linear) relationship with another time series x.
- We will then use the value of x (called the *predictor variable*) to forecast the value of y (the *forecast variable*).

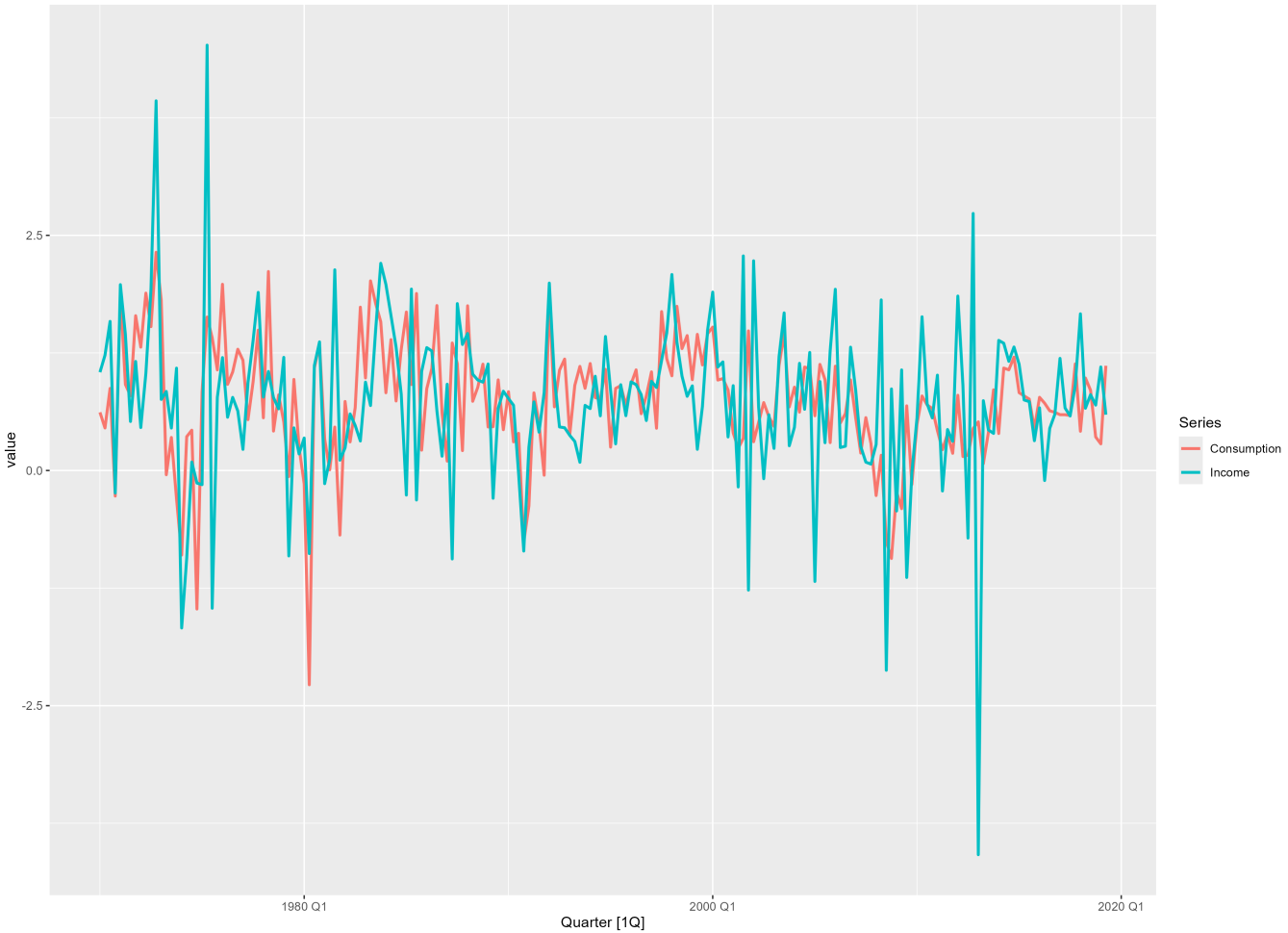
Simple linear regression

- In the simplest case, we have a linear relationship between the forecast variable and one single predictor variable:

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

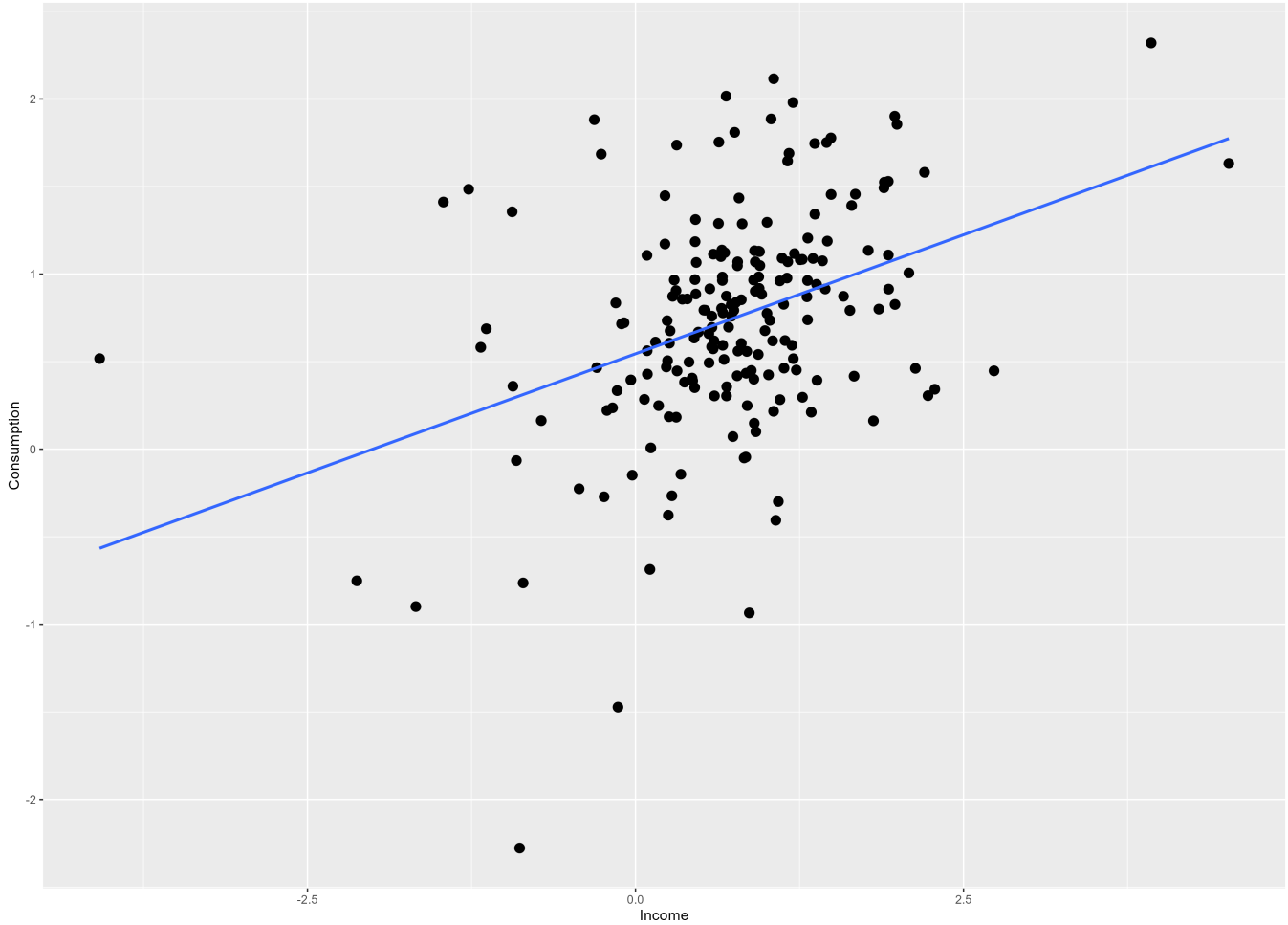
- where $\beta_0 + \beta_1 * x_t$ is called the *explained* part of the model and ϵ_t is the *random error* part.
- Let's consider the *us_change* data set, containing quarterly information about the percentage change in income, production, savings, consumption and unemployment from 1970 to 2019.
- Let's plot the Consumption and the Income from it.

```
us_change |>
  pivot_longer(c(Consumption, Income), names_to = "Series") |>
  autoplot(value)
```


2.5
0.0
-2.5
value
1980 Q1
2000 Q1
2020 Q1
Quarter [1Q]
Series
Consumption
Income

- We can see that the two time series often move together in the same direction. This suggests that there might be a relation between them.
- Let us do a scatter plot of consumption against income (and plot the regression line as well).

```
us_change |>
  ggplot(mapping = aes(x = Income, y = Consumption)) +
  geom_point() +
  geom_smooth(method = "lm", se = FALSE)
```



- In order to see the exact equation we can use the *TSLM* (time series linear model) function, passed to the model function.

```r
us_change |>
  model(lm = TSLM(Consumption ~ Income)) |> #notation means: build model to
     predict Consumption using Income as predictor
  report()
```

```
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
```

- From the output (section Coefficients), we can see that the linear regression model is

$$\hat{y}_t = 0.54 + 0.27 * x_t$$

$$\hat{Consumption}_t = 0.54 + 0.27 * Income_t$$

- We can see that even if we specified in the TSLM model that we want Consumption predicted by Income, an intercept (β_0) was also added by default.

- We can also see that the actual points are not on the line, but around it. The deviation from the line is the *random error* part, while the equation of the line is the *explained* part.

Multiple linear regression

- Sometimes we might have more than one predictor variable(s), in these cases we have multiple linear regression:

$$y_t = \beta_0 + \beta_1 * x_{1,t} + \beta_2 * x_{2,t} + \cdots + \beta_k * x_{k,t} + \epsilon_t$$

- The coefficient β_i measures the effect of predictor x_i after taking into account the effects of all the other predictors.

- Our *us_production* data set contains other time series as well, which might be relevant for forecasting consumption: personal savings and unemployment rate.
- We can use them in a model as predictors:

```
us_change |>
  model(TSLM(Consumption ~ Income + Savings)) |> report()
```

```
Series: Consumption
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-0.96934 -0.14829 -0.01167  0.14240  1.41274

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)  0.21543    0.03393   6.349 1.49e-09 ***
Income       0.83664    0.03748  22.324  < 2e-16 ***
Savings     -0.05901    0.00282 -20.927  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.3286 on 195 degrees of freedom
Multiple R-squared: 0.7373, Adjusted R-squared: 0.7346
F-statistic: 273.6 on 2 and 195 DF, p-value: < 2.22e-16
```

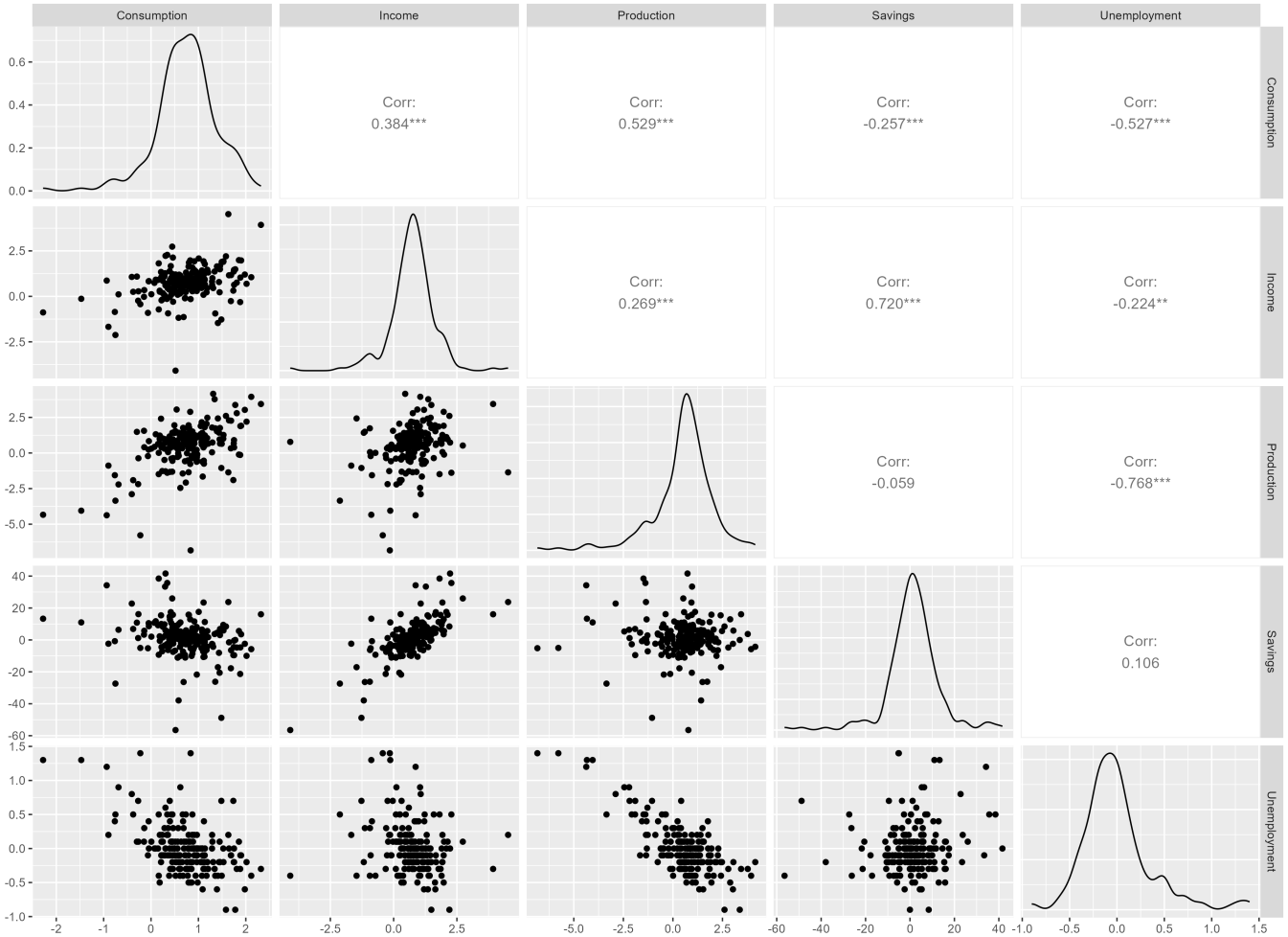
- The report function tells us that the regression model is the line:

$$\hat{y}_t = 0.22 + 0.84 * Income_t - 0.06 * Savings_t$$

- When we have several possible predictors, it might be a good idea to first do some visualizations, to get an intuition about which ones to add to the model.
- We can do simple time series plots in parallel for them (to visually evaluate correlation) or we can do scatterplots.

```
us_change |>
  pivot_longer(c(Consumption, Income, Production, Savings, Unemployment), names_
      to = "Series") |>
  autoplot(value, size = 1) +
  facet_wrap(vars(Series), nrow = 5, scales = "free")

us_change |>
  GGally::ggpairs(columns = 2:6)
```


Consumption
Income
Production
Savings
Unemployment
value
Quarter [1Q]
1980 Q1
2000 Q1
2020 Q1
Series
Consumption
Income
Production
Savings
Unemployment


Consumption
Income
Production
Savings
Unemployment
Corr:
0.384***
Corr:
0.529***
Corr:
-0.257***
Corr:
-0.527***
Corr:
0.269***
Corr:
0.720***
Corr:
-0.224**
Corr:
-0.059
Corr:
-0.768***
Corr:
0.106

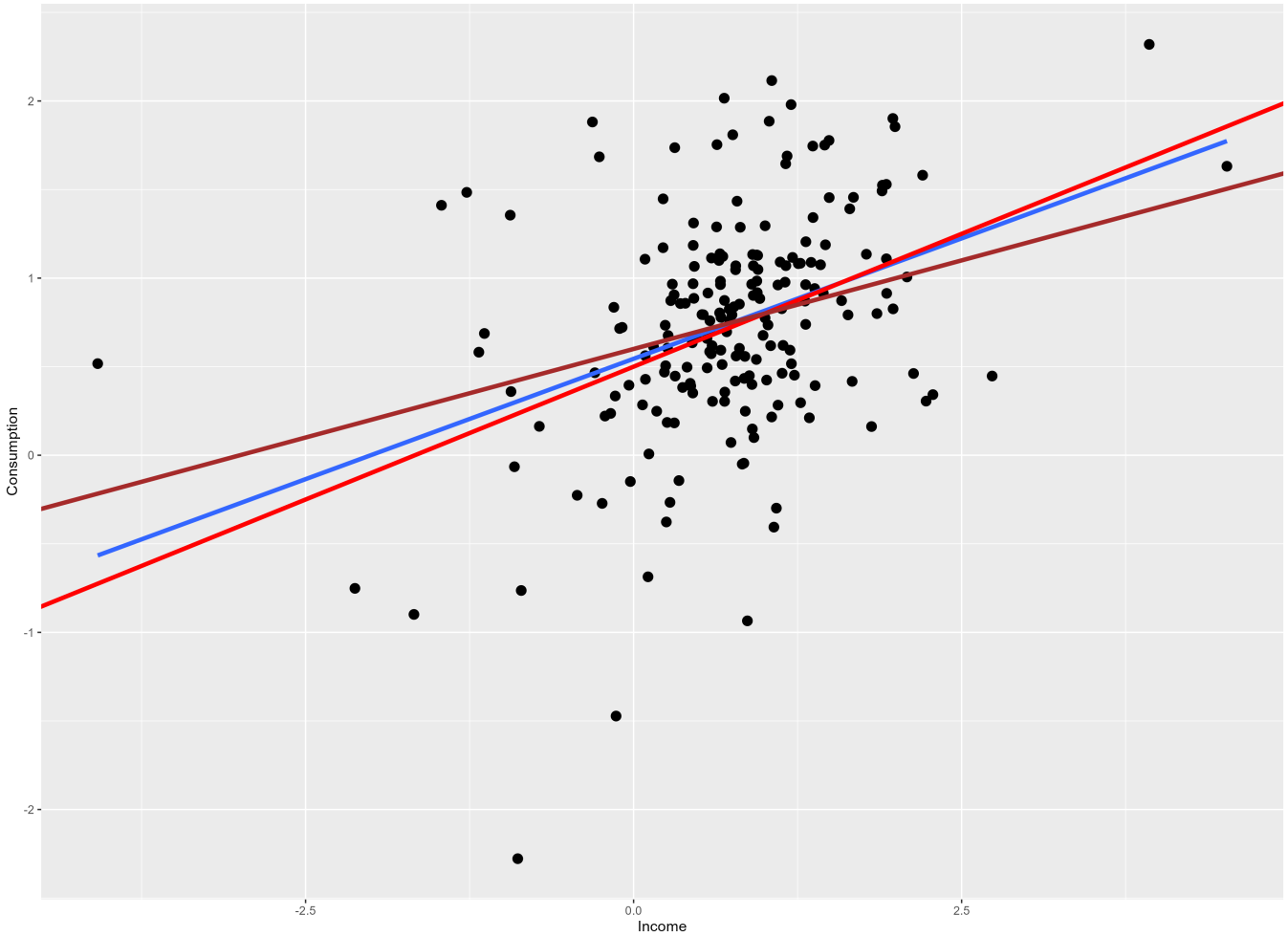
- We can see (both from the scatter plots below the main diagonal and from the correlations above it) that Consumption is positively correlated with Production and negatively correlated with Unemployment.

- Also, there is a strong positive correlation between Income and Savings and a strong negative correlation between Unemployment and Production.

Assumptions

- When using a linear regression model, we automatically make some assumptions about the above variables:
  - Since we are using linear regression, obviously, we assume that there is a linear relationship between x and y.
  - the errors (ϵ_t) have zero mean and they have no autocorrelation
  - The errors (ϵ_t) are not correlated with the predictor variables
- It is useful to also have the errors normally distributed.

- The blue line is the one returned by the linear regression model. But why is that the best? Why not the others?



Least squares estimation

- For finding the values of coefficients $\beta_1, \beta_2, etc.$ a method called *least squares estimation* is used. This method tries to reduce the sum of the squares of the error term:

$$\sum_{i=1}^{T} \epsilon_t^2 = \sum_{i=1}^{T} (y_t - \beta_0 - \beta_1 * x_{1,t} - \beta_2 * x_{2,t} - \cdots - \beta_k * x_{k,t})^2$$

- Determining the best values for the coefficients is called *fitting* or *training* the model.
- In R the *TSLM* function can be used to fit a linear regression model.

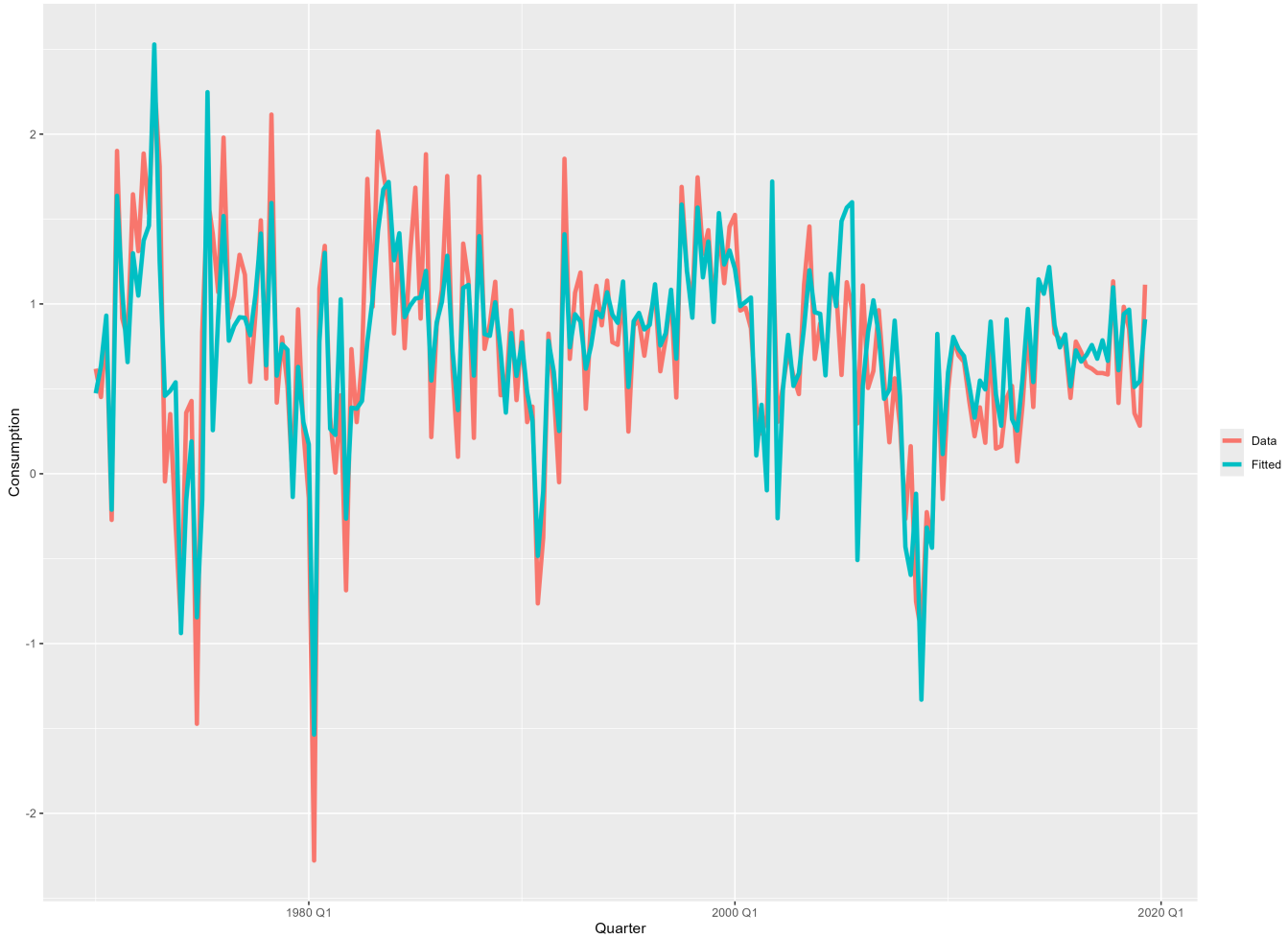
```
us_change |>
  model(TSLM(Consumption ~ Income + Production + Unemployment + Savings +
      Unemployment)) |> report()

us_change |>
  model(TSLM(Consumption ~ 0 + Income + Production + Unemployment + Savings +
      Unemployment)) |> report() # 0+ means that we do not want intercept term
```

Fitted values

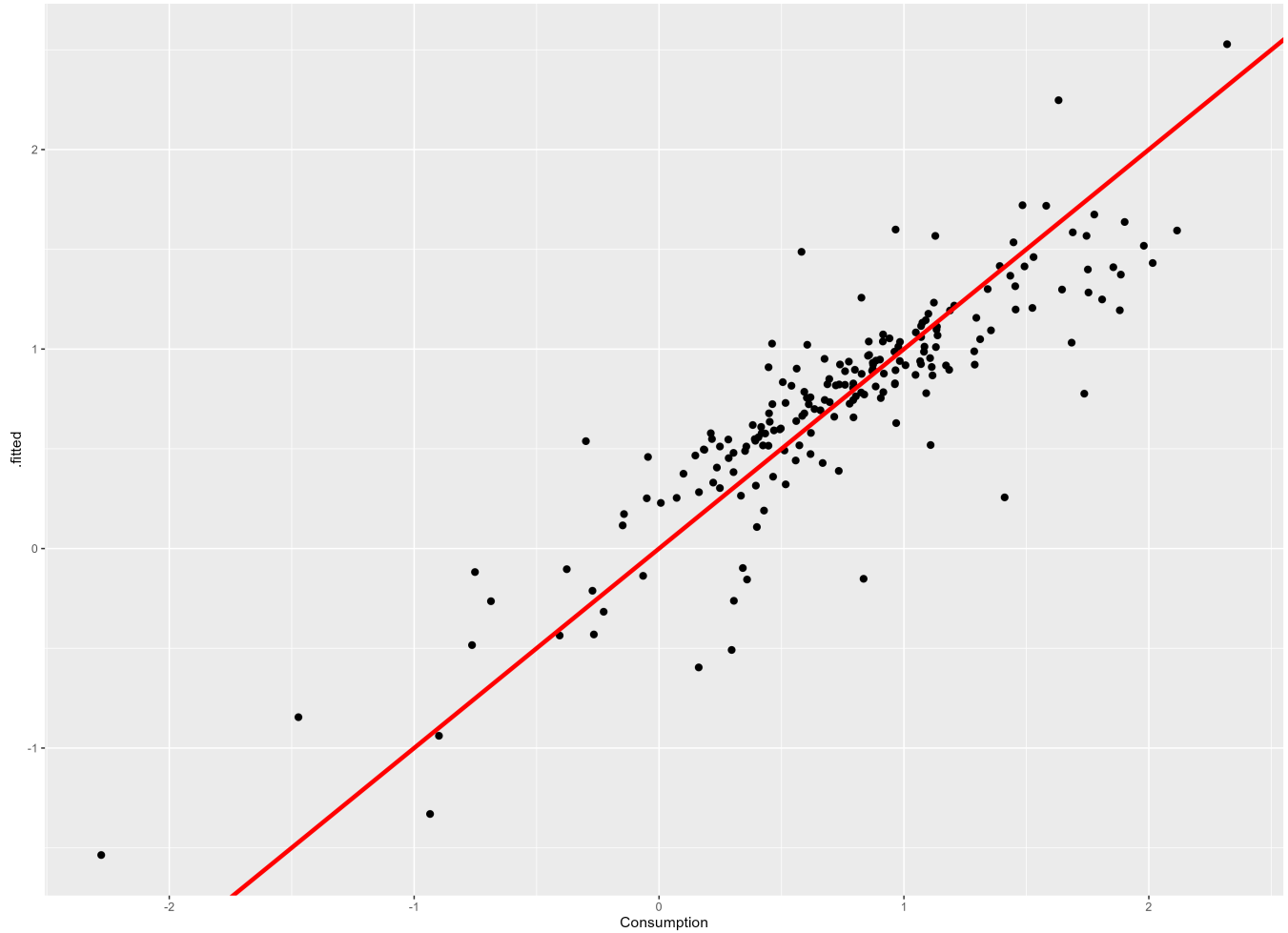
- The fitted values (predictions over the training data) can be computed using the coefficients provided by the model and setting the error term to zero.

- Remember, the *augment* function will display the fitted values and the residuals (which is the error term of the equation)

```
us_change |>
  model(TSLM(Consumption ~ Income + Savings + Unemployment + Production)) |>
  augment() |>
  ggplot(mapping = aes(x = Quarter)) +
  geom_line(mapping = aes(y = Consumption, color = "Data")) +
  geom_line(mapping = aes(y = .fitted, color = "Fitted")) +
  guides(color = guide_legend(title = NULL))

us_change |>
  model(TSLM(Consumption ~ Income + Savings + Unemployment + Production)) |>
  augment() |>
  ggplot(mapping = aes(x = Consumption, y = .fitted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1)
```



- We can see that the fitted values cover the actual data quite well.



- We can see that the points are quite close to the red line (red line means perfect prediction, where fitted value = actual data)

Goodness of fit

- A commonly used function to see how well a linear model fits the set of training data is the R^2. It can be computed in the following way:

$$R^2 = \frac{\sum(\hat{y}_t - \bar{y})^2}{\sum(y_t - \bar{y})^2}$$

- where $\bar{y}$ is the mean of the values from the training data. The numerator represents the variance in the fitted values, while the denominator represents the variance in the actual data.
- One way to interpret R^2 is to say that it measures the proportion of the variation in the forecast variable that is explained by the model.
- R^2 takes values between 0 and 1, the value 1 meaning perfect fit for the model. Since R^2 is defined on the train data, a proper evaluation with test set and forecasts is needed.
- In R, the value of R^2 can be seen in the output of the *report* function.

Residual standard error

- Another measure to see how well the data fits is the *residual standard error* computed in the following way:

$$\hat{\sigma}_e = \sqrt{\frac{1}{T-k-1} * \sum_{t=1}^{T} e_t^2}$$

- where k is the number of predictors in the model.
- One way of looking at the residual standard error is to consider that it is the average error that the model produces.
- **Obs.** One important difference between R^2 and $\hat{\sigma}_e$ is that the latter considers the number of predictor variables (through the parameter k).
- The *report* function in R will return the value of the *residual standard error* as well (besides the value of R^2).

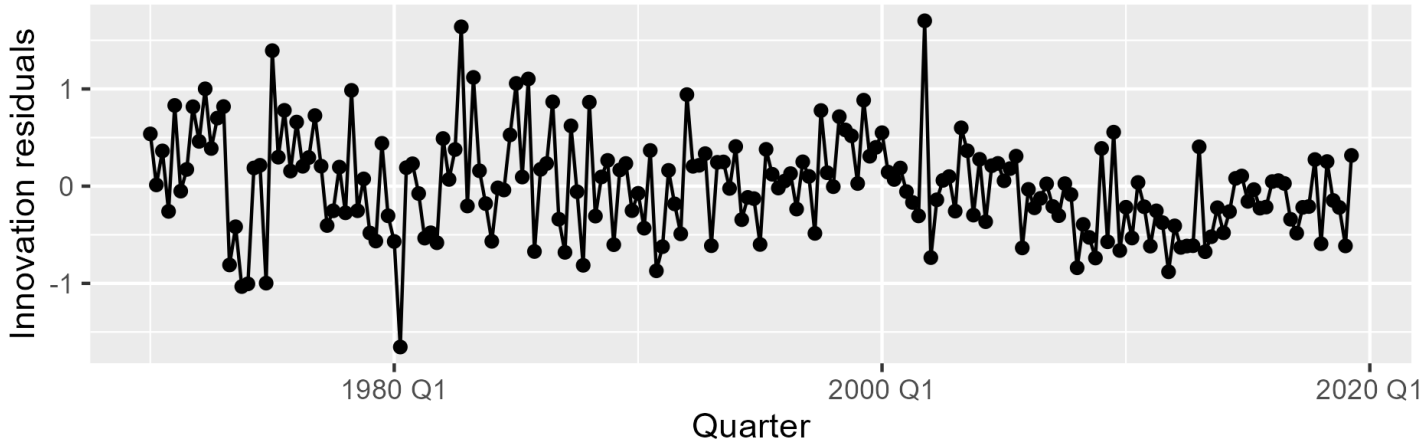
```r
us_change |>
  model(TSLM(Consumption ~ Income)) |>
  report()
```

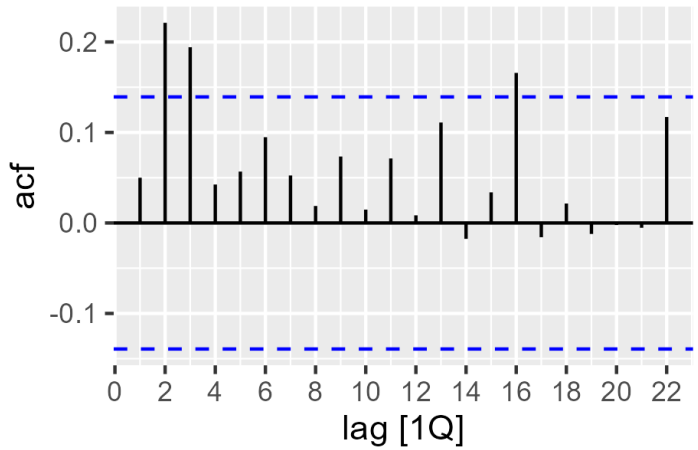
```
Series: Consumption
Model: TSLM

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
```

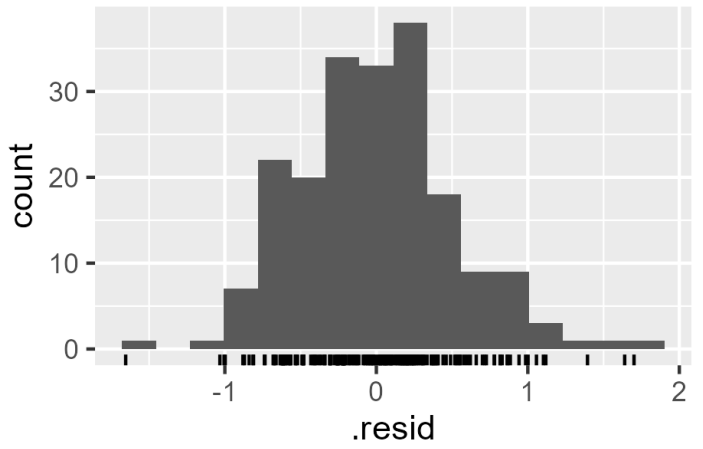
Evaluating the regression model

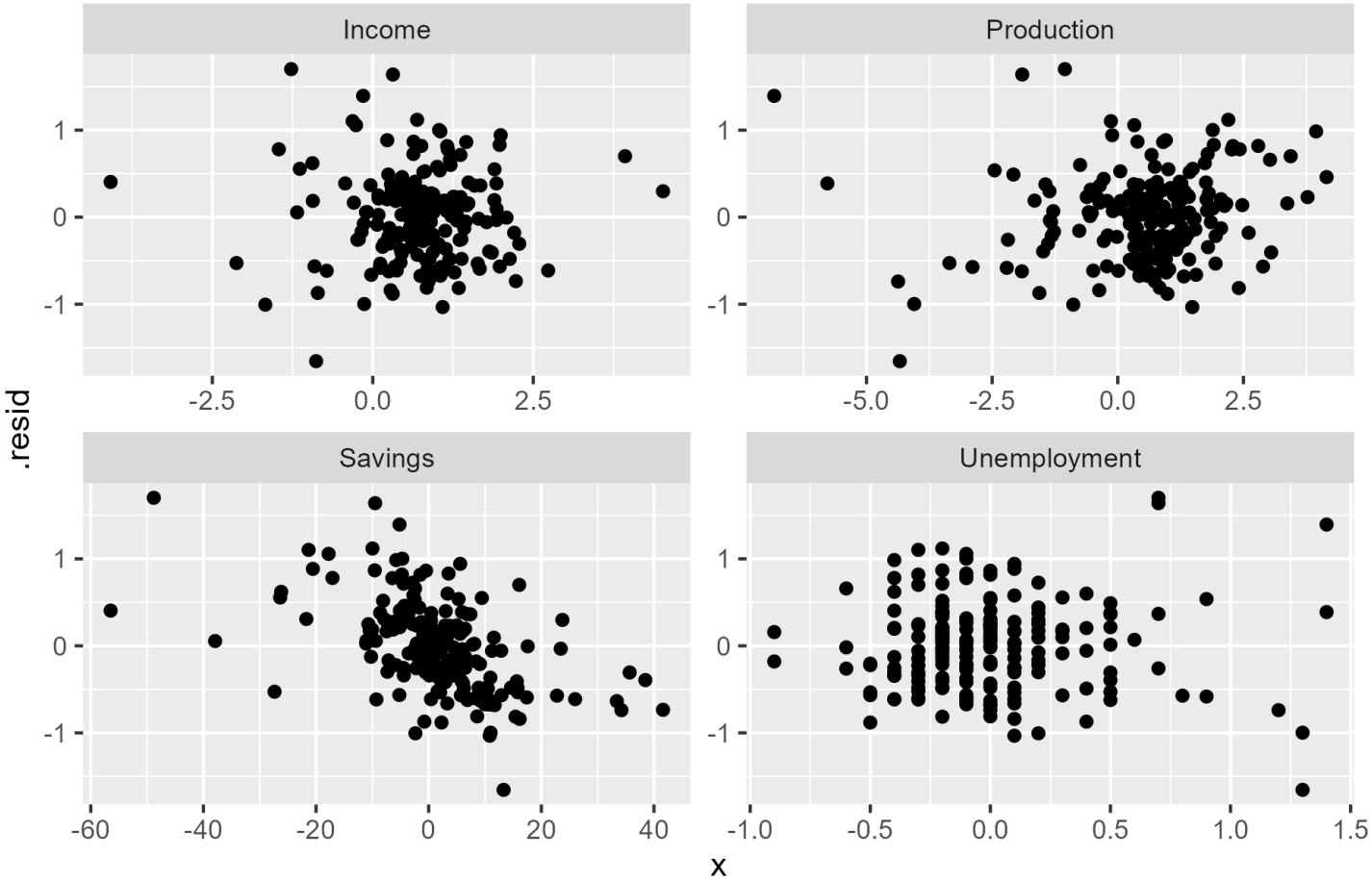
- What was discussed in the previous lesson about evaluating a model is valid here as well; we can do residual diagnostics, using the *gg_tsresiduals* function:
  - ACF plot (possibly with statistical test)
  - Histogram of the residuals
  - plot of the residuals
- Another thing that can be done is to create scatter plot of the residuals against the predictor variables. If they show a pattern then it might happen that the predictor should be included in a non-linear form.
- Also, if there are predictors not included in the model, they should also be plot against the residuals. If there is a pattern, these predictors should be added, but possibly in a non-linear form.

```
us_change_model <- us_change |>
  model( all = TSLM(Consumption ~ Unemployment + Income))

us_change_model |> gg_tsresiduals()

us_change_resid <- us_change_model |> augment()
us_change |>
  left_join(us_change_resid, by = "Quarter") |>
  pivot_longer(c(Income, Unemployment, Savings, Production), names_to = "
      predictor", values_to = "x") |>
  ggplot(mapping = aes(x = x, y = .resid)) +
  geom_point() +
  facet_wrap(vars(predictor), scales = "free")
```


Innovation residuals
1
0
-1
1980 Q1
2000 Q1
2020 Q1
Quarter


acf
0.2
0.1
0.0
-0.1
0 2 4 6 8 10 12 14 16 18 20 22
lag [1Q]

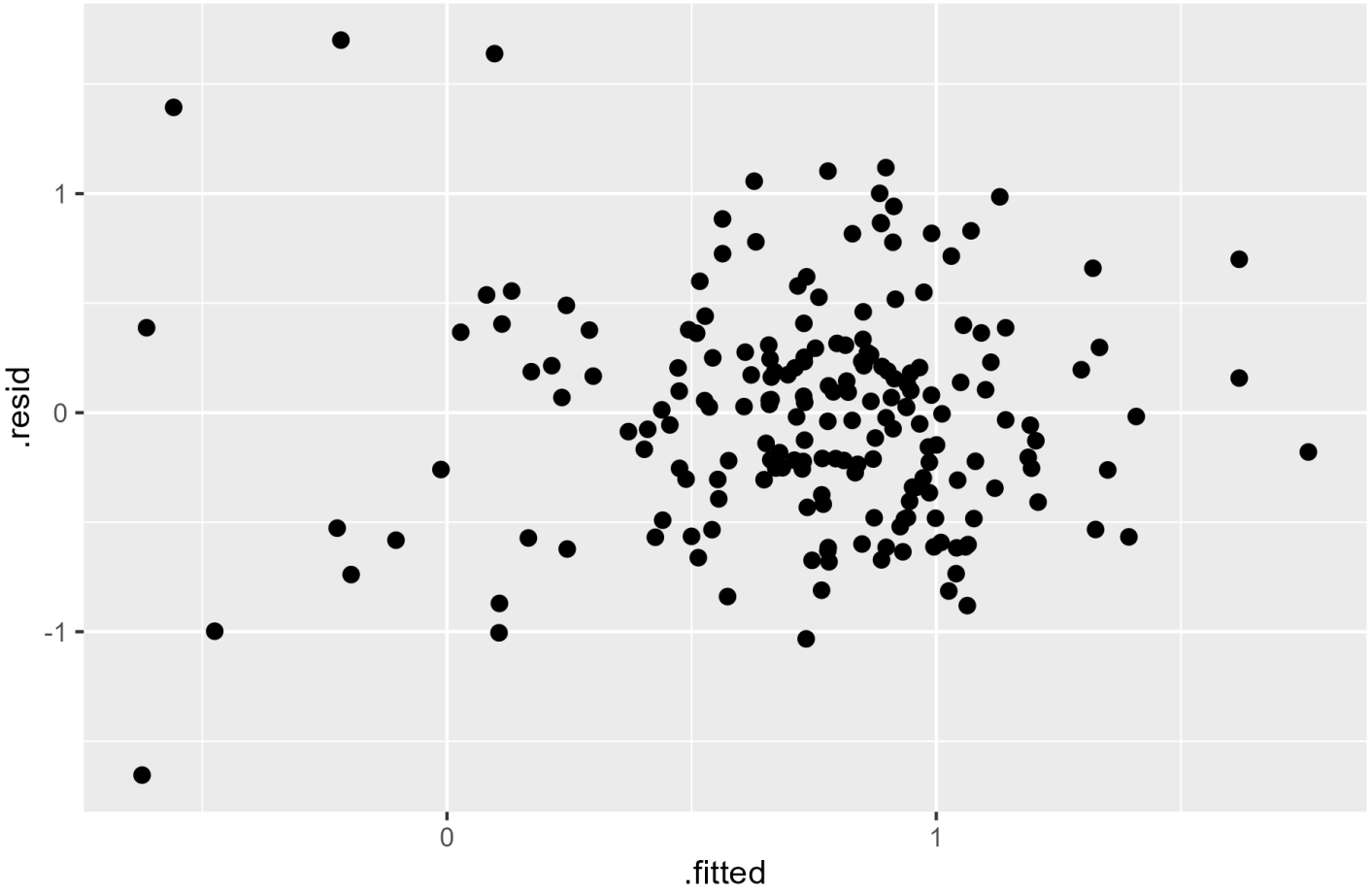

count
30
20
10
0
-1 0 1 2
.resid



- There is no pattern in the plots for Income and Unemployment (the variables used to build the model), but there might be a slight linear pattern with Production. It might be worth experimenting with adding it to the model.

- We can also plot the residuals against the fitted values, and there should be no pattern there either.
- If there is a pattern, maybe a transformation of the forecast variable is required.

```
us_change_resid |>
  ggplot(mapping = aes(x = .fitted, y = .resid)) +
  geom_point(size = 2)
```

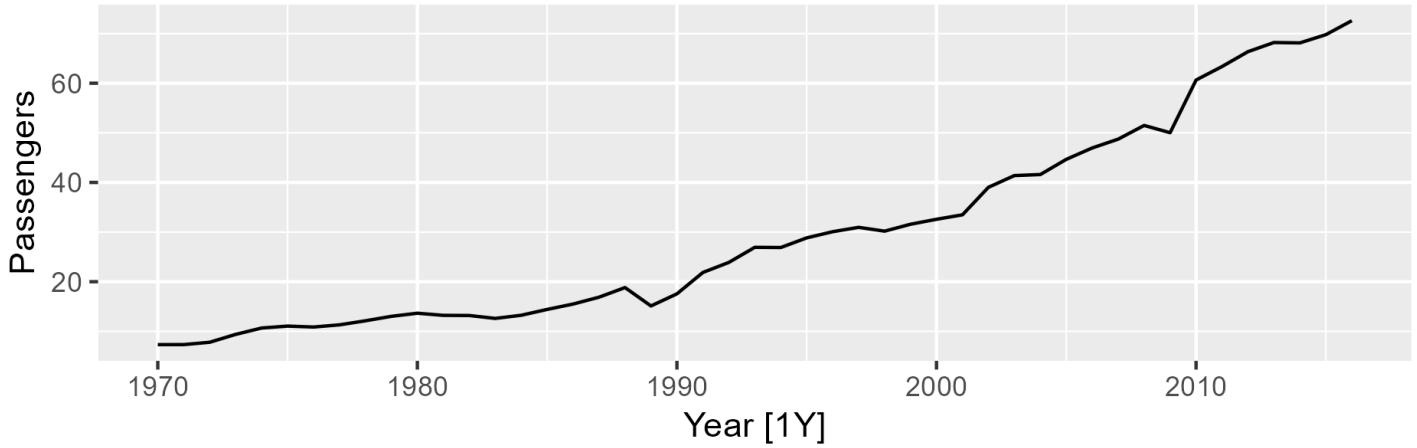


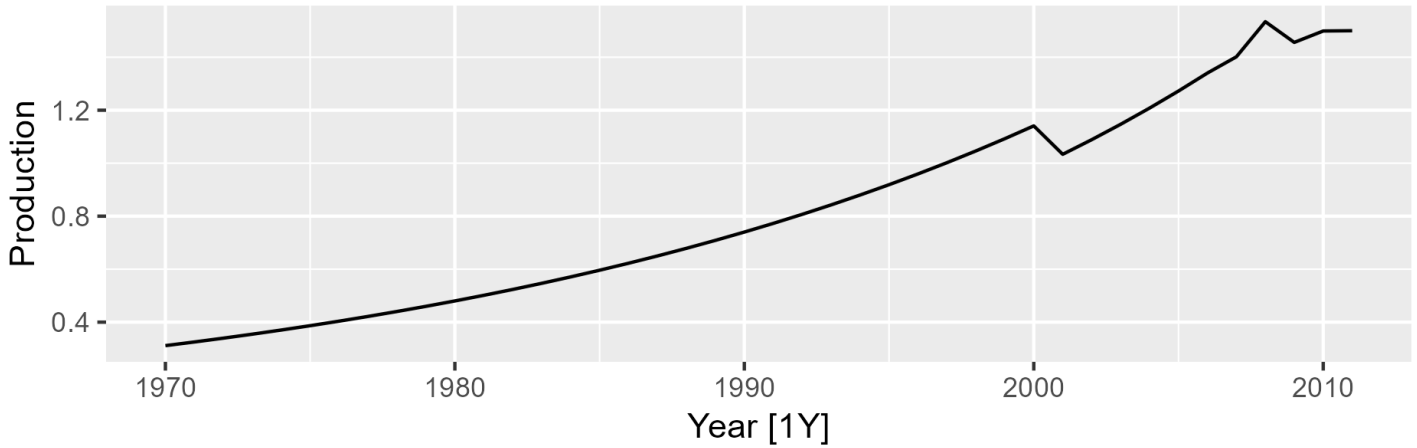
- There is no strong pattern in the plot.

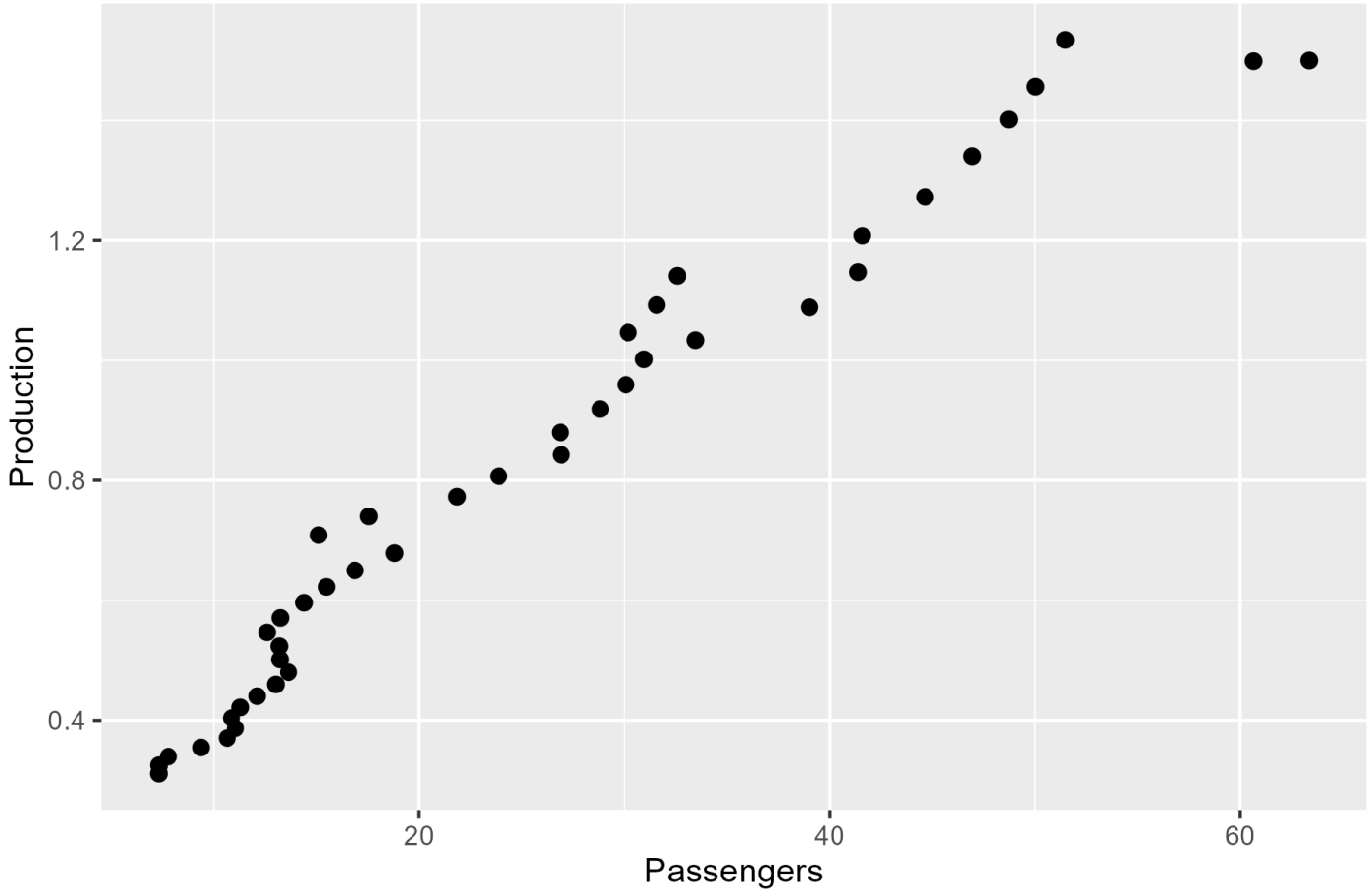
Spurious regression

- We have talked about spurious correlation previously: the phenomenon that two time series are correlated (have a high correlation coefficient), simply because they have a similar trend.
- Let us take two time series that have nothing to do with each other: *aus_airpassengers* - the total annual air passengers in million between 1970 and 2016 - and *guinea_rice* - the total annual rice production for Guinea 1970 - 2011.
- If we plot them, we will get very similar plots.

```r
aus_airpassengers
guinea_rice

autoplot(aus_airpassengers) -> p1
autoplot(guinea_rice) -> p2
plot_grid(p1, p2, nrow = 2) #requires library cowplot

cor(aus_airpassengers$Passengers |> head(n = 42), guinea_rice$Production)

#merge them in one tsibble
joined <- aus_airpassengers |>
  head(n = 42) |>
  left_join(guinea_rice, by = "Year")

joined |>
  ggplot(mapping = aes(x = Passengers, y = Production)) +
  geom_point(size = 2)
```


Passengers
60
40
20
1970
1980
1990
2000
2010
Year [1Y]


Production
1.2
0.8
0.4
1970
1980
1990
2000
2010
Year [1Y]

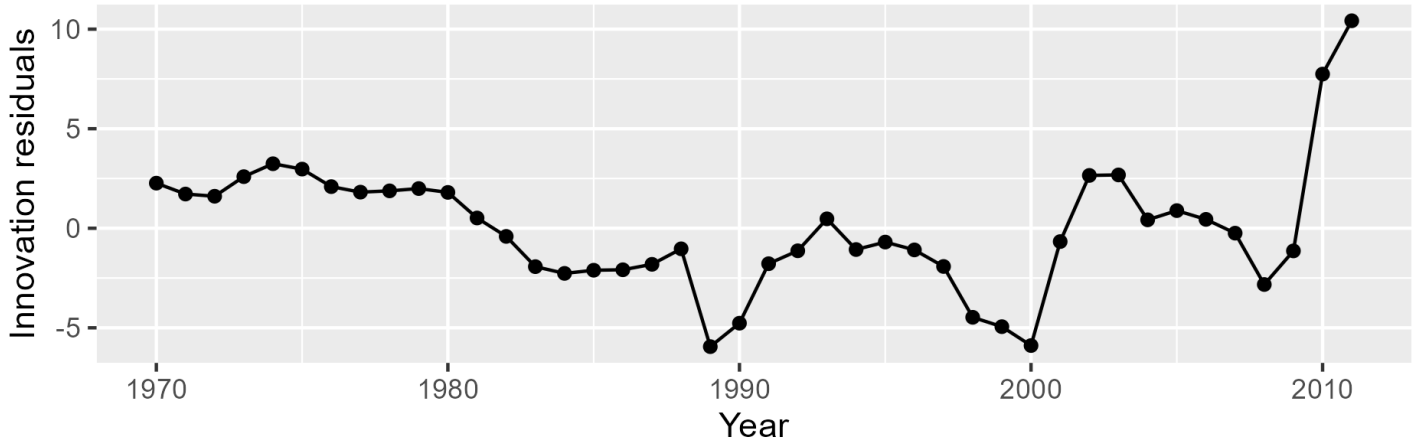


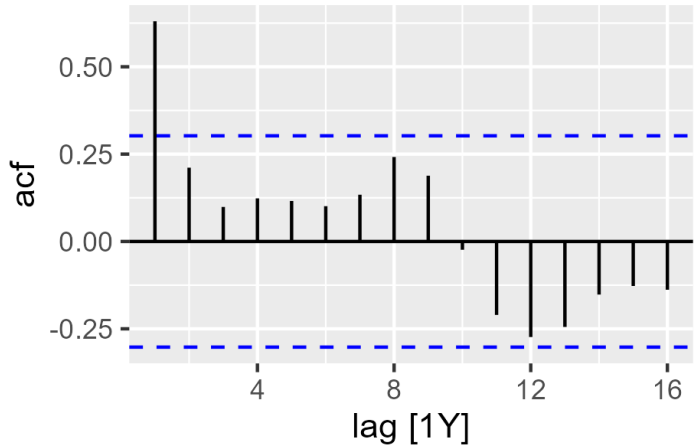
- Looking just at the plots, it looks like a linear regression model would be a great option.

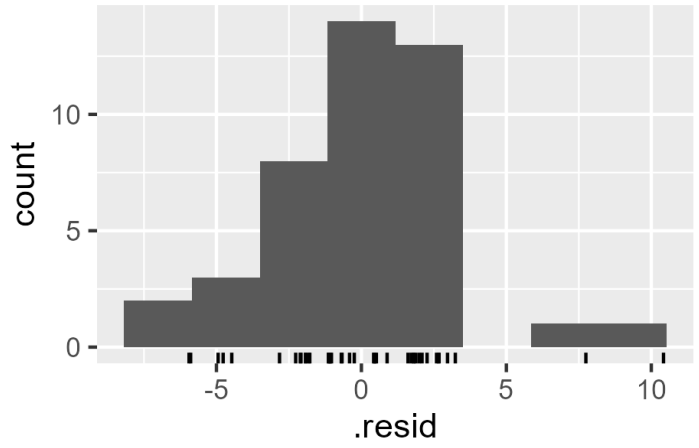
```r
model <- joined |>
  model(model = TSLM(Passengers ~ Production))

model |> report()
```

```
Series: Passengers
Model: TSLM

Residuals:
    Min      1Q  Median      3Q     Max
-5.9448 -1.8917 -0.3272  1.8620 10.4210

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept)   -7.493      1.203  -6.229 2.25e-07 ***
Production    40.288      1.337  30.135  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 3.239 on 40 degrees of freedom
Multiple R-squared: 0.9578, Adjusted R-squared: 0.9568
F-statistic: 908.1 on 1 and 40 DF, p-value: < 2.22e-16
```

```
model |> gg_tsresiduals()

model |>
  augment() |>
  features(.innov, ljung_box, lag = 8) #lag should be minimum between 10 and T/5
```


Innovation residuals
10
5
0
-5
1970
1980
1990
2000
2010
Year


acf
0.50
0.25
0.00
-0.25
4
8
12
16
lag [1Y]


count
10
5
0
-5
0
5
10
.resid

```
.model lb_stat lb_pvalue
  <chr>    <dbl>     <dbl>
1 model     26.5  0.000867
```

- When we see a high R^2 value but also high residual autocorrelation, it is a sign of spurious regression.
- Even if for short periods spurious regression might provide good forecasts, obviously, on the long term they will not have good performance.

Useful predictors for time series

- In the previous example, we assumed that we have a data set made of several time series which can be used as predictors.
- Now we will look at what other predictors might be useful.

Trend as predictor I

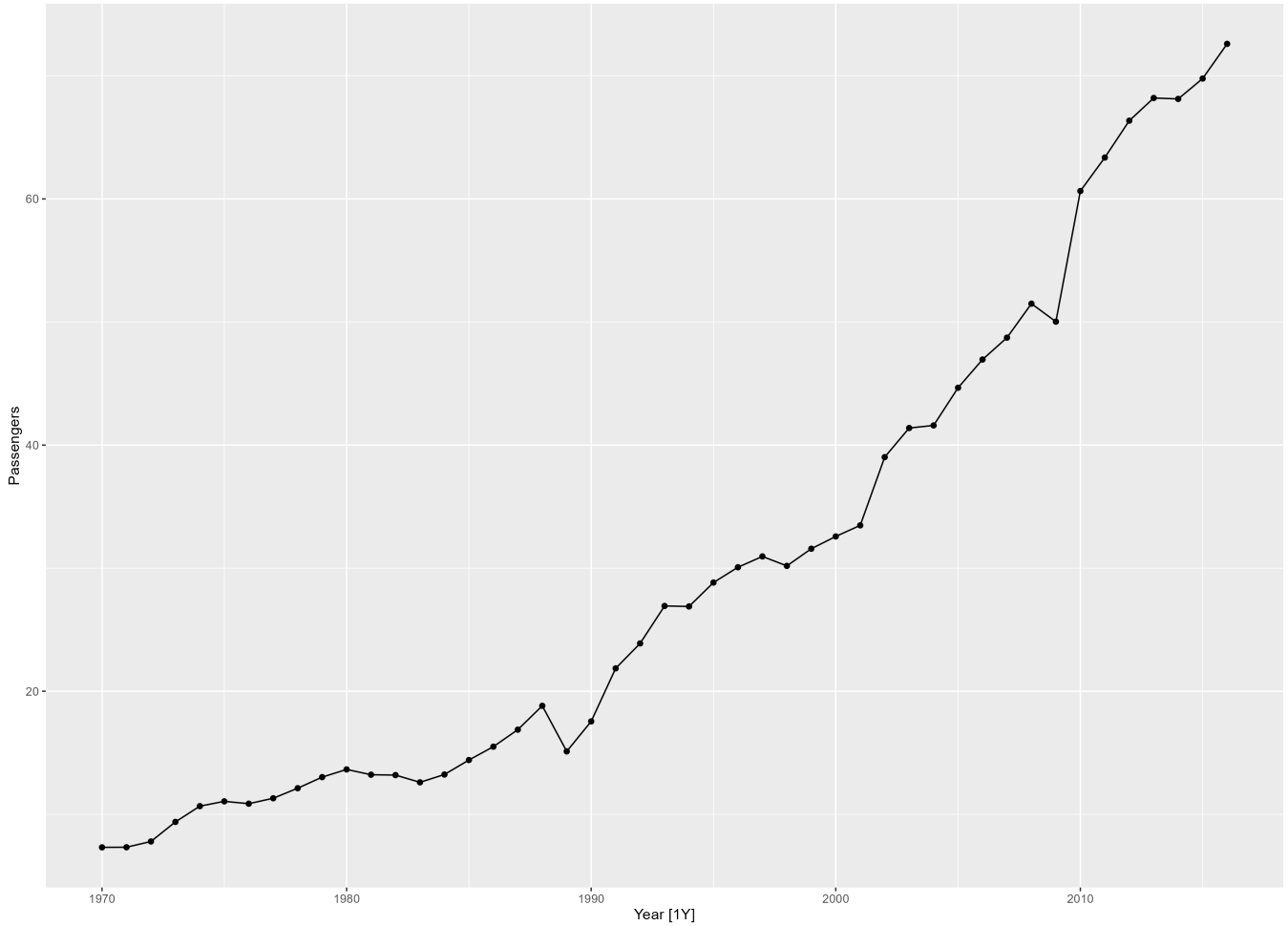
- If the time series has a trend, we can create a feature which is simply the time of an observation and use it as predictor.

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

- In R this can be done by specifying *trend()* as predictor in the TSLM model.

```
aus_airpassengers |>
    autoplot() + geom_point()

air_model <- aus_airpassengers |>
  model(m = TSLM(Passengers ~ trend()))
air_model |>
  report()
```

Trend as predictor II

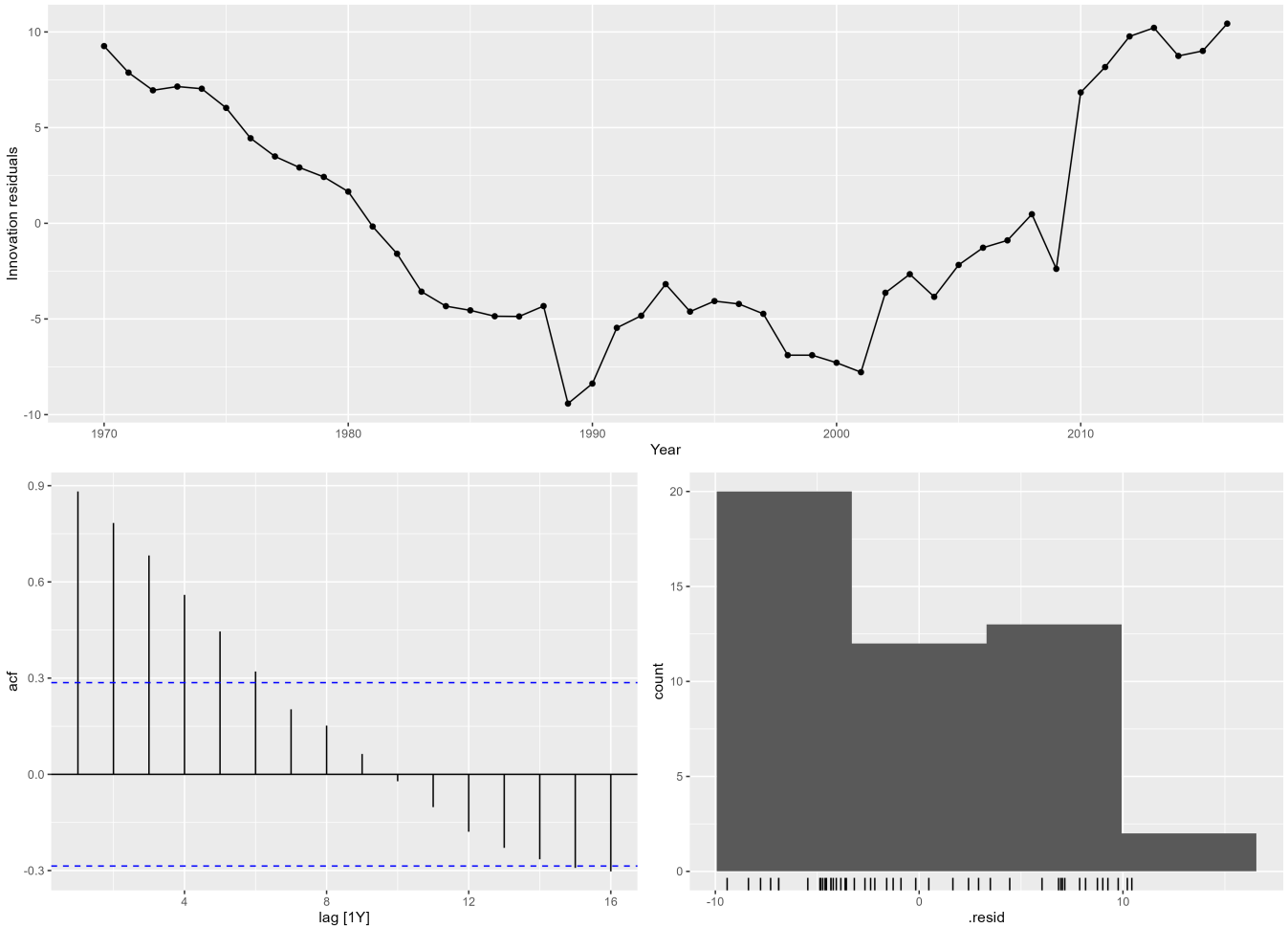


Trend as predictor III

```
Series: Passengers
Model: TSLM

Residuals:
   Min     1Q Median     3Q    Max
-9.422 -4.582 -2.176  6.434 10.435

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -3.33603    1.78943  -1.864   0.0688 .
trend()      1.39360    0.06491  21.470   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 6.036 on 45 degrees of freedom
Multiple R-squared: 0.9111, Adjusted R-squared: 0.9091
F-statistic:   461 on 1 and 45 DF, p-value: < 2.22e-16
```

```
air_model |>
  gg_tsresiduals()
```

Trend as predictor IV



Trend as predictor V

- Residual diagnostics tell us, that this is not a good model.
- But even in cases when trend is a good predictor, it might not be a good predictor for a long time.

Dummy variables I

- What happens when we have categorical data in a data set and we want to use it as predictor? For example, in a sales-related time series an attribute specifying if the given day was a public holiday or not. This attribute would have the values *yes* or *no*.
- This situation can be handled by adding a dummy variable, which takes the values 0 (ex. for *no*) and 1 (for *yes*).

- Dummy variables can also be used to deal with outliers. Sometimes outliers are bad data (possibly registered incorrectly) but sometimes outliers are valid values that appear due to some special conditions (for example: tourist arrivals to a specific city might have outlier values if some special events - like the Olympics - are held in the city). In this case, instead of removing the values, it is easier to add a dummy variable to take value 1 for the outlier instance(s) and 0 for regular ones.

Dummy variables II

- If there are more than two possible values for the attribute, we need several dummy variables. More exactly, we need one less dummy variables than the number of possible values, since one of the values will be encoded with all zeros.
- For example, if we have daily data, we might want to consider the day of the week as a predictor. In this case, we need 6 dummy variables, and this is one example of how we can use them:
  - Monday - first dummy is 1, the others are zero
  - Tuesday - second dummy is 1, the others are zero
  - Wednesday - third dummy is 1, the others are zero
  - ...
  - Saturday - last dummy is 1, the others are zero
  - Sunday - all dummies are 0

Dummy variables III

- We need one less dummy, because the regression model that we will build also contains an intercept (the term β_0) which will measure the effect of the variable with all zeros.
- The interpretation of the coefficient that the regression will assign to each dummy is that it is a measure of the effect of that category *relative to the omitted category*.
- The coefficient that will be assigned to the first dummy in the previous example (the one which has a value of 1 for Monday) will show the relative influence of Mondays compared to Saturday.
- We could create dummies similarly for months in case of monthly data, or quarters in case of quarterly data. They are called *seasonal dummy variables*.

Dummy variables IV

- If you provide categorical features as predictors to the TSLM function, it will automatically create dummy variables for it. If you want to use seasonal dummy variables as predictors, you can use the *season* special as a predictor in the TSLM model.

Example I

- Let's consider the Australian beer production time series that we have already used and fit a linear regression model on the data from 1992 considering the trend and dummies for the quarters. This means that our model is going to be:

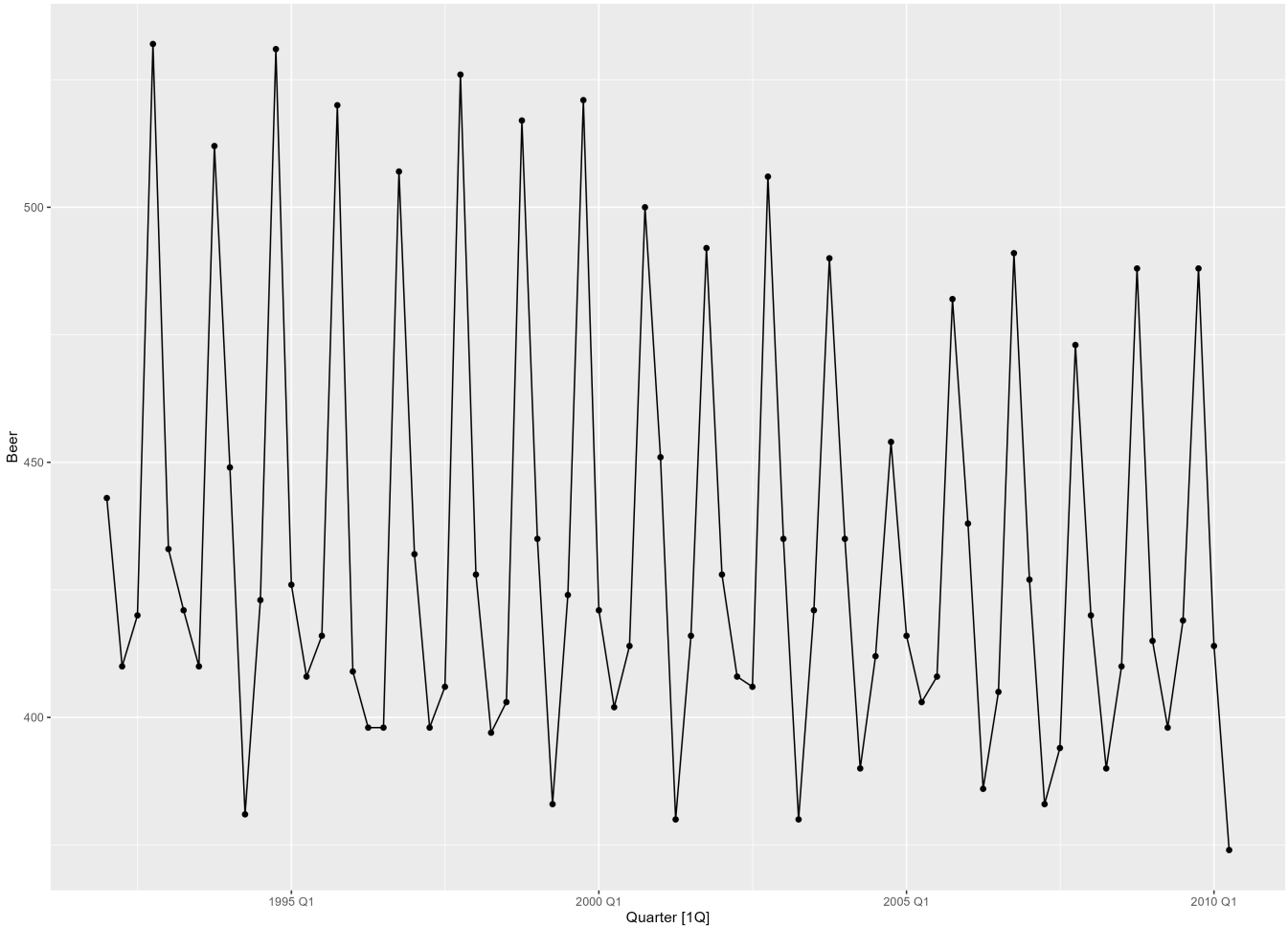
$$y_t = \beta_0 + \beta_1 * t + \beta_2 * d_{2,t} + \beta_3 * d_{3,t} + \beta_4 * d_{4,t} + \epsilon_t$$

- where $d_{i,t}$ is 1 if t is in quarter i. This means that we do not have a dummy for the first quarter, so the coefficients of the dummies will measure the effect of the specific quarter relative to the first one.

Example II

```
recent_production <- aus_production |>
  filter(year (Quarter) >= 1992)

recent_production |>
  autoplot(Beer) +
  geom_point()

beer_fit <- recent_production |>
  model(TSLM(Beer ~ trend() + season()))
report(beer_fit)
```

Example III



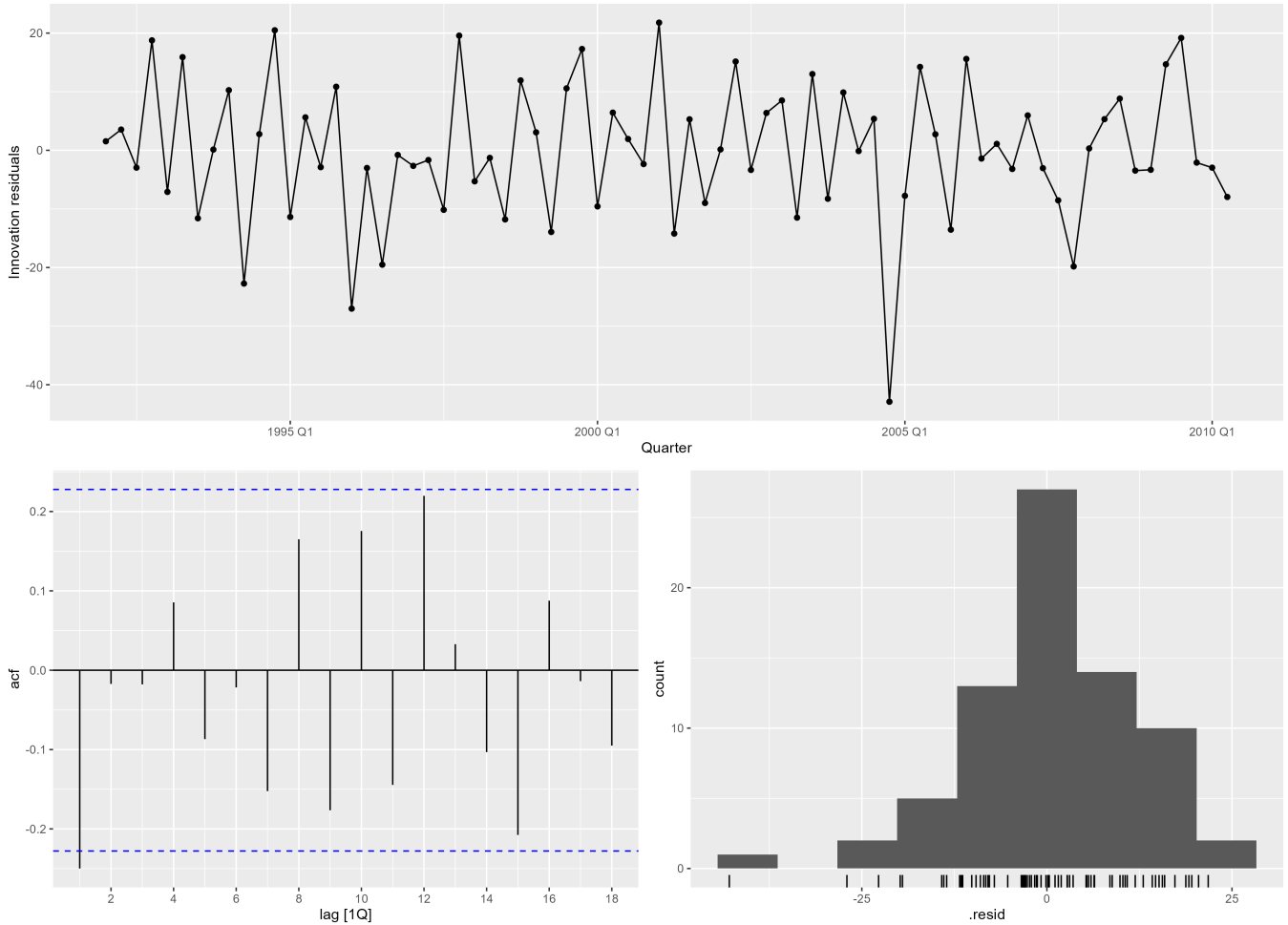
Example IV

```
Series: Beer
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-42.9029  -7.5995  -0.4594   7.9908  21.7895

Coefficients:
               Estimate Std. Error t value Pr(>|t|)
(Intercept)   441.80044    3.73353 118.333  < 2e-16 ***
trend()        -0.34027    0.06657  -5.111 2.73e-06 ***
season()year2 -34.65973    3.96832  -8.734 9.10e-13 ***
season()year3 -17.82164    4.02249  -4.430 3.45e-05 ***
season()year4  72.79641    4.02305  18.095  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 12.23 on 69 degrees of freedom
Multiple R-squared: 0.9243, Adjusted R-squared: 0.9199
F-statistic: 210.7 on 4 and 69 DF, p-value: < 2.22e-16
```

Example V

```
beer_fit |>
  gg_tsresiduals()

beer_fit |>
  augment() |>
  features(.innov, ljung_box, lag = 8) #white noise
```

Example VI



- This is not a bad residual diagnostics.

Example VII

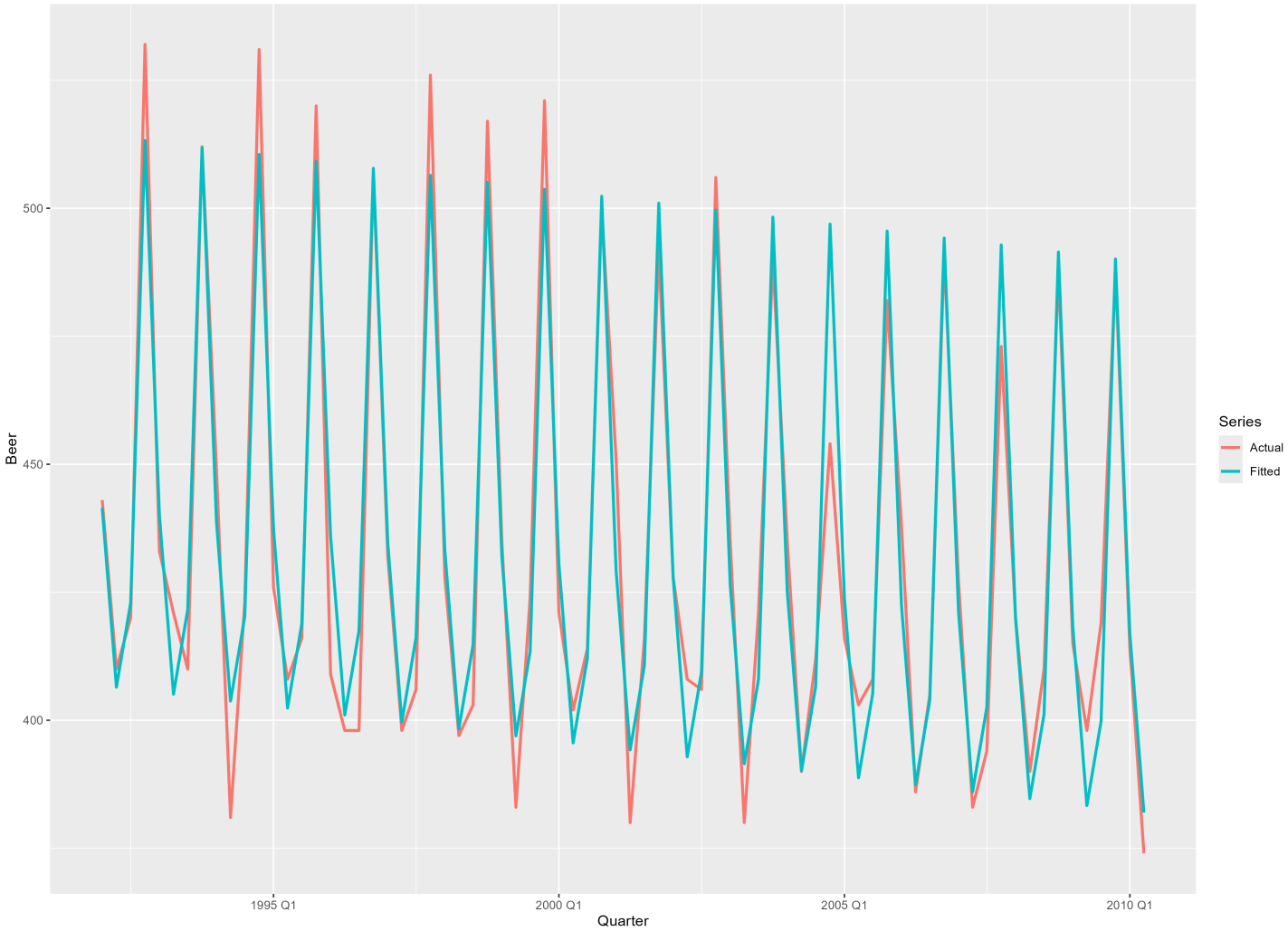
```
  .model                           lb_stat lb_pvalue
   <chr>                             <dbl>     <dbl>
1 TSLM(Beer ~ trend() + season())    10.4      0.240
```

Example VIII

- Let's look at the fitted values, how they match the actual data.

```
beer_fit |> augment() |>
  ggplot(mapping = aes(x = Quarter)) +
  geom_line(mapping = aes(y = Beer, colour = "Actual"), linewidth = 1) +
  geom_line(mapping = aes(y = .fitted, colour = "Fitted"), linewidth = 1) +
  guides(colour = guide_legend(title = "Series"))

beer_fit |> augment() |>
  ggplot() +
  geom_point(mapping = aes(x = Beer, y = .fitted, colour = factor(quarter(
      Quarter)))) +
  geom_abline(intercept = 0, slope = 1) +
  scale_color_manual(values = c("red", "blue", "green", "black")) +
  guides (colour = guide_legend("Title"))

beer_fit |> augment() |>
  ggplot() +
  geom_point(mapping = aes(x = .innov, y = .fitted, colour = factor(quarter(
      Quarter)))) +
  scale_color_manual(values = c("red", "blue", "green", "black")) +
  guides (colour = guide_legend("Title"))
```

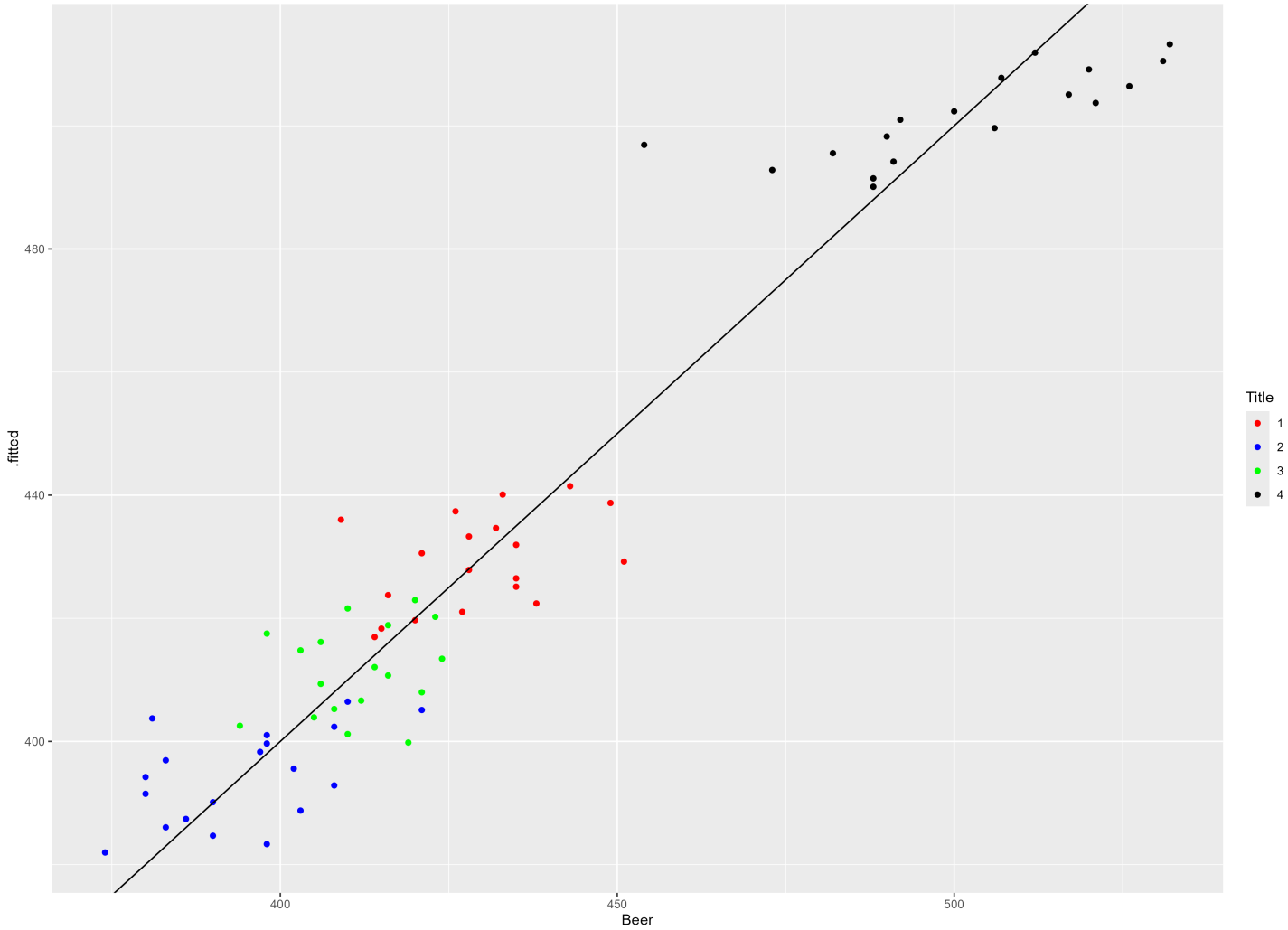
Example IX


Beer
500
450
400
1995 Q1
2000 Q1
2005 Q1
2010 Q1
Quarter
Series
Actual
Fitted

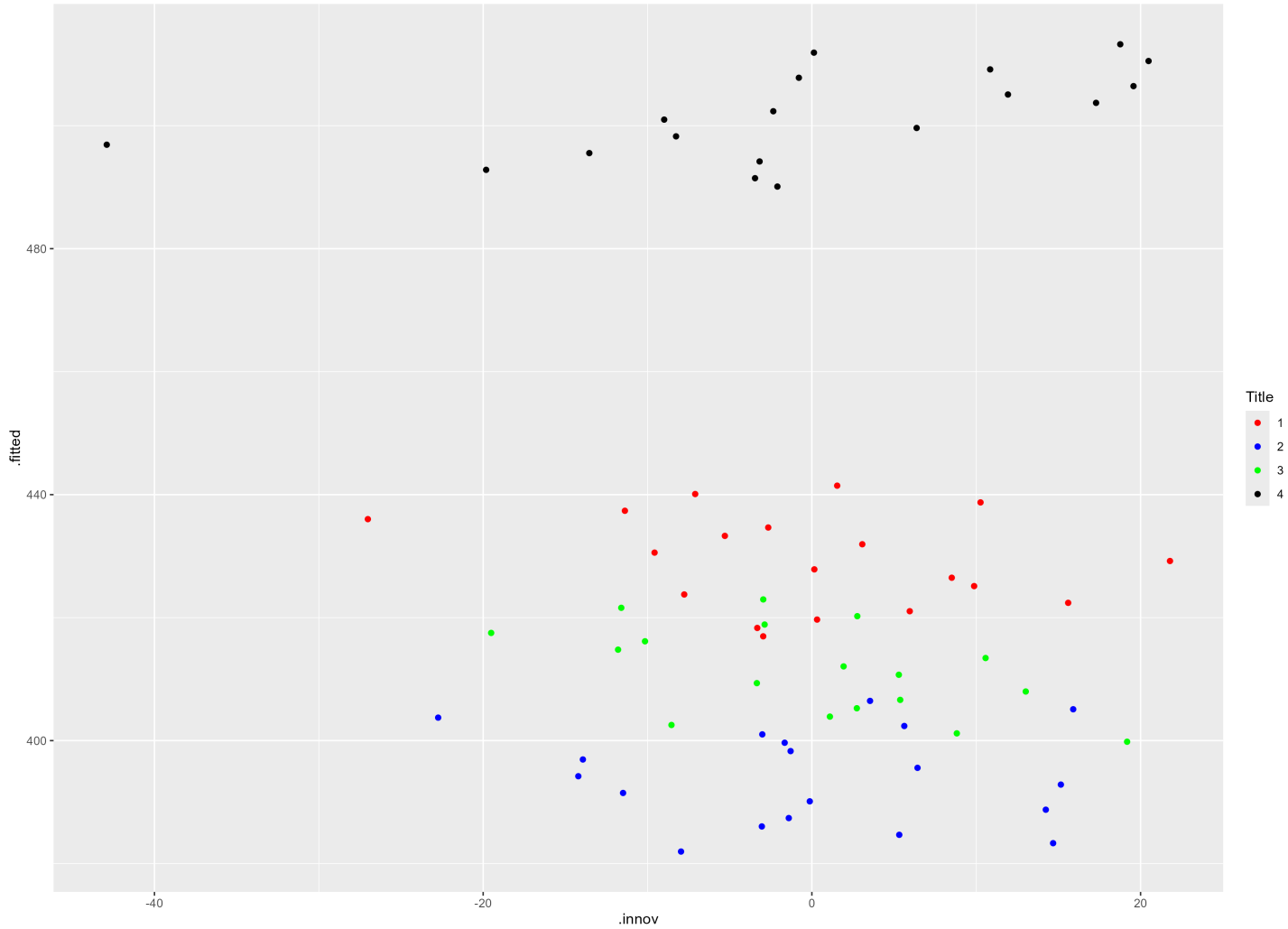
Example X

- It looks like a pretty good estimate. However, if we look at the peaks, we can see that at the beginning of the time period the model underforcasts the true value (fitted value is less than the actual), while at the end of the period, it overforecasts it.

Example XI



Example XII



Example XIII

- Pattern in the scatterplot of the fitted values vs. the actual data shows good fit.
- Lack of pattern in the scatterplot of the residuals vs. the fitted value also shows a good fit.

Example 2 I

- As a second example, let's look at the *vic_elec* time series again. This has recordings from 30 minutes intervals, information about demans, temperature and a column showing whether the current day is a holiday.

- We can have a model built to forecast Demand considering temperature and to use the holiday column.

```
vic_elec
vic_elec_model <- vic_elec |>
    model(TSLM(Demand ~ Temperature + Holiday))
vic_elec_model |>
    report()
```

Example 2 II

```
Series: Demand
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-1843.74  -669.59   -19.89   541.14  3723.80

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 4009.6452    11.0912  361.52   <2e-16 ***
Temperature   41.5731     0.6456   64.39   <2e-16 ***
HolidayTRUE -721.3234    22.0363  -32.73   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 835.9 on 52605 degrees of freedom
Multiple R-squared: 0.08596, Adjusted R-squared: 0.08592
F-statistic:  2473 on 2 and 52605 DF, p-value: < 2.22e-16
```

- We can see the coefficient for *HolidayTrue* dummy variable.

Example 2 III

- We might also consider that the actual day of the week is important as well. We can get this information with the *weekdays* function in R.

```
vic_elec |>
    mutate (Day = weekdays(Time)) -> vic_elec_day

vic_elec_model2 <- vic_elec_day |>
    model(TSLM(Demand ~ Temperature + Holiday + Day ))
vic_elec_model2 |>
report()
```

Example 2 IV

```
Series: Demand
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-1931.85  -532.76   -25.25   466.21  3463.01

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)  4195.9259    13.0588 321.309  < 2e-16 ***
Temperature    41.5492     0.5948  69.848  < 2e-16 ***
HolidayTRUE  -895.3166    20.4928 -43.689  < 2e-16 ***
DayMonday     -11.5789    12.5950  -0.919   0.3579
DaySaturday  -612.8982    12.5994 -48.645  < 2e-16 ***
DaySunday    -770.7426    12.5741 -61.296  < 2e-16 ***
DayThursday    75.4327    12.5832   5.995 2.05e-09 ***
DayTuesday     28.7184    12.5644   2.286   0.0223 *
DayWednesday   24.3154    12.5646   1.935   0.0530 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 769.9 on 52599 degrees of freedom
Multiple R-squared: 0.2247, Adjusted R-squared: 0.2245
F-statistic:  1905 on 8 and 52599 DF, p-value: < 2.22e-16
```

- We can see the coefficients for the weekdays except Friday.

Other dummy variables I

- There are other situations in which we can add dummy variables in our data set.
- **Intervention variables** - when something happened that influences the value of the forecast variable (very close to the idea of outliers). This can be of several types:
  - *spike* - the special effect lasts for one period only. The dummy takes 1 for this period and zero for the others (e.g., "We had a huge sales event").
  - *step* - an intervention that causes a level change. The step variable is 0 before the shift and 1 after it. (e.g., "We have introduced free shipping").
  - change of slope in the trend

Other dummy variables II

- **Trading days** - in case of monthly data, it might be worth adding a dummy with the number of trading days for that month. Alternatively, instead of just the number of trading days, we could have seven dummies denoting the number of Mondays, Tuesdays, etc. in that month.

Other dummy variables III

- **Lags** - for factors which might make their effect later (for example, advertising expenditure) it might be worth including the lagged values of that factor. (e.g. a dummy for advertising last month, a dummy for advertising two months ago, ...).

- **Easter** - Easter is a special holiday because it is not always on the same day (not even same month). We can use a dummy for the day(s) of Easter, or, in case of monthly data, we can add two dummies to show if Easter is in March or April.
  - **Obs.** In this case, we need two dummies, since technically we have three cases: Easter in March, Easter in April, and Easter in both (in which case the values in the dummy are proportionally split).

Other dummy variables IV

- There is a special case of dummy variables using Fourier terms, which is particularly useful if you have long periods (think about yearly periodicity in the vic_elec time series) or if you have non-integer periods (weekly data with yearly periods).

- The Fourier terms are pairs of terms, made of a sine term and a cosine term.

$$s_k(t) = sin(\frac{2 * \pi * k * t}{m})$$

$$c_k(t) = cos(\frac{2 * \pi * k * t}{m})$$

- where m is the seasonal period, which can be non-integer (e.g. 52.18 for weekly data).

Other dummy variables V

- The time series regression expression is then:

$$y_t = a + b * t + \sum_{k=1}^{K} [\alpha_k * s_k(t) + \beta_k * c_k(t)] + \epsilon_t$$

- K is a parameter of the model which should be chosen to minimize AICc (will be discussed later).
- For every value of k, a pair of terms will be estimated. The maximum number of such terms is the number of seasons - 1.
- In *R*, you can use the *fourier* special in TSLM to estimate such coefficients.

```
vic_elec |>
    model(TSLM(Demand ~ Temperature + fourier(period = 48, K = 4))) |> report()
```

Other dummy variables VI

```
Series: Demand
Model: TSLM

Residuals:
      Min        1Q    Median        3Q       Max
-2037.826  -380.667    -1.442   371.159  4070.622

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                     4543.775      9.267 490.301  < 2e-16 ***
Temperature                        7.480      0.543  13.774  < 2e-16 ***
fourier(period = 48, K = 4)C1_48  231.495      3.985  58.091  < 2e-16 ***
fourier(period = 48, K = 4)S1_48  683.120      4.313 158.378  < 2e-16 ***
fourier(period = 48, K = 4)C2_48  218.337      3.968  55.027  < 2e-16 ***
fourier(period = 48, K = 4)S2_48 -249.037      3.985 -62.495  < 2e-16 ***
fourier(period = 48, K = 4)C3_48   55.890      3.968  14.086  < 2e-16 ***
fourier(period = 48, K = 4)S3_48 -142.430      3.968 -35.897  < 2e-16 ***
fourier(period = 48, K = 4)C4_48 -108.622      3.968 -27.375  < 2e-16 ***
fourier(period = 48, K = 4)S4_48   26.441      3.968   6.664 2.69e-11 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 643.5 on 52598 degrees of freedom
Multiple R-squared: 0.4583, Adjusted R-squared: 0.4583
F-statistic:  4945 on 9 and 52598 DF, p-value: < 2.22e-16
```

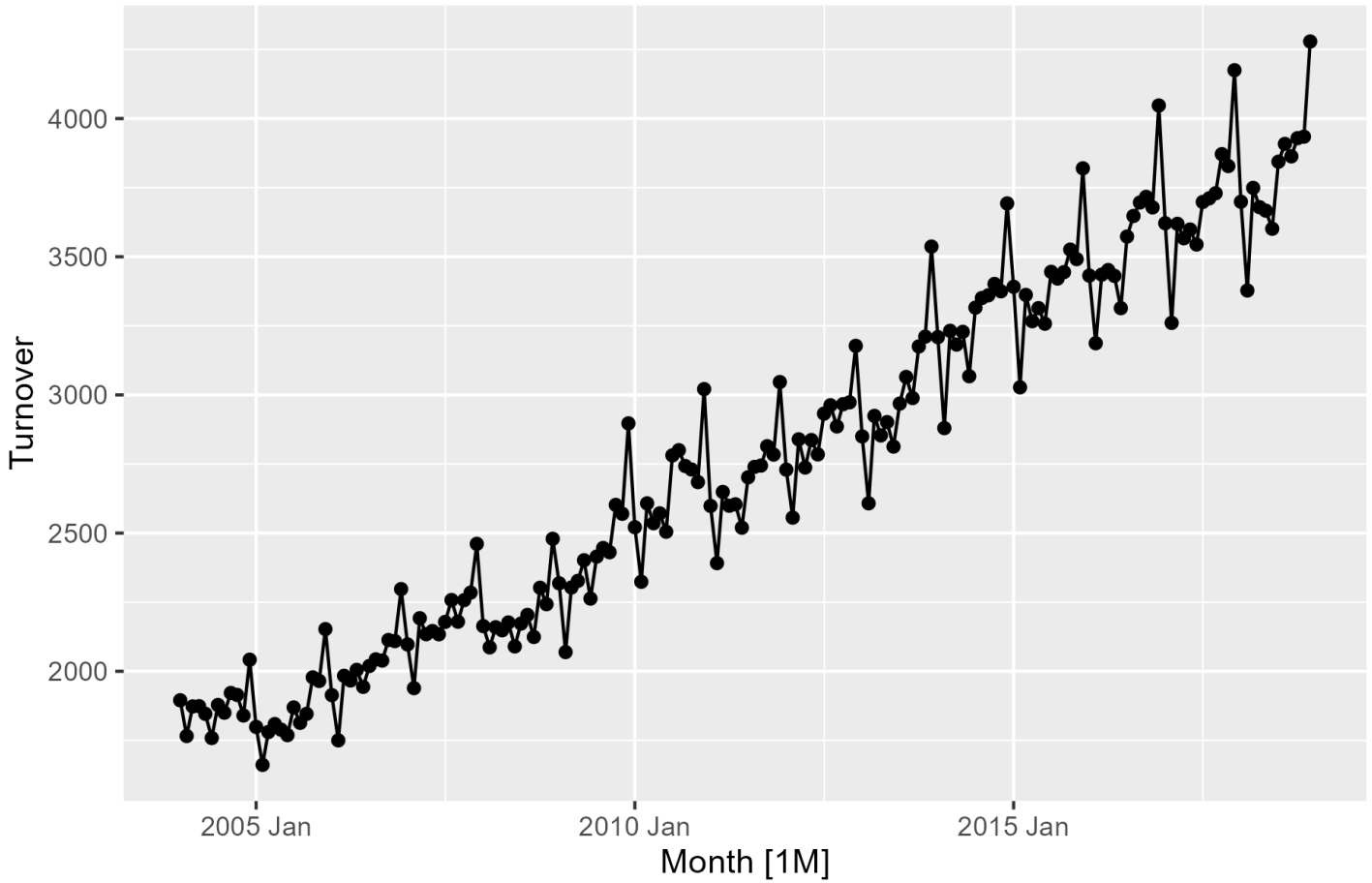
Another example I

- Let's consider another example, the eating out expenditure from the Australian retail (*aus_retail*) time series.
- It is monthly data, so we can have at most 11 seasonal dummies. We will build 6 models with different values of K for the fourier terms (1 to 6) and plot the data, the fitted values and forecasts for the next 2 years.

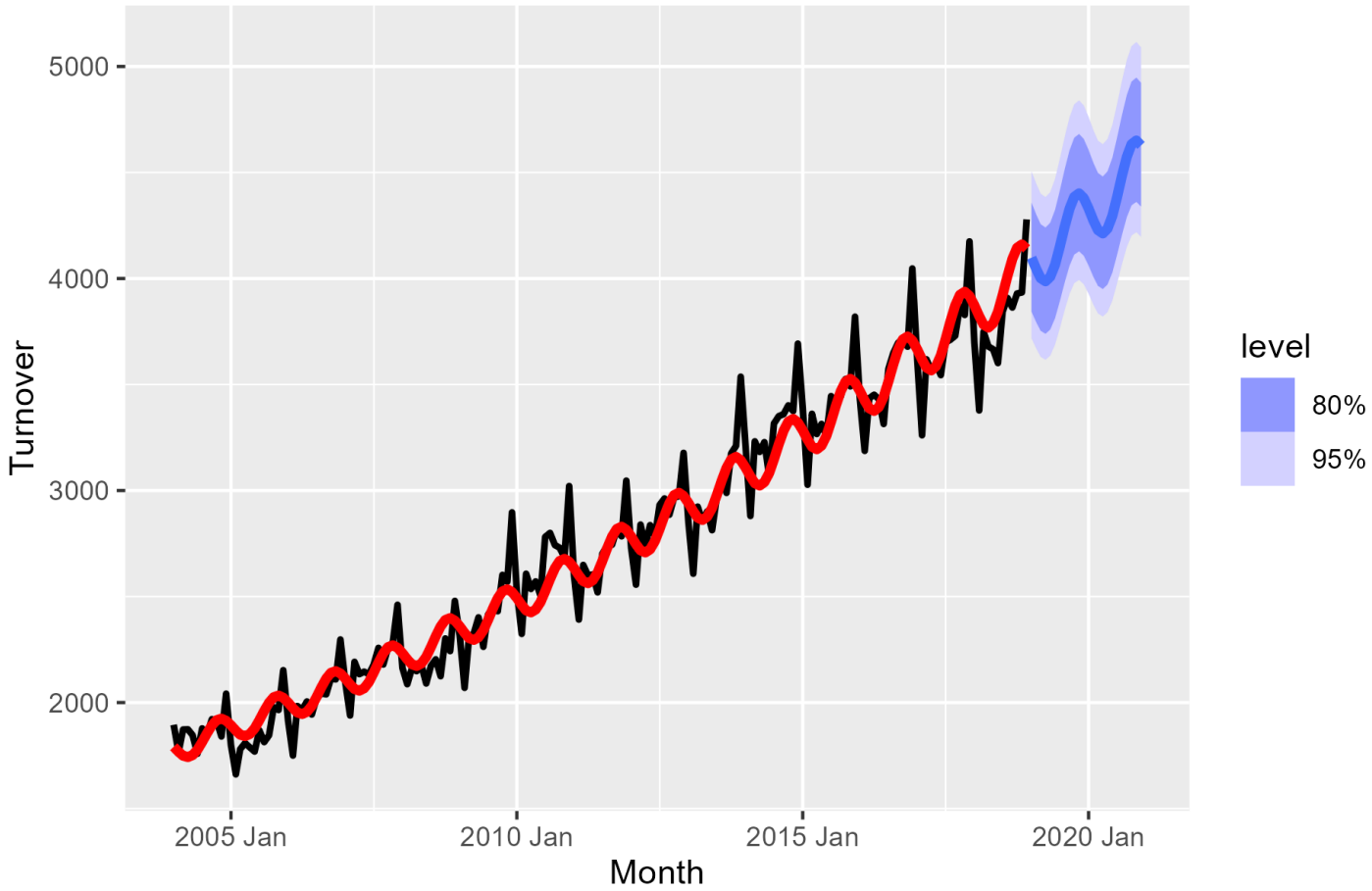
Another example II

```
aus_cafe <- aus_retail |>
  filter(Industry == "Cafes, restaurants and takeaway food services",
         year(Month) >= 2004, year(Month) <= 2018) |>
  summarise(Turnover = sum(Turnover))
aus_cafe
aus_cafe |> autoplot() + geom_point()
ggsave("aus_cafe_plot.png")

fit <- aus_cafe |>
  model(
    K1 = TSLM(log(Turnover) ~ trend() + fourier(K = 1)),
    K2 = TSLM(log(Turnover) ~ trend() + fourier(K = 2)),
    K3 = TSLM(log(Turnover) ~ trend() + fourier(K = 3)),
    K4 = TSLM(log(Turnover) ~ trend() + fourier(K = 4)),
    K5 = TSLM(log(Turnover) ~ trend() + fourier(K = 5)),
    K6 = TSLM(log(Turnover) ~ trend() + fourier(K = 6)),
  )

fit |> forecast(h = 24) -> cafe_forecast
fit |> augment() -> fit_aug

cafe_forecast |> filter(.model == "K1") |>
  autoplot(linewidth = 1.5) +
  autolayer(aus_cafe, Turnover, linewidth = 1) +
  autolayer(fit_aug |> filter(.model == "K1"), .fitted, color = "red", linewidth
      = 1.5)
```

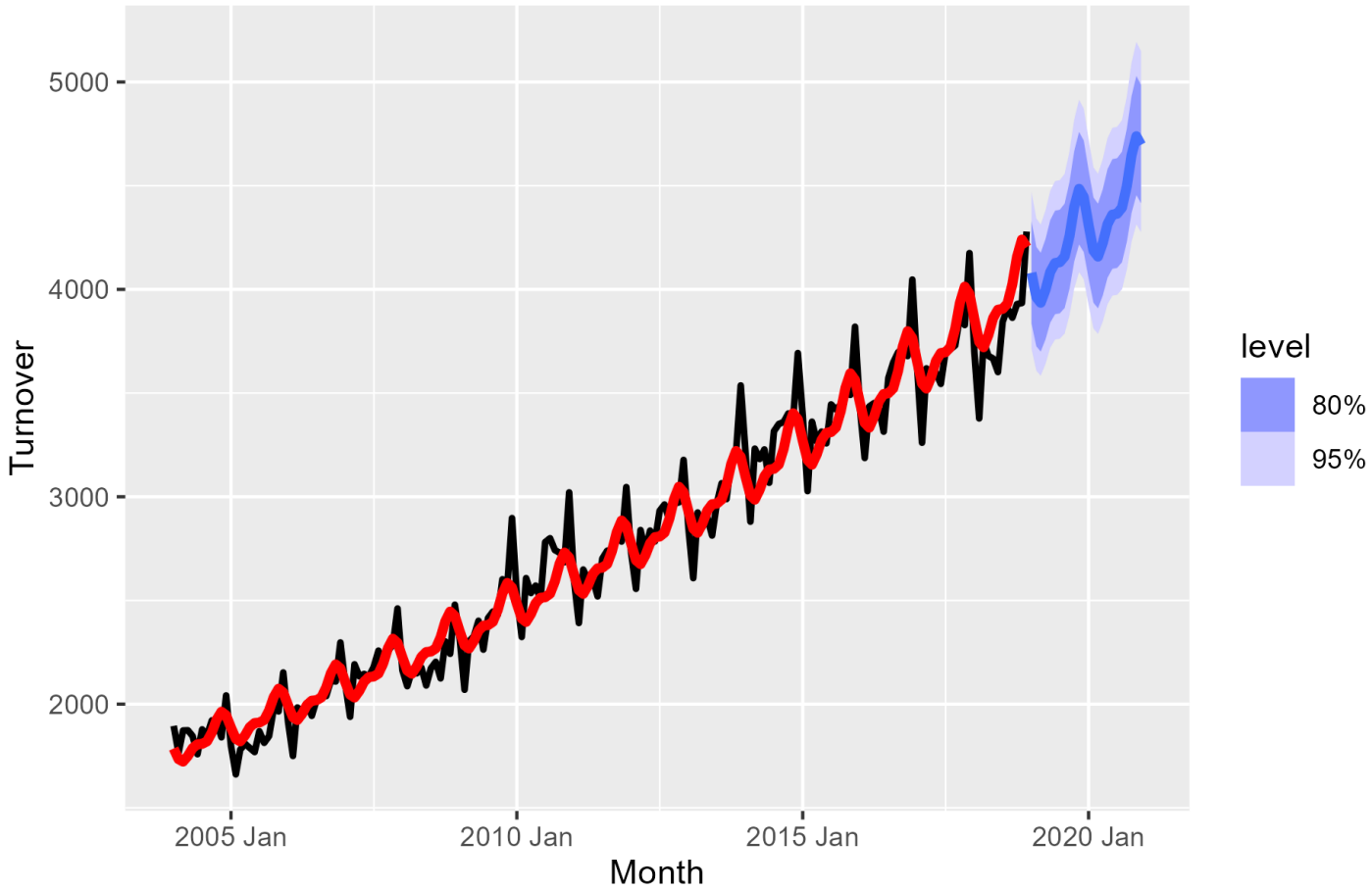
Another example III



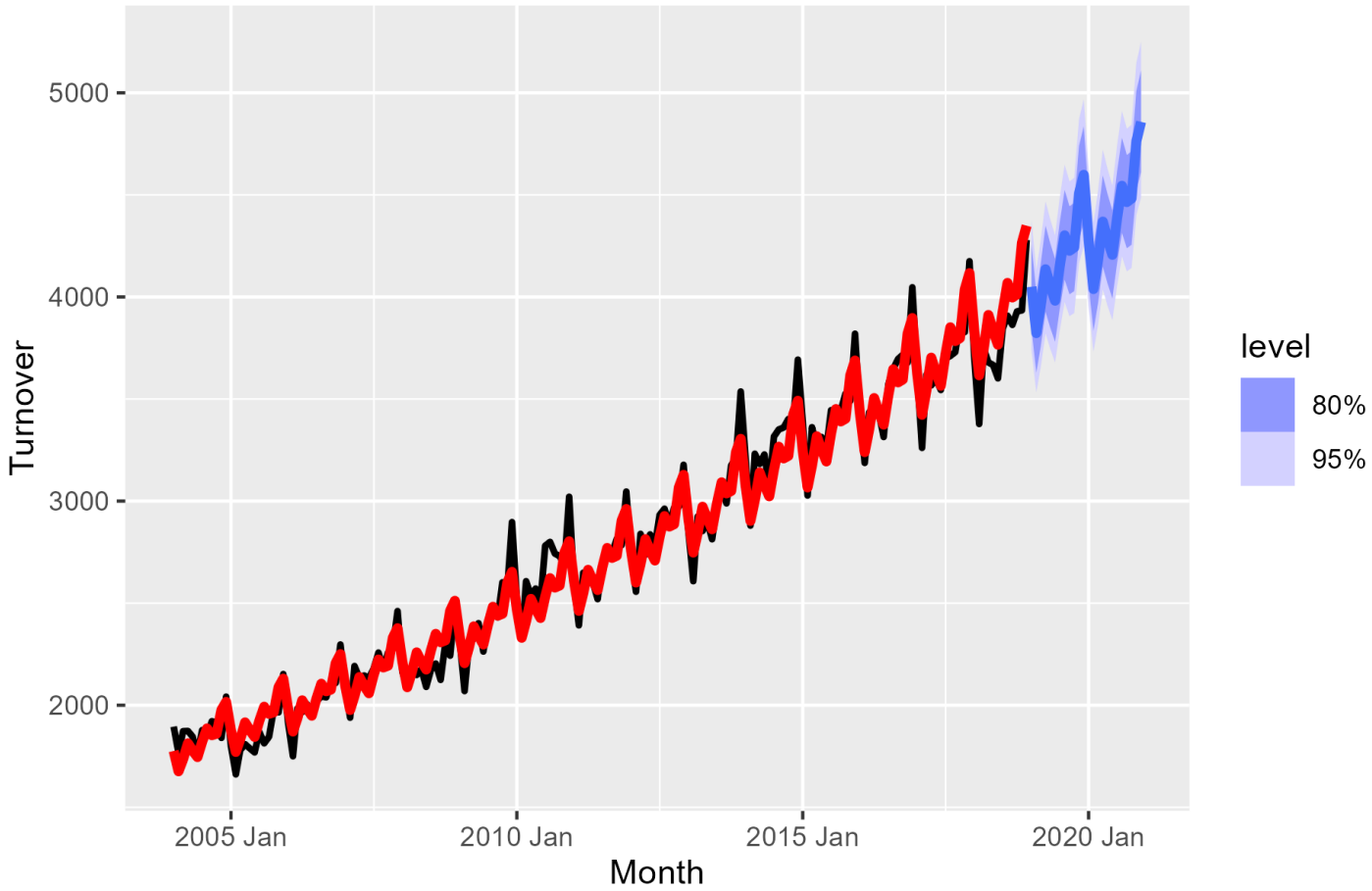
Another example IV



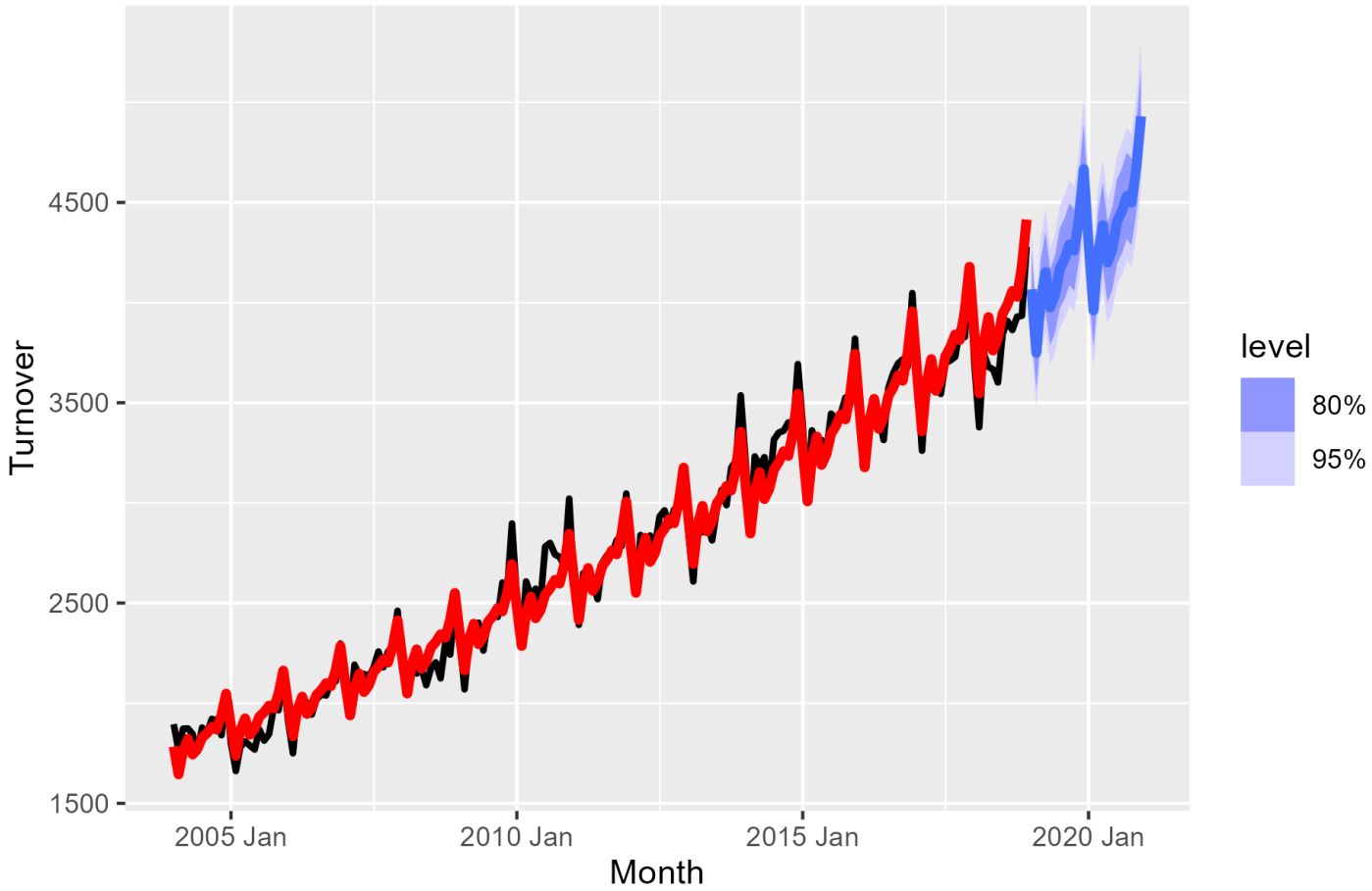
Another example V



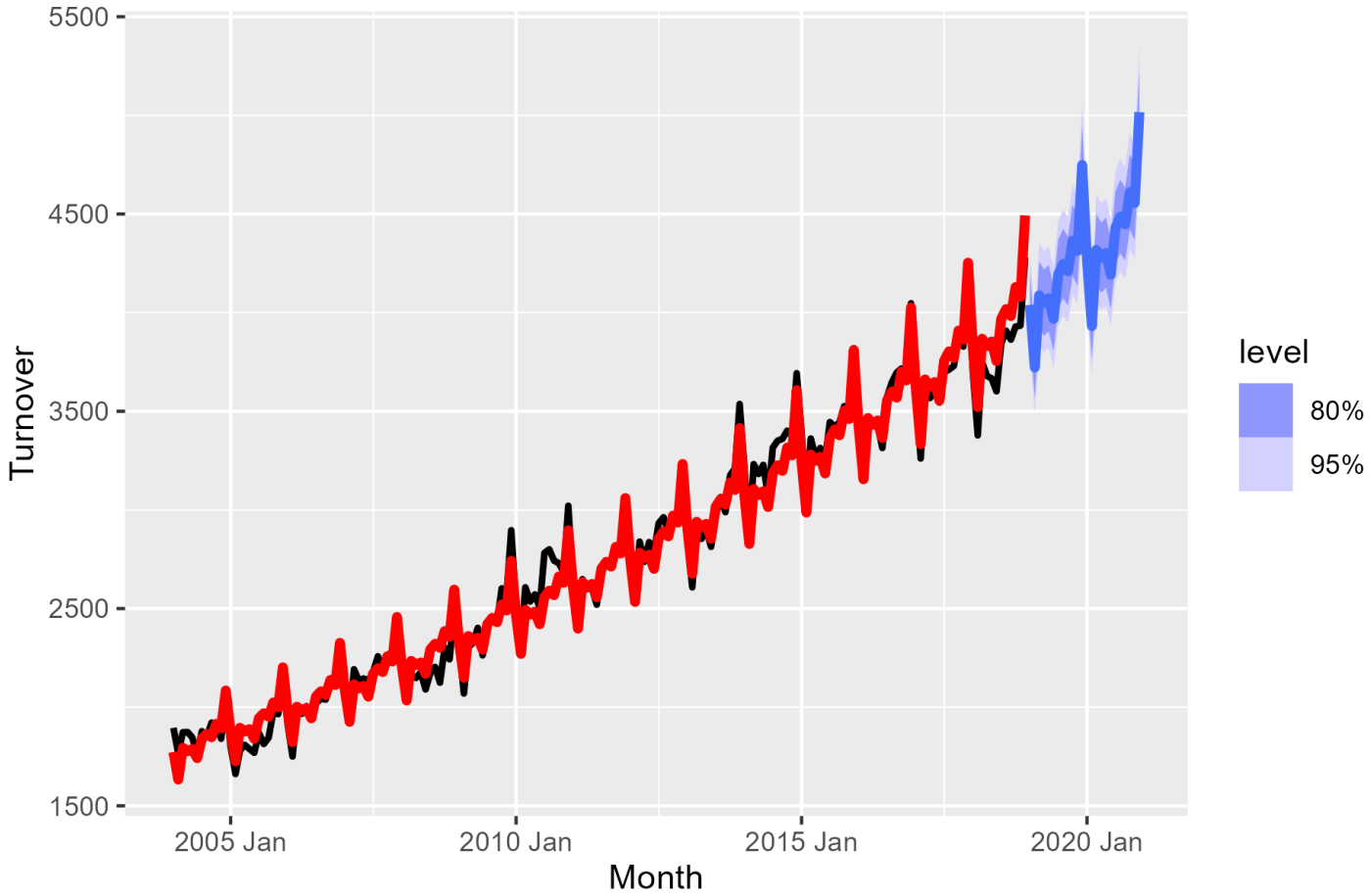
Another example VI



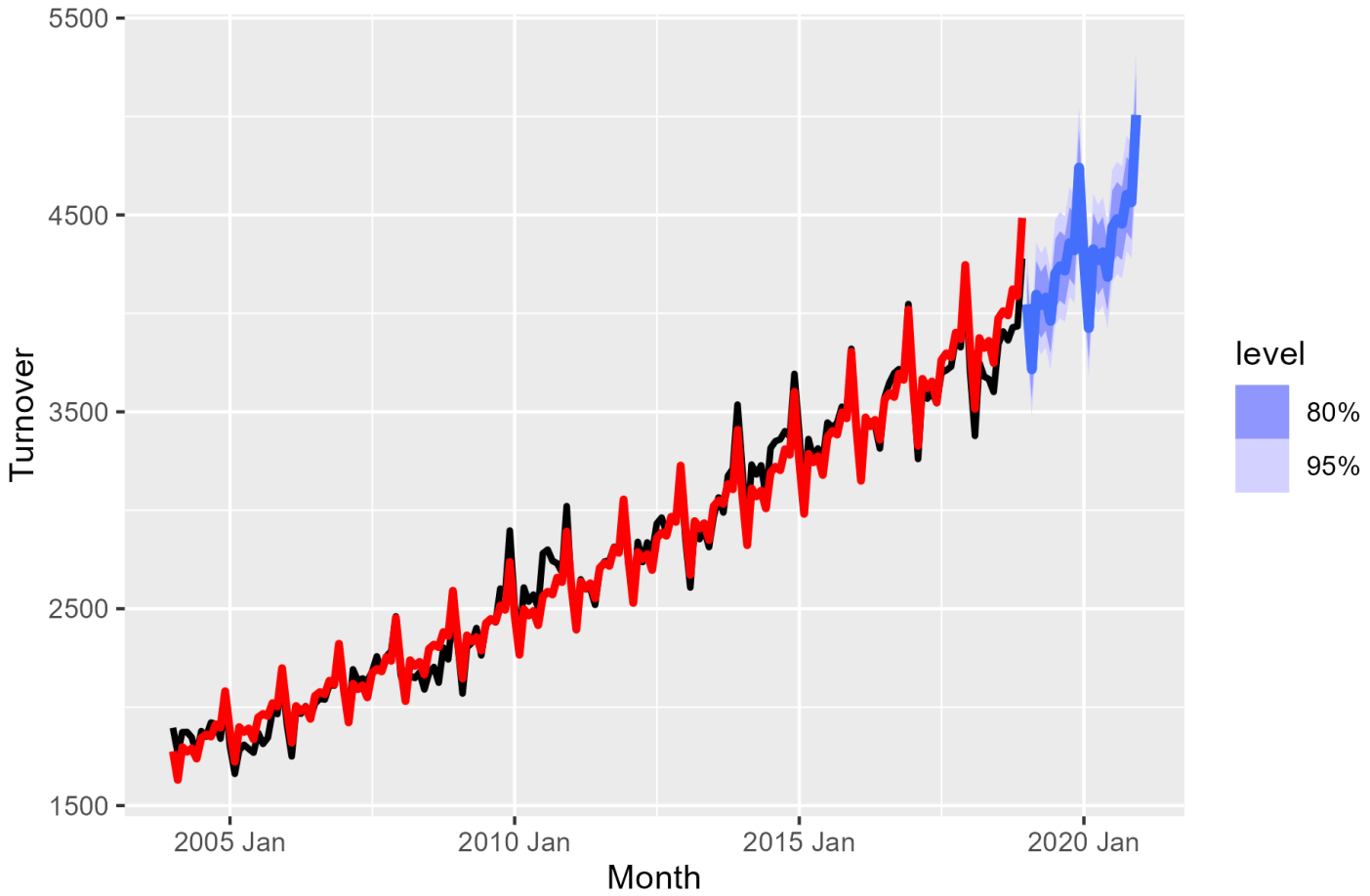
Another example VII



Another example VIII



Another example IX



Selecting predictors I

- Sometimes, we have a lot of possible predictors (especially when we consider dummies) and we need a strategy to select the ones to be used in the regression model.
- A **bad** strategy is to create scatterplots of each possible predictor variable and the forecast variable and drop everything where no relationship can be seen. Sometimes the relationship cannot be identified from the plot.
- Another **bad** strategy is to build a model including all predictors and look at their p-values (in the output of the *report* function) and discard predictors with small p-values.
- A **good** approach is to use the the so-called *measures of predictive accuracy*, which can be seen using the *glance* function on a model.

Selecting predictors II

```
glance(fit) |>
    select(.model, r_squared, adj_r_squared, CV, AICc)
```

```
# A tibble: 6 × 5
  .model r_squared adj_r_squared      CV   AICc
  <chr>      <dbl>         <dbl>   <dbl>  <dbl>
1 K1         0.962         0.962 0.00238 -1085.
2 K2         0.966         0.965 0.00220 -1099.
3 K3         0.976         0.975 0.00157 -1160.
4 K4         0.980         0.979 0.00138 -1183.
5 K5         0.985         0.984 0.00104 -1234.
6 K6         0.985         0.984 0.00105 -1232.
```

- Out of the selected 5 measures, *adjusted* R^2 should be maximized while the other 4 should be minimized.

Adjusted R^2

- The R^2 reported by the *report* function for a model was computed on the same data which was used for building the model. It measures how well the model fits the historical data, but it does not tell us how well the model will forecast.
- Also, it was observed that by adding predictors to the model the value of R^2 will increase, even if the predictor is useless.
- A measure which overcomes these problems is the *adjusted* R^2, denoted by $\bar{R}^2$.

$$\bar{R}^2 = 1 - (1 - R^2) * \frac{T - 1}{T - k - 1}$$

- where T is the number of observations and k is the number of predictors in the model.

CV

- CV - comes from cross-validation.
- We have discussed in the previous lecture how to do cross-validation for time series, however, for regression it is possible to do classical *leave-one-out* cross validation, which is done in the following way:
  - Remove observation t from the data set and build a regression model on the rest of the observations. Predict the value of the removed observation and compute the corresponding error e_t.
  - Repeat for all observations, getting an error for each.
  - Compute the MSE of the errors. This is the value of CV.

Akaike's Information Criterion (AIC)

- AIC is defiend in the following way:

$$AIC = T * log(\frac{SSE}{T}) + 2 * (k + 2)$$

- where T is the number of observations, k is the number of predictors and SSE is the sum of the squared errors, computed in the following way:

$$SSE = \sum_{t=1}^{T} e_t^2$$

- For small values of T the AIC tends to select models with too many predictors, so a corrected version of AIC was proposed:

$$AIC_c = AIC + \frac{2(k+2)(k+3)}{T-k-3}$$

Schwarz's Bayesian Information Criterion (BIC)

- A related measure is BIC, computed in the following way:

$$BIC = T * log(\frac{SSE}{T}) + (k + 2) * log(T)$$

- BIC penalizes models with more predictors more than AIC does, so BIC will select either the same method as AIC or one with fewer terms.

Which one to use?

- While $\bar{R}^2$ is frequently used, it tends to find models with many predictors.
- AICc, AIC and CV are recommended to be used, since they have forecasting as their objective. If T is large enough, they will all lead to the same model.

Selecting the predictors

- If possible, all possible combinations of predictor variables should be fitted and then the best one can be selected.
- However, if the number of possible predictors is large, this might not be an option. In this case a strategy is needed to limit the number of combinations to experiment with.
- There are three approaches:
  - *backwards stepwise regression* - build a model on all predictors. Eliminate one predictor at a time. Keep the new model with the best performance (if it is better than the one with all predictors). Repeat until no more improvement happens.
  - *forward stepwise regression* - start with a model having only an intercept. Add one-by-one all predictors, keep the best model. Repeat until no further improvement is possible.
  - **hybrid** procedure - start with some predictors already then either include new ones or remove existing ones.
- The above presented stepwise approaches do not guarantee that the best model is found, but they almost always find a good model.

How to forecast? I

- We saw many things to do/check considering the fitted values. But how to forecast *future* values?
- There are two types of forecasts: *ex-ante* and *ex-post* forecasts.
- To understand the difference between them, let's look at the example with the Consumption again and assume we have the model where Consumption is computed using the other four attributes.

How to forecast? II

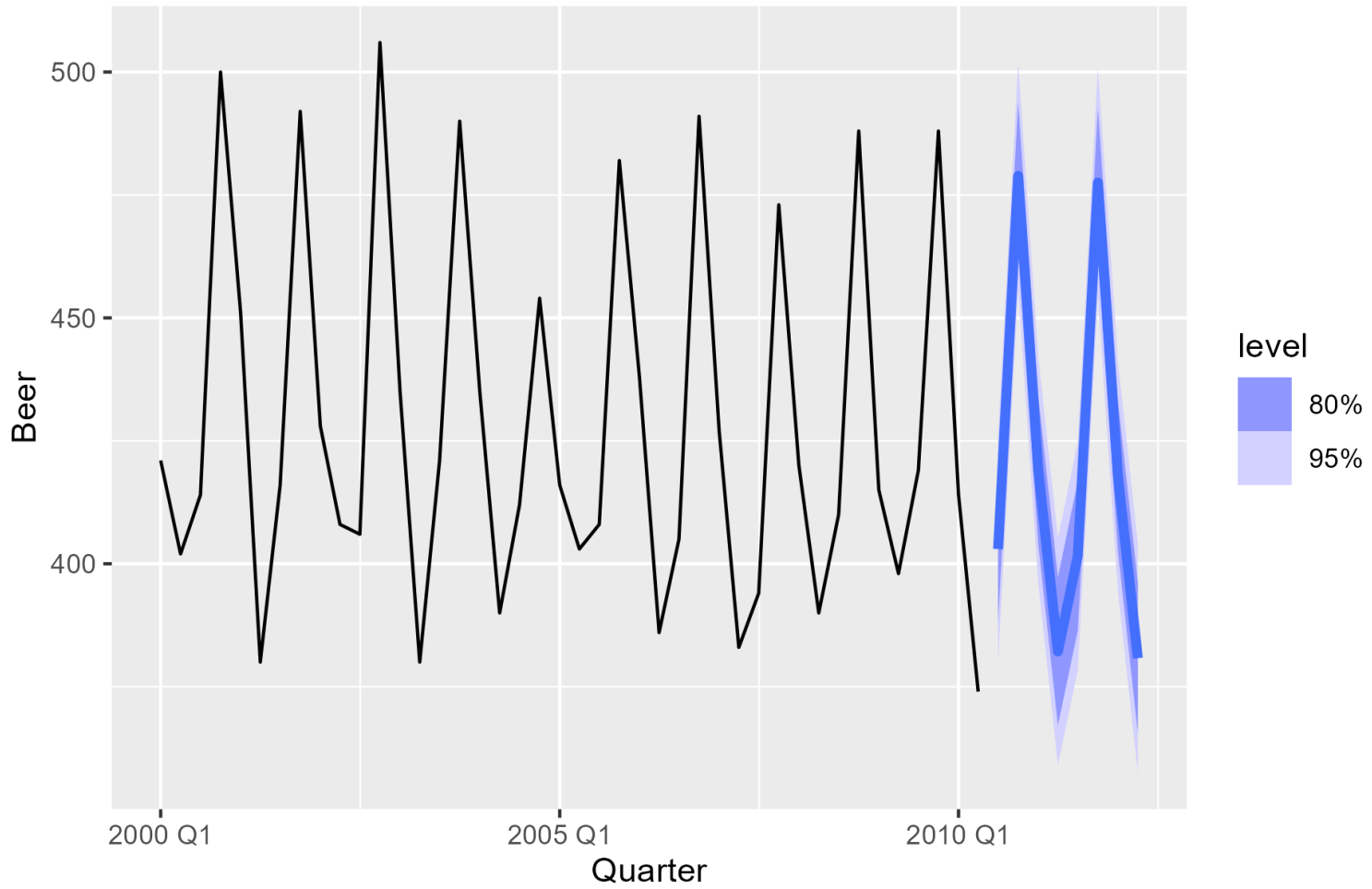
- In **ex-ante** forecast, we build the model using only information which is available in advance. In our case it means to build a model from all data available (up until 1992 Q2). This means that when we want to forecast Consumption, we first need to forecast the other attributes. This can be done with simple or more sophisticated methods.
- In **ex-post** forecasting, later information about the predictors is considered. This is not genuine forecasting, but might be useful as analysis.

Example

- Let's consider the beer production again.

```
recent_beer |>
    model(TSLM(Beer ~ trend()  + season())) |>
    forecast() |>
    autoplot(recent_beer, linewidth = 1.5)
```

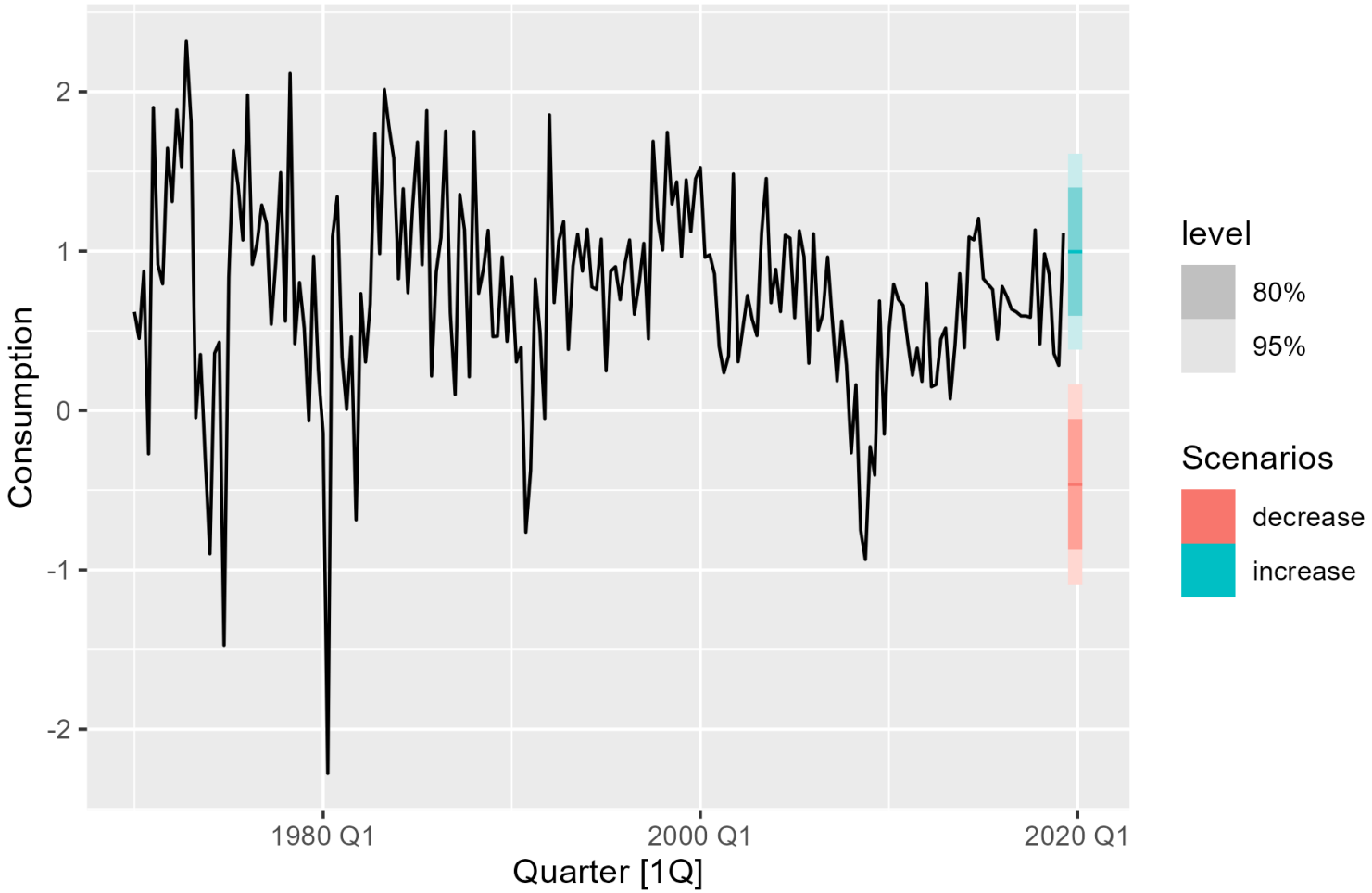
- The *trend* and the *season* are dummy variables, and they are known in advance, so we have genuine forecasts.


500
450
400
Beer
2000 Q1
2005 Q1
2010 Q1
Quarter
level
80%
95%

Scenario-based forecasting

- In this version, the predictor variables are estimated by simply considering different scenarios for them.
- For example, how does a future increase or decrease in Income and Savings influence the value of Consumption?

```
consumptionFit <- us_change |>
  model(all = TSLM(Consumption ~ Income + Savings + Unemployment))

future_scenarios <- scenarios(
  increase = new_data(us_change, 4) |>
    mutate(Income = 1, Savings = 0.5, Unemployment = 0),
  decrease = new_data(us_change, 4) |>
    mutate(Income=-1, Savings = -0.5, Unemployment = 0),
  names_to = "Scenarios"
  )

consumptionForecast <- forecast(consumptionFit, new_data = future_scenarios)

us_change |>
  autoplot(Consumption) +
  autolayer(consumptionForecast)
```


Consumption
2
1
0
-1
-2
1980 Q1
2000 Q1
2020 Q1
Quarter [1Q]
level
80%
95%
Scenarios
decrease
increase

Non-linear regression I

- Sometimes a linear model is not enough and we need a non-linear model.
- The simplest solution is to transform the predictor and/or the forecast variable (for example, using the log transformation) and then use linear regression

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

- The above example is called **log-log** functional form, because we took the logarithm from both terms.
- There is **log-linear** form, when only the forecast variable is transformed and **linear-log** form when only the predictor variable is transformed.
- One possible issue with the logarithm is that the values have to be greater than zero. If x contains 0, we can use $log(x+1)$ as transformation, with the nice property that the transformed value will be 0 when x is 0.

Non-linear regression II

- When a simple logarithmic transformation is not enough, we can consider a more general non-linear function f:

$$y_t = f(x_t) + \epsilon_t$$

- One of the simplest cases is when we consider that f is *piecewise linear*, meaning that it is linear in segments, but there are points when the slope of the line changes. These points are called **knots**.
- Assuming that there is only one *knot* at point c, we can deal with it, in the following way:
  - We introduce $x_1 = x$ (just renaming it to have indexes)
  - We introduce $x_2 = (x - c)_+$, where $(x - c)_+$ is $x - c$ if the value is positive ($x \geq c$) and it is zero otherwise ($x < c$).
- If we have more knots, we can add more attributes in the similar manner, however, finding the knot points is not an easy thing.

Forecasting with a non-linear trend

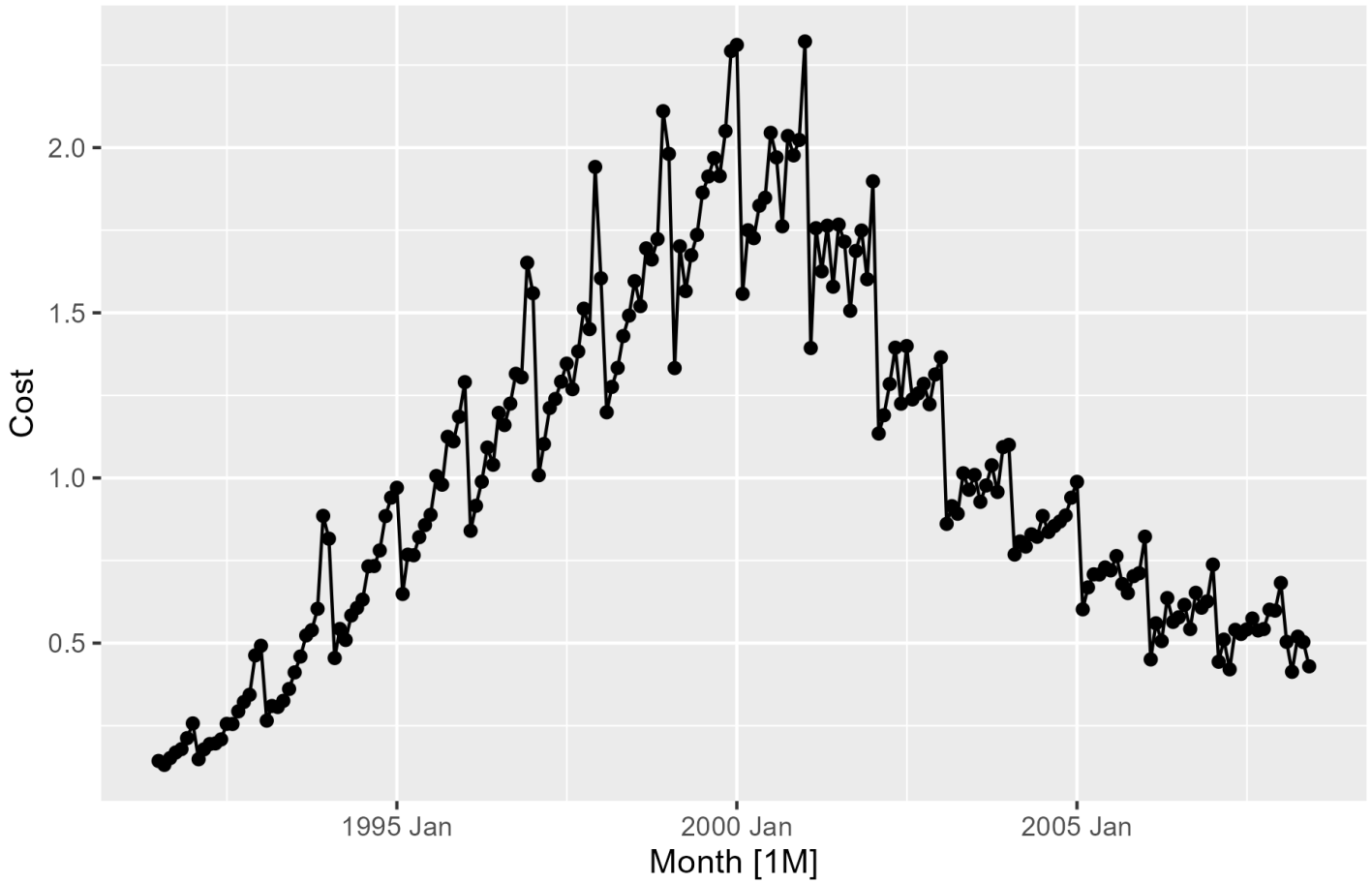
- We have seen that we can use $x_{1,t} = t$ as a predictor variable to account for trend.
- If we think that the trend is not linear, one thing that we could do is to use quadratic or higher order trends, but this is not recommended.
- It is better to use piecewise linear models which bend at some point.
- If we think that the trend bends at time τ, we can do the following:
  - Use $x_{1,t} = t$
  - Introduce $x_{2,t} = (t - \tau)_+$, which is $(t - \tau)$ if the value is positive and 0 otherwise.

Example 1 I

- Remember the vitamin prescription data from Australia? It had one increasing trend and then at a given point the trend became decreasing.

```
vitamins <- PBS |>
  filter(ATC2 == "A11") |>
  select(Month, Concession, Type, Cost) |>
  summarize(TotalCost = sum(Cost)) |>
  mutate(Cost = TotalCost / 1000000)

autoplot(vitamins, Cost) +
  geom_point()
```

Example 1 II



Example 1 III

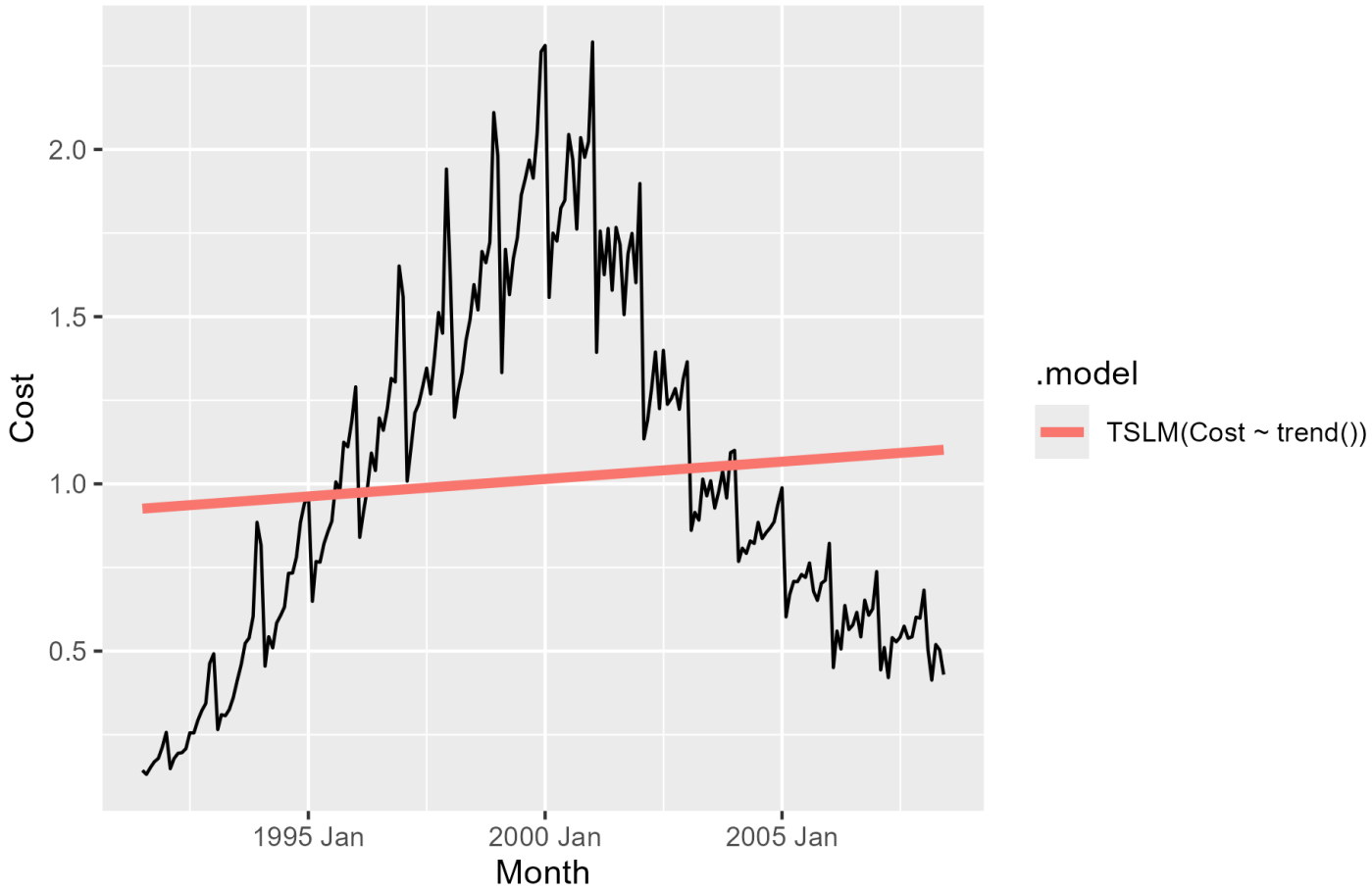
```
vitamins_model1 <- vitamins |>
  model(TSLM(Cost ~ trend()))
vitamins_model1 |> report()
```

```
Series: Cost
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-0.79551 -0.46137 -0.09127  0.36100  1.29657

Coefficients:
             Estimate Std. Error t value Pr(>|t|)
(Intercept) 0.9251069  0.0755289  12.248   <2e-16 ***
trend()     0.0008671  0.0006389   1.357    0.176
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.5374 on 202 degrees of freedom
Multiple R-squared: 0.009035, Adjusted R-squared: 0.00413
F-statistic: 1.842 on 1 and 202 DF, p-value: 0.17626
```

Example 1 IV

```
vitamins |> ggplot(mapping = aes(x = Month, y = Cost)) +
  geom_line() +
  geom_line(data = fitted(vitamins_model1), aes(y = .fitted, colour = .model),
      linewidth = 1.5)
```

Example 1 V


2.0
1.5
1.0
0.5
Cost
1995 Jan
2000 Jan
2005 Jan
Month
.model
TSLM(Cost ~ trend())

Example 1 VI

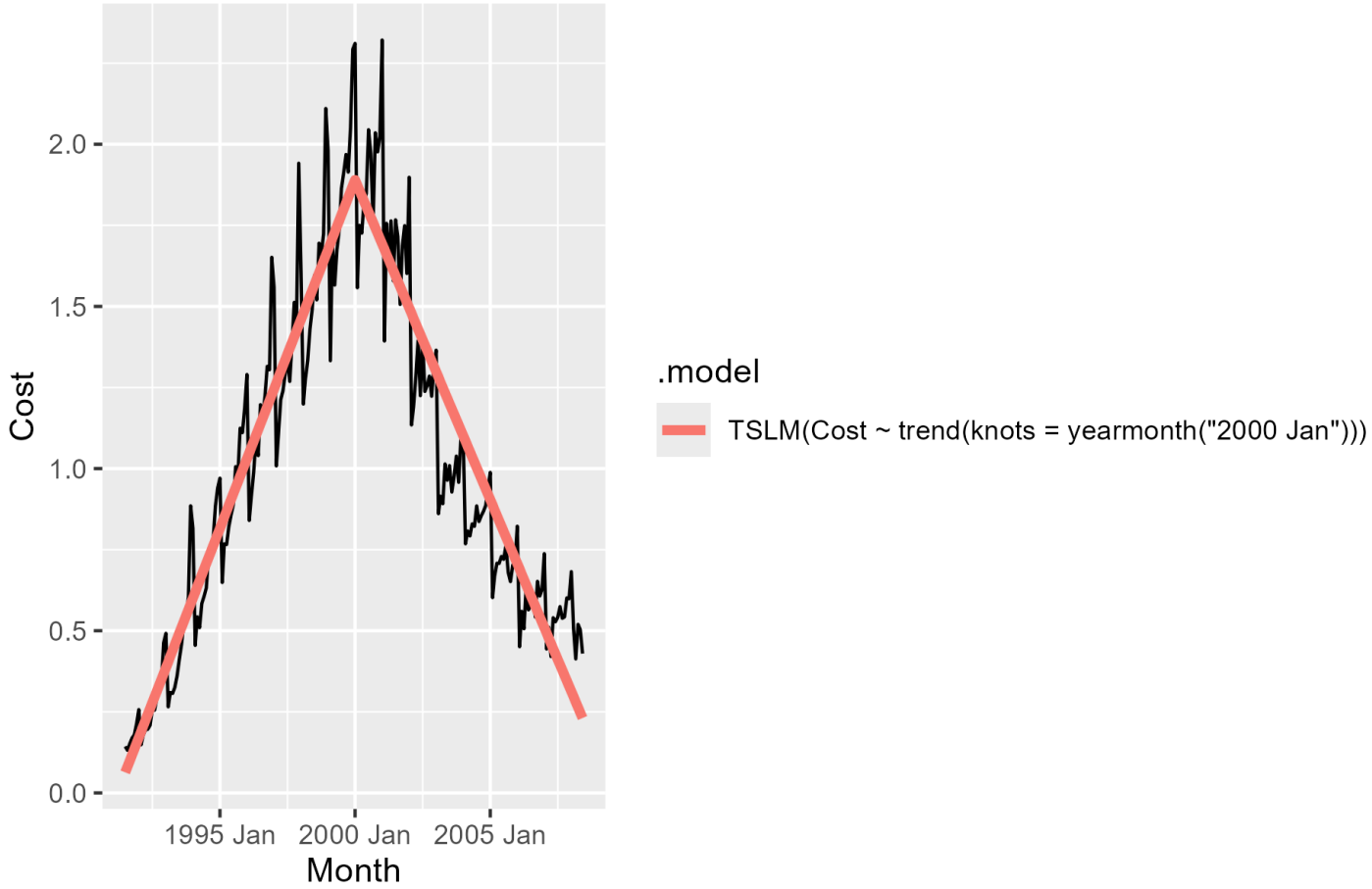
```
vitamins_model2 <- vitamins |>
  model(TSLM(Cost ~ trend(knots = yearmonth("2000 Jan"))))
vitamins_model |> report()
```

```
Series: Cost
Model: TSLM

Residuals:
     Min       1Q   Median       3Q      Max
-0.42113 -0.10839 -0.01503  0.09709  0.62865

Coefficients:
                                                 Estimate Std. Error t value Pr(>|t|)
(Intercept)                                     0.0454080  0.0327238   1.388    0.167
trend(knots = yearmonth("2000 Jan"))trend       0.0179070  0.0004649  38.515   <2e-16 ***
trend(knots = yearmonth("2000 Jan"))trend_103 -0.0343322  0.0008366 -41.037   <2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 0.1759 on 201 degrees of freedom
Multiple R-squared: 0.8943, Adjusted R-squared: 0.8933
F-statistic: 850.6 on 2 and 201 DF, p-value: < 2.22e-16
```

Example 1 VII

```
vitamins |> ggplot(mapping = aes(x = Month, y = Cost)) +
  geom_line() +
  geom_line(data = fitted(vitamins_model2), aes(y = .fitted, colour = .model))
```

Example 1 VIII

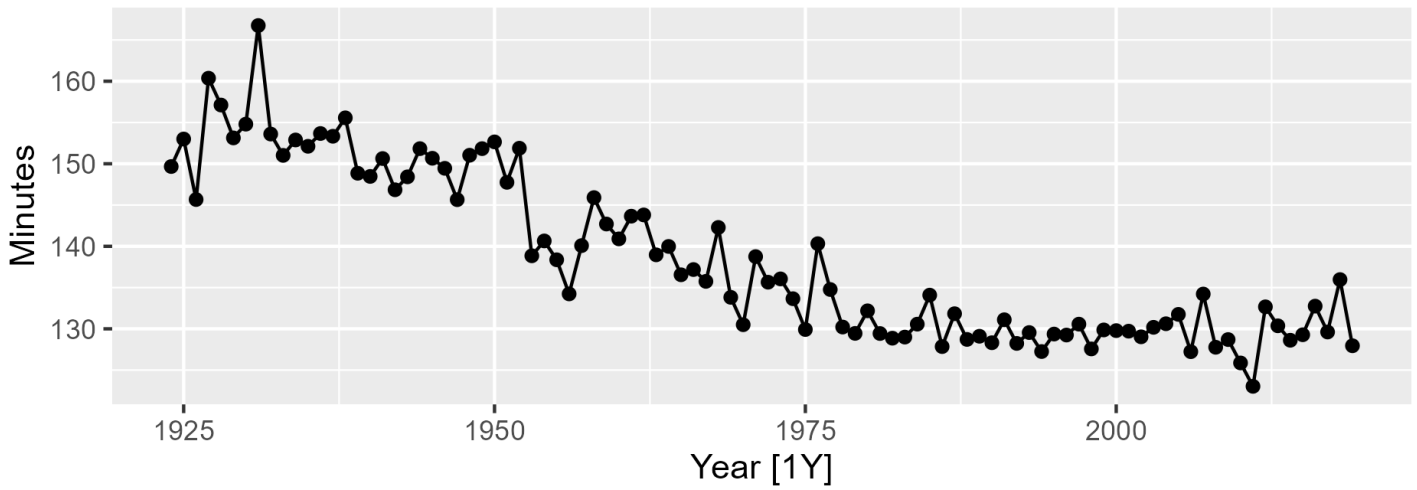


Example2 I

- Let's look at the winning times for men at the Boston marathon after 1924.

```
boston_marathon |>
  filter(Year >= 1924) |>
  filter(Event == "Men's open division") |>
  mutate(Minutes = as.numeric(Time)/60) ->boston_men

boston_men
boston_men |> autoplot(Minutes) +
    geom_point()

boston_model <- boston_men |>
  model(linear = TSLM(Minutes ~ trend()),
        logarithmic = TSLM(log(Minutes) ~ trend()),
        piecewise = TSLM(Minutes ~ trend(knots = c(1950, 1980)))
  )
boston_model |>
  accuracy() |>
  arrange(MAE)
```

Example2 II



Event	.model	.type	ME	RMSE	MAE	MPE	MAPE	MASE	RMSSE	ACF1
<fct>	<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1 Men's open division	piecewise	Training	-2.66e-15	3.42	2.48	-0.0573	1.76	0.723	0.743	0.0901
2 Men's open division	loga	Training	7.38e- 2	4.42	3.51	-0.0501	2.53	1.02	0.959	0.453
3 Men's open division	linear	Training	-2.96e-16	4.55	3.62	-0.100	2.62	1.05	0.987	0.483

Example2 III

```
boston_forecast <- boston_model |> forecast(h = 10)
boston_forecast

boston_men |>
  autoplot(Minutes) +
  autolayer(boston_forecast) +
  geom_line(data = fitted(boston_model), aes(y = .fitted, colour = .model))
```