FORECASTING AND PREDICTIVE MODELING

LECTURE 8

Lect. PhD. Oneț-Marian Zsuzsanna

Babeș - Bolyai University
Computer Science and Mathematics Faculty

2024 - 2025

In Lecture 7

- Time series features
- The forecasting workflow
  - Simple forecasting methods (mean, naive, seasonal naive, drift)
  - Fitted values, residuals and residual diagnostics
  - Autocorrelation tests (Ljung-Box and Box-Pierce)

Today

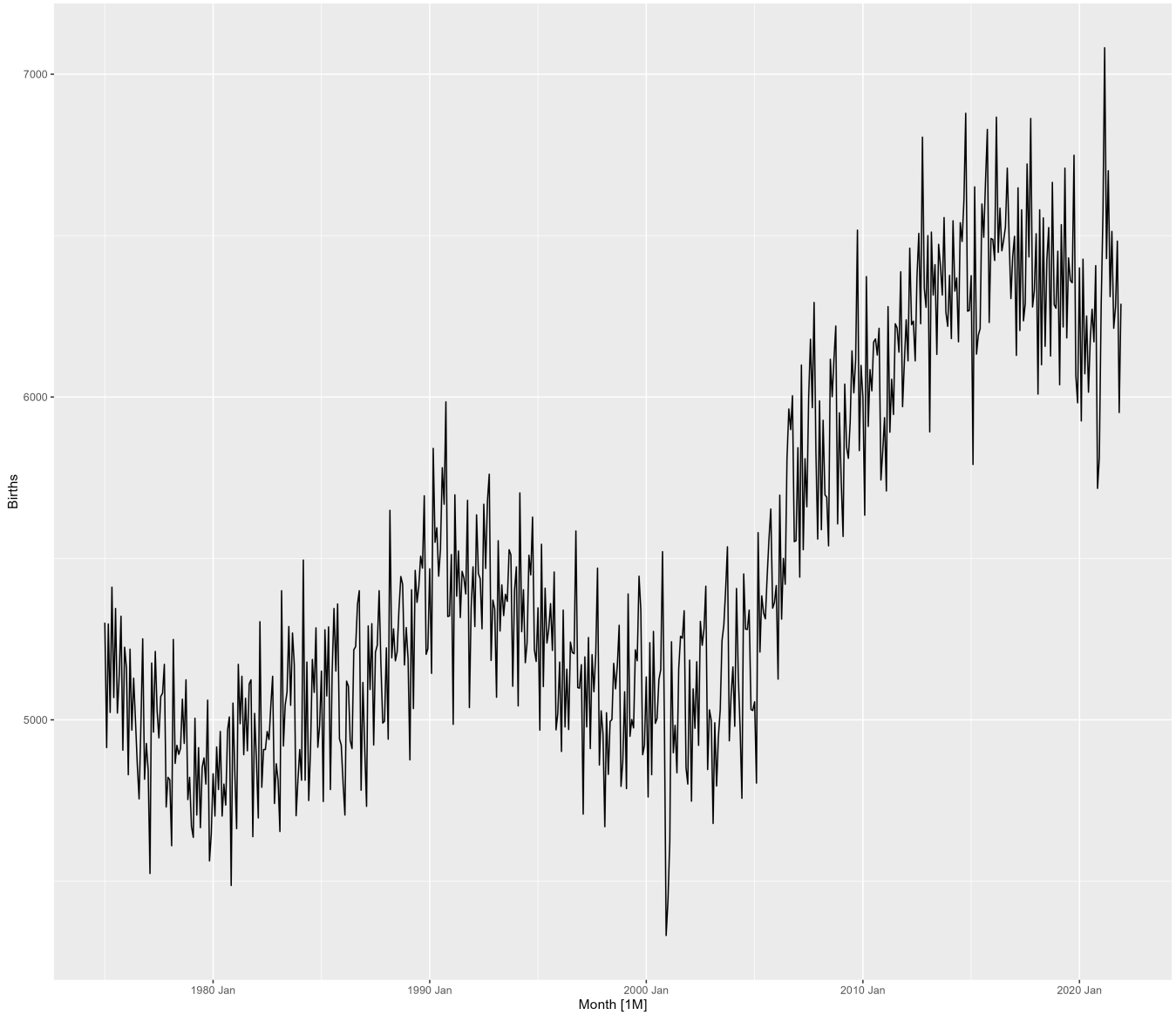
- The forecasting workflow

Recap

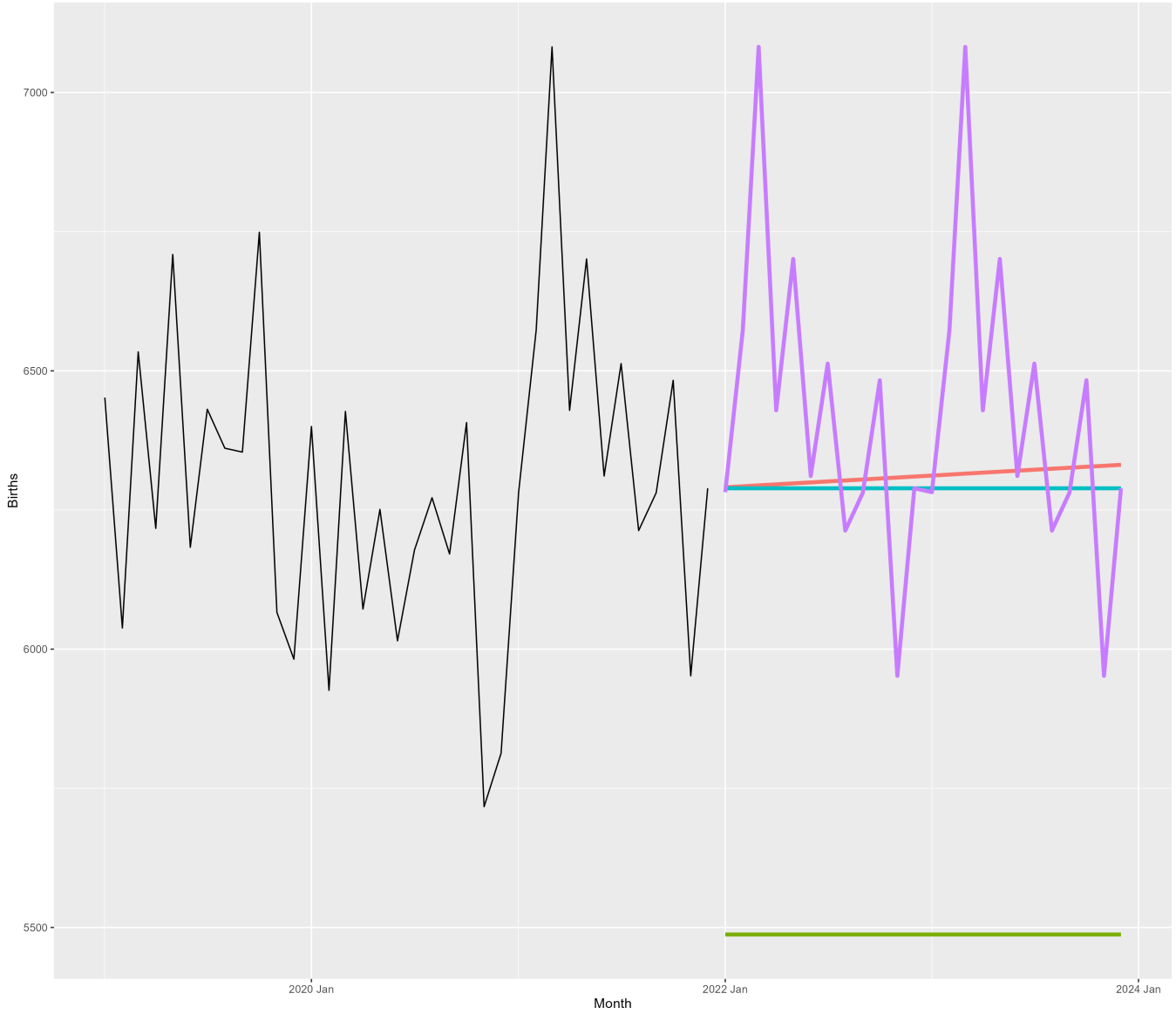
- Simple forecasting methods:
  - Mean method - forecast is the mean of historical data.
  - Naive method - forecast is equal to the last observation.
  - Seasonal naive method - forecast just repeats the values of the last season.
  - Drift method - connect with a line the first and last observation and forecasts are the points on this line.

```
aus_births |> filter(State == "VIC") -> vic_births
vic_births |> autoplot()

vic_births |> model(
  mean = MEAN(Births),
  naive = NAIVE(Births),
  s_naive = SNAIVE(Births ~lag(12)),
  drift = RW(Births ~ drift())) -> vic_births_models

vic_births_models |> forecast( h = "2 years") -> vic_births_forecast
vic_births_forecast |> autoplot(level = NULL, size = 1.5) +
  autolayer (vic_births |> tail(n = 36), Births, colour = "black") +
```

- The initial births in Victoria state time series



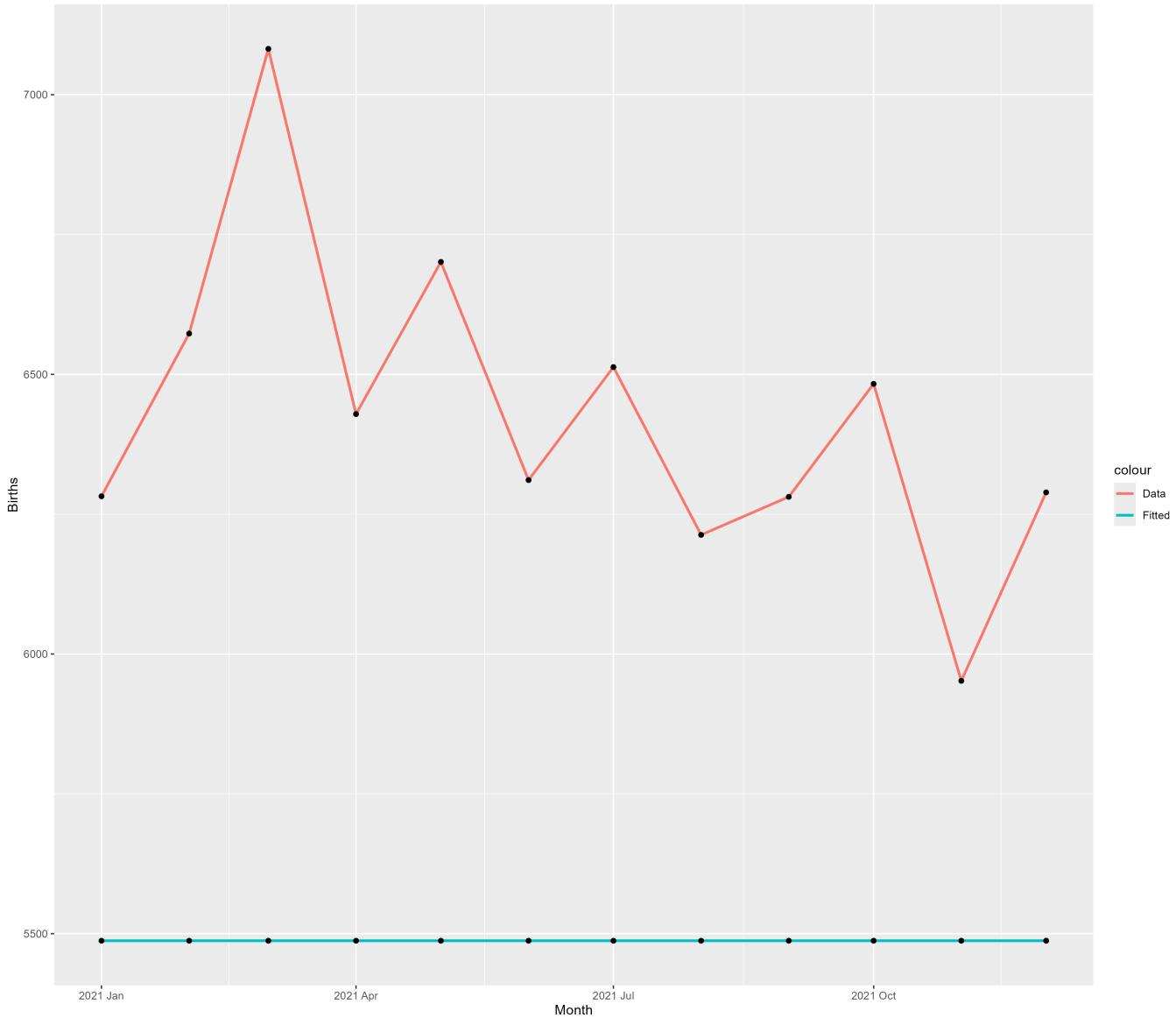
- The last three years of the historical data and the forecasts of the four simple models. Can you guess which color belongs to which method?



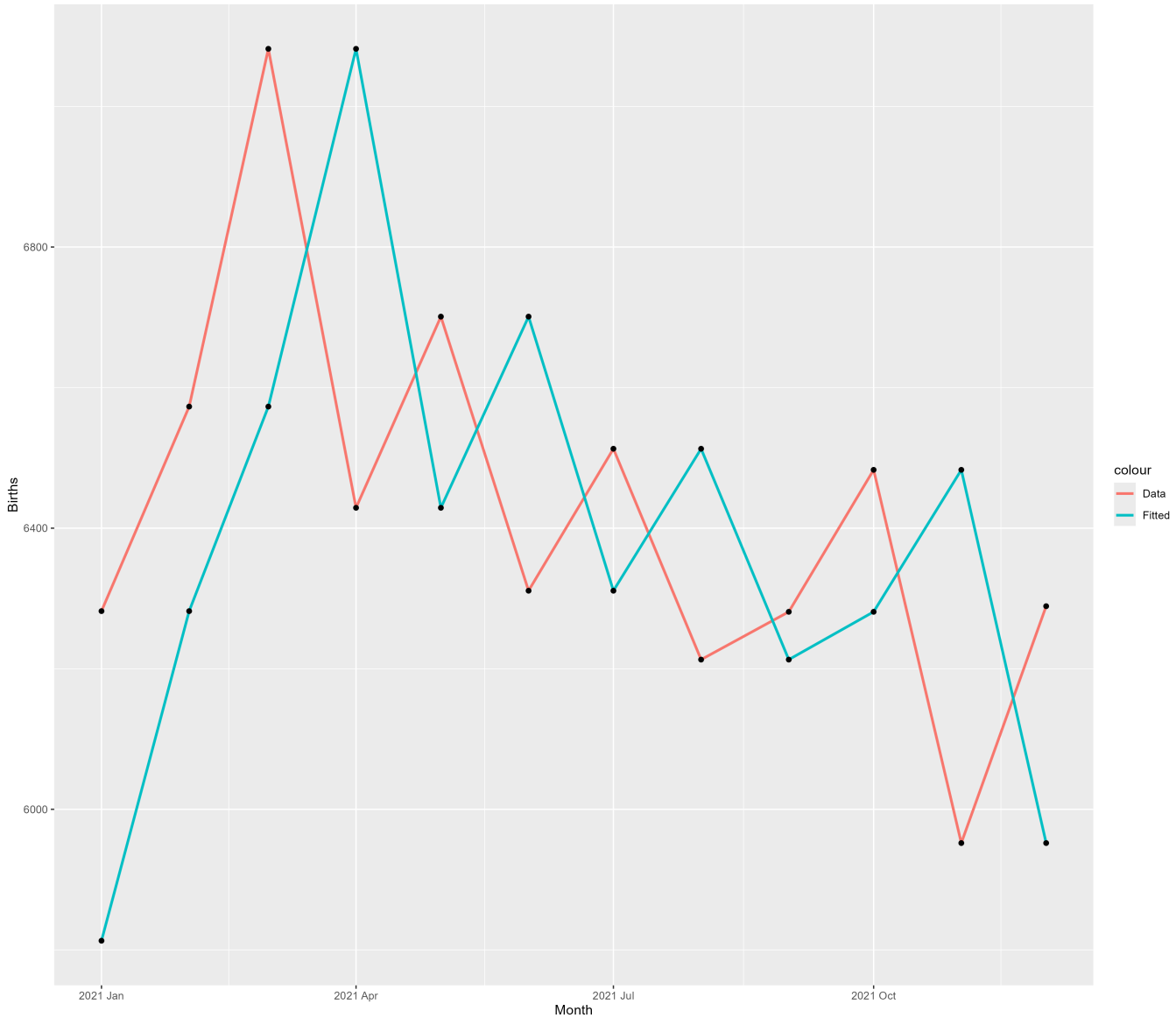
- The fitted values are computed by applying an estimated model on the historical data used to create the model.
- The residual is simply the difference between the actual observation and the fitted value
- Residuals should be centered around 0, have no autocorrelation in them, and optionally be normally distributed and have constant variance.

```
vic_births_models |> augment() -> vic_births_augmented
vic_births_augmented

vic_births_augmented |> filter(.model == "mean") |> tail(n = 12) |>
  ggplot(mapping = aes(x = Month)) +
  geom_line(mapping = aes(y = Births, color = "Data"), size = 1) +
  geom_line(mapping = aes(y = .fitted, color = "Fitted"), size = 1) +
  geom_point(mapping = aes(y = Births)) +
  geom_point(mapping = aes(y = .fitted))
```

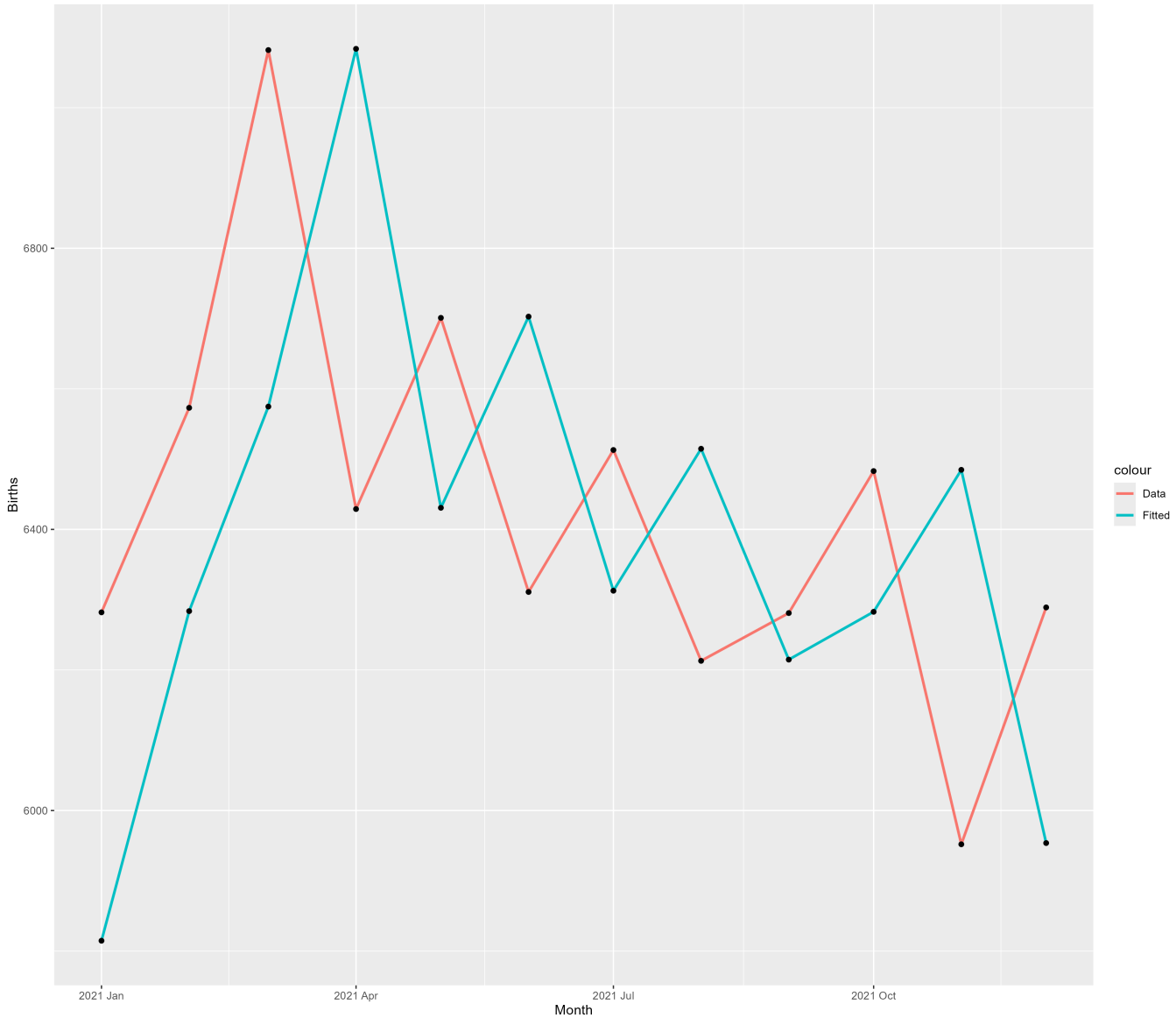
- Fitted values for the last year for the mean model



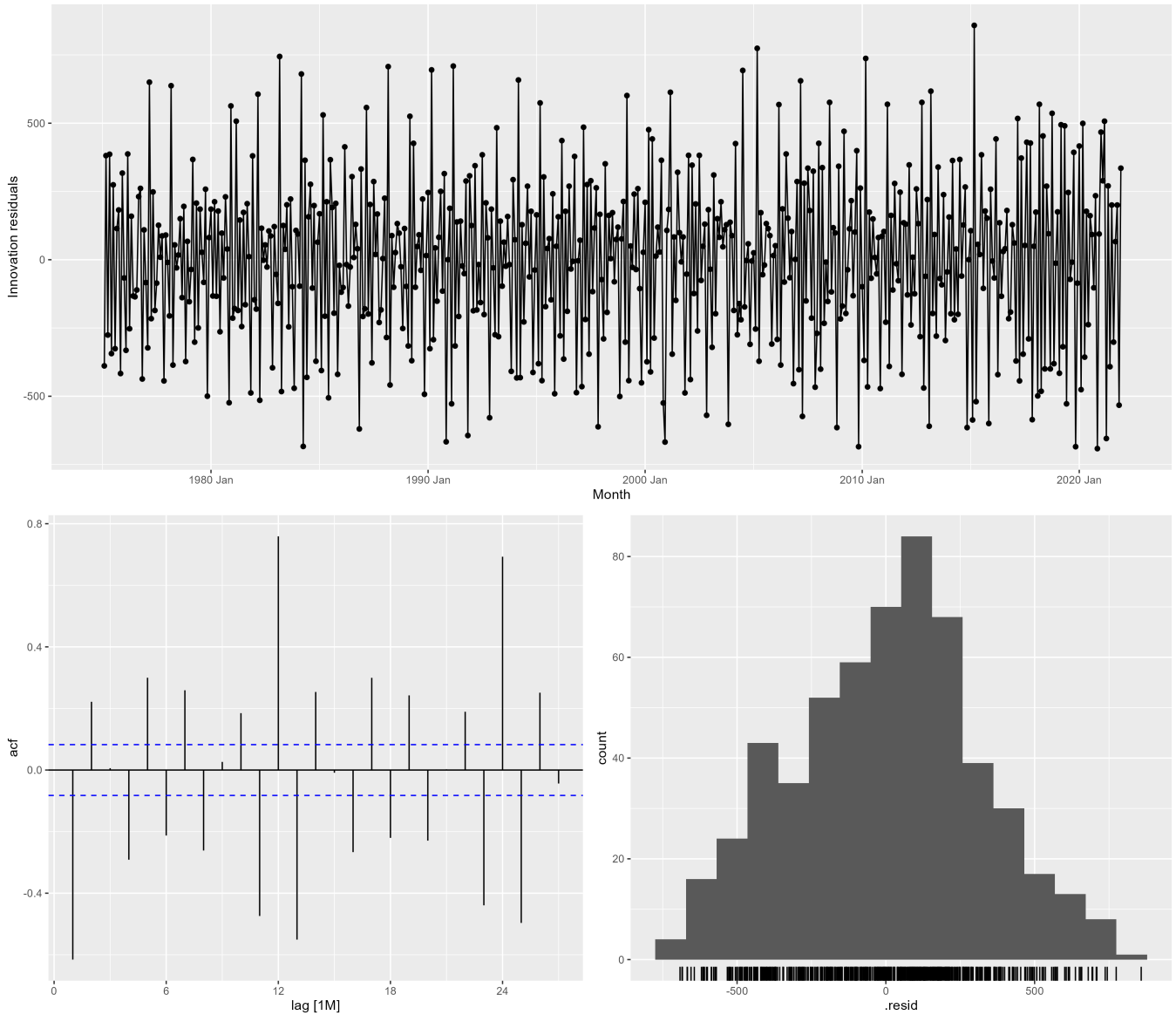
- Fitted values for the last year for the Naive model



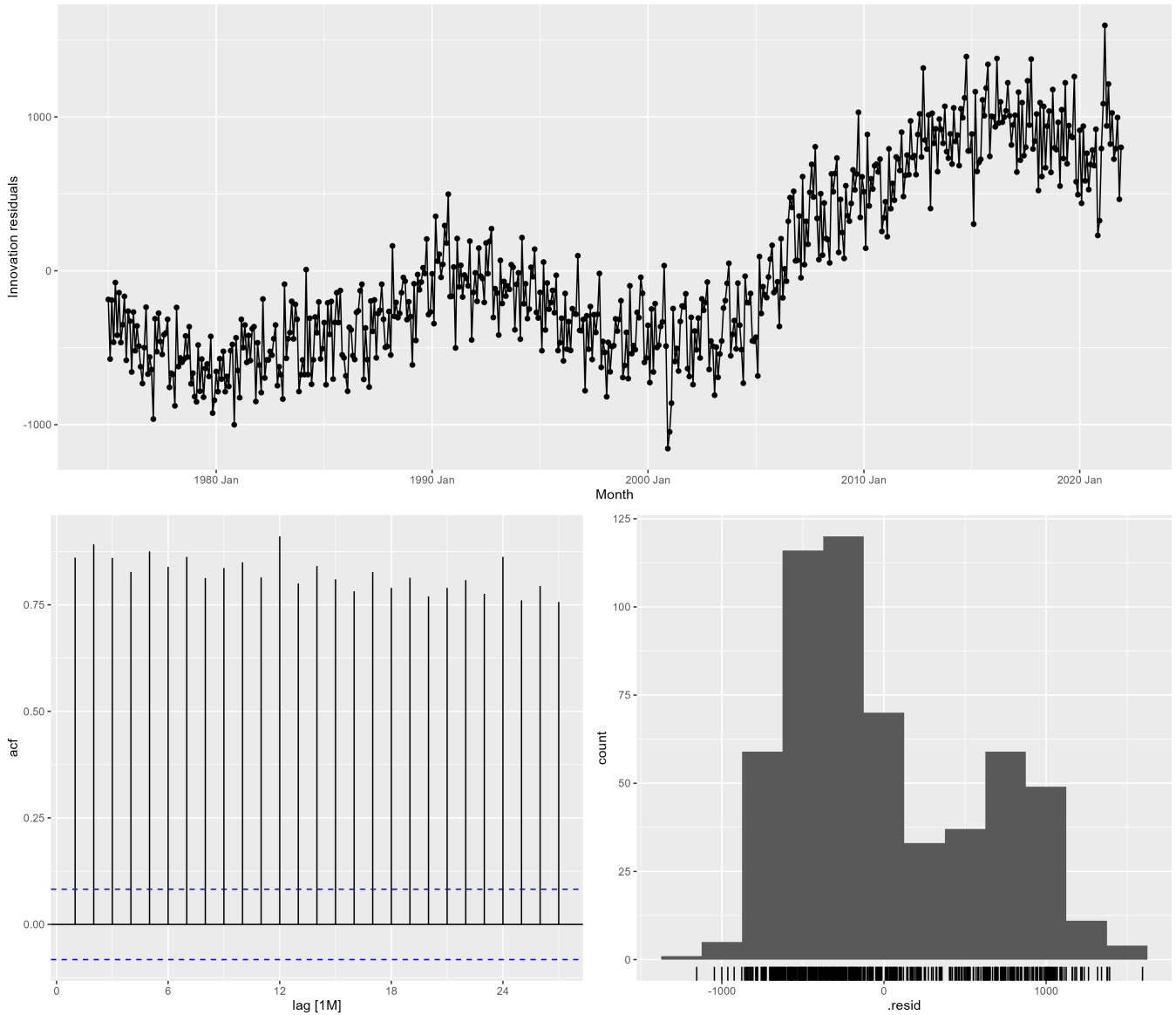
- Fitted values for the last year for the drift model



- Residual diagnostics for the drift model



- Residual diagnostics for the mean model



- We can use the ljung_box feature to test if the residuals are white noise.

```
vic_births_models |> select(mean) |>
  augment() |>
  features(.innov, ljung_box, lag = 10)
```

```
# A tibble: 1 × 3
  .model lb_stat lb_pvalue
  <chr>    <dbl>     <dbl>
1 mean     4147.         0
```

- The p-values shows the probability of seeing the associated *lb_stat* value for white noise. Since this probability is 0, we can conclude that there is autocorrelation in the residuals.

Prediction intervals

- While we have looked at (and plotted) only the forecast values, forecasting methods actually return a probability distribution of possible forecast values. Out of these values, the point forecast is the mean of the distribution. (This is why we used the $level = NULL$ parameter when plotting the baseline methods, to plot only the point forecast).
- For most models this probability distribution is the normal distribution (with a mean and a standard deviation), provided that the residuals are normally distributed.
- In forecasting prediction intervals are important, they show us how much uncertainty we have in our forecast. This is why having just the point forecast is basically useless.

Forecast distributions I

- For the baseline models discussed previously, assuming that the residuals are normally distributed and uncorrelated, with a constant standard deviation of $\hat{\sigma}$, the forecast distributions for $y_{T+h|T}$ are the following (mean and variance):
  - Mean:
$$N(\bar{y}, (1 + \frac{1}{T}) * \hat{\sigma}^2)$$
  - Naive:
$$N(y_T, h * \hat{\sigma}^2)$$
  - Seasonal naive (where k is the integer part of (h-1)/m):
$$N(y_{T+h-m(k+1)}, (k+1) * \hat{\sigma}^2)$$

Forecast distributions II

- Drift:

$$N(y_T + \frac{h}{T-1} * (y_T - y_1), h * \frac{T+h}{T} * \hat{\sigma}^2)$$

- **Obs. 1** When h $= 1$ and T is large, the variance is approximately $\hat{\sigma}^2$ for all models.

- A *prediction interval* is an interval in which we expect the actual value y_t to be with a certain probability p. For example, assuming a normal distribution for the future values, the 95% prediction interval for an h-step forecast is:

$$\hat{y}_{T+h|T} = \pm 1.96 * \hat{\sigma}_h$$

- where $\hat{\sigma}_h$ is the standard deviation of the h-step forecast distribution.
- The value 1.96 is specific for the 95% confidence interval, there exist other multipliers for other probabilities. Some of them are presented in the table below:

Percentage	Multiplier
50	0.67
75	1.15
80	1.28
90	1.64
95	1.96
99	2.58

One step prediction intervals

- In order to compute the prediction interval, we need the standard deviation of the forecast distribution.
- When we forecast only one step ahead (i.e., $h = 1$), this can be estimated by the standard deviation of the residuals:

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

- where K is the number of parameters estimated in the forecasting method and M is the number of missing values in the residuals (observations for which we do not have a forecast).

Multi-step prediction intervals

- In general, if the prediction horizon increases, so does the length of the prediction interval $\rightarrow$ the further away we forecast, the more uncertainty we have.
- For computing the prediction interval for an h-step forecast, we need the value of σ_h. We have seen how to compute σ_1, for larger values the computations are more complicated.

- Assuming that the residuals are not correlated, the value of σ_h was mathematically determined for the four benchmark methods:

Method	h-step forecast standard deviation
Mean	$\hat{\sigma}_h = \hat{\sigma}_1 * \sqrt{1 + 1/T}$
Naive	$\hat{\sigma}_h = \hat{\sigma}_1 * \sqrt{h}$
Seasonal naive	$\hat{\sigma}_h = \hat{\sigma}_1 * \sqrt{k + 1}$
Drift	$\hat{\sigma}_h = \hat{\sigma}_1 * \sqrt{h * (1 + h/(T - 1))}$

- where m is the seasonal period and k is the integer part of $\frac{h-1}{m}$ (i.e. the number of complete years in the forecast period, prior to the time $T + h$).

- We can get the prediction intervals in R using the *hilo* function which, by default, displays the 80% and 95% prediction interval.
- If you want to get prediction interval for another percentage, it can be set with the *level* argument.
- The *level* argument can be used in the *autoplot* function as well, to specify what prediction interval to display (or, by setting it to *NULL*, we can specify that we do not want the prediction interval displayed).

```
vic_births_forecast |>
  hilo()

google2015Data |>
  model(NAIVE(Close)) |>
  forecast(h = 10) |>
  hilo()
```

	State	.model	Month	Births	.mean	‘80%‘	‘95%‘
	<chr>	<chr>	<mth>	<dist>	<dbl>	<hilo>	<hilo>
1	VIC	mean	2022 Jan	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
2	VIC	mean	2022 Feb	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
3	VIC	mean	2022 Mar	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
4	VIC	mean	2022 Apr	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
5	VIC	mean	2022 May	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
6	VIC	mean	2022 Jun	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
7	VIC	mean	2022 Jul	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
8	VIC	mean	2022 Aug	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
9	VIC	mean	2022 Sep	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95
10	VIC	mean	2022 Oct	N(5487, 344666)	5487.	[4734.98, 6239.733]80	[4336.695, 6638.017]95

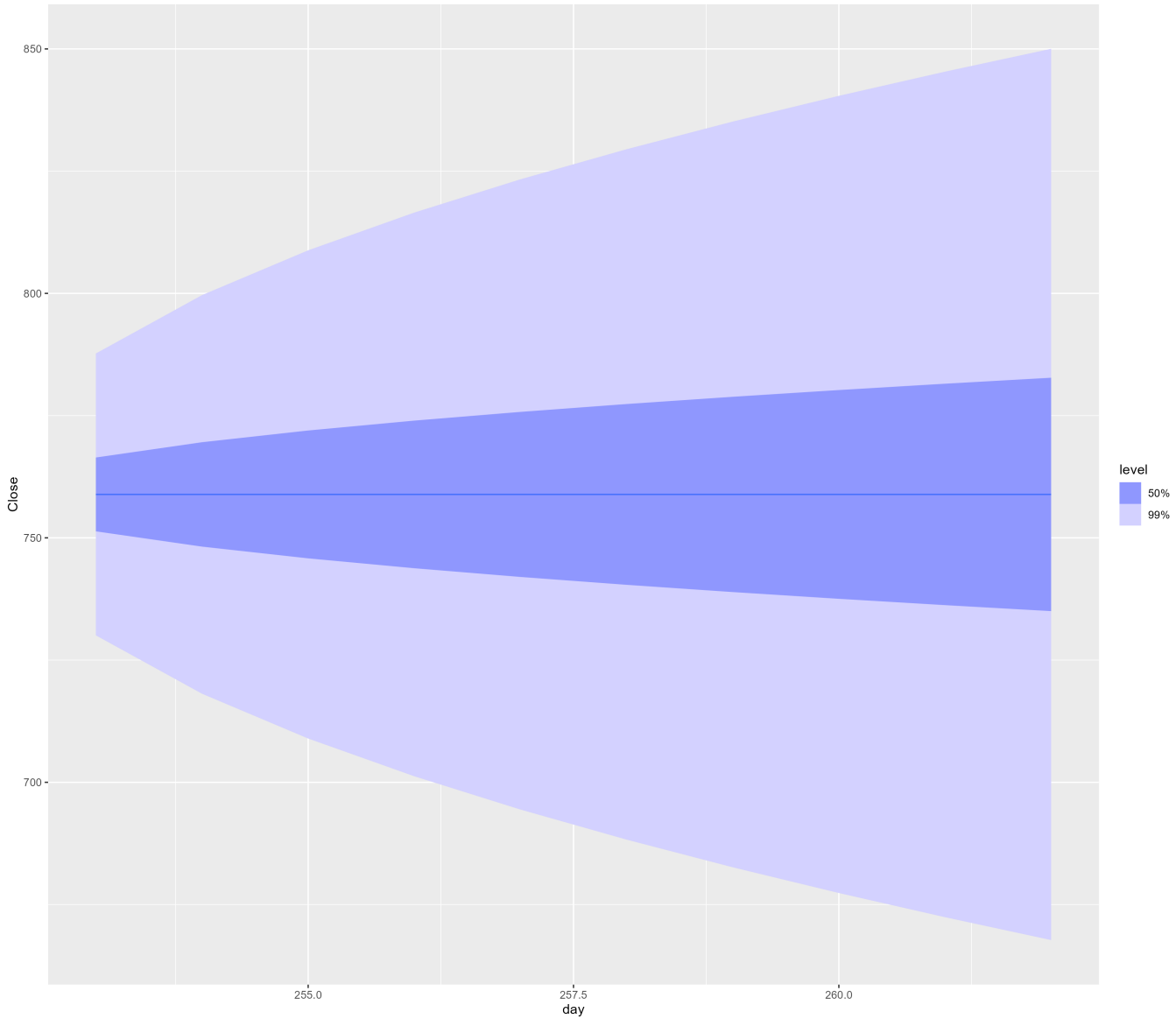
```
.model          day        Close .mean                 ‘80%‘                  ‘95%‘
  <chr>         <dbl>       <dist> <dbl>                 <hilo>                 <hilo>
1 NAIVE(Close)   253  N(759, 125)  759. [744.5400, 773.2200]80 [736.9488, 780.8112]95
2 NAIVE(Close)   254  N(759, 250)  759. [738.6001, 779.1599]80 [727.8646, 789.8954]95
3 NAIVE(Close)   255  N(759, 376)  759. [734.0423, 783.7177]80 [720.8941, 796.8659]95
4 NAIVE(Close)   256  N(759, 501)  759. [730.1999, 787.5601]80 [715.0176, 802.7424]95
5 NAIVE(Close)   257  N(759, 626)  759. [726.8147, 790.9453]80 [709.8404, 807.9196]95
6 NAIVE(Close)   258  N(759, 751)  759. [723.7543, 794.0058]80 [705.1598, 812.6002]95
7 NAIVE(Close)   259  N(759, 876)  759. [720.9399, 796.8202]80 [700.8556, 816.9045]95
8 NAIVE(Close)   260 N(759, 1002)  759. [718.3203, 799.4397]80 [696.8493, 820.9108]95
9 NAIVE(Close)   261 N(759, 1127)  759. [715.8599, 801.9001]80 [693.0865, 824.6735]95
10 NAIVE(Close)   262 N(759, 1252)  759. [713.5329, 804.2272]80 [689.5275, 828.2325]95
```

```
google2015Data |>
    model(NAIVE(Close)) |>
    forecast(h = 10) |>
    hilo(level = 99)
```

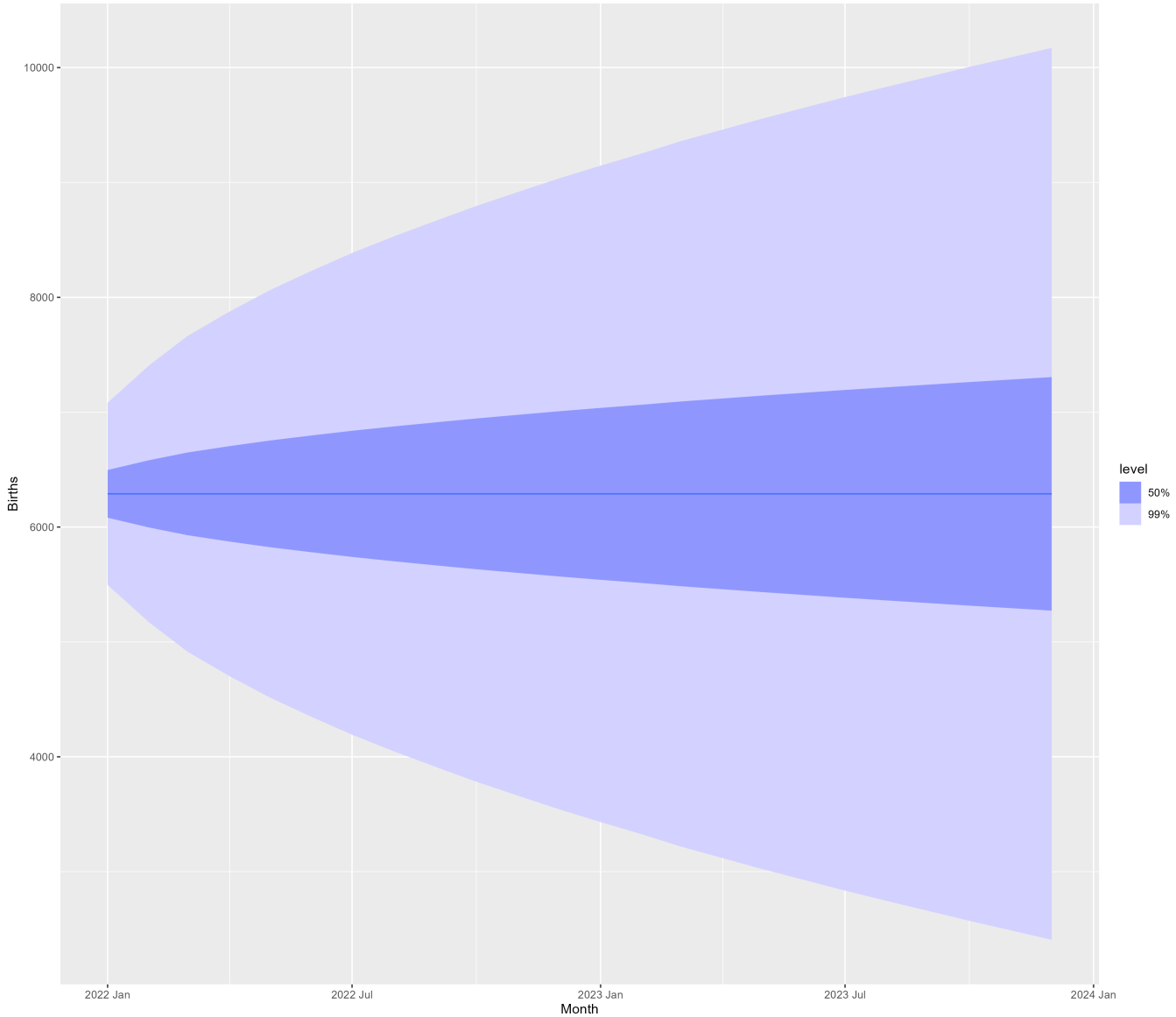
```
# A tibble: 10 × 5
   .model         day        Close .mean                 `99%`
   <chr>          <dbl>       <dist> <dbl>                <hilo>
 1 NAIVE(Close)   253  N(759, 125)   759. [730.0575, 787.7025]99
 2 NAIVE(Close)   254  N(759, 250)   759. [718.1189, 799.6411]99
 3 NAIVE(Close)   255  N(759, 376)   759. [708.9580, 808.8020]99
 4 NAIVE(Close)   256  N(759, 501)   759. [701.2351, 816.5249]99
 5 NAIVE(Close)   257  N(759, 626)   759. [694.4310, 823.3290]99
 6 NAIVE(Close)   258  N(759, 751)   759. [688.2797, 829.4803]99
 7 NAIVE(Close)   259  N(759, 876)   759. [682.6230, 835.1371]99
 8 NAIVE(Close)   260 N(759, 1002)   759. [677.3578, 840.4022]99
 9 NAIVE(Close)   261 N(759, 1127)   759. [672.4126, 845.3474]99
10 NAIVE(Close)   262 N(759, 1252)   759. [667.7354, 850.0246]99
```

```
google2015Data |>
     model(NAIVE(Close)) |>
     forecast(h = 10) |>
     autoplot(level = c(50, 99))

vic_births_forecast |>
  filter(.model == "mean") |>
  autoplot(level = c(50, 99))
```

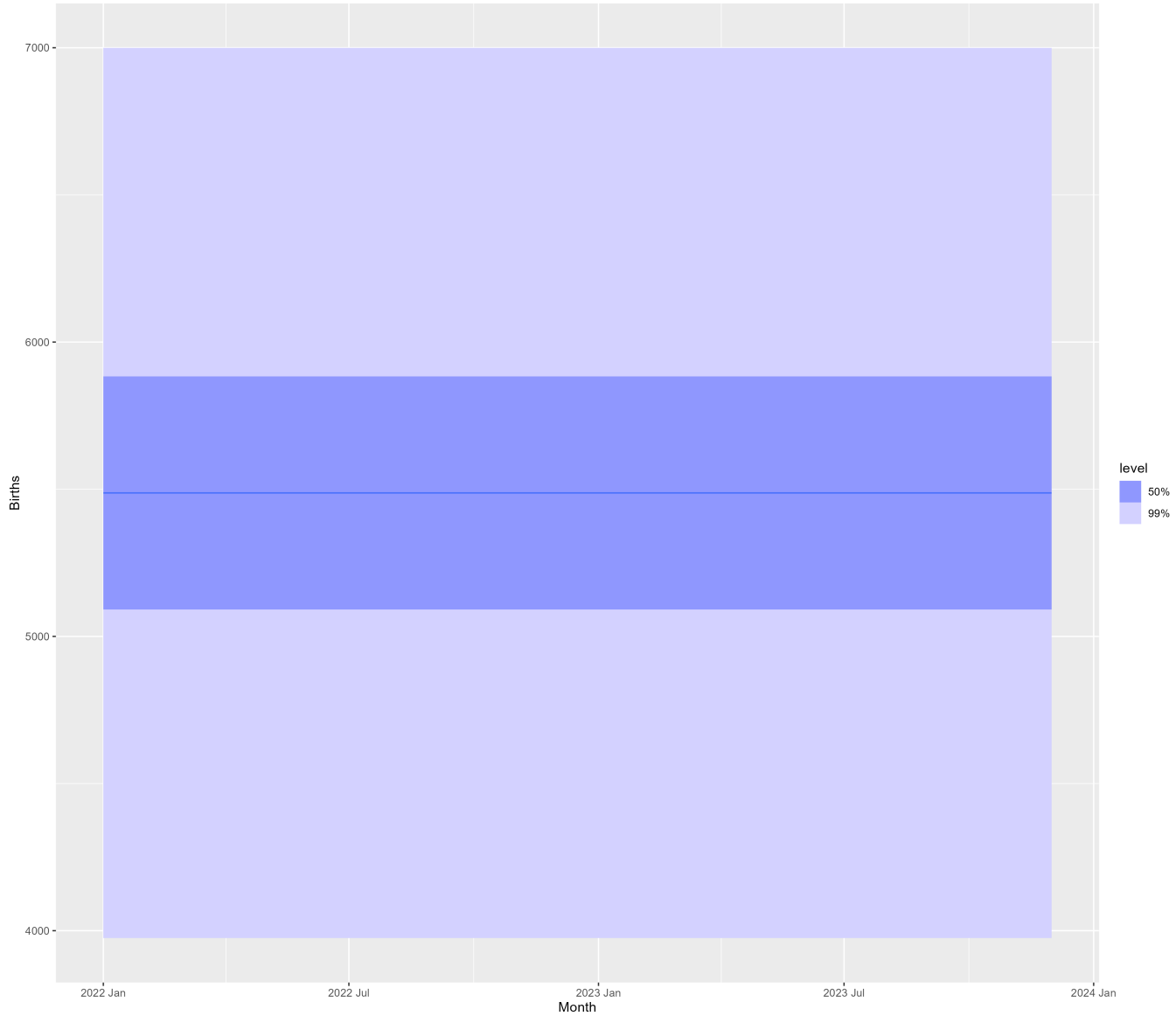
- 50% and 99% Confidence interval for the Naive model on Google data



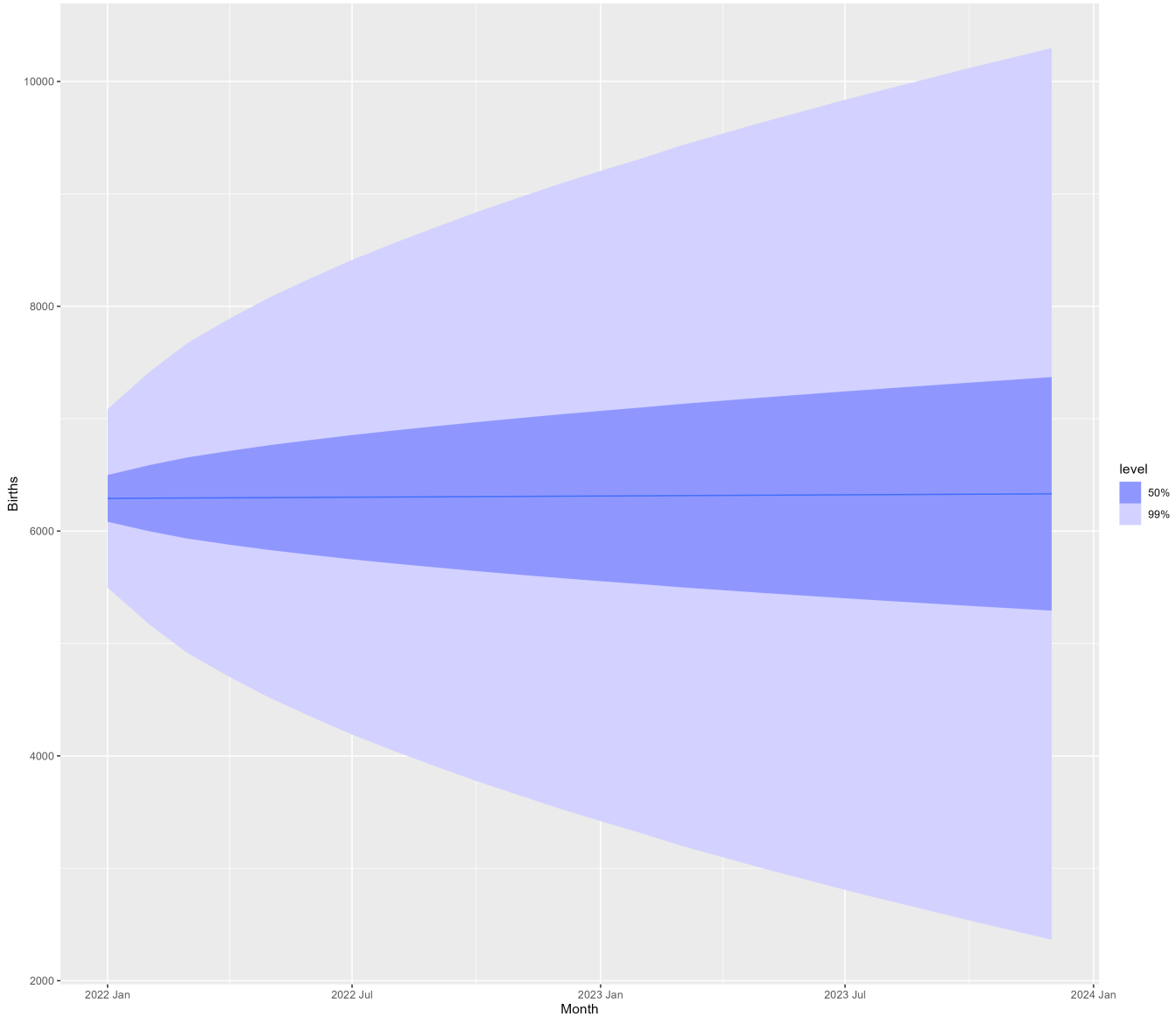
- 50% and 99% Confidence interval for the Naive model on Victorian births data



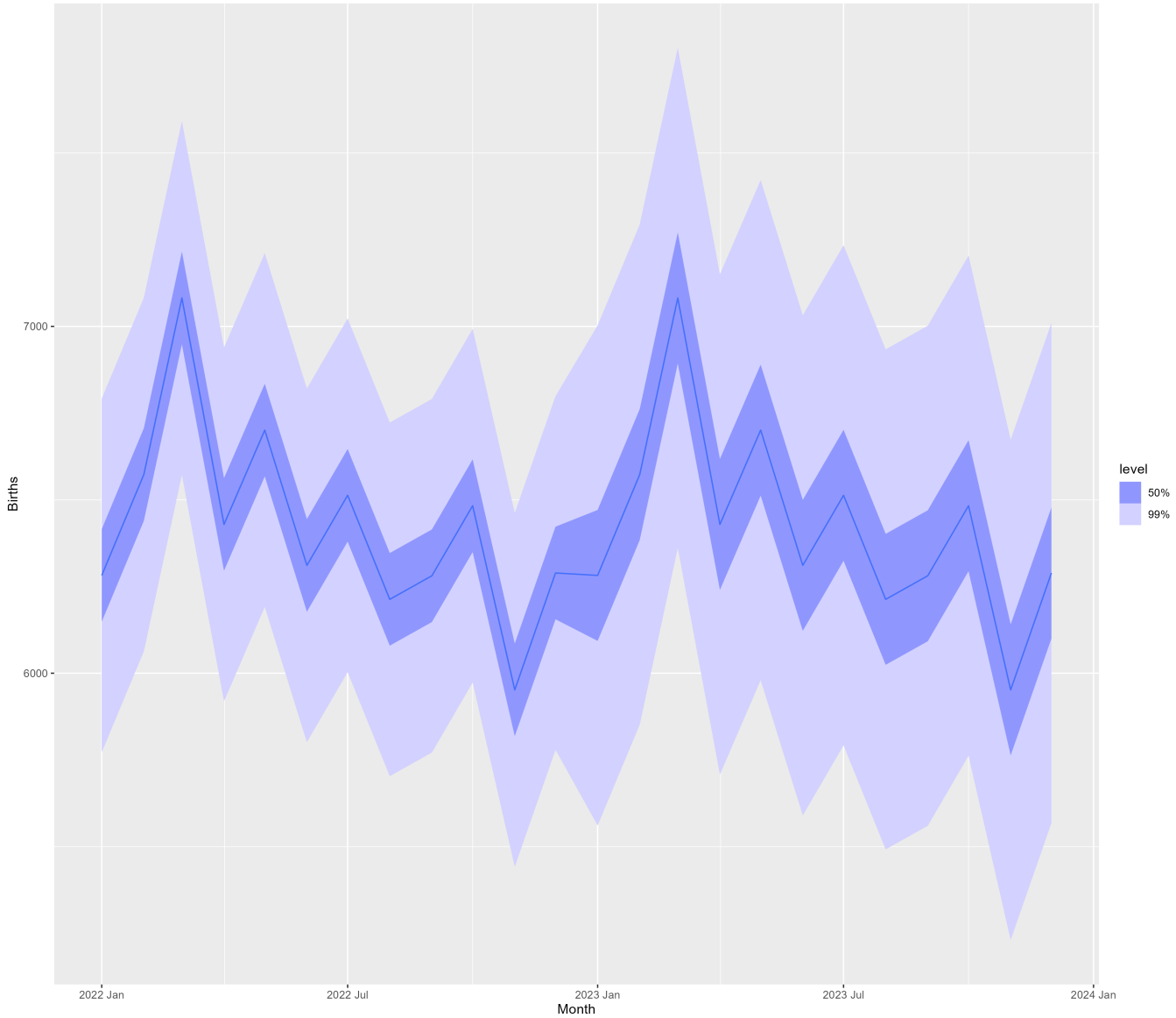
- 50% and 99% Confidence interval for the mean model on Victorian births data



- 50% and 99% Confidence interval for the drift model on Victorian births data



- 50% and 99% Confidence interval for the seasonal naive model on Victorian births data

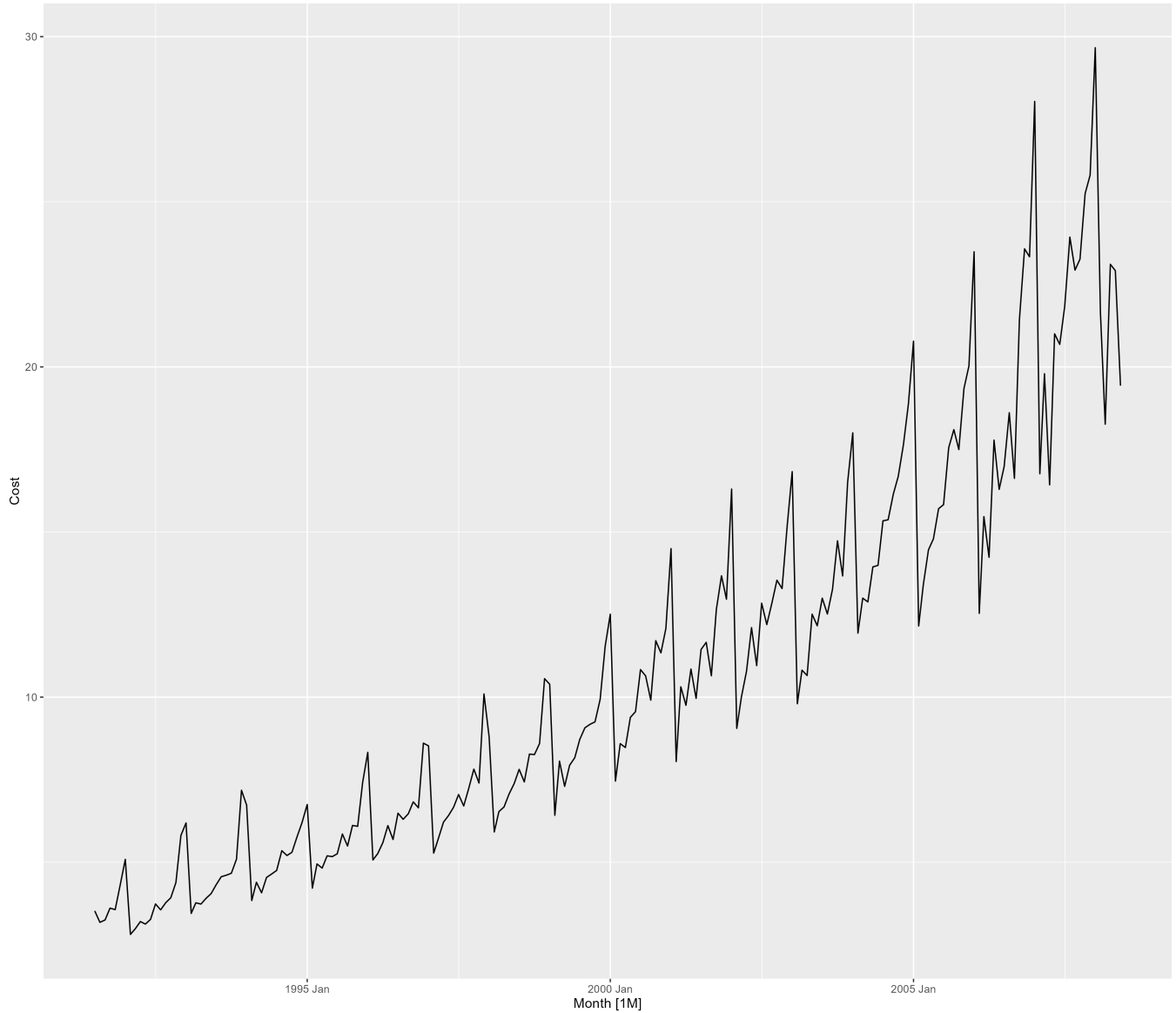


Forecasting using transformations

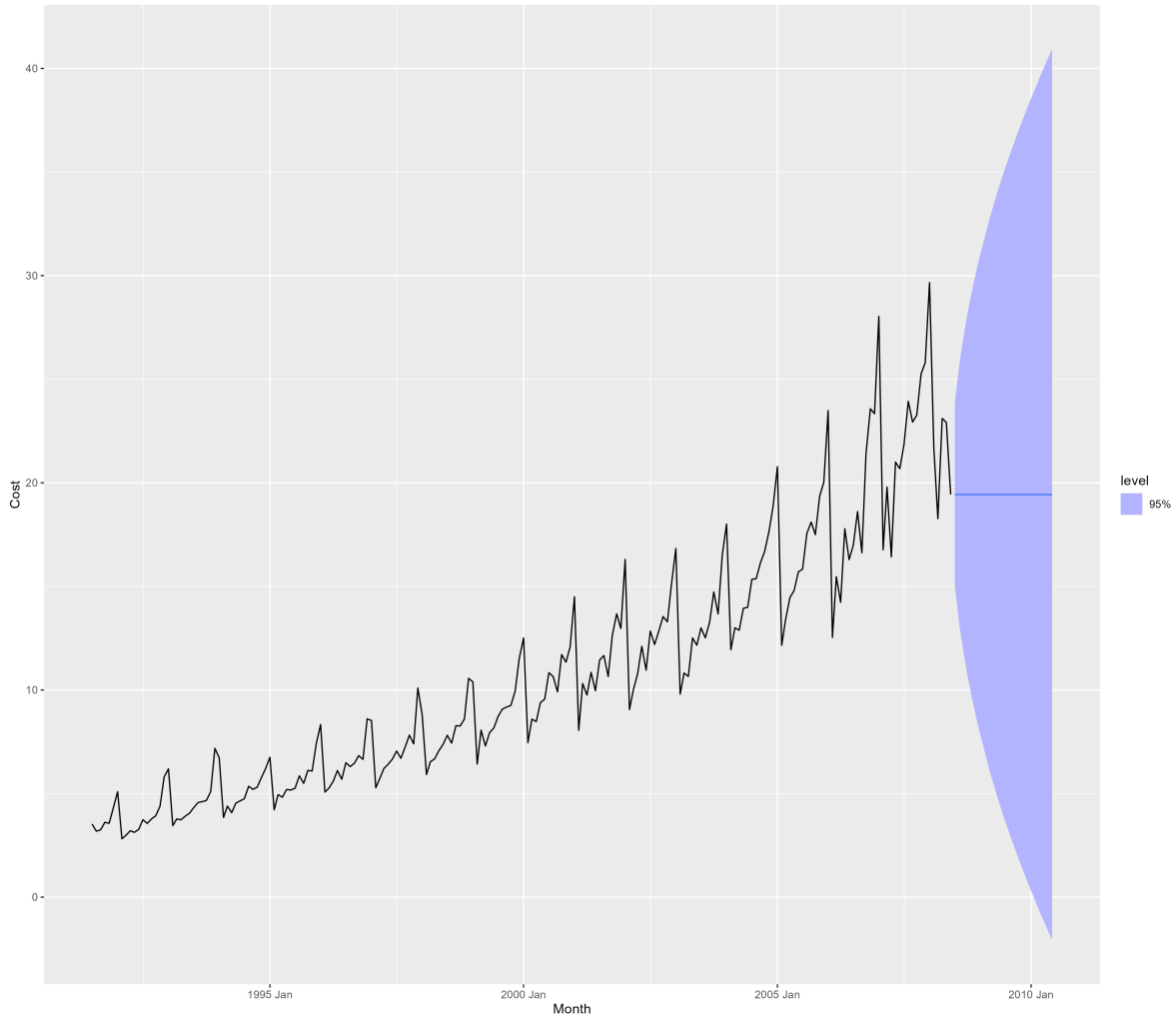
- We have talked about performing transformations on the data before forecasting (for example the log transformation, or Box-Cox to stabilize variance in the data).
- When working with transformed data, the forecasts and the prediction intervals will be on the transformed scale, so we need to *back-transform* them to obtain values on the original scale.
- Transformations will not influence the point forecast that much, but they have a great influence on the prediction interval: after the back-transformation the prediction interval will no longer be symmetric around the point.
- In R if we specify the transformation when specifying the model, it will take care automatically of the transformation and back-transformation.
- Using the *augment* function we can check the residuals and the innovative residuals.

```
antidiabetic <- PBS |>
  filter(ATC2 == "A10") |>
  select(Month, Concession, Type, Cost) |>
  summarize(TotalCost = sum(Cost)) |>
  mutate(Cost = TotalCost / 1000000) #just to have smaller values
antidiabetic |> autoplot(Cost)

antidiabetic |>
  model(naive = NAIVE(Cost)) |>
  forecast(h = 24) -> antiDForecast

antiDForecast |>
  autoplot(level = 95) +
  autolayer(antidiabetic, Cost)
```

- Antidiabetic drug prescriptions plot



- Forecast and 95% confidence interval for the Naive model on antidiabetic drugs sales data.



```
antiDForecast |> hilo()
```

```
   .model    Month      Cost .mean                 `80%`                  `95%`
   <chr>     <mth>     <dist> <dbl>               <hilo>                 <hilo>
 1 naive  2008 Jul  N(19, 5)  19.4 [16.56374, 22.29974]80 [15.045511, 23.81797]95
 2 naive  2008 Aug N(19, 10)  19.4 [15.37577, 23.48771]80 [13.228676, 25.63480]95
 3 naive  2008 Sep N(19, 15)  19.4 [14.46422, 24.39926]80 [11.834569, 27.02891]95
 4 naive  2008 Oct N(19, 20)  19.4 [13.69574, 25.16774]80 [10.659283, 28.20420]95
 5 naive  2008 Nov N(19, 25)  19.4 [13.01870, 25.84478]80 [ 9.623835, 29.23965]95
 6 naive  2008 Dec N(19, 30)  19.4 [12.40660, 26.45688]80 [ 8.687718, 30.17576]95
 7 naive  2009 Jan N(19, 35)  19.4 [11.84372, 27.01976]80 [ 7.826870, 31.03661]95
 8 naive  2009 Feb N(19, 40)  19.4 [11.31981, 27.54367]80 [ 7.025612, 31.83787]95
 9 naive  2009 Mar N(19, 45)  19.4 [10.82774, 28.03574]80 [ 6.273054, 32.59043]95
10 naive  2009 Apr N(19, 50)  19.4 [10.36233, 28.50115]80 [ 5.561267, 33.30221]95
```

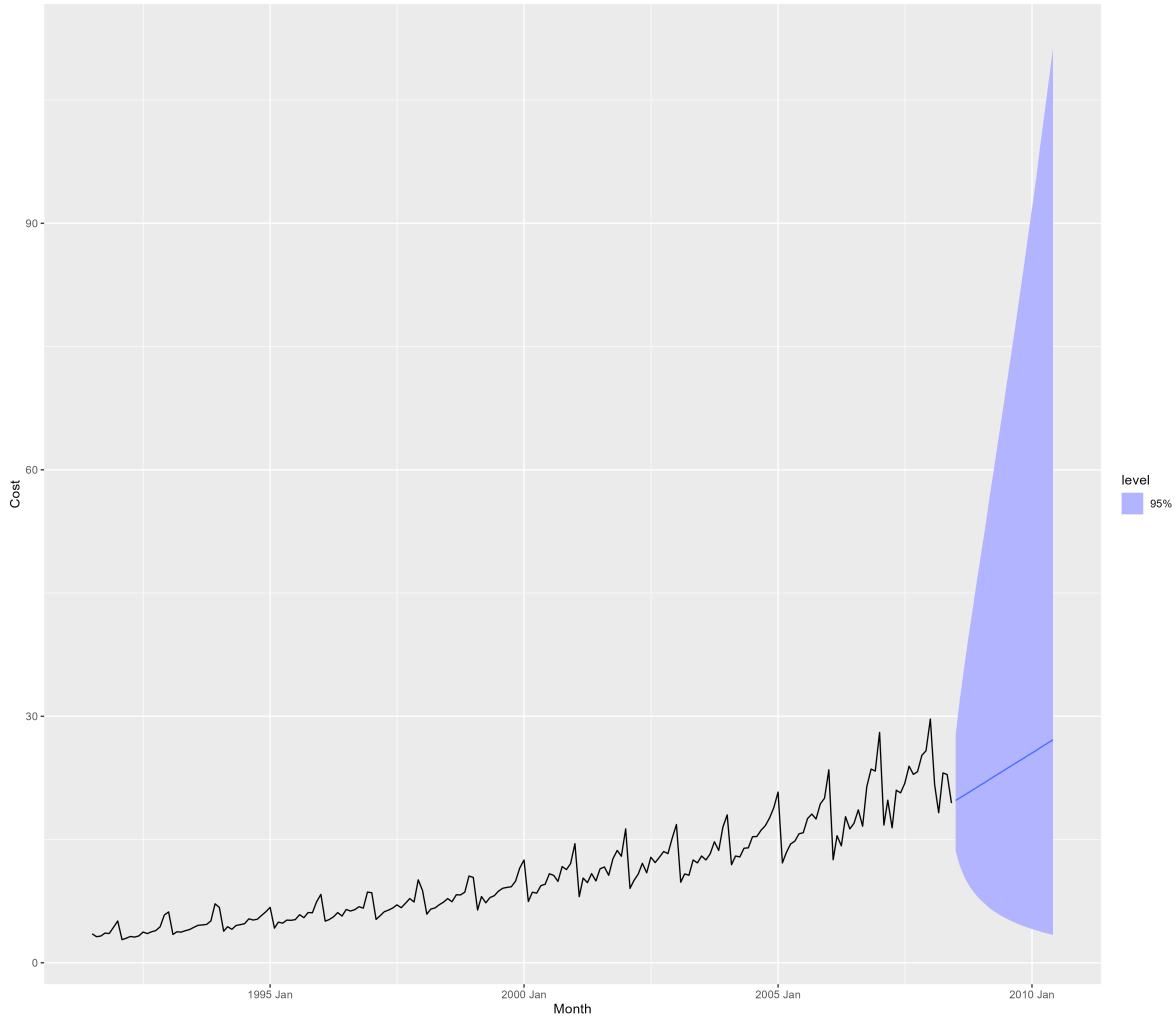
```
antidiabetic |>
  model(naive = NAIVE(log(Cost))) |>
  forecast(h= 24) -> antiDLogForecast
antidiabetic |>
  model(naive = NAIVE(log(Cost))) |>
  augment()
```

```
   .model    Month  Cost .fitted  .resid  .innov
   <chr>     <mth> <dbl>   <dbl>   <dbl>   <dbl>
 1 naive  1991 Jul  3.53   NA     NA       NA
 2 naive  1991 Aug  3.18    3.53 -0.346  -0.103
 3 naive  1991 Sep  3.25    3.18  0.0713  0.0222
 4 naive  1991 Oct  3.61    3.25  0.359   0.105
 5 naive  1991 Nov  3.57    3.61 -0.0451 -0.0126
 6 naive  1991 Dec  4.31    3.57  0.741   0.189
 7 naive  1992 Jan  5.09    4.31  0.782   0.167
 8 naive  1992 Feb  2.81    5.09 -2.27   -0.592
 9 naive  1992 Mar  2.99    2.81  0.171   0.0591
10 naive  1992 Apr  3.20    2.99  0.219   0.0708
```

```
antiDLogForecast |> hilo()
```

```
    .model    Month           Cost .mean                 `80%`                  `95%`
   <chr>     <mth>         <dist> <dbl>                <hilo>                 <hilo>
 1 naive  2008 Jul t(N(3, 0.033))  19.8 [15.394061, 24.52845]80 [13.608277, 27.74727]95
 2 naive  2008 Aug t(N(3, 0.066))  20.1 [13.978224, 27.01291]80 [11.741424, 32.15901]95
 3 naive  2008 Sep t(N(3, 0.099))  20.4 [12.980758, 29.08863]80 [10.484521, 36.01428]95
 4 naive  2008 Oct  t(N(3, 0.13))  20.7 [12.195363, 30.96198]80 [ 9.530037, 39.62131]95
 5 naive  2008 Nov  t(N(3, 0.17))  21.0 [11.542887, 32.71214]80 [ 8.761387, 43.09735]95
 6 naive  2008 Dec   t(N(3, 0.2))  21.4 [10.983102, 34.37941]80 [ 8.119975, 46.50169]95
 7 naive  2009 Jan  t(N(3, 0.23))  21.7 [10.492319, 35.98752]80 [ 7.571664, 49.86916]95
 8 naive  2009 Feb  t(N(3, 0.26))  22.0 [10.055236, 37.55183]80 [ 7.094631, 53.22229]95
 9 naive  2009 Mar   t(N(3, 0.3))  22.3 [ 9.661315, 39.08293]80 [ 6.673997, 56.57667]95
10 naive  2009 Apr  t(N(3, 0.33))  22.6 [ 9.302945, 40.58849]80 [ 6.299126, 59.94364]95
```

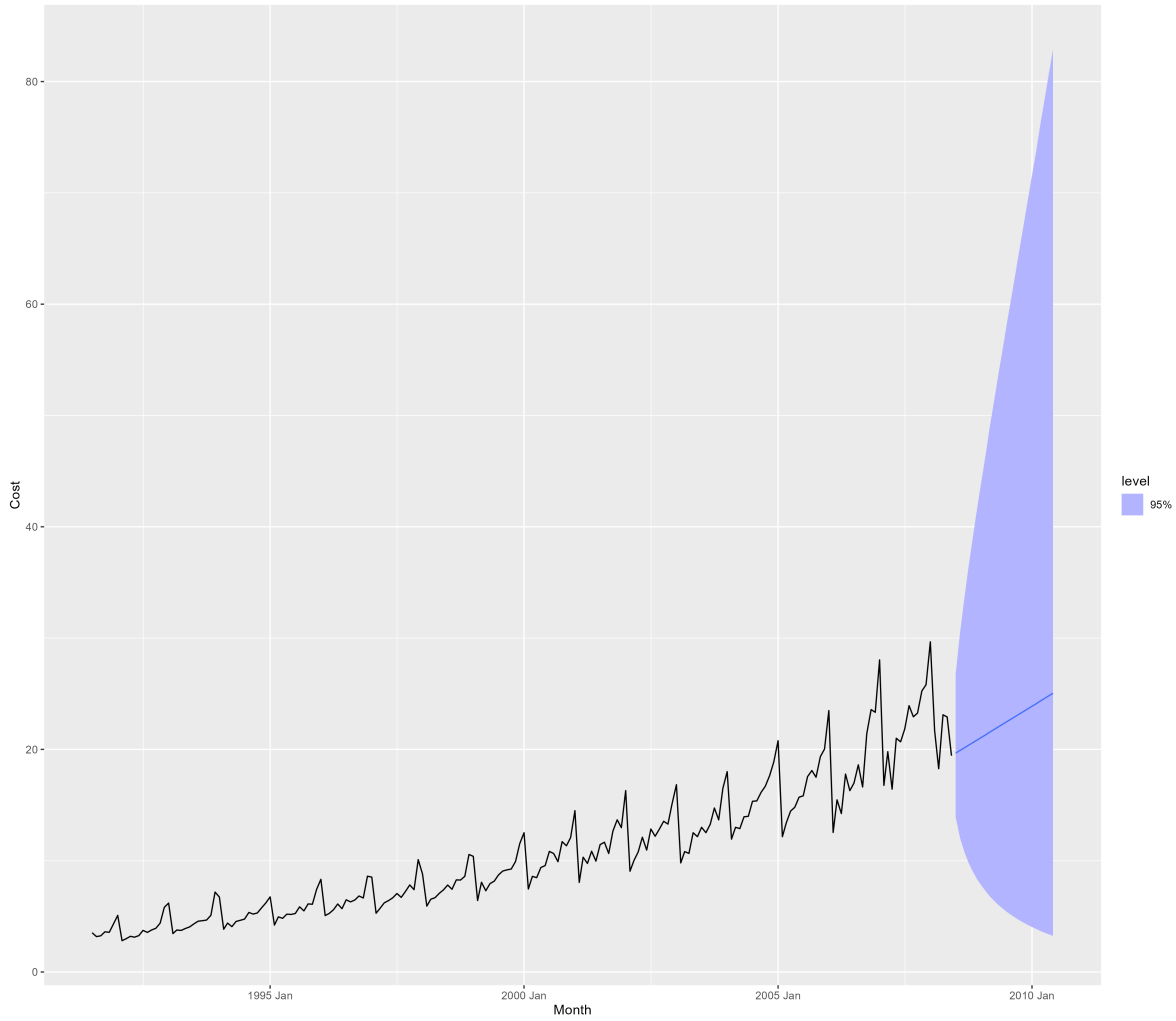
```
antiDLogForecast |>
  autoplot(level = 95) +
  autolayer(antidiabetic, Cost)
```


90
60
30
0
Cost
1995 Jan
2000 Jan
2005 Jan
2010 Jan
Month
level
95%

```
antidiabetic |> features(Cost, guerrero)
antidiabetic |>
  model(naive = NAIVE(box_cox(Cost, lambda = 0.13))) |>
  forecast(h= 24) -> antiDBCForecast
antiDBCForecast |> hilo() |> print(n = 24)
antiDBCForecast |>
  autoplot(level = 95) +
  autolayer(antidiabetic, Cost)
```

```
   .model    Month              Cost .mean                 `80%`                  `95%`
   <chr>     <mth>             <dist> <dbl>                <hilo>                 <hilo>
 naive  2008 Jul t(N(3.6, 0.06))  19.7 [15.655695, 23.97657]80 [13.928643, 26.73710]95
 2 naive  2008 Aug t(N(3.6, 0.12))  19.9 [14.289510, 26.11416]80 [12.081738, 30.40063]95
 3 naive  2008 Sep t(N(3.6, 0.18))  20.1 [13.312713, 27.86498]80 [10.812781, 33.50061]95
 4 naive  2008 Oct t(N(3.6, 0.24))  20.4 [12.534897, 29.41946]80 [ 9.834732, 36.32409]95
 5 naive  2008 Nov  t(N(3.6, 0.3))  20.6 [11.882698, 30.85094]80 [ 9.037773, 38.98093]95
 6 naive  2008 Dec t(N(3.6, 0.36))  20.8 [11.318681, 32.19687]80 [ 8.366242, 41.52680]95
 7 naive  2009 Jan t(N(3.6, 0.42))  21.1 [10.820720, 33.47939]80 [ 7.787455, 43.99443]95
 8 naive  2009 Feb t(N(3.6, 0.48))  21.3 [10.374467, 34.71282]80 [ 7.280360, 46.40488]95
 9 naive  2009 Mar t(N(3.6, 0.54))  21.5 [ 9.970006, 35.90711]80 [ 6.830500, 48.77265]95
10 naive  2009 Apr  t(N(3.6, 0.6))  21.8 [ 9.600153, 37.06942]80 [ 6.427465, 51.10816]95
11 naive  2009 May t(N(3.6, 0.66))  22.0 [ 9.259509, 38.20517]80 [ 6.063488, 53.41920]95
12 naive  2009 Jun t(N(3.6, 0.72))  22.2 [ 8.943902, 39.31852]80 [ 5.732600, 55.71178]95
13 naive  2009 Jul t(N(3.6, 0.78))  22.5 [ 8.650028, 40.41279]80 [ 5.430108, 57.99061]95
14 naive  2009 Aug t(N(3.6, 0.84))  22.7 [ 8.375224, 41.49063]80 [ 5.152252, 60.25948]95
15 naive  2009 Sep  t(N(3.6, 0.9))  22.9 [ 8.117302, 42.55424]80 [ 4.895963, 62.52148]95
16 naive  2009 Oct t(N(3.6, 0.96))  23.2 [ 7.874446, 43.60544]80 [ 4.658709, 64.77917]95
17 naive  2009 Nov    t(N(3.6, 1))  23.4 [ 7.645126, 44.64575]80 [ 4.438370, 67.03469]95
18 naive  2009 Dec  t(N(3.6, 1.1))  23.6 [ 7.428038, 45.67649]80 [ 4.233155, 69.28987]95
```

```
19 naive  2010 Jan  t(N(3.6, 1.1))  23.9 [ 7.222067, 46.69876]80 [ 4.041534, 71.54625]95
20 naive  2010 Feb  t(N(3.6, 1.2))  24.1 [ 7.026243, 47.71355]80 [ 3.862193, 73.80518]95
21 naive  2010 Mar  t(N(3.6, 1.3))  24.3 [ 6.839722, 48.72170]80 [ 3.693992, 76.06782]95
22 naive  2010 Apr  t(N(3.6, 1.3))  24.6 [ 6.661764, 49.72394]80 [ 3.535937, 78.33519]95
23 naive  2010 May  t(N(3.6, 1.4))  24.8 [ 6.491714, 50.72095]80 [ 3.387156, 80.60819]95
24 naive  2010 Jun  t(N(3.6, 1.4))  25.0 [ 6.328990, 51.71329]80 [ 3.246877, 82.88762]95
```


80
60
40
20
0
Cost
1995 Jan
2000 Jan
2005 Jan
2010 Jan
Month
level
95%

Forecasting with decomposition

- We have talked about ways to decompose a time series into three components: trend, seasonal and remainder. We can use decomposition before forecasting: decompose the data, forecast the components separately, then combine the results.
- In order to do this, we will consider that the time series is made of two components only: the seasonal component ($\hat{S}_t$) and the seasonally adjusted components ($\hat{A}_t$), which contains the trend and the remainder.
- So we have:

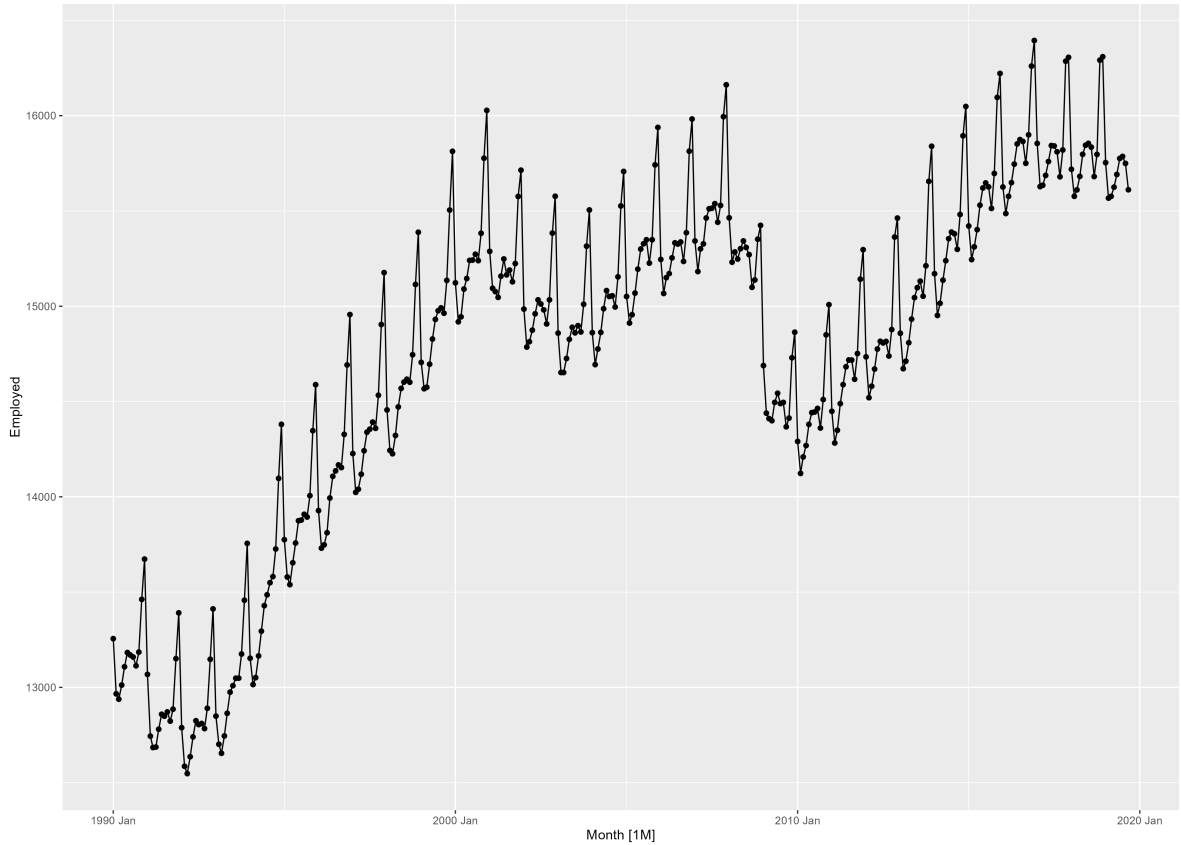
$$y_t = \hat{S}_t + \hat{A}_t \text{ where } \hat{A}_t = \hat{T}_t + \hat{R}_t$$

- or

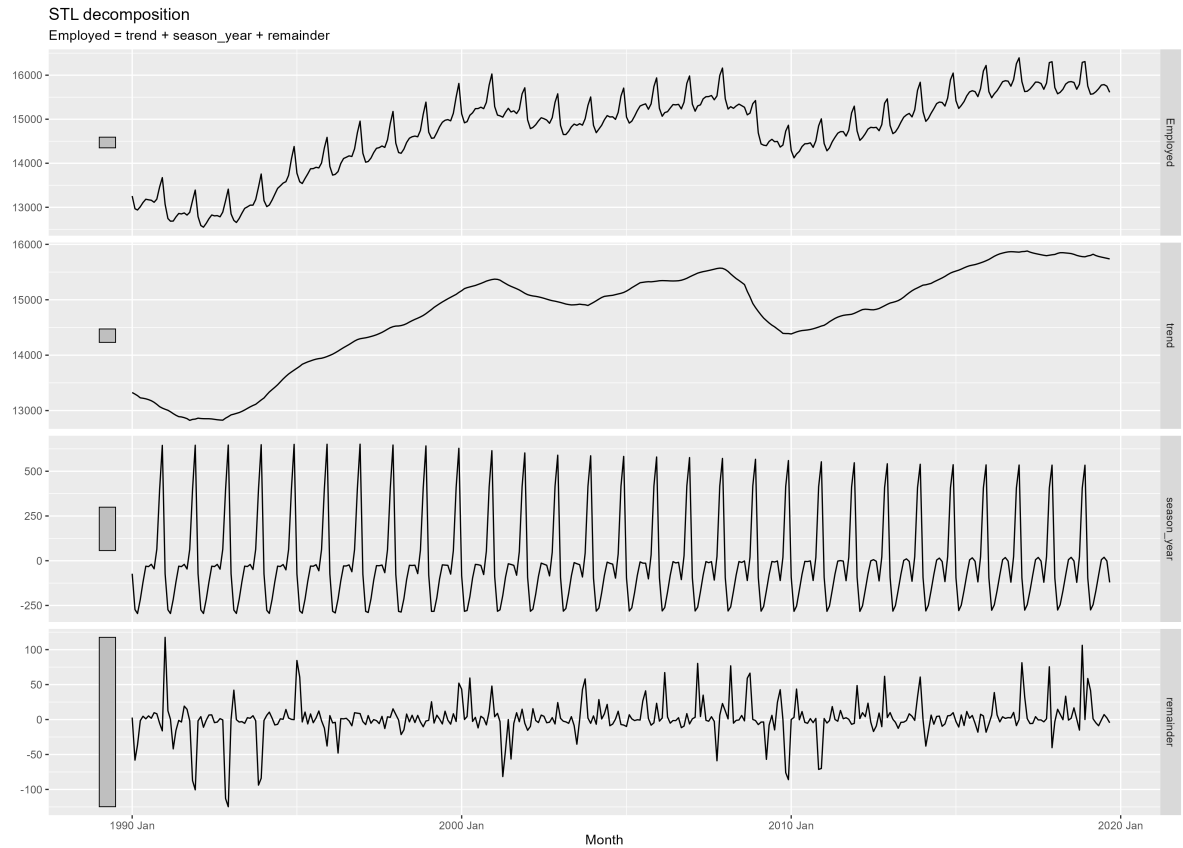
$$y_t = \hat{S}_t * \hat{A}_t \text{ where } \hat{A}_t = \hat{T}_t * \hat{R}_t$$

- Since the seasonal component in general changes slowly in time, it can be forecast with a seasonal naive model.
- The seasonally adjusted component can be forecast with any non-seasonal forecasting method, for example the drift method (or more complex methods that will be discussed later).
- Once you have the forecasts for the components, the prediction interval can be computed by adding the prediction intervals of the forecast components.
- Let's take an example, where we use as data the *us_employment* information from 1990 in the retail trade.

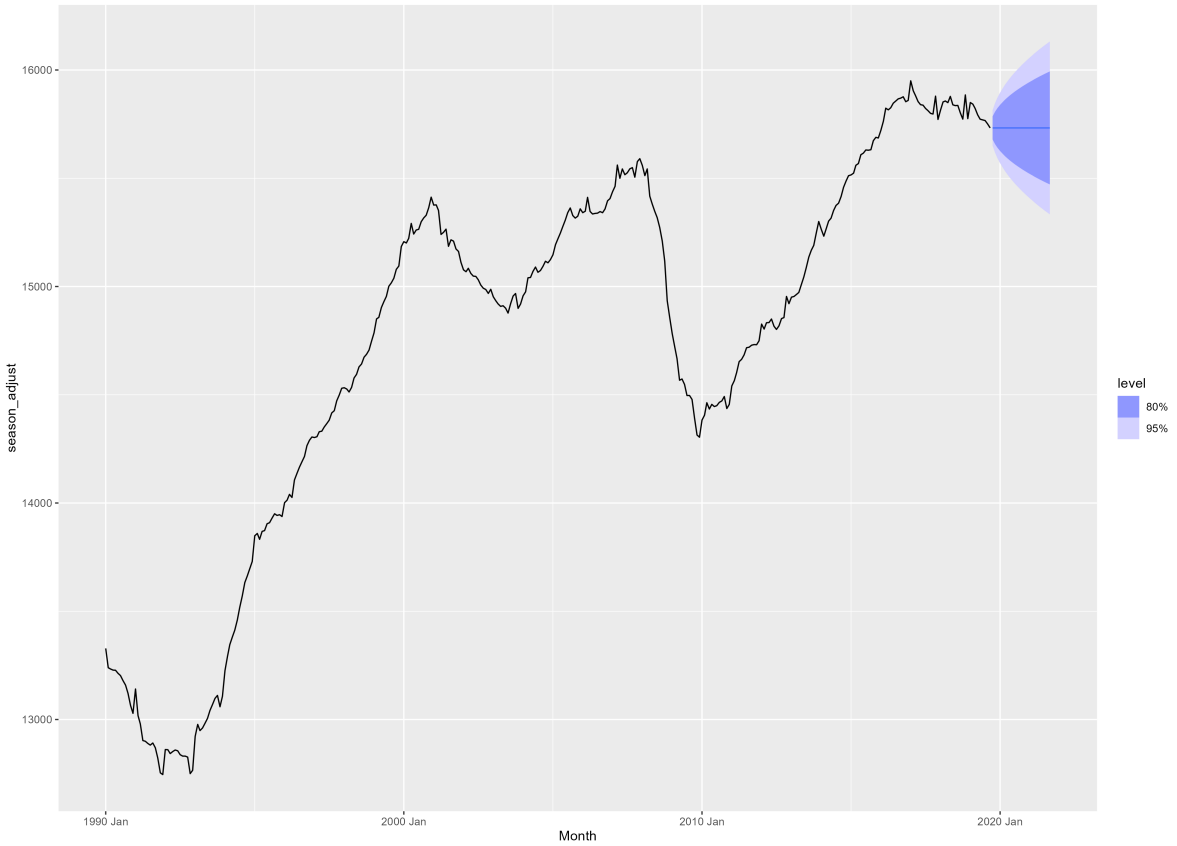
```r
us_retail_employment <- us_employment |>
  filter(year(Month)>= 1990, Title == "Retail Trade") |>
  select(Month, Employed)

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



```
dcmp <- us_retail_employment |>
  model(STL(Employed ~ trend(window = 7), robust = TRUE)) |>
  components()

dcmp |> autoplot()
```



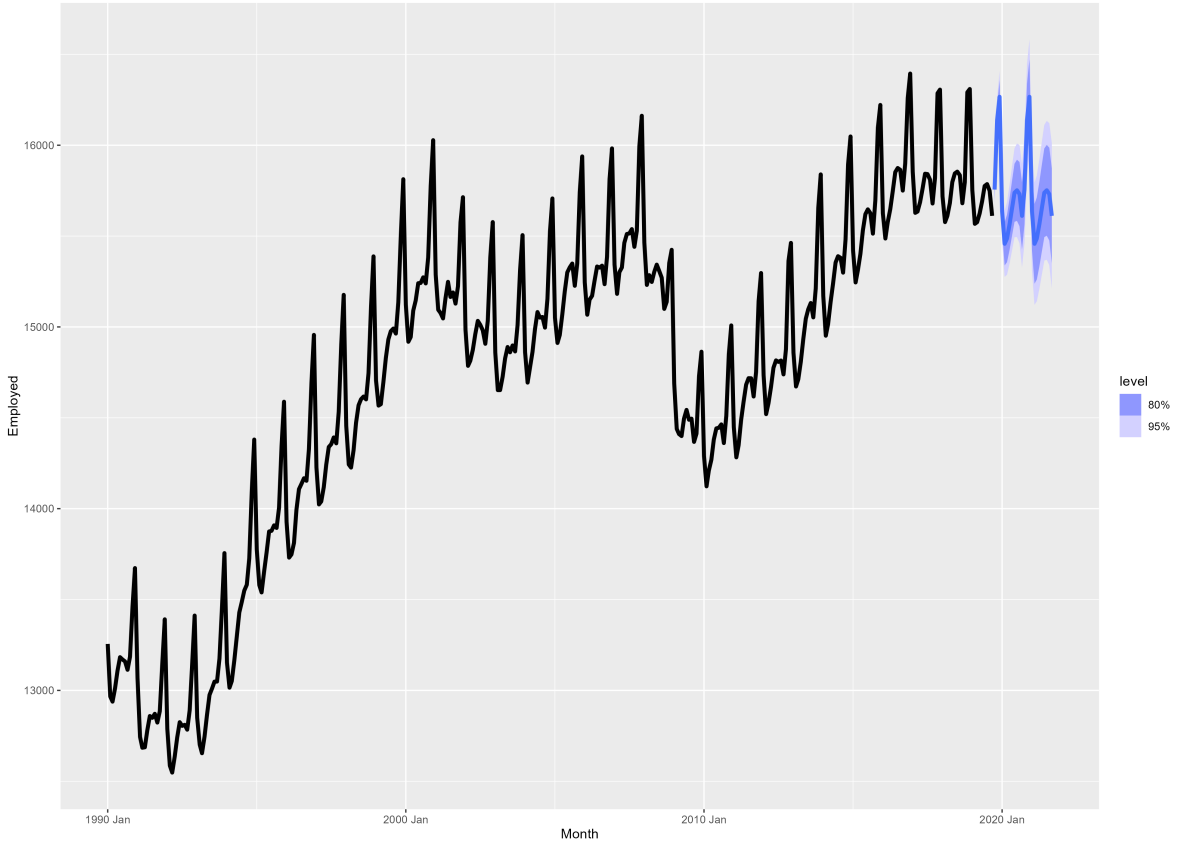
```
dcmp |>
  select(-.model) |> #should not have attribute called .model
  model(NAIVE(season_adjust)) |>
  forecast(h = 24) -> dcmpForecast

dcmpForecast |>
  autoplot() +
  autolayer(dcmp, season_adjust)
```



- In the above code we have decomposed the time series and created a naive model for the seasonally adjusted component.

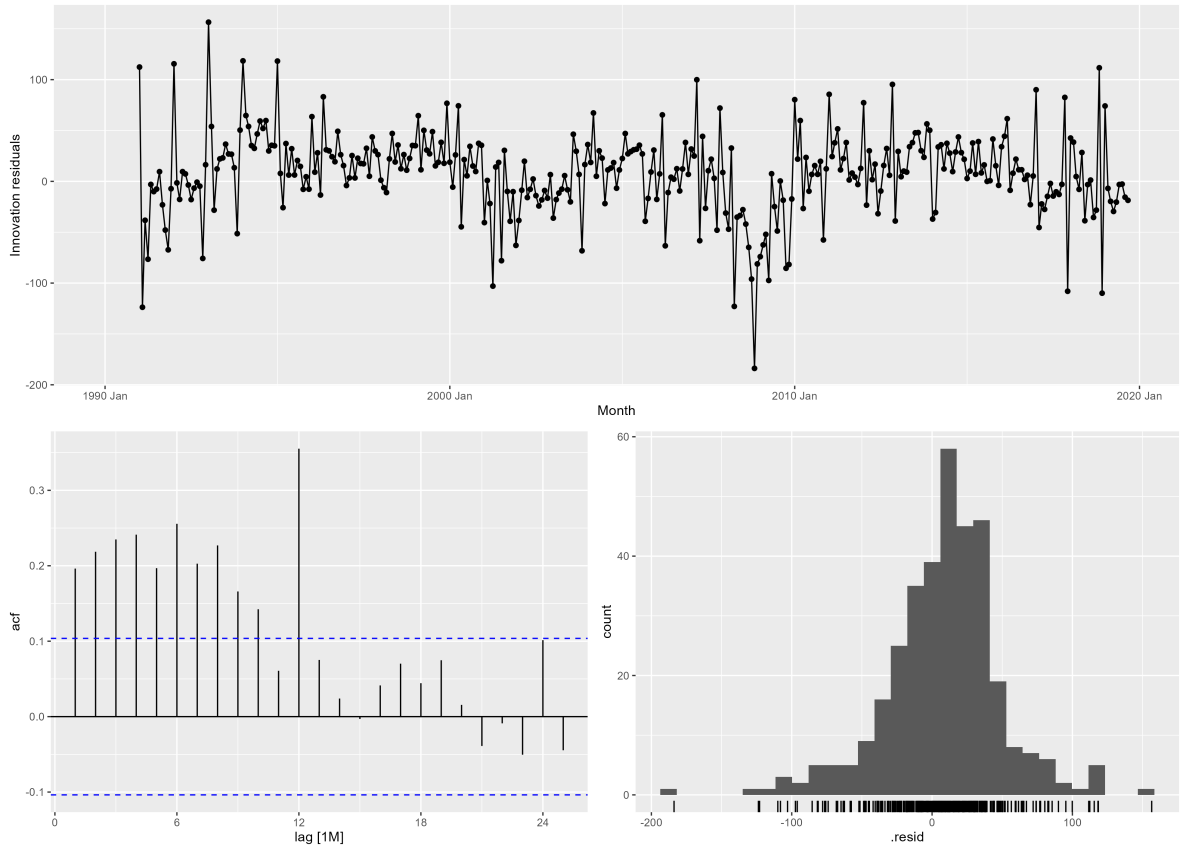
- The next step is adding the seasonal component. This can be done easily with the *decomposition_model* function, which allows you to compute the forecasts using any additive decomposition, using other models of functions to forecast the components. The function will also combine the forecasted values of the components.

```
forecast_dcmp <- us_retail_employment |>
  model(stlf = decomposition_model(STL(Employed ~trend(window = 7), robust =
      TRUE),
         NAIVE(season_adjust),
         SNAIVE(season_year ~ lag("year"))))
forecast_dcmp
forecast_dcmp |>
  forecast(h = 24) -> dcmp2Forecast
dcmp2Forecast |>
  autoplot() +
  autolayer(us_retail_employment, Employed)
```



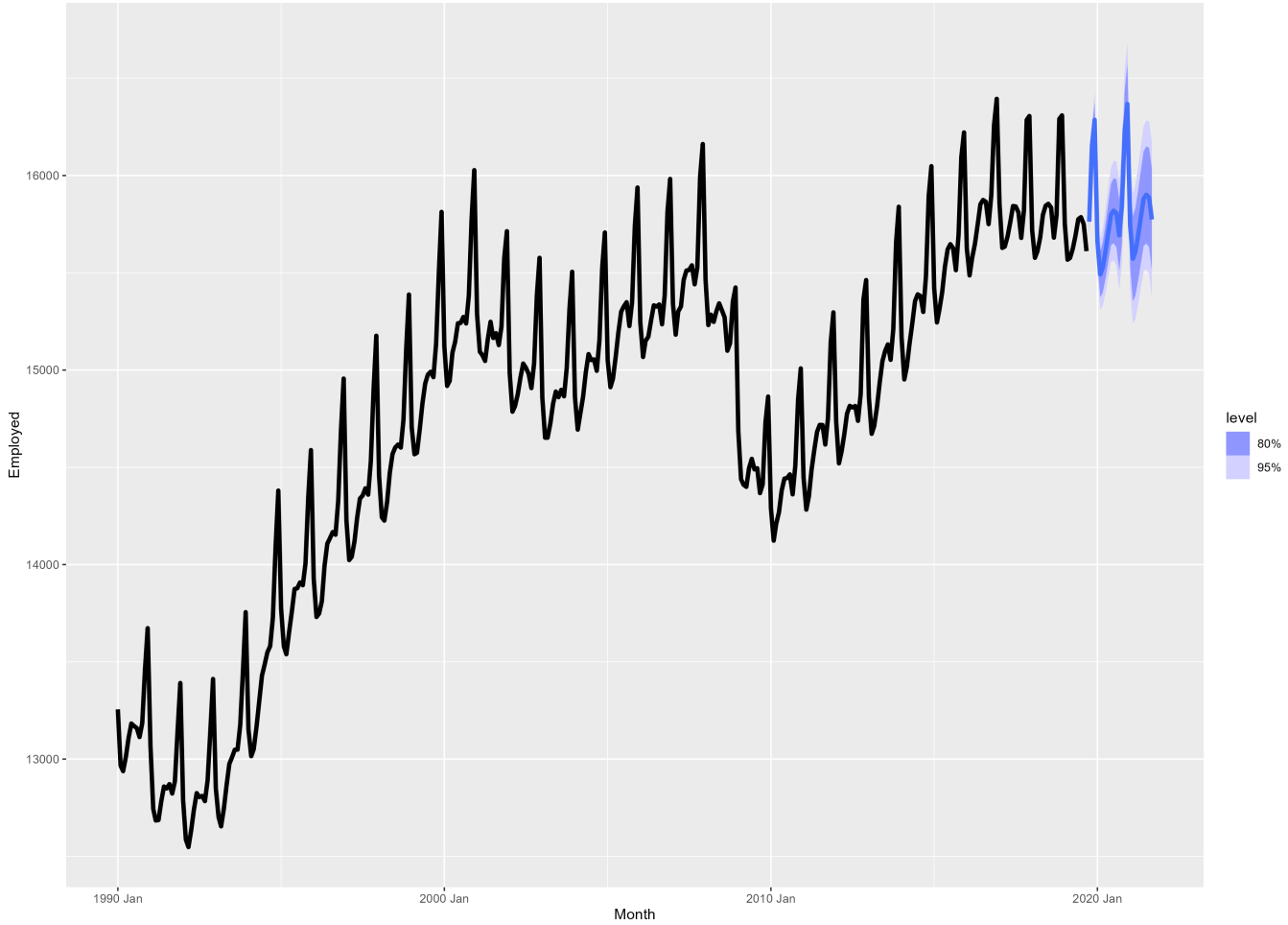
- The seasonal naive model might be omitted, if no other model is specified the *decomposition_model* will automatically use SNAIVE to forecast the seasonal component.
- Let's look at the residuals of the model, to see if this is a good model

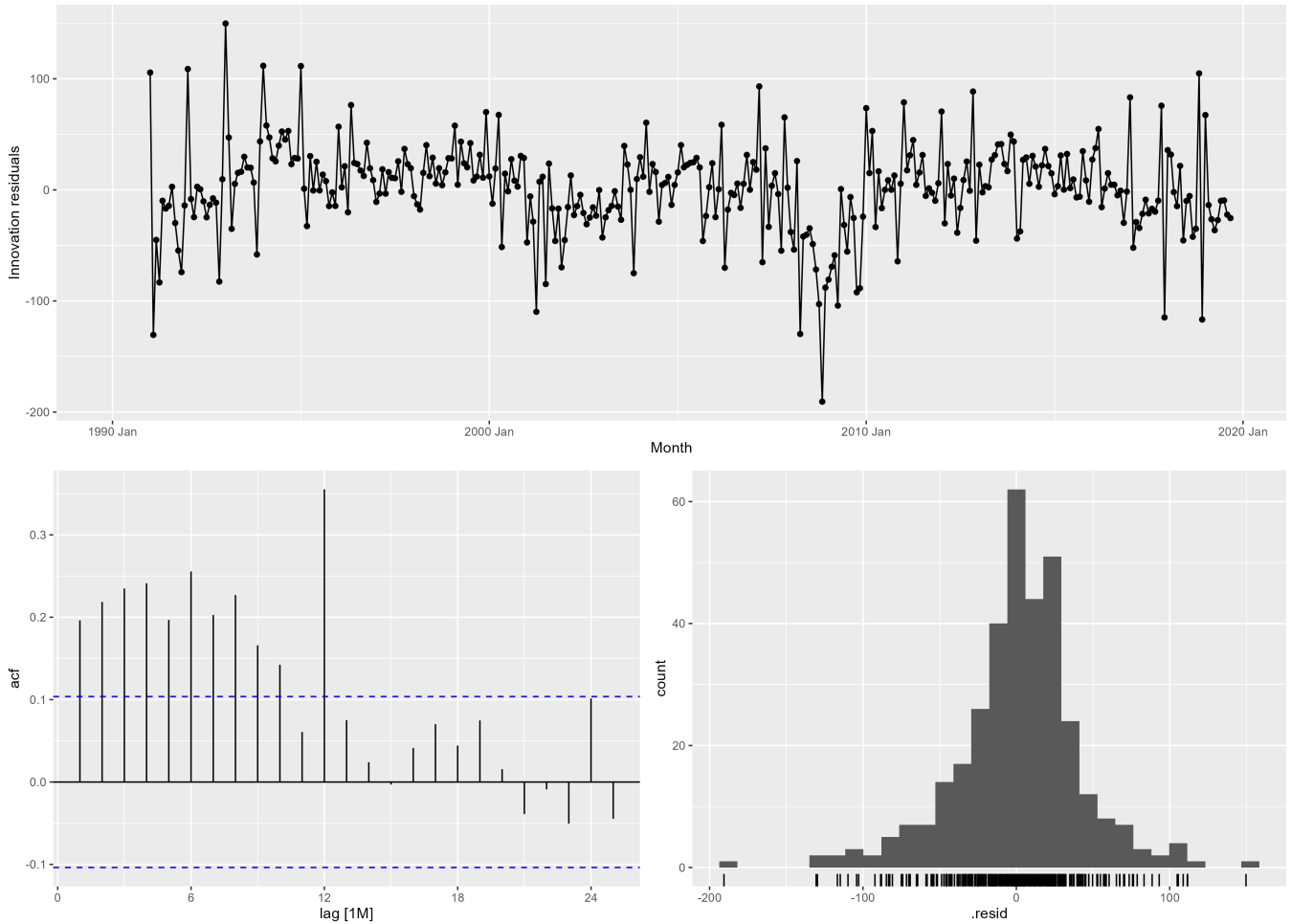
```
forecast_dcmp |> gg_tsresiduals()
```



- That is a lot of autocorrelation, suggesting that this is not a good model.
- Let's try to forecast separately all three components of the time series: the seasonal component with SNAIVE, the trend with the drift and the remainder with the NAIVE component.

```
forecast_dcmp2 <- us_retail_employment |>
  model(stlf = decomposition_model(STL(Employed ~trend(window = 7), robust =
      TRUE),
         RW(trend ~ drift()),
         NAIVE(remainder),
         SNAIVE(season_year ~ lag("year"))))
forecast_dcmp2 |>
  forecast(h = 24) -> dcmp2Forecast2
dcmp2Forecast2 |>
  autoplot() +
  autolayer(us_retail_employment, Employed)

forecast_dcmp2 |> gg_tsresiduals()
```


16000
15000
14000
13000
Employed
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month
level
80%
95%


Innovation residuals
100
0
-100
-200
1990 Jan
2000 Jan
2010 Jan
2020 Jan
Month
acf
0.3
0.2
0.1
0.0
-0.1
0
6
12
18
24
lag [1M]
count
60
40
20
0
-200
-100
0
100
.resid

Evaluating the accuracy of the forecast

- Residuals are computed for the data that is used to build the model. Their value is not a good indicator of true performance. For evaluation we need data that is not used for building the model $\rightarrow$ we split our data in *training* (used to estimate parameters and build the model) and *testing* (used for evaluating the model).
- The *test data*:
  - Should be at least as large as the maximum forecast horizon required.
  - In general is about 20% of the data (but this depends on how much data you have).
- **Obs:** Since we are talking about time series, the order of observations is important, we simply *cut* the time series at the required point to split in training and testing (no random shuffling).

- Similar to other domains where models are trained and predictions are made (ex. Machine Learning), in case of forecasting we have the following observations as well:
  - A model which performs well on training data will not necessarily have good performance on unseen test data.
  - You can always achieve good performance on your training data, if you use a model which is complex enough (overfitting)
  - Overfitting the training data is just as bad as not identifying real patterns in it.

- Functions that can be used to split data:
  - *filter* - you can specify conditions on the time index (ex. only data until year 2000; only data starting from 2000, etc.)
  - *slice* - allows the usage of indexes.
  - *head* and *tail*
- Consider the *ausbeer* data set containing information about quarterly beer production in Australia from 1956 to 2010 (only 2 quarters from 2010).

```
aus_beer <- aus_production |> select(Beer)
aus_beer |>
  filter(year(Quarter) < 1995)
aus_beer |>
  filter(year(Quarter) >= 1995)

aus_beer |> slice(0:100)
aus_beer |> slice(-1)
aus_beer |> slice(n())
aus_beer |> slice(5: n())
aus_beer |> slice(-200:0)
aus_beer |> slice(n()-19: 0)

trainSize <- dim(aus_beer)[1] * 0.8
testSize <- dim(aus_beer)[1] * 0.2
trainData <- aus_beer |> head(trainSize)
testData <- aus_beer |> tail(testSize)
trainData |> tail()
testData
```

Forecast error

- The *forecast error* is the difference between the predicted and the actual value. Considering that the training data is $\{y_1, \ldots, y_T\}$ and the testing data is $\{y_{T+1}, y_{T+2}...\}$ the error is $e_{T+h} = y_{T+h} - \hat{y}_{T+h}$
- The error of the prediction model can be computed by aggregating the forecast errors in different ways.
- **Obs:** Forecast error is not the same as the residual, for two reasons:
  - The residual is something you compute on the training data, while the error is computed on the test data
  - The residual always refers to one-step prediction, while in case of the error we can have multiple steps.

Scale dependent measures

- Two popular measures of forecast accuracy are: *Mean absolut error (MAE)* and *Root mean squared error (RMSE)* computed in the following way:

$$MAE = mean(|e_t|)$$

$$RMSE = \sqrt{mean(e_t^2)}$$

- Both MAE and RMSE are measures that should be minimised. A method that minimizes MAE will forecast the median, while a method minimizing the RMSE will forecast the mean.
- **Obs:** e_t, MAE and RMSE use the same units of measurement (scale) as the input data, so they are NOT suitable for comparing results on different data sets. They can be used to compare the performance of different models on the same data set or performance on different data sets using the same units of measurement.

Percentage errors

- The percentage error can be computed by $p_t = 100 \times \frac{e_t}{y_t}$.
- One measure built based on the percentage error is the *Mean absolute percentage error (MAPE)* which is computed in the following way:

$$MAPE = mean(|p_t|)$$

- The advantage of MAPE is that it can be used to compare results across different data sets.
- However, it has a few disadvantages:
  - You have a problem when y_t is 0 (error is either infinite or undefined).
  - You might also have a problem when y_t is close to zero (p_t being extremely small or large).
  - It assumes that the unit of measurement has a meaningful zero value.

Symmetric MAPE

- Another disadvantage of MAPE is that it has no upper bounds for over - predictions. Consider the following situation:
  - $y_t = 10$ and $\hat{y}_t = 0$ (under-forecast). In this case MAPE = 100%. Assuming data with positive values, $\hat{y}_t$ cannot be lower than that. However, in case of an over-forecast, there is no limit how big the error can get: if $\hat{y}_t = 100$, MAPE = 900%.
  - This property is also the reason why blindly optimizing MAPE of a model, might lead to a model which always under-forecasts (which might not be desirable in certain domains, for ex. sales).
- This is why *sMAPE* (symmetric MAPE) was proposed:

$$sMAPE = mean(200 * \frac{|y_t - \hat{y}_t|}{y_t + \hat{y}_t})$$

- sMAPE still suffers from the division by a number close to 0. And it can be negative.

Scaled errors

- An alternative for the percentage error is based on scaling the error considering the *training MAE* from a simple forecast method.
- In case of non-seasonal data, we can use the naive method as a simple forecast method and define the scaled error q_j in the following way:

$$q_j = \frac{e_j}{\frac{1}{T-1}\sum_{t=2}^{T}|y_t - y_{t-1}|}$$

- q_j is less than one if it comes from a better forecast than the average one-step naive method on the training data.
- In case of seasonal data, the seasonal naive method can be used as a simple forecast method for defining q_j:

$$q_j = \frac{e_j}{\frac{1}{T-m}\sum_{t=m+1}^{T}|y_t - y_{t-m}|}$$

- Then, the *mean absolute scaled error (MASE)* can be computed as:

$$MASE = mean(|q_j|)$$

- MASE is a scale-free error measure as well, it can be used to compare performance across different data sets.

- It is also possible to define a Root Mean Squared Scaled Error (RMSSE) in the following way:

$$RMSSE = \sqrt{mean(q_i^2)}$$

Examples

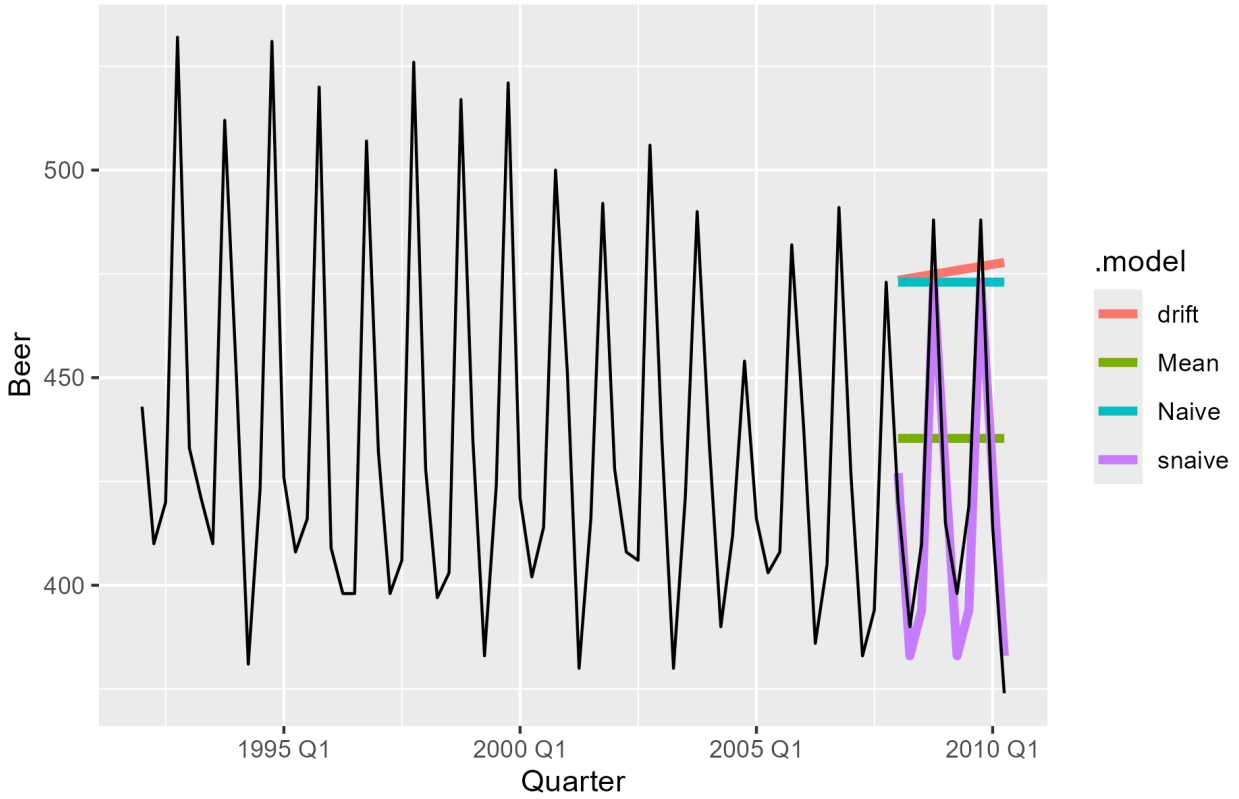
- We split part of the beer production data in two subsets: our training set will contain observations from 1992 to 2007, while the test set will contain observations from 2008 to the end (2010 Q2) - a total of 10 observation points.
- On the training data we will fit four simple models: the mean model, the naive model, the seasonal naive model and the drift model and get prediction for the next 10 observations
- We will plot them (original data - both for train and test period) and the four predictions to see how they fit
- We will evaluate the performance of these three models using the test data. For this we use the function *accuracy* from R, which displays the value of several error measures.

```r
allData <- aus_beer |>
  filter(year(Quarter) >= 1992)
trainData <- allData |>
  filter(year(Quarter) <= 2007)
testData <- allData |>
  filter(year(Quarter) > 2007)

beer_models <- trainData |>
  model(Mean = MEAN(Beer),
        Naive = NAIVE(Beer),
        snaive = SNAIVE(Beer),
        drift = RW(Beer ~drift()))

accuracy(beer_models) |>
  arrange(.model) |>
  select(.model, .type, RMSE, MAE, MAPE, MASE, RMSSE)
```

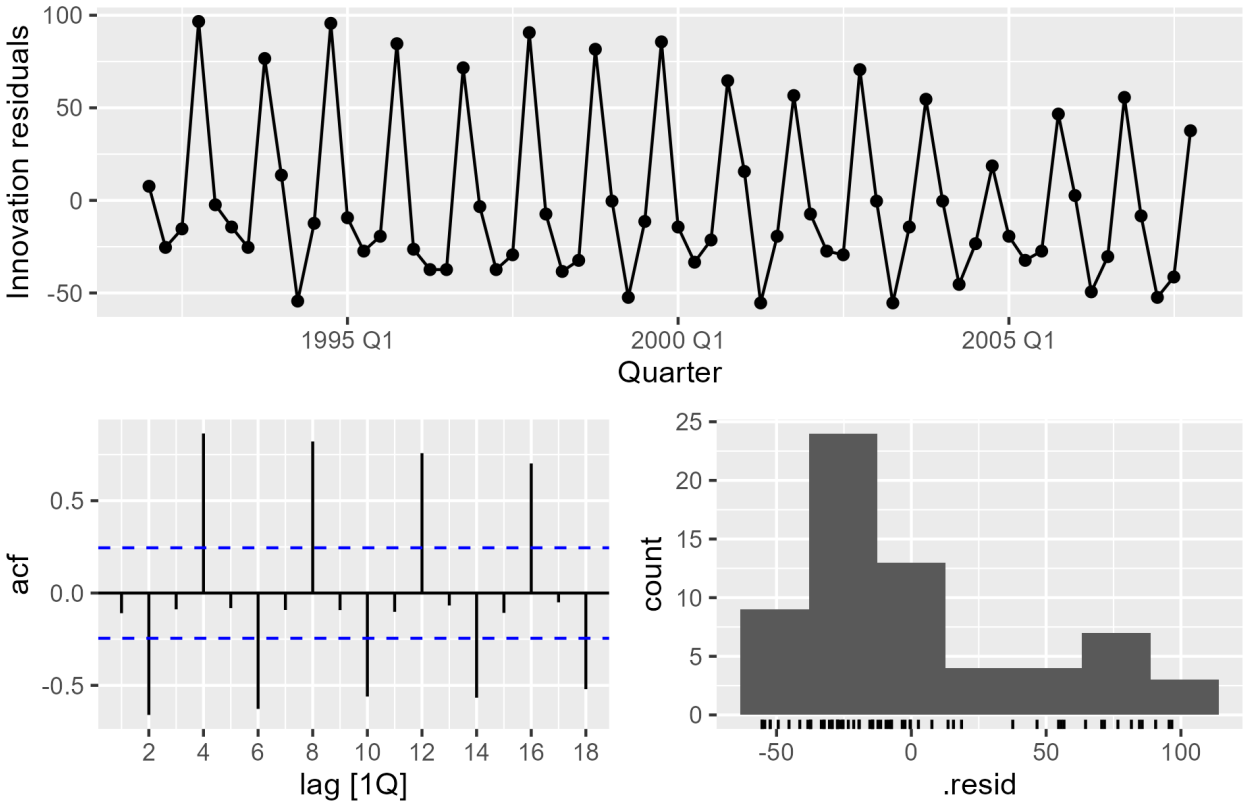
```
  .model .type      RMSE   MAE  MAPE  MASE RMSSE
  <chr>  <chr>     <dbl> <dbl> <dbl> <dbl> <dbl>
1 Mean   Training  43.6  35.2  7.89  2.46  2.60
2 Naive  Training  65.3  54.7 12.2   3.83  3.89
3 drift  Training  65.3  54.8 12.2   3.83  3.89
4 snaive Training  16.8  14.3  3.31  1     1
```

```
beer_forecast <- beer_models |> forecast(h = 10)

beer_forecast |>
  autoplot(allData, level = NULL) +
  guides(colous = guide_legend(title= "Forecast"))

beer_models[1] |> gg_tsresiduals()
beer_models[2] |> gg_tsresiduals()
beer_models[3] |> gg_tsresiduals()
```

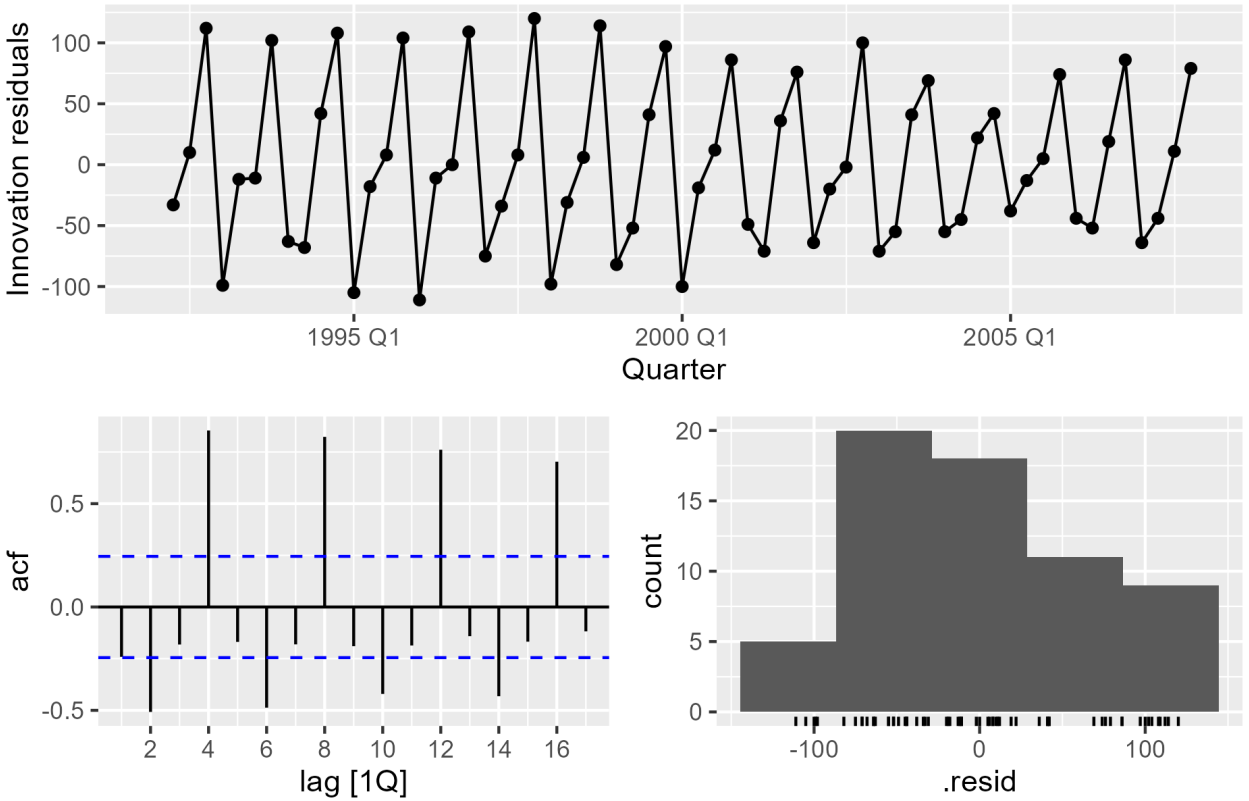
- Forecasts of the 4 simple models.



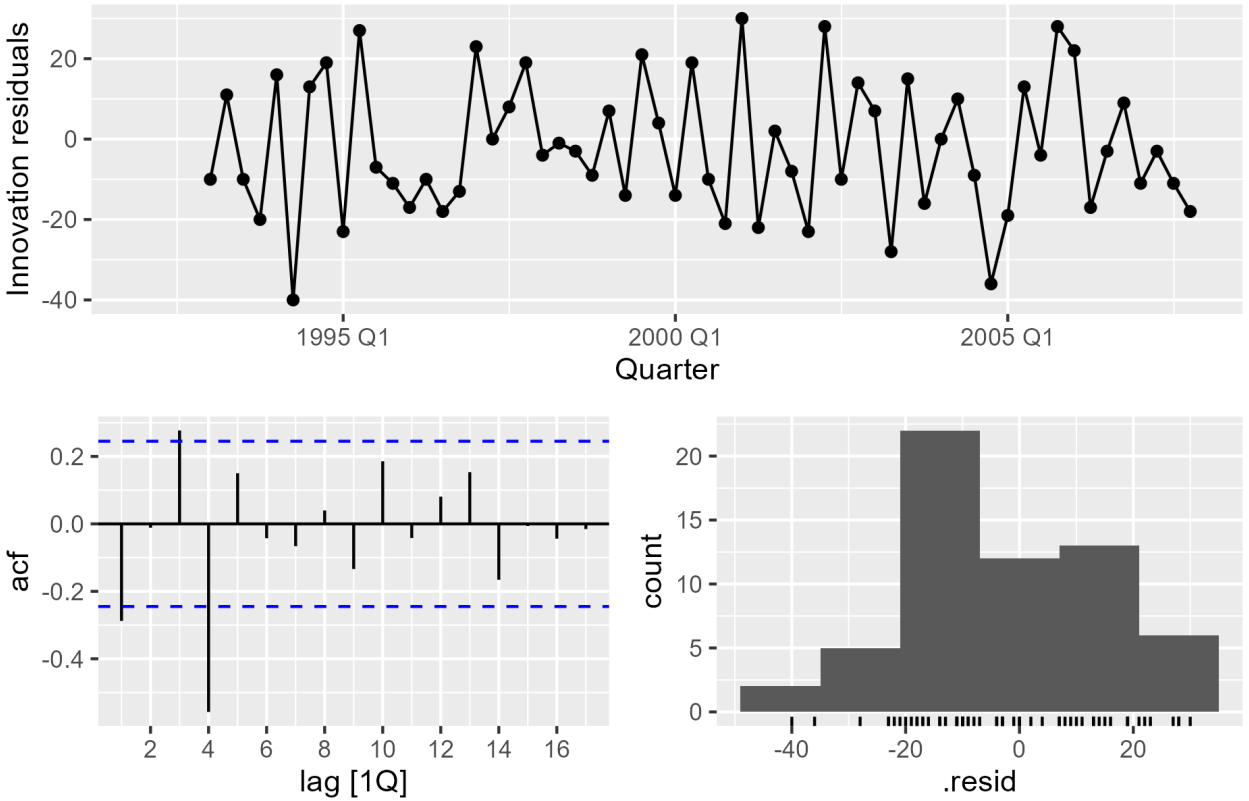
- Residual diagnostics of the mean model



- Residual diagnostics of the naive model.



- Residual diagnostics of the seasonal naive model.

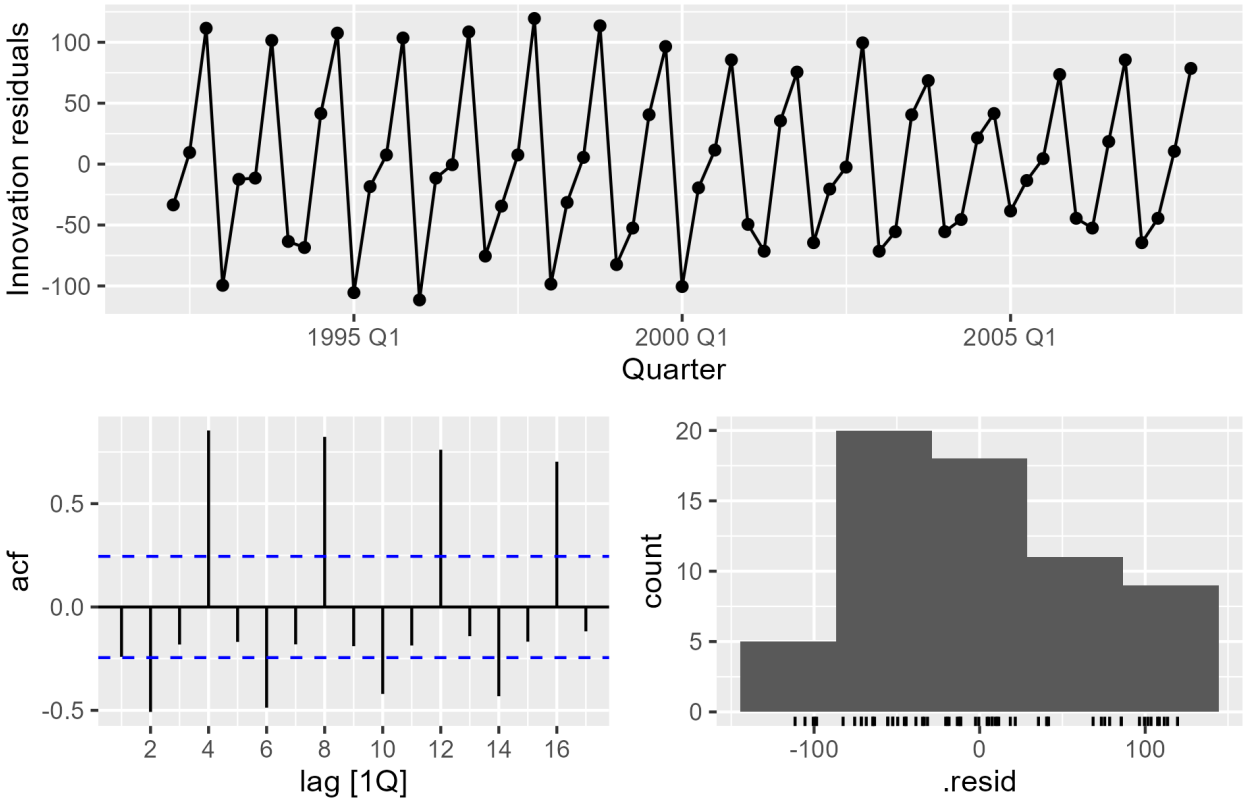


```
beer_models[3] |>  augment() |>
  features(.innov, ljung_box, lag = 8)
```

```
  .model lb_stat lb_pvalue
  <chr>    <dbl>     <dbl>
1 snaive    32.9 0.0000630

#not white noise
```

```
beer_models[4] |> gg_tsresiduals()

accuracy(beer_forecast, allData)
```

- Residual diagnostics of the drift model.



.model	.type	RMSE	MAE	MAPE	MASE	RMSSE
<chr>	<chr>	<dbl>	<dbl>	<dbl>	<dbl>	<dbl>
1 Mean	Test	38.4	34.8	8.28	2.44	2.29
2 Naive	Test	62.7	57.4	14.2	4.01	3.74
3 drift	Test	64.9	58.9	14.6	4.12	3.87
4 snaive	Test	14.3	13.4	3.17	0.937	0.853

- According to all values, the seasonal naive model seems to be the best. However, residual diagnostics show that there still is autocorrelation in the residuals, so the model did not capture all patterns from the data.