Forecasting with transformations and decompositions

EC 361–001

Prof. Santetti
Spring 2024

Materials

Required readings:

- `Hyndman & Athanasopoulos, ch. 6`
  - sections 6.6—6.7.

Motivation

Motivation

Before we move on into further exploring benchmark forecasting models, let us incorporate our lectures on **data transformation** and **decomposition methods** into forecasting exercises.

Regarding **transformations**, we use them whenever some adjustment is necessary to either stabilize the **variance** of our variable (e.g., *log-transforming*) or make it **better suited** for our practices (e.g., *adjusting for inflation*).

Second, learning time series **decomposition** methods allowed us to *break down* our variables into its *trend-cycle*, *seasonal*, and *remainder* components.

One of the main advantages of decompositions is obtaining **seasonally adjusted** data.

- But **how** to forecast with transformations and decompositions?

Forecasting with transformations

Forecasting with transformations

When using **transformations**, our forecasts will be produced on the *transformed* variable(s).

But for better communication, we should ***back-transform*** the data to bring the forecasts to the **original scale**.

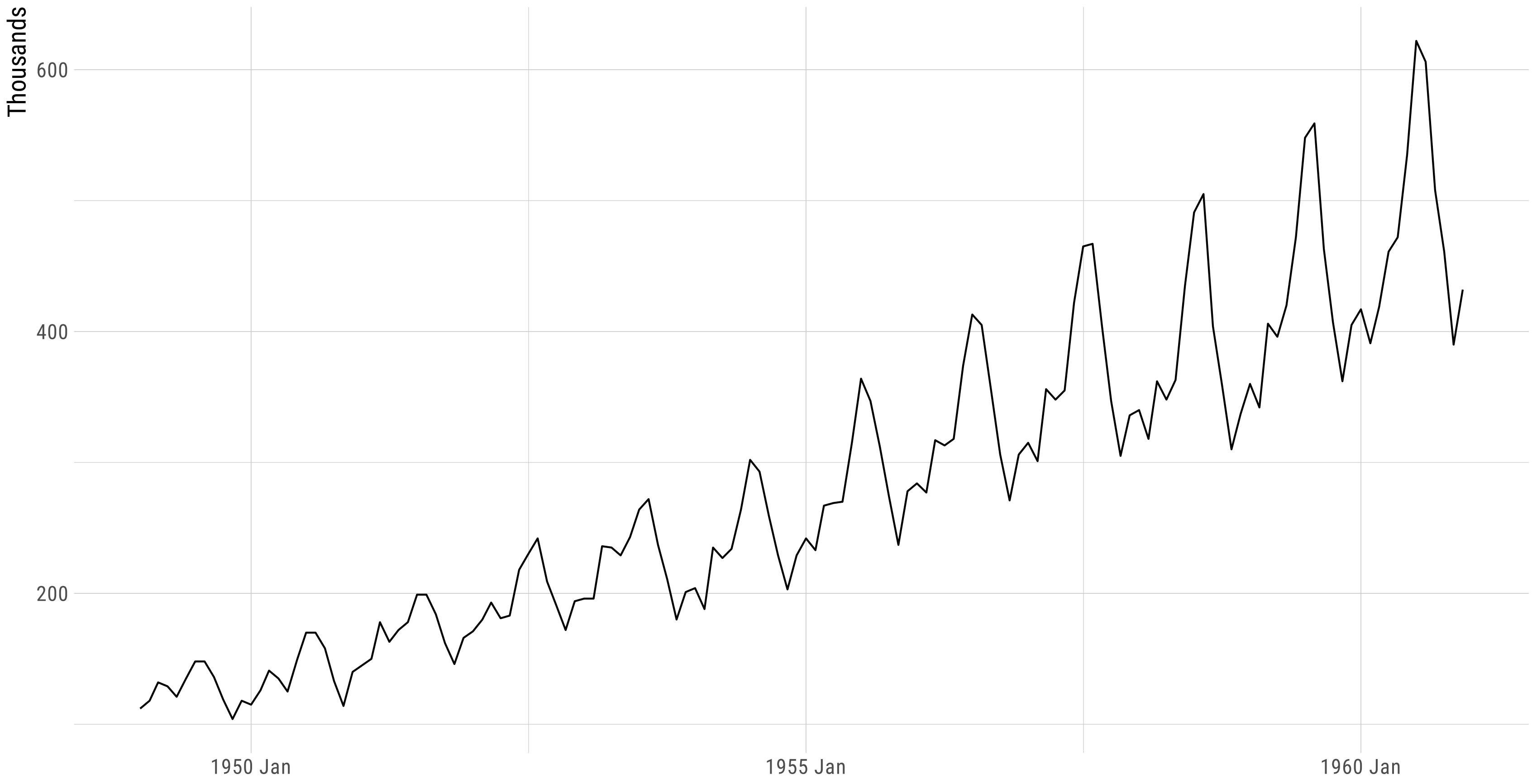
For our purposes, the `{fable}` package handles this back-transformation automatically, as long as we **explicitly** inform the transformation we have used in the model specification.

Let us better grasp this idea through an example.

Forecasting with transformations

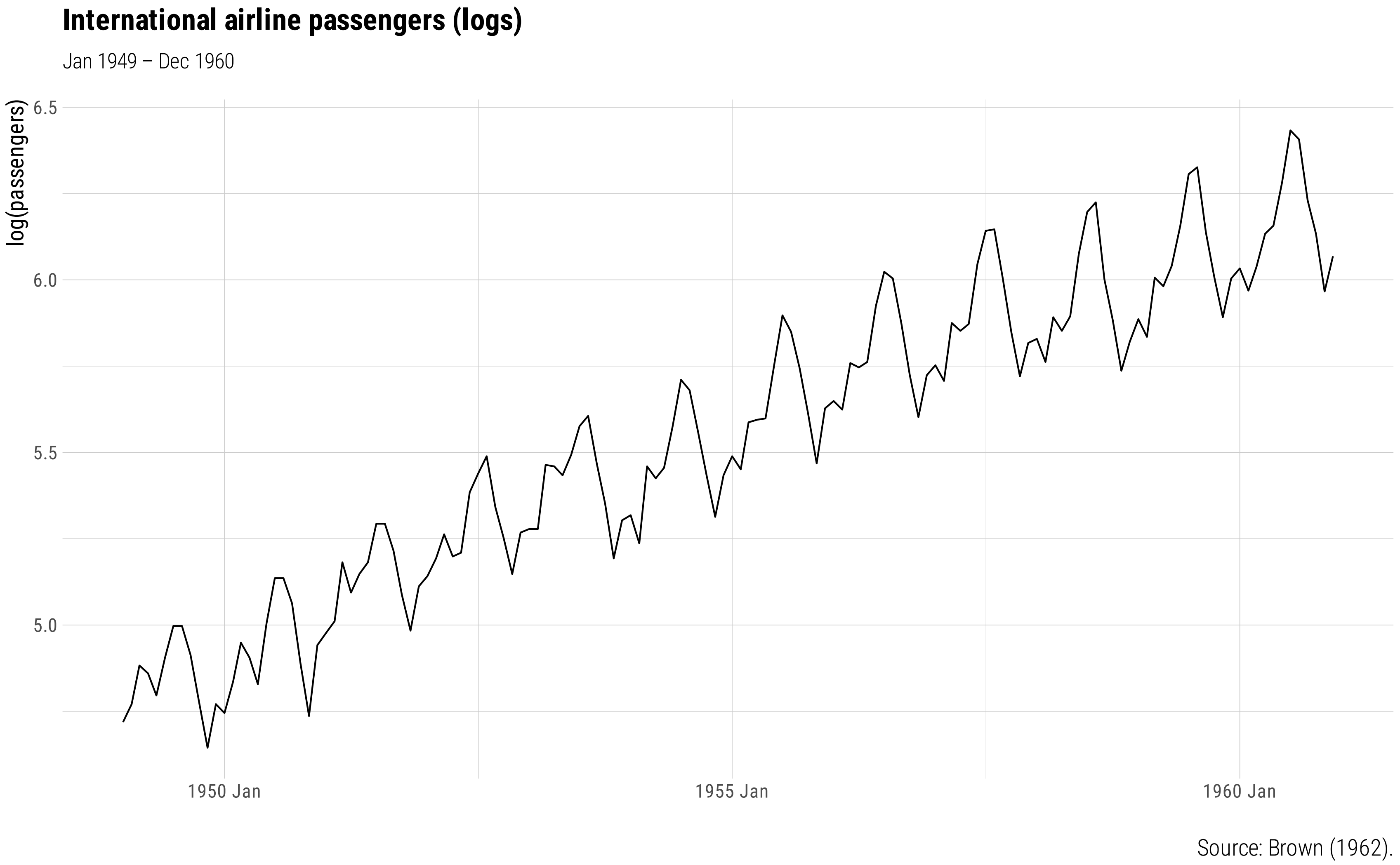
International airline passengers

Jan 1949 – Dec 1960



Source: Brown (1962).

Forecasting with transformations

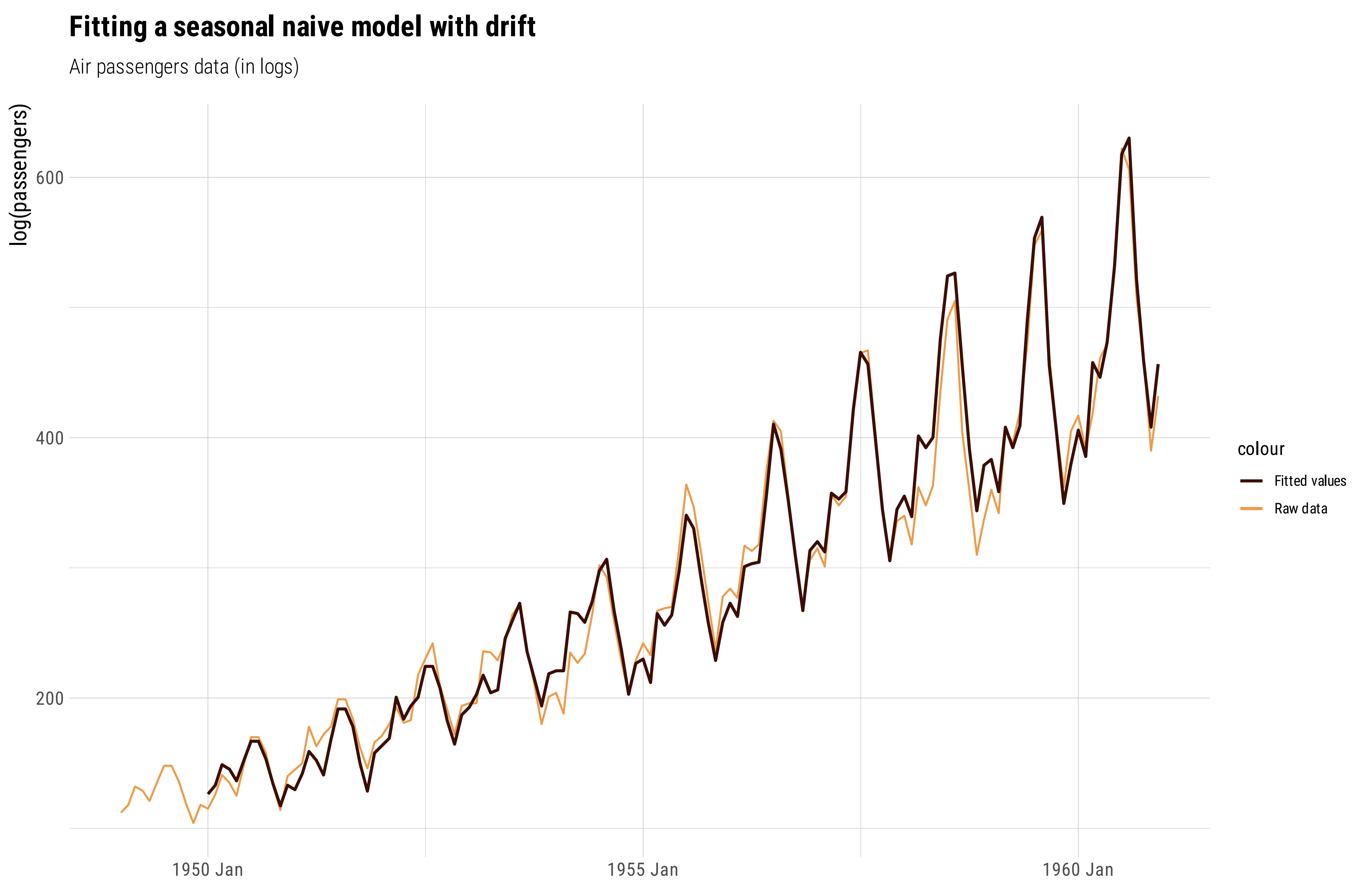


Forecasting with transformations

Recalling:

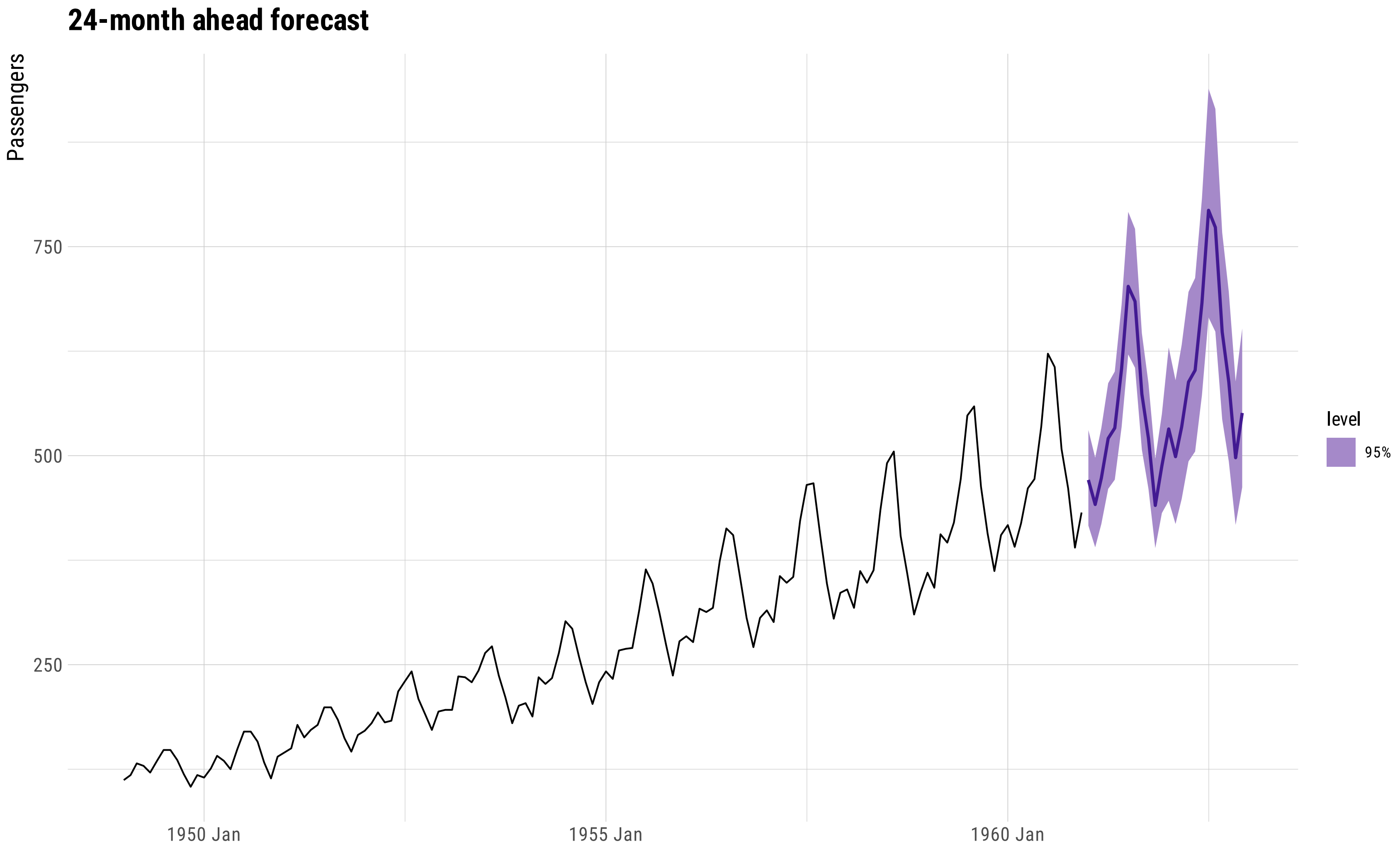
- For our purposes, the `{fable}` package handles this back-transformation automatically, as long as we **explicitly** inform the transformation we have used in the model specification.

```
air_snaive <- air_ts |>
  model(snaive_model = RW(log(passengers) ~ drift() + lag(12))) ## Fitting a seasonal naive model with drift.
```



Fitting a seasonal naive model with drift
Air passengers data (in logs)
log(passengers)
600
400
200
1950 Jan
1955 Jan
1960 Jan
colour
Fitted values
Raw data

Forecasting with transformations

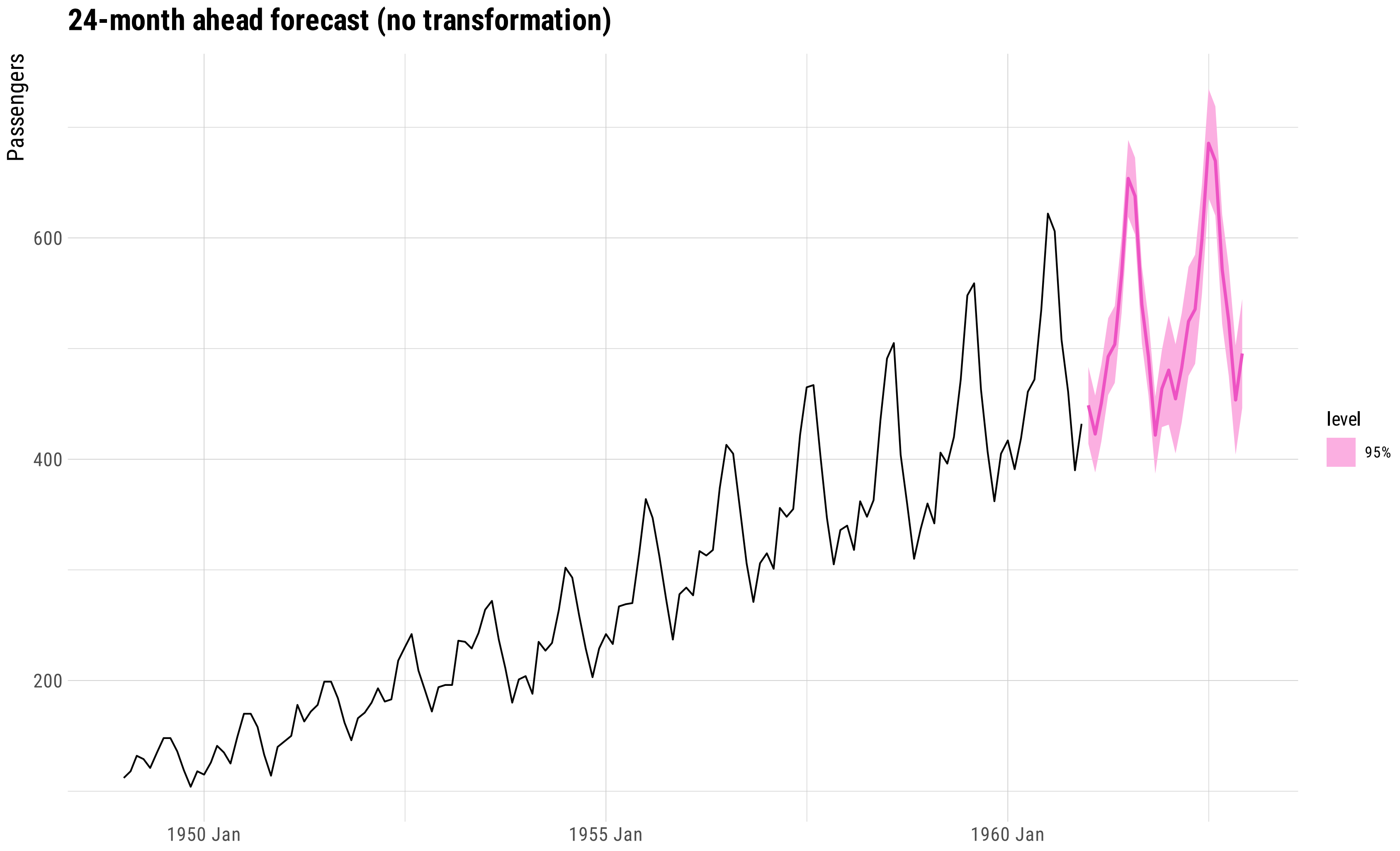


Forecasting with transformations

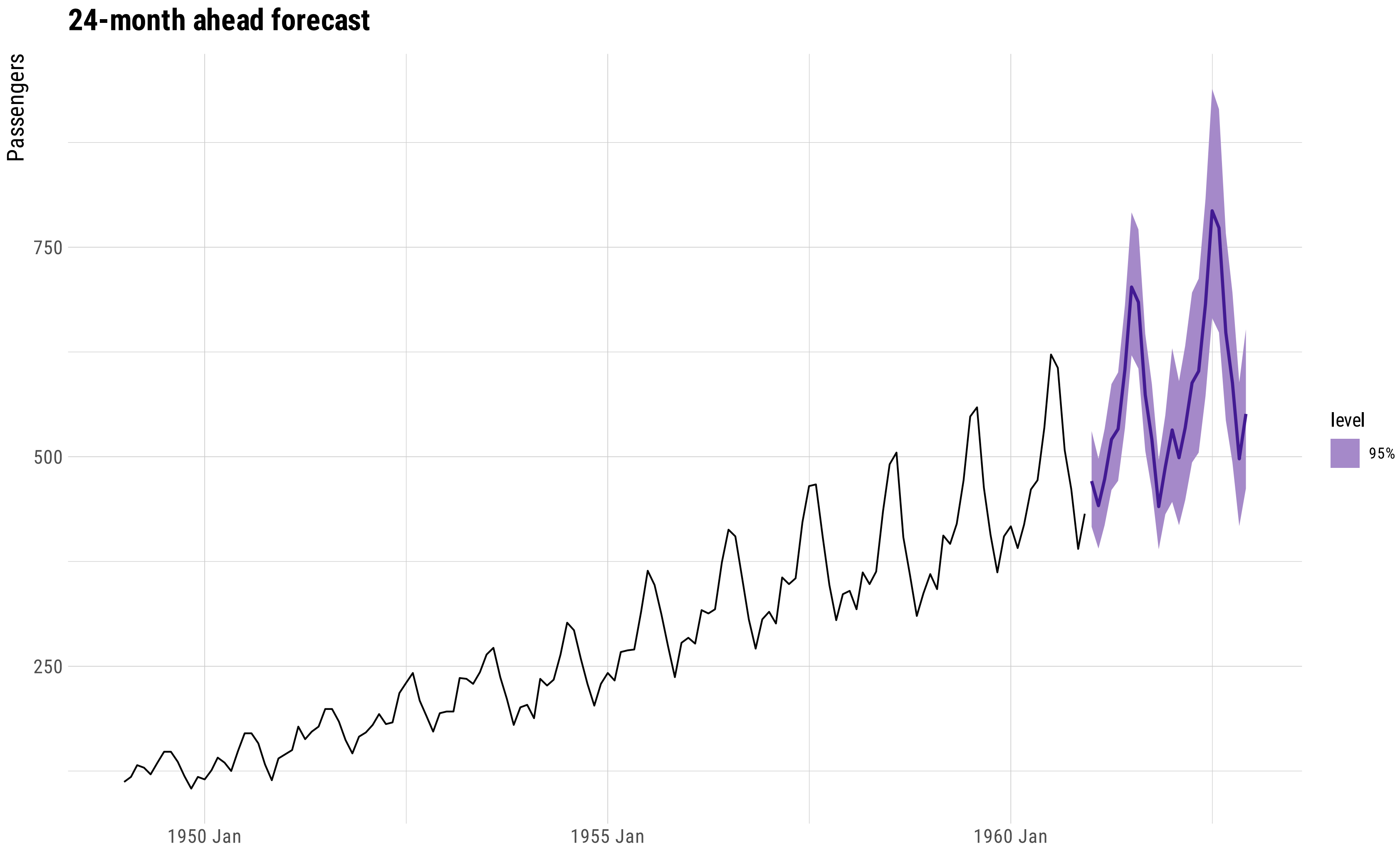
Now let us compare the **same model**, but **without** a transformation:

```
air_snaive_no <- air_ts |>
  model(snaive_model = RW(passengers ~ drift() + lag(12)))
```

Forecasting with transformations



Forecasting with transformations



Forecasting with transformations

If a transformation has been used, then the prediction interval is first computed on the *transformed* scale, and the end points are **back-transformed** to give a prediction interval on the original scale.

Thus, forecasts produced with transformations may generate prediction intervals that are **not symmetric**.

Transformations sometimes make little difference to the point forecasts, but have a large effect on **prediction intervals**.

Forecasting with decompositions

Forecasting with decompositions

Recalling a decomposed time series:

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

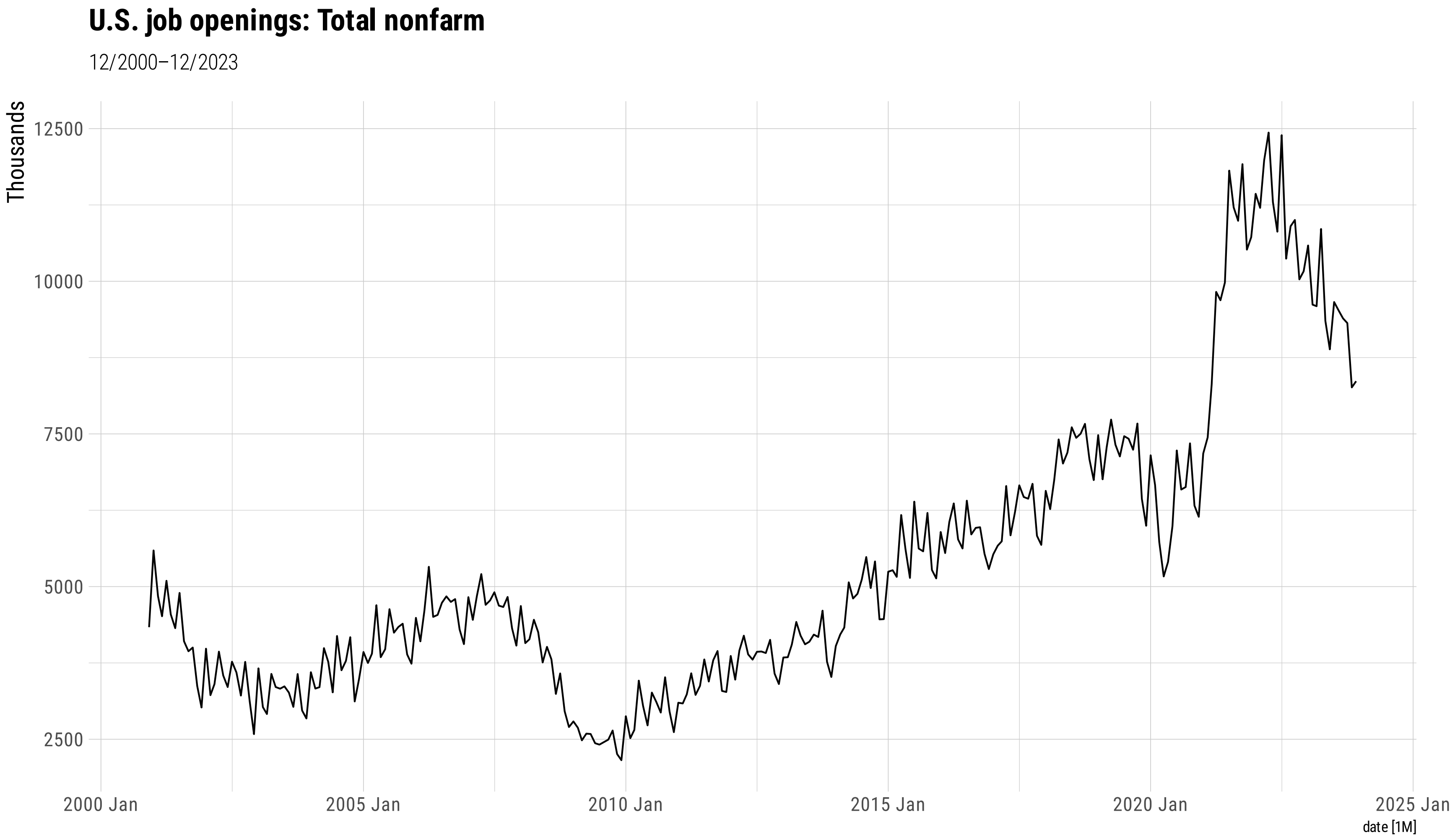
where $\hat{T}_t + \hat{R}_t$ make up the **seasonally adjusted** component, while $\hat{S}_t$ alone is the **seasonal** piece.

When one wants to forecast a decomposed time series, the seasonally adjusted and seasonal components are forecast **separately**, then *added together*.

For the **seasonal** component, the standard assumption is to use a **seasonal naïve** method, whereas any other **non-seasonal model** can be applied to forecast the **seasonally adjusted** portion.

Let us see an *example*.

Forecasting with decompositions

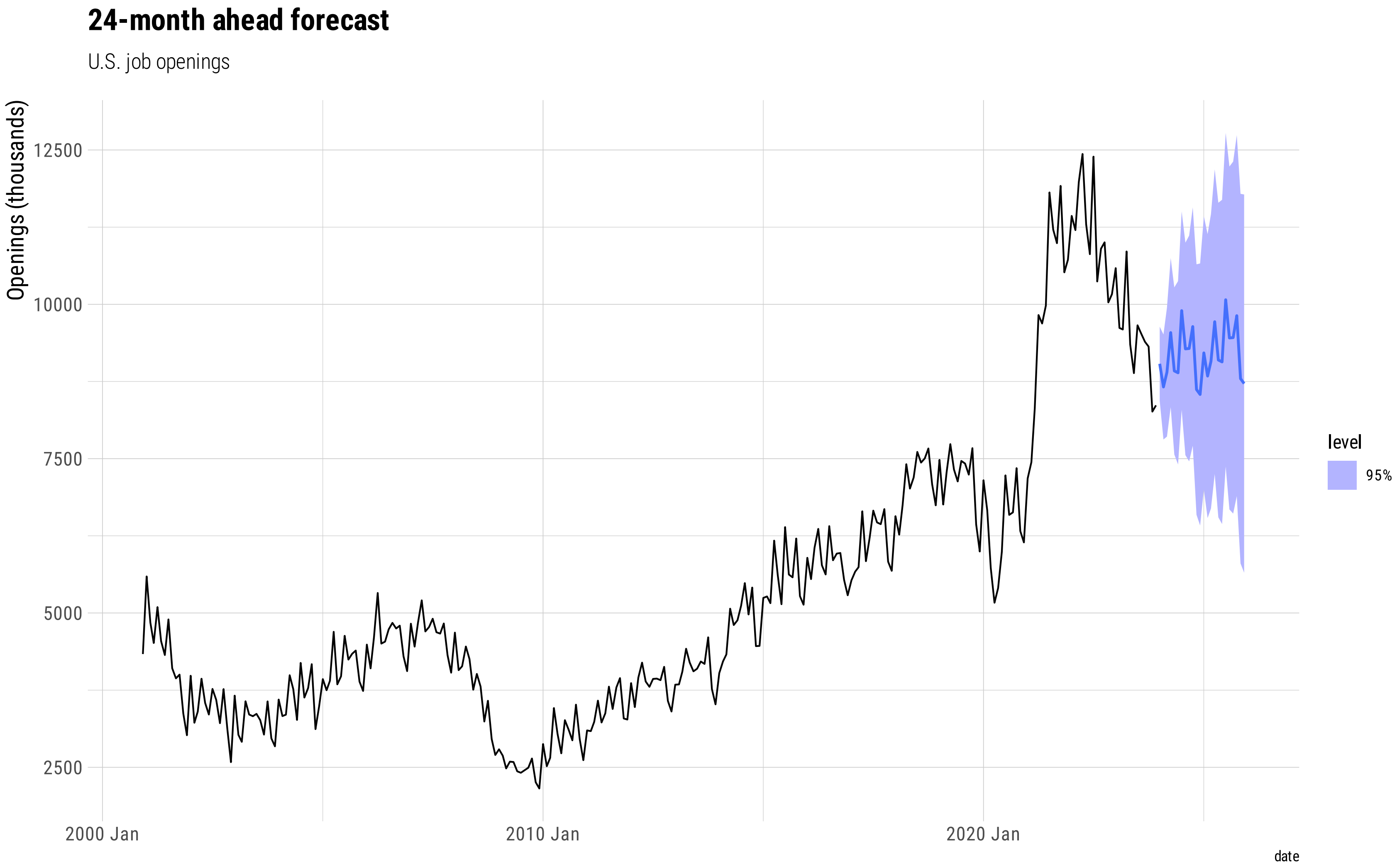


Source: U.S. Bureau of Labor Statistics.

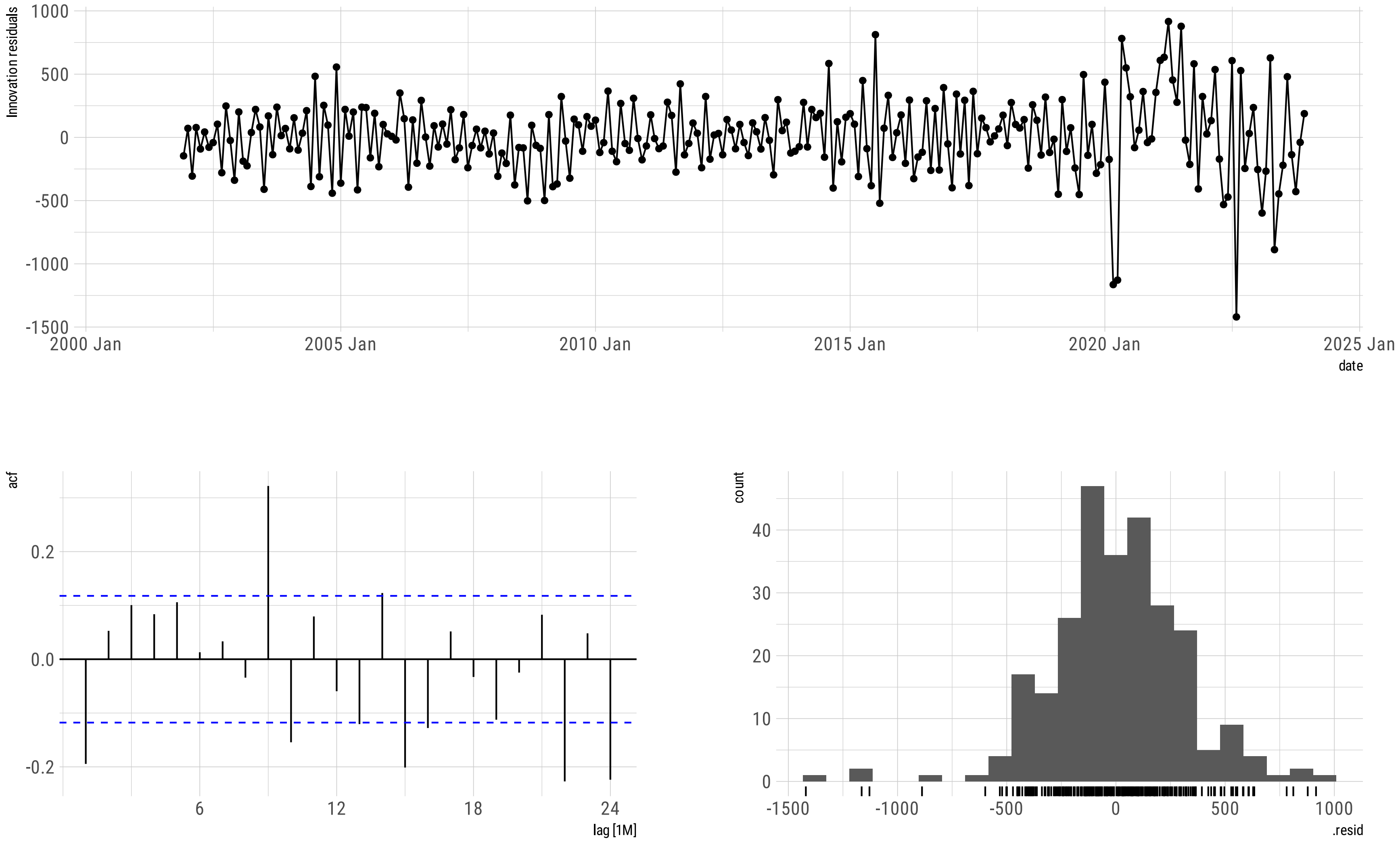
Forecasting with decompositions

```
job_decomp <- job_ts |>
  model(stlf = decomposition_model(STL(openings), ## applying an STL decomposition.
                                   RW(season_adjust ~ drift()), ## seasonally adjusted portion.
                                   SNAIVE(season_year))) ## can leave blank if we want.
```

Forecasting with decompositions



As in any other forecasting procedure, we should check our **residuals**:



Forecasting with decompositions

```
job_decomp |>
  augment() |>
  features(.innov, box_pierce, lags = 2 * 12)
```

```
#> # A tibble: 1 × 3
#>   .model bp_stat bp_pvalue
#>   <chr>    <dbl>     <dbl>
#> 1 stlf      10.0   0.00154
```

```
job_decomp |>
  augment() |>
  features(.innov, ljung_box, lags = 2 * 12)
```

```
#> # A tibble: 1 × 3
#>   .model lb_stat lb_pvalue
#>   <chr>    <dbl>     <dbl>
#> 1 stlf      10.1   0.00145
```

Next time: Forecast accuracy measures