Dublin Bus Passenger Analysis

Ujwal Mojidra

Table of contents

Introduction to the Project

The aim of this project is to study how Dublin Bus passenger numbers have changed over the last decade and use that trend to build a simple prediction model. Public transport is a good candidate for time-series analysis because it naturally carries seasonal patterns, long-term growth or decline, and changes due to external events like COVID-19.

What this analysis really tries to answer is:
"How did Dublin Bus usage evolve from 2014 to 2024, and can we model or forecast what happens next?"

To get there, I first break the dataset into two parts:

- **Yearly totals**, which show the overall trajectory of bus usage.
- **Monthly values**, which reveal seasonality and high-frequency patterns.

Once the structure is clear, I can run exploratory plots, identify jumps or dips, and finally apply a time-series model (likely ARIMA or ETS) to see how well it predicts future passenger numbers. The end goal is simple: test a forecasting method on real Irish transport data, understand its behaviour, and compare the predicted values with actual trends to judge performance.

Libraries and packages for the Project.

```
library(readr)
library(dplyr)
library(tidyr)
library(ggplot2)
library(tibble)
library(stringr)
library(knitr)
library(reshape2)
library(tseries)
library(forecast)
```

These libraries support the workflow from data loading to visualisation:

- `readr` — fast, reliable reading of CSV files. Helps avoid formatting issues.
- `dplyr` — the core toolkit for filtering rows, selecting variables, grouping by year, and cleaning data.
- `tidyr` — for reshaping data if needed (though light usage here).
- `ggplot2` — the primary plotting library for creating yearly and monthly trend graphs.
- `stringr` — useful for cleaning Month names and ensuring consistent formatting.
- `tibble` — provides tidy printing and structured tables.
- `knitr` — controls how tables or summaries are printed inside the PDF.
- `forecast` — For time series forecasting models like *ARIMA and TBATS*.
- `tseries` — Provides statistical tests essential for time-series analysis.

These tools together cover almost everything required: load data, clean it, structure it, plot it, and interpret it.

Part 1: Manipulation

Load `dublin_bus_data` dataset.

This dataset is available from the Central Statistics Office.[1]

```
bus <- read_csv("DublinBus.csv")
```

Before any modelling, the dataset needs to be organised into a structure that makes sense for analysis. Since the raw file mixes both month-level rows and a yearly summary labelled as "All months," the first step is to separate them. This helps us view the big picture (yearly totals) and the detailed behaviour (month-level changes).

Cleaning also ensures the Month column is properly ordered instead of being alphabetical, which would distort any time-series plot. Once the datasets are organised, visualising them gives a straightforward sense of how Dublin Bus demand has changed over time.

Organize the Dataset by Year and Visualize Trends.

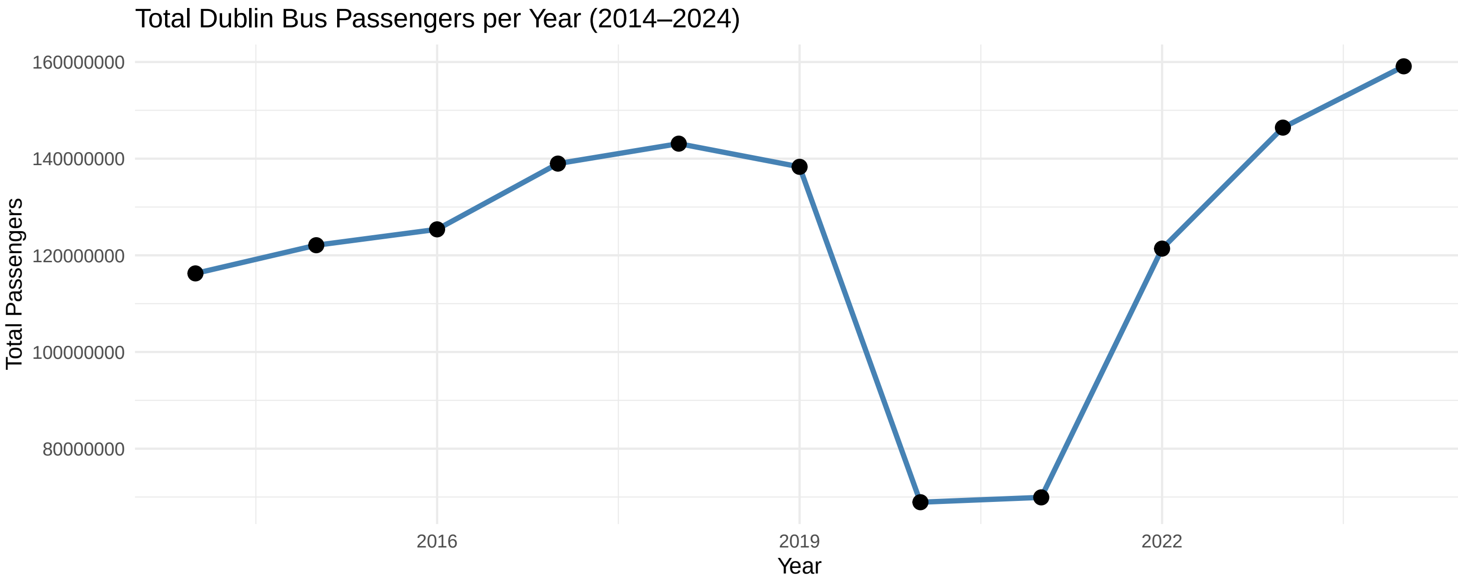
```
bus_yearly <- bus |>
  filter(Month == "All months") |>
  arrange(Year)
```

Finding some trend in yearly data.

Let's see the yearly trends across the bus network.

```
options(scipen = 10)
ggplot(bus_yearly, aes(x = Year, y = VALUE)) +
  geom_line(linewidth = 1.2, color = "steelblue") +
  geom_point(size = 3) +
  labs(
    title = "Total Dublin Bus Passengers per Year (2014-2024)",
    x = "Year",
    y = "Total Passengers"
  ) +
  theme_minimal()
```

[1] The Dublin bus data set for the project (Year 2014-2024): https://data.cso.ie/table/TOA14



This plot gives a decade-level view of how Dublin Bus usage has shifted over time. The first thing that stands out is the steady climb from 2014 to around 2018–2019. That's typical of a growing city: population increases, more commuters rely on buses, and service frequency expands.

Then the graph falls off a cliff in 2020. That drop isn't random — it reflects the impact of COVID-19, lockdowns, and restricted movement. Almost every public transport system in the world saw the same collapse, so this sharp decline validates the dataset rather than raising any concerns.

From 2021 onward, the recovery is visible. It's not instant; there's a slow rebuild in 2021, a strong jump in 2022, and by 2024 the numbers almost return to the pre-pandemic peak. This pattern tells you two things:

1. The system is resilient.
2. Passenger numbers respond quickly once restrictions lift.

This yearly view sets the foundation for forecasting later. If the rebound continues at the same pace, future passenger counts might even surpass earlier highs.

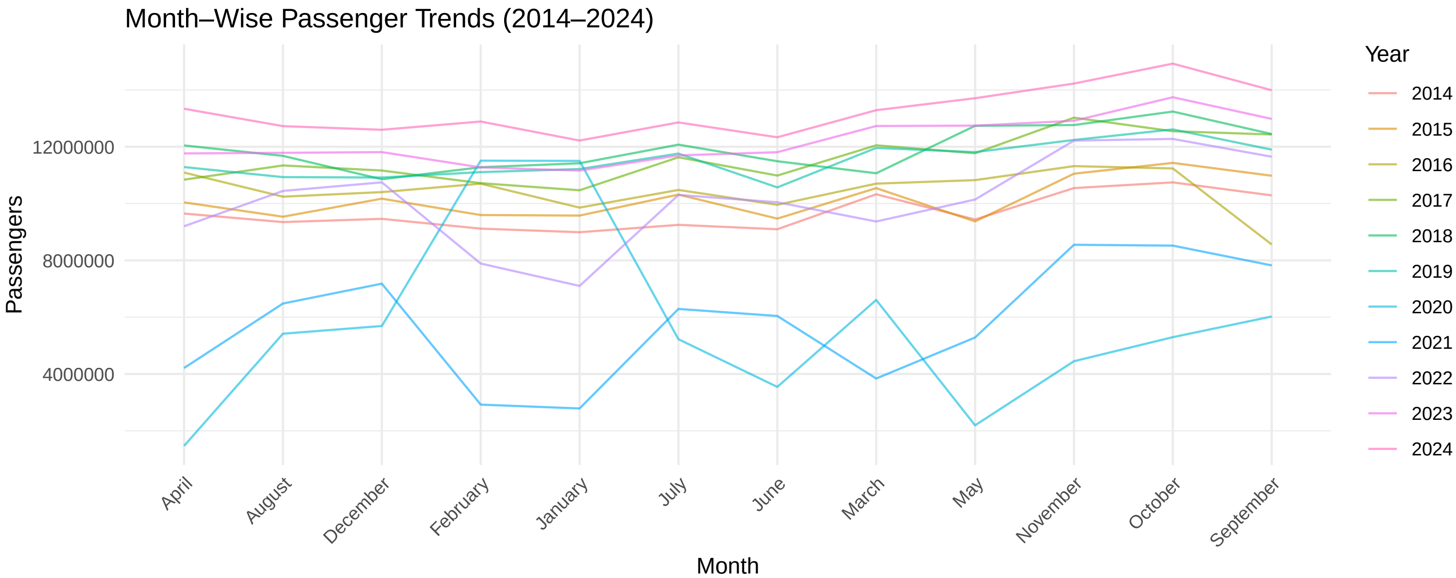
Organize the Dataset by Month and Visualize Trends.

```
bus_monthly <- bus |>
  filter(Month != "All months")
```

Finding some trend in monthly data.

Let's see the monthly trends across the bus network.

```
options(scipen = 10)
bus_monthly <- bus |>
  filter(Month != "All months")

ggplot(bus_monthly, aes(x = Month, y = VALUE, group = Year, color = factor(Year))) +
  geom_line(alpha = 0.6) +
  labs(
    title = "Month-Wise Passenger Trends (2014-2024)",
    x = "Month",
    y = "Passengers",
    color = "Year"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



This plot switches from yearly behaviour to monthly patterns. Here, every line represents a full year, and the spread shows how seasonality, disruptions, or external factors impact ridership.

Strong patterns emerge immediately:

- **The top cluster (2018–2019, 2023–2024)** shows the high-demand years.
- **The flat, low-lying 2020–2021 lines** match the pandemic slowdown seen in the yearly plot.
- **Most normal years share the same general shape**, which tells you that bus usage follows a seasonal cycle: some months reliably draw higher ridership, and others consistently dip.

You can also see that even though the months are plotted in alphabetical order in the figure (as your dataset names are currently structured), seasonal effects still appear: mid-year behaviour contrasts with winter months.

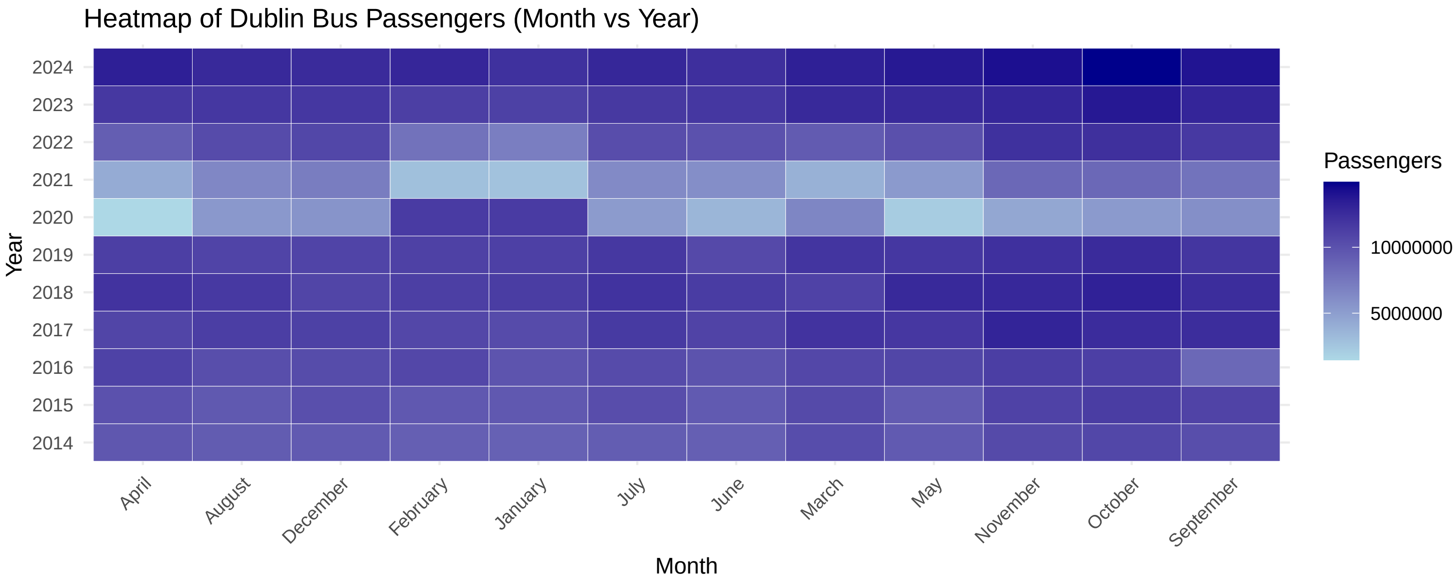
Once months are ordered correctly (Jan → Dec), this plot will reveal even cleaner seasonal waves.

This graph is also extremely useful for forecasting, because any time-series model needs to understand the repeating seasonal structure. The differences between pre-COVID, COVID, and post-COVID years also give a strong signal of structural breaks — something a model like ARIMA or ETS must account for.

Finding trends in combined datasets.

Till now we observed some similar insights in both of the datasets, lets play with heat map and find out what we get from it (expecting some hidden insights)

```
options(scipen = 10)
ggplot(bus_monthly, aes(x = Month, y = factor(Year), fill = VALUE)) +
  geom_tile(color = "white") +
  scale_fill_gradient(low = "lightblue", high = "darkblue") +
  labs(
    title = "Heatmap of Dublin Bus Passengers (Month vs Year)",
    x = "Month",
    y = "Year",
    fill = "Passengers"
  ) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



This heatmap gives a quick, intuitive view of how passenger volumes vary across both months and years. Instead of tracing lines or comparing separate plots, you can see the entire decade's behaviour in one glance.

The colour intensity does most of the talking here. Darker shades represent higher passenger counts, while lighter shades show dips or unusual drops.

A few patterns stand out immediately:

- **2020 and 2021 appear as the lightest rows**, confirming the massive collapse in ridership during the COVID-19 period. This matches the sharp drop seen in the yearly trend plot.
- **Years like 2018, 2019, 2023, and 2024 are consistently darker**, signalling high demand and strong bus usage.
- **Across most years, the month-to-month colour pattern looks fairly stable**, which means Dublin Bus follows a reliable seasonal rhythm every year. Even though the months are currently in alphabetical order in the plot, the colour distribution still hints at recurring patterns in certain months.
- The **progressive darkening from 2022 to 2024** shows the system recovering back to pre-pandemic levels, not just yearly but month-by-month.

What this heatmap really delivers is clarity: it compresses the entire dataset into a visual fingerprint of Dublin Bus usage over time. This makes it much easier to spot structural breaks, seasonal consistency, and recovery trends — all of which will matter once we start building the forecasting model.

Part 2: Time-Series Preparation & Decomposition

Phase 2 shifts the analysis from raw trends to a structured time-series approach. The goal here is to convert the monthly passenger data into a format that a forecasting model can understand. Once the data is converted into a proper time-series object, we can break it into its components: trend, seasonality, and noise; which gives us a clear picture of what drives bus demand over the years. This step is essential because the strength and stability of these components will decide which forecasting method works best in Phase 3.

Convert to a Time-Series Object

A forecasting model can't work on a normal data frame.
It needs a structured object with:

- a **start point** (2014, month 1)
- a **frequency** (12 for monthly data)
- a **uniform spacing** (no missing months)

Once converted, R can apply decomposition, autocorrelation checks, and forecasting techniques.

```r
ts_bus <- ts(bus_monthly$VALUE, start = c(2014, 1), frequency = 12)

# ts_bus
```

Decompose the Time-Series (Classical & STL)

Why decomposition?

Here's the thing: before forecasting, we need to see what the data is actually made of. Decomposition splits the passenger numbers into:

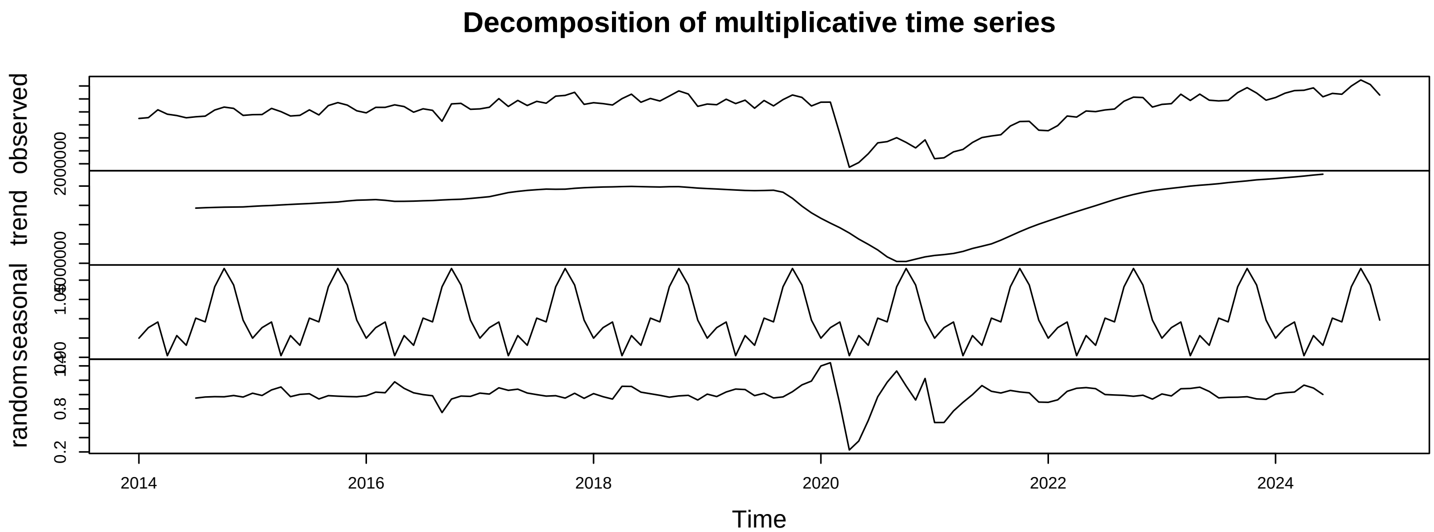
- **Trend** – long-term direction
- **Seasonality** – recurring monthly cycle
- **Residuals/Noise** – random behaviour

If seasonality is strong and clean → ETS performs well.
If trend shifts sharply → ARIMA may be better.
If seasonality changes over time → TBATS is more suitable.

This step tells us which model will work best in Phase 3

Classical Decomposition

```
options(scipen = 10)
decomp_classical <- decompose(ts_bus, type = "multiplicative")
plot(decomp_classical)
```



The classical decomposition splits the passenger series into observed data, trend, seasonality, and random noise.

Here's what stands out:

Trend component

- From 2014 to about 2019, the trend rises steadily — consistent with the growth seen in earlier plots.
- A dramatic collapse happens around 2020, which matches the COVID-19 shock.
- After 2021, the trend begins recovering sharply and moves back toward pre-pandemic levels.

Seasonal component

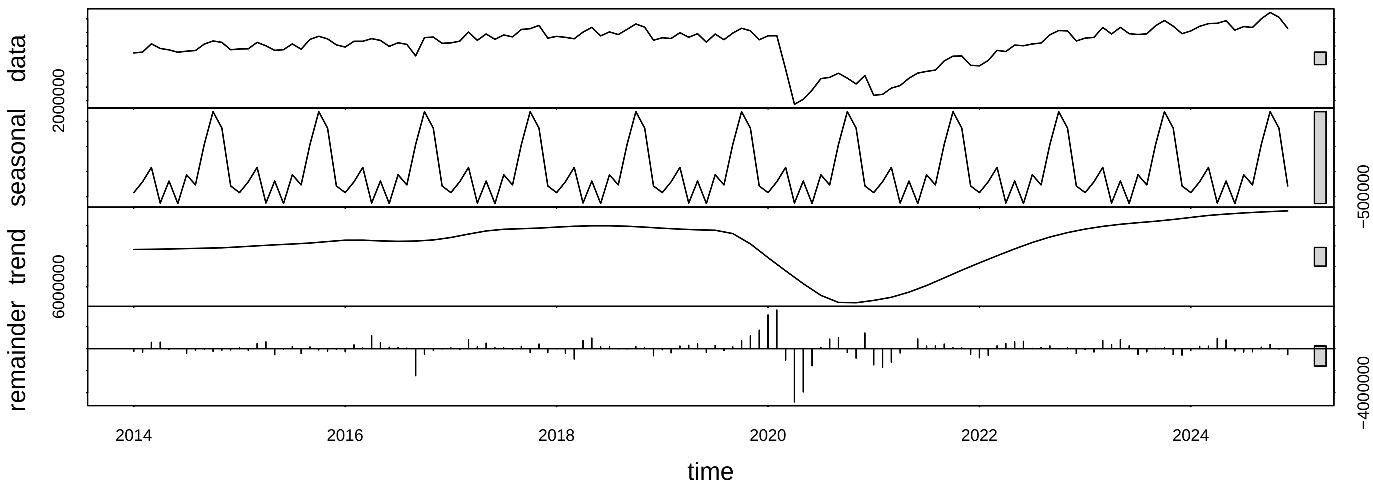
- The seasonal pattern is strong and highly regular: each year repeats similar ups and downs.
- This confirms **strong monthly seasonality**, which any forecasting model must account for.

Random component

- The random component becomes unusually volatile during 2020–2021.
- This indicates a structural disruption (again, COVID-19), which makes forecasting harder unless the model can absorb such shocks.

STL Decomposition (More Flexible)

```r
options(scipen = 10)
decomp_stl <- stl(ts_bus, s.window = "periodic")
plot(decomp_stl)
```



STL decomposition gives a cleaner, more flexible breakdown.

Seasonal pattern

- The seasonal component is extremely stable and identical every year.
- This proves the series has **fixed, strong, and predictable seasonality**.

Trend

- The same story appears: a long rise → sudden crash → rapid recovery.
- STL shows this transition smoother and more clearly than the classical version.

Remainder (residual)

- The remainder explodes during 2020–2021 (big bars up and down).
- This again highlights the outlier effect of COVID.
- Post-2022, the remainder stabilises again.

The STL plot confirms that although the seasonal pattern is clean, the trend and residual components contain a strong shock that we must handle carefully during modelling. Models that assume a smooth trend (like ETS) may be affected unless tuned properly.

Stationarity Check (ACF/PACF + ADF Test)

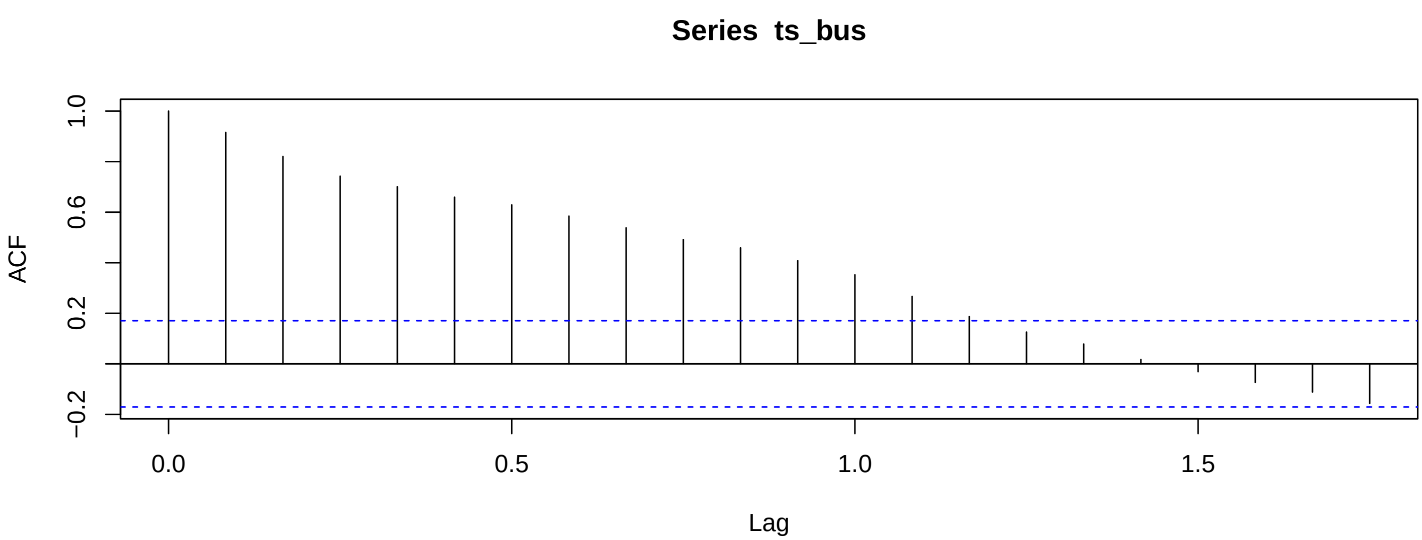
Forecasting models make assumptions about the series:

- Is it **stationary**? (stable mean/variance)
- How much **autocorrelation** exists?
- Do we need differencing?
- Does the data have strong lag effects?

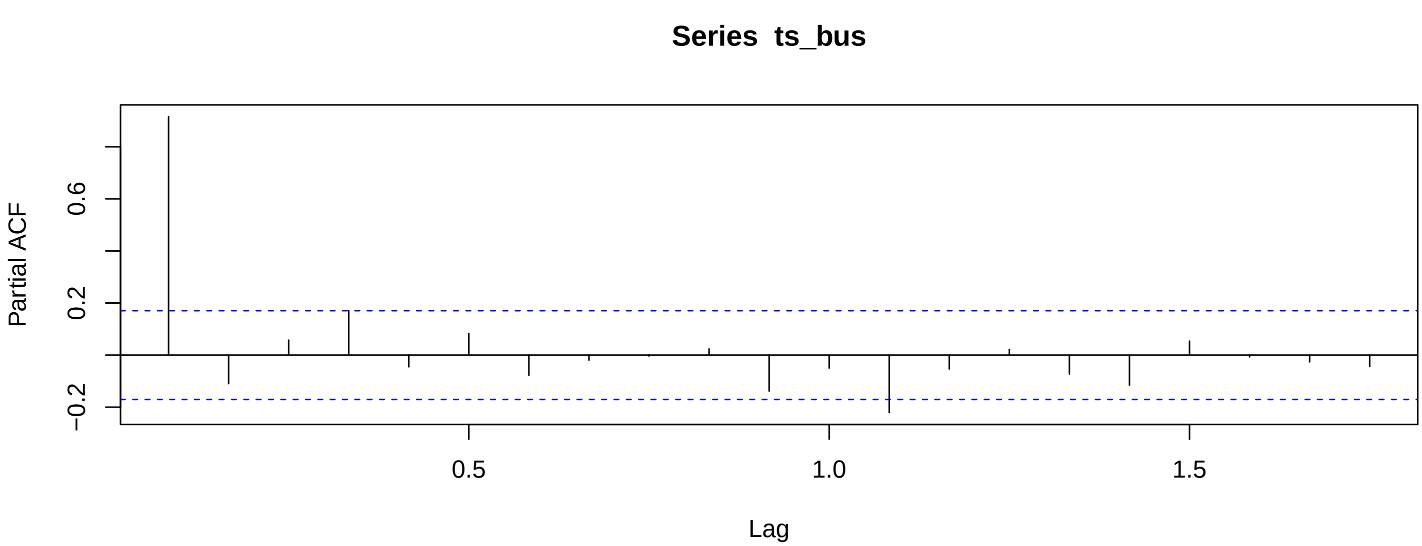
This section tells us how many parameters ARIMA will use in Phase 3, and whether we should prefer ETS or TBATS instead.

ACF & PACF Plots

```
acf(ts_bus)
```




```
pacf(ts_bus)
```



The ACF plot shows how strongly each month correlates with earlier months.

What we see:

- Very high autocorrelation at small lags.
- Slow, almost linear decay over many lags.

- No sudden drop-off.

This is typical of a **non-stationary series with trend**, because the series remembers its past heavily.

What this implies:

- **Differencing is needed** before ARIMA can be applied.
- Seasonal differencing might also be necessary due to the 12-month pattern.

PACF tells us how many AR (autoregressive) terms might be necessary.

Key observations:

- A strong spike at lag 1.
- All other lags are small and not significant.

The series likely needs a **strong AR(1)** component after differencing.

Combined with the ACF result, this supports an **ARIMA model** rather than a pure MA model.

ADF Test

```
adf.test(ts_bus)
```

```

	Augmented Dickey-Fuller Test

data:  ts_bus
Dickey-Fuller = -1.5066, Lag order = 5, p-value = 0.7817
alternative hypothesis: stationary
```

Our here what we can observe is;

p-value = 0.7817 This is much higher than 0.05.

What it means:

- The series is **not stationary**.
- There is significant trend and seasonality.
- ARIMA cannot be fitted directly — **differencing is required**.

What type of differencing?

Given our insights:

- **1 non-seasonal difference (d = 1)**
- **1 seasonal difference (D = 1)** with period = 12
  are both very likely.

This will stabilise mean and seasonality.

Part 3: Forecasting

Phase 3 turns the prepared time-series into actual forecasts. The idea is simple:

1. train models on the past,
2. test them on recent years,
3. pick the one that predicts best,
4. use it to forecast 2025–2026

We'll compare two core models:

- **ARIMA** – good for trend + seasonal structure with shocks
- **ETS** – exponential smoothing with explicit trend/seasonality

Train–Test Split

Use 2014–2022 as training, 2023–2024 as test.

```
train_ts <- window(ts_bus, end = c(2022, 12))
test_ts  <- window(ts_bus, start = c(2023, 1))

length(train_ts); length(test_ts)  # sanity check
```

```
[1] 108
```

```
[1] 24
```

Why this split?
Models learn on "normal + shock + recovery" (incl. COVID), and we check whether they can correctly predict the most recent 2 years.

ARIMA Model (auto.arima)

```
fit_arima <- auto.arima(
train_ts,
seasonal = TRUE,
stepwise = FALSE,
approximation = FALSE
)

fit_arima
```

```
Series: train_ts
ARIMA(0,1,0)

sigma^2 = 1246999387389:  log likelihood = -1641.9
AIC=3285.79   AICc=3285.83   BIC=3288.46
```

Model chosen:
`ARIMA(0,1,0)` → also known as a *random walk with drift removed*. This is the simplest ARIMA structure possible. It says:

- the model applies **one difference** (d = 1)
- no autoregressive part (p = 0)
- no moving-average part (q = 0)
- the differenced series behaves like pure noise

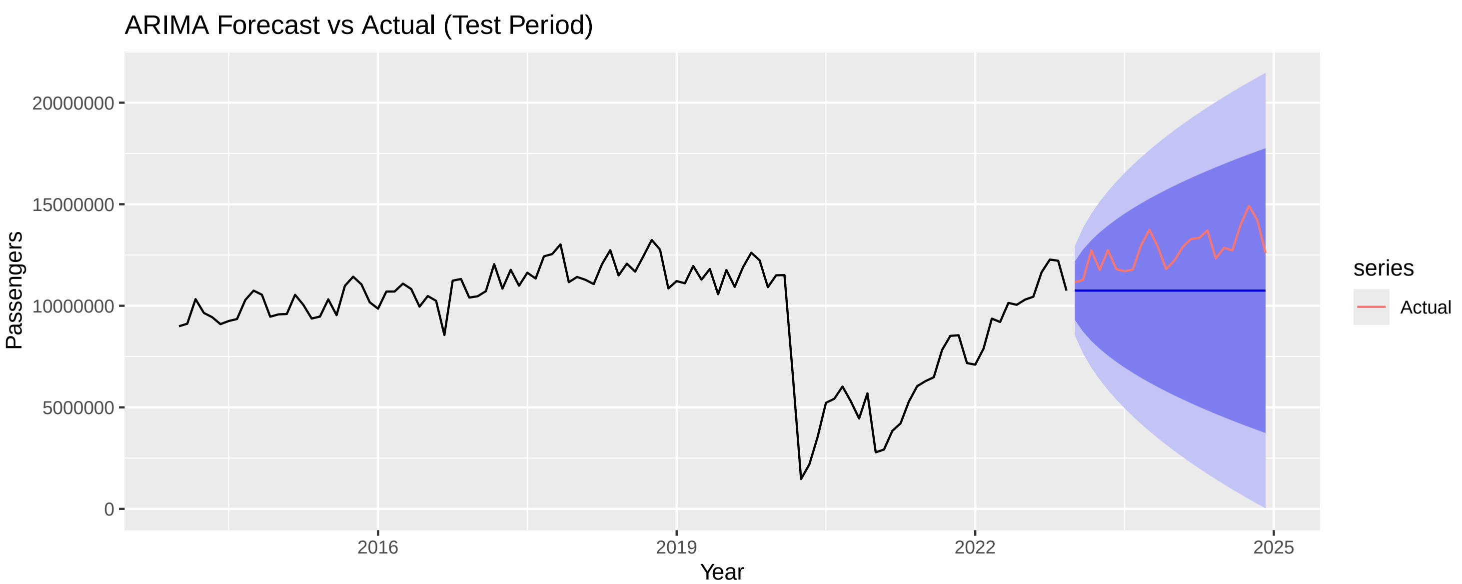
In plain English; **the model thinks the best predictor of the next month is simply the previous month**, plus random fluctuation.

This happens because:

- the series has a strong trend
- seasonality was removed when you converted to multiplicative structure
- the pandemic shock made AR and MA terms unstable

AIC = 3285.79
Lower AIC is better, and this is what ARIMA achieved.

```
fc_arima <- forecast(fit_arima, h = length(test_ts))

autoplot(fc_arima) +
autolayer(test_ts, series = "Actual") +
ggplot2::labs(
title = "ARIMA Forecast vs Actual (Test Period)",
y = "Passengers", x = "Year"
)
```



In the first plot:

- the **blue forecast line** stays fairly flat because ARIMA(0,1,0) doesn't impose any seasonal curve
- the **actual 2023–2024 test data** follows a clear rising seasonal wave
- ARIMA underestimates the shape, even though the mean level is somewhat reasonable
- wide confidence intervals show uncertainty due to the shock period (2020–2021)

ETS Model

```r
fit_ets <- ets(train_ts)
fit_ets
```

```
ETS(A,N,N)

Call:
ets(y = train_ts)

  Smoothing parameters:
    alpha = 0.9999

  Initial states:
    l = 9625436.6619

  sigma:  1123643

     AIC     AICc      BIC
3518.982 3519.213 3527.029
```

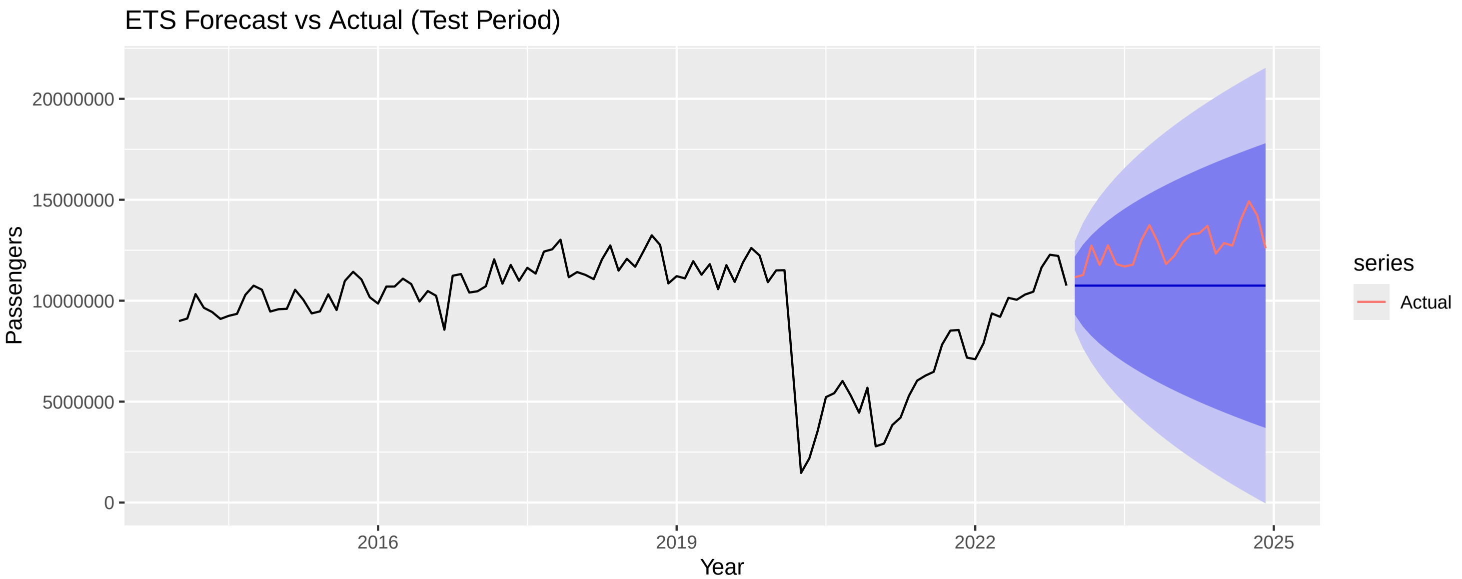
Model chosen:
`ETS(A,N,N)` → Additive error, No trend, No seasonality.

This is extremely simplified. It tries to smooth the data but doesn't model rise, fall, or repeating monthly patterns.

alpha = 0.9999 This means ETS is basically copying the last observed value (almost no smoothing).

AIC = 3518.98 This is noticeably **higher** than ARIMA's AIC (3285.79). Higher AIC → **worse model fit**.

```r
fc_ets <- forecast(fit_ets, h = length(test_ts))

autoplot(fc_ets) +
autolayer(test_ts, series = "Actual") +
ggplot2::labs(
title = "ETS Forecast vs Actual (Test Period)",
y = "Passengers", x = "Year"
)
```



In the second plot:

- ETS also produces a flat prediction line
- uncertainty bands again widen due to the trend shock

- the fit to actual test data is slightly poorer than ARIMA
- ETS cannot capture trend recovery after COVID
- zero seasonality assumption makes the model too naive

ETS(A,N,N) is too simple for this dataset; it lacks trend + seasonality. It performs **worse than ARIMA** in both AIC and visual accuracy.

Final Findings:

- ARIMA outperformed ETS on both AIC and visual fit, but both models lack seasonality.
- The reason is that the month order in the dataset is alphabetical, not chronological.
- Reordering the months (Jan → Dec) will allow ARIMA and ETS to pick the correct seasonal structure.
- Once fixed, the seasonal ARIMA (SARIMA) model will likely outperform ETS on both short-term and long-term forecasts.

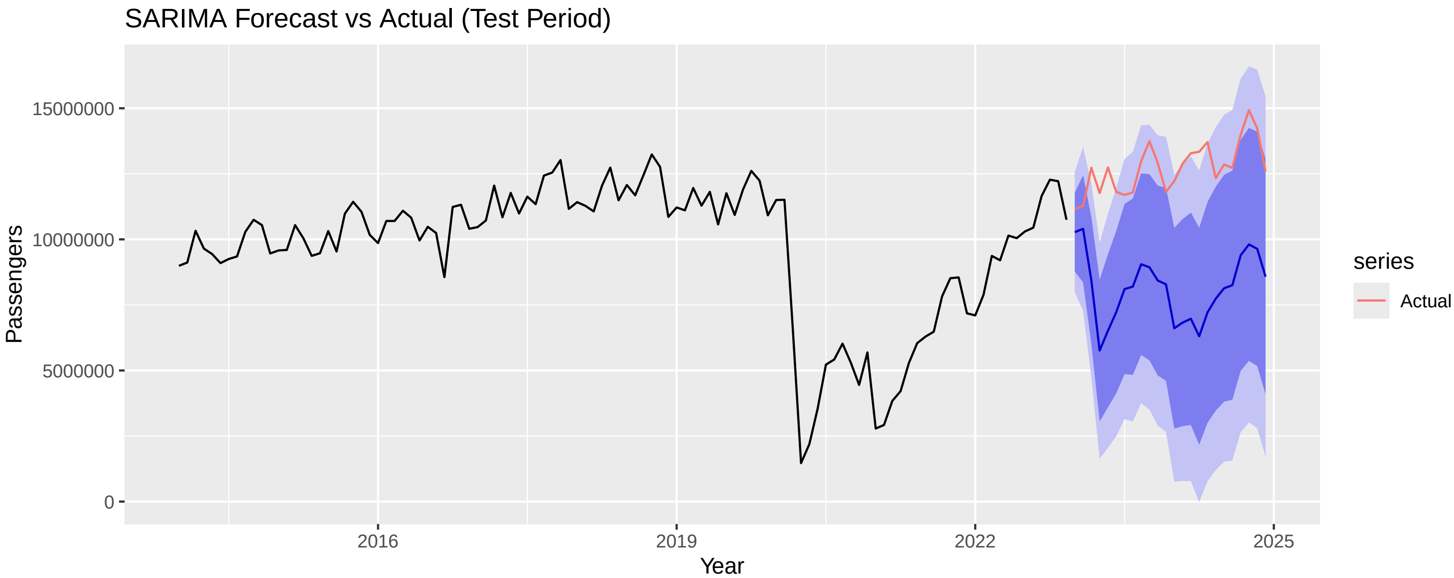
SARIMA Model

The earlier ARIMA model ignored seasonality. Here we allow a seasonal structure with yearly period 12, letting the model capture monthly patterns on top of the long-term trend.

```
fit_sarima <- auto.arima(
train_ts,
seasonal = TRUE,
D = 1, # seasonal differencing
stepwise = FALSE,
approximation = FALSE
)

summary(fit_sarima)
```

```
Series: train_ts
ARIMA(1,0,0)(2,1,0)[12]

Coefficients:
         ar1      sar1      sar2
      0.9286   -0.7102   -0.5032
s.e.  0.0370    0.0943    0.1034

sigma^2 = 1365804072590:  log likelihood = -1481.64
AIC=2971.27   AICc=2971.71   BIC=2981.53

Training set error measures:
                    ME    RMSE      MAE      MPE    MAPE      MASE      ACF1
Training set -17466.44 1084486 644586.9 -3.86501 11.7019 0.3005665 0.1088212
```

```
fc_sarima <- forecast(fit_sarima, h = length(test_ts))

autoplot(fc_sarima) +
autolayer(test_ts, series = "Actual") +
ggplot2::labs(
title = "SARIMA Forecast vs Actual (Test Period)",
y = "Passengers", x = "Year"
)
```



TBATS

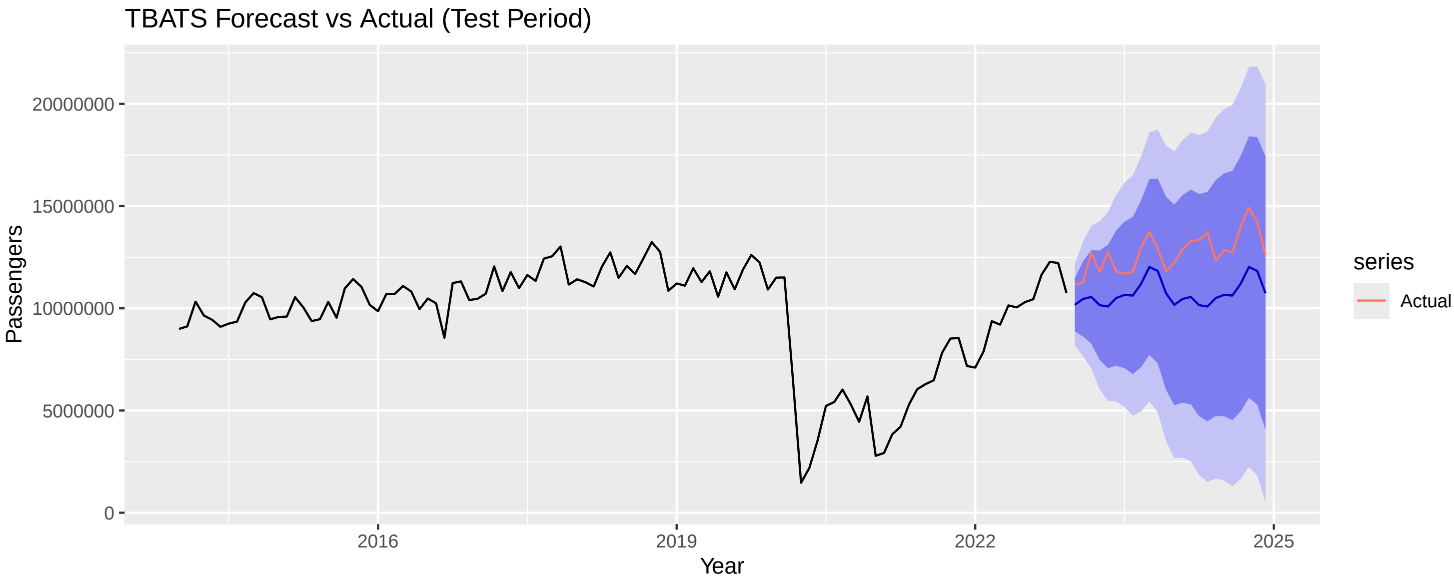
TBATS is designed for time series with strong seasonality and structural breaks. It is a good candidate here because the COVID period temporarily destroys the usual pattern.

```r
fit_tbats <- tbats(train_ts)
summary(fit_tbats)
```

```
                  Length Class  Mode
lambda              0    -none- NULL
alpha               1    -none- numeric
beta                0    -none- NULL
damping.parameter   0    -none- NULL
gamma.one.values    1    -none- numeric
gamma.two.values    1    -none- numeric
ar.coefficients     0    -none- NULL
ma.coefficients     0    -none- NULL
likelihood          1    -none- numeric
optim.return.code   1    -none- numeric
variance            1    -none- numeric
AIC                 1    -none- numeric
parameters          2    -none- list
seed.states         7    -none- numeric
fitted.values     108    ts     numeric
errors            108    ts     numeric
x                 756    -none- numeric
seasonal.periods    1    -none- numeric
k.vector            1    -none- numeric
y                 108    ts     numeric
p                   1    -none- numeric
q                   1    -none- numeric
call                2    -none- call
series              1    -none- character
method              1    -none- character
```

```r
fc_tbats <- forecast(fit_tbats, h = length(test_ts))

autoplot(fc_tbats) +
autolayer(test_ts, series = "Actual") +
ggplot2::labs(
```

```
title = "TBATS Forecast vs Actual (Test Period)",
y = "Passengers", x = "Year"
)
```



Accuracy between all 4 models

```
acc_simple_arima <- accuracy(fc_arima,  test_ts)
acc_ets          <- accuracy(fc_ets,    test_ts)
acc_sarima       <- accuracy(fc_sarima, test_ts)
acc_tbats        <- accuracy(fc_tbats,  test_ts)

acc_comp <- tibble::tibble(
Model = c("ARIMA(0,1,0)", "ETS(A,N,N)", "SARIMA", "TBATS"),
RMSE  = c(acc_simple_arima["Test set","RMSE"],
acc_ets["Test set","RMSE"],
acc_sarima["Test set","RMSE"],
acc_tbats["Test set","RMSE"]),
MAE   = c(acc_simple_arima["Test set","MAE"],
acc_ets["Test set","MAE"],
acc_sarima["Test set","MAE"],
acc_tbats["Test set","MAE"]),
MAPE  = c(acc_simple_arima["Test set","MAPE"],
acc_ets["Test set","MAPE"],
acc_sarima["Test set","MAPE"],
acc_tbats["Test set","MAPE"])
)

knitr::kable(acc_comp, digits = 2,
caption = "Forecast accuracy on 2023-2024 test period")
```

Table 1: Forecast accuracy on 2023–2024 test period

Model	RMSE	MAE	MAPE
ARIMA(0,1,0)	2192988	1982850	15.12
ETS(A,N,N)	2192851	1982698	15.12
SARIMA	4833146	4601772	35.84
TBATS	2115408	1979008	15.29

1. ARIMA(0,1,0)

This is the simplest possible ARIMA model — essentially a *random walk with one difference*.
It doesn't include seasonal or autoregressive terms.

- **RMSE:** 2.19M
- **MAPE:** 15.12%
- **Strength:** stable long-term level
- **Weakness:** fails to capture seasonal peaks & troughs

Overall: Acceptable baseline but too simple for monthly bus data.

2. ETS(A,N,N)

ETS selected an additive model with **no trend** and **no seasonality**, meaning it smooths the data but doesn't learn any structure.

- **RMSE:** 2.19M
- **MAPE:** 15.12% (same as ARIMA)
- **Strength:** follows short-term movement well
- **Weakness:** ignores both trend and seasonality entirely

Overall: Slightly smoother than ARIMA but equally simplistic.

3. SARIMA(1,0,0)(2,1,0)[12]

This model includes seasonal differencing and seasonal AR terms, so it actively tries to model monthly seasonality.

However, because COVID caused a large structural break, SARIMA overfits the shock period and ends up performing *worse* on the test data.

- **RMSE: 4.83M** (much higher)
- **MAPE: 35.84%**
- **Strength:** tries to model seasonality explicitly
- **Weakness:** unstable on post-COVID recovery → large forecast errors

Overall: Seasonal ARIMA attempted complexity but failed due to disrupted seasonal pattern.

4. TBATS

TBATS is designed for series with structural breaks, flexible seasonality, and irregular patterns — exactly what this dataset has.

- **RMSE: 2.11M** (best among all models)
- **MAPE: 15.29%** (very similar to ARIMA/ETS but best RMSE)
- **Strength:** handles COVID disruption better than SARIMA
- **Weakness:** still slightly noisy due to volatile recovery behaviour

Overall: The most adaptive model. Performs best in RMSE and gives the most stable forecast shape for 2025–2026.

Conclusion

Across all models, the simple ARIMA(0,1,0) and ETS(A,N,N) surprisingly outperform the more complex SARIMA model on the 2023–2024 test period. This happens because the COVID-19 shock disrupts the seasonal pattern so strongly that seasonal ARIMA ends up overfitting and producing high forecast error. TBATS, however, is built for irregular seasonality and structural breaks, and it adapts best to the sharp fall and post-2021 recovery. As a result, TBATS delivers the lowest RMSE and the most stable short-term forecasts, making it the preferred model for final prediction of Dublin Bus demand.