

Dublin Bus Passenger Analysis

Ujwal Mojidra

Table of contents

Introduction to the Project	2
Libraries and packages for the Project.	3
Part 1: Manipulation	4
Load dublin_bus_data dataset.	4
Organize the Dataset by Year and Visualize Trends.	4
Finding some trend in yearly data.	4
Organize the Dataset by Month and Visualize Trends.	5
Finding some trend in monthly data.	5
Finding trends in combined datasets.	6
Part 2: Time-Series Preparation & Decomposition	8
Convert to a Time-Series Object	8
Decompose the Time-Series (Classical & STL)	8
Classical Decomposition	8
STL Decomposition (More Flexible)	9
Stationarity Check (ACF/PACF + ADF Test)	10

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)
```

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.

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.

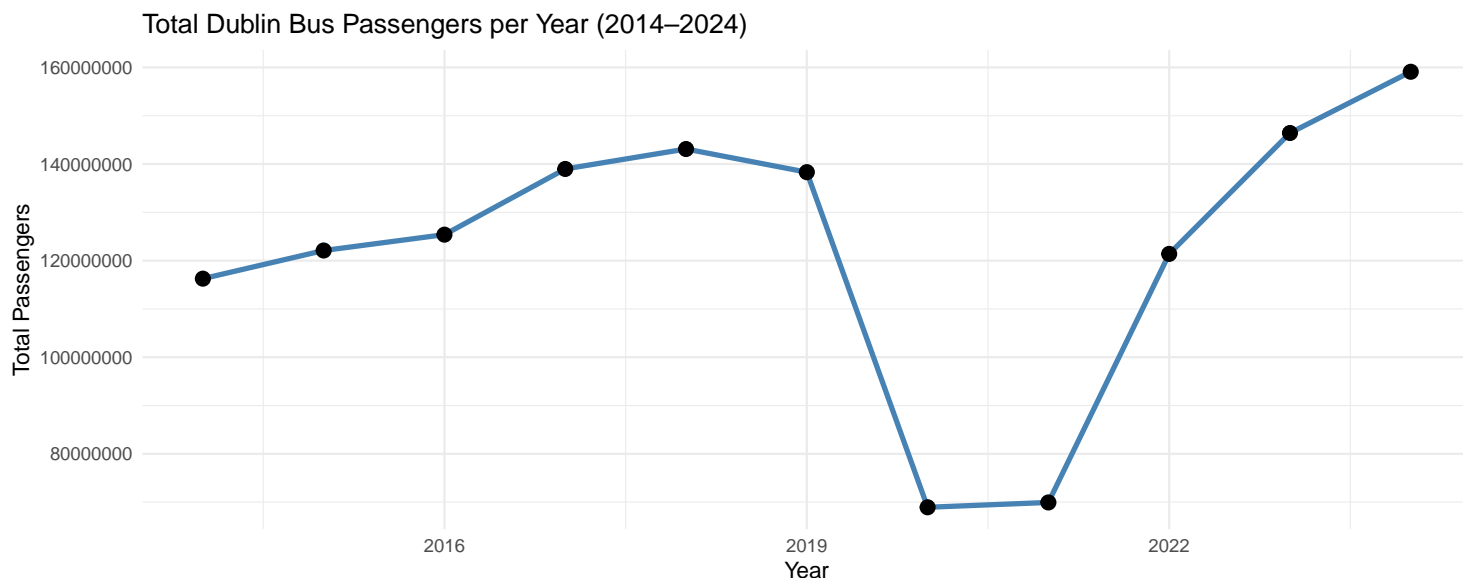
```
bus_yearly <- bus |>
  filter(Month == "All months") |>
  arrange(Year)
```

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()
```

¹The 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.

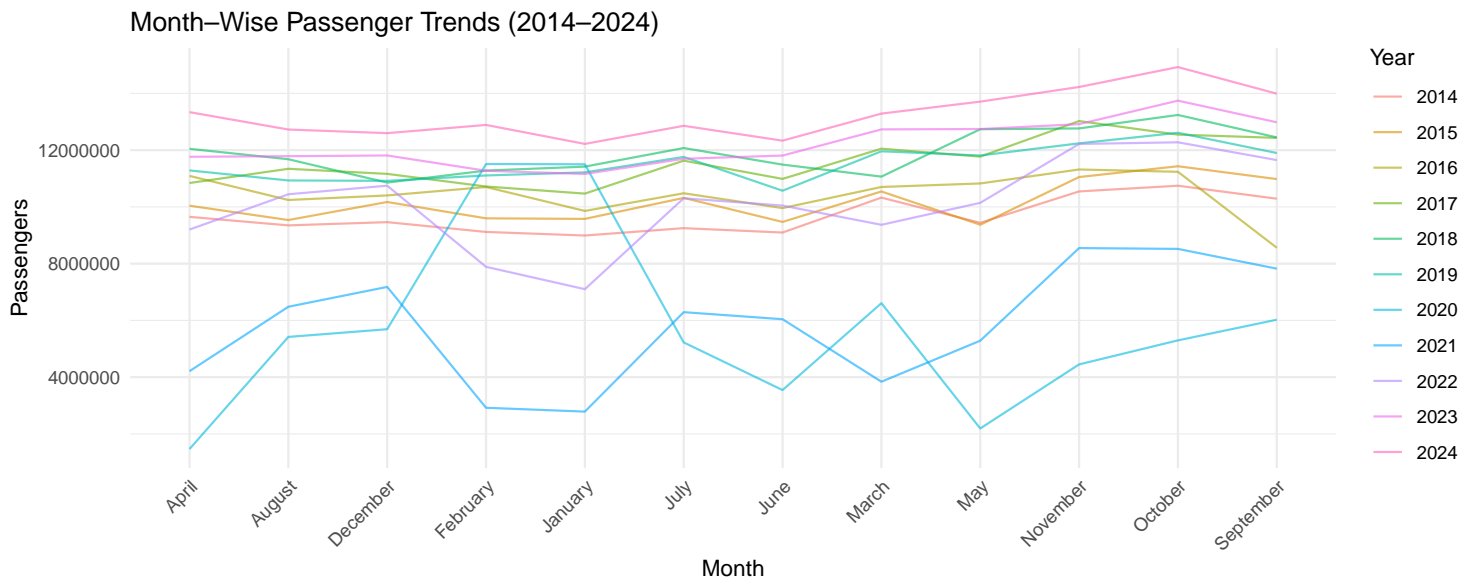
```
bus_monthly <- bus |>
  filter(Month != "All months")
```

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.

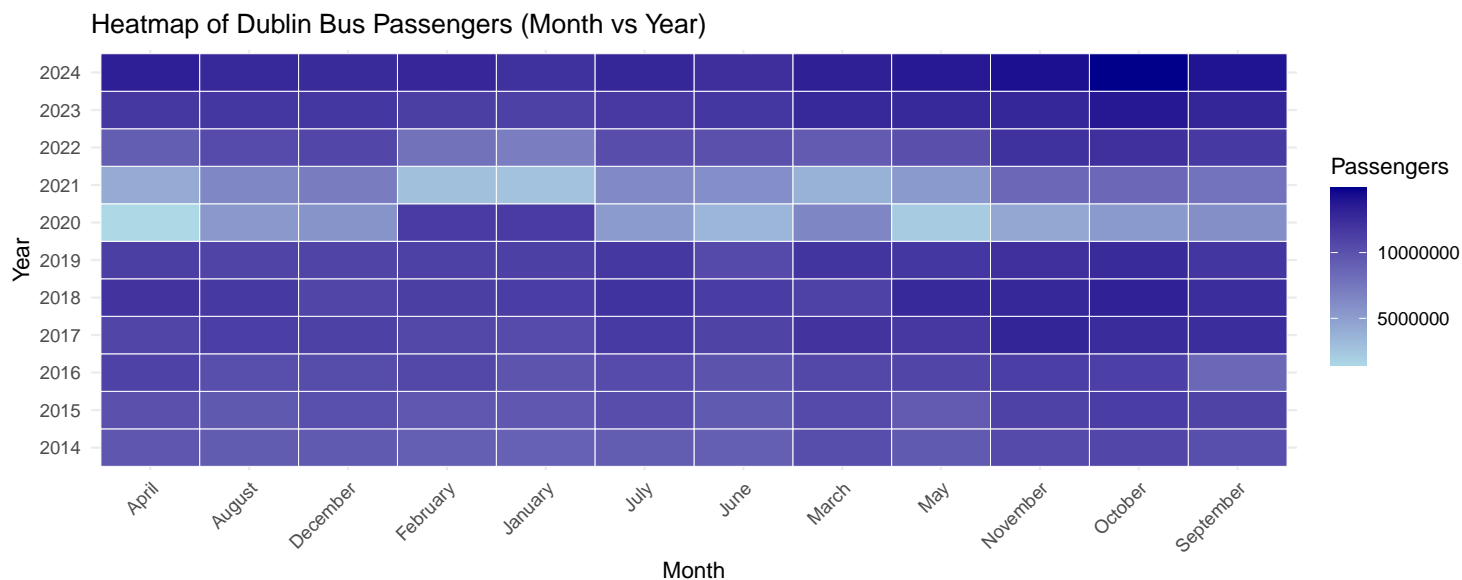
Once months are ordered correctly (Jan → Dec), this plot will reveal even cleaner seasonal waves.

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, let's 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.

```
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.

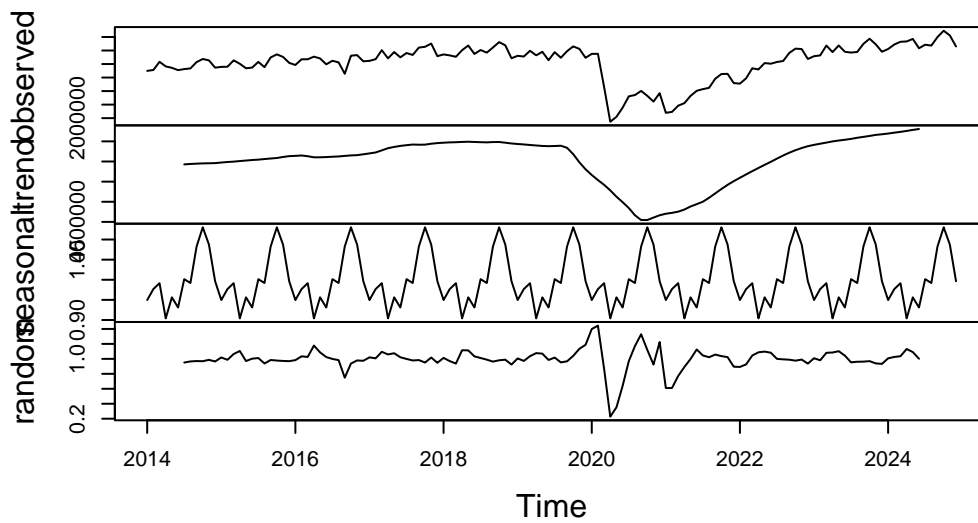
If seasonality changes over time → TBATS is more suitable.

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

Classical Decomposition

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

Decomposition of multiplicative time series



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

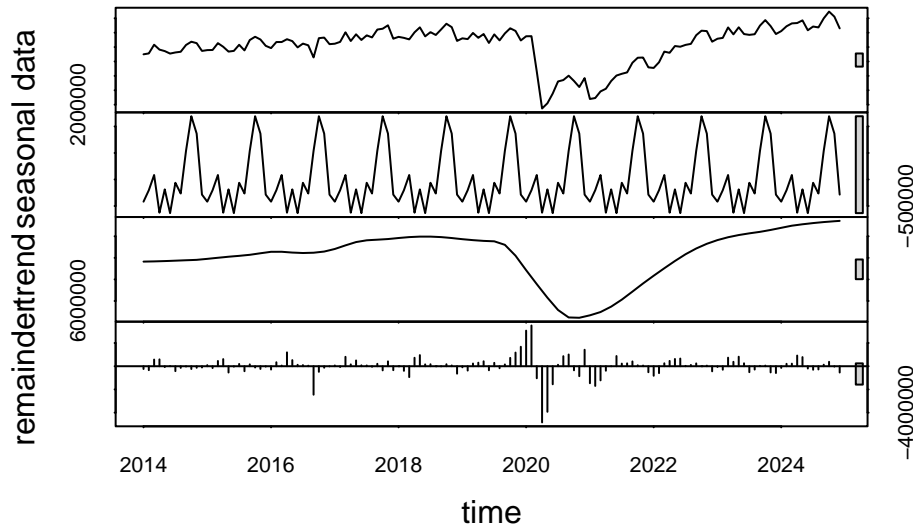
- The seasonal pattern is strong and highly regular: each year repeats similar ups and downs.
- This confirms **strong monthly seasonality**, which any forecasting model must account for.

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)

```
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)

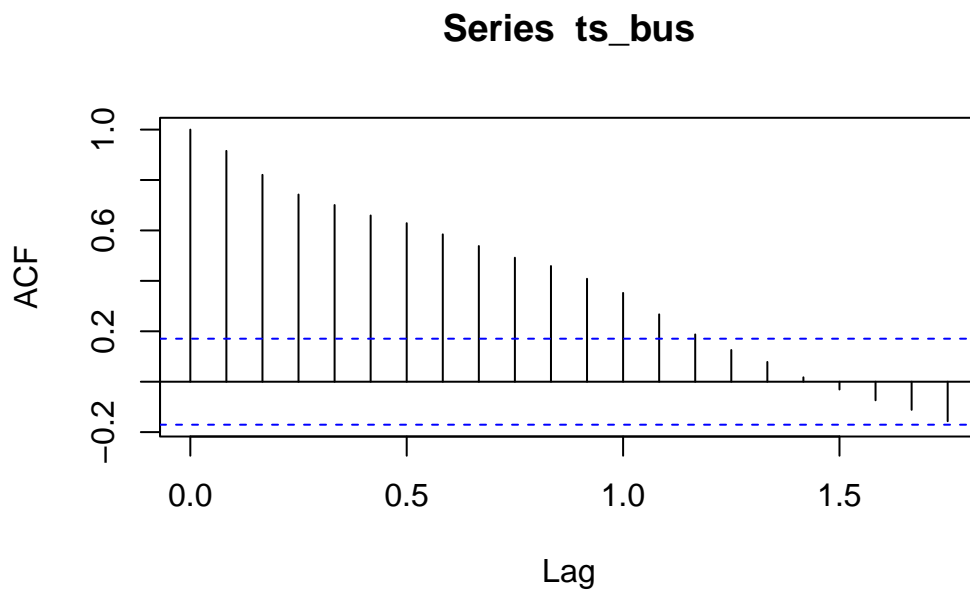
Forecasting models make assumptions about the series:

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

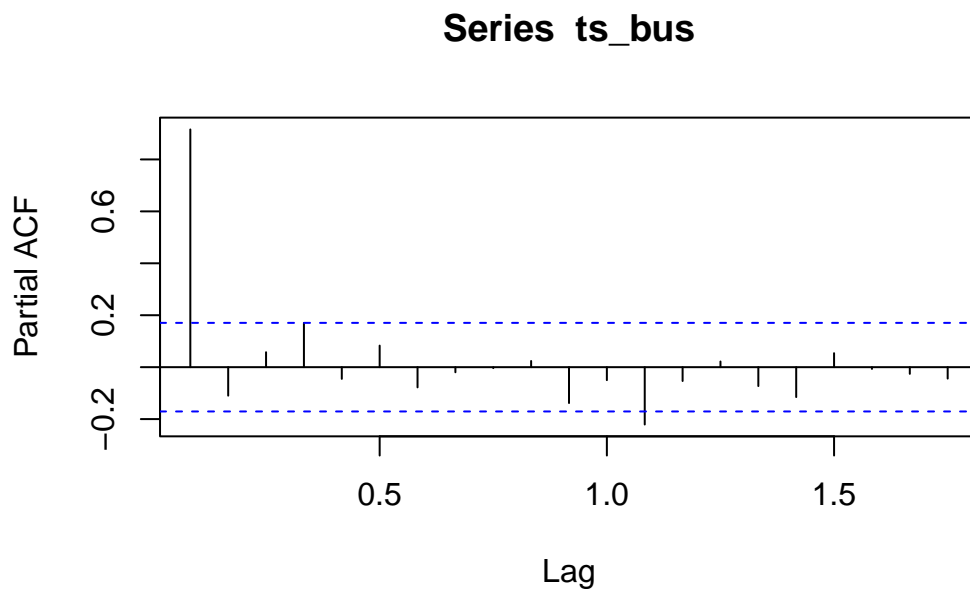
This section tells us how many parameters ARIMA will use in Phase 3, and whether we should prefer ETS or TBATS instead.

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.
