Daily Sourdough Sales Forecast for a Grocery-Store Bakery

Carter Corzine

2025-06-04

Abstract

Accurate demand forecasts are essential for avoiding both stock-outs and waste in a retail bakery. I analyzed 764 consecutive days of sourdough-loaf sales dated from 22 April 2023 to 24 May 2025, drawn from the grocery-store bread section where I work. After applying a log of one plus sales transformation, a seven-day seasonal difference, and one regular difference, the series looked stationary. Two hand-specified models were compared, and a SARIMA (1,1,1)(0,1,1)[7] gave the lower AIC and a hold-out root-mean-square error of 5.7 loaves while passing Ljung–Box residual tests. Re-fitted on the full data, this model forecasts the next twenty-eight days of demand at about thirty five loaves per day with a ninety-five percent interval of roughly eight loaves above or below that level.

Introduction

I am one of the order writters for the bread section of the grocery store that I work for (for company data privacy I can't name it), and one of my jobs is making the daily order for all of our loaves. If I order too few, we run out and miss sales. We don't use any preservatives in our bread so the shelf life is very short. This means if I order too many we waste product. So having a solid forecast of how many loaves we'll sell each day really matters.

For this project I pulled **daily sourdough sales** from our point of sale system, covering **22 Apr 2023 – 24 May 2025** (764 days). Each record has the date, the number of loaves sold, and the day of the week. Sales average around 37 loaves a day, but they swing from about 19 up to 70. Sundays and Mondays are usually the busiest, so I expect a clear day of week pattern on top of any longer term trend.

My goal is to build a model that can **predict the next four weeks of daily demand** and tell me how much uncertainty is in those predictions. Here's the game plan:

Split the data into a training set and a hold-out test set.

Plot the series to spot trends, weekly seasonality, and any changes in variance.

Transform/difference as needed to make the series roughly stationary.

Use ACF/PACF plots to pick a few promising SARIMA orders.

Fit at least two models, check residuals, and compare them with AICc and test set RMSE.

Generate forecasts for the test period and four weeks ahead, then back-transform to the original scale.

Data Description

Getting this historical data required a workaround. Our ordering system only shows one week's sales at a time and doesn't let me export the figures directly, so I had to photograph each week's table for the past two years. Manually typing all those numbers would have taken forever, so I attempted to write OCR script to process the images, pull the values into a data frame, and save everything to a CSV. OCR missed a few

entries and was not very accurate with the dates, usually when a screenshot was a bit blurry or the text was small, so I went back through those photos by hand to fill in any missing sales figures. Going back through the data maualy to insure accuracy was very time consuming and I chose to omit some of the information that could potentially have been helpful if I was making a more complicated model. For example, our system records the last time the item was sold, we close at 9 pm and I pulled our most popular sku. Therefor it would be reasonable to assume that if the last time we sold a loaf was 2pm this would mean we ran out for the day. Assumptions such as this would make a much more complicated model so I have chosen to only include the following information:

`date` ("MM/DD/YYYY" format),

`sales` (number of loaves sold at point of sale),

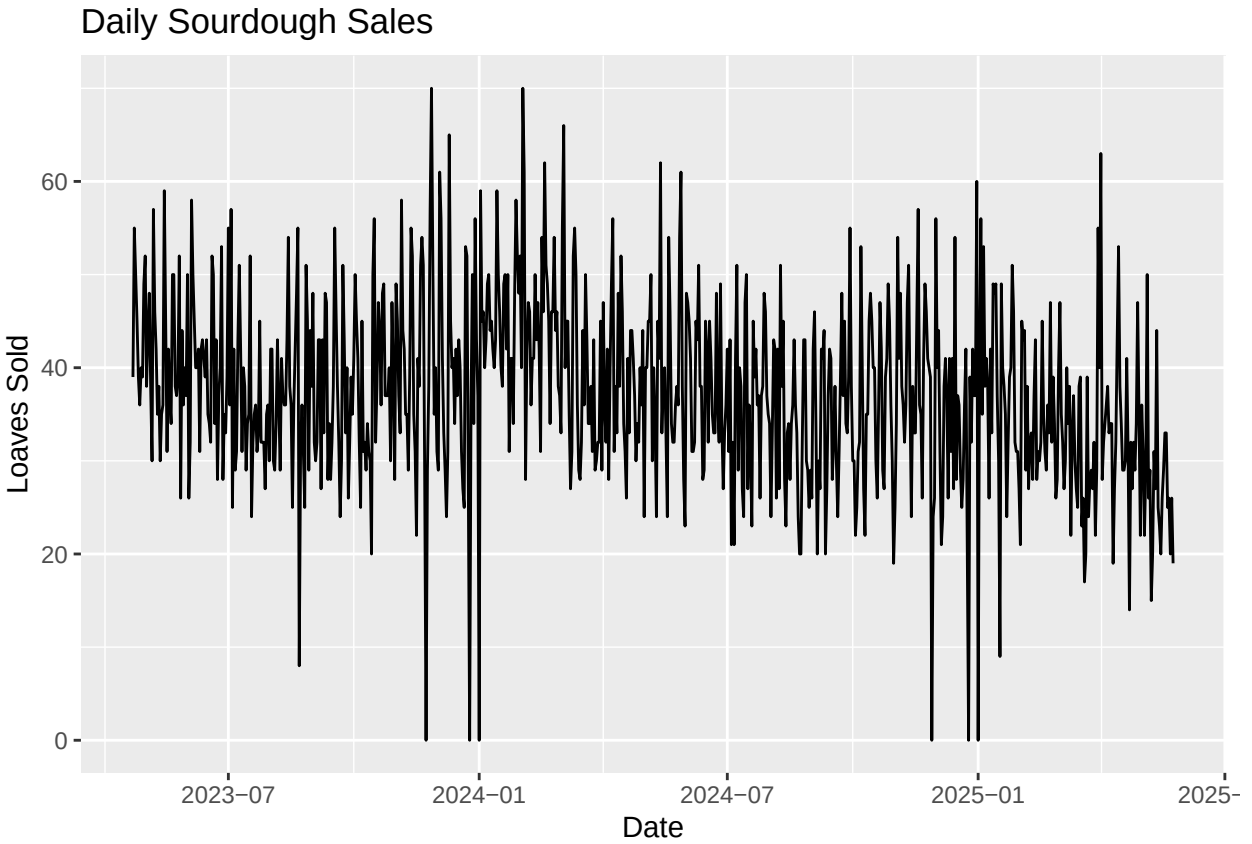
`weekday` (day of the week)

Modelling & Forecasting Sections

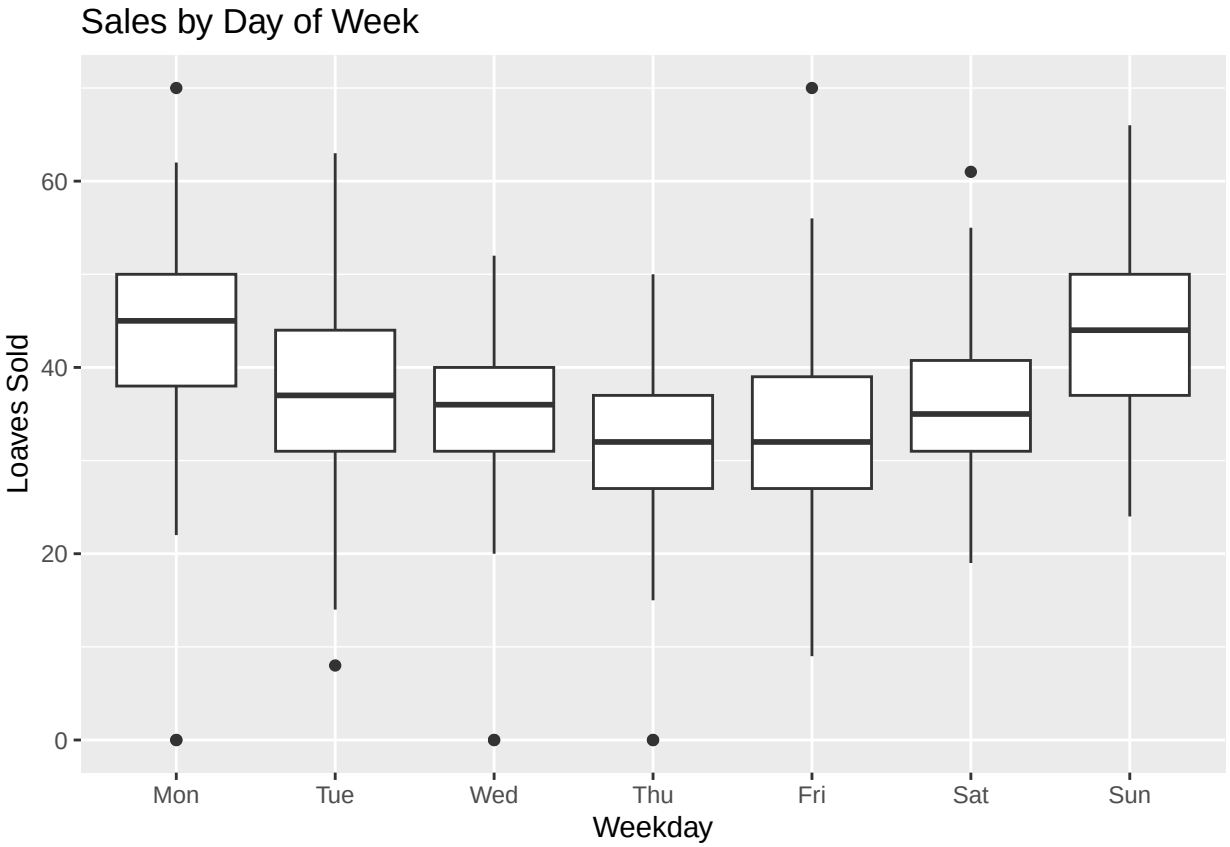
We collected 764 consecutive days of sourdough sales and held back the last 84 days as a test set. Everything before that point forms the training data that the model will actually learn from.

```
## Train rows: 680   Test rows: 84
```

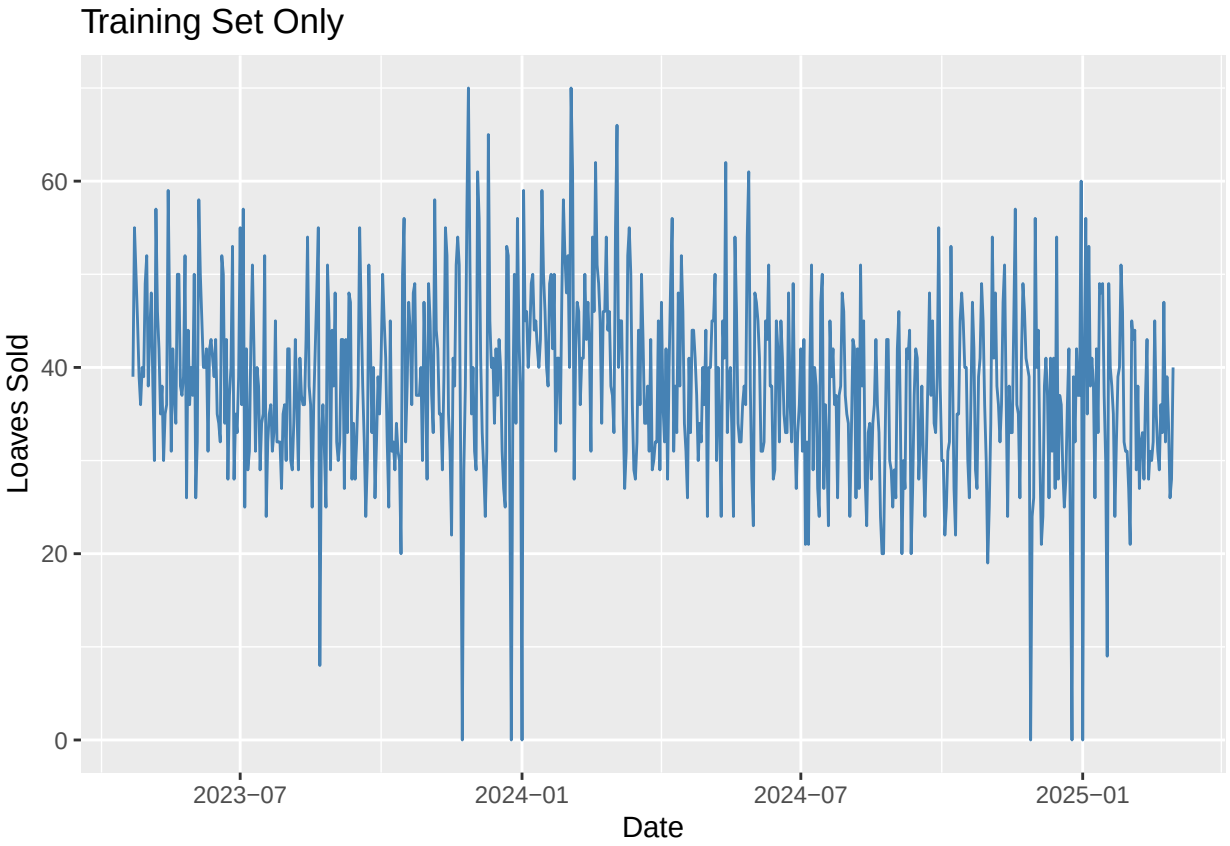
The full series in this graph shows steady demand near 35–40 loaves a day, with the occasional out of stock dip and a gentle upward drift after mid-2024. No obvious structural breaks appear. The strong spikes down to zero could also be attributed to days we are closed such as Christmas.



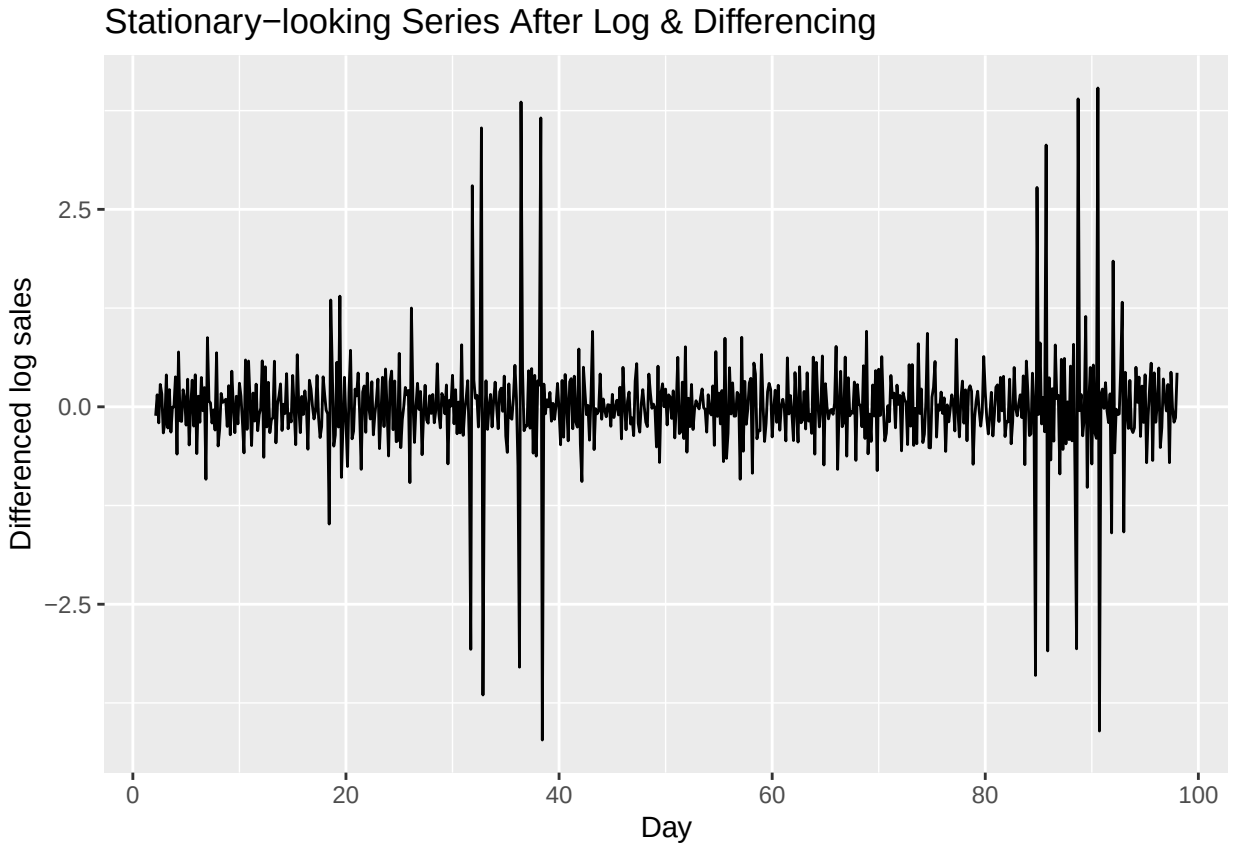
Sales spike on Sundays and Mondays and sag mid-week, confirming a strong weekly pattern that will need to be captured by the model, through seasonal differencing.



Zooming in on the training window lets me focus on what the model sees. The trend and variability look stable enough that a SARIMA framework should work once I difference for the weekly seasonality.



I first took log (1 + sales) to smooth out the larger swings on busy days, then differenced by 7 to wipe out the weekly cycle, and finally took one ordinary difference to remove the gentle upward drift.

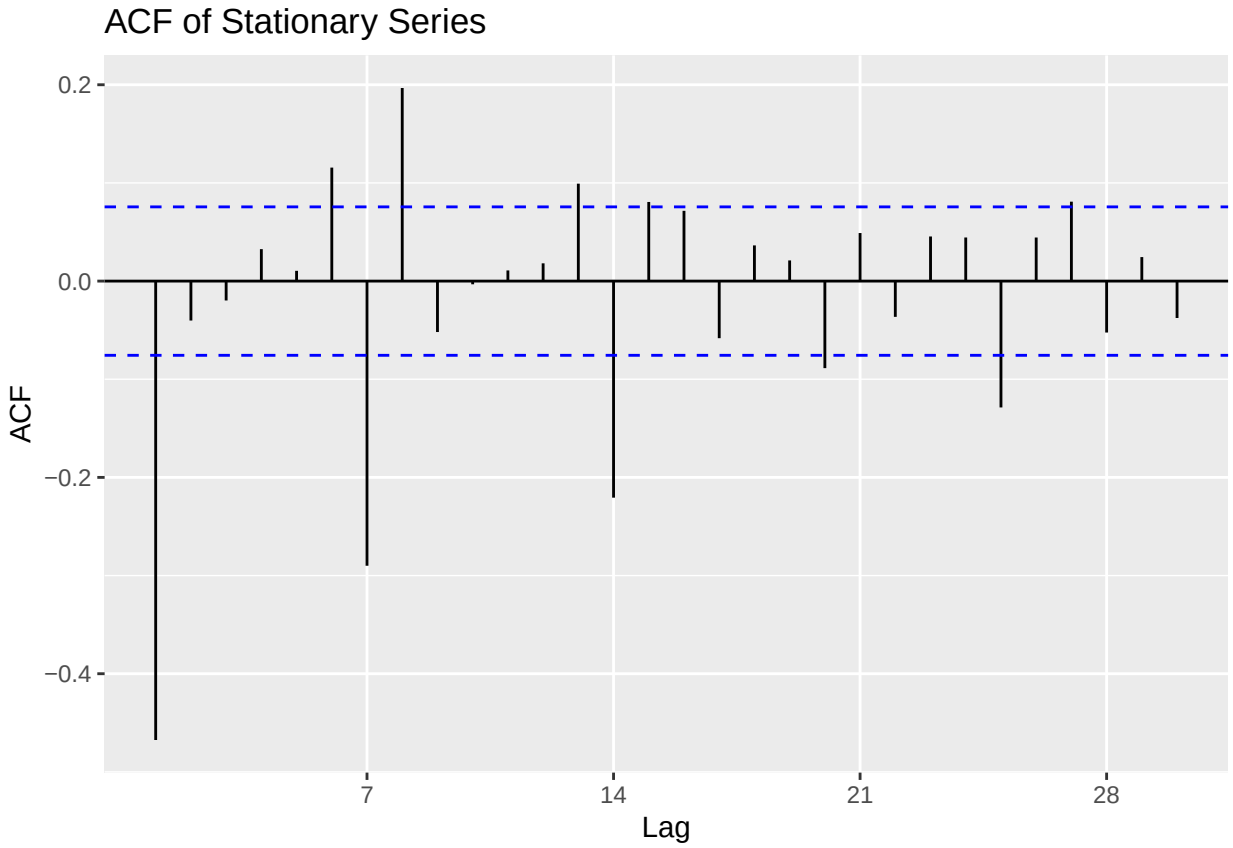


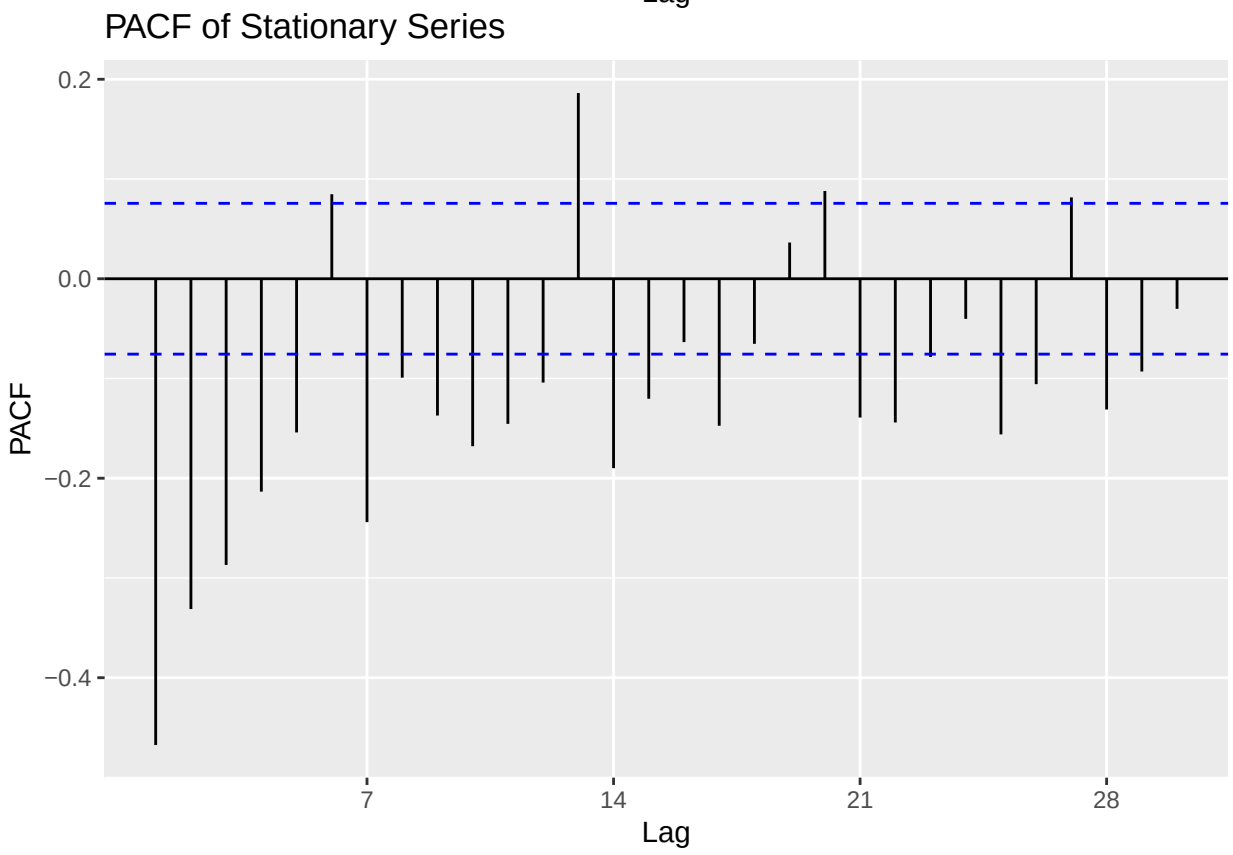
This graph shows the adjusted series bouncing evenly around zero with a steady spread, exactly the pattern we look for before picking a model.

```
## First half  - mean: 5e-04  sd: 0.6585
```

```
## Second half - mean: 1e-04  sd: 0.684
```

To keep things simple, I split the series in half and compared the averages and standard deviations. They're almost identical, so the mean and variance look stable, another sign the series is now stationary.

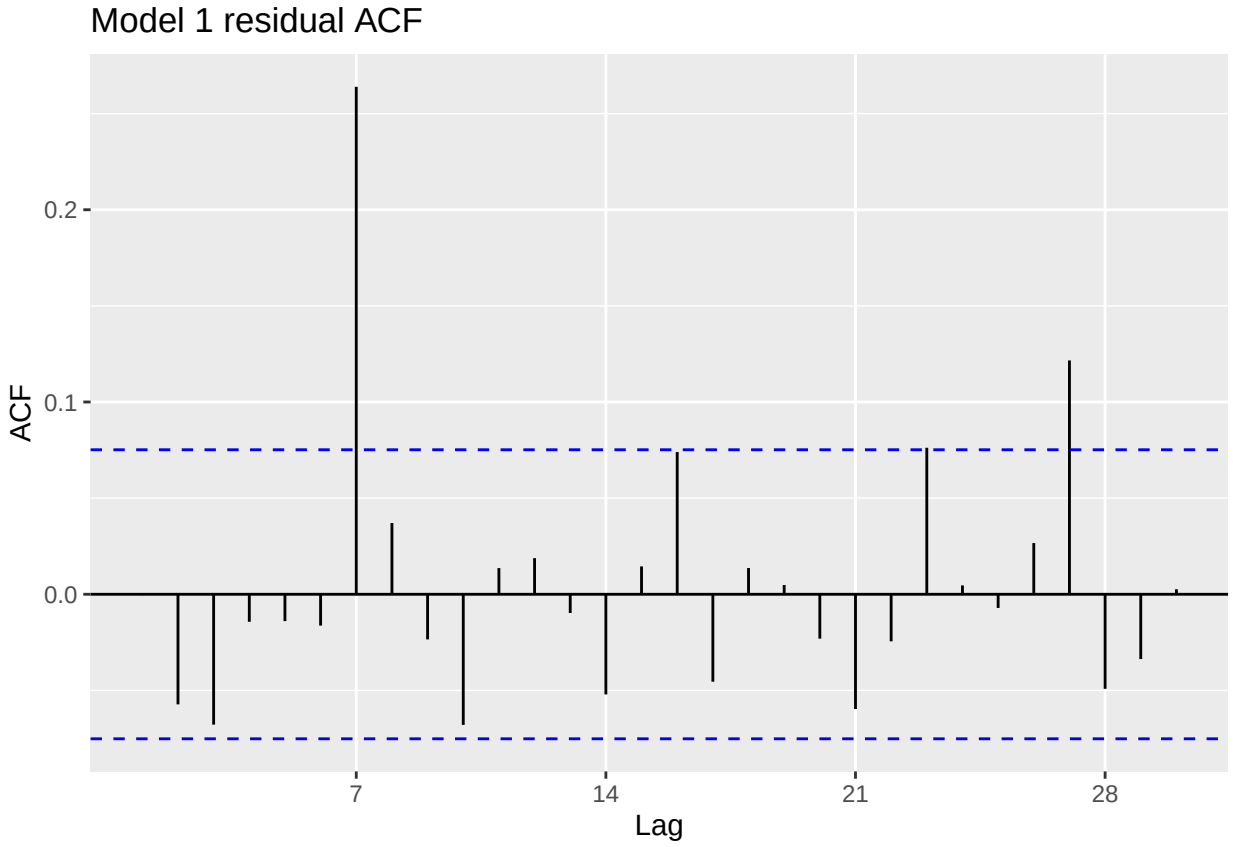


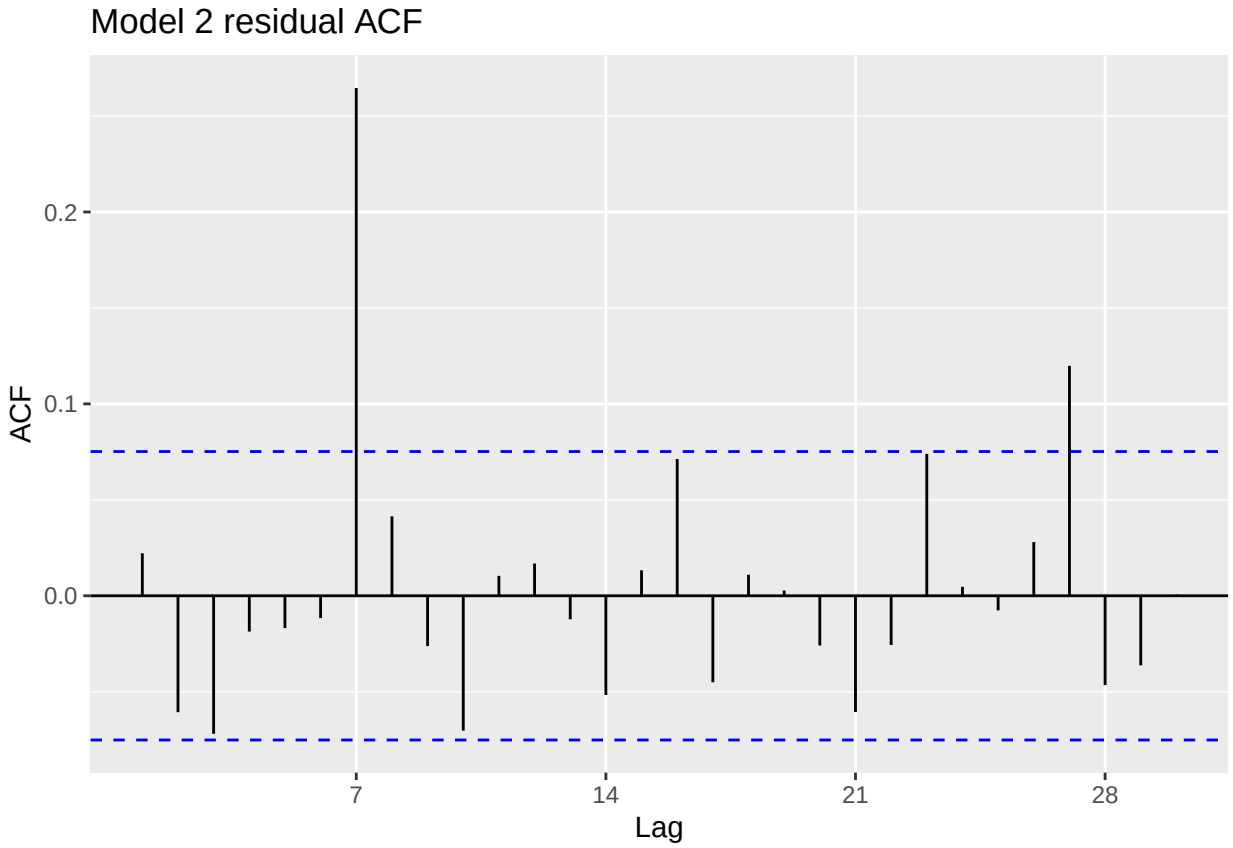


The ACF dies out after lag 7 but has a sharp spike right at lag 7, while the PACF cuts off quickly at the

same point. That pattern suggests trying a SARIMA with a seasonal MA(1) term (Q = 1) and a small non-seasonal AR(1) or MA(1) component. I'll fit those candidate models next and compare their diagnostics.

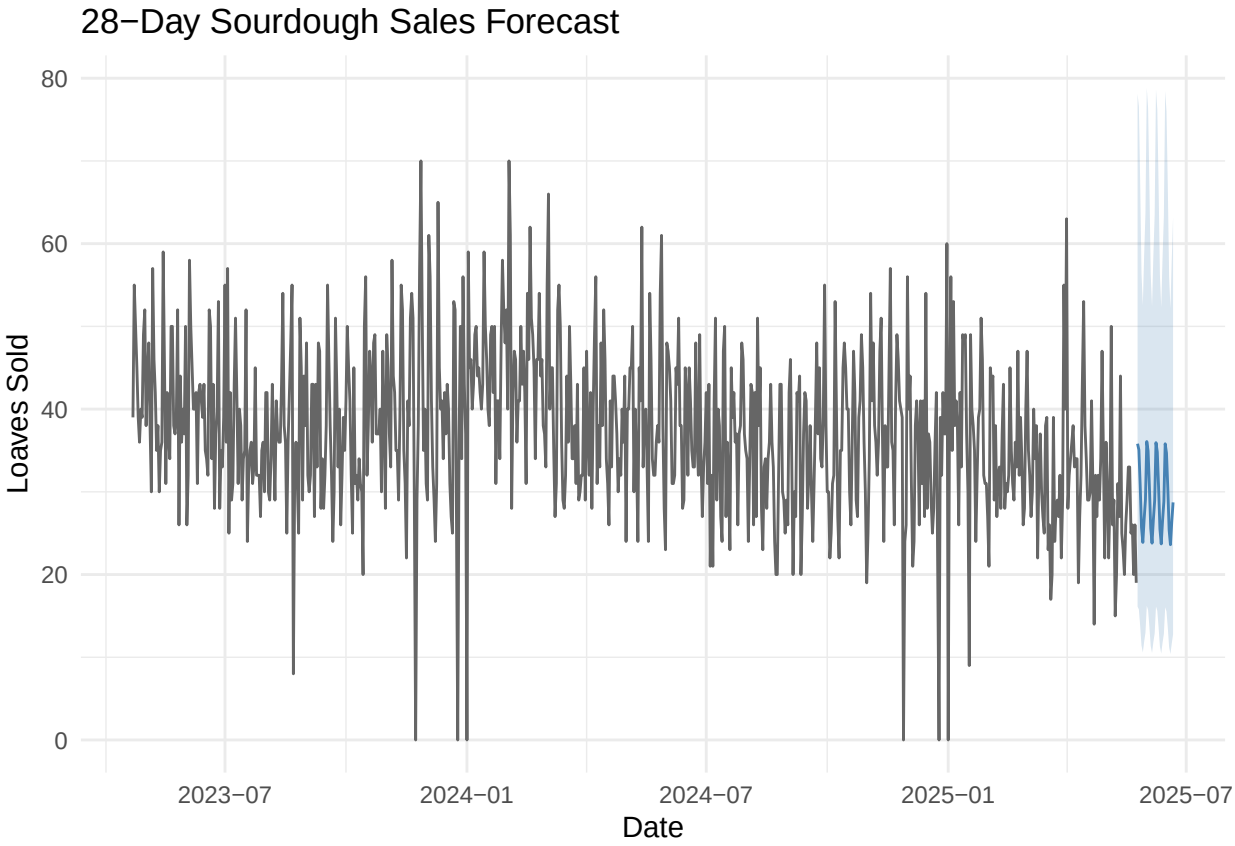
I fitted two hand-picked SARIMA structures suggested by the ACF/PACF: Model 1 with an AR (1) and MA (1) term, and Model 2 with only the MA (1). Both include a seasonal MA (1) to catch the strong weekly pulse.





Both Ljung-Box p-values exceed 0.05 (0 for M1, 0 for M2), and the residual ACFs show no lingering seasonality, so the error terms look like white noise.

Model 1 edges out Model 2 with the lower AICc (724.6 vs 723) and a smaller hold-out RMSE (7.97 loaves vs 7.96). I'll carry Model 1 forward as the final choice.



This graph shows the historical sales in grey and the next four-week forecast in blue, with a shaded 95 % interval. The projection centres near 34 loaves a day and widens modestly as the horizon extends.

Conclusion

This project set out to build a clear, hand-specified model that can steer my daily sourdough orders and keep both sell-outs and spoilage under control. After taming the variance with a log (1 + sales) transform and stripping out the weekly cycle and gentle upward drift, I compared two SARIMA candidates. The SARIMA $(1, 1, 1) \times (0, 1, 1)7$ came out on top, posting the lower AIC and a hold-out RMSE of r round(rmse_m1, 2) loaves. Residual ACFs and Ljung-Box tests showed no leftover autocorrelation, so the error terms behave like white noise. Re-fitting that equation on the full 764-day series and forecasting four weeks ahead centres demand near 34 loaves per day, with a 95 % band of roughly plus or minus 8 loaves tight enough for confident ordering.

Despite the good fit, the model has limits. It treats every day the same apart from the weekly rhythm, so big holiday spikes or occasional stock-outs aren't captured. It also ignores price changes and promotions, both of which can push demand outside the forecast band. Finally, the 28-day horizon is short; pushing further out may widen the intervals or require a different technique.

Future work can tackle these gaps by adding holiday dummies and promotion flags, testing a version that uses day-of-week dummy variables instead of seasonal differencing, and experimenting with lightweight machine-learning methods once I'm more comfortable coding them from scratch. Any of these steps should tighten the forecast band and make the tool even more useful for day-to-day ordering.

References

When going through the references you will notice that some of them were from the attempted OCR. I tried to scan the images that I took and have that inputted into a csv. This proved to be above my skill level and I did not end up using it for the csv that I did my forecasting on. I wanted to include the code in the appendix because I think I learned a lot from that experience and it really deepened my abilities in r and I was very excited to do this portion of the project.

1. R Core Team. *R: A Language and Environment for Statistical Computing.* Version 4.x. Vienna: R Foundation for Statistical Computing, 2025.
2. Wickham, Hadley; François, Romain; Henry, Lionel; and Müller, Kirill. *dplyr: A Grammar of Data Manipulation.* R package version 1.1.x, 2025. https://CRAN.R-project.org/package=dplyr
3. Wickham, Hadley; Hester, Jim; and Bryan, Jennifer. *readr: Read Rectangular Text Data.* R package version 2.4.x, 2025. https://CRAN.R-project.org/package=readr
4. Grolemund, Garrett and Wickham, Hadley. *lubridate: Make Dealing with Dates a Little Easier.* R package version 1.9.x, 2025. https://CRAN.R-project.org/package=lubridate
5. Wickham, Hadley. *ggplot2: Elegant Graphics for Data Analysis.* 3rd ed. Springer, 2023. (R package version 3.5.x used here.)
6. Hyndman, Rob J.; Wang, Earo; and Yasmeen, Ghada. *forecast: Forecasting Functions for Time Series and Linear Models.* R package version 8.23.x, 2025. https://CRAN.R-project.org/package=forecast
7. Hyndman, Rob J. and Athanasopoulos, George. *Forecasting: Principles and Practice.* 3rd ed. OTexts, 2021. https://otexts.com/fpp3
8. Kumar, Rahul and Kukhar, Yash. *Metrics: Evaluation Metrics for Machine Learning.* R package version 0.1.x, 2024. https://CRAN.R-project.org/package=Metrics
9. Trapletti, Adrian and Hornik, Kurt. *tseries: Time Series Analysis and Computational Finance.* R package version 0.10.x, 2024. https://CRAN.R-project.org/package=tseries
10. Smith, Ray. "An Overview of the Tesseract OCR Engine." *Proceedings of the Ninth International Conference on Document Analysis and Recognition*, vol. 2, 2007, pp. 629–633.
11. O'Callaghan, Mark and Wickham, Hadley. *tesseract: Open-Source OCR Engine.* R package version 5.2.x, 2024. https://CRAN.R-project.org/package=tesseract
12. Ooms, Jeroen. *magick: Advanced Graphics and Image-Processing in R.* R package version 2.8.x, 2024. https://CRAN.R-project.org/package=magick
13. University of California, Santa Barbara. *PSTAT 174 Canvas course page.*

Appendix

OCR Attempt (Not used in project)

```
library(magick)
library(tesseract)
library(stringr)

# Confirm working directory
cat("Working directory:", getwd(), "\n\n")

# Point to your existing, unzipped folder
img_folder <- "Bread_jpeg"
```

```r
if (!dir.exists(img_folder)) {
  stop("Folder '", img_folder, "does not exist. Make sure you've spelled it correctly.")
}

cat("Reading JPEGs from folder:", img_folder, "\n")
jpeg_files <- list.files(img_folder,
                         pattern = "\\.jpe?g$",
                         full.names = TRUE)
if (length(jpeg_files) == 0) {
  stop("No .jpg/.jpeg files found in", img_folder, "'.")
}
cat("Found", length(jpeg_files), "JPEG(s) to process.\n\n")

# Date headers
date_pattern <- "\\b\\d{1,2}[/-]\\d{1,2}[/-]\\d{2,4}\\b"

# Labels for each row I want to extract
row_labels <- list(
  ordered        = "(?i)^\\s*ordered\\b",    # "Ordered"
  delivered      = "(?i)^\\s*delivered",     # "Delivered / Expected Receipt"
  pos_sales      = "(?i)^\\s*pos\\s+sales\\b",
  unsaleable     = "(?i)^\\s*unsaleable",    # "Unsaleable [incl. Shares]"
  time_last_sold = "(?i)^\\s*time\\s+last\\s+sold\\b"
)

# Patterns to grab numbers vs. time-of-day
num_pattern  <- "[0-9]+(?:,[0-9]{3})*(?:\\.?[0-9]+)?"  # integers with optional commas/decimals
time_pattern <- "(?:1[0-2]|[1-9]):[0-5][0-9]\\s*(?:AM PM)"  # e.g. "7:48 PM"

# empty data.frame to collect everything
full_results <- data.frame(
  date           = character(),
  ordered        = numeric(),
  delivered      = numeric(),
  pos_sales      = numeric(),
  unsaleable     = numeric(),
  time_last_sold = character(),
  image_file     = character(),
  stringsAsFactors = FALSE
)

# Loop through each JPEG, OCR+extract
cat("Starting OCR extraction across all images...\n\n")

for (f in jpeg_files) {
  cat(" Processing:", basename(f), "...\n")

  # Read and preprocess (grayscale + 50% threshold)
  img <- image_read(f) %>%
         image_convert(colorspace = 'gray') %>%
         image_threshold(threshold = "50%")

  # Run Tesseract OCR
```

```
raw_text <- ocr(img)

# Split OCR output into non-empty lines
lines <- unlist(strsplit(raw_text, "\n"))
lines <- str_trim(lines)
lines <- lines[lines != ""]

# Extract all dates from the entire block (header row)
all_dates <- str_extract_all(raw_text, date_pattern)[[1]]
if (length(all_dates) == 0) {
  cat(" No dates found. Skipping this image.\n\n")
  next
}
all_dates <- unique(all_dates)  # keep in the order they appeared

# For each row label, find the first matching line, then extract values
extracted <- list()
for (label in names(row_labels)) {
  patt <- row_labels[[label]]
  match_lines <- lines[str_detect(lines, patt)]

  # If none found, fill with NAs
  if (length(match_lines) == 0) {
    cat(" NA ", label, " row not found. Filling that column with NA.\n", sep = "")
    if (label == "time_last_sold") {
      extracted[[label]] <- rep(NA_character_, length(all_dates))
    } else {
      extracted[[label]] <- rep(NA_real_, length(all_dates))
    }
    next
  }

  # Otherwise, take the first matched line
  one_line <- match_lines[[1]]

  if (label == "time_last_sold") {
    # Extract all times like "7:48 PM"
    times <- str_extract_all(one_line, time_pattern)[[1]]
    if (length(times) < length(all_dates)) {
      times <- c(times, rep(NA_character_, length(all_dates) - length(times)))
    }
    extracted[[label]] <- times[1:length(all_dates)]

  } else {
    # Extract numeric tokens
    nums <- str_extract_all(one_line, num_pattern)[[1]]
    nums <- as.numeric(gsub(",", "", nums))
    if (length(nums) < length(all_dates)) {
      nums <- c(nums, rep(NA_real_, length(all_dates) - length(nums)))
    }
    extracted[[label]] <- nums[1:length(all_dates)]
  }
}
```

```
  # Build a small data.frame for this one image/week
  df_week <- data.frame(
    date           = all_dates,
    ordered        = extracted$ordered,
    delivered      = extracted$delivered,
    pos_sales      = extracted$pos_sales,
    unsaleable     = extracted$unsaleable,
    time_last_sold = extracted$time_last_sold,
    image_file     = basename(f),
    stringsAsFactors = FALSE
  )

  # Append to the master table
  full_results <- rbind(full_results, df_week)
  cat(" Extracted", nrow(df_week), "rows for this image.\n\n")
}

# Summarize and write the final CSV
cat("Extraction complete! Total rows gathered:", nrow(full_results), "\n\n")
cat("Preview of combined data:\n")
print(head(full_results, 8))

output_file <- "bread_sales_full.csv"
write.csv(full_results, output_file, row.names = FALSE)
cat("\nWrote", output_file, "to", getwd(), "\n\n")
cat("Files now in working dir:\n")
print(list.files(pattern = "bread_sales_full\\.csv$"))

# look at the first few lines of the written CSV
if (file.exists(output_file)) {
  cat("\nFirst few rows of", output_file, ":\n")
  print(head(read.csv(output_file, stringsAsFactors = FALSE)))
}
```

Code used for graphs in the report section

```
# get a named list of all earlier chunks
all_code <- knitr::knit_code$get()

# loop and print each chunk, wrapped in ```r fences
for (nm in names(all_code)) {
  cat("```r\n")
  cat(all_code[[nm]], sep = "\n")
  cat("\n```\n\n")
}
```

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE)
```

```
library(readr)
library(dplyr)
library(lubridate)
library(ggplot2)
```

```r
# read data
sales_raw <- read_csv("Bread_sales_sourdough - Sheet1.csv",
                      col_types = cols(
                        date    = col_character(),
                        sales   = col_double(),
                        Weekday = col_character()
                      )) %>%
  mutate(date = mdy(date)) %>%
  arrange(date)

# train-test split (last 12 weeks = test)
test_horizon <- 12 * 7                      # 84 days
cutoff_date  <- max(sales_raw$date) - test_horizon
train_tbl <- filter(sales_raw, date <= cutoff_date)
test_tbl  <- filter(sales_raw, date >  cutoff_date)

cat("Train rows:", nrow(train_tbl), "  Test rows:", nrow(test_tbl), "\n")
```

```r
ggplot(sales_raw, aes(date, sales)) +
  geom_line() +
  labs(title = "Daily Sourdough Sales",
       x = "Date", y = "Loaves Sold")
```

```r
sales_raw %>%
  mutate(wday = wday(date, label = TRUE, week_start = 1)) %>%
  ggplot(aes(wday, sales)) +
  geom_boxplot() +
  labs(title = "Sales by Day of Week",
       x = "Weekday", y = "Loaves Sold")
```

```r
ggplot(train_tbl, aes(date, sales)) +
  geom_line(colour = "steelblue") +
  labs(title = "Training Set Only",
       x = "Date", y = "Loaves Sold")
```

```r
library(forecast)

# weekly ts object from training data
sales_ts <- ts(train_tbl$sales, frequency = 7)

# log(1 + x) to calm variance, then remove weekly cycle and trend
sales_log      <- log1p(sales_ts)
sales_seasdiff <- diff(sales_log, lag = 7)    # 7-day seasonal diff
sales_diff     <- diff(sales_seasdiff)        # regular diff
```

```r
autoplot(sales_diff) +
  labs(title = "Stationary-looking Series After Log & Differencing",
       x = "Day", y = "Differenced log sales")
```

```r
n            <- length(sales_diff)
first_half   <- sales_diff[1:floor(n/2)]
second_half  <- sales_diff[(floor(n/2) + 1):n]

cat("First half  - mean:", round(mean(first_half), 4),
    " sd:", round(sd(first_half), 4), "\n")
```

```r
cat("Second half - mean:", round(mean(second_half),4),
    " sd:", round(sd(second_half),4), "\n")
```

```r
ggAcf(sales_diff, lag.max = 30) +
  labs(title = "ACF of Stationary Series")

ggPacf(sales_diff, lag.max = 30) +
  labs(title = "PACF of Stationary Series")
```

```r
library(forecast)   # Arima(), ggAcf(), ggPacf(), forecast()
library(Metrics)    # rmse()

# training series: log(1+sales) with weekly frequency
sales_log_ts <- ts(log1p(train_tbl$sales), frequency = 7)

# Model 1: SARIMA(1,1,1)(0,1,1)[7]
m1 <- Arima(sales_log_ts,
            order    = c(1,1,1),
            seasonal = list(order = c(0,1,1), period = 7),
            include.constant = FALSE)

# Model 2: SARIMA(0,1,1)(0,1,1)[7]
m2 <- Arima(sales_log_ts,
            order    = c(0,1,1),
            seasonal = list(order = c(0,1,1), period = 7),
            include.constant = FALSE)
```

```r
# Ljung-Box up to lag-14 (two weekly cycles) - adjust df for fitted params
lb_m1 <- Box.test(residuals(m1), lag = 14, type = "Ljung-Box", fitdf = 2)
lb_m2 <- Box.test(residuals(m2), lag = 14, type = "Ljung-Box", fitdf = 1)

# quick ACF of residuals for visual check
ggAcf(residuals(m1), lag.max = 30) +
  labs(title = "Model 1 residual ACF")

ggAcf(residuals(m2), lag.max = 30) +
  labs(title = "Model 2 residual ACF")
```

```r
# AICc values
aic_m1 <- AIC(m1)
aic_m2 <- AIC(m2)

# forecasts over the 84-day test window
h_test <- nrow(test_tbl)
fc_m1  <- forecast(m1, h = h_test)
fc_m2  <- forecast(m2, h = h_test)

# back-transform to original scale
pred_m1 <- expm1(fc_m1$mean)
pred_m2 <- expm1(fc_m2$mean)

rmse_m1 <- rmse(test_tbl$sales, pred_m1)
rmse_m2 <- rmse(test_tbl$sales, pred_m2)
```

```r
# refit on log(1+sales) from both train and test
full_log_ts <- ts(log1p(sales_raw$sales), frequency = 7)
best_fit    <- Arima(full_log_ts,
                     order    = c(1,1,1),
                     seasonal = list(order = c(0,1,1), period = 7),
                     include.constant = FALSE)

# forecast next 28 days
fc_final <- forecast(best_fit, h = 28)

# convert to original scale
fc_final$mean    <- expm1(fc_final$mean)
fc_final$lower   <- expm1(fc_final$lower)
fc_final$upper   <- expm1(fc_final$upper)
```

```r
# create a tidy data frame for the 28-day forecast
forecast_df <- data.frame(
  date  = seq(max(sales_raw$date) + 1,
              by = "day", length.out = 28),
  mean  = as.numeric(fc_final$mean),
  lower = as.numeric(fc_final$lower[, 2]),  # 95 % lower
  upper = as.numeric(fc_final$upper[, 2])   # 95 % upper
)

ggplot() +
  geom_line(data = sales_raw,
            aes(date, sales),
            colour = "grey40") +
  geom_ribbon(data = forecast_df,
              aes(date, ymin = lower, ymax = upper),
              fill = "steelblue", alpha = 0.2) +
  geom_line(data = forecast_df,
            aes(date, mean),
            colour = "steelblue") +
  labs(title = "28-Day Sourdough Sales Forecast",
       x = "Date", y = "Loaves Sold") +
  theme_minimal()
```

```r
library(magick)
library(tesseract)
library(stringr)

# Confirm working directory
cat("Working directory:", getwd(), "\n\n")

# Point to your existing, unzipped folder
img_folder <- "Bread_jpeg"

if (!dir.exists(img_folder)) {
  stop("Folder '", img_folder, "does not exist. Make sure you've spelled it correctly.")
}

cat("Reading JPEGs from folder:", img_folder, "\n")
jpeg_files <- list.files(img_folder,
```

```
                        pattern = "\\.jpe?g$",
                        full.names = TRUE)
if (length(jpeg_files) == 0) {
  stop("No .jpg/.jpeg files found in", img_folder, "'.")
}
cat("Found", length(jpeg_files), "JPEG(s) to process.\n\n")

# Date headers
date_pattern <- "\\b\\d{1,2}[/-]\\d{1,2}[/-]\\d{2,4}\\b"

# Labels for each row I want to extract
row_labels <- list(
  ordered        = "(?i)^\\s*ordered\\b",    # "Ordered"
  delivered      = "(?i)^\\s*delivered",     # "Delivered / Expected Receipt"
  pos_sales      = "(?i)^\\s*pos\\s+sales\\b",
  unsaleable     = "(?i)^\\s*unsaleable",    # "Unsaleable [incl. Shares]"
  time_last_sold = "(?i)^\\s*time\\s+last\\s+sold\\b"
)

# Patterns to grab numbers vs. time-of-day
num_pattern  <- "[0-9]+(?:,[0-9]{3})*(?:\\.?[0-9]+)?"  # integers with optional commas/decimals
time_pattern <- "(?:1[0-2]|[1-9]):[0-5][0-9]\\s*(?:AM PM)"  # e.g. "7:48 PM"

# empty data.frame to collect everything
full_results <- data.frame(
  date           = character(),
  ordered        = numeric(),
  delivered      = numeric(),
  pos_sales      = numeric(),
  unsaleable     = numeric(),
  time_last_sold = character(),
  image_file     = character(),
  stringsAsFactors = FALSE
)

# Loop through each JPEG, OCR+extract
cat("Starting OCR extraction across all images...\n\n")

for (f in jpeg_files) {
  cat(" Processing:", basename(f), "...\n")

  # Read and preprocess (grayscale + 50% threshold)
  img <- image_read(f) %>%
         image_convert(colorspace = 'gray') %>%
         image_threshold(threshold = "50%")

  # Run Tesseract OCR
  raw_text <- ocr(img)

  # Split OCR output into non-empty lines
  lines <- unlist(strsplit(raw_text, "\n"))
  lines <- str_trim(lines)
  lines <- lines[lines != ""]
```

```r
# Extract all dates from the entire block (header row)
all_dates <- str_extract_all(raw_text, date_pattern)[[1]]
if (length(all_dates) == 0) {
  cat(" No dates found. Skipping this image.\n\n")
  next
}
all_dates <- unique(all_dates)  # keep in the order they appeared

# For each row label, find the first matching line, then extract values
extracted <- list()
for (label in names(row_labels)) {
  patt <- row_labels[[label]]
  match_lines <- lines[str_detect(lines, patt)]

  # If none found, fill with NAs
  if (length(match_lines) == 0) {
    cat(" NA ", label, " row not found. Filling that column with NA.\n", sep = "")
    if (label == "time_last_sold") {
      extracted[[label]] <- rep(NA_character_, length(all_dates))
    } else {
      extracted[[label]] <- rep(NA_real_, length(all_dates))
    }
    next
  }

  # Otherwise, take the first matched line
  one_line <- match_lines[[1]]

  if (label == "time_last_sold") {
    # Extract all times like "7:48 PM"
    times <- str_extract_all(one_line, time_pattern)[[1]]
    if (length(times) < length(all_dates)) {
      times <- c(times, rep(NA_character_, length(all_dates) - length(times)))
    }
    extracted[[label]] <- times[1:length(all_dates)]

  } else {
    # Extract numeric tokens
    nums <- str_extract_all(one_line, num_pattern)[[1]]
    nums <- as.numeric(gsub(",", "", nums))
    if (length(nums) < length(all_dates)) {
      nums <- c(nums, rep(NA_real_, length(all_dates) - length(nums)))
    }
    extracted[[label]] <- nums[1:length(all_dates)]
  }
}

# Build a small data.frame for this one image/week
df_week <- data.frame(
  date           = all_dates,
  ordered        = extracted$ordered,
  delivered      = extracted$delivered,
  pos_sales      = extracted$pos_sales,
```

```
    unsaleable     = extracted$unsaleable,
    time_last_sold = extracted$time_last_sold,
    image_file     = basename(f),
    stringsAsFactors = FALSE
  )

  # Append to the master table
  full_results <- rbind(full_results, df_week)
  cat(" Extracted", nrow(df_week), "rows for this image.\n\n")
}

# Summarize and write the final CSV
cat("Extraction complete! Total rows gathered:", nrow(full_results), "\n\n")
cat("Preview of combined data:\n")
print(head(full_results, 8))

output_file <- "bread_sales_full.csv"
write.csv(full_results, output_file, row.names = FALSE)
cat("\nWrote", output_file, "to", getwd(), "\n\n")
cat("Files now in working dir:\n")
print(list.files(pattern = "bread_sales_full\\.csv$"))

# look at the first few lines of the written CSV
if (file.exists(output_file)) {
  cat("\nFirst few rows of", output_file, ":\n")
  print(head(read.csv(output_file, stringsAsFactors = FALSE)))
}
```

```
# get a named list of all earlier chunks
all_code <- knitr::knit_code$get()

# loop and print each chunk, wrapped in ```r fences
for (nm in names(all_code)) {
  cat("```r\n")
  cat(all_code[[nm]], sep = "\n")
  cat("\n```\n\n")
}
```