## Technical Implementation: The R Pipeline

The following R code is structured into modular blocks. It is designed to be run in a local RStudio environment or a containerized batch job.

### Environment Setup and Library Loading

R  
  
# Load required packages  
if (!require("pacman")) install.packages("pacman")  
pacman::p\_load(  
 readxl, # For reading Excel sheets  
 dplyr, # For data manipulation  
 tidyr, # For reshaping data  
 stringr, # For regex and string cleaning  
 lubridate, # For date handling  
 purrr, # For iteration  
 janitor, # For cleaning column names  
 readr # For CSV export  
)  
  
# Configuration Parameters  
INPUT\_FILE <- "Project 1\_Bus Terminal Passenger Prediction Data Fall 2025.xlsx"  
OUTPUT\_FILE <- "PAB\_Cleaned\_Dataset.csv"  
  
# -----------------------------------------------------------------------------  
# Domain Mapping  
# -----------------------------------------------------------------------------  
# Hardcoded mapping based on snippet vs analysis.  
# This resolves the entity resolution challenge.  
carrier\_type\_map <- c(  
 "Academy" = "Commuter Bus Departures",  
 "C & J Bus Lines" = "Commuter Bus Departures",  
 "C&J" = "Commuter Bus Departures",  
 "Coach USA" = "Commuter Bus Departures",  
 "DeCamp" = "Commuter Bus Departures",  
 "Greyhound" = "Intercity Bus Departures",  
 "HCEE - Community" = "Commuter Bus Departures",  
 "HCEE" = "Commuter Bus Departures",  
 "Lakeland" = "Commuter Bus Departures",  
 "Martz" = "Commuter Bus Departures",  
 "NJ Transit" = "Commuter Bus Departures",  
 "Peter Pan/Bonanza" = "Intercity Bus Departures",  
 "Peter Pan" = "Intercity Bus Departures",  
 "Trailways" = "Intercity Bus Departures",  
 "TransBridge" = "Commuter Bus Departures",  
 "Trans-Bridge" = "Commuter Bus Departures",  
 "Megabus" = "Intercity Bus Departures"  
)  
  
# Seasonality Lookup Function  
get\_season <- function(dt) {  
 m <- month(dt)  
 case\_when(  
 m %in% c(12, 1, 2) ~ "Winter",  
 m %in% c(3, 4, 5) ~ "Spring",  
 m %in% c(6, 7, 8) ~ "Summer",  
 m %in% c(9, 10, 11)~ "Fall",  
 TRUE ~ "Unknown"  
 )  
}

### The Extraction Logic (Functional)

R

# -----------------------------------------------------------------------------  
# Function: Parse a single sheet, find the data grid, and normalize columns.  
# -----------------------------------------------------------------------------  
process\_sheet <- function(sheet\_name, file\_path, current\_year\_context) {  
 message(paste("Processing Sheet:", sheet\_name, "| Year Context:", current\_year\_context))  
 # Step 1: Read raw content to find the header anchor  
 # We read the first 20 rows to locate the 'Carrier' column  
 raw\_header\_scan <- read\_excel(file\_path, sheet = sheet\_name, n\_max = 20, col\_names = FALSE,.name\_repair = "minimal")  
 # Find row index containing "Carrier" (case insensitive)  
 anchor\_row <- which(apply(raw\_header\_scan, 1, function(x) any(str\_detect(x, regex("Carrier", ignore\_case = TRUE)), na.rm = TRUE)))  
 if (is.na(anchor\_row)) {  
 warning(paste("Skipping sheet", sheet\_name, "- No 'Carrier' anchor found."))  
 return(NULL)  
 }  
 # Step 2: Extract Temporal Info from Headers (Rows 1 to anchor\_row)  
 header\_text <- paste(raw\_header\_scan[1:(anchor\_row-1), ], collapse = " ")  
 # Regex to find date ranges like "Dec 7-11" or "Jan 4-8"  
 # This helps us confirm if we need to increment the year.  
 # We extract the first 3-letter month string found.  
 month\_match <- str\_extract(sheet\_name, "[A-Za-z]{3}")  
 # Step 3: Read the actual Data Table  
 df <- read\_excel(file\_path, sheet = sheet\_name, skip = anchor\_row - 1,.name\_repair = "unique") %>%  
 clean\_names()  
 # Step 4: Normalize Column Names  
 # We need 'carrier', 'departures', and 'passengers'.  
 # The raw data often has 'bus\_departures', 'bus\_departures\_2', etc.  
 # The FIRST 'departures' column is typically the "Current Week".  
 # Identify columns  
 col\_names <- names(df)  
 carrier\_col <- col\_names[str\_detect(col\_names, "carrier")]  
 # Finding the correct departure column is tricky due to comparisons.  
 # Usually, column B is current week departures.  
 # We look for columns containing "departures" and take the first one.  
 dept\_cols <- col\_names[str\_detect(col\_names, "departures")]  
 dept\_col <- dept\_cols   
 # Passenger column detection  
 # Snippet shows passengers might be in a separate block.  
 # Snippet shows columns like 'passengers'.   
 # If standard 'passengers' isn't found, we look for 'pax'.  
 pass\_col <- col\_names[str\_detect(col\_names, "passengers|pax")]  
 # Renaming for consistency  
 df\_clean <- df %>%  
 rename(CARRIER =!!carrier\_col)  
 if (!is.na(dept\_col)) {  
 df\_clean <- df\_clean %>% rename(Departures =!!dept\_col)  
 } else {  
 df\_clean$Departures <- NA  
 }  
   
 if (!is.na(pass\_col)) {  
 df\_clean <- df\_clean %>% rename(PASSENGERS =!!pass\_col)  
 } else {  
 df\_clean$PASSENGERS <- NA   
 }  
   
 # Step 5: Clean Rows  
 df\_clean <- df\_clean %>%  
 select(CARRIER, Departures, PASSENGERS) %>%  
 filter(!is.na(CARRIER)) %>%  
 # Remove aggregate rows often found in raw sheets  
 filter(!str\_detect(CARRIER, regex("Total|Variance|Average|Normal", ignore\_case = TRUE))) %>%  
 # Remove empty spacer rows  
 filter(!is.na(Departures) |!is.na(PASSENGERS))  
   
 # Step 6: Parse Dates based on Sheet Name (Most reliable source in this dataset)  
 # Cleaning sheet name: "Dec 7-11" -> "Dec 7"  
 clean\_date\_str <- str\_replace(sheet\_name, "\\.csv$", "")  
 clean\_date\_str <- str\_extract(clean\_date\_str, "[A-Za-z]+\\s\*[0-9]+") # Extracts "Dec 7"  
   
 # Construct full date string with context year  
 full\_date\_str <- paste(current\_year\_context, clean\_date\_str)  
 start\_date <- parse\_date\_time(full\_date\_str, orders = c("Y b d", "Y B d"))  
   
 # Handle parsing failures (e.g., if sheet name is just text)  
 if (is.na(start\_date)) {  
 # Fallback to header text  
 month\_day <- str\_extract(header\_text, "(January|February|March|April|May|June|July|August|September|October|November|December)\\s+[0-9]{1,2}")  
 if (!is.na(month\_day)) {  
 full\_date\_str <- paste(current\_year\_context, month\_day)  
 start\_date <- parse\_date\_time(full\_date\_str, orders = c("Y B d"))  
 }  
 }  
 end\_date <- start\_date + days(4) # Assuming 5-day week as per "Week of..."  
   
 # Step 7: Enrichment  
 df\_clean <- df\_clean %>%  
 mutate(  
 SOURCE\_SHEET = sheet\_name,  
 YEAR = year(start\_date),  
 MONTH = month(start\_date, label = TRUE, abbr = TRUE),  
 START\_DATE = start\_date,  
 END\_DATE = end\_date,  
 WEEK = sheet\_name,  
 `WeekDay Name` = wday(start\_date, label = TRUE, abbr = FALSE),  
 Seasonality = get\_season(start\_date)  
 )  
   
 return(df\_clean)  
}  
Pipeline Orchestration and Accumulation

R

# -----------------------------------------------------------------------------  
# Execution Block  
# -----------------------------------------------------------------------------  
  
# Get all sheet names  
all\_sheets <- excel\_sheets(INPUT\_FILE)  
  
# Logic to handle Year rollover  
# We assume the file is sorted chronologically.  
# Start year: 2020   
current\_year <- 2020  
previous\_month\_index <- 12 # December  
  
master\_list <- list()  
  
for (sheet in all\_sheets) {  
   
 # 1. Determine Month of current sheet to check for Year Rollover  
 # Regex to find month name  
 sheet\_month\_str <- str\_extract(sheet, "[A-Za-z]+")  
 # Convert to integer (1-12)  
 current\_month\_index <- match(tolower(substr(sheet\_month\_str, 1, 3)), tolower(month.abb))  
   
 # 2. Logic: If we go from Dec (12) to Jan (1), increment year  
 if (!is.na(current\_month\_index)) {  
 if (previous\_month\_index == 12 && current\_month\_index == 1) {  
 current\_year <- current\_year + 1  
 message(paste(">>> Year Transition Detected. Now processing:", current\_year))  
 }  
 previous\_month\_index <- current\_month\_index  
 }  
   
 # 3. Process the sheet  
 processed\_data <- process\_sheet(sheet, INPUT\_FILE, current\_year)  
   
 if (!is.null(processed\_data)) {  
 master\_list[[sheet]] <- processed\_data  
 }  
}  
  
# Accumulate all sheets into one dataframe  
pab\_dataset <- bind\_rows(master\_list)

### Post-Processing: Entity Resolution and Feature EngineeringR

# -----------------------------------------------------------------------------  
# Post-Processing & Enrichment  
# -----------------------------------------------------------------------------  
  
final\_pab\_dataset <- pab\_dataset %>%  
 # 1. Clean Carrier Names (Remove whitespace, standardize)  
 mutate(  
 CARRIER = str\_trim(CARRIER),  
 # Map synonyms to standard names  
 CARRIER = case\_when(  
 str\_detect(CARRIER, "(?i)Trans.\*Bridge") ~ "Trans-Bridge",  
 str\_detect(CARRIER, "(?i)HCEE") ~ "HCEE - Community",  
 str\_detect(CARRIER, "(?i)Peter.\*Pan") ~ "Peter Pan/Bonanza",  
 str\_detect(CARRIER, "(?i)C.\*&.\*J") ~ "C & J Bus Lines",  
 TRUE ~ CARRIER  
 )  
 ) %>%  
   
 # 2. Map Routes (Commuter vs Intercity)  
 mutate(  
 ROUTE = carrier\_type\_map,  
 # Fallback for unknown carriers  
 ROUTE = ifelse(is.na(ROUTE), "Commuter Bus Departures", ROUTE)  
 ) %>%  
   
 # 3. Add Weather and Holiday Placeholders  
 # Real-world implementation would join with a NOAA API here.   
 # We will approximate based on Seasonality as per target snippet.  
 mutate(  
 `Weather Condition` = case\_when(  
 Seasonality == "Winter" ~ "Cold / Wintry Mix",  
 Seasonality == "Spring" ~ "Mild / Variable",  
 Seasonality == "Summer" ~ "Hot / Humid",  
 Seasonality == "Fall" ~ "Warm / Pleasant",  
 TRUE ~ "Variable"  
 ),  
   
 # Simple Holiday Logic (Example)  
 `Holiday Status` = case\_when(  
 MONTH == "Dec" & day(START\_DATE) > 20 ~ "Christmas",  
 MONTH == "Jul" & day(START\_DATE) < 10 ~ "Independence Day",  
 MONTH == "Sep" & day(START\_DATE) < 8 ~ "Labor Day",  
 MONTH == "Nov" & day(START\_DATE) > 20 ~ "Thanksgiving",  
 TRUE ~ "Regular Week"  
 ),  
   
 `Holiday\_In\_Week` = ifelse(`Holiday Status`!= "Regular Week", 1, 0),  
 `Holiday\_Names` = ifelse(`Holiday\_In\_Week` == 1, `Holiday Status`, NA)  
 ) %>%  
   
 # 4. Final Formatting to Match Target Schema   
 mutate(  
 DATE = paste(format(START\_DATE, "%m/%d/%Y"), "-", format(END\_DATE, "%m/%d/%Y"))  
 ) %>%  
 select(  
 DATE, WEEK, YEAR, CARRIER, ROUTE, Departures, PASSENGERS,  
 SOURCE\_SHEET, MONTH, START\_DATE, END\_DATE, `WeekDay Name`,  
 Seasonality, `Weather Condition`, `Holiday Status`,   
 `Holiday\_In\_Week`, `Holiday\_Names`  
 ) %>%  
   
 # 5. Remove Duplicates  
 distinct()  
  
# Export  
write\_csv(final\_pab\_dataset, OUTPUT\_FILE)  
message("Transformation Complete. File saved.")