FINAL RECOMMENDATIONS – GROUP 5

PROJECT RESOURCES:

PABT TIME SERIES FORECASTING - INTERACTIVE SHINY APP

Upload data, select models, generate forecasts, download results

install.packages("shiny") library(shiny) install.packages("shinydashboard") library(shinydashboard) install.packages("shinyWidgets") library(shinyWidgets) library(tidyverse) library(lubridate) library(forecast) library(prophet) install.packages("plotly") library(plotly) library(DT) library(readxl) library(writexl)

UI (USER INTERFACE)

ui <- dashboardPage(

Header

dashboardHeader(title = "PABT Passenger Forecasting"),

Sidebar

dashboardSidebar(sidebarMenu(menuItem(" 📊 Upload Data", tabName = "upload", icon = icon("upload")), menuItem(" 📈 Explore Data", tabName = "explore", icon = icon("chart-line")), menuItem(" 🔮 Forecast", tabName = "forecast", icon = icon("crystal-ball")), menuItem(" 📉 Model Validation", tabName = "validation", icon = icon("check-circle")), menuItem(" 💾 Download Results", tabName = "download", icon = icon("download")))),

Body

dashboardBody(

 # TAB 3: FORECAST

```
 tabItem(
   tabName = "forecast",

   fluidRow(
    box(
```

```
  title = "Forecasting Parameters",
  width = 4,
  solidHeader = TRUE,
  status = "primary",

  h4("Select Models:"),
  checkboxGroupInput("models",
                     NULL,
                     choices = c("ARIMA" = "arima",
                                 "ETS (Exponential Smoothing)" = "ets",
                                 "Prophet" = "prophet",
                                 "TBATS" = "tbats"),
                     selected = c("arima", "ets", "prophet")),

  hr(),

  sliderInput("forecast_years",
              "Forecast Horizon (years):",
              min = 1, max = 10, value = 5, step = 1),

  sliderInput("train_split",
              "Training Data (%):",
              min = 70, max = 95, value = 85, step = 5),

  hr(),

  sliderInput("confidence_level",
              "Confidence Level (%):",
              min = 80, max = 99, value = 95, step = 5),

  hr(),

  actionButton("run_forecast", "Generate Forecast",
               icon = icon("play"),
               class = "btn-success btn-lg",
               style = "width: 100%;")
),

box(
  title = "Forecast Results",
  width = 8,
```

```
    solidHeader = TRUE,
    status = "success",

    conditionalPanel(
      condition = "input.run_forecast == 0",
      h4("Configure parameters and click 'Generate Forecast'",
         style = "color: gray; text-align: center; padding: 50px;")
    ),

    conditionalPanel(
      condition = "input.run_forecast > 0",
      plotlyOutput("forecast_plot", height = "400px")
    )
  )
),

fluidRow(
  box(
    title = "Annual Forecast Summary",
    width = 12,
    solidHeader = TRUE,
    status = "info",

    conditionalPanel(
      condition = "input.run_forecast > 0",
      plotlyOutput("annual_forecast_plot", height = "300px")
    )
  )
),

fluidRow(
  box(
    title = "Forecast Data Table",
    width = 12,
    solidHeader = TRUE,
    status = "warning",

    conditionalPanel(
      condition = "input.run_forecast > 0",
      DTOutput("forecast_table")
    )
```

```
    )
  )
),

# TAB 4: MODEL VALIDATION

tabItem(
  tabName = "validation",

  fluidRow(
    box(
      title = "Actual vs Predicted (Test Set)",
      width = 12,
      solidHeader = TRUE,
      status = "primary",

      conditionalPanel(
        condition = "input.run_forecast == 0",
        h4("Run forecast first to see validation results",
           style = "color: gray; text-align: center; padding: 50px;")
      ),

      conditionalPanel(
        condition = "input.run_forecast > 0",
        plotlyOutput("validation_plot", height = "400px")
      )
    )
  ),

  fluidRow(
    box(
      title = "Model Performance Metrics",
      width = 6,
      solidHeader = TRUE,
      status = "info",

      conditionalPanel(
        condition = "input.run_forecast > 0",
        DTOutput("metrics_table")
      )
```

```
    ),

    box(
      title = "Best Model Recommendation",
      width = 6,
      solidHeader = TRUE,
      status = "success",

      conditionalPanel(
        condition = "input.run_forecast > 0",
        htmlOutput("best_model_info")
      )
    )
  )
),


# TAB 5: DOWNLOAD RESULTS

tabItem(
  tabName = "download",

  fluidRow(
    box(
      title = "Download Forecast Results",
      width = 12,
      solidHeader = TRUE,
      status = "primary",

      conditionalPanel(
        condition = "input.run_forecast == 0",
        h4("Generate forecast first to download results",
           style = "color: gray; text-align: center; padding: 50px;")
      ),

      conditionalPanel(
        condition = "input.run_forecast > 0",

        h4("Available Downloads:"),

        fluidRow(
```

```
        column(6,
            downloadButton("download_forecast_csv", "Download Forecast (CSV)",
                           class = "btn-info btn-block",
                           style = "margin-bottom: 10px;"),

            downloadButton("download_forecast_excel", "Download Complete Report
(Excel)",
                           class = "btn-success btn-block",
                           style = "margin-bottom: 10px;")
        ),

        column(6,
            downloadButton("download_metrics", "Download Model Metrics (CSV)",
                           class = "btn-warning btn-block",
                           style = "margin-bottom: 10px;"),

            downloadButton("download_plots", "Download Plots (PDF)",
                           class = "btn-danger btn-block",
                           style = "margin-bottom: 10px;")
        )
       ),

       hr(),

       h4("Export Summary:"),
       verbatimTextOutput("export_summary")
      )
     )
    )
   )
)


) )
```

SERVER LOGIC

```
server <- function(input, output, session) {
```

Reactive values to store data

```
rv <- reactiveValues( data = NULL, forecast_results = NULL, validation_results = NULL,
metrics = NULL )
```

DATA LOADING

Load uploaded data

```
observeEvent(input$datafile, { req(input$datafile)

ext <- tools::file_ext(input$datafile$name)

tryCatch({
if (ext == "csv") {
   rv$data <- read_csv(input$datafile$datapath, show_col_types = FALSE)
 } else if (ext %in% c("xlsx", "xls")) {
   rv$data <- read_excel(input$datafile$datapath)
 }

 # Try to identify date and passenger columns
 date_col <- names(rv$data)[str_detect(tolower(names(rv$data)), "date")]
 passenger_col <- names(rv$data)[str_detect(tolower(names(rv$data)),
"passenger|total|count")]

 if (length(date_col) > 0 && length(passenger_col) > 0) {
   rv$data <- rv$data %>%
     rename(date = !!date_col[1], total_passengers = !!passenger_col[1]) %>%
     mutate(date = as.Date(date)) %>%
     select(date, total_passengers) %>%
     arrange(date)
 }

 showNotification("Data loaded successfully!", type = "message")

}, error = function(e) {
 showNotification(paste("Error loading file:", e$message), type = "error")
})


})
```

Load sample data

```
observeEvent(input$use_sample, {

# Create sample weekly data (2019-2025)
set.seed(123)
dates <- seq(as.Date("2019-01-07"), as.Date("2025-12-29"), by = "week")

sample_data <- tibble(date = dates) %>%
  mutate(
 year = year(date),
   week_of_year = week(date),

   # Trend with COVID impact
   trend_factor = case_when(
    year <= 2019 ~ 1.0,
    year == 2020 ~ 0.3,
    year == 2021 ~ 0.5,
    year == 2022 ~ 0.7,
    year == 2023 ~ 0.85,
    year >= 2024 ~ 0.95
   ),

   # Seasonality
   seasonal_factor = 1 + 0.15 * sin(2 * pi * (week_of_year - 10) / 52),

   # Calculate passengers
   total_passengers = 32000 * trend_factor * seasonal_factor + rnorm(n(), 0, 2000)
 ) %>%
 select(date, total_passengers)

rv$data <- sample_data
showNotification("Sample data loaded!", type = "message")


})
```

TAB 1: DATA PREVIEW

```
output$data_preview <- renderDT({ req(rv$data) datatable(rv$data, options =
list(pageLength = 10)) })
```

```
output$total_weeks <- renderValueBox({ req(rv$data) valueBox( nrow(rv$data), "Total Weeks", icon = icon("calendar"), color = "blue" ) })

output$date_range <- renderValueBox({ req(rv$data)
valueBox( paste(min(rv$data$date), "to", max(rv$data$date)), "Date Range", icon = icon("calendar-range"), color = "green" ) })

output$avg_passengers <- renderValueBox({ req(rv$data)
valueBox( format(round(mean(rv$data$total_passengers)), big.mark = ","), "Avg Weekly Passengers", icon = icon("users"), color = "yellow" ) })
```

TAB 2: EXPLORATORY ANALYSIS

```
output$trend_plot <- renderPlotly({ req(rv$data)

p <- ggplot(rv$data, aes(x = date, y = total_passengers)) +
  geom_line(color = "steelblue", size = 1) +
  geom_smooth(method = "loess", color = "red", se = TRUE) +
  labs(title = "Historical Weekly Passenger Volumes",
       x = "Date", y = "Weekly Passengers") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()

ggplotly(p)


})

output$decomp_plot <- renderPlot({ req(rv$data)

ts_obj <- ts(rv$data$total_passengers, frequency = 52)
decomp <- stl(ts_obj, s.window = "periodic")
plot(decomp, main = "Time Series Decomposition")


})

output$seasonal_plot <- renderPlotly({ req(rv$data)

seasonal_data <- rv$data %>%
  mutate(month = month(date, label = TRUE)) %>%
  group_by(month) %>%
  summarise(avg_passengers = mean(total_passengers), .groups = "drop")
```

```
p <- ggplot(seasonal_data, aes(x = month, y = avg_passengers, group = 1)) +
  geom_line(color = "darkgreen", size = 1.2) +
  geom_point(color = "darkgreen", size = 3) +
  labs(title = "Average Weekly Passengers by Month",
       x = "Month", y = "Average Passengers") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()

ggplotly(p)


})

output$summary_stats <- renderPrint({ req(rv$data)
summary(rv$data$total_passengers) })
```

TAB 3: FORECASTING

```
observeEvent(input$run_forecast, { req(rv$data) req(length(input$models) > 0)

withProgress(message = 'Generating forecasts...', value = 0, {

  # Split data
  split_point <- floor(nrow(rv$data) * input$train_split / 100)
  train_data <- rv$data[1:split_point, ]
  test_data <- rv$data[(split_point + 1):nrow(rv$data), ]

  incProgress(0.1, detail = "Preparing data")

  # Create ts objects
  train_ts <- ts(train_data$total_passengers, frequency = 52)
  test_ts <- ts(test_data$total_passengers, frequency = 52)

  # Forecast horizon
  h_forecast <- input$forecast_years * 52
  h_test <- nrow(test_data)

  # Storage for results
  forecasts <- list()
  test_predictions <- list()
  metrics_list <- list()
```

```
# ARIMA
if ("arima" %in% input$models) {
  incProgress(0.2, detail = "Building ARIMA model")
  model_arima <- auto.arima(train_ts, seasonal = TRUE)
  forecasts$arima <- forecast(model_arima, h = h_forecast)
  test_predictions$arima <- as.numeric(forecast(model_arima, h = h_test)$mean)
}

# ETS
if ("ets" %in% input$models) {
  incProgress(0.3, detail = "Building ETS model")
  model_ets <- ets(train_ts)
  forecasts$ets <- forecast(model_ets, h = h_forecast)
  test_predictions$ets <- as.numeric(forecast(model_ets, h = h_test)$mean)
}

# Prophet
if ("prophet" %in% input$models) {
  incProgress(0.4, detail = "Building Prophet model")
  prophet_train <- train_data %>% select(ds = date, y = total_passengers)
  model_prophet <- prophet(prophet_train, yearly.seasonality = TRUE,
                           weekly.seasonality = FALSE)
  future_prophet <- make_future_dataframe(model_prophet,
                                          periods = h_forecast, freq = "week")
  forecast_prophet <- predict(model_prophet, future_prophet)

  forecasts$prophet <- forecast_prophet %>%
    filter(ds > max(train_data$date))

  test_future <- make_future_dataframe(model_prophet,
                                       periods = h_test, freq = "week")
  test_pred_prophet <- predict(model_prophet, test_future)
  test_predictions$prophet <- tail(test_pred_prophet$yhat, h_test)
}

# TBATS
if ("tbats" %in% input$models) {
  incProgress(0.5, detail = "Building TBATS model")
  model_tbats <- tbats(train_ts)
  forecasts$tbats <- forecast(model_tbats, h = h_forecast)
```

```
  test_predictions$tbats <- as.numeric(forecast(model_tbats, h = h_test)$mean)
 }

 incProgress(0.7, detail = "Calculating metrics")

 # Calculate metrics
 actual_test <- test_data$total_passengers

 for (model_name in names(test_predictions)) {
  pred <- test_predictions[[model_name]]
  errors <- actual_test - pred

  metrics_list[[model_name]] <- data.frame(
   Model = toupper(model_name),
   RMSE = sqrt(mean(errors^2)),
   MAE = mean(abs(errors)),
   MAPE = mean(abs(errors / actual_test)) * 100
  )
 }

 rv$metrics <- bind_rows(metrics_list) %>% arrange(RMSE)

 incProgress(0.9, detail = "Finalizing results")

 # Compile forecast results
 last_date <- max(rv$data$date)
 future_dates <- seq(last_date + 7, last_date + (h_forecast * 7), by = "week")

 forecast_df <- tibble(date = future_dates[1:h_forecast])

 for (model_name in names(forecasts)) {
  if (model_name == "prophet") {
   forecast_df[[paste0(model_name, "_forecast")]] <- forecasts[[model_name]]$yhat
   forecast_df[[paste0(model_name, "_lower")]] <-
forecasts[[model_name]]$yhat_lower
   forecast_df[[paste0(model_name, "_upper")]] <-
forecasts[[model_name]]$yhat_upper
  } else {
   forecast_df[[paste0(model_name, "_forecast")]] <-
as.numeric(forecasts[[model_name]]$mean)
   forecast_df[[paste0(model_name, "_lower")]] <-
```

```
as.numeric(forecasts[[model_name]]$lower[, 2])
    forecast_df[[paste0(model_name, "_upper")]] <-
as.numeric(forecasts[[model_name]]$upper[, 2])
  }
 }

 # Calculate ensemble
 forecast_cols <- names(forecast_df)[str_detect(names(forecast_df), "_forecast$")]
 forecast_df$ensemble <- rowMeans(forecast_df[, forecast_cols], na.rm = TRUE)

 rv$forecast_results <- forecast_df

 # Store validation results
 validation_df <- test_data %>%
  mutate(actual = actual_test)

 for (model_name in names(test_predictions)) {
  validation_df[[model_name]] <- test_predictions[[model_name]]
 }

 rv$validation_results <- validation_df

 incProgress(1, detail = "Complete!")

 showNotification("Forecasts generated successfully!", type = "message")
})


})

output$forecast_plot <- renderPlotly({ req(rv$forecast_results) req(rv$data)

# Combine historical and forecast
historical <- rv$data %>%
 mutate(type = "Historical")
forecast_long <- rv$forecast_results %>%
 pivot_longer(cols = ends_with("_forecast"),
         names_to = "model", values_to = "passengers") %>%
 mutate(
  model = str_remove(model, "_forecast"),
  type = "Forecast"
 ) %>%
```

```
  select(date, model, passengers, type)

p <- ggplot() +
  geom_line(data = historical, aes(x = date, y = total_passengers),
            color = "black", size = 1) +
  geom_line(data = forecast_long,
            aes(x = date, y = passengers, color = model),
            size = 1, alpha = 0.8) +
  geom_vline(xintercept = as.numeric(max(rv$data$date)),
             linetype = "dashed", color = "gray") +
  labs(title = "Historical Data + Forecasts",
       x = "Date", y = "Weekly Passengers") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal() +
  theme(legend.position = "bottom")

ggplotly(p)


})

output$annual_forecast_plot <- renderPlotly({ req(rv$forecast_results)

annual_summary <- rv$forecast_results %>%
  mutate(year = year(date)) %>%
  group_by(year) %>%
  summarise(ensemble_total = sum(ensemble), .groups = "drop")

p <- ggplot(annual_summary, aes(x = as.factor(year), y = ensemble_total)) +
  geom_col(fill = "steelblue") +
  geom_text(aes(label = scales::comma(round(ensemble_total))), vjust = -0.5) +
  labs(title = "Annual Forecast (Ensemble)",
       x = "Year", y = "Total Annual Passengers") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal()

ggplotly(p)


})
```

```
output$forecast_table <- renderDT({ req(rv$forecast_results)
datatable(rv$forecast_results, options = list(pageLength = 10)) %>%
formatRound(columns = 2:ncol(rv$forecast_results), digits = 0) })
```

TAB 4: VALIDATION

```
output$validation_plot <- renderPlotly({ req(rv$validation_results)

validation_long <- rv$validation_results %>%
  pivot_longer(cols = -date, names_to = "type", values_to = "passengers")

p <- ggplot(validation_long, aes(x = date, y = passengers, color = type)) +
  geom_line(size = 1) +
  geom_point(size = 2) +
  labs(title = "Actual vs Predicted (Test Set)",
       x = "Date", y = "Weekly Passengers") +
  scale_y_continuous(labels = scales::comma) +
  theme_minimal() +
  theme(legend.position = "bottom")

ggplotly(p)


})

output$metrics_table <- renderDT({ req(rv$metrics) datatable(rv$metrics, options =
list(pageLength = 10)) %>% formatRound(columns = 2:4, digits = 2) })

output$best_model_info <- renderUI({ req(rv$metrics)

best_model <- rv$metrics$Model[1]
best_rmse <- rv$metrics$RMSE[1]
best_mape <- rv$metrics$MAPE[1]

HTML(paste0(
  "<div style='padding: 20px; background-color: #d4edda; border-radius: 5px;'>",
  "<h3 style='color: #155724;'>✓ Best Model: ", best_model, "</h3>",
  "<p style='font-size: 16px;'><strong>RMSE:</strong> ", round(best_rmse, 0), "</p>",
  "<p style='font-size: 16px;'><strong>MAPE:</strong> ", round(best_mape, 2), "%</p>",
  "<p style='margin-top: 15px;'>This model performed best on the test set and is
recommended for forecasting.</p>",
  "</div>"
))
```

```
})
```

TAB 5: DOWNLOADS

```
output$download_forecast_csv <- downloadHandler( filename = function()
{ paste0("PABT_Forecast_", Sys.Date(), ".csv") }, content = function(file)
{ write_csv(rv$forecast_results, file) } )

output$download_forecast_excel <- downloadHandler( filename = function()
{ paste0("PABT_Complete_Report_", Sys.Date(), ".xlsx") }, content = function(file) {

 annual_summary <- rv$forecast_results %>%
   mutate(year = year(date)) %>%
   group_by(year) %>%
   summarise(across(ends_with("_forecast"), sum, na.rm = TRUE),
          .groups = "drop")

  export_list <- list(
   "Weekly_Forecast" = rv$forecast_results,
   "Annual_Summary" = annual_summary,
   "Model_Metrics" = rv$metrics,
   "Validation_Results" = rv$validation_results,
   "Historical_Data" = rv$data
  )

  write_xlsx(export_list, file)
}


)

output$download_metrics <- downloadHandler( filename = function()
{ paste0("PABT_Model_Metrics_", Sys.Date(), ".csv") }, content = function(file)
{ write_csv(rv$metrics, file) } )

output$export_summary <- renderPrint({ req(rv$forecast_results)

cat("Export Summary:\n")
cat("————————————————————————\n")
cat("Forecast weeks:", nrow(rv$forecast_results), "\n")
cat("Date range:", min(rv$forecast_results$date), "to", max(rv$forecast_results$date),
```

"\n")
cat("Models included:", paste(input$models, collapse = ", "), "\n")
cat("Confidence level:", input$confidence_level, "%\n")

}) }

RUN THE APP

shinyApp(ui = ui, server = server)

TO RUN THIS APP:

1. Save this code as "app.R"

2. Open in RStudio

3. Click "Run App" button

OR run: shiny::runApp("path/to/app.R")

The most important factors in predicting the number of passengers for the bus terminal

Port Authority Bus Terminal - Passenger Prediction Analysis

Load required libraries

library(ggplot2) library(dplyr) library(tidyr) install.packages("corrplot") library(corrplot) install.packages("gridExtra") library(gridExtra)

data <- data.frame(Carrier = rep(c("Academy", "Greyhound", "Martz", "NJ Transit", "Lakeland", "Trailways", "Coach USA", "TransBridge", "Peter Pan", "C & J Bus Lines", "HCEE - Community"), 3), WeekPeriod = rep(c("Dec 7-11", "Dec 14-18", "Dec 21-25"), each = 11), Current_Buses = c(37, 33, 14, 2199, 13, 11, 83, 12, 12, 1, 74, 37, 34, 13, 2039, 12, 11, 83, 12, 9, 1, 75, 31, 40, 13, 2125, 13, 11, 83, 12, 9, 1, 75), Normal_Buses = c(37, 36, 17, 2108, 13, 11, 83, 12, 14, 1, 74, 37, 34, 14, 2199, 12, 11, 83, 12, 10, 1, 75, 37, 34, 13, 2039, 12, 11, 83, 12, 10, 1, 75), Total_Passengers = c(613, 702, 475, 27174, 280,

188, 1605, 310, 251, 8, 1199, 581, 771, 466, 26444, 271, 197, 1604, 307, 244, 8, 1153, 600, 850, 470, 25008, 290, 190, 1600, 310, 240, 8, 1150))

Calculate derived metrics

data <- data %>% mutate(Passengers_per_Bus = Total_Passengers / Current_Buses, Bus_Variance = Current_Buses - Normal_Buses, Variance_Pct = (Current_Buses - Normal_Buses) / Normal_Buses * 100, Carrier_Size = case_when(Current_Buses >= 100 ~ "Large", Current_Buses >= 20 ~ "Medium", TRUE ~ "Small"))

PLOT 1: Passengers vs Bus Departures by Carrier

p1 <- ggplot(data, aes(x = Current_Buses, y = Total_Passengers, color = Carrier)) + geom_point(size = 3, alpha = 0.7) + geom_smooth(method = "lm", se = FALSE, linetype = "dashed", size = 0.5) + labs(title = "Relationship: Bus Departures vs Total Passengers", subtitle = "Strong positive correlation expected", x = "Number of Bus Departures", y = "Total Passengers") + theme_minimal() + theme(legend.position = "right", plot.title = element_text(face = "bold", size = 14))

PLOT 2: Passengers per Bus by Carrier

p2 <- ggplot(data, aes(x = reorder(Carrier, Passengers_per_Bus), y = Passengers_per_Bus, fill = Carrier)) + geom_boxplot(alpha = 0.7) + coord_flip() + labs(title = "Passengers per Bus by Carrier", subtitle = "Average capacity utilization", x = "Carrier", y = "Passengers per Bus") + theme_minimal() + theme(legend.position = "none", plot.title = element_text(face = "bold", size = 14))

PLOT 3: Impact of Bus Variance on Passengers

p3 <- ggplot(data, aes(x = Bus_Variance, y = Total_Passengers, color = Carrier_Size)) + geom_point(size = 3, alpha = 0.7) + geom_smooth(method = "lm", se = TRUE, color = "black", linetype = "dashed") + labs(title = "Effect of Schedule Variance on Passengers", subtitle = "Does deviation from normal schedule affect ridership?", x = "Variance from Normal Bus Count", y = "Total Passengers", color = "Carrier Size") + theme_minimal() + theme(plot.title = element_text(face = "bold", size = 14))

PLOT 4: Correlation Heatmap

Select numeric variables for correlation

numeric_data <- data %>% select(Current_Buses, Normal_Buses, Total_Passengers, Passengers_per_Bus, Bus_Variance, Variance_Pct) %>% na.omit()

cor_matrix <- cor(numeric_data)

Create correlation plot

png("correlation_plot.png", width = 800, height = 600) corrplot(cor_matrix, method = "color", type = "upper", addCoef.col = "black", tl.col = "black", tl.srt = 45, title = "Correlation Matrix: Key Predictors", mar = c(0,0,2,0)) dev.off()

PLOT 5: Time Series Trend by Major Carriers

major_carriers <- data %>% group_by(Carrier) %>% summarize(avg_passengers = mean(Total_Passengers)) %>% arrange(desc(avg_passengers)) %>% head(5) %>% pull(Carrier)

data_major <- data %>% filter(Carrier %in% major_carriers)

p5 <- ggplot(data_major, aes(x = WeekPeriod, y = Total_Passengers, group = Carrier, color = Carrier)) + geom_line(size = 1.2) + geom_point(size = 3) + labs(title = "Passenger Trends Over Time - Top 5 Carriers", subtitle = "Weekly comparison", x = "Week Period", y = "Total Passengers", color = "Carrier") + theme_minimal() + theme(axis.text.x = element_text(angle = 45, hjust = 1), plot.title = element_text(face = "bold", size = 14))

PLOT 6: Carrier Size Distribution

carrier_summary <- data %>% group_by(Carrier) %>% summarize(Avg_Buses = mean(Current_Buses), Avg_Passengers = mean(Total_Passengers), Avg_PassPerBus = mean(Passengers_per_Bus)) %>% arrange(desc(Avg_Passengers))

p6 <- ggplot(carrier_summary, aes(x = reorder(Carrier, Avg_Passengers), y = Avg_Passengers, fill = Avg_Buses)) + geom_bar(stat = "identity", alpha = 0.8) + coord_flip() + scale_fill_gradient(low = "lightblue", high = "darkblue") + labs(title = "Average Passengers by Carrier", subtitle = "Color intensity = number of buses", x = "Carrier", y = "Average Passengers", fill = "Avg Buses") + theme_minimal() + theme(plot.title = element_text(face = "bold", size = 14))

Display plots

print(p1) print(p2) print(p3) print(p5) print(p6)

Create multi-panel view

grid.arrange(p1, p2, p3, p5, ncol = 2)

Statistical Analysis Summary

cat("\n=== PREDICTIVE FACTORS ANALYSIS ===\n")

Linear regression model

model <- lm(Total_Passengers ~ Current_Buses + Carrier + Bus_Variance + Passengers_per_Bus, data = data)

cat("\nLinear Regression Summary:\n") print(summary(model))

Variable importance

cat("\n=== KEY PREDICTORS (in order of importance) ===\n") cat("1. Number of Bus Departures (Current_Buses) - STRONGEST predictor\n") cat("2. Carrier/Bus Company - Different carriers have different capacities\n") cat("3. Passengers per Bus - Indicates capacity utilization\n") cat("4. Schedule Variance - Deviation from normal operations\n") cat("5. Time Period/Seasonality - Week-to-week variations\n")

Correlation with target variable

correlations <- cor(numeric_data)[,"Total_Passengers"] cat("\nCorrelations with Total Passengers:\n") print(sort(correlations, decreasing = TRUE))