```
#  Step 1: Set Up SQLite in Posit
install.packages("RSQLite")
install.packages("DBI")

library(RSQLite)
library(DBI)

# Step 2: Create a SQLite Database
# Create (or connect to) a SQLite database
conn <- dbConnect(SQLite(), "hospital_er_data.db")

#  Step 3: Create the Table
query <- "
CREATE TABLE Hospital_ER_Data_2 (
    patient_id INTEGER PRIMARY KEY,
    gender TEXT,
    age INTEGER,
    race TEXT,
    department TEXT,
    wait_time INTEGER,
    satisfaction_score INTEGER,
    admission_flag BOOLEAN,
    referral_department TEXT,
    visit_date DATE
);"

dbExecute(conn, query)

# Step 4: Verify the Table Structure
dbListTables(conn)  # Lists all tables in the database
dbGetQuery(conn, "PRAGMA table_info(Hospital_ER_Data_2);")

#  Step 2: Import the CSV Data into SQLite
# Load the CSV into R
getwd()  # Get current working directory

setwd("C:/Users/HP USER/Documents/HN-Personal/20_projects_challenge_14012025/challenge_3")
hospital_data <- read.csv("Hospital_ER_Data_2.csv", stringsAsFactors = FALSE)
getwd()  # Get current working directory
hospital_data <- read.csv("Hospital_ER_Data_2.csv", stringsAsFactors = FALSE)

list.files()
hospital_data <- read.csv("/cloud/project/Hospital_ER_Data_2.csv", stringsAsFactors = FALSE)
head(hospital_data)

# Step 2: Clean Column Names (Optional) - replace spaces with underscore _
library(dplyr)

colnames(hospital_data) <- c(
  "patient_id", "admission_date", "first_initial", "last_name", "gender",
  "age", "race", "referral_department", "admission_flag",
  "satisfaction_score", "wait_time", "patients_cm"
)

head(hospital_data)

#  Step 3: Connect to SQLite Database
library(DBI)
library(RSQLite)

# Connect to the database
conn <- dbConnect(RSQLite::SQLite(), "/cloud/project/hospital_er_data.db")

# Check if the connection is successful
```

```
dbListTables(conn)

# Step 1: Check Column Names in R
colnames(hospital_data)

install.packages("dplyr")  # Only run this if dplyr is not installed
library(dplyr)


#  Step 1: Drop the Extra Column
hospital_data <- hospital_data %>%
  select(-patients_cm)  # Remove the extra column

colnames(hospital_data)  # "patients_cm" should be gone

# Step 4: Insert Data into SQL Table
dbWriteTable(conn, "Hospital_ER_Data_2", hospital_data, append = TRUE, row.names = FALSE)


# Issue: Column Names Do Not Match SQL Table Schema - rename the CSV columns to match those in the
SQL table.
hospital_data <- hospital_data %>%
  rename(
    first_initial = first_initial,
    last_name = last_name,
    admission_date = admission_date,
    referral_department = referral_department,  # Confirm correct mapping
    gender = gender,
    age = age,
    race = race,
    admission_flag = admission_flag,
    satisfaction_score = satisfaction_score,
    wait_time = wait_time
  )

# Verify the updated column names
colnames(hospital_data)

# Retry Inserting Data
dbWriteTable(conn, "Hospital_ER_Data_2", hospital_data, append = TRUE, row.names = FALSE)


# Solving issue -  Issue: Extra Columns (first_initial, last_name) Not in SQL Table
# Remove Unnecessary Columns Before Inserting
#  Fix: Select Only Required Columns
library(dplyr)  # Ensure dplyr is loaded

# Select only columns that match the SQL table schema
hospital_data <- hospital_data %>%
  select(patient_id, visit_date, gender, age, race,
         referral_department, admission_flag,
         satisfaction_score, wait_time)

# Verify the updated column names
colnames(hospital_data)

# Retry Inserting Data
dbWriteTable(conn, "Hospital_ER_Data_2", hospital_data, append = TRUE, row.names = FALSE)

# Error: bad_weak_ptr in dbWriteTable
# Reconnect to the Database
# Load required package
library(DBI)

# Close any existing connection (if active)
```

```
dbDisconnect(conn)

# Reconnect to the database
conn <- dbConnect(RSQLite::SQLite(), "/cloud/project/hospital_er_data.db")

# Now retry inserting the data
dbWriteTable(conn, "Hospital_ER_Data_2", hospital_data, append = TRUE, row.names = FALSE)

# Error: datatype mismatch in dbWriteTable
#  Check Data Types in Your SQL Table
dbGetQuery(conn, "PRAGMA table_info(Hospital_ER_Data_2);")


# Check Data Types in hospital_data
str(hospital_data)

# Datatype mismatch -  Fixing the Data Types Before Inserting
hospital_data$patient_id <- as.integer(gsub("-", "", hospital_data$patient_id))

# Convert visit_date to Proper Date Format
hospital_data$visit_date <- as.Date(hospital_data$visit_date, format="%d-%m-%Y %H:%M")

# Trying to Insert Again
dbWriteTable(conn, "Hospital_ER_Data_2", hospital_data, append = TRUE, row.names = FALSE)

# Verify Data Import
dbGetQuery(conn, "SELECT COUNT(*) FROM Hospital_ER_Data_2;")  # Check total rows
dbGetQuery(conn, "SELECT * FROM Hospital_ER_Data_2 LIMIT 10;")  # Preview data

# Step 1: Fix visit_date Format
# Check the Current Format
dbGetQuery(conn, "SELECT DISTINCT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")

# Convert Serial Dates to Actual Dates
# Use SQLite’s DATE() function:
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(visit_date, '1899-12-30');")
dbGetQuery(conn, "SELECT DISTINCT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(visit_date, 'unixepoch');")
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(visit_date, '1899-12-30');")
dbGetQuery(conn, "SELECT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")

# Convert visit_date Properly
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(visit_date - 25569, '1970-01-
01');")
dbGetQuery(conn, "SELECT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")

# Use JULIANDAY() Instead
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(JULIANDAY(visit_date,
'-2415018.5'));")
dbGetQuery(conn, "SELECT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")

# Alternative Fix: Use DATE() with Correct Offset
dbExecute(conn, "UPDATE Hospital_ER_Data_2 SET visit_date = DATE(visit_date + 24107, '1970-01-
01');")
dbGetQuery(conn, "SELECT visit_date FROM Hospital_ER_Data_2 LIMIT 10;")

# STarting afresh with a new table
# Drop the existing table
dbExecute(conn, "DROP TABLE IF EXISTS Hospital_ER_Data_2;")

# Create a new table with correct column names and data types
dbExecute(conn, "
CREATE TABLE hospital_data (
    patient_id INTEGER,
    admission_date TEXT,
```

```
    first_initial TEXT,
    last_name TEXT,
    gender TEXT,
    age INTEGER,
    race TEXT,
    referral_department TEXT,
    admission_flag INTEGER,
    satisfaction_score INTEGER,
    wait_time INTEGER,
    patients_cm TEXT
);
")

# Insert the data from hospital_data into the newly created table
dbGetQuery(conn, "SELECT * FROM hospital_data LIMIT 10;")


# Error - meaning data base connection maybe invalidated
conn <- dbConnect(RSQLite::SQLite(), "hospital_db.sqlite")

# Verify the Connection
dbListTables(conn)

# Check If the Table Exists
dbGetQuery(conn, "SELECT name FROM sqlite_master WHERE type='table';")

#  Check if the database file exists
file.exists("hospital_db.sqlite")

#  Recreate the Table
dbExecute(conn, "
CREATE TABLE hospital_data (
    patient_id INTEGER,
    admission_date TEXT,
    first_initial TEXT,
    last_name TEXT,
    gender TEXT,
    age INTEGER,
    race TEXT,
    referral_department TEXT,
    admission_flag INTEGER,
    satisfaction_score INTEGER,
    wait_time INTEGER,
    patients_cm TEXT
);
")

dbListTables(conn)

# Reinsert the Data
dbWriteTable(conn, "hospital_data", hospital_data, append = TRUE, row.names = FALSE)

# Fixing issue with visiting_date not found
# Add visit_date Column to the Table
dbExecute(conn, "ALTER TABLE hospital_data ADD COLUMN visit_date TEXT;")
dbGetQuery(conn, "PRAGMA table_info(hospital_data);")


# Reinsert the Data
dbWriteTable(conn, "hospital_data", hospital_data, append = TRUE, row.names = FALSE)

dbGetQuery(conn, "SELECT * FROM hospital_data LIMIT 10;")

# Fixing issues with visit_date column having numerical values
#  Convert visit_date to a Proper Date Format
```

```
dbExecute(conn, "UPDATE hospital_data SET visit_date = DATE(visit_date, '1899-12-30');")
dbGetQuery(conn, "SELECT visit_date FROM hospital_data LIMIT 10;")

# Issues with visit_date returning with NA
# Checking the Data Type of visit_date
dbGetQuery(conn, "SELECT typeof(visit_date) FROM hospital_data LIMIT 10;")

# Checking the Raw Data in R
summary(hospital_data$visit_date)
head(hospital_data$visit_date, 20)

# Check if visit_date is NULL in SQLite
dbGetQuery(conn, "SELECT COUNT(*) AS null_count FROM hospital_data WHERE visit_date IS NULL;")

# Fixing this issue - All 9216 rows returned as null
# Drop the existing table
dbExecute(conn, "DROP TABLE IF EXISTS hospital_data;")

# Recreate the table
dbExecute(conn, "
CREATE TABLE hospital_data (
    patient_id INTEGER,
    admission_date TEXT,
    first_initial TEXT,
    last_name TEXT,
    gender TEXT,
    age INTEGER,
    race TEXT,
    referral_department TEXT,
    admission_flag INTEGER,
    satisfaction_score INTEGER,
    wait_time INTEGER,
    patients_cm TEXT,
    visit_date TEXT
);
")

# Reinsert data with correct visit_date format
dbWriteTable(conn, "hospital_data", hospital_data, row.names = FALSE, append = TRUE)

# Verify the data
dbGetQuery(conn, "SELECT visit_date FROM hospital_data LIMIT 10;")

# Trying to fix the issue using strftime
dbGetQuery(conn, "SELECT strftime('%Y-%m-%d', visit_date, 'unixepoch') AS visit_date FROM
hospital_data LIMIT 10;")

# Trying to fix the reason why all the dates as same by removing the unixepoch from the strftime
code
dbGetQuery(conn, "SELECT strftime('%Y-%m-%d', visit_date) AS visit_date FROM hospital_data LIMIT
10;")

# Trying to fix another issue with the visit_date returning as Julian Day numbers
dbGetQuery(conn, "SELECT date(visit_date - 2415019, 'unixepoch') AS visit_date FROM hospital_data
LIMIT 10;")
dbGetQuery(conn, "SELECT strftime('%Y-%m-%d', visit_date - 2415019) AS visit_date FROM
hospital_data LIMIT 10;")

# STill trying to correct the visit_date issue
dbGetQuery(conn, "SELECT DATE('1899-12-30', visit_date || ' days') AS visit_date FROM
hospital_data LIMIT 10;")

# Fixing the discrepancy in dates btw the raw date dispalyed in R and that in SQLite
dbGetQuery(conn, "SELECT DATE('1970-01-01', visit_date || ' days') AS visit_date FROM
hospital_data LIMIT 10;")
```

```
# Restarting the Connection after session closed to fix bad_weak_ptr error
# Disconnect existing connection if still open
if (dbIsValid(conn)) {
  dbDisconnect(conn)
}

# Reconnect to the database
conn <- dbConnect(RSQLite::SQLite(), "hospital_db.sqlite")



#  Investigating the department Column
# Checking for Missing Values in referral_department
# Count Missing Values
dbGetQuery(conn, "SELECT COUNT(*) AS missing_count FROM hospital_data WHERE referral_department IS
NULL OR referral_department = 'None';")

# Checking Distribution of Departments
dbGetQuery(conn, "SELECT referral_department, COUNT(*) AS count FROM hospital_data GROUP BY
referral_department ORDER BY count DESC;")

# Deciding on Missing Data Handling - Checking If Missing Values Are Random or Systematic
dbGetQuery(conn, "
SELECT gender, COUNT(*) AS count
FROM hospital_data
WHERE referral_department IS NULL OR referral_department = 'None'
GROUP BY gender;
")


# Assessing If Imputation Is Needed and what kind - Check the admission_flag distribution for
missing vs. non-missing cases
dbGetQuery(conn, "
SELECT admission_flag, COUNT(*)
FROM hospital_data
WHERE referral_department IS NULL OR referral_department = 'None'
GROUP BY admission_flag;
")


# How to Handle Missing referral_department Values -
# Compare Patient Profiles - Compare Age Distribution
dbGetQuery(conn, "
SELECT
  'General Practice' AS category,
  AVG(age) AS avg_age,
  MIN(age) AS min_age,
  MAX(age) AS max_age
FROM hospital_data
WHERE referral_department = 'General Practice'
UNION ALL
SELECT
  'Missing Referral' AS category,
  AVG(age) AS avg_age,
  MIN(age) AS min_age,
  MAX(age) AS max_age
FROM hospital_data
WHERE referral_department IS NULL OR referral_department = 'None';
")


#  Compare Gender Distribution
dbGetQuery(conn, "
SELECT
```

```
    referral_department,
    gender,
    COUNT(*) AS count
  FROM hospital_data
  WHERE referral_department = 'General Practice'
     OR referral_department IS NULL
     OR referral_department = 'None'
  GROUP BY referral_department, gender
  ORDER BY referral_department, gender;
  ")

  # Compare Admission Flag Distribution
  dbGetQuery(conn, "
  SELECT
    referral_department,
    admission_flag,
    COUNT(*) AS count
  FROM hospital_data
  WHERE referral_department = 'General Practice'
     OR referral_department IS NULL
     OR referral_department = 'None'
  GROUP BY referral_department, admission_flag
  ORDER BY referral_department, admission_flag;
  ")


  # Impute Most Common Value (General Practice)
  dbExecute(conn, "
  UPDATE hospital_data
  SET referral_department = 'General Practice'
  WHERE referral_department IS NULL OR referral_department = 'None';
  ")

  # verify the changes
  dbGetQuery(conn, "
  SELECT referral_department, COUNT(*)
  FROM hospital_data
  GROUP BY referral_department
  ORDER BY COUNT(*) DESC;
  ")


  # Step 3: Basic Data Quality Checks
  # Are there duplicate rows
  dbGetQuery(conn, "
  SELECT COUNT(*) AS total_rows,
         COUNT(DISTINCT age || '-' || gender || '-' || referral_department || '-' || admission_flag
  || '-' || visit_date) AS unique_rows,
         COUNT(*) - COUNT(DISTINCT age || '-' || gender || '-' || referral_department || '-' ||
  admission_flag || '-' || visit_date) AS duplicate_count
  FROM hospital_data;
  ")


  #  Checking for Outliers in Key Numeric Fields - Check Age Distribution
  dbGetQuery(conn, "
  SELECT
      MIN(age) AS min_age,
      MAX(age) AS max_age,
      AVG(age) AS avg_age,
      PERCENTILE_CONT(0.25) WITHIN GROUP (ORDER BY age) AS Q1,
      PERCENTILE_CONT(0.50) WITHIN GROUP (ORDER BY age) AS median,
      PERCENTILE_CONT(0.75) WITHIN GROUP (ORDER BY age) AS Q3
  FROM hospital_data;
  ")
```

```
# List the tables in the database
dbListTables(conn)

dbGetQuery(conn, 'SELECT name FROM sqlite_master WHERE type="table";')

# Assuming the ages are not distinct but they may be duplicate ages - using R maybe an alternative
for  calculate the quartiles (Q1, median, Q3) using SQLite:

library(RSQLite)

# Connect to the SQLite database
conn <- dbConnect(RSQLite::SQLite(), "hospital_data.db")

dbGetQuery(conn, "SELECT name FROM sqlite_master WHERE type='table';")
dbWriteTable(conn, "hospital_data", hospital_data, overwrite = TRUE)


# Query the data
data <- dbGetQuery(conn, "SELECT age FROM hospital_data")

# Calculate the quartiles
quartiles <- quantile(data$age, probs = c(0.25, 0.5, 0.75))

# Print the quartiles
print(quartiles)


# Identifying Any Remaining Missing Values
dbGetQuery(conn, "
SELECT
    SUM(CASE WHEN age IS NULL THEN 1 ELSE 0 END) AS missing_age,
    SUM(CASE WHEN gender IS NULL THEN 1 ELSE 0 END) AS missing_gender,
    SUM(CASE WHEN admission_flag IS NULL THEN 1 ELSE 0 END) AS missing_admission_flag,
    SUM(CASE WHEN visit_date IS NULL THEN 1 ELSE 0 END) AS missing_visit_date
FROM hospital_data;
")

# Handling Duplicate Rows
dbExecute(conn, "DROP TABLE IF EXISTS temp_rows")
dbExecute(conn, "CREATE TEMP TABLE temp_rows AS SELECT MIN(ROWID) AS min_rowid FROM hospital_data
GROUP BY age, gender, referral_department, admission_flag, visit_date")
temp_rows <- dbGetQuery(conn, "SELECT * FROM temp_rows")



# DELETE FROM hospital_data
dbExecute(conn, "DELETE FROM hospital_data WHERE ROWID NOT IN (SELECT min_rowid FROM temp_rows)")

# DROP TABLE temp_rows;
dbExecute(conn, "DROP TABLE IF EXISTS temp_rows")



# Perform Exploratory Data Analysis (EDA)
# Install and load necessary libraries
install.packages("dplyr")
install.packages("ggplot2")
install.packages("RSQLite")
install.packages("corrplot")


# Load ggplot2, corrplot, dplyr library
suppressPackageStartupMessages(library(ggplot2))
library(corrplot)
```

```
library(dplyr)
library(RColorBrewer)



# Connect to the SQLite database
conn <- dbConnect(RSQLite::SQLite(), "hospital_data.db")

# Load the hospital_data table
hospital_data <- dbReadTable(conn, "hospital_data")

# Summary statistics for numerical columns
summary(hospital_data[, sapply(hospital_data, is.numeric)])

# Summary statistics for categorical columns
summary(hospital_data[, sapply(hospital_data, is.factor)])

# Creating Visualizations to explore the data
# Check column names
column_names <- capture.output(colnames(hospital_data))
cat("Column Names:\n")
cat(column_names, sep = "\n")
cat("\n")

# Calculate the wait times by referral department
wait_times_by_referral <- hospital_data %>%
  group_by(referral_department) %>%
  summarise(wait_time = sum(wait_time))

# Print the results
cat("Wait Times by Referral Department:\n")
print(wait_times_by_referral)

# Create a bar chart to visualize the results
ggplot(wait_times_by_referral, aes(x = referral_department, y = wait_time)) +
  geom_bar(stat = "identity") +
  labs(title = "Wait Times by Referral Department", x = "Referral Department", y = "Wait Time")

# Save the plot
ggsave("wait_times_by_referral_department.png")


# Calculate the average wait times for each department
avg_wait_times <- hospital_data %>%
  group_by(referral_department) %>%
  summarise(avg_wait_time = mean(wait_time))

# Print the results
cat("Average Wait Times by Referral Department:\n")
print(avg_wait_times)

# Create a bar chart to visualize the results
ggplot(avg_wait_times, aes(x = referral_department, y = avg_wait_time)) +
  geom_bar(stat = "identity") +
  labs(title = "Average Wait Times by Referral Department", x = "Referral Department", y =
"Average Wait Time")

# Save the plot as a PNG file
ggsave("avg_wait_times_by_department.png")


# Heatmap: Correlation between variables
# Remove NA/NaN/Inf values from the correlation matrix
cor_matrix <- cor(hospital_data[, sapply(hospital_data, is.numeric)], use =
"pairwise.complete.obs")
```

```
## Create the correlation matrix
cor_matrix <- cor(hospital_data[, sapply(hospital_data, is.numeric)], use =
"pairwise.complete.obs")


# Create the heatmap
heatmap(cor_matrix, main = "Correlation Matrix")


## Create the heatmap and save it as a PNG file
png("correlation_heatmap.png", width = 800, height = 600)
corrplot(cor_matrix, method = "color", type = "upper", order = "hclust", p.mat = cor_matrix,
sig.level = 0.05)
dev.off()

# Save the plot
ggsave("correlation_matrix.png")


# Box plot: Wait times by department
box_plot <- ggplot(hospital_data, aes(x = referral_department, y = wait_time)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Wait Times by Department", x = "Department", y = "Wait Time") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
        axis.title = element_text(size = 14),
        axis.text = element_text(size = 12))

print(box_plot)
ggsave("box_plot_wait_times_by_department.png", width = 8, height = 6, dpi = 300)

# Create age_group column
hospital_data$age_group <- cut(hospital_data$age,
                               breaks = c(0, 18, 30, 50, 65, Inf),
                               labels = c("0-17", "18-29", "30-49", "50-64", "65+"))


# Heatmap: Wait times by patient age group
heatmap <- ggplot(hospital_data, aes(x = age_group, y = wait_time, fill = age_group)) +
  geom_boxplot() +
  labs(title = "Wait Times by Patient Age Group", x = "Age Group", y = "Wait Time") +
  scale_fill_brewer(palette = "Dark2") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
        axis.title = element_text(size = 14),
        axis.text = element_text(size = 12))

print(heatmap)
ggsave("heatmap_wait_times_by_age_group.png", width = 8, height = 6, dpi = 300)


#Check for missing values: Use the summary() function to check for missing values in the wait_time
and satisfaction_score columns.
summary(hospital_data$wait_time)
summary(hospital_data$satisfaction_score)


# Remove missing values: Use the na.omit() function to remove rows with missing values.
hospital_data <- na.omit(hospital_data)

# Check for outliers: Use the boxplot() function to check for outliers in the wait_time and
satisfaction_score columns.
boxplot(hospital_data$wait_time)
```

```
boxplot(hospital_data$satisfaction_score)

# Remove outliers: Use the filter() function from the dplyr package to remove outliers.
library(dplyr)
hospital_data <- hospital_data[!is.na(hospital_data$satisfaction_score), ]

nrow(hospital_data)

# Update the plot: After removing missing values and outliers, update the plot using the same code
# Scatter plot: Wait times vs satisfaction scores
scatter_plot <- ggplot(hospital_data, aes(x = wait_time, y = satisfaction_score, color =
referral_department)) +
  geom_point() +
  labs(title = "Wait Times vs Satisfaction Scores", x = "Wait Time", y = "Satisfaction Score") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
        axis.title = element_text(size = 14),
        axis.text = element_text(size = 12)) +
  scale_color_brewer(palette = "Dark2")

print(scatter_plot)
ggsave("scatter_plot_wait_times_vs_satisfaction_scores.png", width = 8, height = 6, dpi = 300)


# Go back to the original dataframe
hospital_data_original <- hospital_data

table(hospital_data_original$race)

head(hospital_data_original$race)
tail(hospital_data_original$race)



# Summary of wait times by race
summary(hospital_data_original$wait_time ~ hospital_data_original$race)

# Bar plot of wait times by race
print("Before plot")
plot <- ggplot(hospital_data, aes(x = race, y = wait_time)) +
  stat_summary(fun = mean, geom = "bar") +
  labs(title = "Mean Wait Times by Race", x = "Race", y = "Mean Wait Time") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
        axis.title = element_text(size = 14),
        axis.text = element_text(size = 12))

library(dplyr)

mean_wait_times <- hospital_data %>%
  group_by(race) %>%
  summarise(mean_wait_time = mean(wait_time))

print(mean_wait_times)


# Save the plot
ggsave("wait_times_by_race.png", plot = plot, width = 8, height = 6, dpi = 300)

# Load the dplyr library
library(dplyr)

# Group the data by race and referral_department, and calculate the mean wait time
mean_wait_times_by_race_and_dept <- hospital_data %>%
  group_by(race, referral_department) %>%
```

```
    summarise(mean_wait_time = mean(wait_time))

  # Print the results
  print(mean_wait_times_by_race_and_dept)

  # Create a bar plot of mean wait times by race and referral department
  plot <- ggplot(mean_wait_times_by_race_and_dept, aes(x = referral_department, y = mean_wait_time,
  fill = race)) +
    geom_col() +
    labs(title = "Mean Wait Times by Race and Referral Department", x = "Referral Department", y =
  "Mean Wait Time") +
    theme_classic() +
    theme(plot.title = element_text(hjust = 0.5, size = 16, face = "bold"),
          axis.title = element_text(size = 14),
          axis.text = element_text(size = 12),
          legend.position = "bottom")

  # Print the plot
  print(plot)

  # Save the plot as a PNG file
  ggsave("mean_wait_times_by_race_and_dept.png", plot = plot, width = 12, height = 8, dpi = 300)


  # Visualize Distributions

  # Load necessary libraries
  library(ggplot2)

  # Create histograms for wait times and ages
  ggplot(hospital_data, aes(x = wait_time)) +
    geom_histogram(bins = 50, color = "black", fill = "lightblue") +
    labs(title = "Distribution of Wait Times", x = "Wait Time (hours)", y = "Frequency") +
    theme_classic()

  ggplot(hospital_data, aes(x = age)) +
    geom_histogram(bins = 50, color = "black", fill = "lightgreen") +
    labs(title = "Distribution of Ages", x = "Age (years)", y = "Frequency") +
    theme_classic()

  # Save the plots
  ggsave("wait_time_distribution.png", width = 8, height = 6, dpi = 300)
  ggsave("age_distribution.png", width = 8, height = 6, dpi = 300)


  # Examine Correlations

  # Calculate correlations
  correlations <- cor(hospital_data[, c("wait_time", "age", "satisfaction_score")])

  # Print the correlation matrix
  print(correlations)

  # Remove NA/NaN/Inf values
  correlations[is.na(correlations)] <- 0

  # Create a PNG file
  png("correlation_heatmap.png", width = 800, height = 600)

  # Create a heatmap
  heatmap(correlations, main = "Correlation Matrix", col = blues9)

  # Close the PNG file
  dev.off()
```

```
# Check for missing values
sum(is.na(hospital_data$satisfaction_score))

# Check for infinite values
sum(is.infinite(hospital_data$satisfaction_score))


# Analyze Satisfaction Scores

# Create a histogram for satisfaction scores
ggplot(hospital_data, aes(x = satisfaction_score)) +
  geom_histogram(bins = 50, color = "black", fill = "lightyellow") +
  labs(title = "Distribution of Satisfaction Scores", x = "Satisfaction Score", y = "Frequency") +
  theme_classic()

# Save the plot
ggsave("satisfaction_score_distribution.png", width = 8, height = 6, dpi = 300)


# Investigate Outliers

# Identify outliers in wait times
outliers <- hospital_data[hospital_data$wait_time > 100, ]

# Print the outliers
print(outliers)



# Perform Clustering Analysis

# Perform k-means clustering
set.seed(123)
clusters <- kmeans(hospital_data[, c("wait_time", "age")], centers = 3)

# Print the cluster assignments
print(clusters$cluster)

# Create a scatter plot with clusters
ggplot(hospital_data, aes(x = age, y = wait_time, color = factor(clusters$cluster))) +
  geom_point() +
  labs(title = "K-Means Clustering", x = "Age (years)", y = "Wait Time (hours)") +
  theme_classic()

# Save the plot
ggsave("kmeans_clustering.png", width = 8, height = 6, dpi = 300)


# Examine Temporal Trends

# Create a time series plot for wait times
ggplot(hospital_data, aes(x = admission_date, y = wait_time)) +
  geom_point() +
  geom_smooth() +
  labs(title = "Temporal Trend of Wait Times", x = "Admission Date", y = "Wait Time (hours)") +
  theme_classic()

# Save the plot
ggsave("temporal_trend_wait_times.png", width = 12, height = 6, dpi = 300)


# Exploring Relationships Between Wait Times and Other Variables

# Correlation Analysis
# Calculate correlations
```

```
correlations <- cor(hospital_data[, c("wait_time", "age")])

# Print the correlation matrix
print("Correlation Matrix:")
print(correlations)

# Visualization
# Load necessary libraries
library(ggplot2)

# Create a scatter plot
print("Scatter Plot: Relationship Between Wait Times and Age")
ggplot(hospital_data, aes(x = age, y = wait_time)) +
  geom_point(color = "blue", alpha = 0.5) +
  labs(title = "Relationship Between Wait Times and Age", x = "Age (years)", y = "Wait Time
(hours)") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5))

# Save the plot
ggsave("wait_time_vs_age.png", width = 8, height = 6, dpi = 300)

# Let's also explore the relationship between wait times and referral department:

# Create a box plot
print("Box Plot: Relationship Between Wait Times and Referral Department")
ggplot(hospital_data, aes(x = referral_department, y = wait_time)) +
  geom_boxplot(fill = "lightblue") +
  labs(title = "Relationship Between Wait Times and Referral Department", x = "Referral
Department", y = "Wait Time (hours)") +
  theme_classic() +
  theme(plot.title = element_text(hjust = 0.5))

# Save the plot
ggsave("wait_time_vs_referral_department.png", width = 12, height = 6, dpi = 300)
```