Table of Contents

Python Part

Q1: Implementation of the transport network

```
import networkx as netx
import matplotlib.pyplot as plot

# Create a graph
G = netx.Graph()

# Define updated lines and their stations for New York City
lines = {
    "1": ["South Ferry", "Chambers St", "14 St", "Times Sq-42 St", "59 St-Columbus
Circle"],
    "2": ["Flatbush Av-Brooklyn College", "Atlantic Av-Barclays Center", "14 St", "Times
Sq-42 St", "96 St"],
    "3": ["New Lots Av", "Atlantic Av-Barclays Center", "14 St", "Times Sq-42 St", "96
St"],
    "4": ["Crown Hts-Utica Av", "Atlantic Av-Barclays Center", "14 St-Union Sq", "Grand
Central-42 St", "125 St"],
    "5": ["Flatbush Av-Brooklyn College", "Atlantic Av-Barclays Center", "14 St-Union
Sq", "Grand Central-42 St", "125 St"],
    "6": ["Brooklyn Bridge-City Hall", "14 St-Union Sq", "Grand Central-42 St", "59 St",
"125 St"],
}

# Add edges and nodes for each line
for line, stations in lines.items():
    for i in range(len(stations) - 1):
        G.add_edge(stations[i], stations[i + 1], line=line)

# Identify transfer stations (nodes shared by multiple lines)
transfer_stations = {station for station in G.nodes if sum(station in line_stations for
line_stations in lines.values()) > 1}

# Adding additional transfer connections for graph connectivity
additional_transfers = [
    ("14 St", "14 St-Union Sq"),
]
for station1, station2 in additional_transfers:
    G.add_edge(station1, station2, line="Transfer")

# Check if graph is connected
is_connected = netx.is_connected(G)
print("Graph is connected:", is_connected)


# Plot for question 1
plot.figure(figsize=(12, 12))
pos = netx.spring_layout(G, seed=42)  # Set seed for consistent layout
```

```
netx.draw(G, pos, with_labels=True, node_size=300, node_color="lightgray")
plot.show()
```

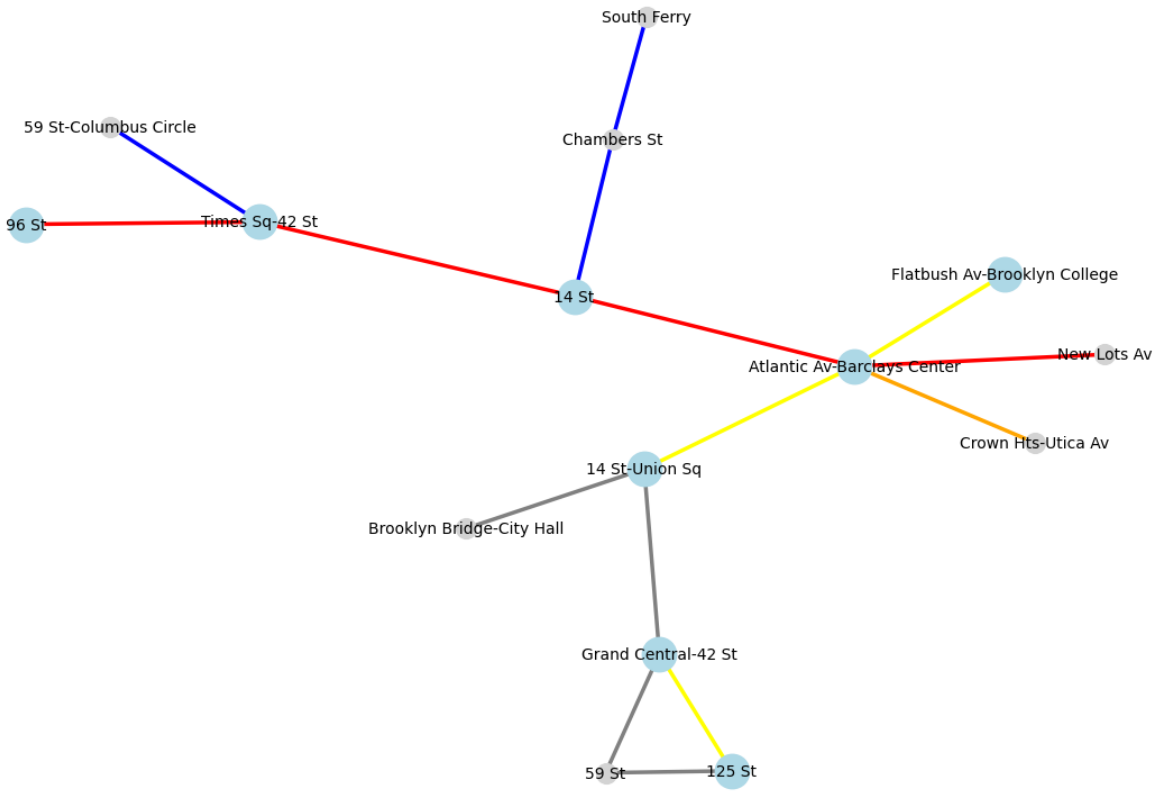
Plot:

Q2: Visualizing the network

```
# Updated color map for new lines
color_map = {
    "1": "blue",
    "2": "green",
    "3": "red",
    "4": "orange",
    "5": "yellow",
    "6": "gray",
}

# Draw network
plot.figure(figsize=(12, 12))
pos = netx.spring_layout(G, seed=42)  # Set seed for consistent layout

# Draw each line with colors
for line, color in color_map.items():
    edges = [(u, v) for u, v, d in G.edges(data=True) if d["line"] == line]
```

```
    netx.draw_networkx_edges(G, pos, edgelist=edges, edge_color=color, width=2,
label=line)

# Draw transfer stations in a larger size and different color
netx.draw_networkx_nodes(G, pos, nodelist=transfer_stations, node_color="lightblue",
node_size=300, label="Transfer Stations")

# Draw other stations
netx.draw_networkx_nodes(G, pos, nodelist=set(G.nodes) - transfer_stations,
node_color="lightgray", node_size=100)

# Draw labels
netx.draw_networkx_labels(G, pos, font_size=8)

# Plot for question 2
plot.figure(figsize=(12, 12))
pos = netx.spring_layout(G, seed=42)  # Set seed for consistent layout
netx.draw(G, pos, with_labels=True, node_size=300, node_color="lightgray")
plot.show()
```

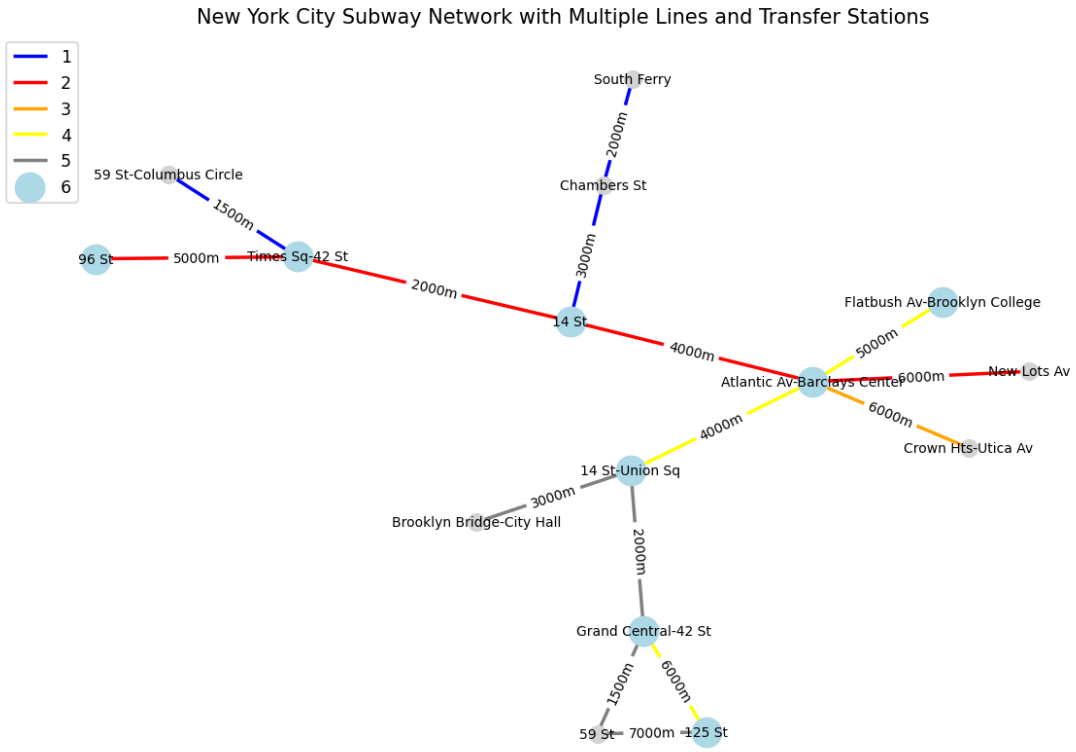
Plot:



Q4: Displaying attributes for lines

```
# Set edge weights (distances in meters) for all pairs
distances = {
    ("South Ferry", "Chambers St"): 2000,
    ("Chambers St", "14 St"): 3000,
    ("14 St", "Times Sq-42 St"): 2000,
    ("Times Sq-42 St", "59 St-Columbus Circle"): 1500,
    ("Flatbush Av-Brooklyn College", "Atlantic Av-Barclays Center"): 5000,
    ("Atlantic Av-Barclays Center", "14 St"): 4000,
    ("14 St-Union Sq", "Grand Central-42 St"): 2000,
    ("Grand Central-42 St", "125 St"): 6000,
    ("Brooklyn Bridge-City Hall", "14 St-Union Sq"): 3000,
    ("Inwood-207 St", "59 St-Columbus Circle"): 8000,
    ("59 St-Columbus Circle", "125 St"): 7000,
    ("Flatbush Av-Brooklyn College", "Atlantic Av-Barclays Center"): 5000,
    ("Atlantic Av-Barclays Center", "14 St"): 4000,
    ("14 St", "Times Sq-42 St"): 2000,
    ("Times Sq-42 St", "96 St"): 5000,
    ("New Lots Av", "Atlantic Av-Barclays Center"): 6000,
    ("Crown Hts-Utica Av", "Atlantic Av-Barclays Center"): 6000,
    ("125 St", "59 St"): 7000,
    ("Grand Central-42 St", "59 St"): 1500,
    ("14 St-Union Sq", "Atlantic Av-Barclays Center"): 4000,
}

# Update graph with distances
for (u, v), distance in distances.items():
    if G.has_edge(u, v):
        G[u][v]['distance'] = distance

# Draw edges with labels (distances)
edge_labels = {(u, v): f"{d['distance']}m" for u, v, d in G.edges(data=True) if
'distance' in d}
netx.draw_networkx_edge_labels(G, pos, edge_labels=edge_labels, font_size=8)

# Add legend and title
plot.legend(color_map.keys(), loc='upper left', fontsize=10)
plot.title("New York City Subway Network with Multiple Lines and Transfer Stations")
plot.axis("off")
plot.show()
```

Plot



Q5: Improvements:

Based on the generated map, here are two suggested improvements:

- Implementing a way to show directions and align each station relative to one another by direction. This will make the map useable by the intended users in real life situations.
- Showing the distance relative to one another based on the length of each edge. If the distance is 3000m the length of the edge should be smaller than the one with 5000m. This will improve the readers sense of distance.

Data Source:

The data used in this plot is taken from the MTA Live website, which provides real-time information about the New York City subway system.

R Part

Question # 1

Data pre-processing and descriptive statistics

1. Creating two data frames:

To create two data frames using dplyr, we first loaded the required libraries and the dataset, then filter the data to create subsets for "Banking and credit industry fraud" and "All charity fraud".

```
# Import required libraries
library(readxl)
library(dplyr)
library(ggplot2)
library(tidyr)
library(scales)

# Load the specified sheet from the Excel file, skipping the initial 8 rows
fraud_dataset <- read_excel("D:/Code/HF/24-11-2024/cw_r.xlsx", sheet = "Table 3d", skip =
8)

# Examine the structure of the loaded data for understanding its format
str(fraud_dataset)

# Update column names for better readability and usage
colnames(fraud_dataset) <- c("Fraud_Type", "Year_2012_2013", "Year_2013_2014",
"Year_2014_2015", "Year_2015_2016",
                             "Year_2016_2017", "Year_2017_2018", "Year_2018_2019",
"Year_2019_2020", "Year_2020_2021",
                             "Year_2021_2022", "Percent_Change")

# Create subsets of data for different types of fraud
# Extract data related to fraud in the banking and credit industry
banking_fraud_data <- fraud_dataset %>%
  filter(Fraud_Type == "Banking and credit industry fraud") %>%
  select(-Fraud_Type, -Percent_Change) %>%
  pivot_longer(cols = starts_with("Year"), names_to = "Year", values_to = "Fraud_Count")

# Extract data related to fraud in charities
charity_fraud_data <- fraud_dataset %>%
  filter(Fraud_Type == "All charity fraud") %>%
  select(-Fraud_Type, -Percent_Change) %>%
  pivot_longer(cols = starts_with("Year"), names_to = "Year", values_to = "Fraud_Count")

# Check the structure of the subsets for accuracy
str(banking_fraud_data)
str(charity_fraud_data)
```

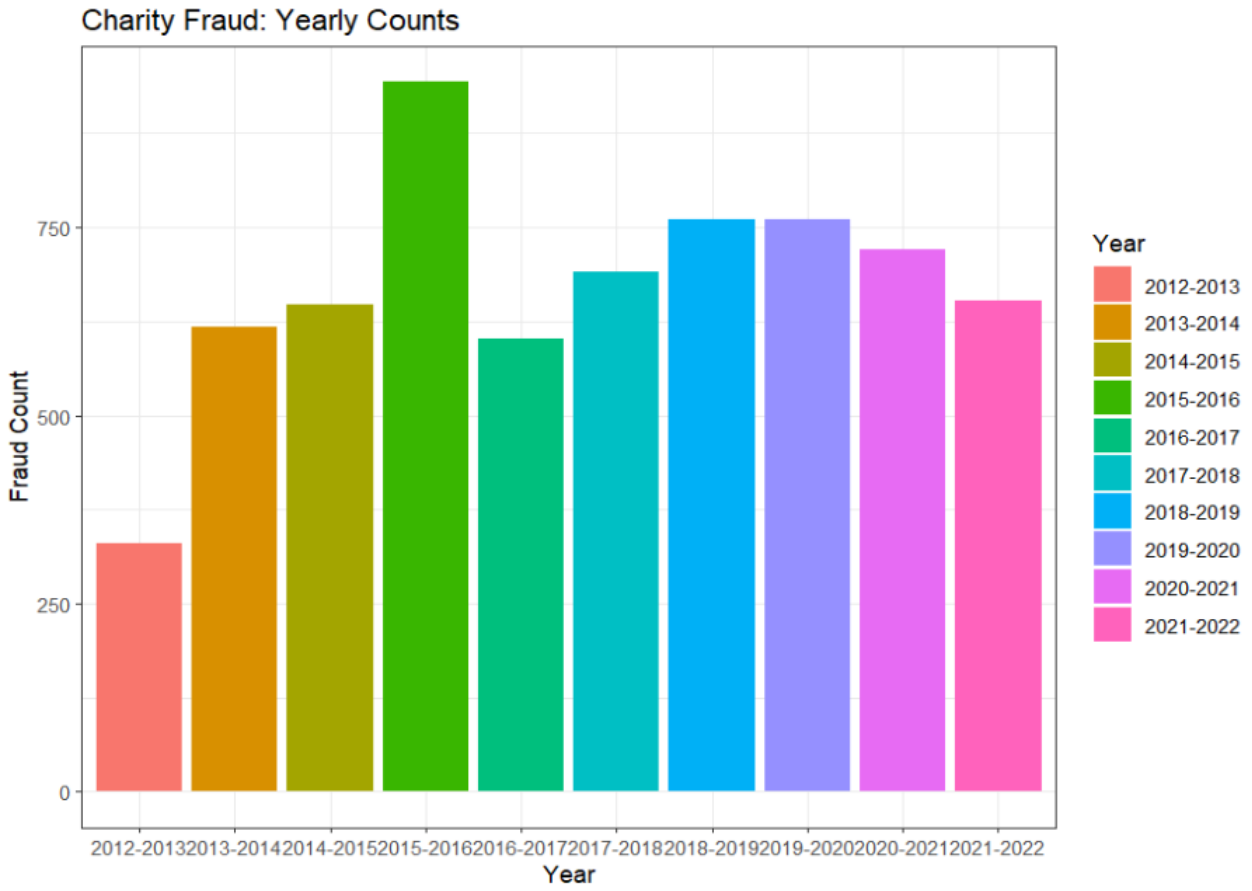
```r
# Preview the first few entries in each subset
head(banking_fraud_data)
head(charity_fraud_data)

# Arrange data by year to ensure proper order for visualization
banking_fraud_data <- banking_fraud_data[order(banking_fraud_data$Year), ]
charity_fraud_data <- charity_fraud_data[order(charity_fraud_data$Year), ]
```

2. Statistical distribution of these datasets:

To visualize the dispersion and central tendency of the two data frames, ggplot2 was used to create bar charts.

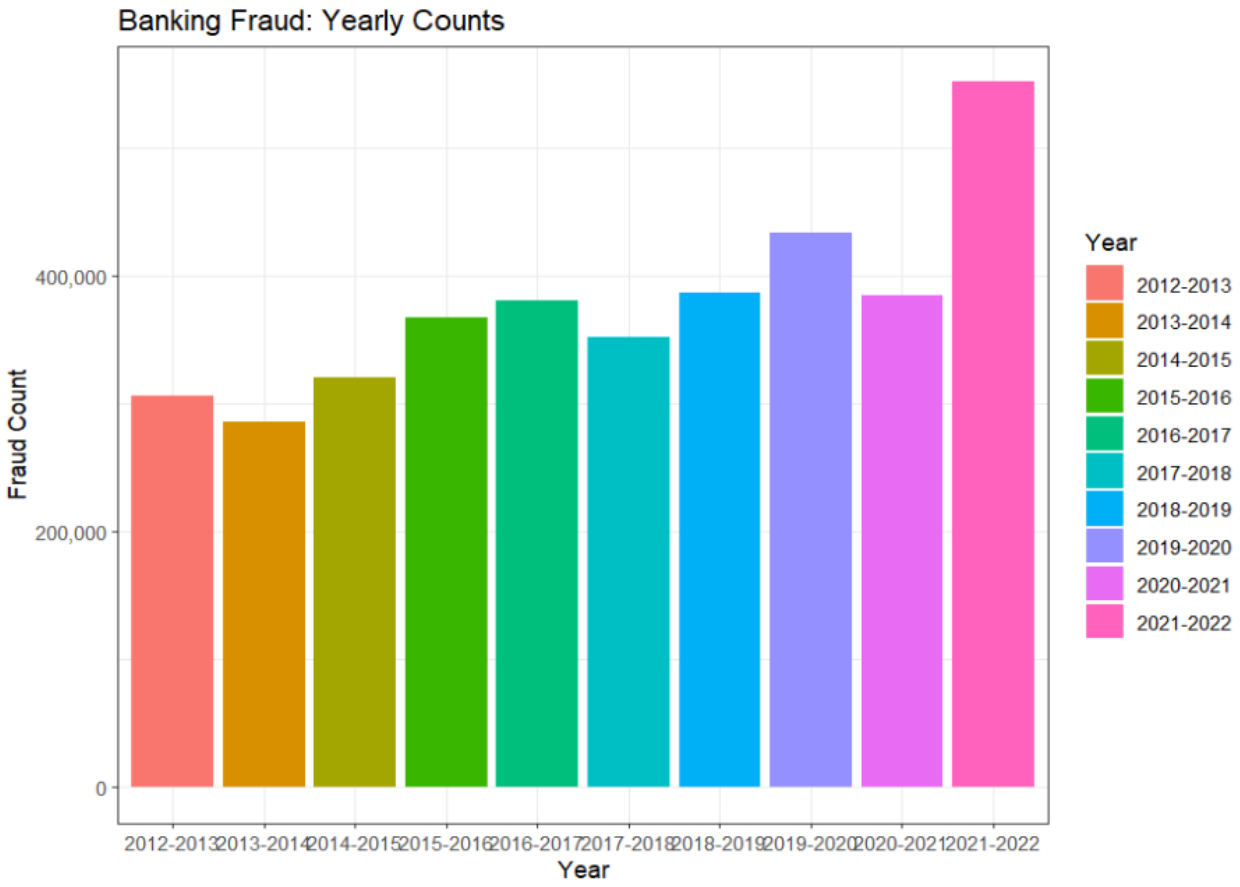
```r
# Visualize yearly counts of charity fraud using a bar chart with theme_bw
ggplot(charity_fraud_data, aes(x = Year, y = Fraud_Count, fill = Year)) +
  geom_bar(stat = "identity") +
  labs(title = "Yearly Charity Fraud Counts",
       x = "Year",
       y = "Number of Cases") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```

Plot:



```r
# Visualize yearly counts of banking fraud using a bar chart with theme_bw
ggplot(banking_fraud_data, aes(x = Year, y = Fraud_Count, fill = Year)) +
  geom_bar(stat = "identity") +
  labs(title = "Yearly Banking Fraud Counts",
       x = "Year",
       y = "Number of Cases") +
  scale_y_continuous(labels = comma) +
  theme_bw()
```

Plot:



3. Description of the workflow and justification of plotting choice:

Workflow description:

a. Data loading and preparation

First loaded the dataset from the specified excel sheet. Then examined the structure of the dataset to understand its format and updated column names for easier understanding.

b. Data subsetting

Secondly created subsets of the data for “Banking and credit industry fraud” and “All charity fraud”.

c. Data examination

Then used str() and head() to examine the structure and preview the first few rows of the subsets.

d. Data visualization

Used ggplot2 to create bar charts for visualizing the yearly counts of frauds.

Justification:

Bar chart is chosen for its effectiveness in comparing quantities across different categories. And ggplot2 package is used for its flexibility and powerful optimization options.

Question # 2

Data transformation and visualization using ggplot2

1. Code for creating a new data frame using package dplyr.

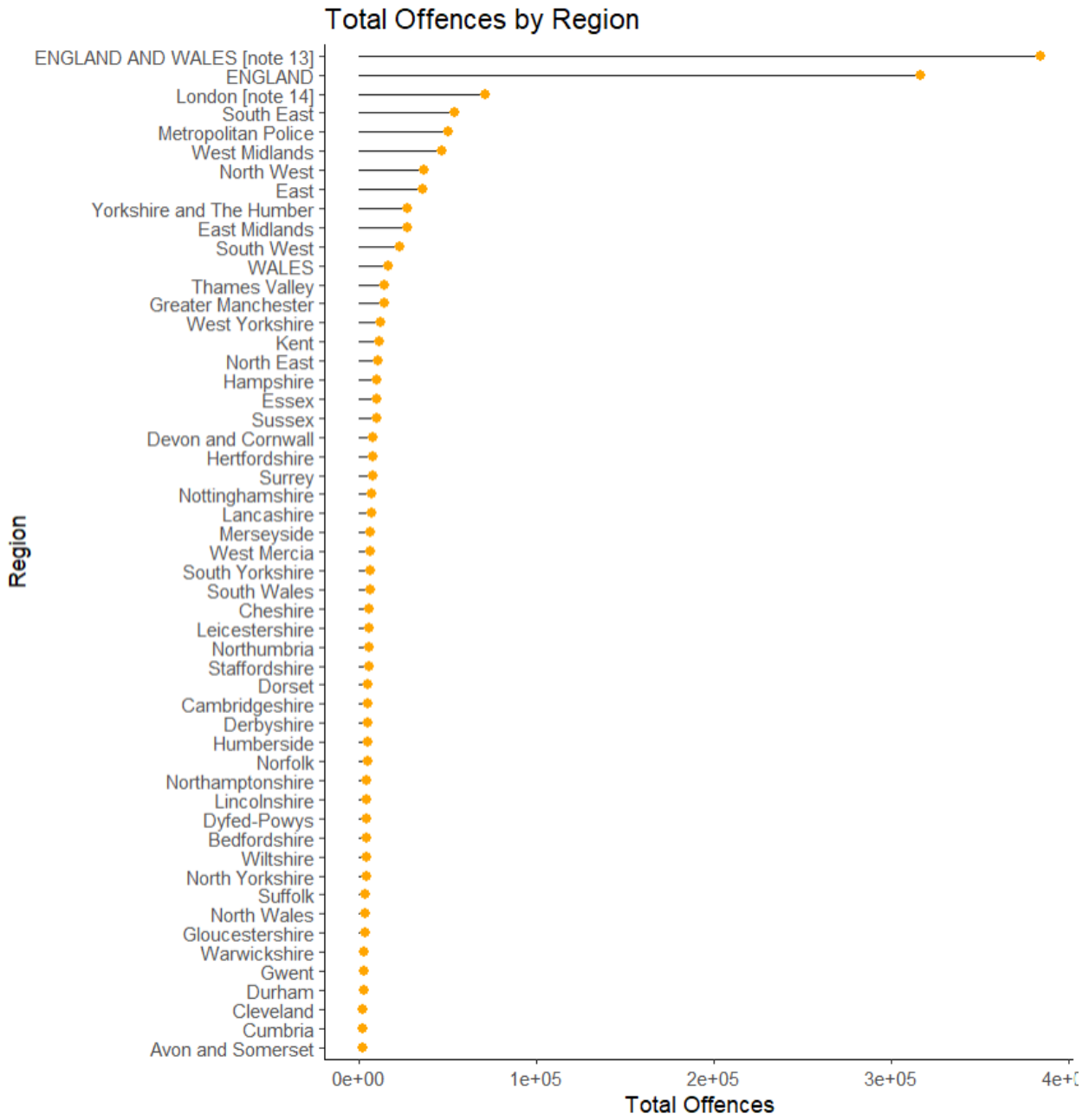
```
# Import necessary libraries
library(readxl)
library(ggplot2)
library(dplyr)
library(tidyr)
# Load the dataset from the "Table 5" sheet, skipping the first 9 rows
crime_records <- read_excel("D:/Code/HF/24-11-2024/cw_r.xlsx", sheet = "Table 5",
skip = 9)

# Examine the structure of the dataset to understand the column names and types
and rename the columns
str(crime_records)
colnames(crime_records) <- c("Region_Code", "Region_Name", "Offence_Count",
"Rate_Per_1000", "Change_Percentage")
# Convert the "Rate_Per_1000" column to numeric to enable proper calculations
crime_records$Rate_Per_1000 <- as.numeric(crime_records$Rate_Per_1000)

# Filter out rows where "Offence_Count" or "Rate_Per_1000" have missing values
crime_records <- crime_records %>%
  filter(!is.na(Offence_Count) & !is.na(Rate_Per_1000))
```

2. **Which region(s) has the highest total count of offences as well as offence rate per 1000 populations?**

Highest total count of offences:

```
# Aggregate the data by region to calculate total offences and average rate per
1000 population
regional_summary <- crime_records %>%
  group_by(Region_Name) %>%
  summarize(
    total_offences = sum(Offence_Count, na.rm = TRUE),
    avg_rate_per_1000 = mean(Rate_Per_1000, na.rm = TRUE)
  )

# Display the aggregated regional data for verification
print(regional_summary)

# Create a lollipop plot showing total offences by region, using a classic theme
ggplot(regional_summary, aes(x = reorder(Region_Name, total_offences), y =
total_offences)) +
  geom_segment(aes(xend = Region_Name, yend = 0), color = "black") +
  geom_point(color = "orange", size = 2) +
  coord_flip() +
  labs(title = "Total Offences by Region", x = "Region", y = "Total Offences") +
  theme_classic()
```

Plot:



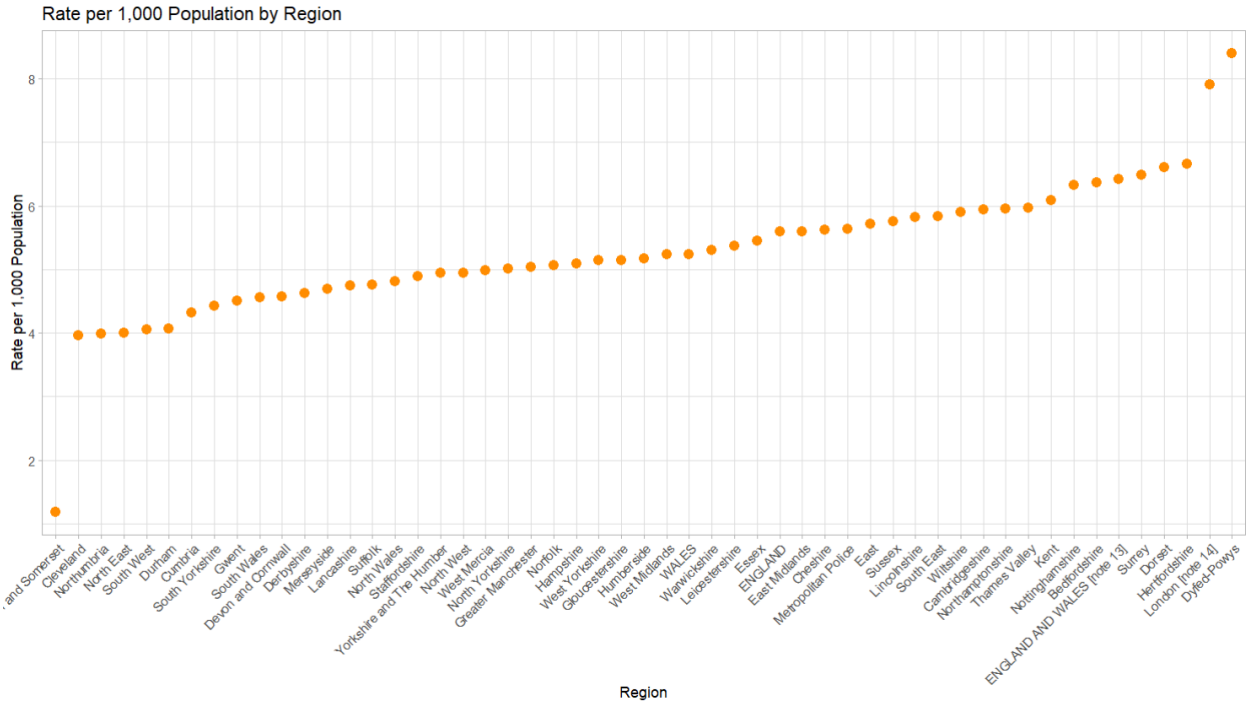
Justification:

The lollipop plot is chosen for its clarity in displaying the total offences by region. The horizontal lines and points makes it easy to compare the values across the different regions.

Offence rate per 1000:

```
# Generate a scatter plot to visualize the rate per 1,000 population by region
ggplot(regional_summary, aes(x = reorder(Region_Name, avg_rate_per_1000), y =
avg_rate_per_1000)) +
  geom_point(color = "darkorange", size = 3) +
  labs(title = "Rate per 1,000 Population by Region", x = "Region", y = "Rate per
1,000 Population") +
  theme_light() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Plot:



Justification:

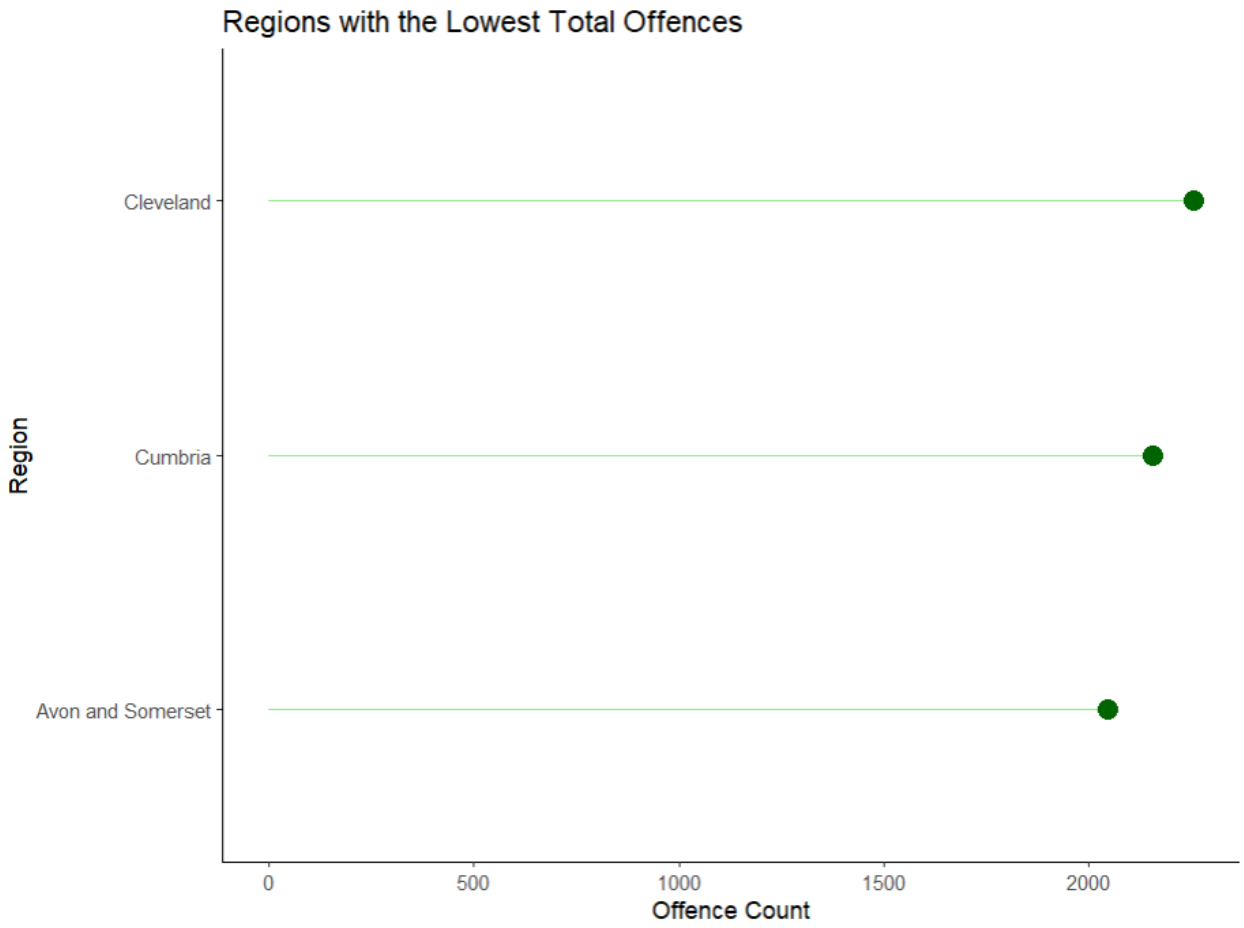
The scatter plot it used to visualize the rate per 1000 population by region because it effectively shows the distribution and allows for easy comparison of rates across regions. The use of orange color and size enhances the visual appeal of the plot.

3. What are the top three counties that have the lowest total count of offences?

```
# Identify the three counties with the lowest total crimes
low_offence_regions <- regional_summary %>%
  arrange(total_offences) %>%
```

```
  slice(1:3)

# Create a lollipop chart for regions with the lowest total offences, using a
minimal theme
ggplot(low_offence_regions, aes(x = reorder(Region_Name, total_offences), y =
total_offences)) +
  geom_segment(aes(xend = Region_Name, yend = 0), color = "lightgreen") +
  geom_point(color = "darkgreen", size = 4) +
  coord_flip() +
  labs(title = "Regions with the Lowest Total Offences", x = "Region", y =
"Offence Count") +
  theme_classic()
```

Plot:



Justification:

The lollipop plot is again chosen for its simplicity and effectiveness in highlighting the regions with the lowest total offences. The classic theme enhances the visual appeal and makes it easier to understand the plot.