1.5.3 ggplot exercises solutions

```
library(readr)

data_path = "data\\superstore.csv"
superstore <- read_csv(data_path, show_col_types = FALSE)
```

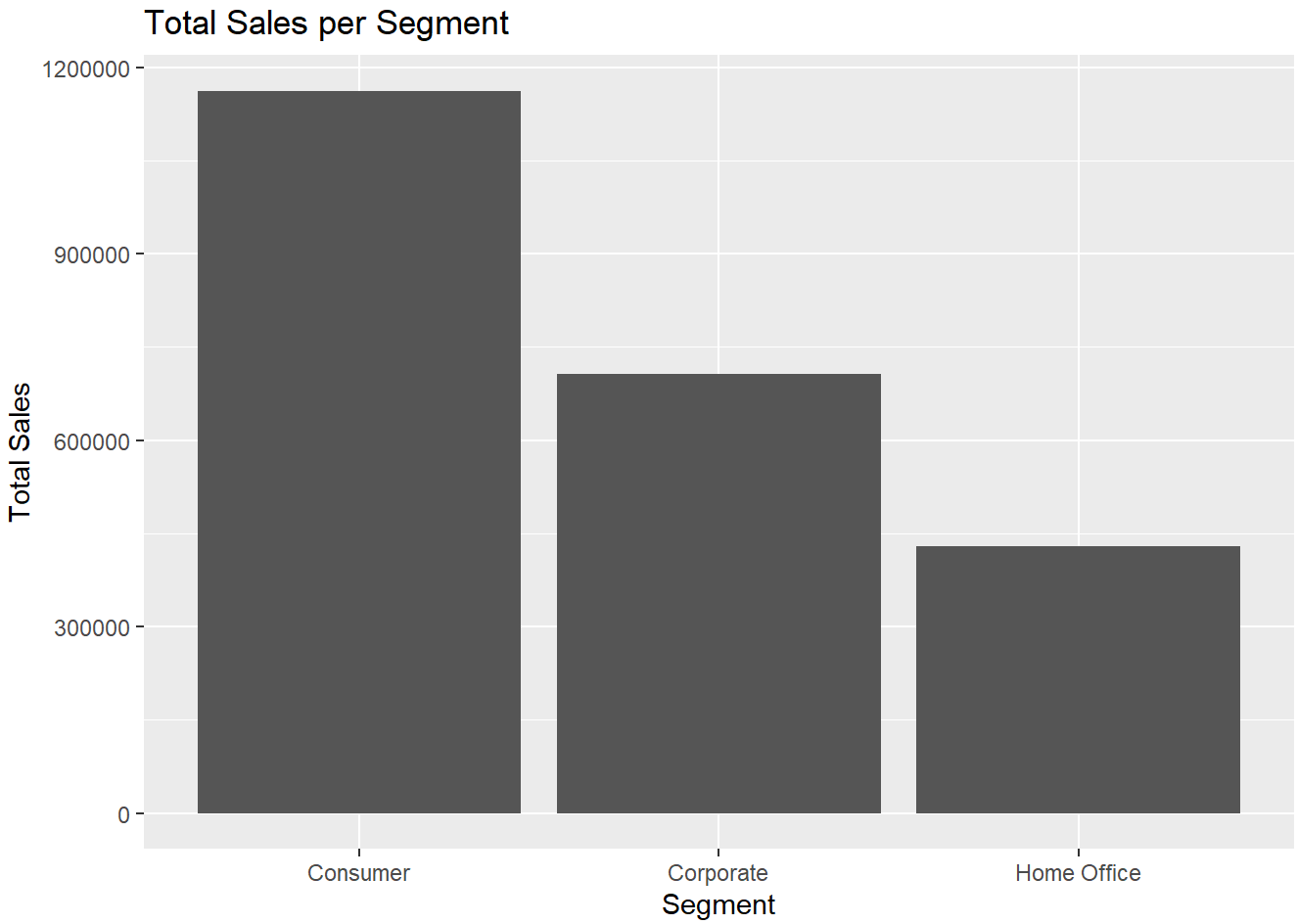
1. Q: Show the total sales per segment using a barplot.

A: We can use the `dplyr` package to summarize the total sales by segment and then plot it with `ggplot2`:

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(ggplot2)

superstore %>%
  group_by(Segment) %>%
  summarize(Total_Sales = sum(Sales)) %>%
  ggplot(aes(x = Segment, y = Total_Sales)) +
  geom_bar(stat = "identity") +
  labs(title = "Total Sales per Segment", x = "Segment", y = "Total Sales")
```

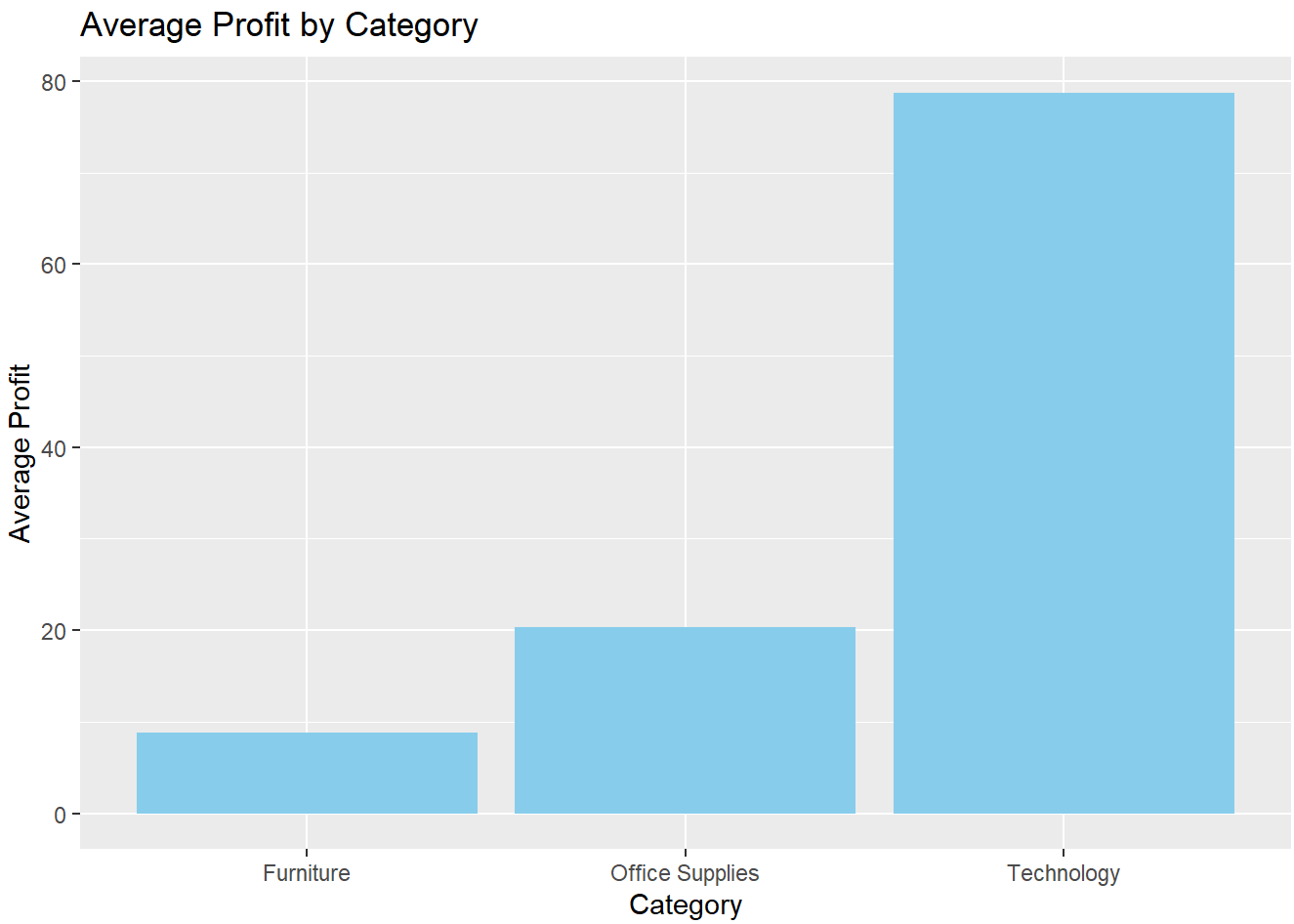


---

2. Q: Display the average profit by category using a barplot.

A: Use `dplyr` to calculate the average profit for each category and then visualize it with a barplot:

```
superstore %>%
  group_by(Category) %>%
  summarize(Average_Profit = mean(Profit)) %>%
  ggplot(aes(x = Category, y = Average_Profit)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  labs(title = "Average Profit by Category", x = "Category", y = "Average Profit")
```

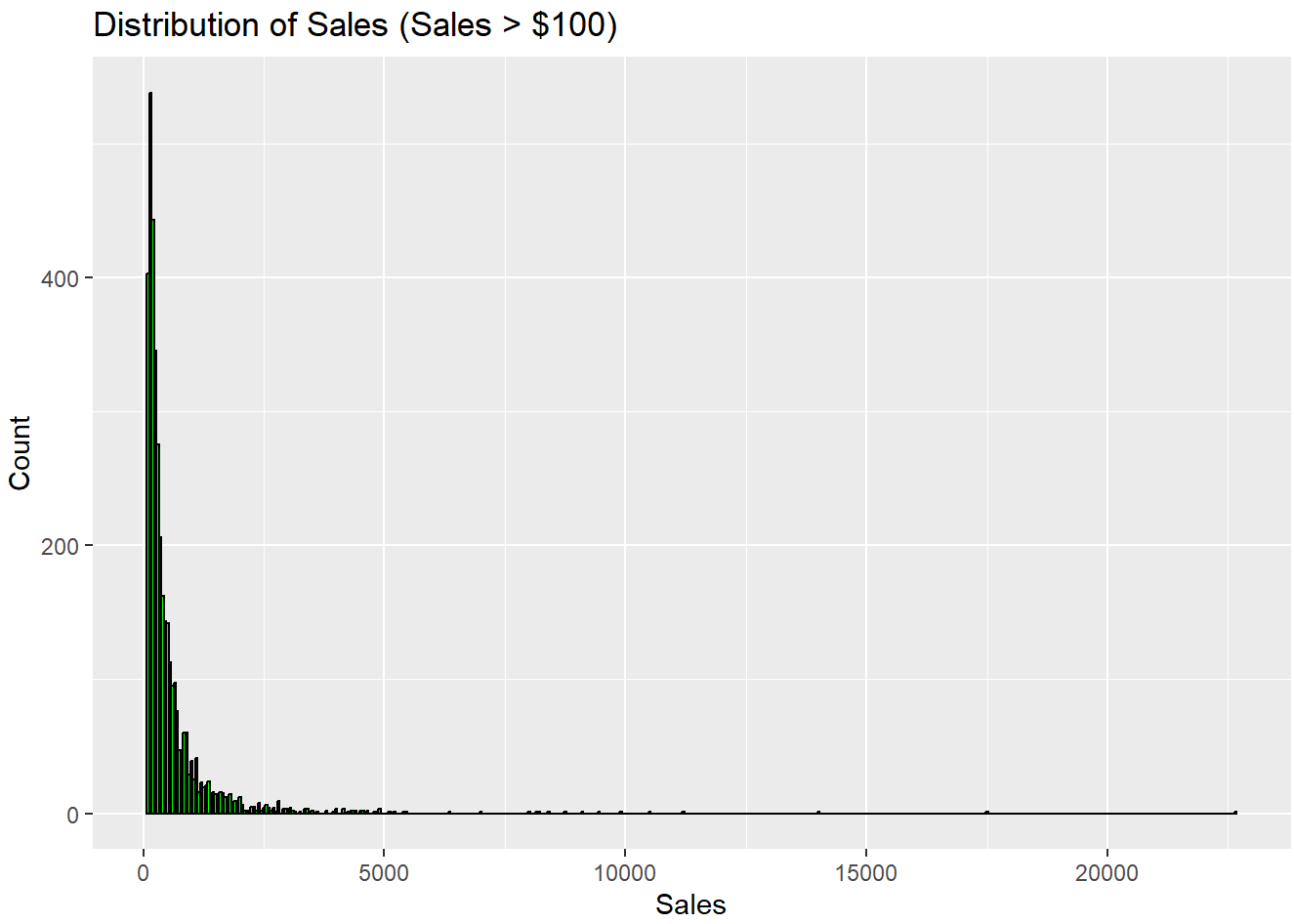


---

3. Q: Plot the distribution of sales using a histogram after filtering for sales greater than $100.

A: Filter the dataset for sales greater than $100 and then plot the distribution using a histogram:

```
superstore %>%
  filter(Sales > 100) %>%
  ggplot(aes(x = Sales)) +
  geom_histogram(binwidth = 50, fill = "green", color = "black") +
  labs(title = "Distribution of Sales (Sales > $100)", x = "Sales", y = "Count")
```

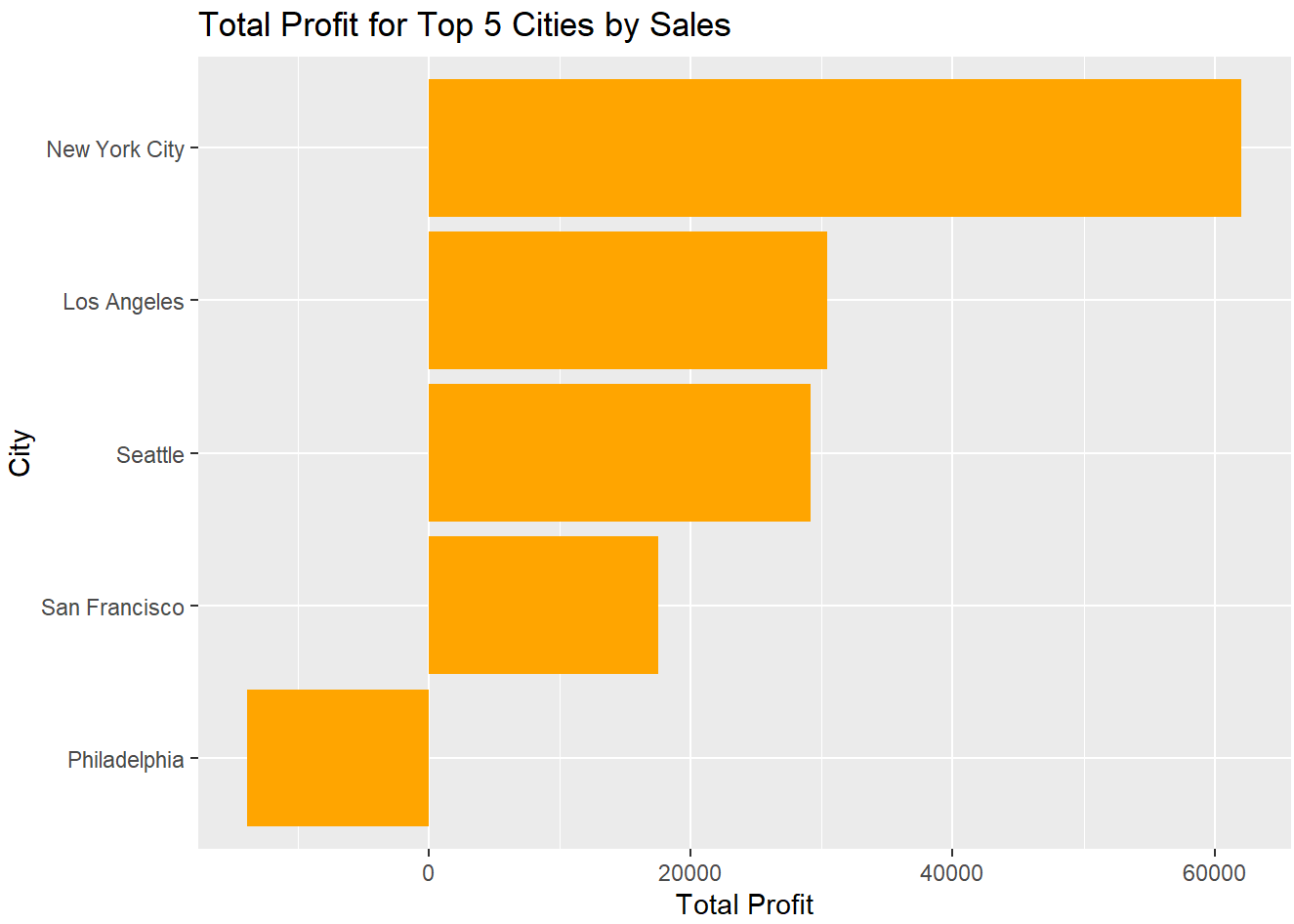


---

4. Q: Show the total profit for the top 5 cities by sales using a barplot.

A: First, find the top 5 cities by sales using `dplyr` and then plot the total profit for these cities:

```
top_cities <- superstore %>%
  group_by(City) %>%
  summarize(Total_Sales = sum(Sales)) %>%
  arrange(desc(Total_Sales)) %>%
  slice(1:5)

superstore %>%
  filter(City %in% top_cities$City) %>%
  group_by(City) %>%
  summarize(Total_Profit = sum(Profit)) %>%
  ggplot(aes(x = reorder(City, Total_Profit), y = Total_Profit)) +
  geom_bar(stat = "identity", fill = "orange") +
  labs(title = "Total Profit for Top 5 Cities by Sales", x = "City", y = "Total Profit")
  coord_flip()
```

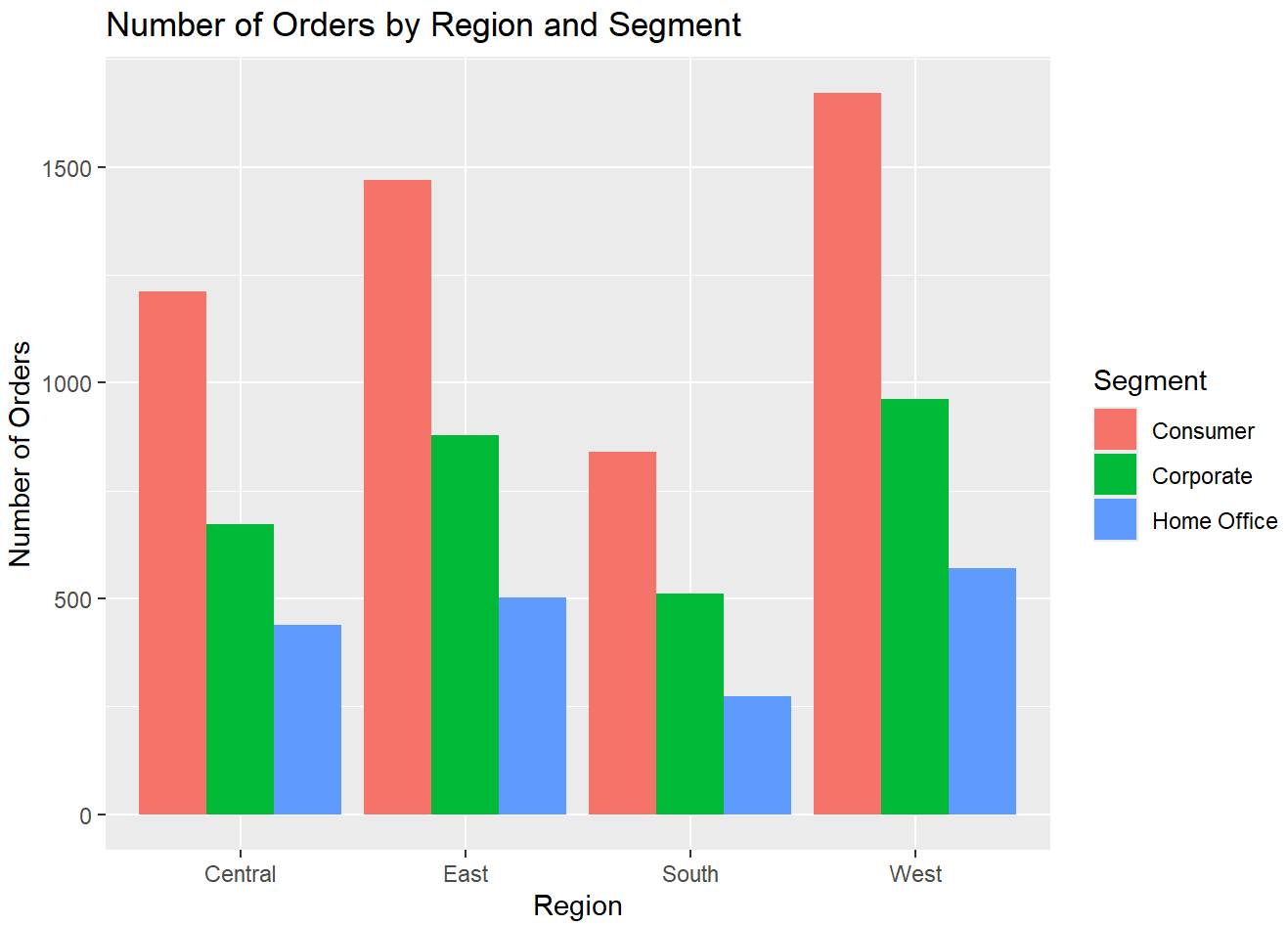


---

5. Q: Visualize the number of orders by region and segment using a grouped barplot.

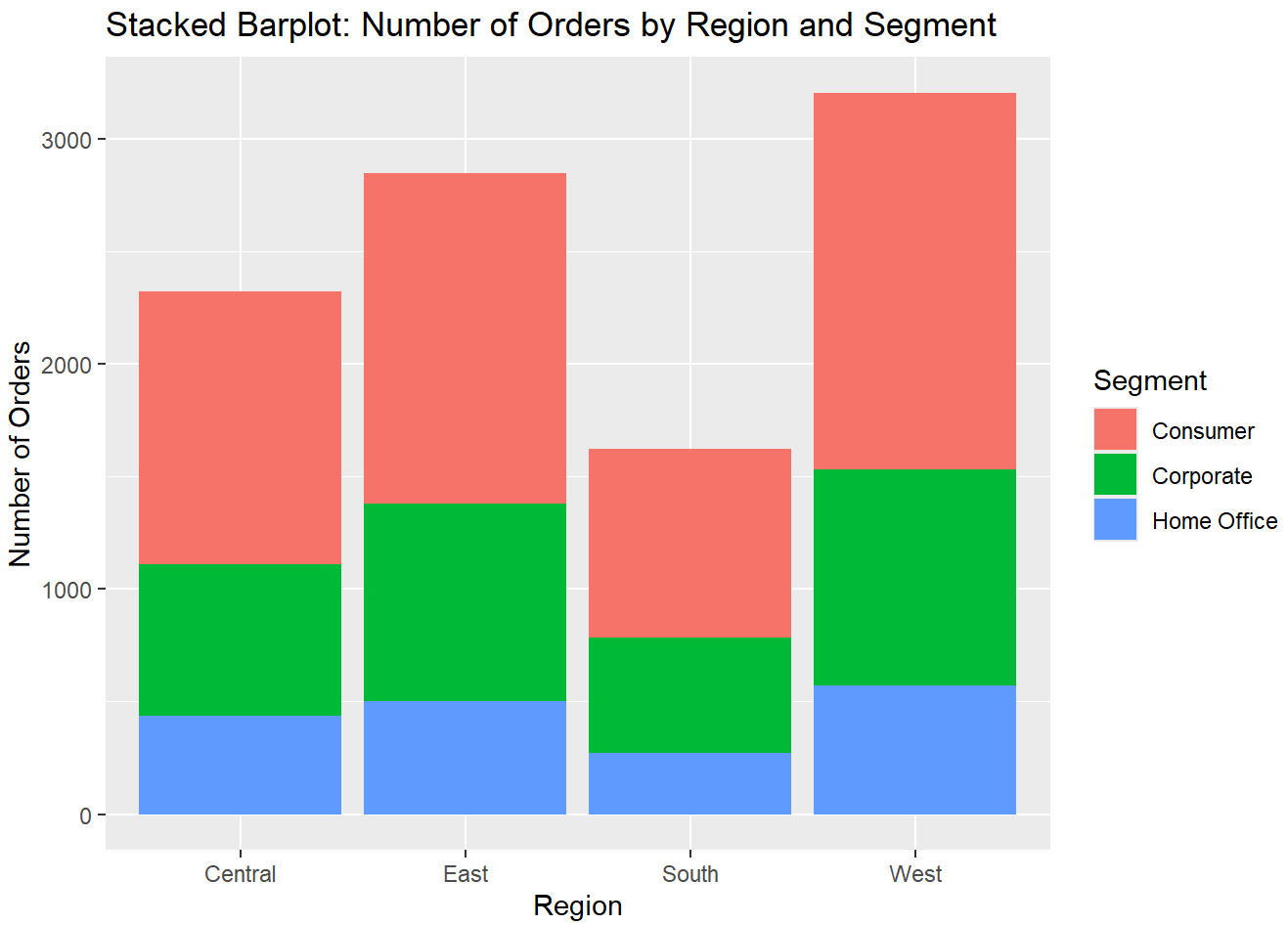
A: You can count the number of orders for each region and segment, then use a grouped barplot:

```
superstore %>%
  count(Region, Segment) %>%
  ggplot(aes(x = Region, y = n, fill = Segment)) +
  geom_bar(stat = "identity", position = "dodge") +
  labs(title = "Number of Orders by Region and Segment", x = "Region", y = "Number of Or
```



Using a stacked bar plot

```
superstore %>%
  count(Region, Segment) %>%
  ggplot(aes(x = Region, y = n, fill = Segment)) +
  geom_bar(stat = "identity", position = "stack") +
  labs(title = "Stacked Barplot: Number of Orders by Region and Segment", x = "Region",
```

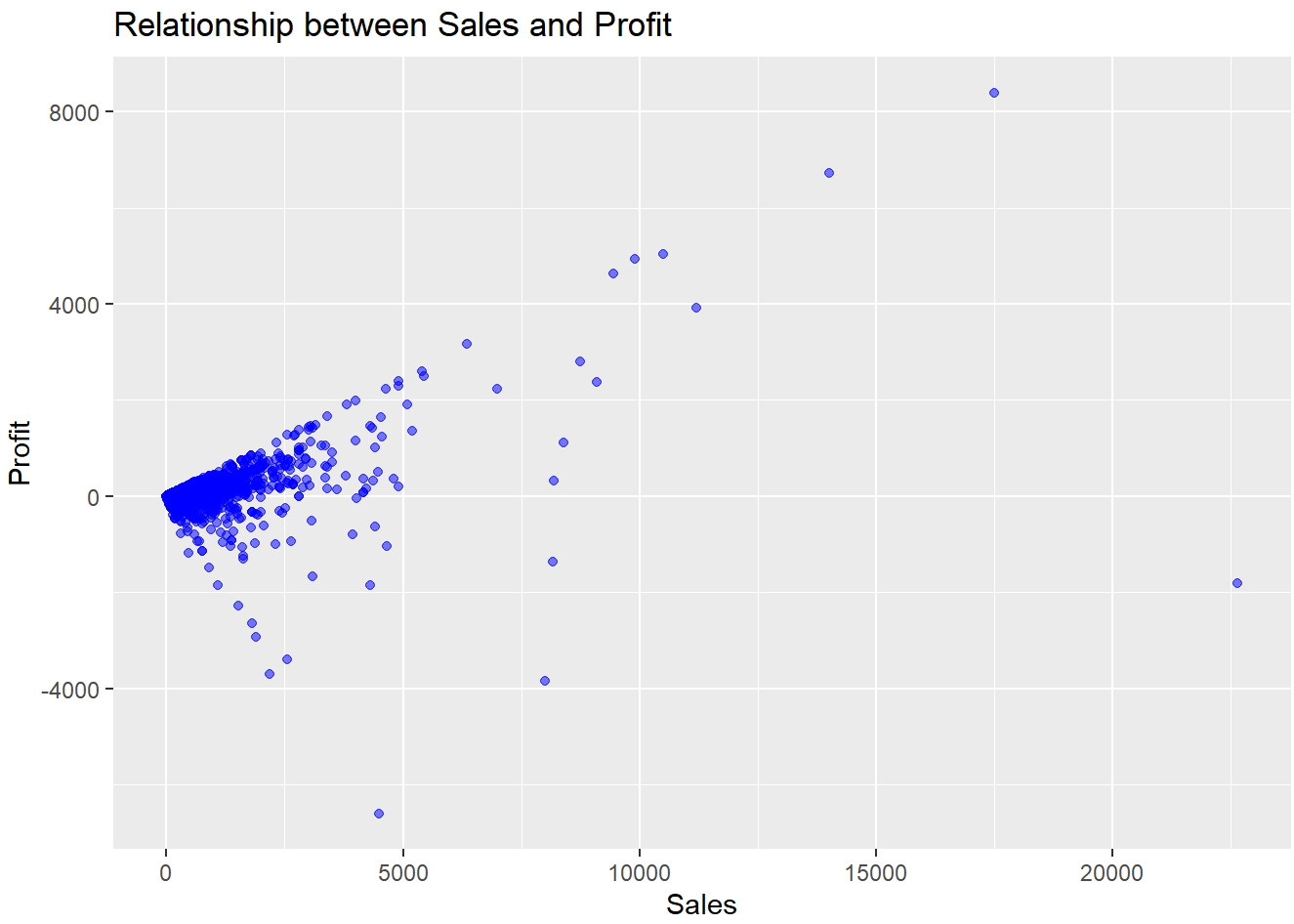


---

6. Q: Show the relationship between sales and profit using a scatter plot.

A: Use `ggplot2` to create a scatter plot to visualize the relationship between sales and profit:

```
superstore %>%
  ggplot(aes(x = Sales, y = Profit)) +
  geom_point(color = "blue", alpha = 0.5) +
  labs(title = "Relationship between Sales and Profit", x = "Sales", y = "Profit")
```

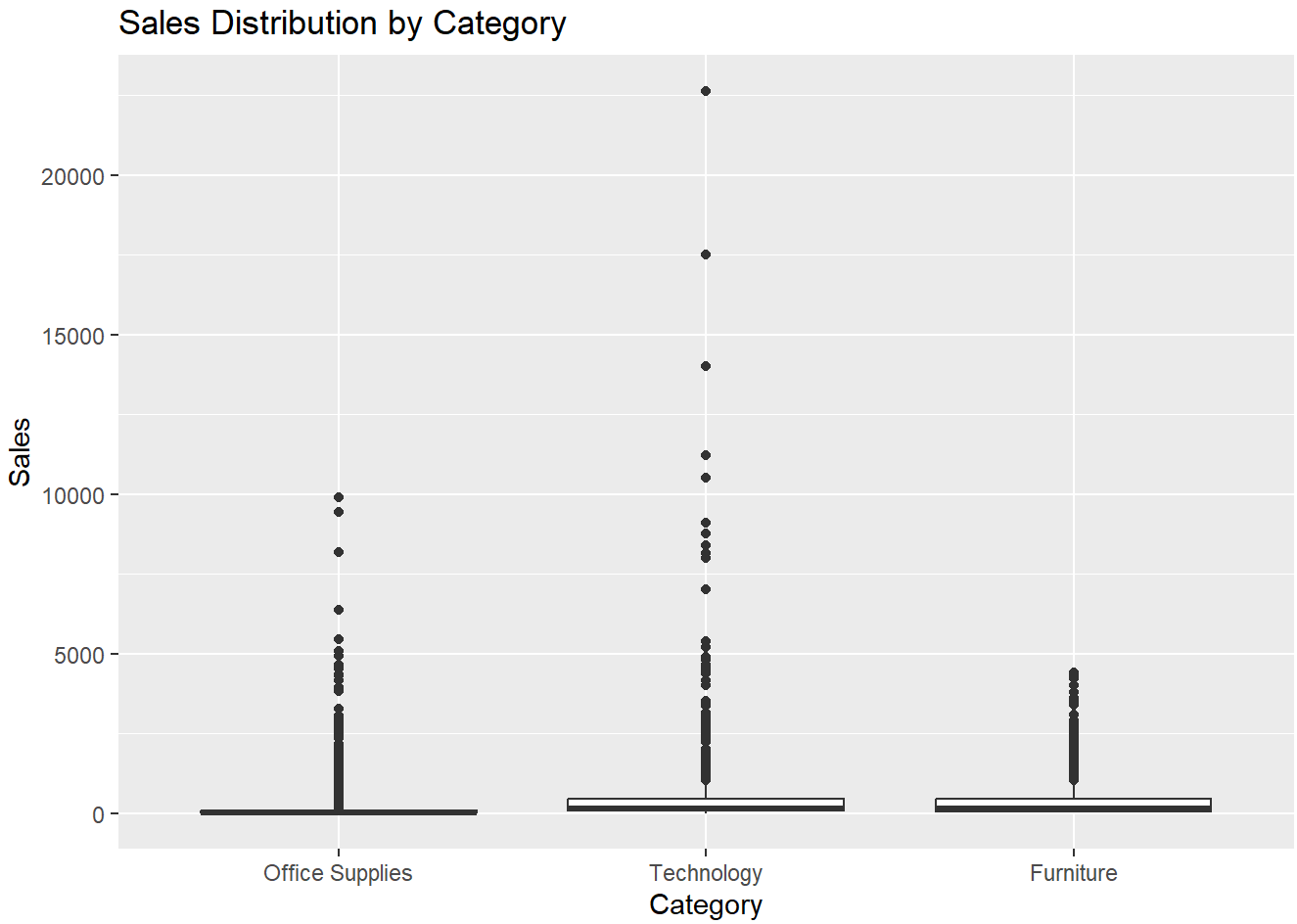


---

7. Q: Plot the sales distribution by category using a boxplot after arranging the categories by median sales.

A: Arrange categories by median sales and then plot the distribution using a boxplot:

```
superstore %>%
  mutate(Category = reorder(Category, Sales, FUN = median)) %>%
  ggplot(aes(x = Category, y = Sales)) +
  geom_boxplot() +
  labs(title = "Sales Distribution by Category", x = "Category", y = "Sales")
```



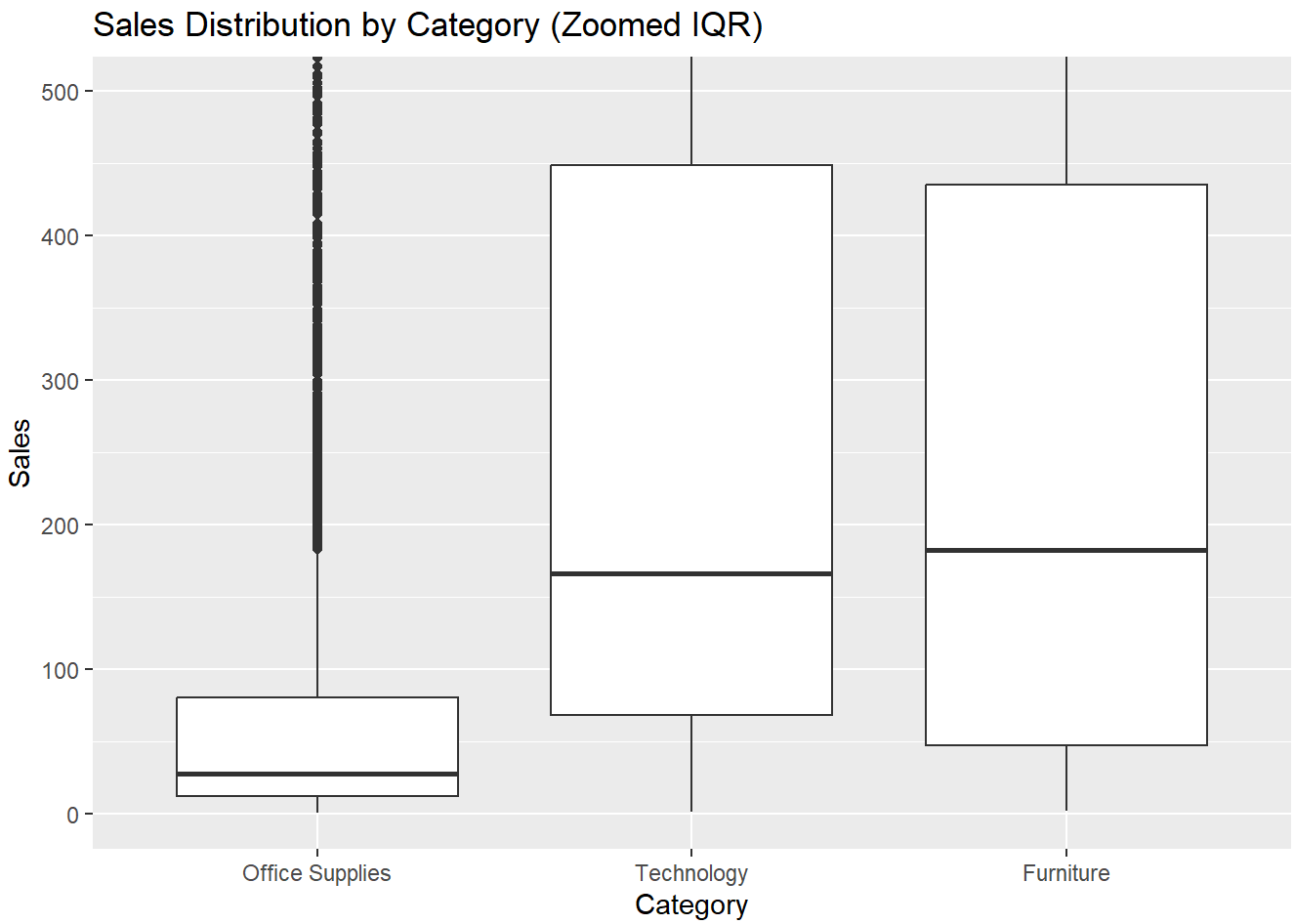
To make the box plot more visible when there are many outliers, you have a few options:

1. **Zoom in on the IQR (Interquartile Range)**: You can zoom in on the middle 50% of the data by adjusting the y-axis limits to focus on the interquartile range.
2. **Remove Outliers**: You can choose to remove outliers from the boxplot to better visualize the main distribution of the data.
3. **Log Transformation**: Applying a log transformation to the y-axis can help compress the range of the data, making the plot more interpretable.

Here's how you can apply each of these methods:

1. Zoom in on the IQR

```
superstore %>%   mutate(Category = reorder(Category, Sales, FUN = median)) %>%   ggplot(
```



The line `Category = reorder(Category, Sales, FUN = median)` in the `mutate()` function is used to reorder the factor levels of the `Category` variable based on the median of the `Sales` within each category. Here's a breakdown of what this does:

- **`Category`**: This refers to the factor variable that you want to reorder. In this case, it's the `Category` column in the Superstore dataset.
- **`reorder(Category, Sales, FUN = median)`**:
  - **`reorder()`**: This function is used to reorder the levels of a factor based on the values of another variable.
  - **`Category`**: The factor variable you are reordering.
  - **`Sales`**: The numeric variable based on which the reordering is done.
  - **`FUN = median`**: This specifies that the factor levels should be reordered based on the median value of `Sales` for each `Category`.

What It Achieves:

- **Reordering**: After applying this, the categories (e.g., Office Supplies, Technology, Furniture) in the `Category` variable will be ordered by the median `Sales` value in each category.
- **Visual Impact**: When you plot the data, the categories will be displayed in this new order. This is particularly useful in plots like boxplots where you might want to see categories ordered by their central tendency (median in this case) rather than their original order.

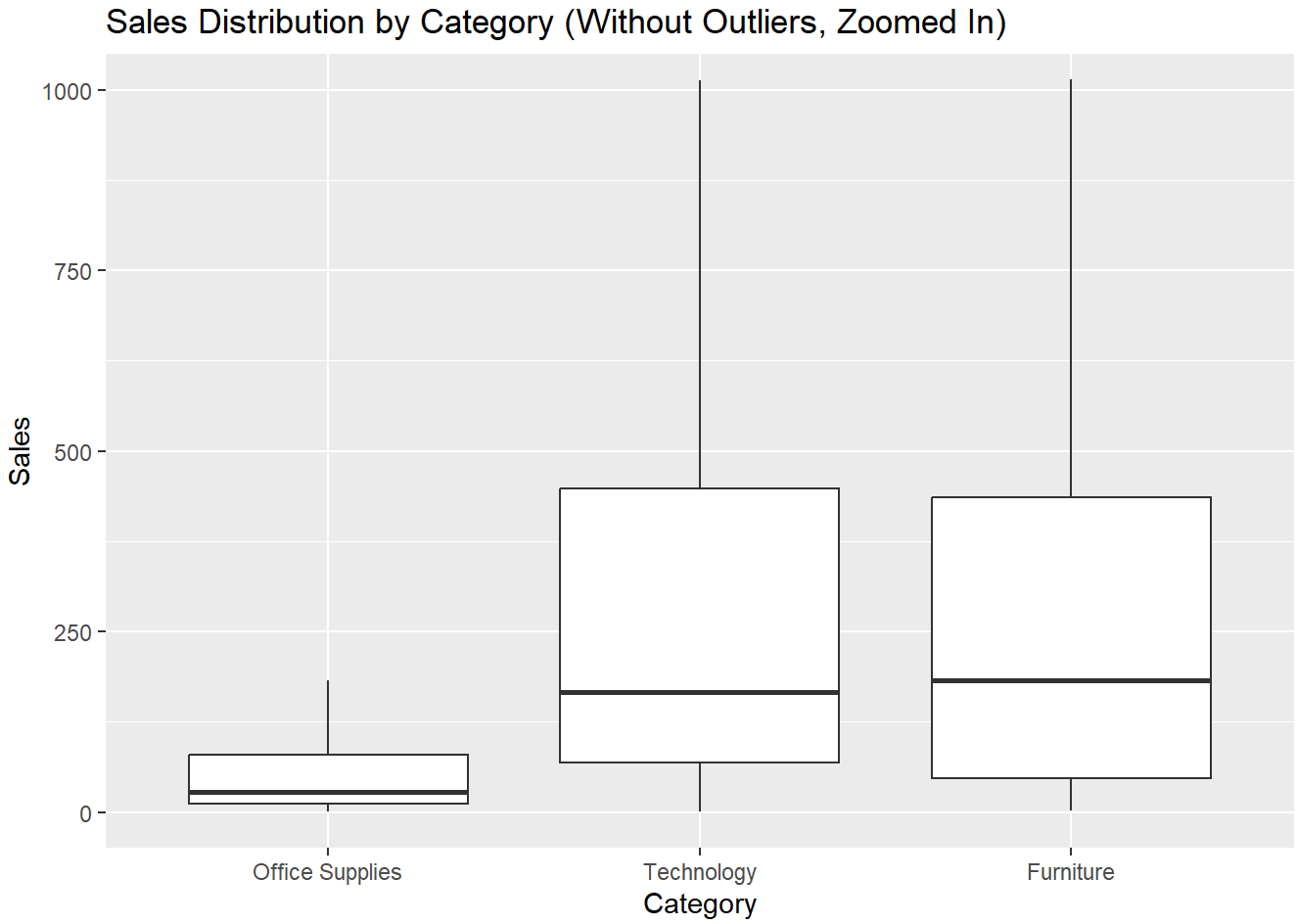
Example:

If the median sales for `Furniture` is higher than `Technology`, and `Technology` is higher than `Office Supplies`, the boxplot will display the categories in the following order: Furniture, Technology, Office Supplies.

This ordering can help make plots more informative by highlighting trends or comparisons in a more logical sequence based on the data.

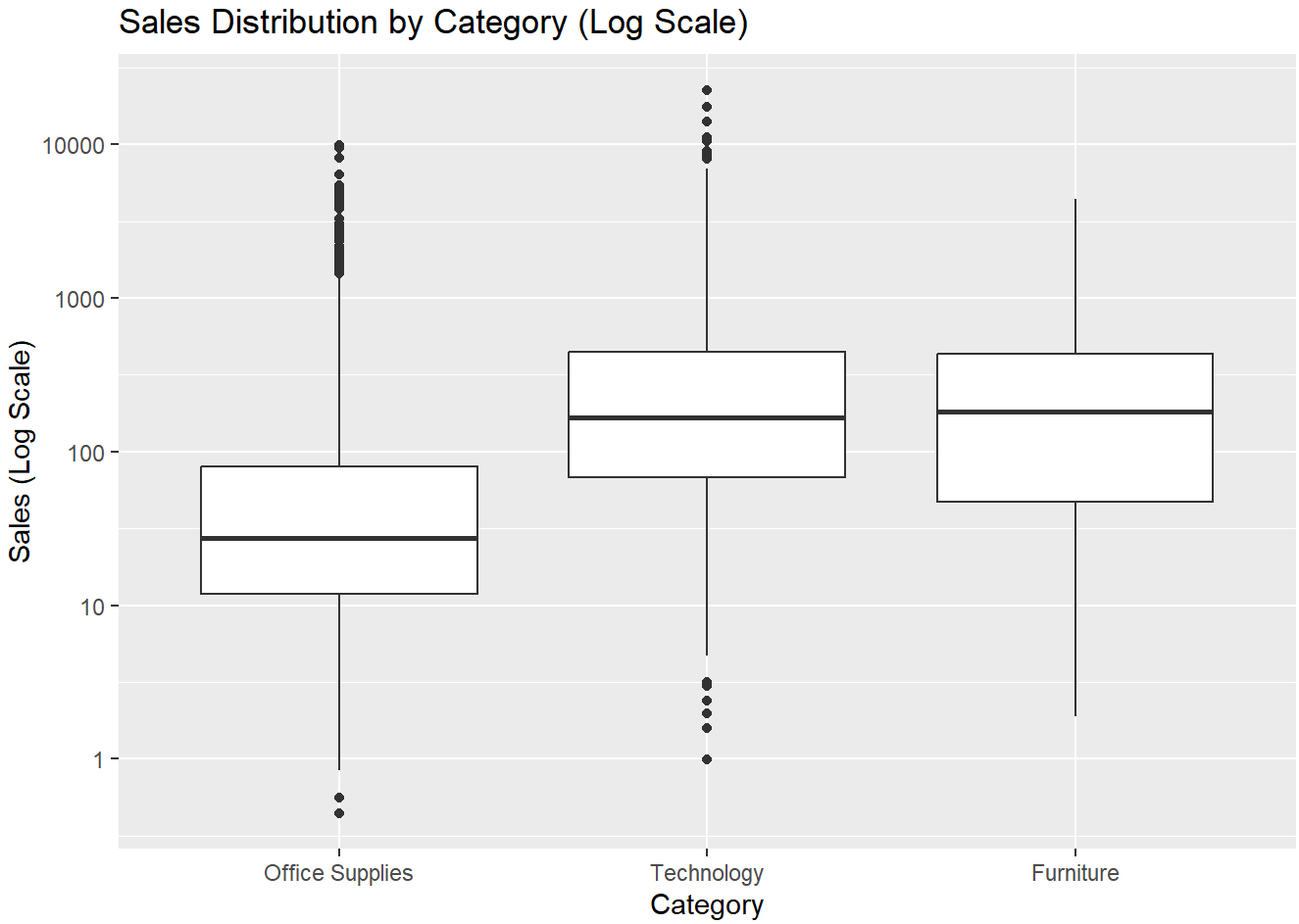
2. **Remove Outliers by Zooming on the Y Axis**

```
superstore %>%
  mutate(Category = reorder(Category, Sales, FUN = median)) %>%
  ggplot(aes(x = Category, y = Sales)) +
  geom_boxplot(outlier.shape = NA) +
  coord_cartesian(ylim = c(0, 1000)) +  # Adjust this limit based on the data range
  labs(title = "Sales Distribution by Category (Without Outliers, Zoomed In)", x = "Cate
```



3. **Log Transformation**

```
superstore %>%   mutate(Category = reorder(Category, Sales, FUN = median)) %>%   ggplot(
```

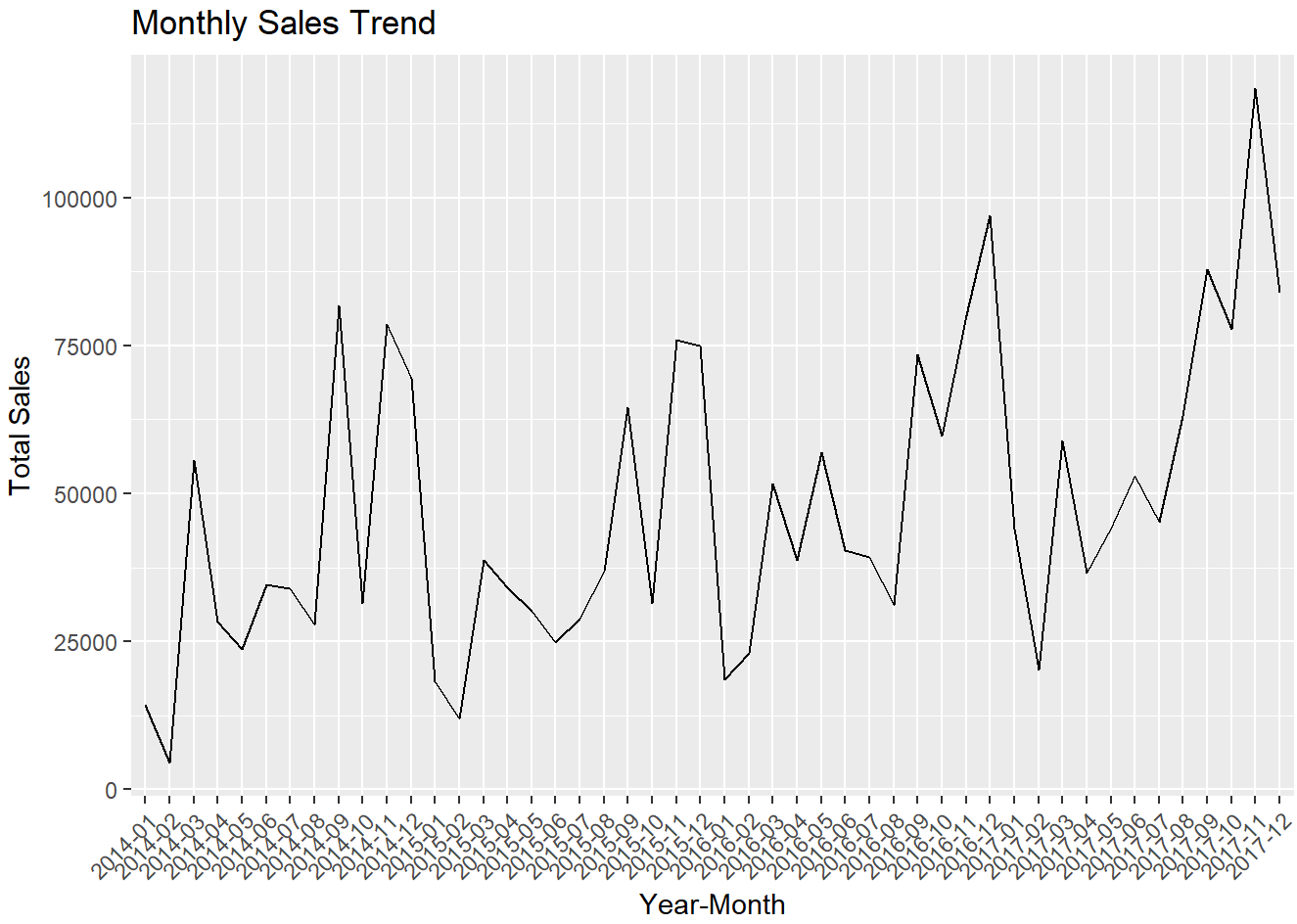


---

8. Q: Show the monthly sales trend over time using a line plot.

A: You can use the `mutate()` function to convert the `Order Date` into a proper date format and then group by month and year to visualize the sales trend:

```
library(dplyr)
library(ggplot2)

superstore %>%
  mutate(`Order Date` = as.Date(`Order Date`, format = "%m/%d/%Y")) %>%
  group_by(Year_Month = format(`Order Date`, "%Y-%m")) %>%
  summarize(Total_Sales = sum(Sales)) %>%
  ggplot(aes(x = Year_Month, y = Total_Sales, group = 1)) +
  geom_line() +
  labs(title = "Monthly Sales Trend", x = "Year-Month", y = "Total Sales") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

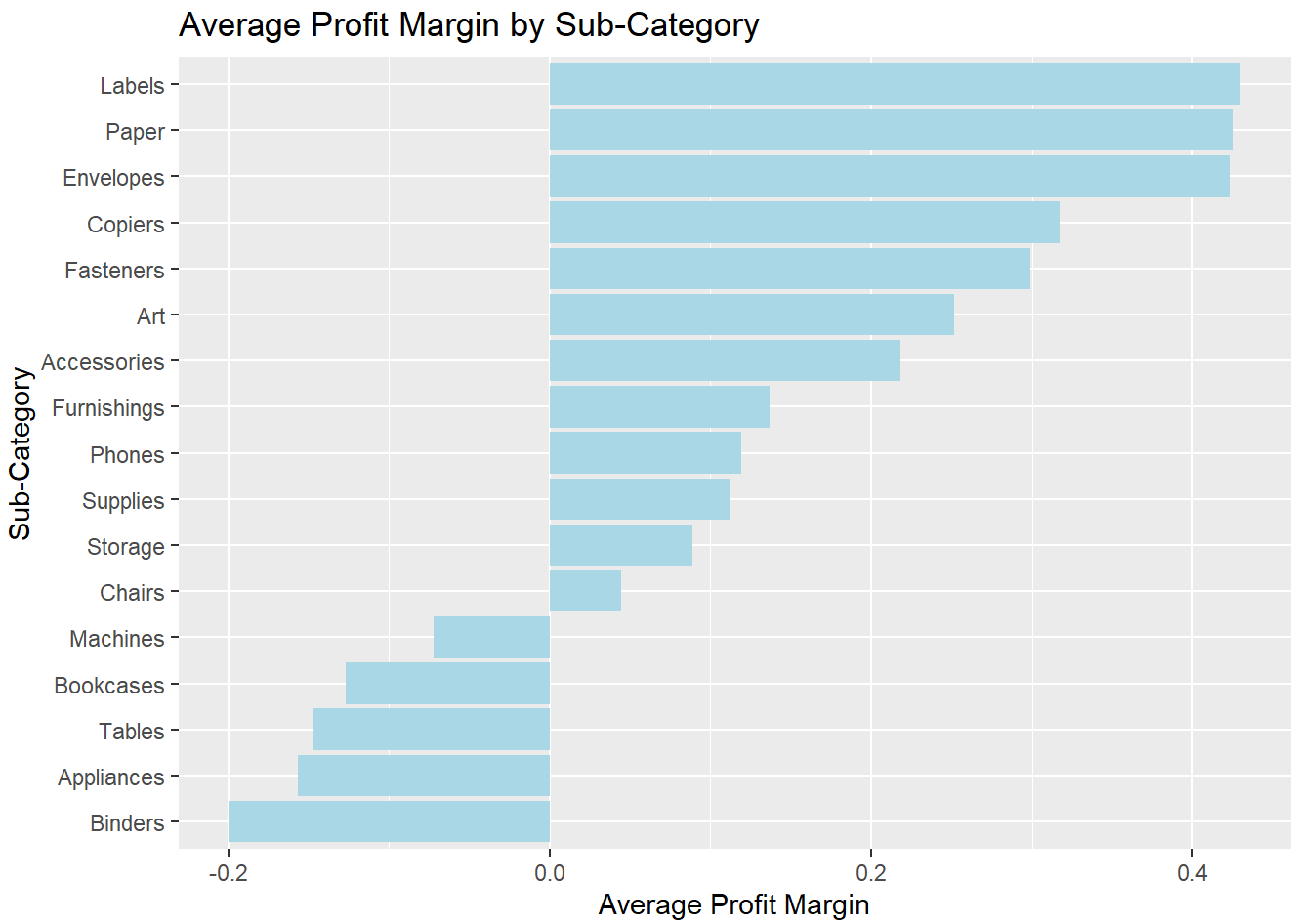


---

9. Q: Display the average profit margin by product sub-category using a barplot.

A: Calculate the average profit margin for each sub-category and visualize it using a barplot:

```
superstore %>%
  group_by(`Sub-Category`) %>%
  summarize(Avg_Profit_Margin = mean(Profit / Sales)) %>%
  ggplot(aes(x = reorder(`Sub-Category`, Avg_Profit_Margin), y = Avg_Profit_Margin)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  coord_flip() +
  labs(title = "Average Profit Margin by Sub-Category", x = "Sub-Category", y = "Average
```

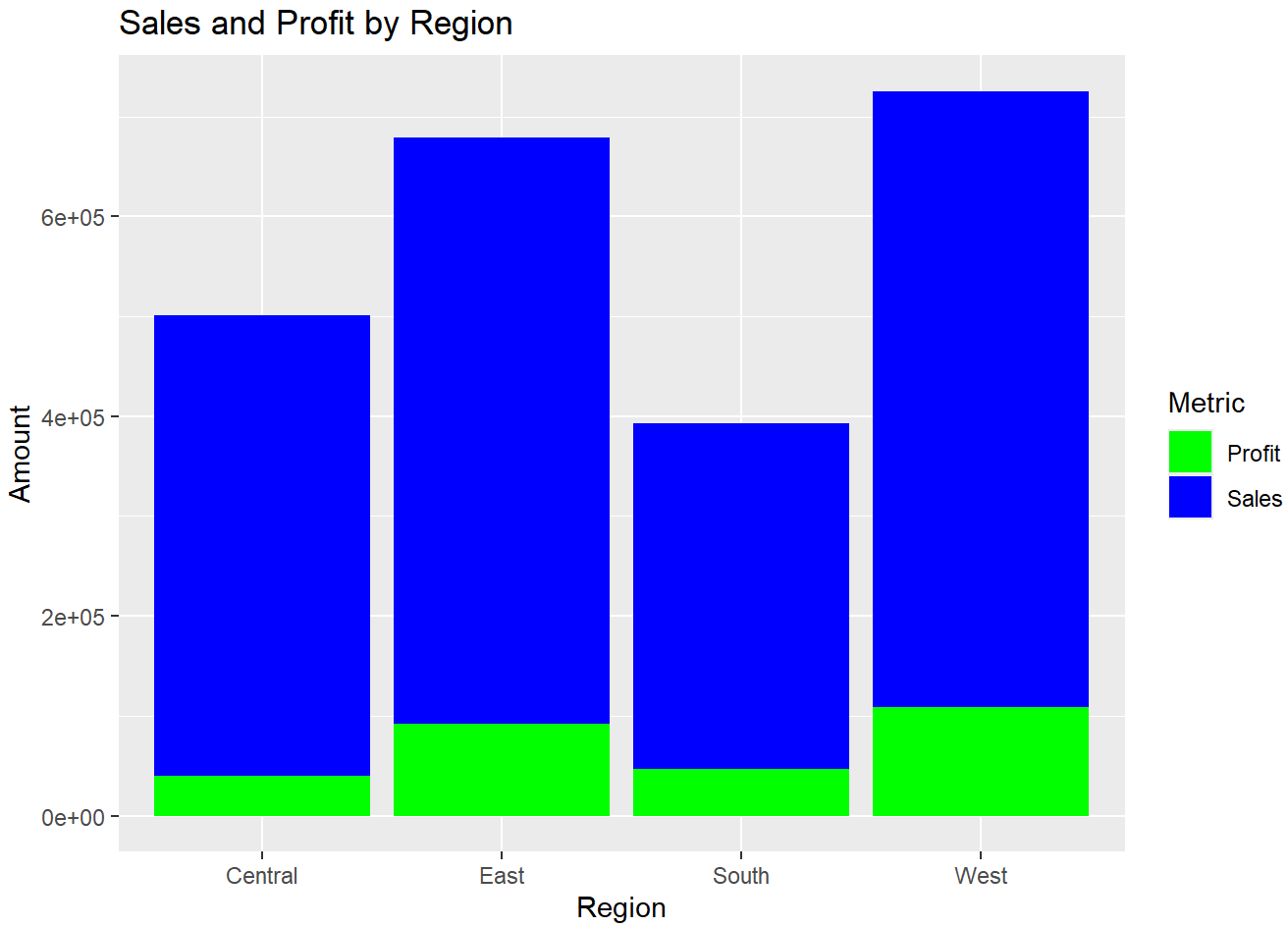


---

10. Q: Show sales and profit by region using side-by-side bar plots.

A: Compare sales and profit across different regions using side-by-side bar plots:

```
superstore %>%
  group_by(Region) %>%
  summarize(Total_Sales = sum(Sales), Total_Profit = sum(Profit)) %>%
  ggplot() +
  geom_bar(aes(x = Region, y = Total_Sales, fill = "Sales"), stat = "identity", position
  geom_bar(aes(x = Region, y = Total_Profit, fill = "Profit"), stat = "identity", positi
  labs(title = "Sales and Profit by Region", x = "Region", y = "Amount") +
  scale_fill_manual(name = "Metric", values = c("Sales" = "blue", "Profit" = "green"))
```

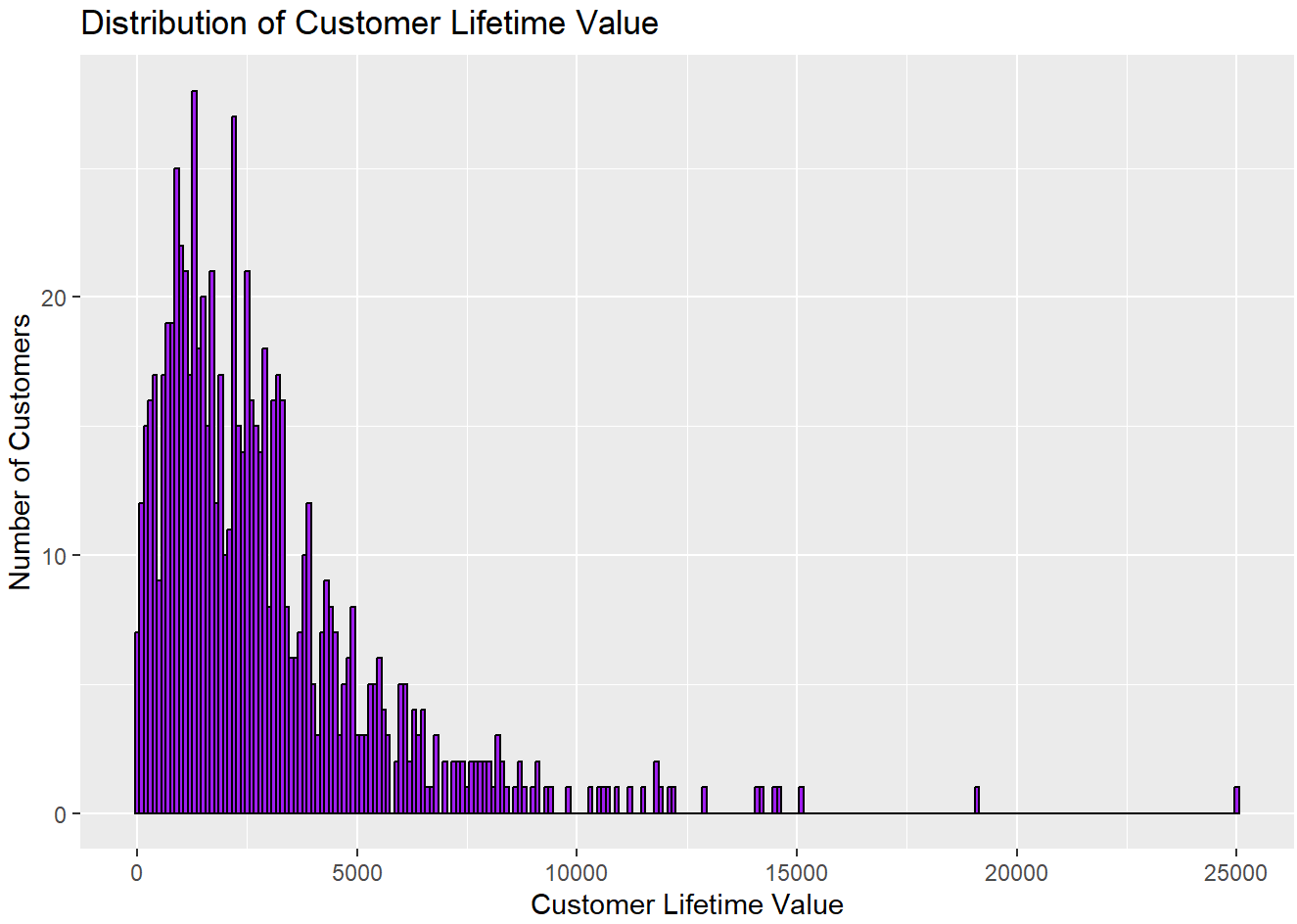


---

11. Q: Visualize the distribution of customer lifetime value (CLTV) using a histogram.

A: Calculate the lifetime value for each customer and plot the distribution using a histogram:

```
superstore %>%
  group_by(`Customer ID`) %>%
  summarize(Customer_Lifetime_Value = sum(Sales)) %>%
  ggplot(aes(x = Customer_Lifetime_Value)) +
  geom_histogram(binwidth = 100, fill = "purple", color = "black") +
  labs(title = "Distribution of Customer Lifetime Value", x = "Customer Lifetime Value",
```



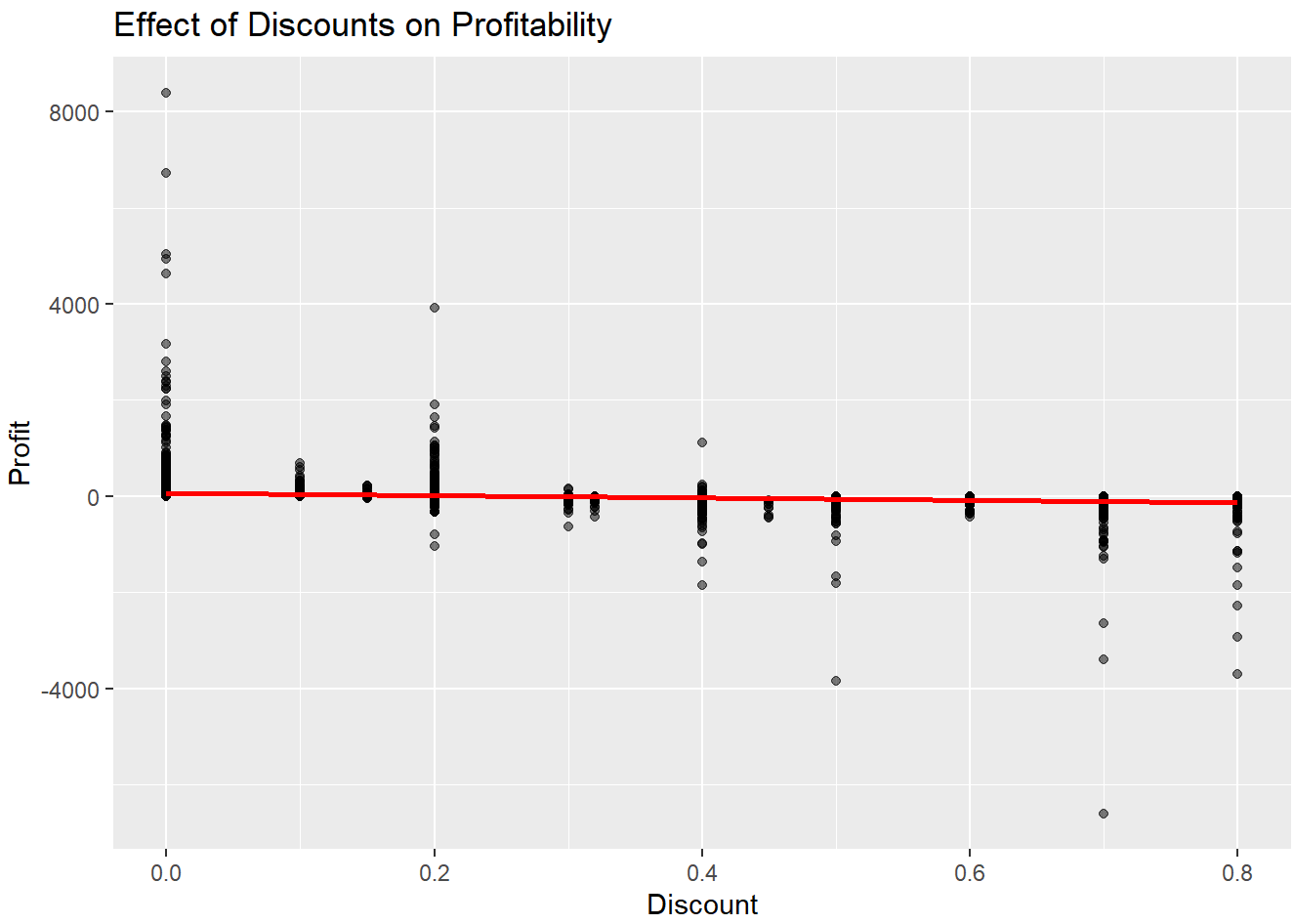
---

12. Q: Analyze the effect of discounts on profitability using a scatter plot.

A: Plot a scatter plot to examine the relationship between discount and profit, including a trend line:

```
superstore %>%
  ggplot(aes(x = Discount, y = Profit)) +
  geom_point(alpha = 0.5) +
  geom_smooth(method = "lm", color = "red") +
  labs(title = "Effect of Discounts on Profitability", x = "Discount", y = "Profit")
```

```
`geom_smooth()` using formula = 'y ~ x'
```



Preparing Data for Market Basket Analysis Here's how you can prepare the Superstore data for market basket analysis:

Group Items by Transaction (Order ID): You need to group the items by Order ID and then convert them into a list format where each list element contains all the items bought in that transaction.

Convert to Transaction Format: The list of items per transaction can then be converted into a transaction object, which can be used for association rule mining with the arules package.

```
library(dplyr)
library(tidyr)
library(arules)
```

```
Loading required package: Matrix


Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack
```

```
Attaching package: 'arules'

The following object is masked from 'package:dplyr':

    recode

The following objects are masked from 'package:base':

    abbreviate, write
```

```
library(arulesViz)


# Prepare the data by grouping products by Order ID
superstore_basket <- superstore %>%
  select(`Order ID`, `Product Name`) %>%
  group_by(`Order ID`) %>%
  summarize(Items = paste(`Product Name`, collapse = ",")) %>%
  separate_rows(Items, sep = ",")

# Convert the data into a transaction format
transactions <- as(split(superstore_basket$Items, superstore_basket$`Order ID`), "transa
```

```
Warning in asMethod(object): removing duplicated items in transactions
```

```
# Support (supp): Lowering this value increases the number of itemsets that meet the fre
# Confidence (conf): Lowering this value increases the number of rules that meet the con

# Apply the apriori algorithm with lower support and confidence
rules <- apriori(transactions, parameter = list(supp = 0.001, conf = 0.1))
```

```
Apriori

Parameter specification:
 confidence minval smax arem  aval originalSupport maxtime support minlen
        0.1    0.1    1 none FALSE            TRUE       5   0.001      1
 maxlen target  ext
     10  rules TRUE

Algorithmic control:
 filter tree heap memopt load sort verbose
    0.1 TRUE TRUE  FALSE TRUE    2    TRUE

Absolute minimum support count: 5

set item appearances ...[0 item(s)] done [0.00s].
```

```
set transactions ...[2183 item(s), 5009 transaction(s)] done [0.00s].
sorting and recoding items ... [1015 item(s)] done [0.00s].
creating transaction tree ... done [0.00s].
checking subsets of size 1 2 3 4 5 6 done [0.00s].
writing ... [1186 rule(s)] done [0.00s].
creating S4 object  ... done [0.00s].
```

```r
# Check if rules were generated
summary(rules)
```

```
set of 1186 rules

rule length distribution (lhs + rhs):sizes
  2   3   4   5   6
668 357 120  35   6

   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  2.000   2.000   2.000   2.612   3.000   6.000

summary of quality measures:
    support           confidence        coverage             lift
 Min.   :0.001198   Min.   :0.1053   Min.   :0.001198   Min.   :  7.312
 1st Qu.:0.001198   1st Qu.:1.0000   1st Qu.:0.001198   1st Qu.:294.647
 Median :0.001397   Median :1.0000   Median :0.001397   Median :626.125
 Mean   :0.001531   Mean   :0.9151   Mean   :0.002032   Mean   :536.068
 3rd Qu.:0.001797   3rd Qu.:1.0000   3rd Qu.:0.001996   3rd Qu.:834.833
 Max.   :0.004791   Max.   :1.0000   Max.   :0.015173   Max.   :834.833
     count
 Min.   : 6.00
 1st Qu.: 6.00
 Median : 7.00
 Mean   : 7.67
 3rd Qu.: 9.00
 Max.   :24.00

mining info:
         data ntransactions support confidence
 transactions          5009   0.001        0.1
                                                                        call
 apriori(data = transactions, parameter = list(supp = 0.001, conf = 0.1))
```

```r
# Visualize the rules if any exist
if (length(rules) > 0) {
  plot(rules, method = "graph", interactive = TRUE)
} else {
  print("No rules were found. Try lowering support or confidence levels.")
}
```

```
Warning in plot.rules(rules, method = "graph", interactive = TRUE): The
parameter interactive is deprecated. Use engine='interactive' instead.

Warning: Too many rules supplied. Only plotting the best 100 using 'lift'
(change control parameter max if needed).
```

```
# # Now, you can apply the apriori algorithm
# rules <- apriori(transactions, parameter = list(supp = 0.01, conf = 0.5))
#
# # To visualize the rules
# library(arulesViz)
# plot(rules, method = "graph", interactive = TRUE)
```

Explanation:

- **Grouping**: The `group_by(Order ID)` groups all products under the same `Order ID`.
- **Collapse and Separate**: The `summarize()` with `paste()` concatenates all product names in a single transaction into one string. The `separate_rows()` function then separates these into individual rows within a transaction.
- **Convert to Transactions**: This prepares the data for the `apriori` function by converting the grouped items into a transaction object.
- **Association Rules**: Finally, the `apriori` algorithm is applied to find frequent itemsets and association rules.

Note: Ensure that the `arules` and `arulesViz` packages are installed for running the market basket analysis.

This process should help us to convert our Superstore dataset into a transaction format suitable for market basket analysis, enabling usto discover interesting product associations and co-purchase patterns.