R Notebook

```r
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 4.2.3
```

```r
library(gridExtra)
```

```
## Warning: package 'gridExtra' was built under R version 4.2.3
```

```r
library(reshape2)
library(plotly)
```

```
## Warning: package 'plotly' was built under R version 4.2.3
```

```
##
## Attaching package: 'plotly'
```

```
## The following object is masked from 'package:ggplot2':
##
##     last_plot
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

```
## The following object is masked from 'package:graphics':
##
##     layout
```

```r
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 4.2.3
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:gridExtra':
##
##     combine
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
library(gplots)
```

```
## Warning: package 'gplots' was built under R version 4.2.3
```

```
##
## Attaching package: 'gplots'
```

```
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(lubridate)
```

```
## Warning: package 'lubridate' was built under R version 4.2.3
```

```
##
## Attaching package: 'lubridate'
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 4.2.3
```

```
## Warning: package 'tibble' was built under R version 4.2.3
```

```
## Warning: package 'readr' was built under R version 4.2.3
```

```
## Warning: package 'purrr' was built under R version 4.2.3
```

```
## Warning: package 'stringr' was built under R version 4.2.3
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ forcats 1.0.0     ✔ stringr 1.5.1
## ✔ purrr   1.0.2     ✔ tibble  3.2.1
## ✔ readr   2.1.4     ✔ tidyr   1.3.0
```

```
## ── Conflicts ──────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::combine() masks gridExtra::combine()
## ✖ dplyr::filter()  masks plotly::filter(), stats::filter()
## ✖ dplyr::lag()     masks stats::lag()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to be
come errors
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.2.3
```

```
## Loading required package: lattice
```

```
## Warning: package 'lattice' was built under R version 4.2.3
```

```
##
## Attaching package: 'caret'
##
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(tidyr)
library(modelr)
```

```
## Warning: package 'modelr' was built under R version 4.2.3
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 4.2.3
```

```
## Loading required package: Matrix
```

```
## Warning: package 'Matrix' was built under R version 4.2.3
```

```
##
## Attaching package: 'Matrix'
##
## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack
##
## Loaded glmnet 4.1-8
```

```
library(recipes)
```

```
## Warning: package 'recipes' was built under R version 4.2.3
```

```
##
## Attaching package: 'recipes'
##
## The following object is masked from 'package:Matrix':
##
##     update
##
## The following object is masked from 'package:stringr':
##
##     fixed
##
## The following object is masked from 'package:stats':
##
##     step
```

```
library(randomForest)
```

```
## Warning: package 'randomForest' was built under R version 4.2.3
```

```
## randomForest 4.7-1.1
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:gridExtra':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
library(FactoMineR)
```

```
## Warning: package 'FactoMineR' was built under R version 4.2.3
```

```
library(factoextra)
```

```
## Warning: package 'factoextra' was built under R version 4.2.3
```

```
## Welcome! Want to learn more? See two factoextra-related books at https://goo.gl/ve3WBa
```

```r
library(Metrics)
```

```
## Warning: package 'Metrics' was built under R version 4.2.3
```

```
##
## Attaching package: 'Metrics'
##
## The following objects are masked from 'package:modelr':
##
##     mae, mape, mse, rmse
##
## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
df_j <- read.csv("D:/FALL 23/CSP 571/Project/2020-Jan.csv")
df_j <- replace(df_j, df_j=='', NA)
#print(df_j)
head(df_j)
```

	event_time <chr>	event_type <chr>	product_id <int>	category_id <dbl>	category_code <chr>	brand <chr>
1	2020-01-01 00:00:00 UTC	view	5809910	1.602944e+18	NA	grattol
2	2020-01-01 00:00:09 UTC	view	5812943	1.487580e+18	NA	kinetics
3	2020-01-01 00:00:19 UTC	view	5798924	1.783999e+18	NA	zinger
4	2020-01-01 00:00:24 UTC	view	5793052	1.487580e+18	NA	NA
5	2020-01-01 00:00:25 UTC	view	5899926	2.115334e+18	NA	NA
6	2020-01-01 00:00:30 UTC	view	5837111	1.783999e+18	NA	staleks

6 rows | 1-8 of 10 columns

```r
newdf_j <- na.omit(df_j)
```

#———————————————————————————————————— # Summary Statistics: #————————————————————————————————————

```
price_col <- newdf_j$price

# Mean
mu_of_price <- mean(price_col)

# Median
median_of_price <- median(price_col)

# Mode (using a custom function)
mode_of_price <- function(x) {
  ux <- unique(x)
  ux[which.max(tabulate(match(x, ux)))]
}
mode_price_value <- mode_of_price(price_col)

# Range
range_price <- max(price_col) - min(price_col)

# Standard Deviation
stddev_of_price <- sd(price_col)

cat("Mean:", mu_of_price, "\n")
```

```
## Mean: 35.47137
```

```
cat("Median:", median_of_price, "\n")
```

```
## Median: 24.44
```

```
cat("Mode:", mode_price_value, "\n")
```

```
## Mode: 1.98
```

```
cat("Range:", range_price, "\n")
```

```
## Range: 149.85
```

```
cat("Standard Deviation:", stddev_of_price, "\n")
```

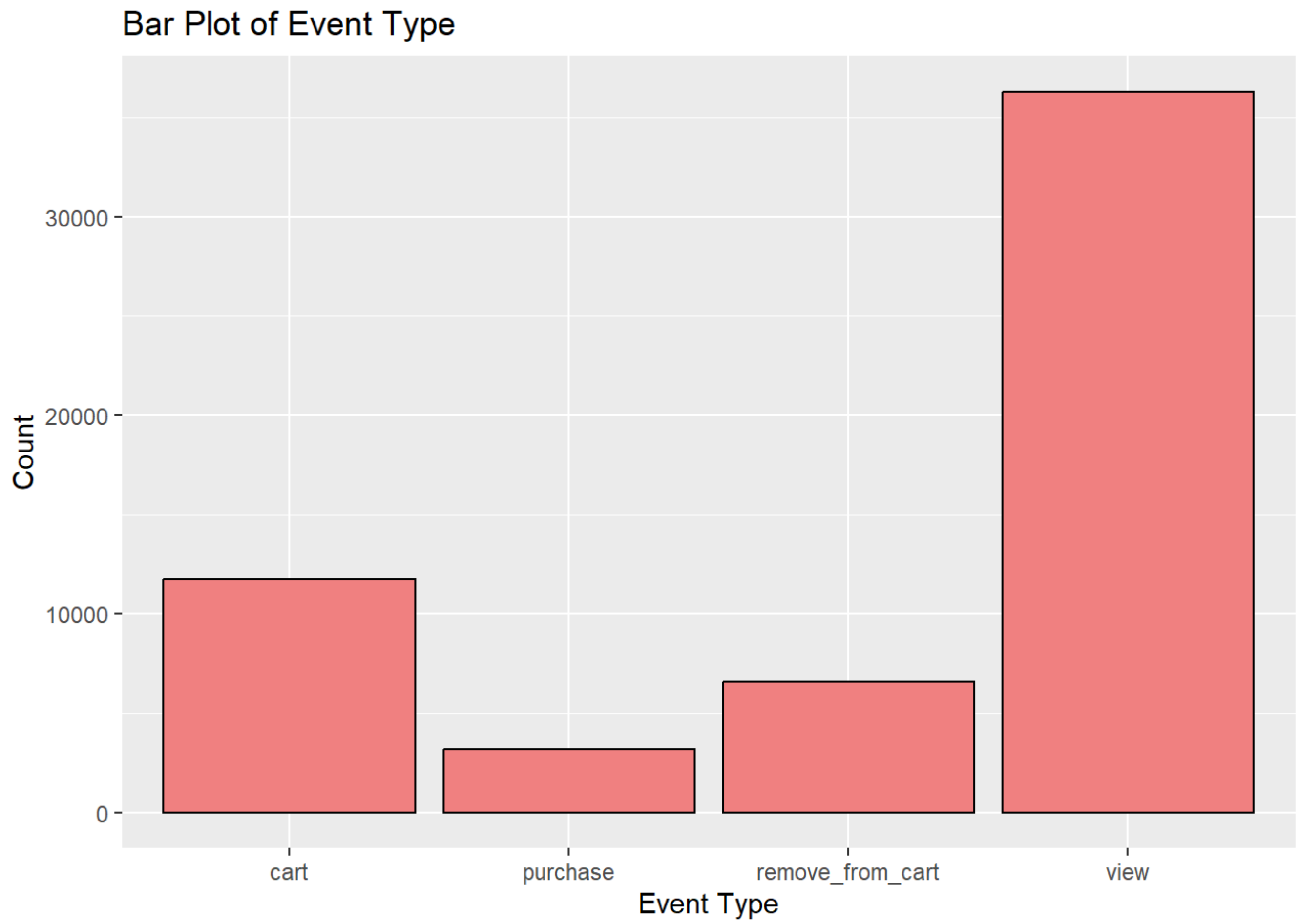
```
## Standard Deviation: 33.68141
```

#———————————————————————————————————— # Univariate Analysis: #————————————————————————————————————

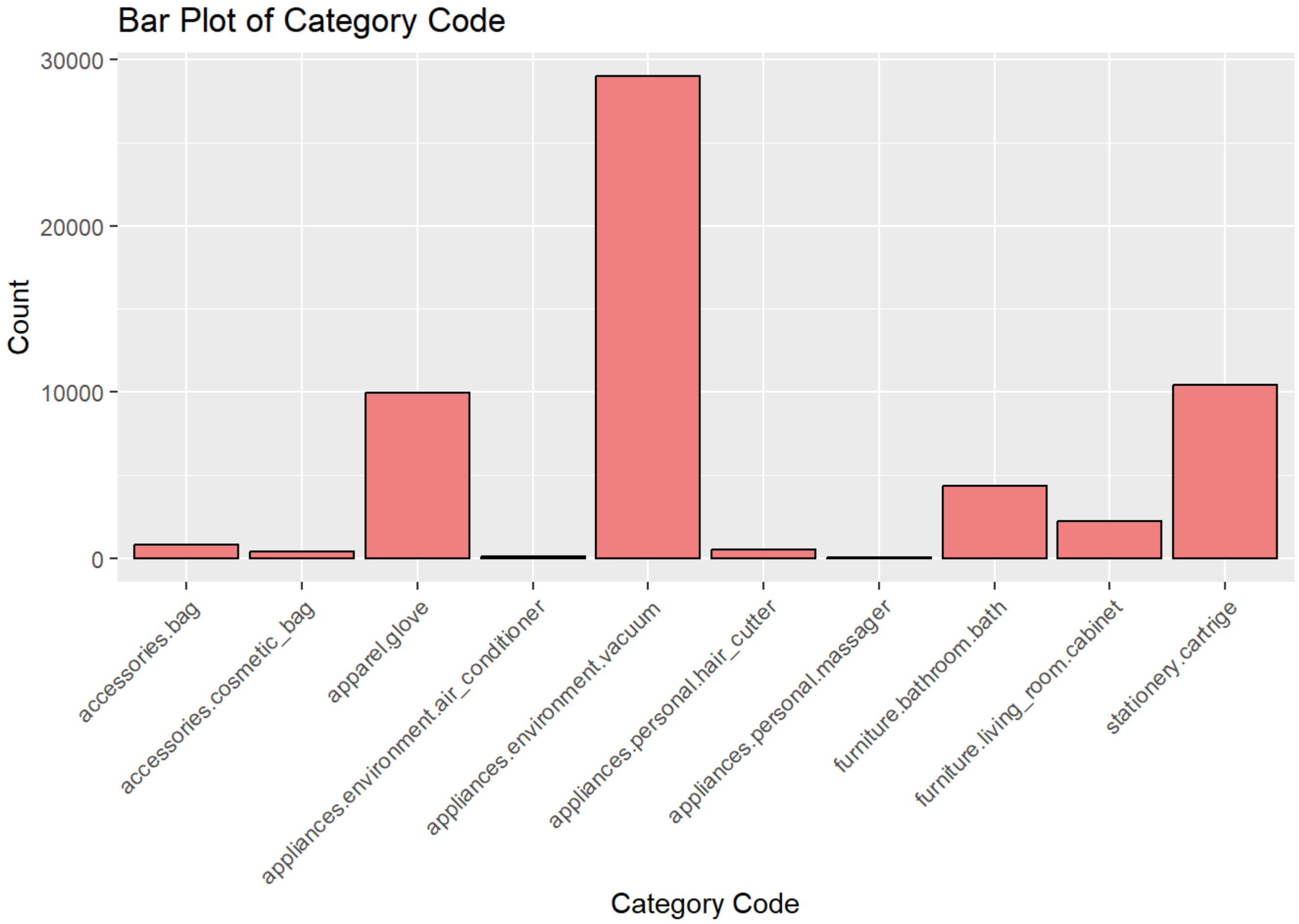
```
colnames(newdf_j)
```

```
## [1] "event_time"    "event_type"    "product_id"    "category_id"
## [5] "category_code" "brand"         "price"         "user_id"
## [9] "user_session"
```

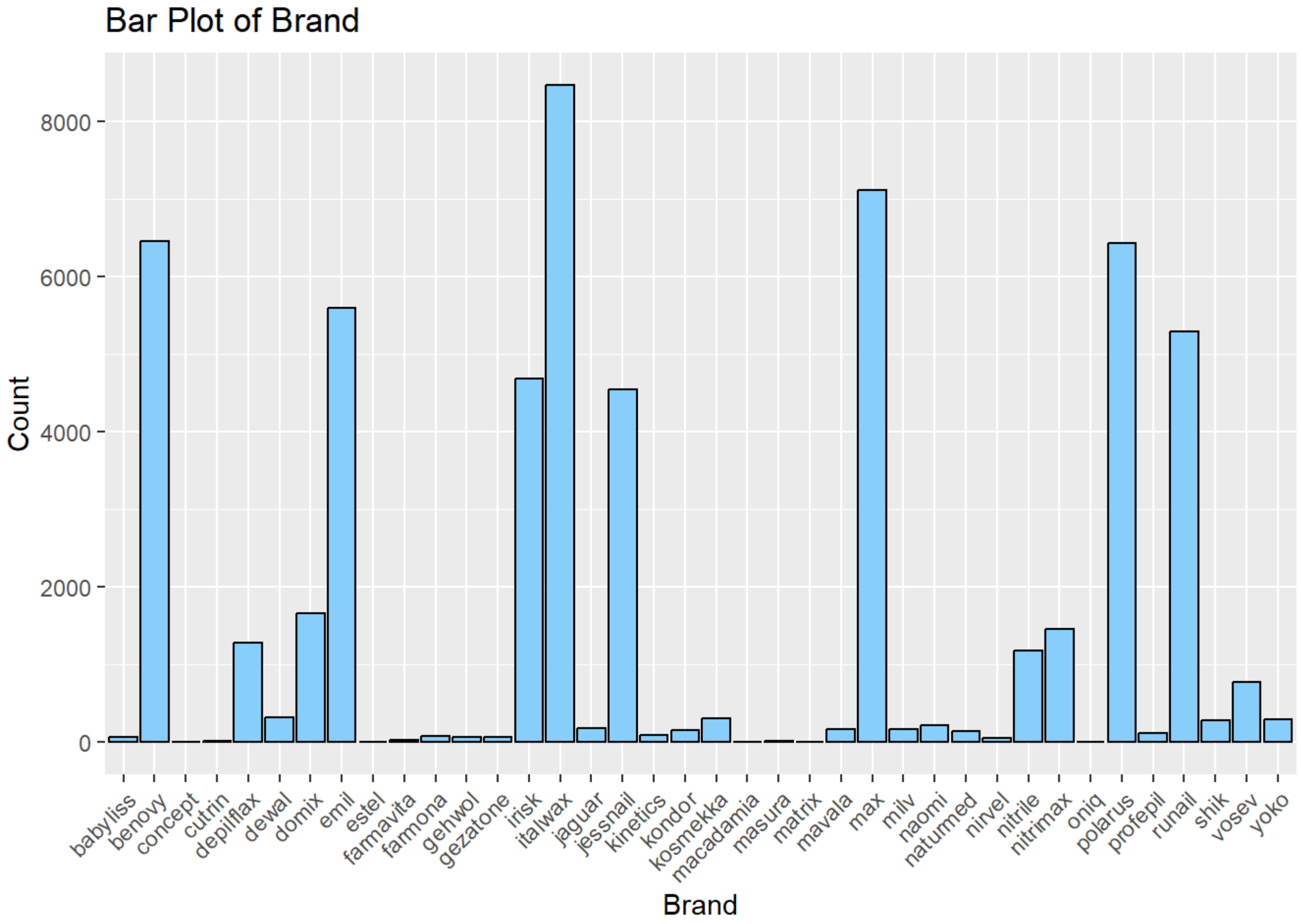
```
# Bar plot for 'event_type'
ggplot(newdf_j, aes(x = event_type)) +
  geom_bar(fill = "lightcoral", color = "black") +
  labs(title = "Bar Plot of Event Type", x = "Event Type", y = "Count")
```



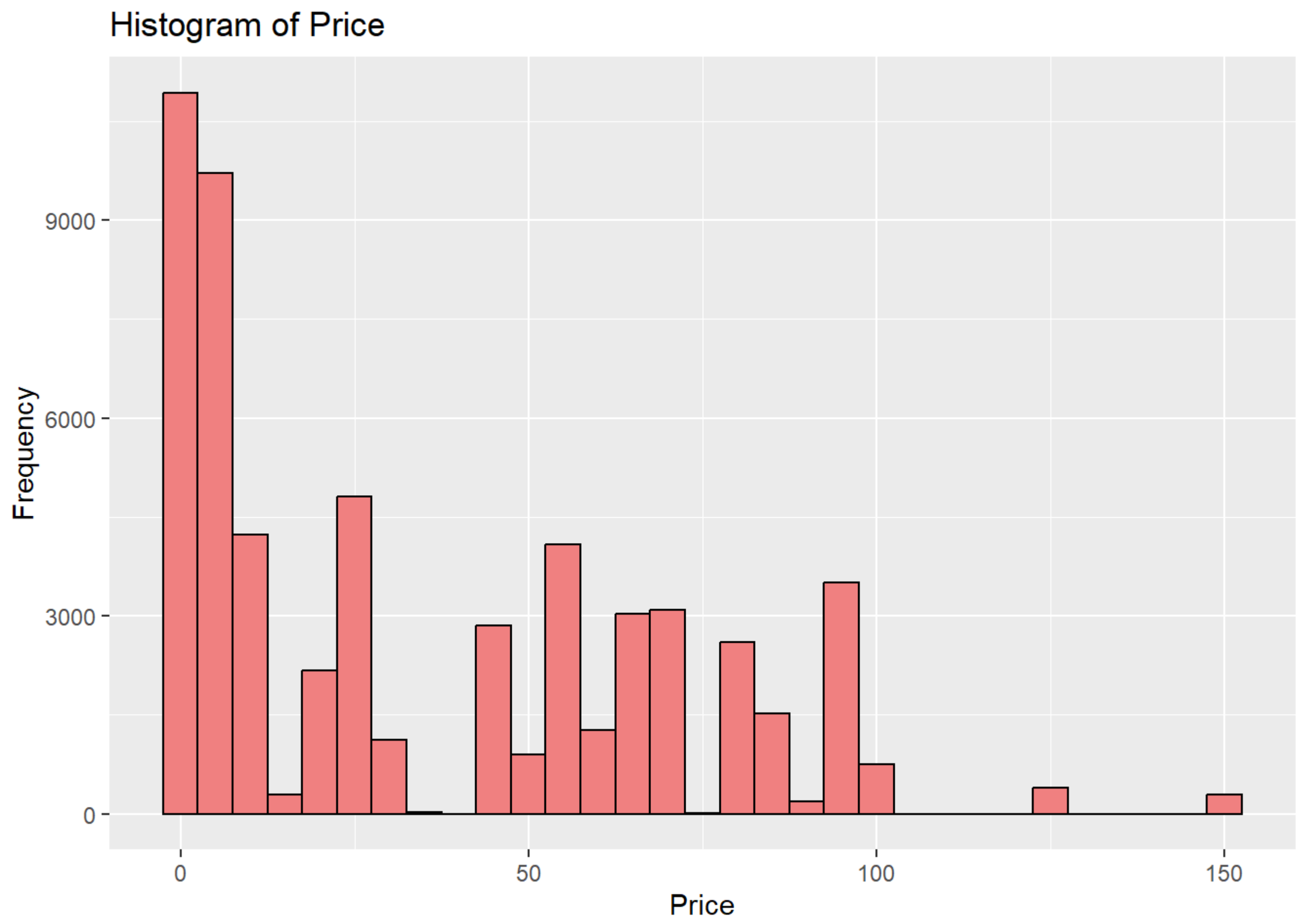

```
# Bar plot for 'category_code'
ggplot(newdf_j, aes(x = category_code)) +
  geom_bar(fill = "lightcoral", color = "black") +
  labs(title = "Bar Plot of Category Code", x = "Category Code", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Bar plot for 'brand'
ggplot(newdf_j, aes(x = brand)) +
  geom_bar(fill = "lightskyblue", color = "black") +
  labs(title = "Bar Plot of Brand", x = "Brand", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

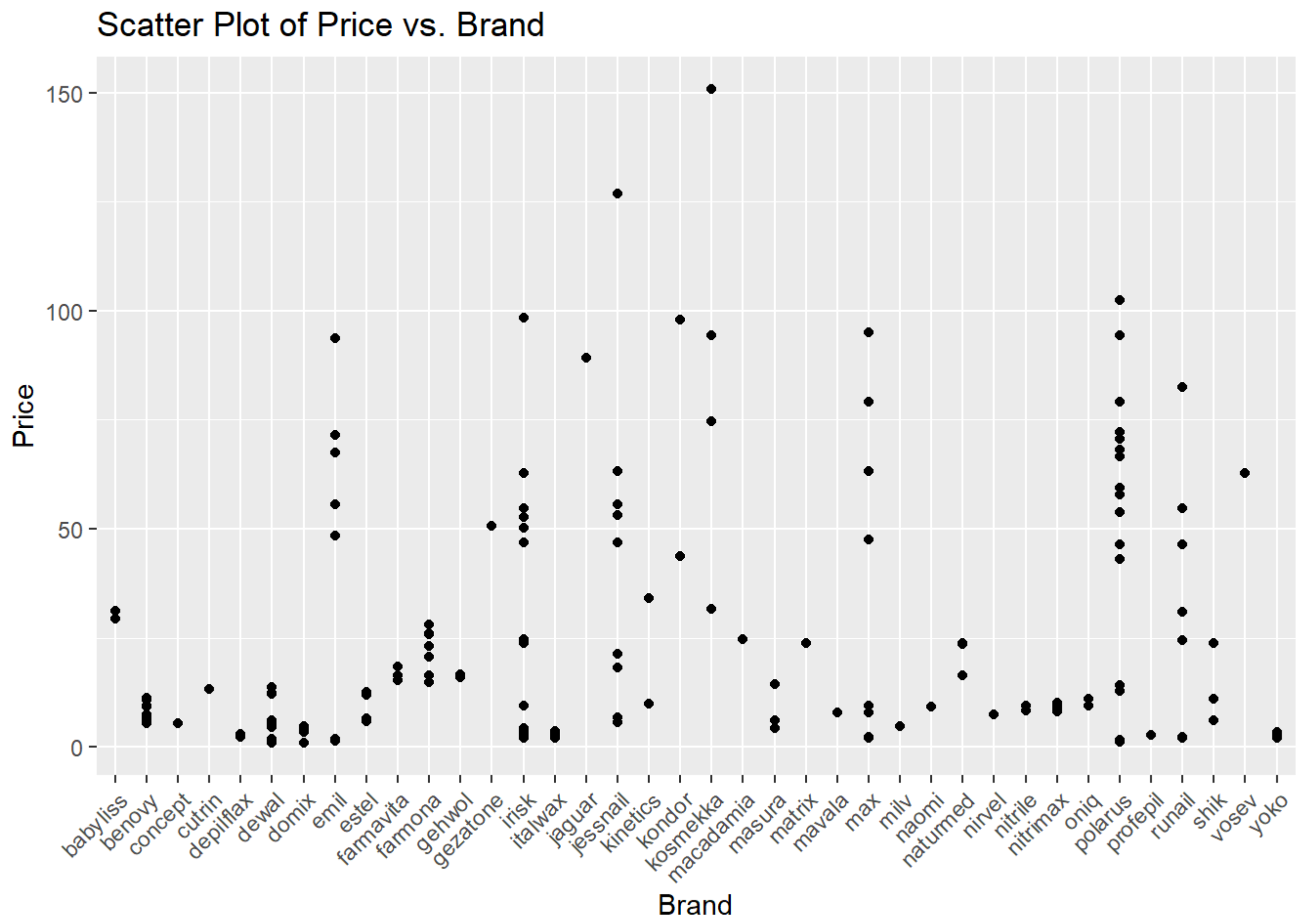



```
# Histogram for 'price'
ggplot(newdf_j, aes(x = price)) +
  geom_histogram(binwidth = 5, fill = "lightcoral", color = "black") +
  labs(title = "Histogram of Price", x = "Price", y = "Frequency")
```



——————————————————————————— # Bivariate Analysis
———————————————————————————

```
ggplot(newdf_j, aes(x = brand, y = price)) +
  geom_point() +
  labs(title = "Scatter Plot of Price vs. Brand", x = "Brand", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



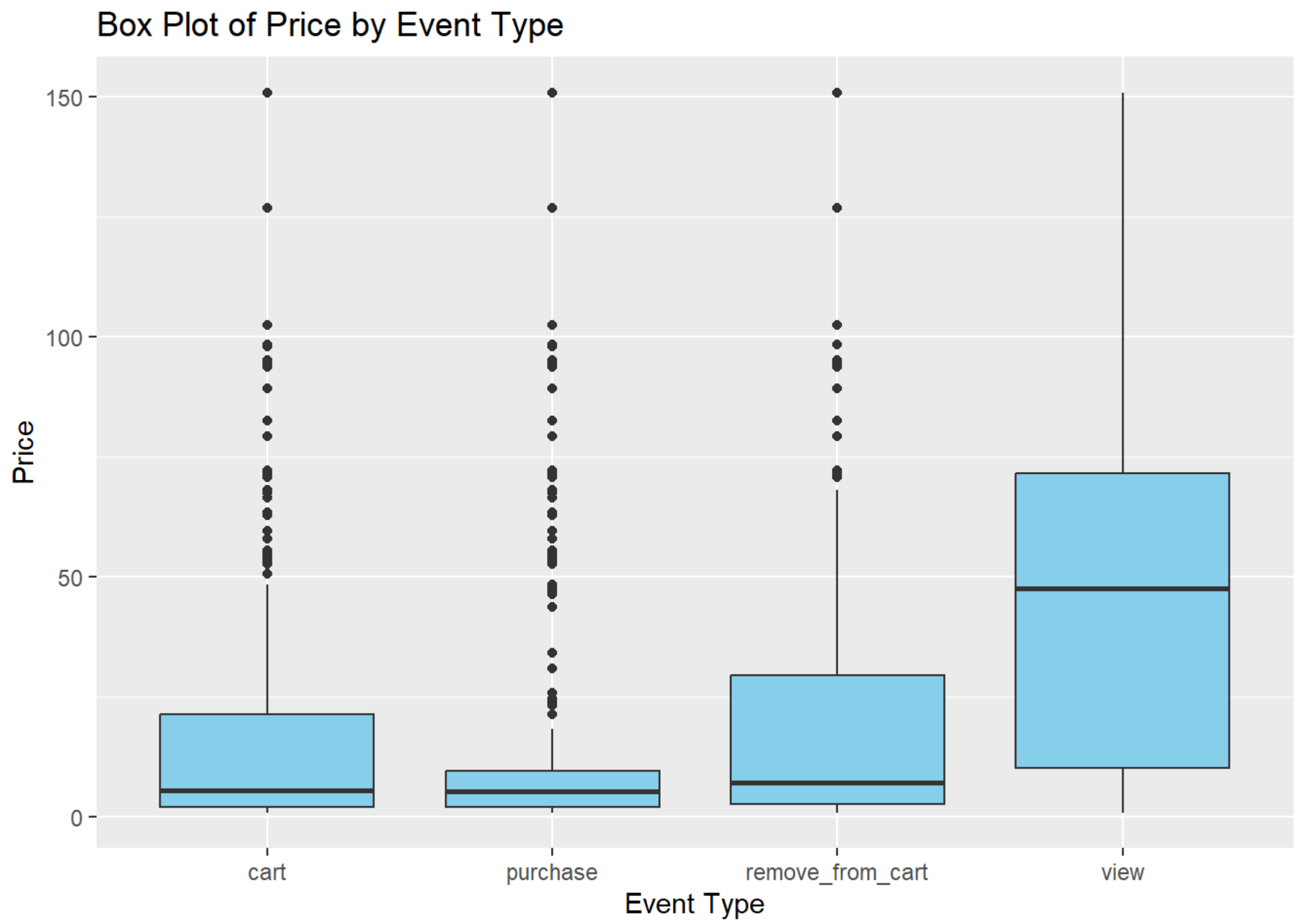

```
cor(newdf_j$price, newdf_j$category_id)
```

```
## [1] -0.009816899
```

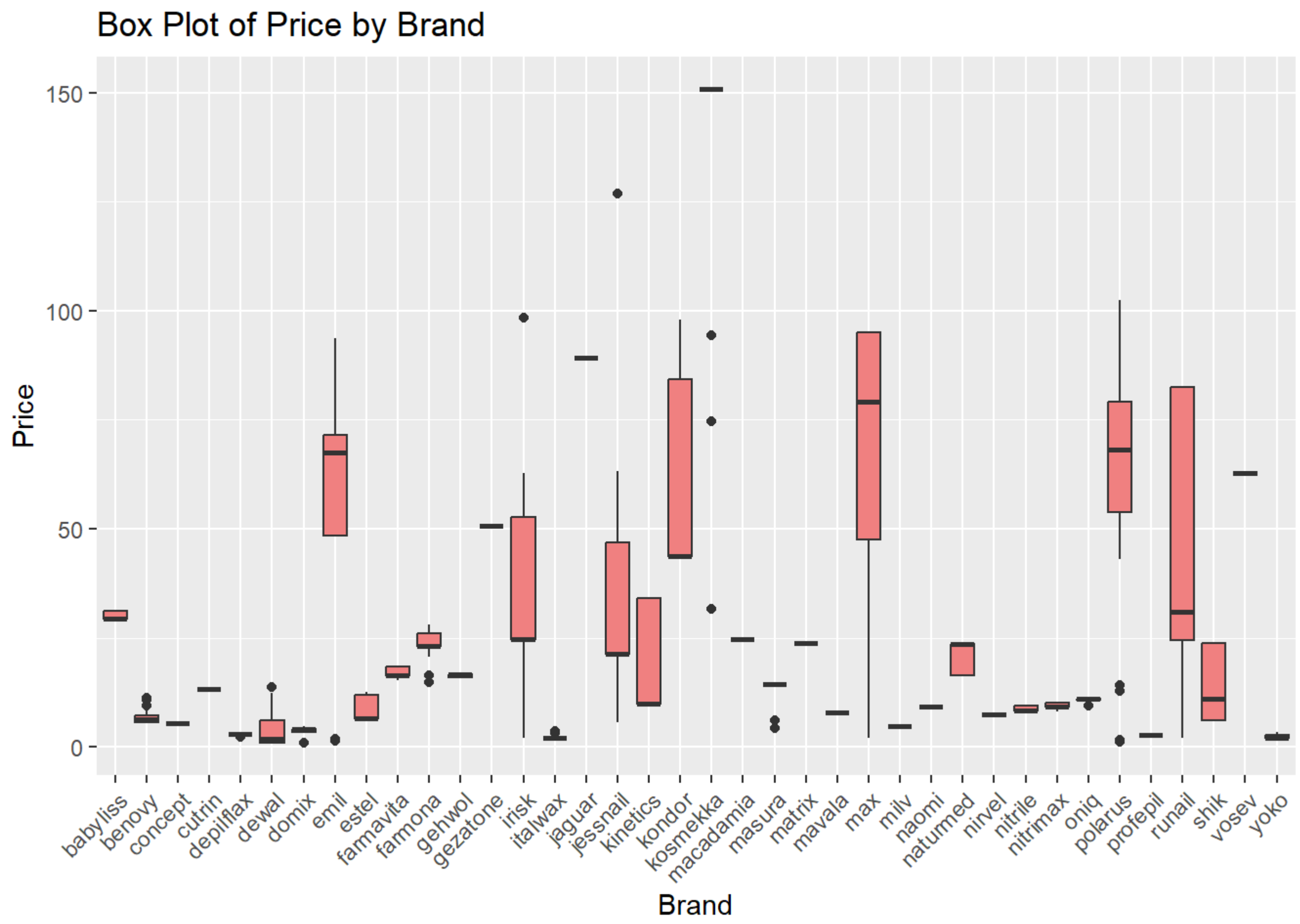
```
cor(newdf_j[, c("price", "category_id")])
```

```
##                    price  category_id
## price        1.000000000 -0.009816899
## category_id -0.009816899  1.000000000
```

```
ggplot(newdf_j, aes(x = event_type, y = price)) +
  geom_boxplot(fill = "skyblue") +
  labs(title = "Box Plot of Price by Event Type", x = "Event Type", y = "Price")
```




```
ggplot(newdf_j, aes(x = brand, y = price)) +
  geom_boxplot(fill = "lightcoral") +
  labs(title = "Box Plot of Price by Brand", x = "Brand", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
cor(newdf_j$price, newdf_j$category_id)
```

```
## [1] -0.009816899
```

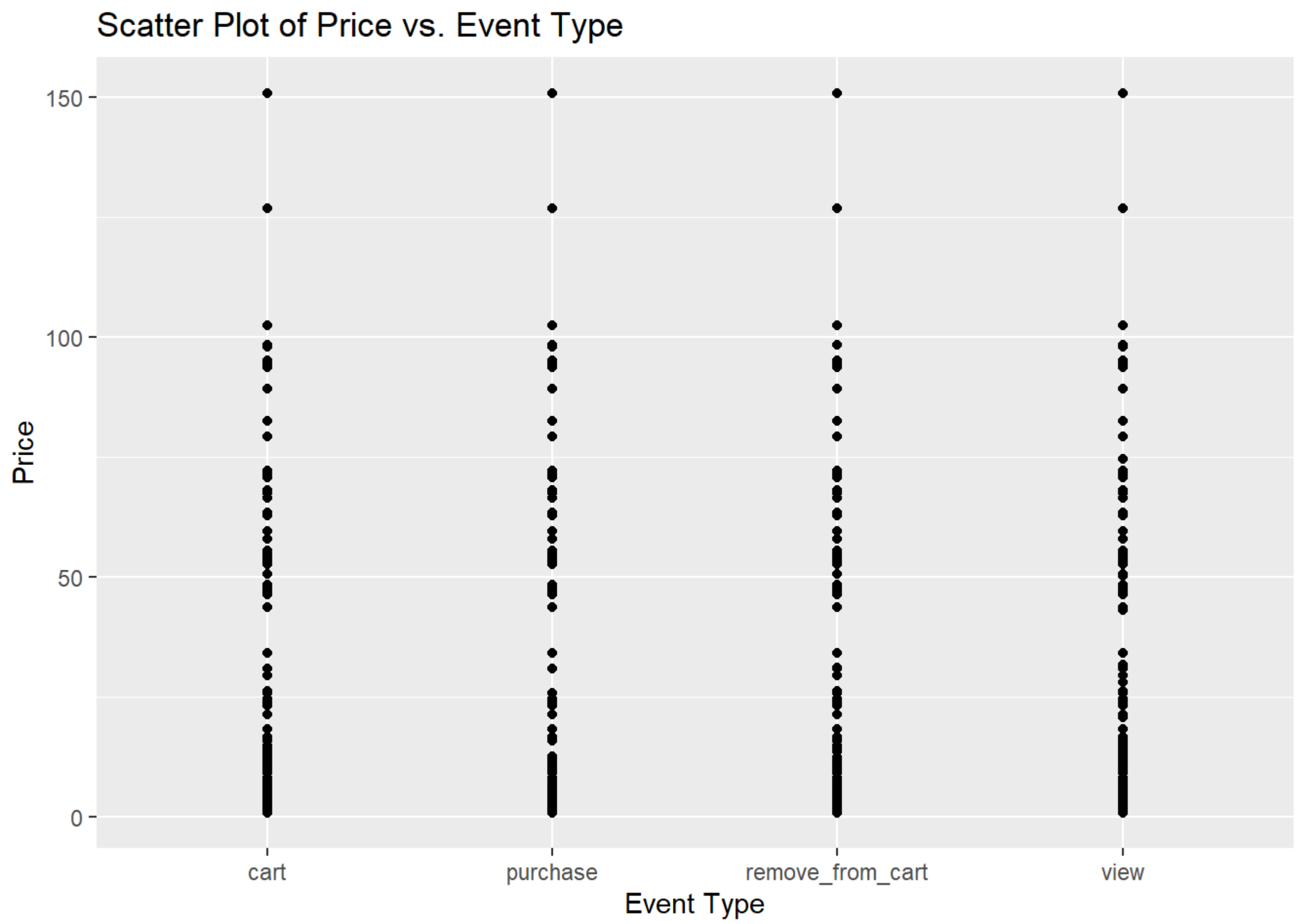
```
numericvar <- newdf_j[, c("price", "category_id")]

cor_matrix <- cor(numericvar)
print(cor_matrix)
```

```
##                    price  category_id
## price        1.000000000 -0.009816899
## category_id -0.009816899  1.000000000
```

```
# Select numeric variables for the heatmap
numericvar <- newdf_j[, c("price", "category_id")]

# Calculate the correlation matrix
cor_matrix <- cor(numericvar)
```

#——————————————————————————— # Outlier Detection:
#———————————————————————————

```
# Scatter plot for Price vs. Event Type
ggplot(newdf_j, aes(x = event_type, y = price)) +
  geom_point() +
  labs(title = "Scatter Plot of Price vs. Event Type", x = "Event Type", y = "Price")
```

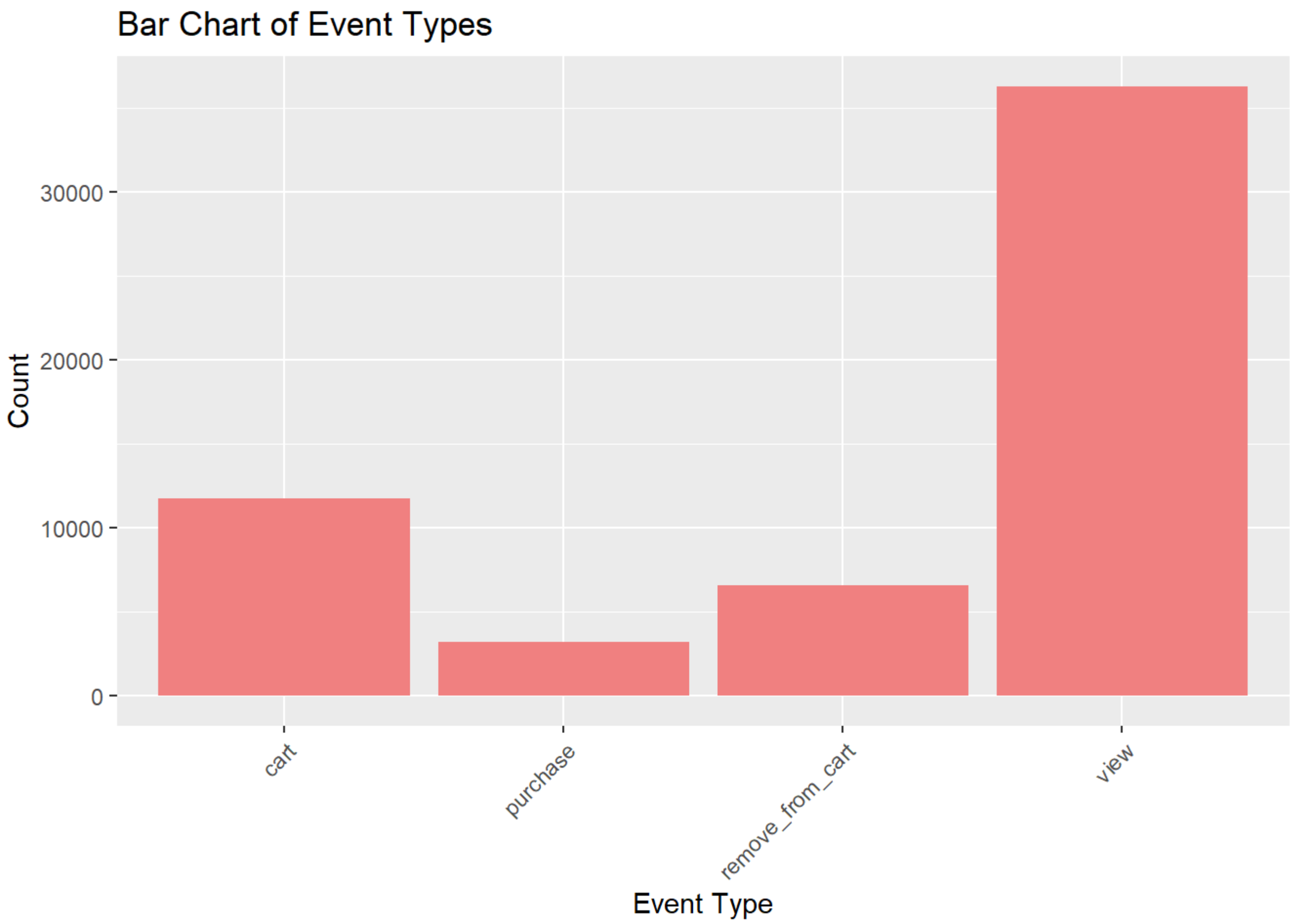



```
# Calculate IQR for Price
price_iqr <- IQR(newdf_j$price)

# Set a threshold for outliers
outlier_threshold <- 1.5 * price_iqr

# Identify outliers
outliers <- newdf_j[newdf_j$price > quantile(newdf_j$price)[4] + outlier_threshold |
                      newdf_j$price < quantile(newdf_j$price)[2] - outlier_threshold, ]

# Display the outliers
print(outliers)
```

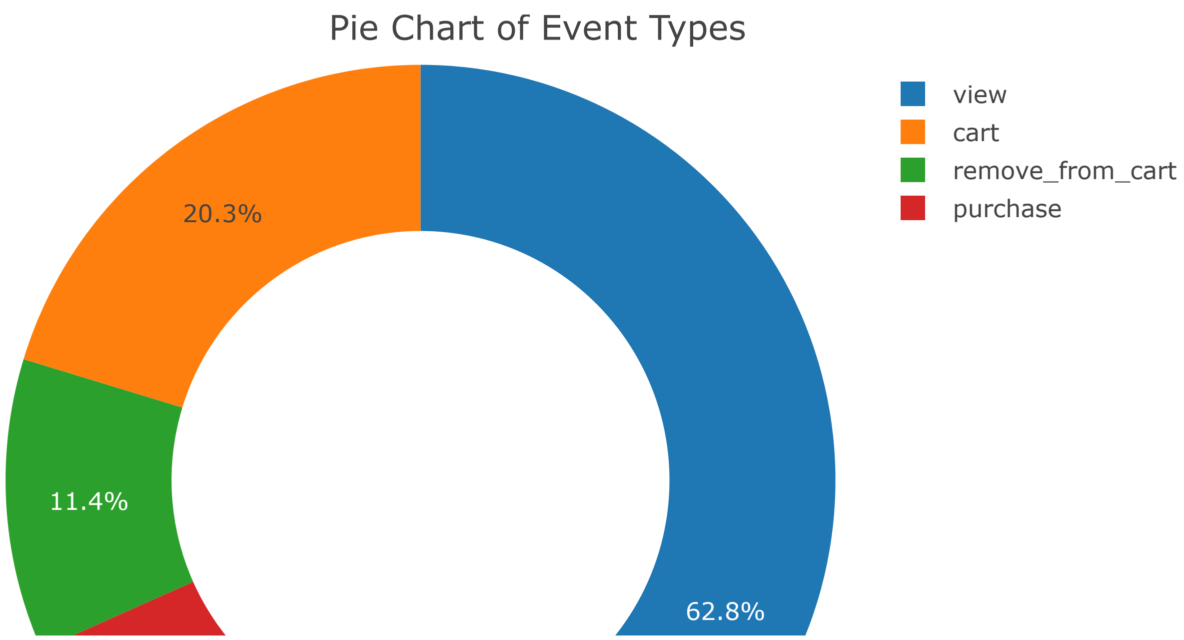
```
## [1] event_time    event_type    product_id    category_id   category_code
## [6] brand         price         user_id       user_session
## <0 rows> (or 0-length row.names)
```

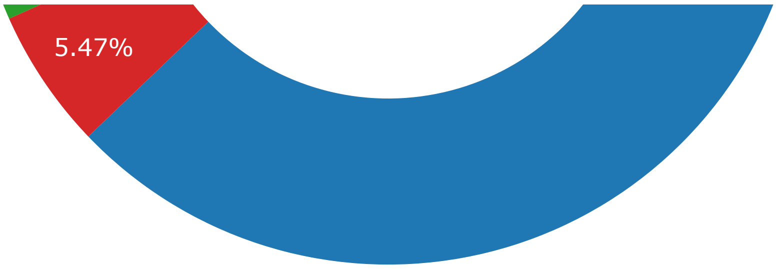
—————————————————————————— # Visualization: # ——————————————————————————

```
# Bar chart for Event Types
ggplot(newdf_j, aes(x = event_type)) +
  geom_bar(fill = "lightcoral") +
  labs(title = "Bar Chart of Event Types", x = "Event Type", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

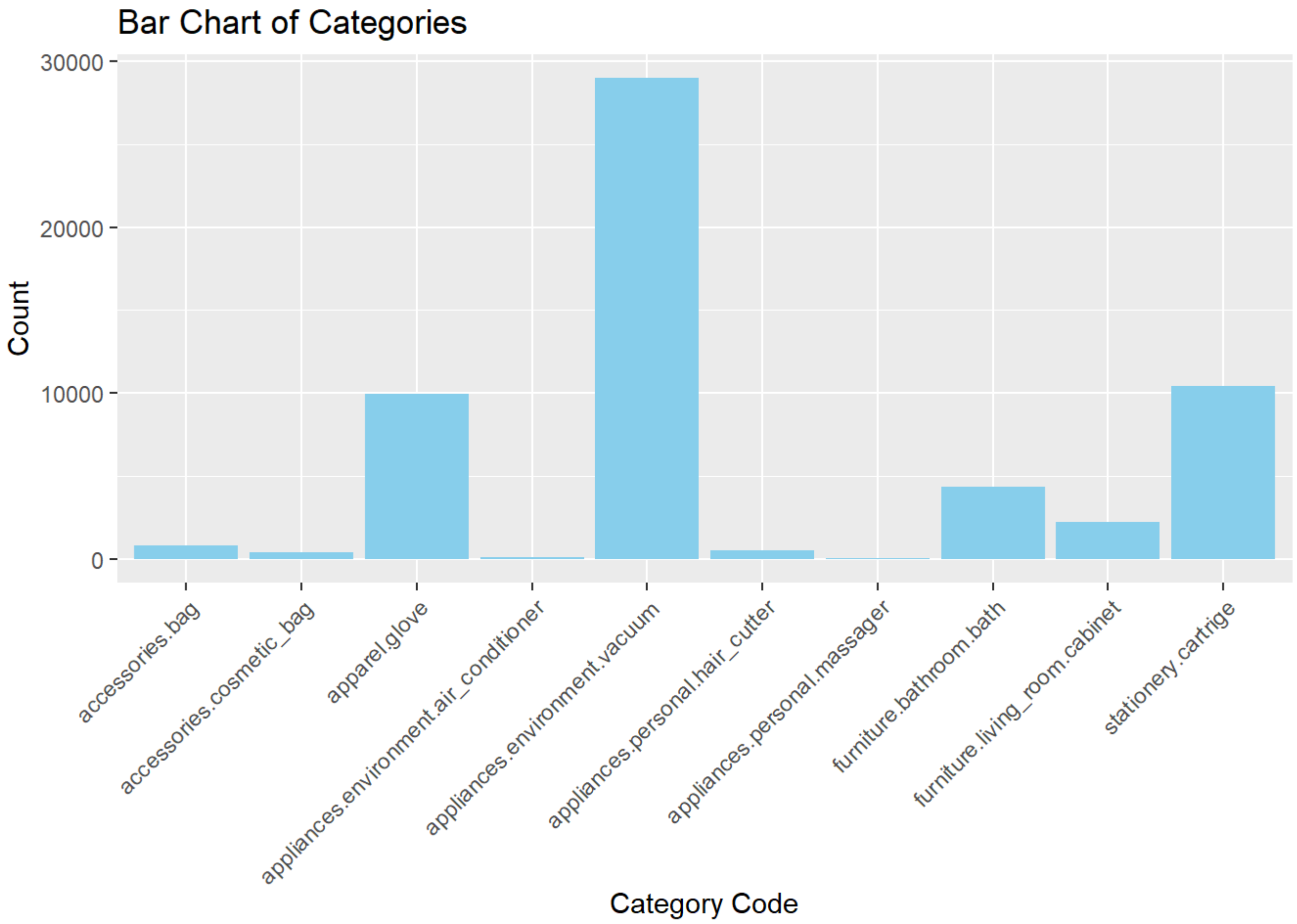



```
# Pie chart for Event Types
event_type_counts <- table(newdf_j$event_type)
plot_ly(labels = names(event_type_counts), values = event_type_counts, type = "pie", hole = 
0.6) %>%
  layout(title = "Pie Chart of Event Types")
```

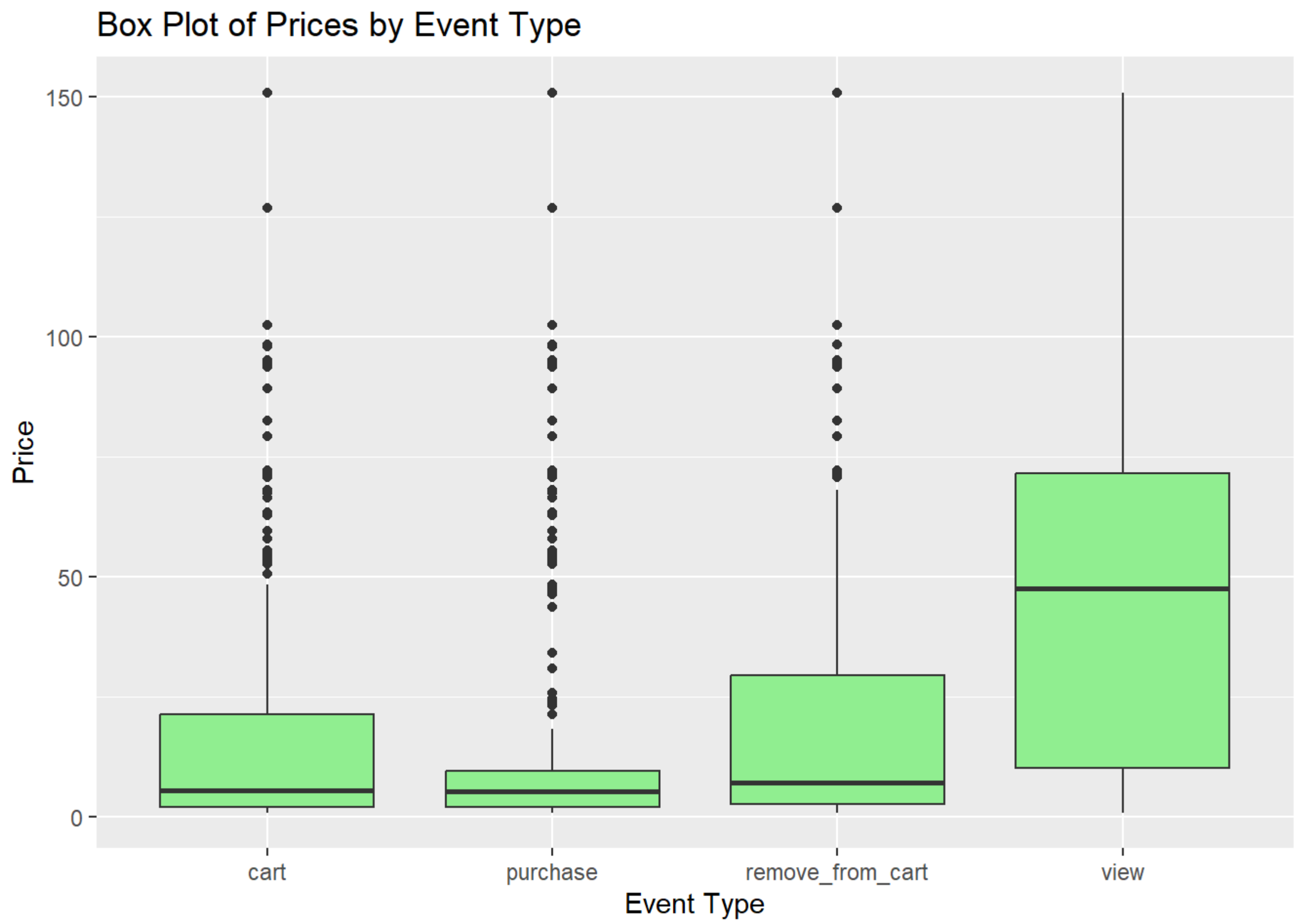





```
# Bar chart for Categories
ggplot(newdf_j, aes(x = category_code)) +
  geom_bar(fill = "skyblue") +
  labs(title = "Bar Chart of Categories", x = "Category Code", y = "Count") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

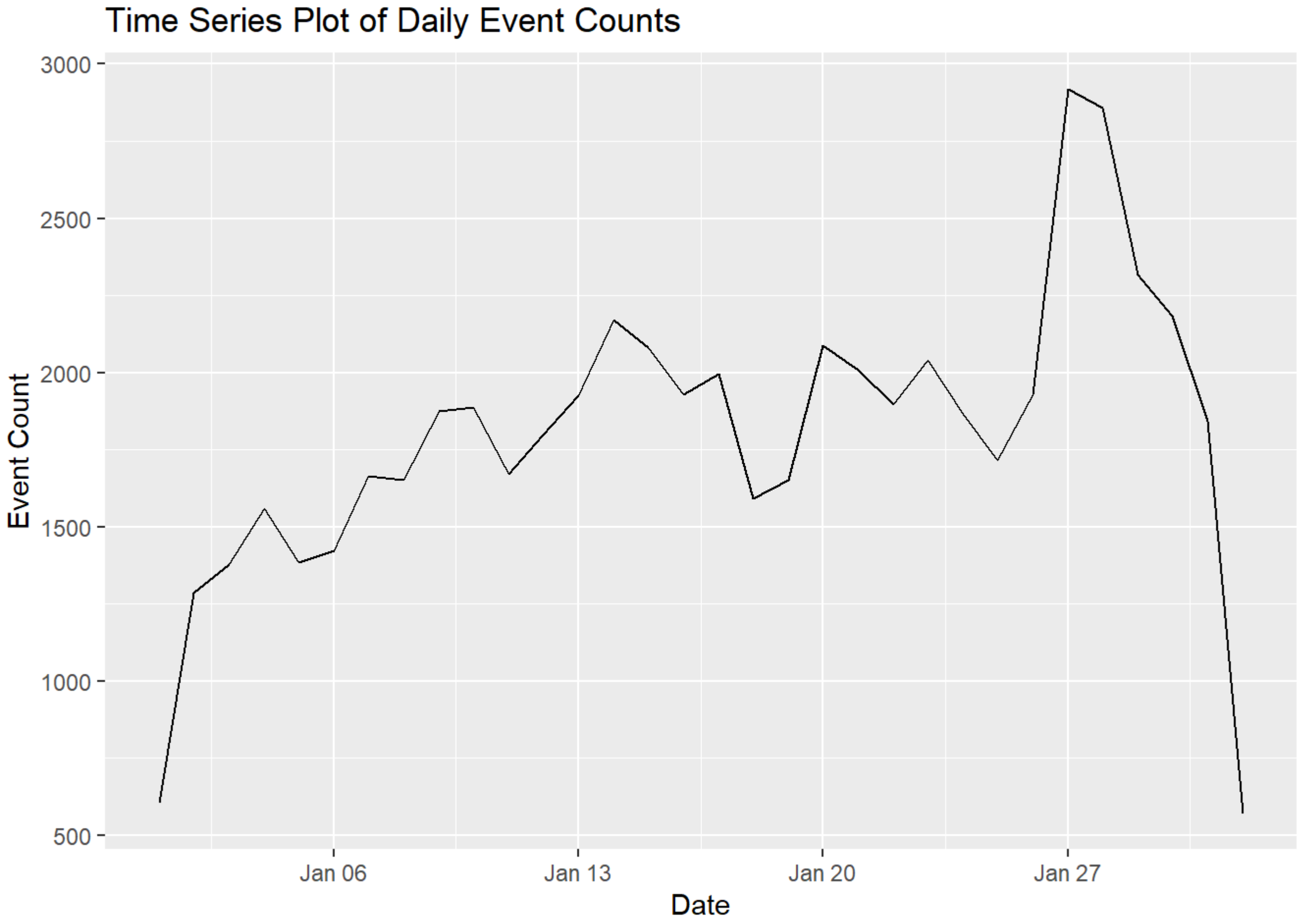



```
# Box plot for Prices by Event Type
ggplot(newdf_j, aes(x = event_type, y = price)) +
  geom_boxplot(fill = "lightgreen") +
  labs(title = "Box Plot of Prices by Event Type", x = "Event Type", y = "Price")
```

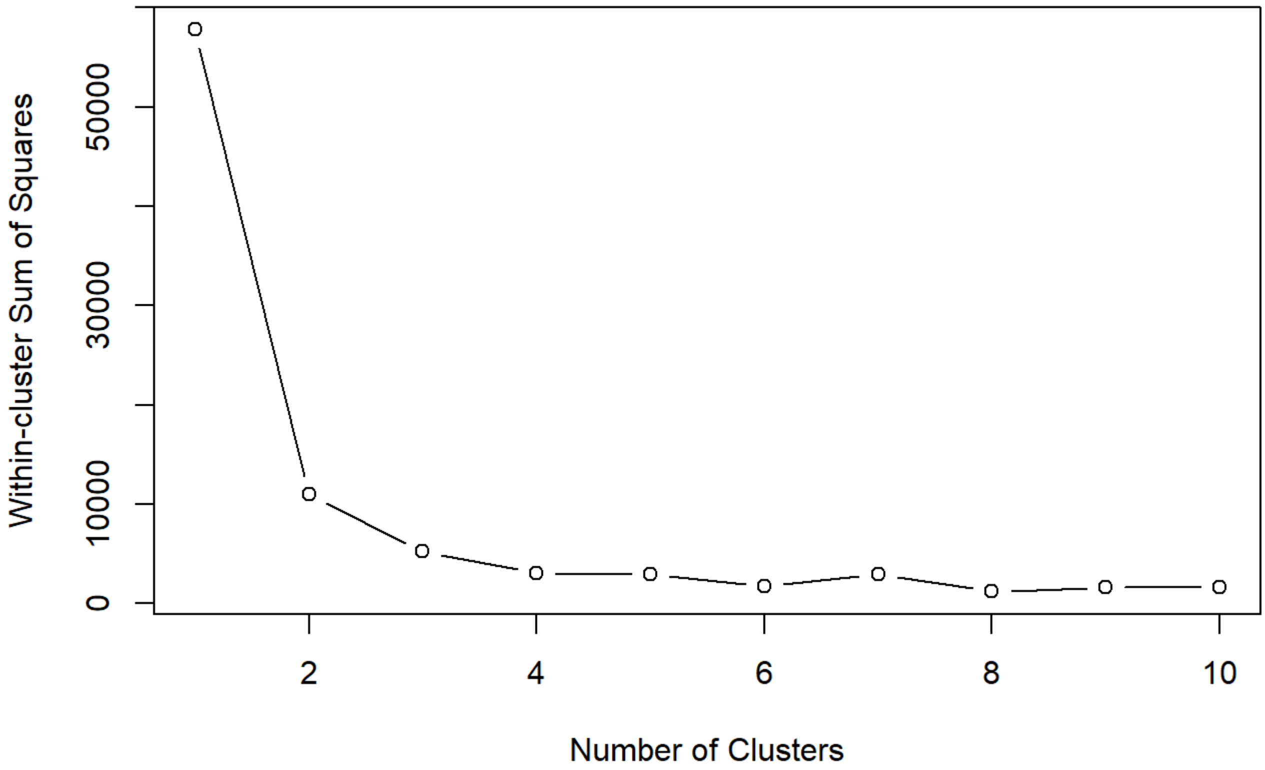


———————————————————————————————— # Time Series Analysis, Clustering and Association Rules # ————————————————————————————————

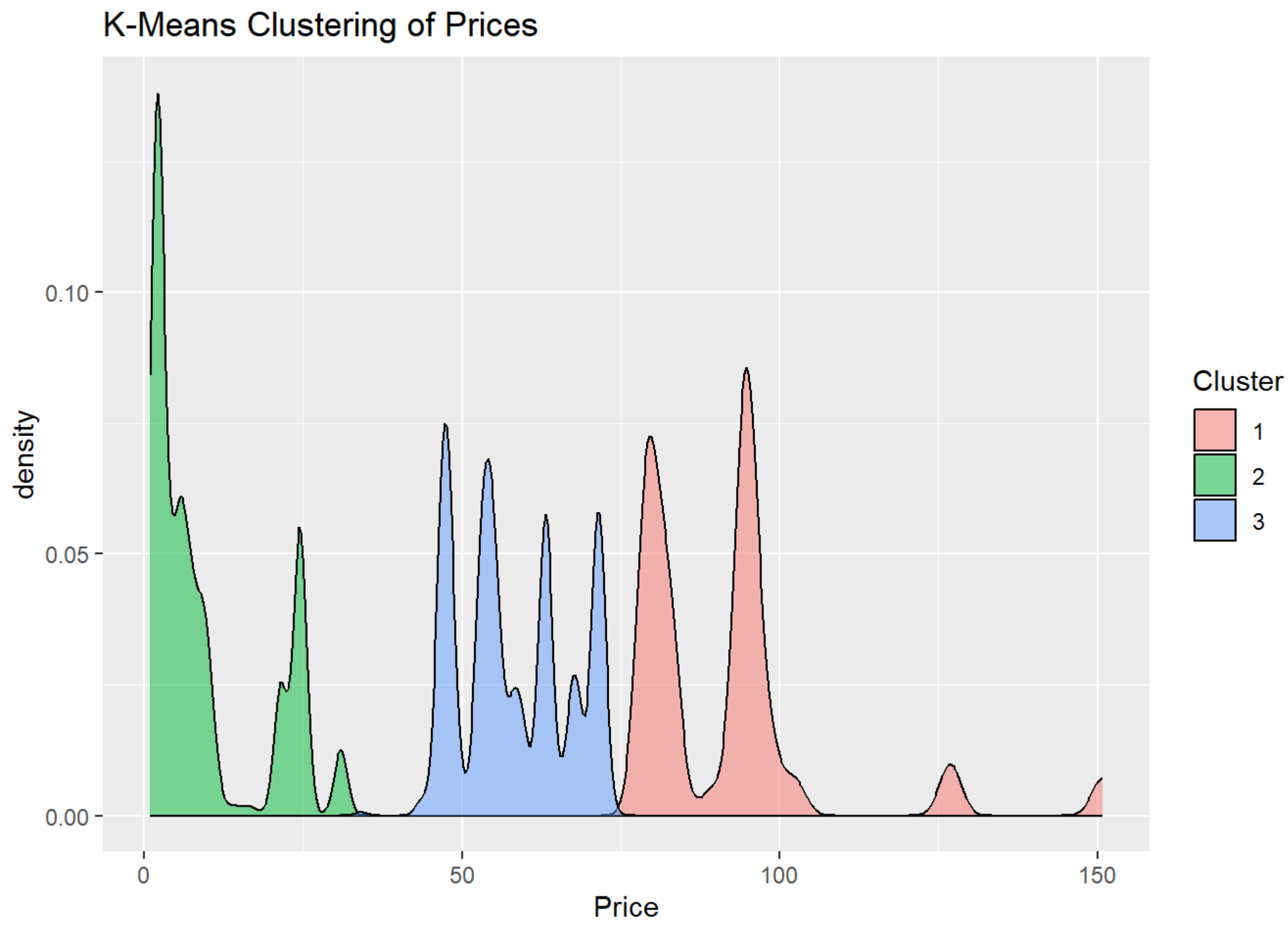
```
# Convert event_time to POSIXct
newdf_j$event_time <- as.POSIXct(newdf_j$event_time)

# Aggregate events by day
daily_events <- aggregate(event_type ~ as.Date(event_time), data = newdf_j, FUN = length)

# Plot time series
ggplot(daily_events, aes(x = `as.Date(event_time)`, y = event_type)) +
  geom_line() +
  labs(title = "Time Series Plot of Daily Event Counts", x = "Date", y = "Event Count")
```




```
# K-Means clustering on numeric variables
numericvar <- newdf_j[, c("price")]

# Standardize numeric variables
scaled_vars <- scale(numericvar)

# Determine optimal number of clusters using the elbow method
wss <- numeric(10)
for (i in 1:10) wss[i] <- sum(kmeans(scaled_vars, centers = i)$withinss)

# Plot the elbow method
plot(1:10, wss, type = "b", xlab = "Number of Clusters",
     ylab = "Within-cluster Sum of Squares")
```




```
# Choose an appropriate number of clusters and perform k-means clustering
num_clusters <- 3
clusters <- kmeans(scaled_vars, centers = num_clusters)

# Visualize clustering results
ggplot(newdf_j, aes(x = price, fill = as.factor(clusters$cluster))) +
  geom_density(alpha = 0.5) +
  labs(title = "K-Means Clustering of Prices", x = "Price", fill = "Cluster")
```



Frequency Tables for Categorical Variables:

```
# Frequency table for Event Type
event_type_freq <- table(newdf_j$event_type)
print(event_type_freq)
```

```
##
##           cart        purchase remove_from_cart             view
##          11725            3160             6577            36306
```

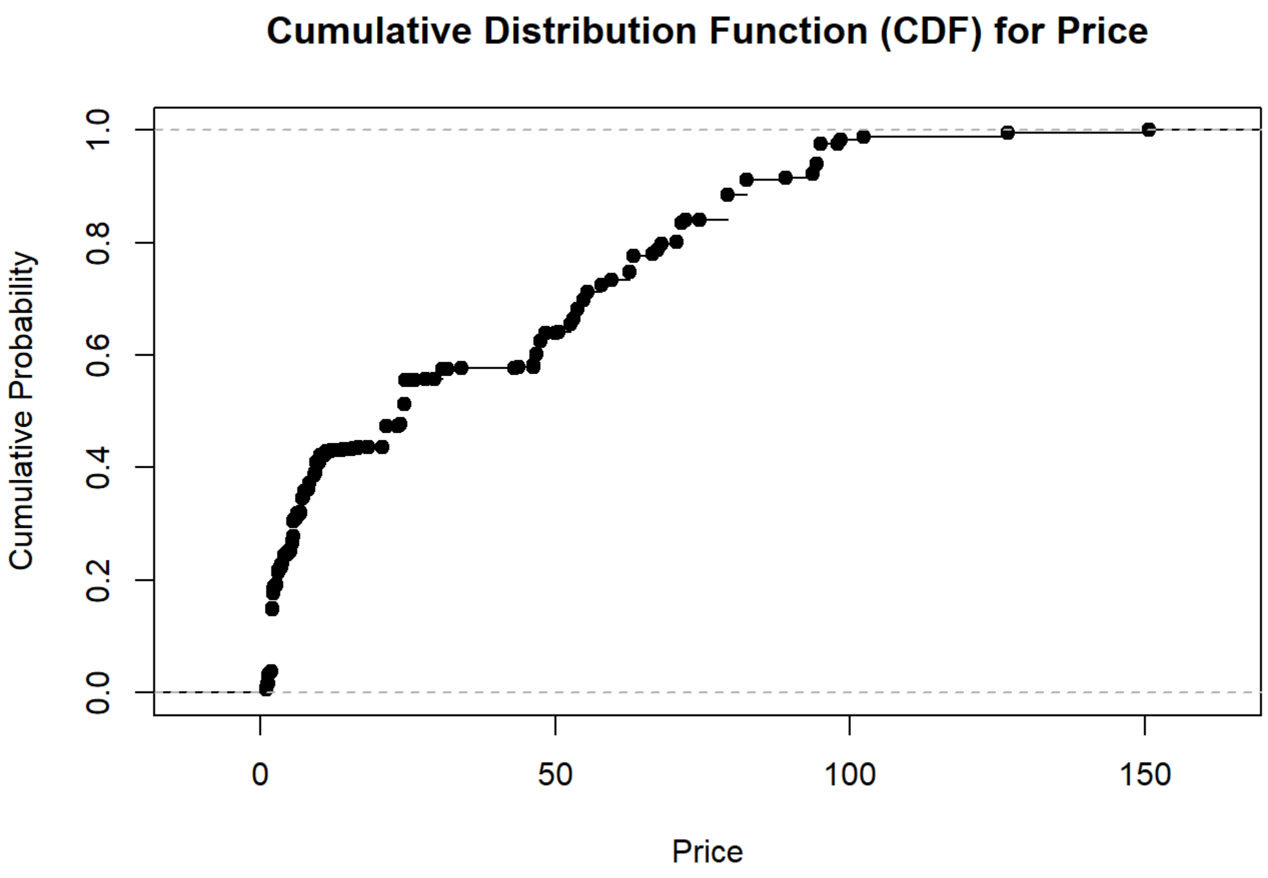
```
# Frequency table for Category Code
category_code_freq <- table(newdf_j$category_code)
print(category_code_freq)
```

```
##
##                  accessories.bag             accessories.cosmetic_bag
##                              780                                  377
##                    apparel.glove appliances.environment.air_conditioner
##                             9958                                  102
##     appliances.environment.vacuum         appliances.personal.hair_cutter
##                            28979                                  517
##      appliances.personal.massager                 furniture.bathroom.bath
##                               64                                 4350
##     furniture.living_room.cabinet                  stationery.cartrige
##                             2249                                10392
```

```
# Summary statistics for Price
summary(newdf_j$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
##    0.94    4.76   24.44   35.47   63.33  150.79
```

```
# CDF for Price
price_sorted <- sort(newdf_j$price)
cdf <- ecdf(price_sorted)
plot(cdf, main = "Cumulative Distribution Function (CDF) for Price", xlab = "Price", ylab = 
"Cumulative Probability")
```



#———————————————————————————— # User and Product Analysis: #————————————————————————————

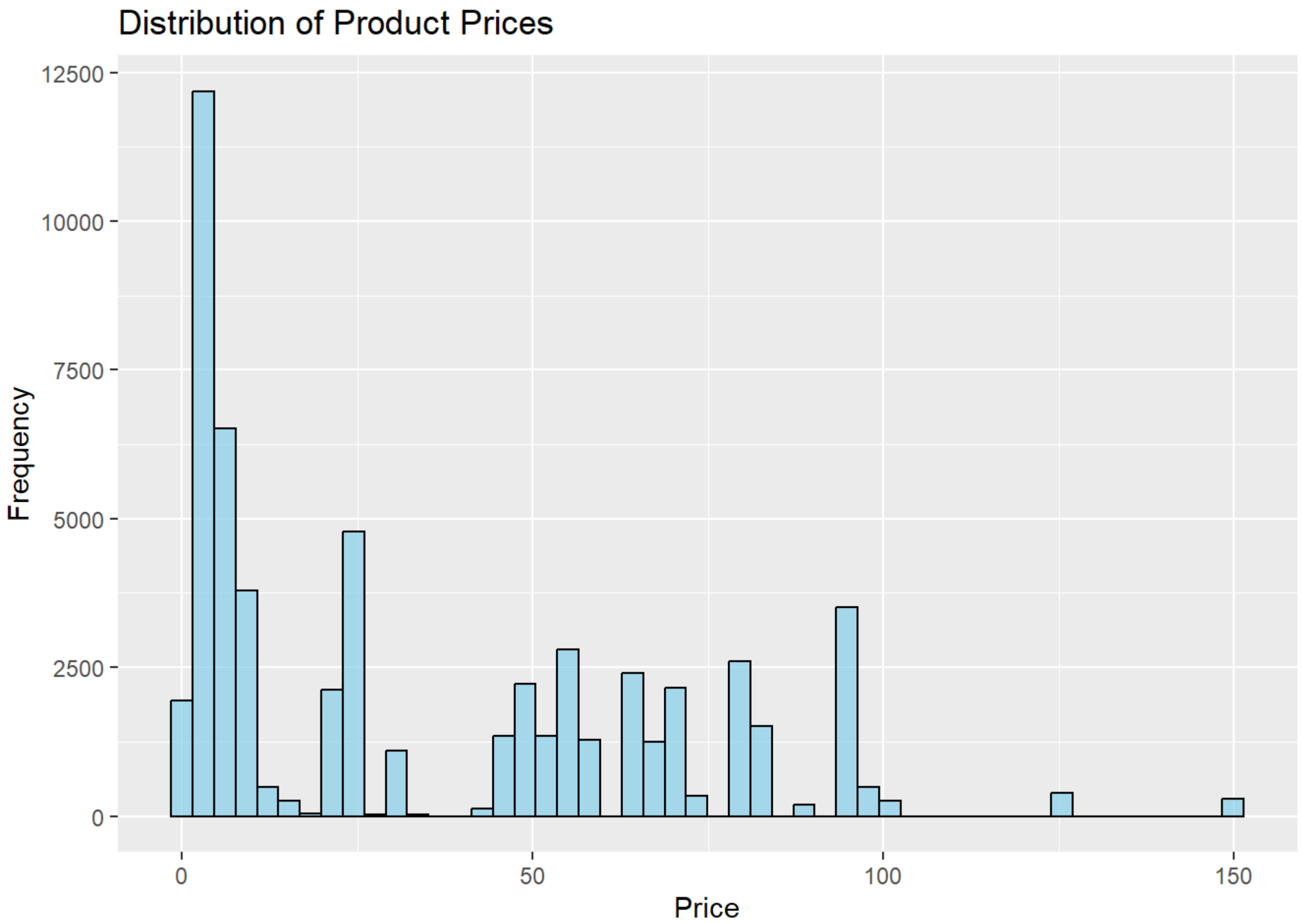
```
# Analyze the number of unique users and products
unique_users <- length(unique(newdf_j$user_id))
unique_products <- length(unique(newdf_j$product_id))

cat("Number of Unique Users: ", unique_users, "\n")
```

```
## Number of Unique Users:  16832
```

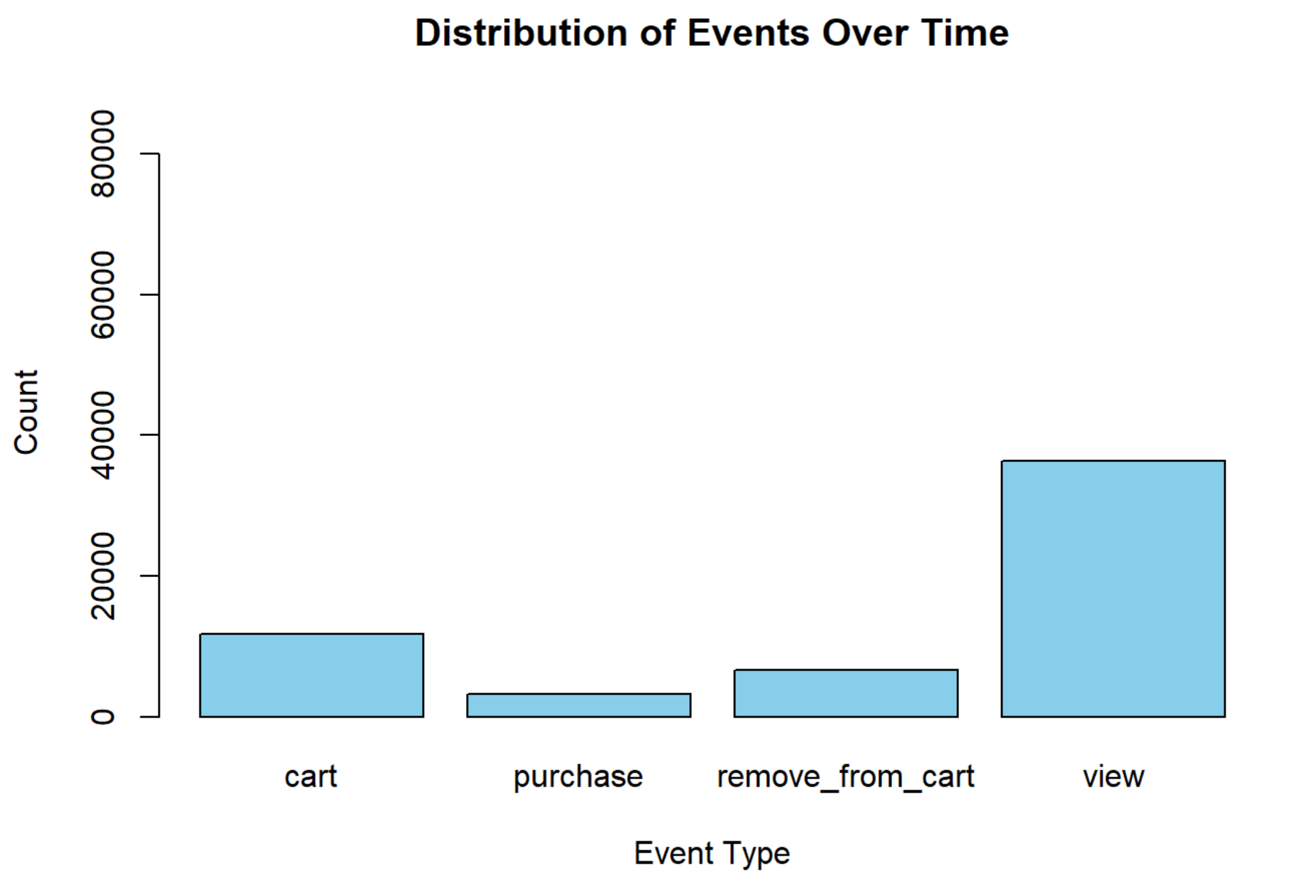
```
cat("Number of Unique Products: ", unique_products, "\n")
```

```
## Number of Unique Products:  306
```

```
# Explore the distribution of product prices
ggplot(newdf_j, aes(x = price)) +
  geom_histogram(bins = 50, fill = "skyblue", color = "black", alpha = 0.7) +
  labs(title = "Distribution of Product Prices", x = "Price", y = "Frequency")
```

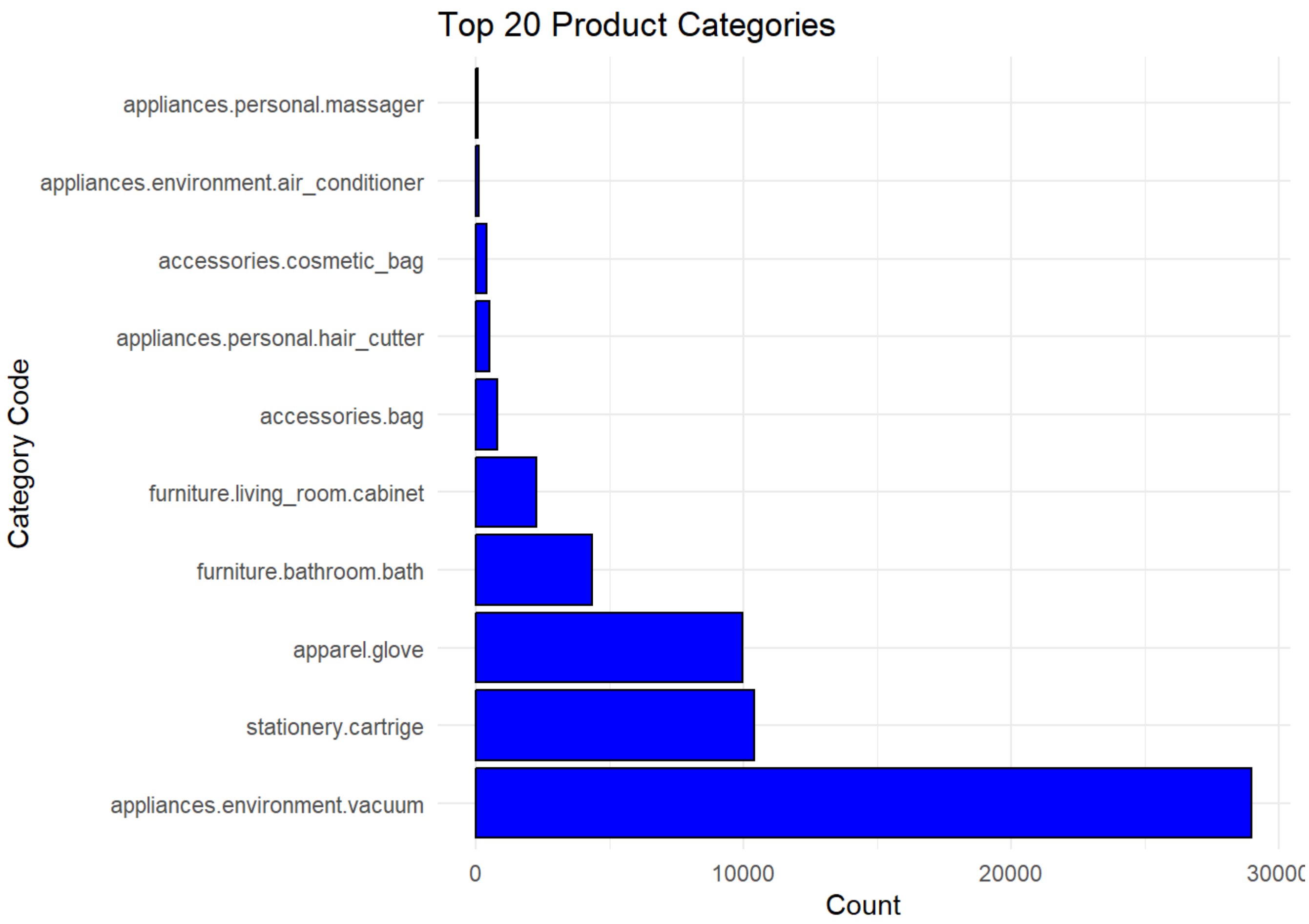



```
# Investigate the distribution of events over time
newdf_j$event_time <- as.POSIXct(newdf_j$event_time, format="%Y-%m-%d %H:%M:%S UTC")
newdf_j$date <- as.Date(newdf_j$event_time)
event_distribution <- table(newdf_j$event_type)
barplot(event_distribution, main = "Distribution of Events Over Time",
        xlab = "Event Type", ylab = "Count", col = "skyblue", border = "black",
        ylim = c(0, max(event_distribution) + 50000))
```

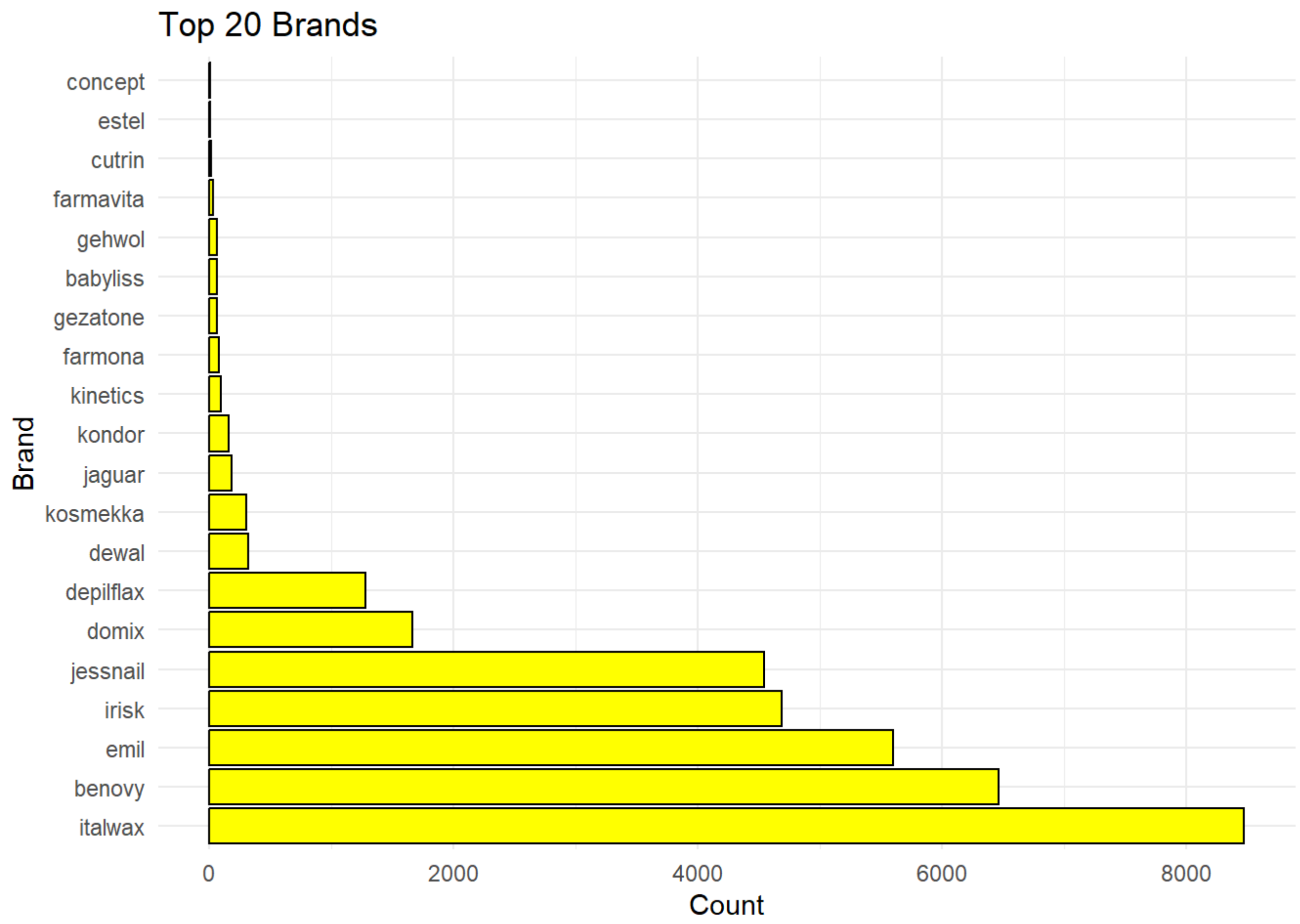


#———————————————————— # Category and Brand Analysis: #————————————————————

```
category_distribution <- head(table(newdf_j$category_code), 20)
ggplot(data = data.frame(category = names(category_distribution), count = as.numeric(category
_distribution)),
       aes(x = count, y = reorder(category, -count))) +
  geom_bar(stat = "identity", fill = "blue", color = "black") +
  labs(title = "Top 20 Product Categories", x = "Count", y = "Category Code") +
  theme_minimal()
```



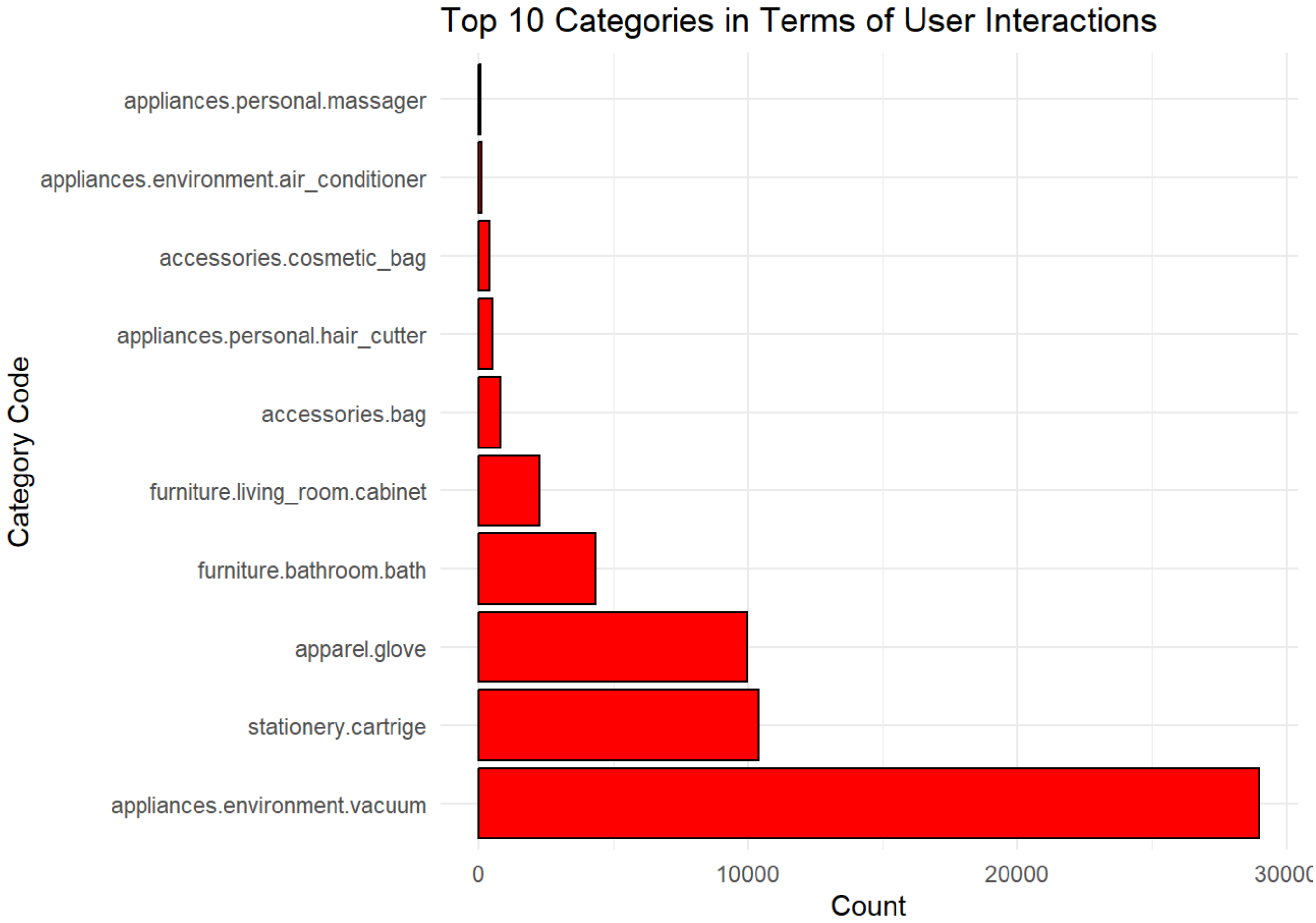

```
# Examine the distribution of products across different brands
brand_distribution <- head(table(newdf_j$brand), 20)
ggplot(data = data.frame(brand = names(brand_distribution), count = as.numeric(brand_distribu
tion)),
       aes(x = count, y = reorder(brand, -count))) +
  geom_bar(stat = "identity", fill = "yellow", color = "black") +
  labs(title = "Top 20 Brands", x = "Count", y = "Brand") +
  theme_minimal()
```




```
# Explore the most popular categories and brands in terms of user interactions
popular_categories <- newdf_j %>%group_by(category_code) %>%summarise(event_count = n()) %>%a
rrange(desc(event_count)) %>%
  top_n(10)
```

```
## Selecting by event_count
```

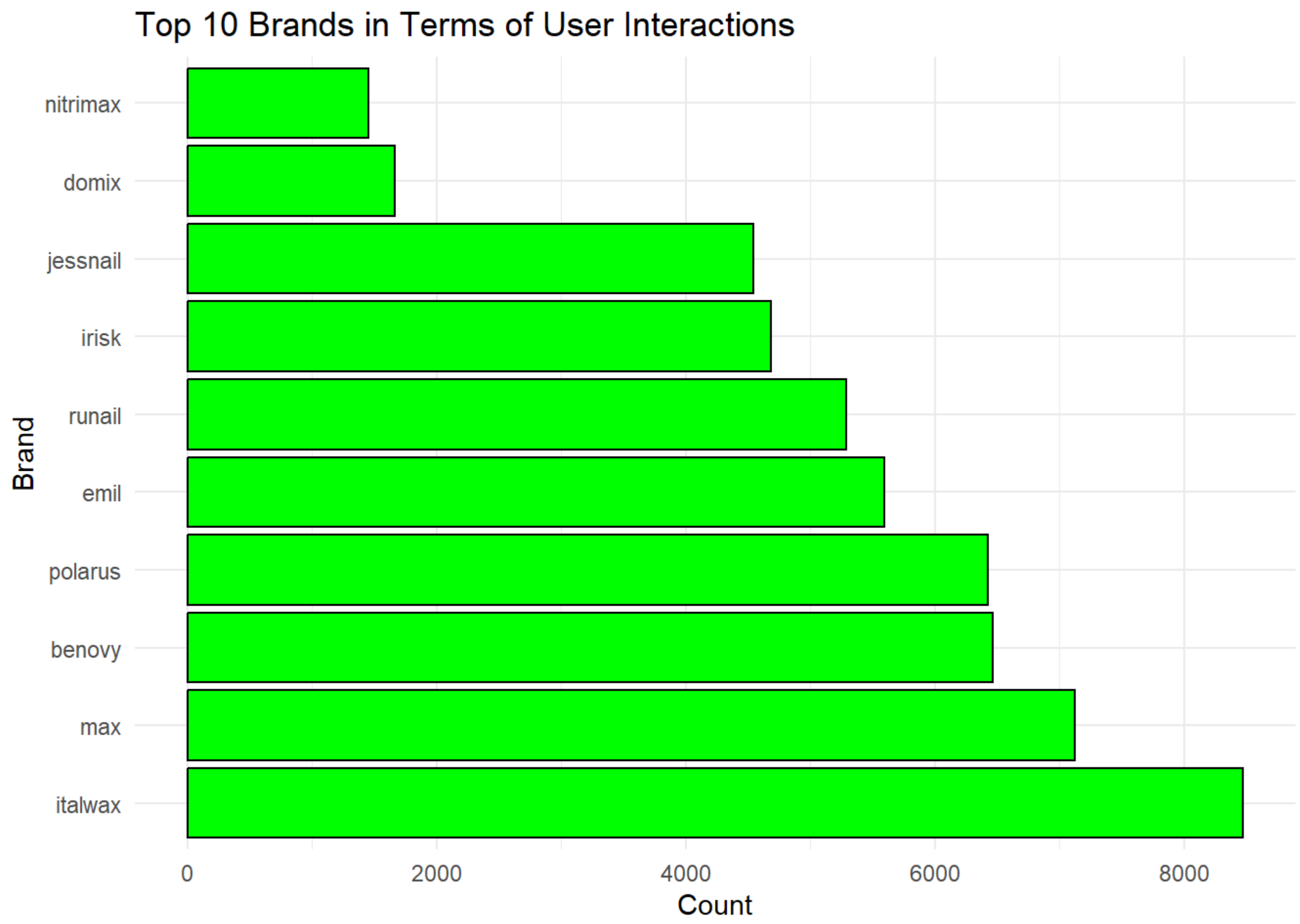
```
ggplot(popular_categories, aes(x = event_count, y = reorder(category_code, -event_count))) +
  geom_bar(stat = "identity", fill = "red", color = "black") +
  labs(title = "Top 10 Categories in Terms of User Interactions", x = "Count", y = "Category
Code") +
  theme_minimal()
```




```
popular_brands <- newdf_j %>%
  group_by(brand) %>%
  summarise(event_count = n()) %>%
  arrange(desc(event_count)) %>%
  top_n(10)
```

```
## Selecting by event_count
```

```
ggplot(popular_brands, aes(x = event_count, y = reorder(brand, -event_count))) +
  geom_bar(stat = "identity", fill = "green", color = "black") +
  labs(title = "Top 10 Brands in Terms of User Interactions", x = "Count", y = "Brand") +
  theme_minimal()
```



#———————————————————————————— # Brand Analysis: # ————————————————————————————

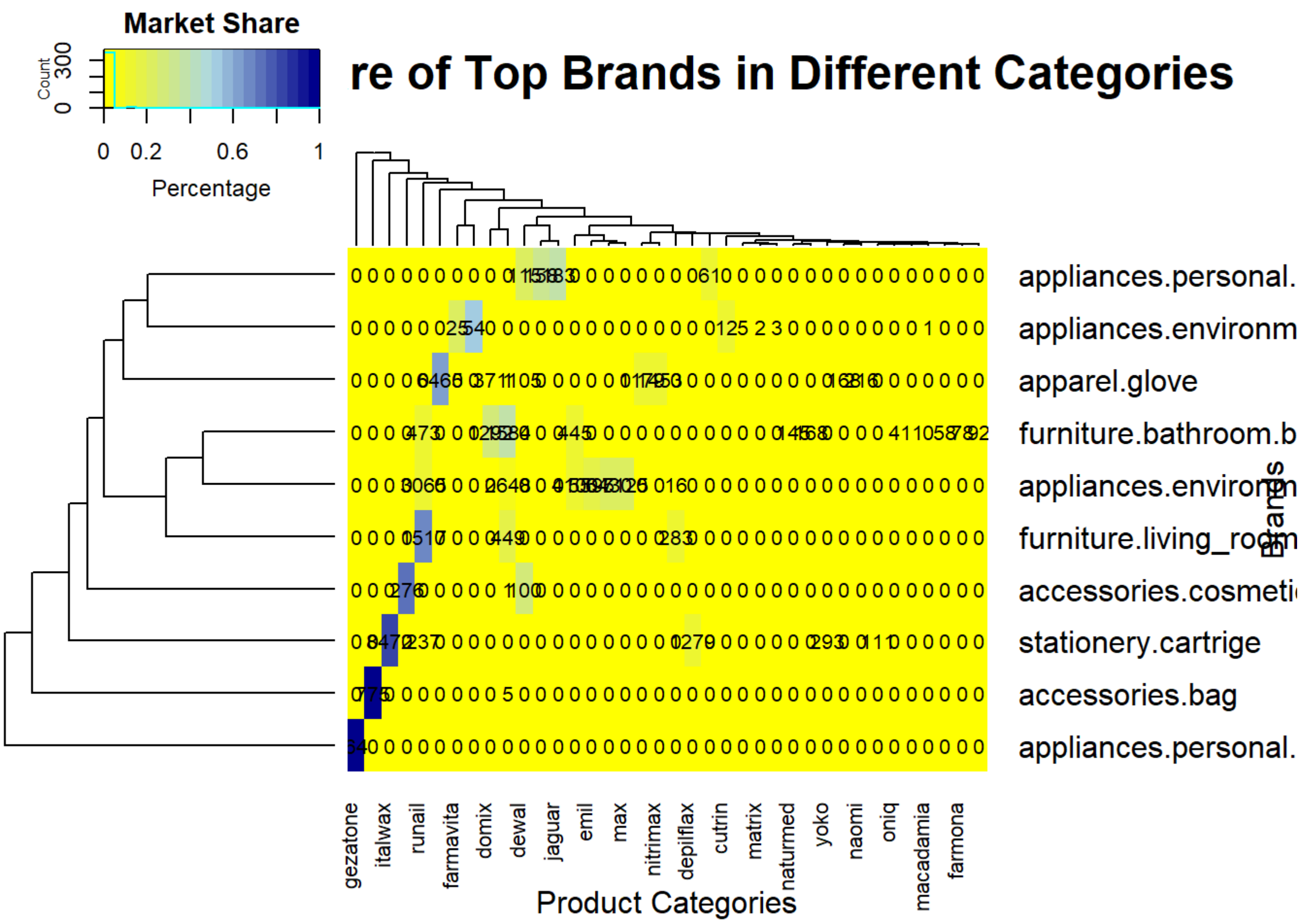
```
# Investigate the popularity of different brands
top_brands <- head(table(newdf_j$brand), 10)
print("Top 10 Brands by Popularity:")
```

```
## [1] "Top 10 Brands by Popularity:"
```

```
print(top_brands)
```

```
##
##  babyliss     benovy    concept     cutrin depilflax      dewal      domix       emil
##        61       6465          3         12       1279        320       1663       5595
##     estel farmavita
##         5         25
```

```
# Explore the market share of brands in different product categories
category_brand_counts <- table(newdf_j$category_code, newdf_j$brand)
category_brand_market_share <- prop.table(category_brand_counts, margin = 1)

# Plotting market share of top brands in each category with numerical values
heatmap.2(category_brand_market_share,
          col = colorRampPalette(c("yellow", "lightblue", "darkblue"))(20),
          trace = "none",
          margins = c(5, 10),
          main = 'Market Share of Top Brands in Different Categories',
          xlab = 'Product Categories',
          ylab = 'Brands',
          cellnote = category_brand_counts,
          notecol = "black",
          notecex = 0.8,
          key = TRUE,
          key.title = "Market Share",
          key.xlab = "Percentage"
)
```



—————————————————————————————— # Feature Engineering: # ——————————————————————————————

```
newdf_j$event_time <- as.POSIXct(newdf_j$event_time, format = "%Y-%m-%d %H:%M:%S", tz = "UT
C")
newdf_j$day <- day(newdf_j$event_time)
newdf_j$month <- month(newdf_j$event_time)
newdf_j$hour <- hour(newdf_j$event_time)
newdf_j$minute <- minute(newdf_j$event_time)

newdf_j <- newdf_j %>%
  mutate(event_time = as.POSIXct(event_time, format = "%Y-%m-%d %H:%M:%S", tz = "UTC"),
         user_session_duration = difftime(max(event_time), min(event_time), units = "secs"))

cpynewdf_j <- newdf_j

# Define columns to one-hot encode and apply one-hot encoding
encodecol <- c('event_type', 'category_code', 'brand')
cpynewdf_j <- cbind(cpynewdf_j, model.matrix(~. - 1, data = cpynewdf_j[, encodecol]))
colnames(cpynewdf_j) <- c(colnames(cpynewdf_j[, -which(colnames(cpynewdf_j) %in% encodeco
l)]),
                          paste0('event_', levels(cpynewdf_j$event_type)),
                          paste0('category_', levels(cpynewdf_j$category_code)),
                          paste0('brand_', levels(cpynewdf_j$brand)))

cpynewdf_j <- cpynewdf_j[, -which(colnames(cpynewdf_j) %in% encodecol)]
print(cpynewdf_j)
```

```
## data frame with 0 columns and 57768 rows
```

```
cpynewdf_j <- newdf_j

product_popularity <- cpynewdf_j %>%
  group_by(product_id, event_type) %>%
  summarize(count = n()) %>%
  pivot_wider(names_from = event_type, values_from = count, values_fill = 0)
```

```
## `summarise()` has grouped output by 'product_id'. You can override using the
## `.groups` argument.
```

```
print(product_popularity)
```

```
## # A tibble: 306 × 5
## # Groups:   product_id [306]
##    product_id  cart purchase remove_from_cart  view
##         <int> <int>    <int>            <int> <int>
##  1       5395    63       14               49   918
##  2       8372     8        2                5    19
##  3       8373     6        1                8     9
##  4      24330    59       14               17   135
##  5      24331    58       21               19    46
##  6      24332    11        4               10    10
##  7      24333    11        1                7    13
##  8      24334    30        5               14    34
##  9      24335     5        0                4    10
## 10      24336    58       17               29    48
## # ℹ 296 more rows
```

```
user_statistic <- newdf_j %>%
  group_by(user_id) %>%
  summarize(mu_of_price = mean(price), std_price = sd(price))

print(user_statistic)
```

```
## # A tibble: 16,832 × 3
##     user_id mu_of_price std_price
##       <int>       <dbl>     <dbl>
##  1 41152636       24.6     NA
##  2 45796898        3.18     0.315
##  3 49163609      105.      31.3
##  4 52729743        1.98     0
##  5 56612519       14.9     22.7
##  6 60197759       54.8      0
##  7 62005536       70.2     17.4
##  8 64284727       60.3      9.71
##  9 71769733        1.43     0
## 10 72559357       31.0     NA
## # ℹ 16,822 more rows
```

```
category_stats <- newdf_j %>%
  group_by(category_code) %>%
  summarize(mu_of_price = mean(price))

print(category_stats)
```

```
## # A tibble: 10 × 2
##    category_code                          mu_of_price
##    <chr>                                        <dbl>
##  1 accessories.bag                              62.5
##  2 accessories.cosmetic_bag                     12.3
##  3 apparel.glove                                 7.21
##  4 appliances.environment.air_conditioner       10.9
##  5 appliances.environment.vacuum                51.3
##  6 appliances.personal.hair_cutter              53.6
##  7 appliances.personal.massager                 50.6
##  8 furniture.bathroom.bath                      39.0
##  9 furniture.living_room.cabinet                94.3
## 10 stationery.cartrige                           2.24
```

```
colnames(newdf_j)
```

```
##  [1] "event_time"            "event_type"            "product_id"
##  [4] "category_id"           "category_code"         "brand"
##  [7] "price"                 "user_id"               "user_session"
## [10] "date"                  "day"                   "month"
## [13] "hour"                  "minute"                "user_session_duration"
```

#——————————————————————————— #Baseline Model Development #———————————————————————————

```
features <- c('event_type', 'category_code', 'brand', 'user_id')
target <- 'price'
df <- newdf_j[, c(features, target)]

set.seed(42)
splitIndex <- createDataPartition(df$price, p = 0.8, list = FALSE)
trainingdata <- df[splitIndex, ]
testingdata <- df[-splitIndex, ]

# Preprocess categorical features using one-hot encoding
trainingdata <- trainingdata %>%
  mutate(across(c('event_type', 'category_code', 'brand'), as.factor)) %>%
  select(-user_id)  # Removing user_id for simplicity

testingdata <- testingdata %>%
  mutate(across(c('event_type', 'category_code', 'brand'), as.factor)) %>%
  select(-user_id)

traininglevel <- levels(trainingdata$brand)
testinglevel <- levels(testingdata$brand)
latest_level <- setdiff(testinglevel, traininglevel)
print(latest_level)
```

```
## [1] "matrix"
```

```
combined_levels <- union(traininglevel, testinglevel)

removerow <- which(testingdata$brand %in% latest_level)
testingdata <- testingdata[-removerow, ]

#fit the linear regression RFE
baselinemod <- lm(price ~ ., data = trainingdata)

# Make predictions on the cleaned test set
prediction_baseline <- predict(baselinemod, newdata = testingdata)
```

```
## Warning in predict.lm(baselinemod, newdata = testingdata): prediction from a
## rank-deficient fit may be misleading
```

```
# MSE and R2
MSE_baselinemod <- mean((testingdata$price - prediction_baseline)^2)
r2_baselinemod <- 1 - (sum((testingdata$price - prediction_baseline)^2) / sum((testingdata$pr
ice - mean(testingdata$price))^2))

cat(paste("Linear Regression Mean Squared Error: ", MSE_baselinemod, "\n"))
```

```
## Linear Regression Mean Squared Error:  265.69034305597
```

```
cat(paste("Linear Regression R-squared: ", r2_baselinemod, "\n"))
```

```
## Linear Regression R-squared:  0.765248350645664
```

#-- #Feature Selection and Preprocessing: #--

```
newdf_j$category_code <- as.factor(newdf_j$category_code)
newdf_j$brand <- as.factor(newdf_j$brand)
newdf_j$price <- as.numeric(newdf_j$price)

# Check if 'price_category' exists in the dataset
if ('price_category' %in% colnames(newdf_j)) {
  newdf_j$price_category <- as.factor(newdf_j$price_category)
}

# Extract relevant columns
price <- newdf_j$price
newdf_j$event_type_numeric <- as.numeric(as.factor(newdf_j$event_type))

# Drop rows with missing values
price_event_type_data <- na.omit(newdf_j[c('price', 'event_type_numeric')])
print(colnames(newdf_j))
```

```
##  [1] "event_time"            "event_type"            "product_id"
##  [4] "category_id"           "category_code"         "brand"
##  [7] "price"                 "user_id"               "user_session"
## [10] "date"                  "day"                   "month"
## [13] "hour"                  "minute"                "user_session_duration"
## [16] "event_type_numeric"
```

```
features <- c('product_id', 'category_code', 'brand', 'day', 'month', 'hour', 'minute',  'eve
nt_type_numeric')
target <- 'price'

selected_data <- newdf_j[, c(features, target), drop = FALSE]
print(str(selected_data))
```

```
## 'data.frame':    57768 obs. of  9 variables:
##  $ product_id        : int  5743974 5743974 5856191 5885596 5824195 5885592 5856193 585457
4 5830789 5809118 ...
##  $ category_code     : Factor w/ 10 levels "accessories.bag",..: 10 10 5 5 3 5 9 5 1 6 ...
##  $ brand             : Factor w/ 38 levels "babyliss","benovy",..: 15 15 35 33 7 33 35 17
37 16 ...
##  $ day               : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ month             : num  1 1 1 1 1 1 1 1 1 1 ...
##  $ hour              : int  0 0 0 1 1 1 1 2 2 3 ...
##  $ minute            : int  34 36 44 16 25 26 55 5 22 35 ...
##  $ event_type_numeric: num  4 4 1 4 1 4 4 4 4 4 ...
##  $ price             : num  1.98 1.98 24.44 102.38 0.94 ...
## NULL
```

```
set.seed(42)
splitIndex <- createDataPartition(selected_data$price, p = 0.8, list = FALSE)
trainingdata <- selected_data[splitIndex, ]
testingdata <- selected_data[-splitIndex, ]

rec <- recipe(price ~ ., data = trainingdata) %>%
  step_dummy(all_nominal(), one_hot = TRUE) %>%
  prep()

X_train <- bake(rec, new_data = trainingdata)
X_test <- bake(rec, new_data = testingdata)
y_train <- trainingdata[, target]
y_test <- testingdata[, target]
```

#——————————————————————————— #Recursive Feature Elimination (RFE): #———————————————————————————

```
RFE <- lm(price ~ ., data = trainingdata)

bestr2value <- -Inf
featureselected <- NULL

for (i in 1:length(features)) {

  formula <- as.formula(paste("price ~", paste(features[1:i], collapse = "+")))

  trainctrl <- trainControl(method = "cv", number = 5)

  cv_results <- train(formula, data = trainingdata, method = "lm", trControl = trainctrl)
  current_r2 <- 1 - cv_results$results$Rsquared[1]

  if (current_r2 > bestr2value) {
    bestr2value <- current_r2
    featureselected <- features[1:i]
  }
}
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
```

```
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient fit
## may be misleading
```

```
cat("Selected Features:", featureselected, "\n")
```

```
## Selected Features: product_id
```

```
cat("Best R-squared:", bestr2value, "\n")
```

```
## Best R-squared: 0.9819551
```

#———————————————————————————— #Chi-square Test for Independence: #————————————————————————————

```
# Perform the Chi-Square Test
# Ensure that 'event_type' and 'category_code' are factors
newdf_j$event_type <- as.factor(newdf_j$event_type)
newdf_j$category_code <- as.factor(newdf_j$category_code)
contingencytable <- table(newdf_j$event_type, newdf_j$category_code)
chi2result <- chisq.test(contingencytable)
```

```
## Warning in chisq.test(contingencytable): Chi-squared approximation may be
## incorrect
```

```
print(chi2result)
```

```
##
##  Pearson's Chi-squared test
##
## data:  contingencytable
## X-squared = 10399, df = 27, p-value < 2.2e-16
```

#———————————————————— #One-way ANOVA: #————————————————————

```
newdf_j$price <- as.numeric(newdf_j$price)
# Perform One-way ANOVA
anova_result <- aov(price ~ event_type, data = newdf_j)
print(anova_result)
```

```
## Call:
##    aov(formula = price ~ event_type, data = newdf_j)
##
## Terms:
##                  event_type Residuals
## Sum of Squares     9640863  55892180
## Deg. of Freedom          3     57764
##
## Residual standard error: 31.1062
## Estimated effects may be unbalanced
```

#———————————————————— #Independent Samples t-test: #————————————————————

```
# Perform Independent Samples t-test
newdf_j$price <- as.numeric(newdf_j$price)
# Choose two specific levels for the t-test
yesgrp <- newdf_j[newdf_j$event_type == 'purchase', 'price']
nogrp <- newdf_j[newdf_j$event_type == 'view', 'price']
ttestanswer <- t.test(yesgrp, nogrp)
print(ttestanswer)
```

```
##
##  Welch Two Sample t-test
##
## data:  yesgrp and nogrp
## t = -67.692, df = 4322.8, p-value < 2.2e-16
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -31.87054 -30.07641
## sample estimates:
## mean of x mean of y
##  14.33965  45.31313
```

#——————————————————————————# Random Forest for prediction #——————————————————————————

```
# Train the random forest RFE
RFmodel <- randomForest(price ~ ., data = trainingdata, ntree = 10)
print(colnames(trainingdata))
```

```
## [1] "product_id"         "category_code"      "brand"
## [4] "day"                "month"              "hour"
## [7] "minute"             "event_type_numeric" "price"
```

```
print(colnames(X_test))
```

```
##  [1] "product_id"
##  [2] "day"
##  [3] "month"
##  [4] "hour"
##  [5] "minute"
##  [6] "event_type_numeric"
##  [7] "price"
##  [8] "category_code_accessories.bag"
##  [9] "category_code_accessories.cosmetic_bag"
## [10] "category_code_apparel.glove"
## [11] "category_code_appliances.environment.air_conditioner"
## [12] "category_code_appliances.environment.vacuum"
## [13] "category_code_appliances.personal.hair_cutter"
## [14] "category_code_appliances.personal.massager"
## [15] "category_code_furniture.bathroom.bath"
## [16] "category_code_furniture.living_room.cabinet"
## [17] "category_code_stationery.cartrige"
## [18] "brand_babyliss"
## [19] "brand_benovy"
## [20] "brand_concept"
## [21] "brand_cutrin"
## [22] "brand_depilflax"
## [23] "brand_dewal"
## [24] "brand_domix"
## [25] "brand_emil"
## [26] "brand_estel"
## [27] "brand_farmavita"
## [28] "brand_farmona"
## [29] "brand_gehwol"
## [30] "brand_gezatone"
## [31] "brand_irisk"
## [32] "brand_italwax"
## [33] "brand_jaguar"
## [34] "brand_jessnail"
## [35] "brand_kinetics"
## [36] "brand_kondor"
## [37] "brand_kosmekka"
## [38] "brand_macadamia"
## [39] "brand_masura"
## [40] "brand_matrix"
## [41] "brand_mavala"
## [42] "brand_max"
## [43] "brand_milv"
## [44] "brand_naomi"
## [45] "brand_naturmed"
## [46] "brand_nirvel"
## [47] "brand_nitrile"
## [48] "brand_nitrimax"
## [49] "brand_oniq"
## [50] "brand_polarus"
## [51] "brand_profepil"
## [52] "brand_runail"
## [53] "brand_shik"
## [54] "brand_vosev"
## [55] "brand_yoko"
```

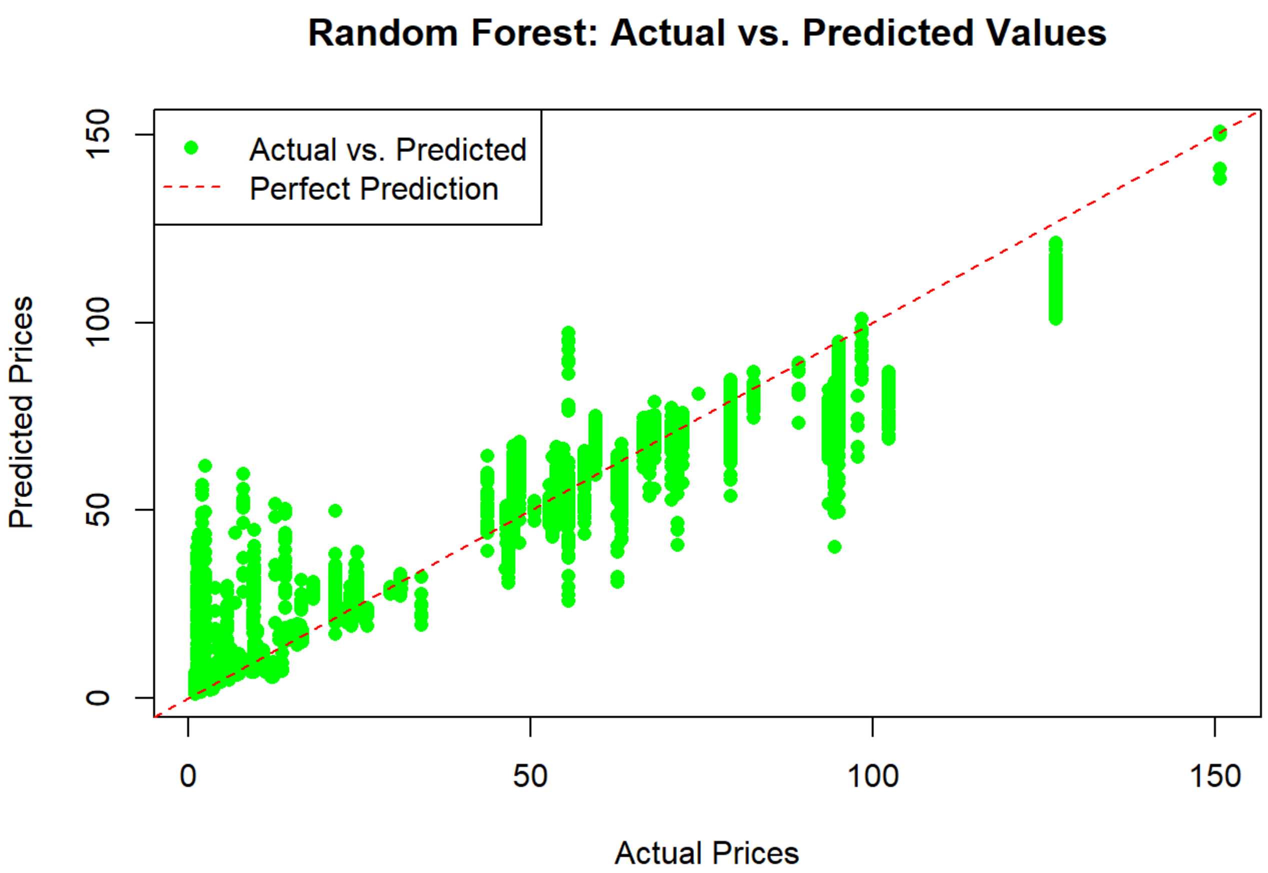
```
# Add category_code to X_test
X_test$category_code <- as.factor(testingdata$category_code)

# Add brand to X_test
X_test$brand <- as.factor(testingdata$brand)
RFpredictions <- predict(RFmodel, newdata = X_test)

# Evaluate the RFE
RFMSE <- mean((y_test - RFpredictions)^2)
RFR2 <- 1 - (sum((y_test - RFpredictions)^2) / sum((y_test - mean(y_test))^2))

cat(paste("Random Forest Mean Squared Error: ", RFMSE, "\n"))
```

```
## Random Forest Mean Squared Error:  62.0833804807873
```

```
cat(paste("Random Forest R-squared: ", RFR2, "\n"))
```

```
## Random Forest R-squared:  0.945137636550741
```

```
plot(y_test, RFpredictions,
     main = "Random Forest: Actual vs. Predicted Values",
     xlab = "Actual Prices",
     ylab = "Predicted Prices",
     col = "green",  # Color of points
     pch = 16        # Point type
)
abline(a = 0, b = 1, col = "red", lty = 2)
legend("topleft", legend = c("Actual vs. Predicted", "Perfect Prediction"),
       col = c("green", "red"), pch = c(16, NA), lty = c(NA, 2))
```


Random Forest: Actual vs. Predicted Values
Actual vs. Predicted
Perfect Prediction
Predicted Prices
0
50
100
150
0
50
100
150
Actual Prices

#———————————————————————————————— #PCA #
————————————————————————————————

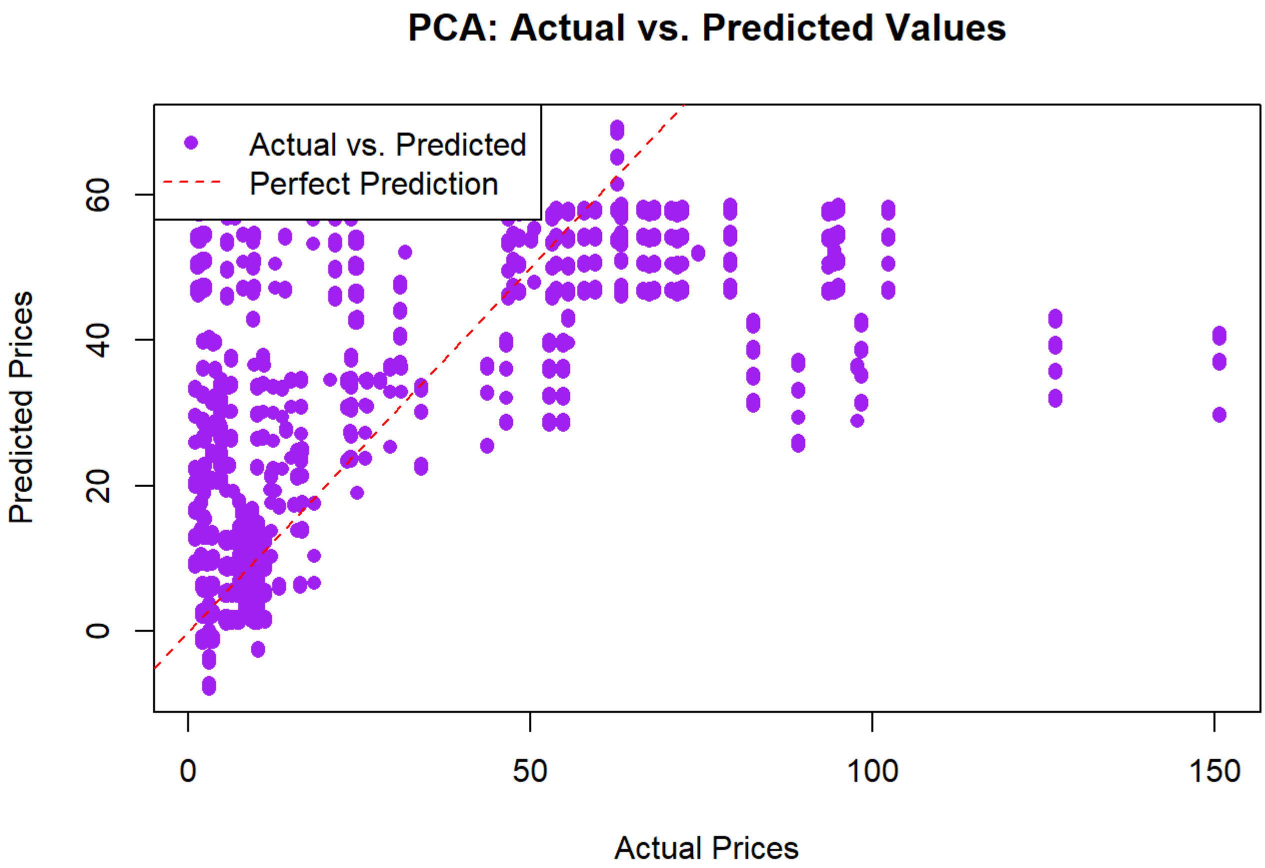
```
trainingdata$price <- as.numeric(trainingdata$price)
features <- c('product_id', 'day', 'month', 'hour', 'minute', 'event_type_numeric')

# One-hot encode categorical variables
encodedtrain <- cbind(trainingdata[, features, drop = FALSE], model.matrix(~ category_code +
brand + event_type_numeric - 1, data = trainingdata))
X <- cbind(encodedtrain[, -ncol(encodedtrain)], trainingdata$price)

# Perform PCA
PCA_final <- PCA(X[, -ncol(X)], scale.unit = TRUE, graph = FALSE)

# Choose the number of principal components to retain
no_of_components <- 5
PCA_X <- as.data.frame(PCA_final$ind$coord[, 1:no_of_components])
colnames(PCA_X) <- paste0("PC", 1:no_of_components)
data_PCA <- cbind(PCA_X, price = trainingdata$price)

# Train a RFE on the PCA-transformed data
RFE <- lm(price ~ ., data = data_PCA)
PCA_pred <- predict(RFE, newdata = PCA_X)

MSE_PCA <- mean((trainingdata$price - PCA_pred)^2)
R2_PCA <- 1 - (sum((trainingdata$price - PCA_pred)^2) / sum((trainingdata$price - mean(traini
ngdata$price))^2))

cat(paste("PCA Mean Squared Error: ", MSE_PCA, "\n"))
```

```
## PCA Mean Squared Error:  601.737316053185
```

```
cat(paste("PCA R-squared: ", R2_PCA, "\n"))
```

```
## PCA R-squared:  0.469888783066722
```

```
plot(trainingdata$price, PCA_pred,
     main = "PCA: Actual vs. Predicted Values",
     xlab = "Actual Prices",
     ylab = "Predicted Prices",
     col = "purple",
     pch = 16
)
abline(a = 0, b = 1, col = "red", lty = 2)
legend("topleft", legend = c("Actual vs. Predicted", "Perfect Prediction"),
       col = c("purple", "red"), pch = c(16, NA), lty = c(NA, 2))
```



—————————————————————— # Random Forrest CV and Hyperparameter Training # ——————————————————————

```
rm(df_j)
rm(df)
trainctrl <- trainControl(method = "cv",  # Cross-validation method
                     number = 2,       # Number of folds
                     verboseIter = TRUE,
                     allowParallel = TRUE)

# Define the parameter grid for tuning
gridparameter <- expand.grid(mtry = c(2, 5, 8))

# Train the random forest RFE using grid search
RF_CV <- train(price ~ .,
                     data = trainingdata,
                     method = "rf",
                     trControl = trainctrl,
                     tuneGrid = gridparameter)
```

```
## + Fold1: mtry=2
## - Fold1: mtry=2
## + Fold1: mtry=5
## - Fold1: mtry=5
## + Fold1: mtry=8
## - Fold1: mtry=8
## + Fold2: mtry=2
## - Fold2: mtry=2
## + Fold2: mtry=5
## - Fold2: mtry=5
## + Fold2: mtry=8
## - Fold2: mtry=8
## Aggregating results
## Selecting tuning parameters
## Fitting mtry = 8 on full training set
```

```
print(RF_CV)
```

```
## Random Forest
##
## 46217 samples
##     8 predictor
##
## No pre-processing
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 23109, 23108
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     22.04780  0.6714515  17.612611
##   5     16.39799  0.7769092  10.532577
##   8     13.67489  0.8427896   8.100597
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 8.
```

```
RF_CV_pred <- predict(RF_CV, newdata = X_test)
RF_MSE_CV <- mean((y_test - RF_CV_pred)^2)
RF_R2_CV <- 1 - (sum((y_test - RF_CV_pred)^2) / sum((y_test - mean(y_test))^2))

cat(paste("Random Forest CV Mean Squared Error: ", RF_MSE_CV, "\n"))
```

```
## Random Forest CV Mean Squared Error:  173.576819581291
```

```
cat(paste("Random Forest CV R-squared: ", RF_R2_CV, "\n"))
```

```
## Random Forest CV R-squared:  0.846612177228618
```

————————————————————————— # PCA CV and Hyperparameter Training

---

```r
numericvar <- sapply(trainingdata, is.numeric)

train_data_filtered <- trainingdata[, numericvar & sapply(trainingdata, function(x) length(un
ique(x)) > 1)]

PCA_grid_parameter <- expand.grid(
  ncomp = c(5, 10)
)

PCA_CV <- train(
  price ~ .,
  data = train_data_filtered,
  method = "pcr",      # Principal Component Regression
  trControl = trainctrl,
  tuneGrid = PCA_grid_parameter,
  preProcess = "pca",  # Specify PCA as the preprocessing method
  verbose = FALSE
)
```

```
## + Fold1: ncomp=10
## - Fold1: ncomp=10
## + Fold2: ncomp=10
## - Fold2: ncomp=10
## Aggregating results
## Selecting tuning parameters
## Fitting ncomp = 5 on full training set
```

```r
print(PCA_CV)
```

```
## Principal Component Analysis
##
## 46217 samples
##     5 predictor
##
## Pre-processing: principal component signal extraction (5), centered (5),
##  scaled (5)
## Resampling: Cross-Validated (2 fold)
## Summary of sample sizes: 23108, 23109
## Resampling results across tuning parameters:
##
##   ncomp  RMSE      Rsquared   MAE
##    5     31.22294  0.1411824  26.07744
##   10     31.22294  0.1411824  26.07744
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 5.
```

```r
PCA_CV_pred <- predict(PCA_CV, newdata = X_test)
PCA_CV_MSE <- mean((y_test - PCA_CV_pred)^2)
PCA_CV_r2 <- 1 - sum((y_test - PCA_CV_pred)^2) / sum((y_test - mean(y_test))^2)

cat(paste("PCA CV Mean Squared Error: ", PCA_CV_MSE, "\n"))
```

```
## PCA CV Mean Squared Error:  964.061822306752
```

```
cat(paste("PCA CV R-squared: ", PCA_CV_r2, "\n"))
```

```
## PCA CV R-squared:  0.148069746309702
```

-- # Model Comparison
--

```
model_comp <- data.frame(
  Model = c("Random Forest", "PCA", "Random Forest CV", "PCA CV"),
  MSE = numeric(4),
  R2 = numeric(4)
)

model_comp[1, c("MSE", "R2")] <- c(RFMSE, RFR2)
model_comp[2, c("MSE", "R2")] <- c(MSE_PCA, R2_PCA)
model_comp[3, c("MSE", "R2")] <- c(RF_MSE_CV, RF_R2_CV)
model_comp[4, c("MSE", "R2")] <- c(PCA_CV_MSE, PCA_CV_r2)

print(model_comp)
```

```
##              Model       MSE        R2
## 1    Random Forest  62.08338 0.9451376
## 2              PCA 601.73732 0.4698888
## 3 Random Forest CV 173.57682 0.8466122
## 4           PCA CV 964.06182 0.1480697
```