**Part A:**

1.

code:

awk 'NR<=6 || NR>(total-5)' total=$(wc -l < consumer\_complaints.csv) consumer\_complaints.csv

answer:



Explanation:

awk 'NR<=6 || NR>(total-5) :

If the current action is in the first 6 or last 5 lines, the action is executed

total=$(wc -l < consumer\_complaints.csv) :

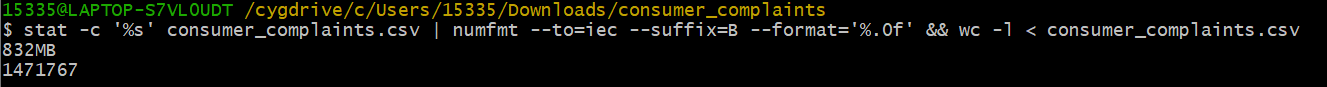
Gets the total number of lines in the file

The console outputs the contents of rows that meet these criteria

2.

code:

stat -c '%s' consumer\_complaints.csv | numfmt --to=iec --suffix=B --format='%.0f' && wc -l < consumer\_complaints.csv

answer:

Explanation:

stat -c '%s' consumer\_complaints.csv:

Used to get the size of the consumer\_complaints.csv file and output it in bytes

| :The output of the previous command is passed to the next command as input

numfmt --to=iec --suffix=B --format='%.0f': Is formatted file size

&& : Execute the current command after it succeeds

wc -l < consumer\_complaints.csv: Calculate the number of lines in the consumer\_complaints.csv file

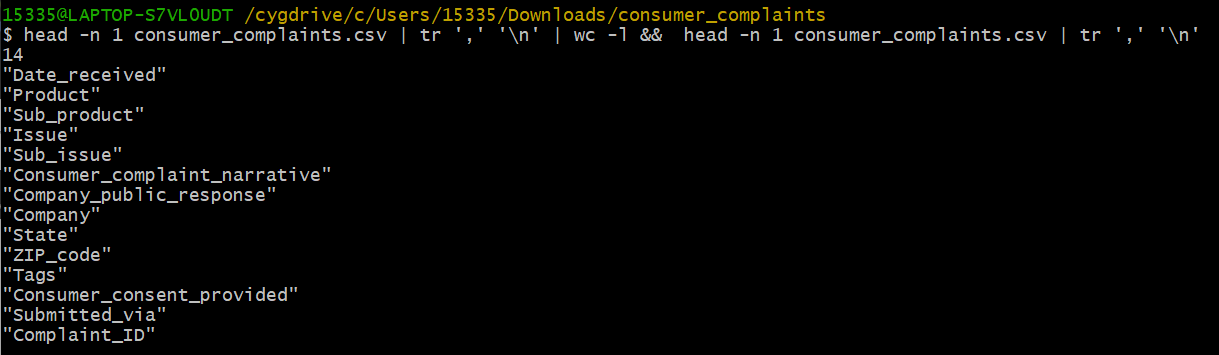
The final output is the file size and the number of lines

3.

code:

head -n 1 consumer\_complaints.csv | tr ',' '\n' | wc -l && head -n 1 consumer\_complaints.csv | tr ',' '\n'

answer:



Explanation:

Run the Head -n 1 consumer\_complaints.csv command to obtain the first line of the consumer\_complaints.csv file

tr ',' \n' This part of the command is used to replace commas (,) with newlines (\n)

wc -l command is used to count the number of rows entered

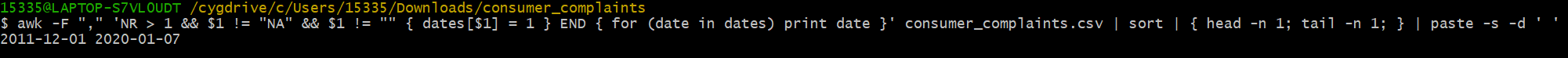
First get the first line of the consumer\_complaints.csv file, then replace the comma with a newline, and finally count the number of replaced lines

4.

code:

awk -F "," 'NR > 1 && $1 != "NA" && $1 != "" { dates[$1] = 1 } END { for (date in dates) print date }' consumer\_complaints.csv | sort | { head -n 1; tail -n 1; } | paste -s -d ' '

answer:



Explanation:

The -F "," option specifies that the field separator is comma.

NR &gt; 1 && $1! = "NA" && $1 ! = "" The filter line number is greater than 1, and the value of the first field is not equal to "NA" or null

{dates[$1] = 1} takes the first field of the row that meets the condition as the index, sets the value to 1, and builds an array of dates for recording unduplicated dates

END {for (date in dates) print date} After processing the file, iterate through the set of dates, printing all the dates

{ head -n 1; tail -n 1; } This section of the command is used to get the first and last lines in the sorted result.

paste -s -d "" The paste -s -d" command is used to combine multiple lines of text into one line and separate them with Spaces

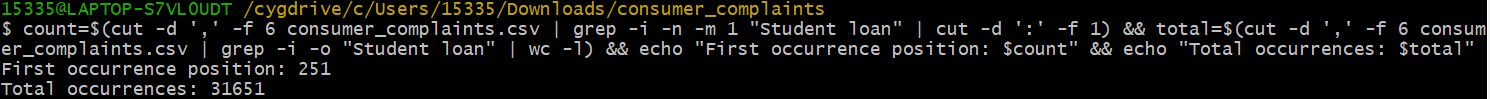
First extract the non-duplicate dates from the consumer\_complaints.csv file and sort them in lexicographical order. Then get the earliest date and the latest date from the sorting result; Finally, combine the two dates into one line, separated by a space

5.

code:

count=$(cut -d ',' -f 6 consumer\_complaints.csv | grep -i -n -m 1 "Student loan" | cut -d ':' -f 1) && total=$(cut -d ',' -f 6 consumer\_complaints.csv | grep -i -o "Student loan" | wc -l) && echo "First occurrence position: $count" && echo "Total occurrences: $total"

answer:



Explanation:

count=$(cut -d ',' -f 6 consumer\_complaints.csv | grep -i -n -m 1 "Student loan" | cut -d ':' -f 1) This section of the command is used to get the location where "Student loan" first appears

total=$(cut -d ',' -f 6 consumer\_complaints.csv | grep -i -o "Student loan" | wc -l) This command is used to collect Student statistics The total number of times loan appears in the sixth field

echo "First occurrence position: $count" && echo "Total occurrences: $total" The echo "first occurrence position: $count" && echo" total occurrences: $total" command outputs the result of the first occurrence and total occurrences

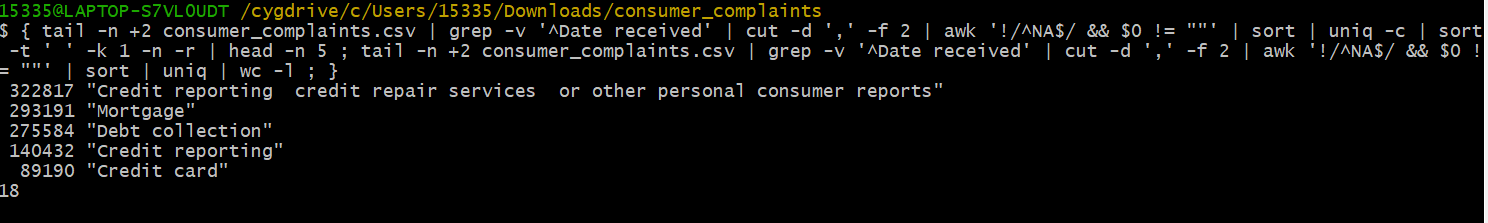
Extract the information containing "Student loan" from the sixth field of the consumer\_complaints.csv file, count the location and total number of occurrences for the first time, and output the result

6.

code:

{ tail -n +2 consumer\_complaints.csv | grep -v '^Date received' | cut -d ',' -f 2 | awk '!/^NA$/ && $0 != ""' | sort | uniq -c | sort -t ' ' -k 1 -n -r | head -n 5 ; tail -n +2 consumer\_complaints.csv | grep -v '^Date received' | cut -d ',' -f 2 | awk '!/^NA$/ && $0 != ""' | sort | uniq | wc -l ; }

answer:



Explanation:

The tail -n +2 consumer\_complaints.csv command is used to extract the content starting from line 2 of the consumer\_complaints.csv file

The grep -v '^Date received' command is used to filter out lines containing "Date received", that is, skip the header line in the file

The cut -d ',' -f2 command is used to extract the second field, using a comma as the field separator

awk '! /^NA$/ && $0 ! The = ""' command is used to filter out lines with a value of "NA" or an empty string

The sort command is used to sort the input.

The uniq -c command is used to count and remove repeated rows and display the number of repeated rows in front of each row

sort -t "-k 1 -n -r Used to sort the first field (number of repetitions) in reverse order

The head -n 5 command is used to extract the first five rows of the sorting result, that is, the top five values that occur most frequently

The uniq | wc -l command It is used to collect statistics on the number of non-duplicate values

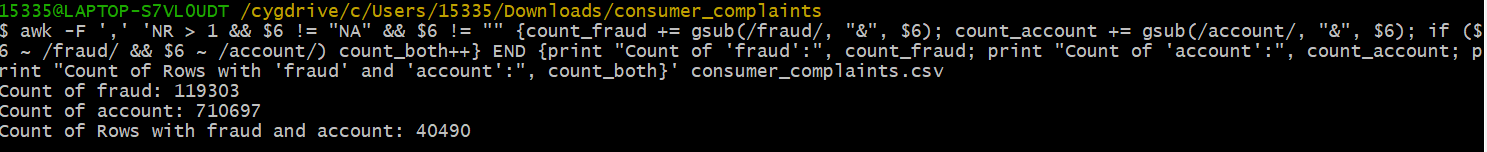
Get the value of the second field from the consumer\_complaint.csv file, filter, sort, count, and extract it to get the top 5 most frequent non-duplicate values, and calculate the total number of non-duplicate values

7.

code:

awk -F ',' 'NR > 1 && $6 != "NA" && $6 != "" {count\_fraud += gsub(/fraud/, "&", $6); count\_account += gsub(/account/, "&", $6); if ($6 ~ /fraud/ && $6 ~ /account/) count\_both++} END {print "Count of 'fraud':", count\_fraud; print "Count of 'account':", count\_account; print "Count of Rows with 'fraud' and 'account':", count\_both}' consumer\_complaints.csv

answer:



Explanation:

awk -F ',' Sets the field separator to comma to specify the format of the csv file.

NR &gt; 1 & $6! = "NA" && $6 ! = "" This condition is used to filter out the first row (header) as well as rows where the sixth field is "NA" or an empty string.

{count\_fraud += gsub(/fraud/, "&", $6); count\_account += gsub(/account/, "&", $6); if ($6 ~ /fraud/ && $6 ~ /account/) count\_both++} This section operates on rows that satisfy the condition

The END block means to do the following after all rows have been processed.

print "Count of 'fraud':", count\_fraud The number of times the keyword "fraud" is printed.

print "Count of 'account':", count\_account The number of times the output contains the keyword "account".

print "Count of Rows with 'fraud' and 'account':", count\_both Print "count of rows with 'fraud' and 'account':", count\_both

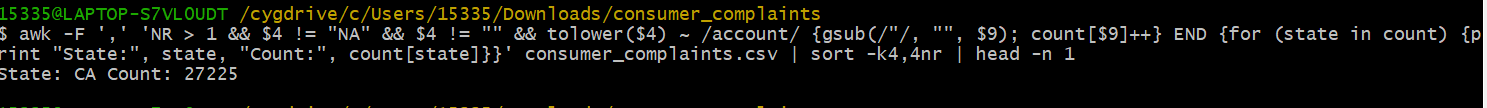
Count the number of keyword "fraud" and "account" in the sixth field of the consumer\_complaints.csv file, and output the corresponding statistical result. Finally, the output shows the number of lines containing "fraud", the number of lines containing "account", and the number of lines containing both "fraud" and "account"

8.

code:

awk -F ',' 'NR > 1 && $4 != "NA" && $4 != "" && tolower($4) ~ /account/ {gsub(/"/, "", $9); count[$9]++} END {for (state in count) {print "State:", state, "Count:", count[state]}}' consumer\_complaints.csv | sort -k4,4nr | head -n 1

answer:



Explanation:

awk -F ',' Sets the field separator to comma to specify the format of the csv file.

NR &gt; 1 && $4! = "NA" && $4 ! = "" &&tolower ($4) ~ /account/ This condition is used to filter out the first row (header) and rows with the fourth field being "NA" or an empty string, and the fourth field containing the "account" keyword after case is ignored.

{gsub(/"/, "", $9); count[$9]++} This section operates on rows that satisfy the condition.

gsub(/"/, "", $9) is used to remove double quotes from the 9th field.

count[$9]++ Counts the number of occurrences of each state using the value of the 9th field as the key.

The END block means to do the following after all rows have been processed.

for (state in count) {print "State:", state, "Count:", count[state]} Loops through each state with a for loop, printing the state and the corresponding number of occurrences.

sort-k4,4nr Sorts the output by the fourth field (number of occurrences) in reverse order.

-k4,4nr Indicates that the sorting key is the fourth field. nr indicates that the sorting is in reverse order.

head-n 1 Retrieves the first row of the sorted result, that is, the state that occurs the most times.

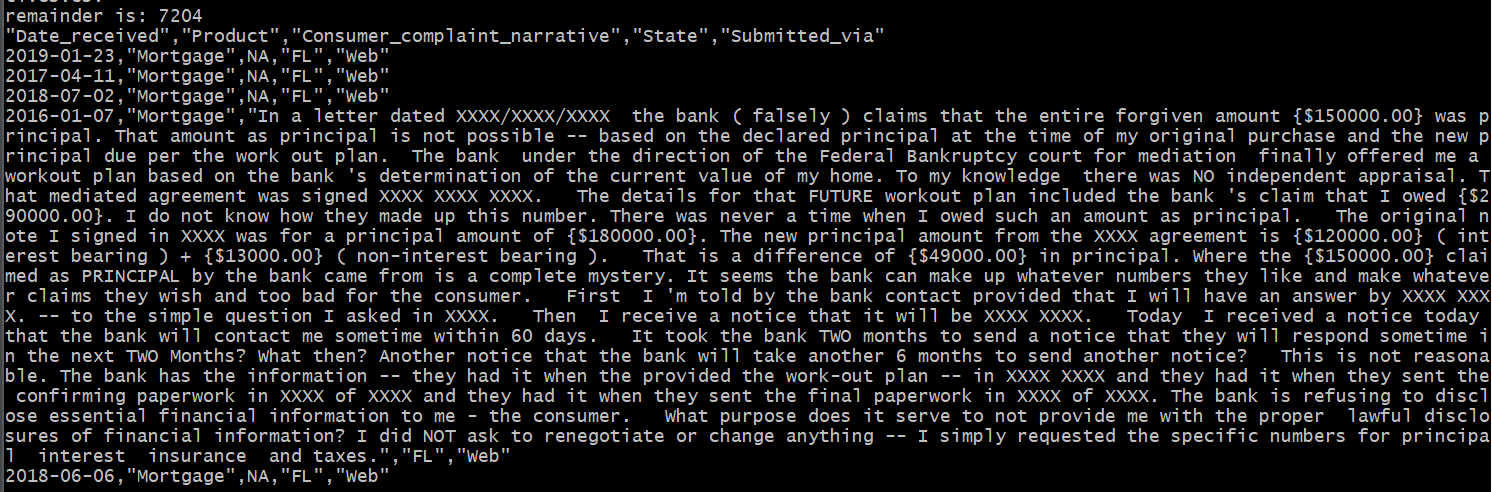
Count the number of occurrences of the fourth field (status field) that contains the keyword "account" in the consumer\_complaints.csv file, and output the status with the highest number of occurrences and the corresponding occurrence times

9.

code:

awk -F',' 'NR == 1 || (substr($1,1,4) > 2015 && $9 ~ /FL/ && $13 ~ /Web/ && $2 ~ /Mortgage/) {print $1 "," $2 "," $6 "," $9 "," $13}' consumer\_complaints.csv > filtered\_narratives.csv ; echo "remainder is: $(wc -l < filtered\_narratives.csv)" ; head -n 6 filtered\_narratives.csv

answer:



Explanation:

awk -F',' Sets the field separator to comma to specify the format of the csv file.

The condition NR == 1 indicates that the first row (header) is processed and the header information is preserved.

(substr($1,1,4) &gt; 2015 && $9 ~ /FL/ && $13 ~ /Web/ && $2 ~ /Mortgage/) This condition is used to filter the eligible rows.

substr($1, 4) &gt; 2015 means that the first 4 characters of the first field (the year part) are extracted and compared to 2015, keeping only the lines larger than 2015.

$9 ~ /FL/ Indicates the line with "FL" in the 9th field (state code field).

$13 ~ /Web/ Indicates the line with "Web" in the 13th field (Complaint Source field).

$2 ~ /Mortgage/ Indicates the row where the second field (product field) contains "Mortgage".

{print $1 "," $2 "," $6 "," $9 "," $13} operates on the rows that meet the condition, extracts the fields 1, 2, 6, 9, and 13, and prints them in the specified format.

consumer\_complaints.csv &gt; filtered\_narratives.csv Saves the extracted results to the filtered\_narratives.csv file.

echo "remainder is: $(wc -l &lt; filtered\_narratives.csv)" Prints the number of lines in the filtered\_narratives.csv file, that is, the number of remaining lines.

head -n 6 filtered\_narratives.csv Print the first six lines of the filtered\_narratives.csv file to display the extraction result.

**Part B:**

code:

# Load required packages

library(ggplot2)

library(dplyr)

library(lubridate)

library(rvest)

library(stringr)

library(reshape2)

# Load HTML content

html <- read\_html("https://www.rba.gov.au/statistics/frequency/exchange-rates.html")

# Locate table element using CSS selector or XPath expression

table <- html %>% html\_node("table") # CSS selector

# Alternatively,

# table <- html %>% html\_nodes(xpath = "//table") # XPath expression

# Extract table data

data <- table %>% html\_table(fill = TRUE)

# Convert data to DataFrame

df <- data.frame(data)

df <- t(df)

df[1, 1] <- 'date'

# Extract the first row of data

col\_names <- df[1, ]

# Set the first row as column names

colnames(df) <- col\_names

# Convert matrix or array to data frame

df <- as.data.frame(df)

# Remove the first row

df <- df[-1, ]

# Get the current number of rows

num\_rows <- nrow(df)

# Set new row names as integer sequence

new\_row\_names <- 1:num\_rows

# Assign the new row names to the data frame

rownames(df) <- new\_row\_names

# Convert date format

df1 <- melt(df, id = "date")

colnames(df1) <- c('date', 'money\_kind', 'value')

# Convert the 'value' column to numeric

df1$value <- as.numeric(df1$value)

# Select the top 16 currency types

top16\_currencies <- df1 %>%

group\_by(money\_kind) %>%

summarise(total\_value = sum(value)) %>%

top\_n(16, total\_value) %>%

arrange(desc(total\_value)) %>%

pull(money\_kind)

# Filter out the data for the top 16 currencies

df\_top16 <- df1 %>%

filter(money\_kind %in% top16\_currencies)

df\_top16$date <- str\_sub(df\_top16$date, start = 1, end = 6)

# Create a line plot, displayed in a 4x4 panel, and apply logarithmic scale to the y-axis

ggplot(df\_top16, aes(x = date, y = value, colour = money\_kind, group = money\_kind)) +

geom\_line(size = 1) +

labs(x = "date", y = "value", colour = "money kind", title = "Units of Foreign Currencies per Australian Dollar") +

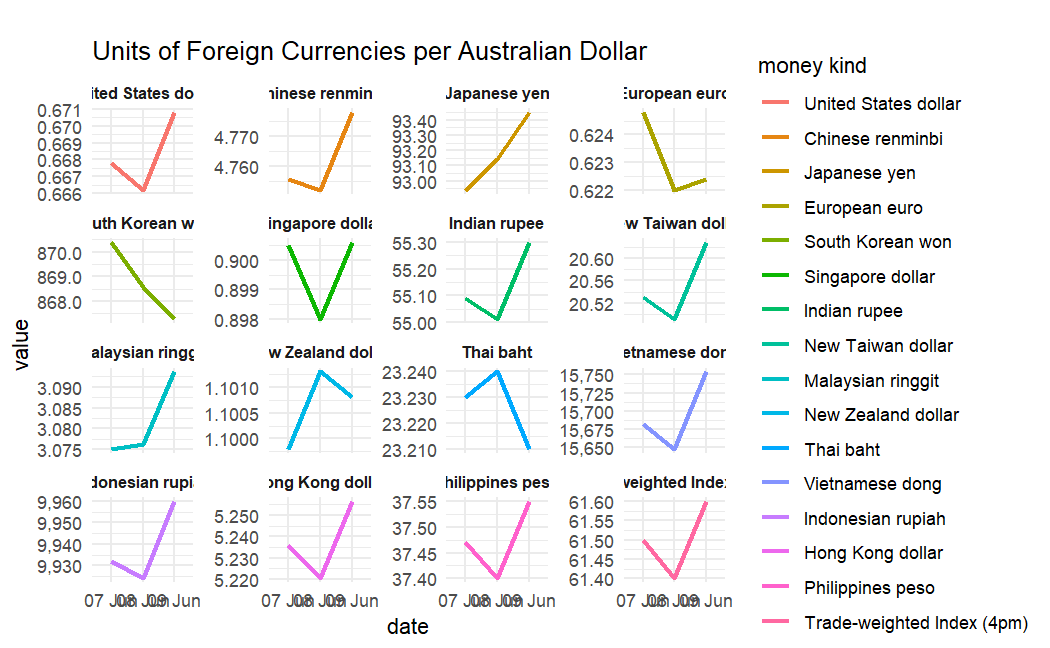
theme\_minimal() +

facet\_wrap(~ money\_kind, nrow = 4, ncol = 4, scales = "free\_y") +

theme(strip.text = element\_text(size = 8, face = "bold")) +

scale\_y\_continuous(trans = "log10", labels = scales::comma) # Use English labels with comma as the thousands separator

answer:



Code:

# Load HTML content

html <- read\_html("https://www.rba.gov.au/statistics/frequency/retail-payments/2023/retail-payments-0423.html")

# Use XPath expressions to locate table elements

table <- html %>% html\_nodes(xpath = "//table")

# Extract table data

data <- table %>% html\_table(fill = TRUE)

# Convert the data to a DataFrame

data <- data[[1]]

data <- data[-nrow(data), ]

data <- data[, 1:2]

data <- data[-c(1:3), ]

colnames(data) <- c("Value of Retail Payments", "Value ($ billion)")

data$`Value ($ billion)` <- as.numeric(gsub("[^0-9.]", "", data$`Value ($ billion)`))

data <- data[!grepl("of which", data$`Value of Retail Payments`), ]

data$Value <- gsub("^([[:alpha:]]+\\s+[[:alpha:]]+).\*", "\\1", data$`Value of Retail Payments`)

# Generated pie chart

library(scales) # The scales package is loaded to format percentages

percent\_values <- percent(data$`Value ($ billion)` / sum(data$`Value ($ billion)`))

pie(data$`Value ($ billion)`,

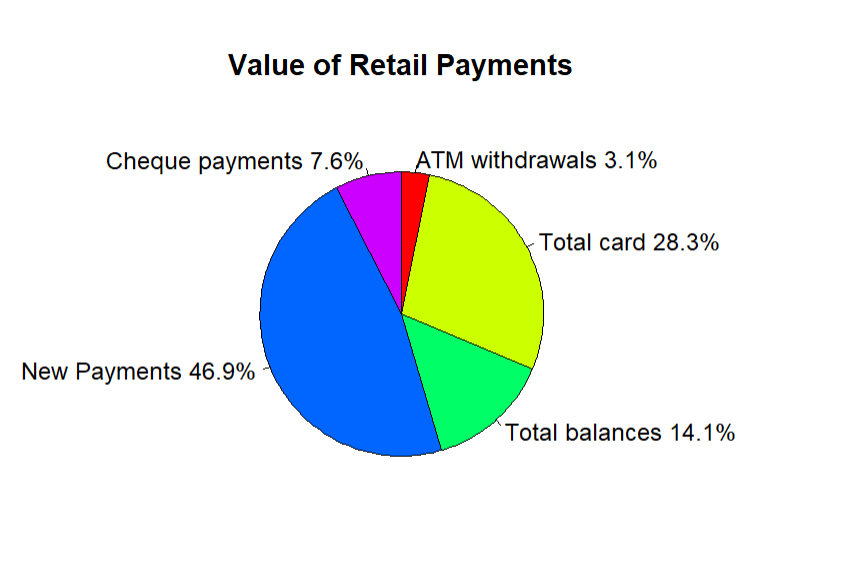
labels = paste(data$Value, percent\_values),

col = rainbow(length(data$`Value ($ billion)`)),

main = "Value of Retail Payments",

clockwise = TRUE)

answer:



Explanation:

First, I loaded some necessary packages, including ggplot2, dplyr, lubrication, vest, and stringr. These packages provide the ability to manipulate data, parse HTML, manipulate strings, and draw graphs.

Use the read\_html function to load HTML content as a processable HTML object.

Use CSS selectors or XPath expressions to locate table elements. The table element is obtained through the html\_node function.

Use the html\_table function to extract table data.

Converts the extracted data to a data frame DataFrame.

Perform a series of processing operations on the data frame, including setting column names, deleting rows, transposing data, processing date formats, and so on.

The top 16 currency categories were selected based on criteria.

Create a linear plot, draw curves by date and currency class, and apply a logarithmic scale on the Y-axis.

Use XPath expressions to locate table elements.

Extract table data.

Converts data into data frames.

Perform a series of processing operations on the data frame, including deleting rows, processing column names, converting data types, and so on.

Generate a pie chart showing retail payments

**Part C:**

Task 1

Code:

library(tidyverse)

library(ggplot2)

library(scales)

# Read the CSV file named "ptv\_data".

ptv\_data <- read.csv('property\_transaction\_victoria.csv')

# Check dimensions (number of rows and columns)

dim(ptv\_data)

# Display the structure of the dataframe

str(ptv\_data)

# Check the unique values in the "state" column

unique(ptv\_data$state)

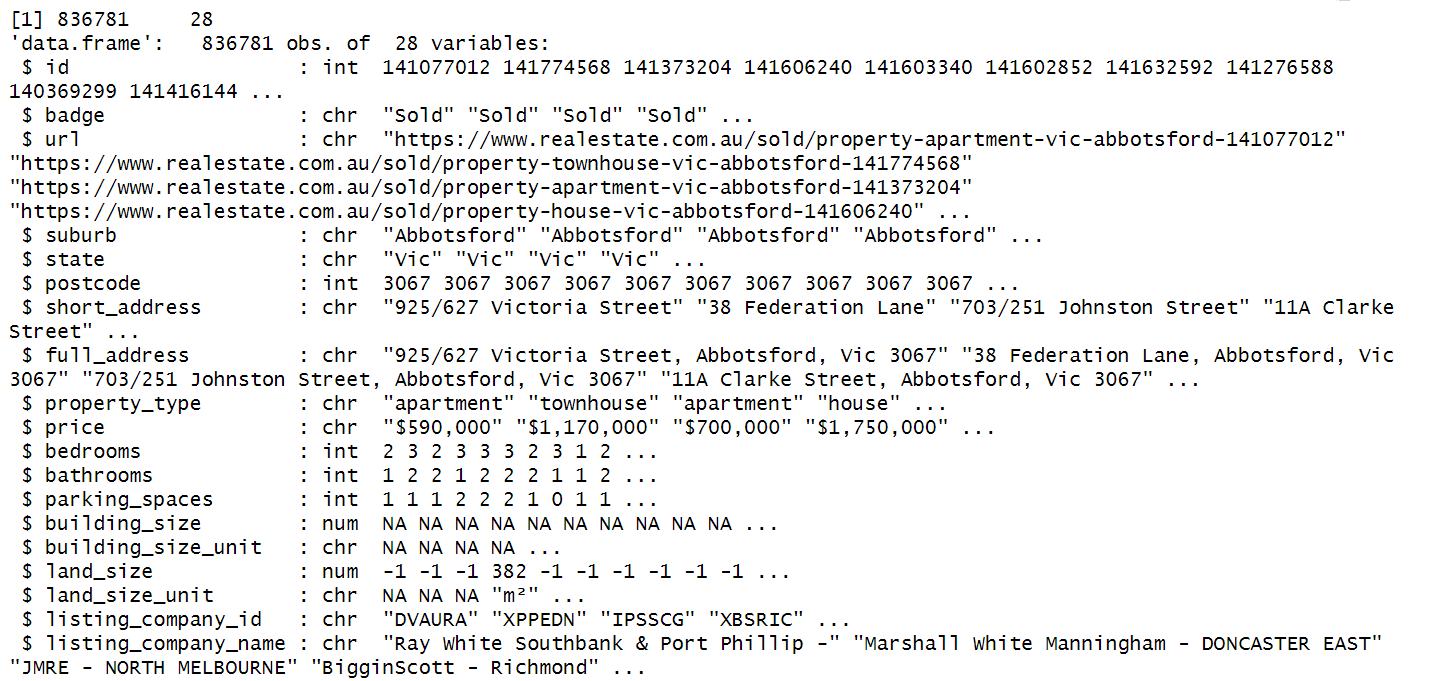
# Only the data in the Vic state is retained

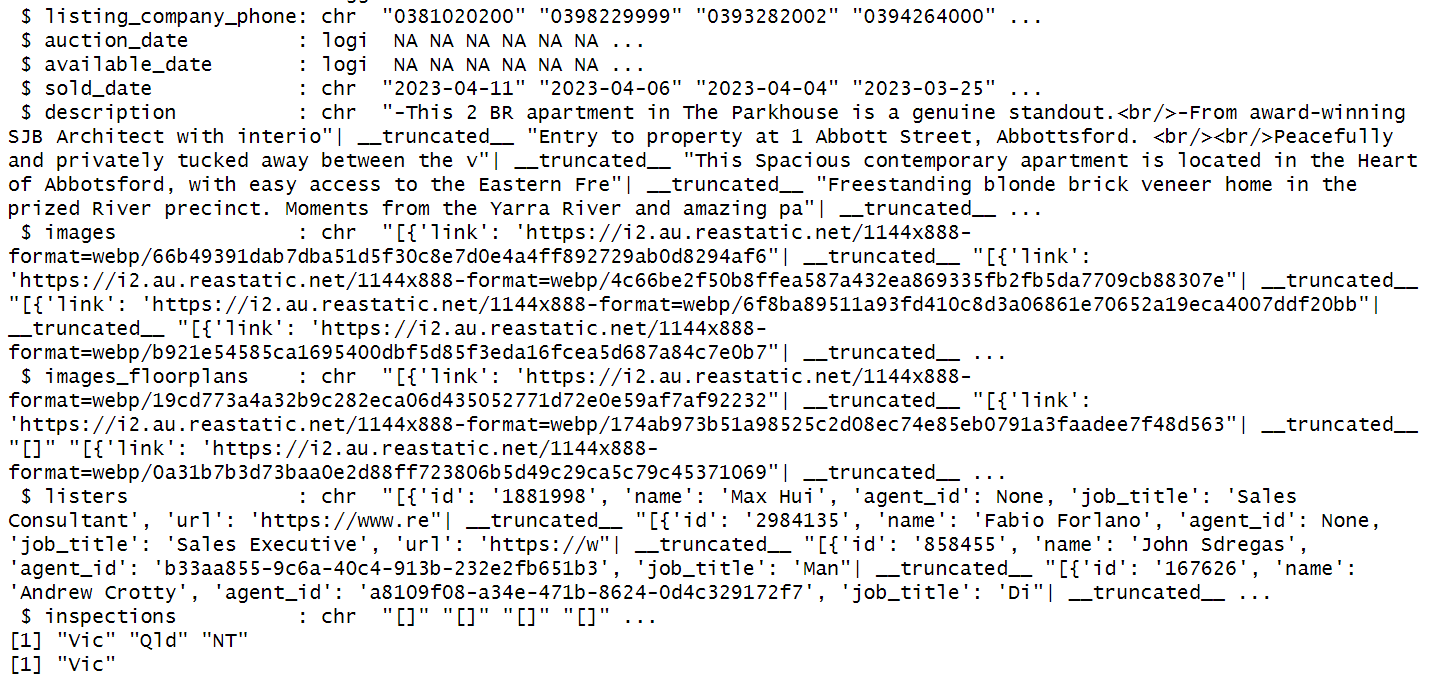
ptv\_data\_vic <- filter(ptv\_data, state == "Vic")

# Check the unique values in the "state" column after filtering

unique(ptv\_data\_vic$state)

Answer:





Explanation:

The code reads a CSV file named "property\_transaction\_victoria.csv" into a data box named "ptv\_data".

It checks the dimensions (number of rows and columns) of the data frame.

It shows the structure of the data frame, displaying column names and their data types.

It checks for unique values in the "state" column.

It filters data frames to keep only records with a status of "Vic".

It checks for unique values in the Status column after filtering.

Task 2

Code:

# Task 2:

# Delete unnecessary columns

ptv\_data\_vic <- ptv\_data\_vic %>%

select(-badge, -url, -building\_size\_unit, -land\_size\_unit, -listing\_company\_id, -listing\_company\_phone, -auction\_date, -available\_date, -images, -images\_floorplans, -listers, -inspections)

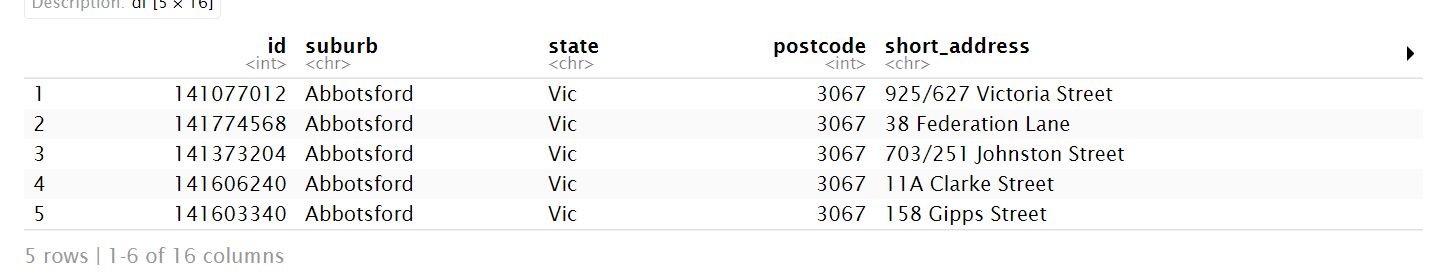
# Check the dimensions of the updated dataframe

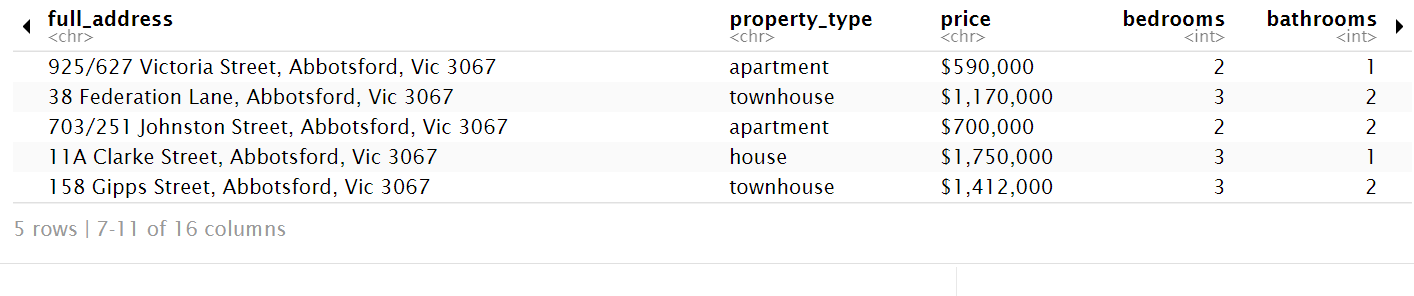
dim(ptv\_data\_vic)

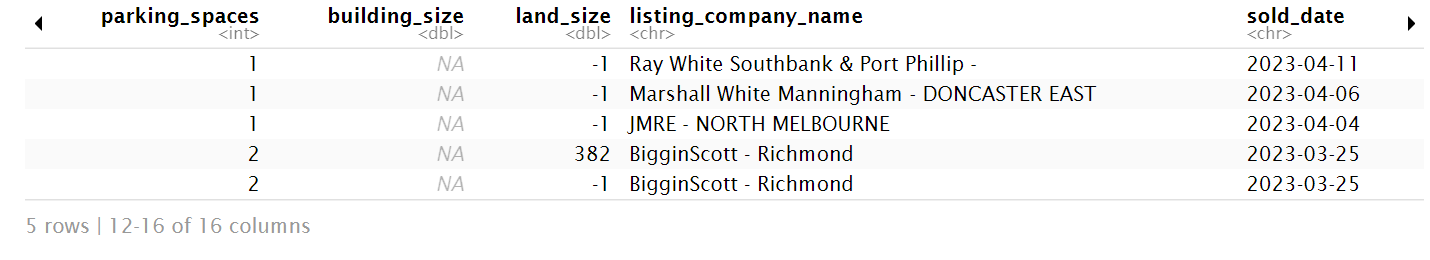
# Display the first 5 lines

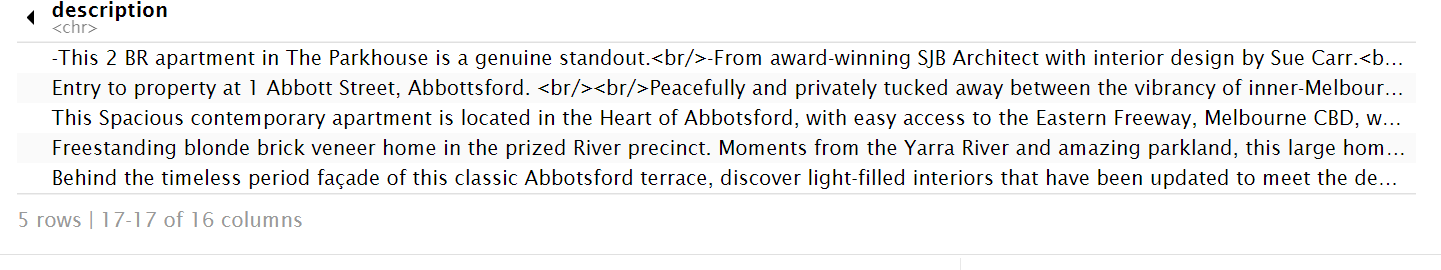
head(ptv\_data\_vic, 5)

Answer:









Explanation:

The code uses the dplyr package to remove unnecessary columns from the "ptv\_data\_vic" data box using the select() function.

It checks the dimensions of the updated data framework.

It displays the first 5 rows of the updated data box.

Task 3

Code:

# Filter data

filtered\_data\_suburb <- ptv\_data\_vic %>%

filter(suburb %in% c('Clayton', 'Mount Waverley', 'Glen Waverley', 'Abbotsford')) %>%

filter(property\_type %in% c('apartment', 'house', 'townhouse', 'unit'))

# Remove rows with missing values

filtered\_data\_suburb\_naomit <- na.omit(filtered\_data\_suburb)

# Select the required columns

filtered\_data\_named <- filtered\_data\_suburb\_naomit %>%

select(suburb, property\_type, price)

# Convert dollar signs and commas to numeric types

filtered\_data\_named$price <- gsub('\\$', "", filtered\_data\_named$price)

filtered\_data\_named$price <- gsub(",", "", filtered\_data\_named$price)

filtered\_data\_named$price <- as.numeric(filtered\_data\_named$price)

# Remove rows with missing values

filtered\_data\_named <- na.omit(filtered\_data\_named)

# Output the filtered and cleaned data

str(filtered\_data\_named)

# Summarize the data by suburb and property type

summary\_filtered\_data <- filtered\_data\_named %>%

group\_by(suburb, property\_type) %>%

summarise(

Max\_Price = max(price),

Min\_Price = min(price),

Mean\_Price = mean(price),

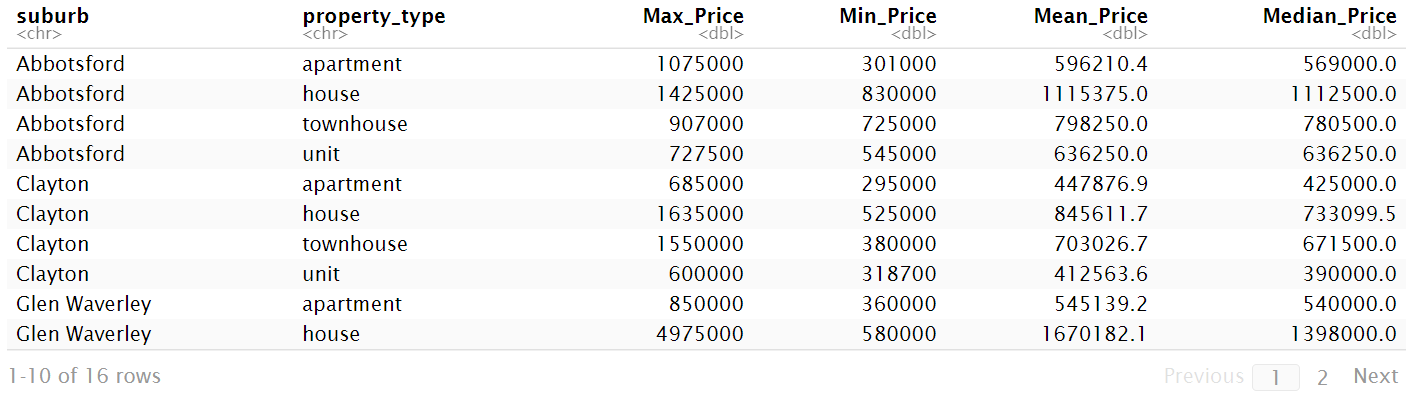
Median\_Price = median(price)

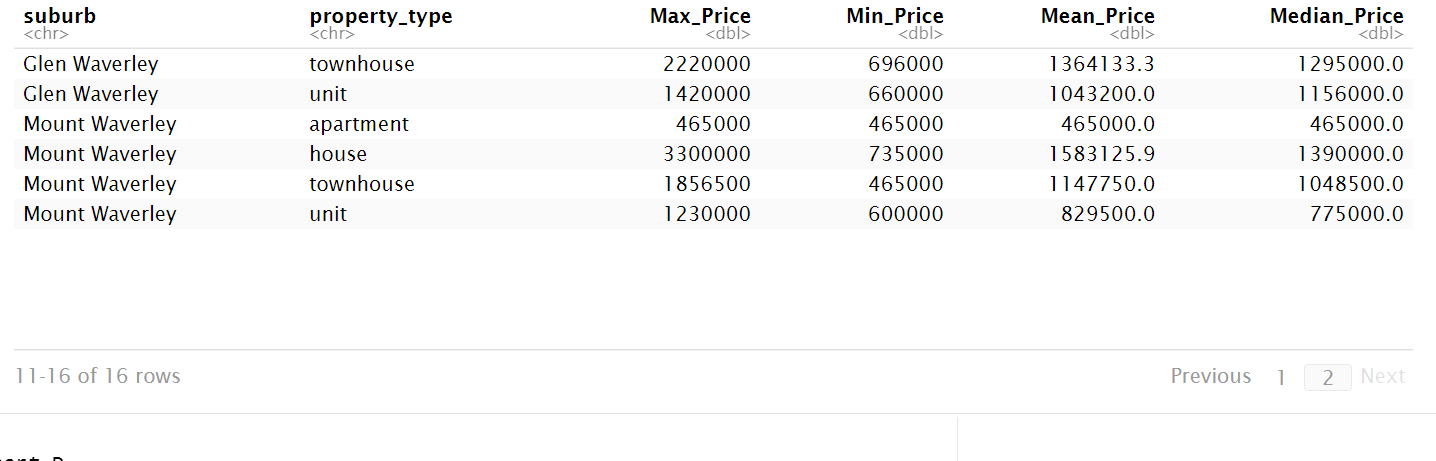
)

# Output summary\_filtered\_data

summary\_filtered\_data

Answer:





Explanation:

Use the filter() function in dplyr to filter the data by specific suburb and attribute type.

Use the na.omit() function to remove missing rows.

Use the select() function to select the required columns.

Clean up the "price" column by removing special characters and converting them to numbers.

Rows with missing values are deleted again.

Print a structure for filtering and cleaning data.

The data was summarized by suburb and property type, and the highest, lowest, average and median prices were calculated.

Task 4

Code:

# See the number of missing values in each column

missing\_value\_counts <- colSums(is.na(ptv\_data\_vic))

# Calculate the percentage of missing values

missing\_value\_percent <- missing\_value\_counts / nrow(ptv\_data\_vic)

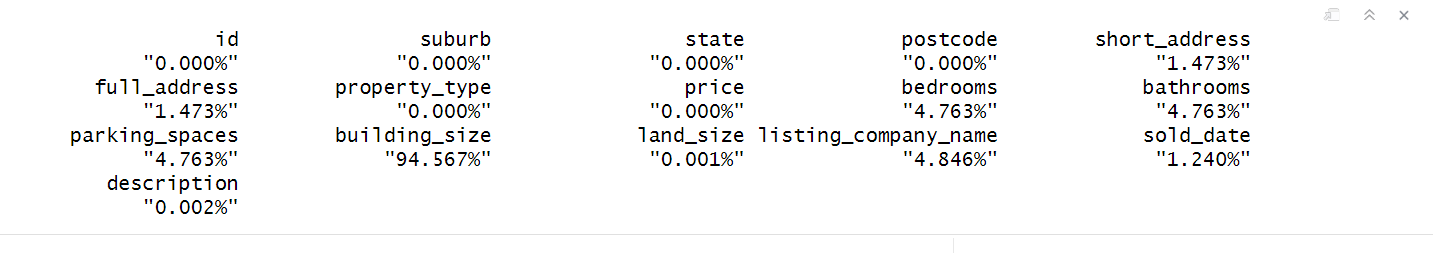
# Conversion percentage value

missing\_value\_percent <- percent(missing\_value\_percent, accuracy = 0.001)

# output missing\_value\_percent value

missing\_value\_percent

Answer:



Explanation:

Use the colsum() and is.na() functions to view and count the number of missing values in each column. The percentage of missing values in each column is then calculated. The percent() function in the scales package is also used to format the percentage value. The percentage of the last output missing value.

Task 5

Code:

# View the structure of the data

str(ptv\_data\_vic)

# Change the "sold\_date" column to Date style

ptv\_data\_vic$sold\_date <- as.Date(ptv\_data\_vic$sold\_date)

# Check the structure of the updated data

str(ptv\_data\_vic)

# Processing date formatting

Sys.setlocale("LC\_TIME", "English")

# Processing year, month, week, day

ptv\_data\_vic$month <- format(ptv\_data\_vic$sold\_date, "%m")

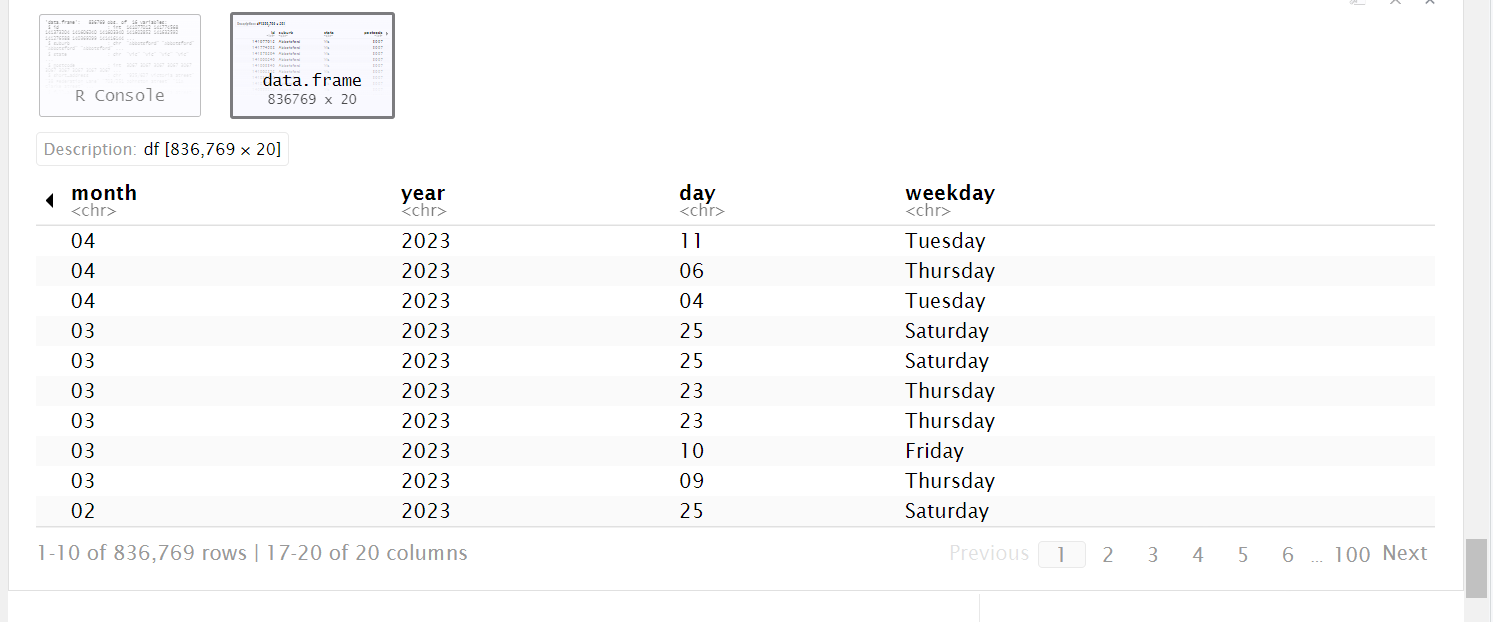
ptv\_data\_vic$day <- format(ptv\_data\_vic$sold\_date, "%d")

ptv\_data\_vic$weekday <- format(ptv\_data\_vic$sold\_date, "%A")

ptv\_data\_vic$year <- format(ptv\_data\_vic$sold\_date, "%Y")

ptv\_data\_vic

Answer:



Explanation:

Examine the structure of the raw data.

Convert the "sold\_date" column to Date format using the as.Date() function.

Check the structure of the updated data.

Use Sys.setlocale() to set the locale for the date format to English.

Use the format() function to extract the year, month, day, and day from the "sold\_date" column.

Task 6

Code:

# Sort the data by sold date

ptv\_sorted <- ptv\_data\_vic %>% arrange(sold\_date)

# Delete the null sold\_date value

unique\_dates <- ptv\_sorted$sold\_date %>% na.omit()

# Print the unique dates

unique(unique\_dates)

# Print the earliest and latest dates

earliest\_date <- head(unique\_dates, 1)

latest\_date <- tail(unique\_dates, 1)

earliest\_date

latest\_date

# Calculate the yearly trend

yearly\_trend\_data <- ptv\_data\_vic %>%

na.omit() %>%

group\_by(year) %>%

summarise(count = n())

# Chart yearly trend

ggplot(yearly\_trend\_data, aes(x = year, y = count, group = 1, color = 'red')) +

geom\_line() +

labs(title = 'Yearly Data Trend Chart', x = 'Year', y = 'Count')

# Calculate the monthly trend

monthly\_trend\_data <- ptv\_data\_vic %>%

na.omit() %>%

group\_by(month) %>%

summarise(count = n())

# Chart monthly trend

ggplot(monthly\_trend\_data, aes(x = month, y = count, group = 1)) +

geom\_line() +

labs(title = 'Monthly Data Trend Chart', x = 'Month', y = 'Count')

# Calculate the weekday trend

weekday\_trend\_data <- ptv\_data\_vic %>%

na.omit() %>%

group\_by(weekday) %>%

summarise(count = n())

# Chart weekday trend

ggplot(weekday\_trend\_data, aes(x = weekday, y = count, group = 1)) +

geom\_line() +

labs(title = 'Weekday Data Trend Chart', x = 'Weekday', y = 'Count')

# Calculate the daily trend

daily\_trend\_data <- ptv\_data\_vic %>%

na.omit() %>%

group\_by(day) %>%

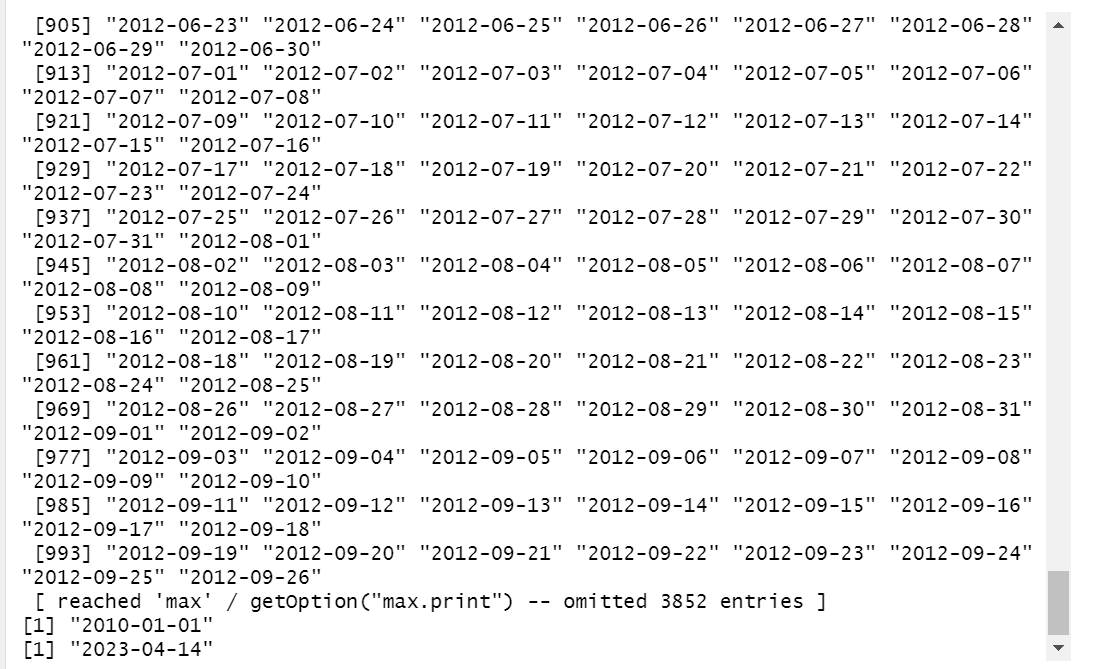
summarise(count = n())

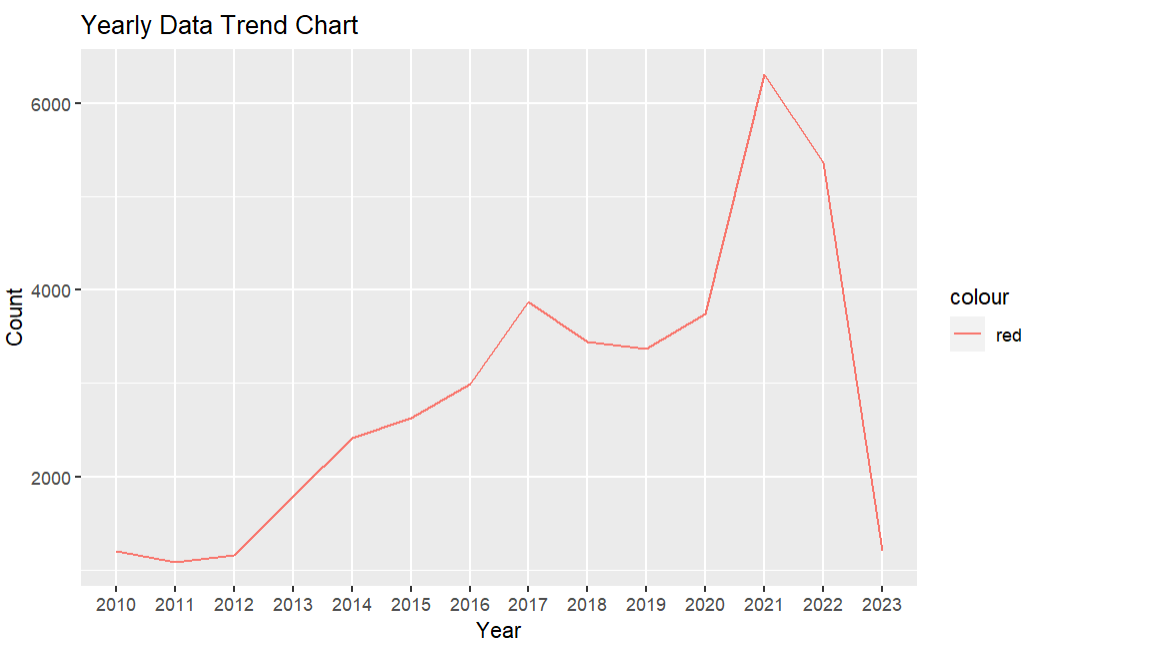
# Chart daily trends

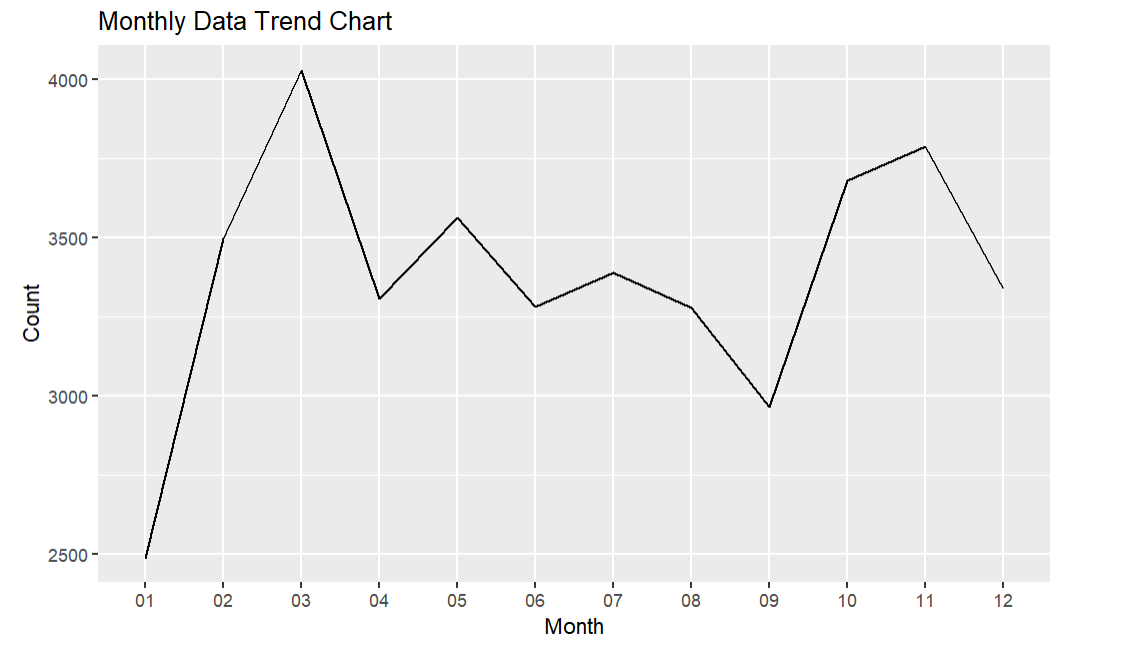
ggplot(daily\_trend\_data, aes(x = day, y = count, group = 1)) +

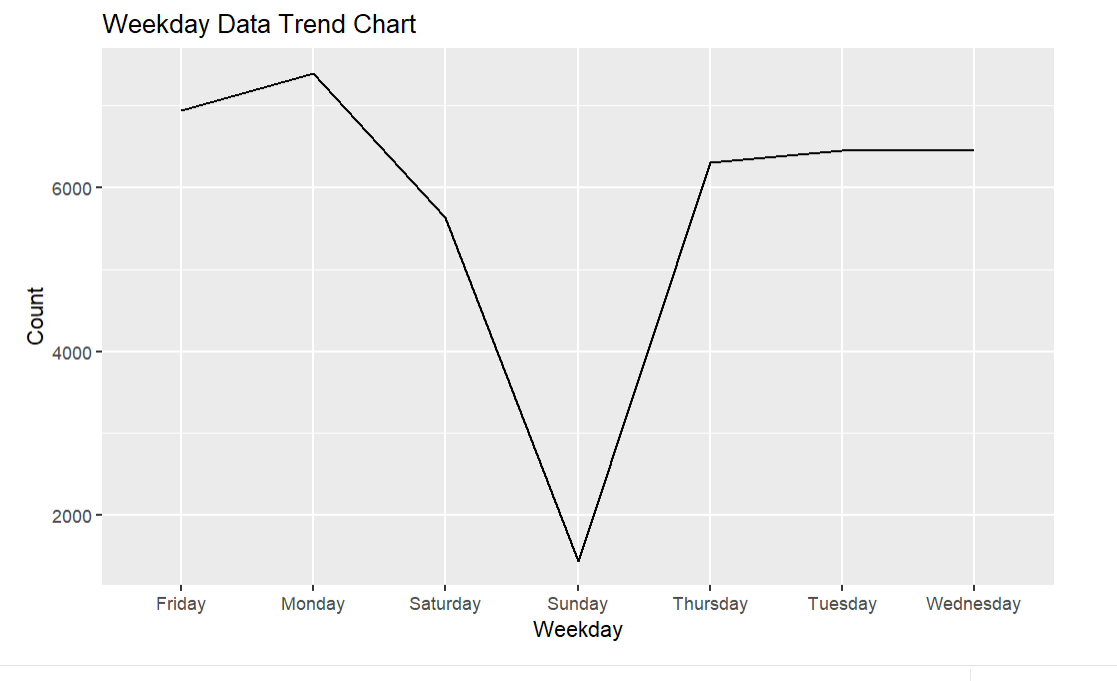
geom\_line() +

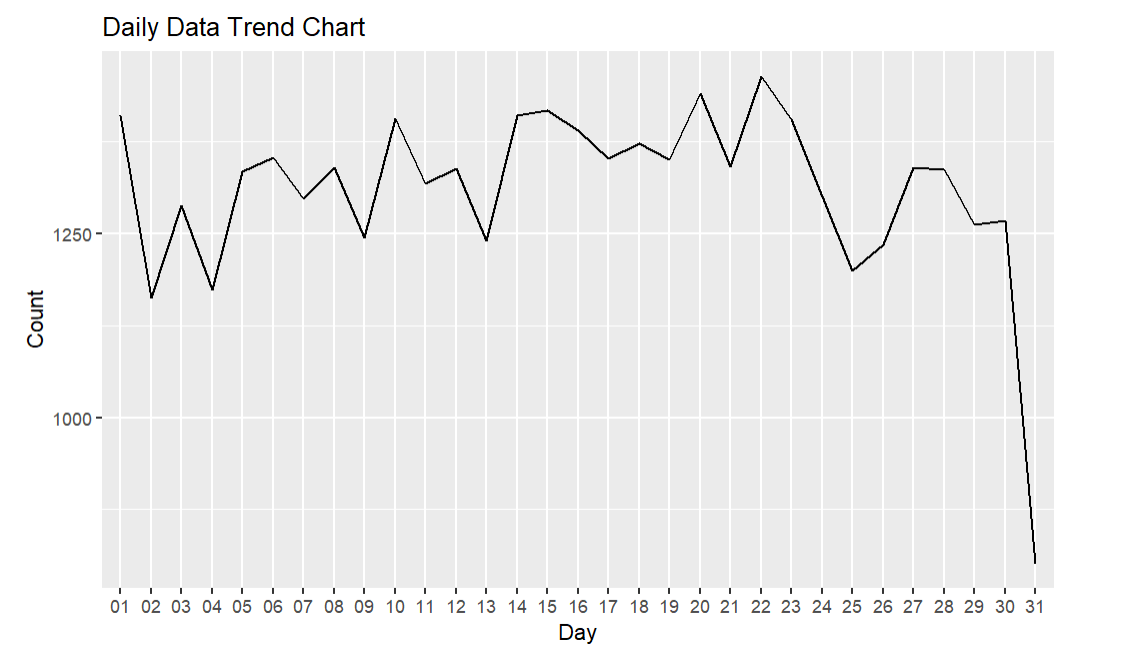
labs(title = 'Daily Data Trend Chart', x = 'Day', y = 'Count')

Answer: 









Explanation:

Sort the data by date of sale.

Extract unique dates without losing values.

Print unique dates, as well as the earliest and latest dates.

Annual trends are calculated by grouping data by year and counting the number of transactions.

Plot the annual trend using ggplot and geom\_line().

Calculate monthly, weekday, and daily trends in a similar way and plot them using ggplot

Task 7

Code:

# Filtered data

ptv\_2022\_vic\_data <- ptv\_data\_vic %>%

filter(year == "2022" & property\_type %in% c('apartment', 'house', 'townhouse', 'unit'))

# Group data by month and property and calculate the number of transactions

ptv\_2022\_vic\_data\_count <- ptv\_2022\_vic\_data %>%

group\_by(month, property\_type) %>%

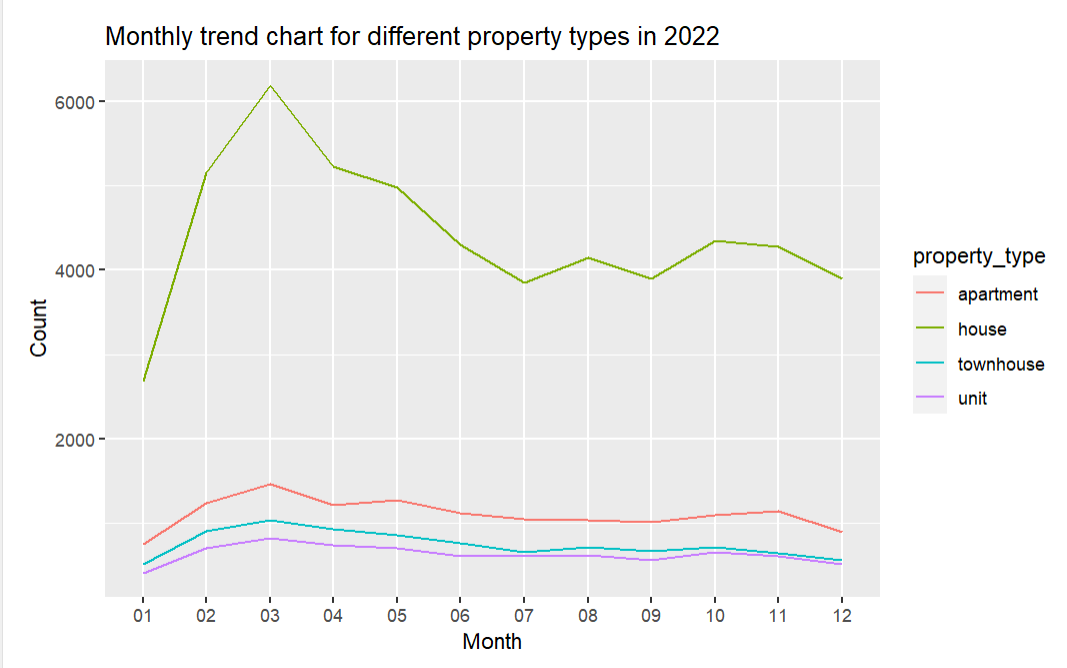
summarise(count = n(), .groups = 'drop')

# Trend mapping

ggplot(ptv\_2022\_vic\_data\_count, aes(x = month, y = count, group = property\_type, color = property\_type)) +

geom\_line() +

labs(title = 'Monthly trend chart for different property types in 2022', x = 'Month', y = 'Count')Answer:



Explanation:

Filter data for 2022 and specific attribute types.

Group the data by month and attribute type, and calculate the transaction count.

Using ggplot and geom\_line(), monthly trends for different property types in 2022 are plotted.

Task 8

Code:

# Remove the dollar sign and comma and convert them to a number type

ptv\_2022\_data\_vic\_price <- ptv\_data\_vic

ptv\_2022\_data\_vic\_price$price <- gsub('\\$', "", ptv\_2022\_data\_vic\_price$price)

ptv\_2022\_data\_vic\_price$price <- gsub(",", "", ptv\_2022\_data\_vic\_price$price)

ptv\_2022\_data\_vic\_price$price <- as.numeric(ptv\_2022\_data\_vic\_price$price)

ptv\_2022\_data\_vic\_price <- na.omit(ptv\_2022\_data\_vic\_price, cols = "price")

# Filter data

ptv\_2022\_data\_vic\_price <- ptv\_2022\_data\_vic\_price %>%

filter(year == "2022" & property\_type %in% c('apartment', 'house', 'townhouse', 'unit'))

# Group by property type and month and calculate total and average prices

ptv\_2022\_data\_vic\_price <- ptv\_2022\_data\_vic\_price %>%

group\_by(property\_type, month) %>%

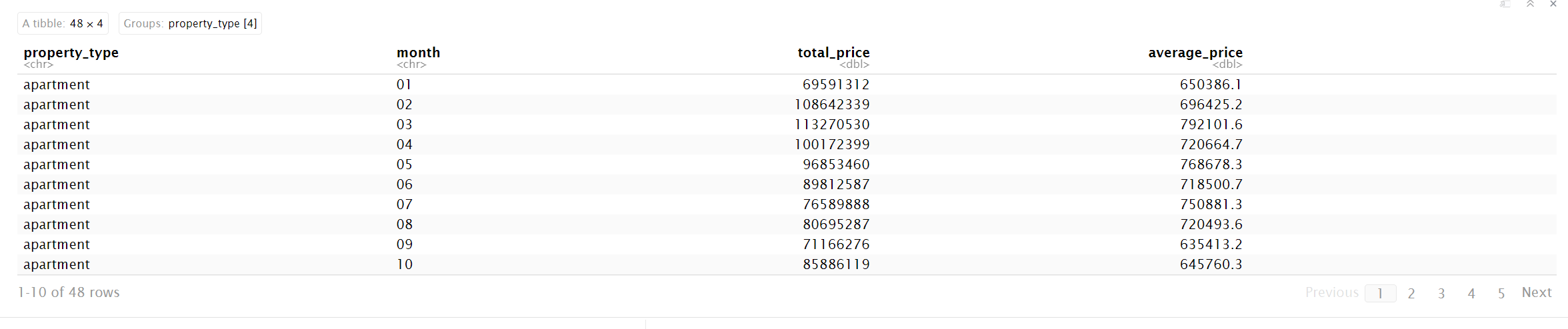
summarise(

total\_price = sum(price),

average\_price = mean(price)

)

ptv\_2022\_data\_vic\_priceAnswer:



Explanation:

Removes special characters from the price column and converts them to numbers.

Filter 2022 and specific property type data and remove NA values.

The data is grouped by property type and month, and total and average prices are calculated.

Task 9.1

Code:

# Filter data for the year 2022

ptv\_2022\_data\_vic\_suburb <- ptv\_data\_vic %>%

filter(year == '2022') %>%

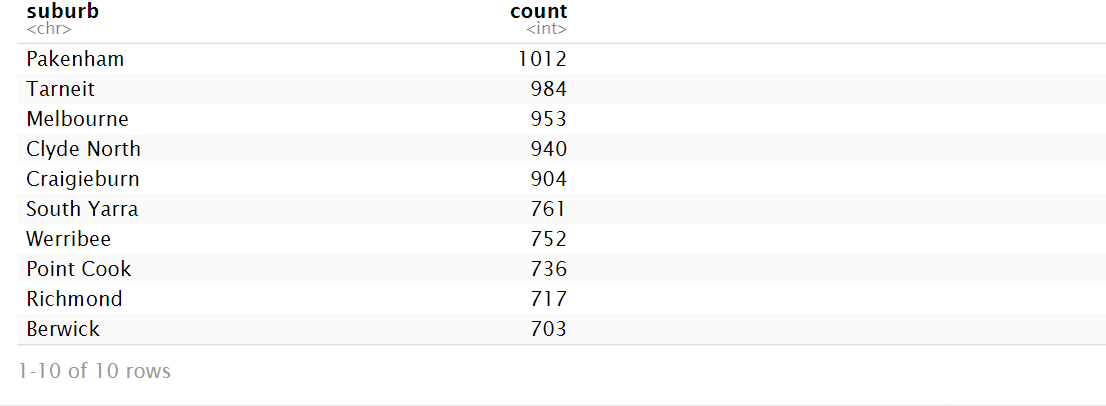
group\_by(suburb) %>%

summarise(count = n(), .groups = 'drop') %>%

arrange(desc(count))

# Top 10 suburbs with the most output

head(ptv\_2022\_data\_vic\_suburb, 10)

Answer: 

Explanation:

I first sifted through the 2022 data and grouped it by suburb, calculating the number of transactions in each suburb. The top 10 suburbs with the highest number of transactions are then displayed in descending order by number of transactions.

Task 9.2

Code:

# Get the top 10 suburbs

vic\_data\_top\_10 <- head(ptv\_2022\_data\_vic\_suburb, 10)

vic\_data\_top\_10\_suburb <- vic\_data\_top\_10$suburb

# Filter data for the top 10 suburbs

ptv\_2022\_vic\_data\_top\_10 <- ptv\_2022\_vic\_data %>%

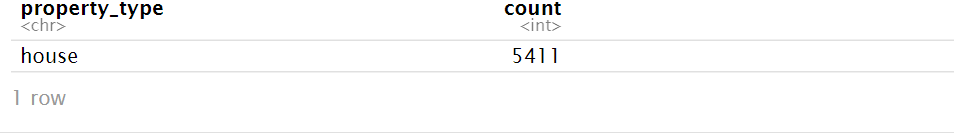
filter(suburb %in% vic\_data\_top\_10\_suburb) %>%

group\_by(property\_type) %>%

summarise(count = n(), .groups = 'drop') %>%

arrange(desc(count))

# Output the most data

head(ptv\_2022\_vic\_data\_top\_10, 1)Answer: 

Explanation:

I extracted data from the top 10 suburbs with the highest number of transactions and filtered it based on those suburbs. It was then grouped by property type to calculate the number of transactions for each property type in these suburbs. Finally, the number of transactions is ranked in descending order, and the most traded property types in these suburbs are output

Task 9.3

Code:

# Group and calculate transaction counts by suburb and property type

ptv\_2022\_top\_10\_data <- ptv\_2022\_vic\_data %>%

filter(suburb %in% vic\_data\_top\_10\_suburb) %>%

group\_by(suburb, property\_type) %>%

summarise(count = n(), .groups = 'drop')

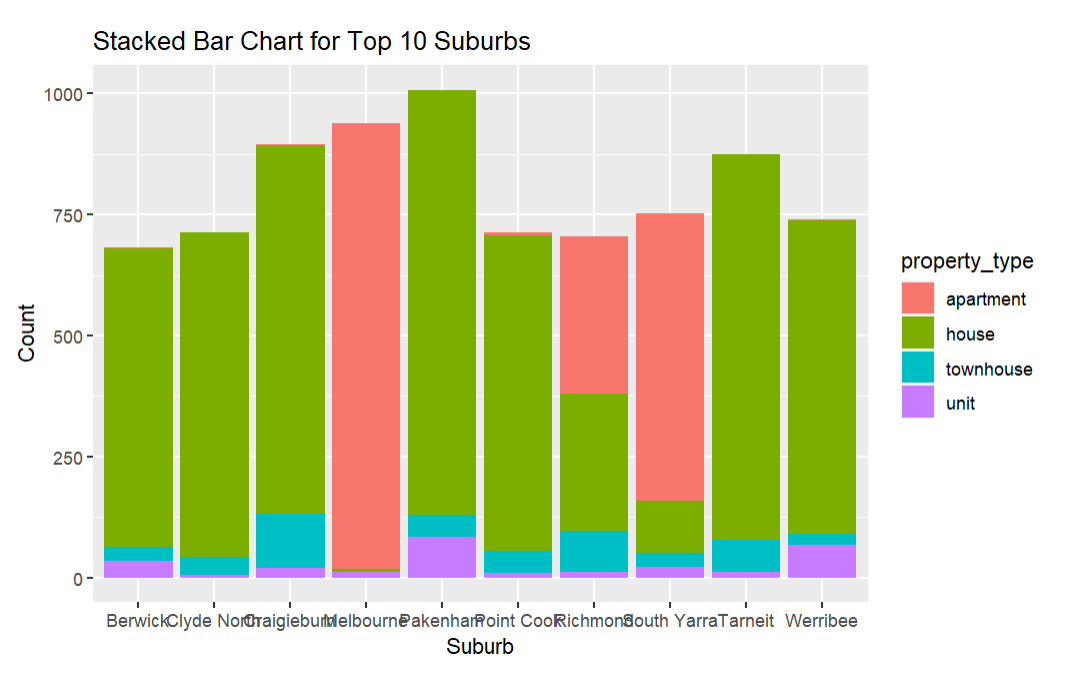
# Draw a stacked bar chart

ggplot(ptv\_2022\_top\_10\_data, aes(x = suburb, y = count, fill = property\_type)) +

geom\_bar(stat = "identity") +

labs(title = "Stacked Bar Chart for Top 10 Suburbs", x = "Suburb", y = "Count")

Answer:



Explanation:

Again, I screened the data based on the top 10 suburbs with the highest number of transactions, grouped them by suburb and property type, and calculated the number of transactions for different property types in each suburb. A stacked bar chart was then used to visualize the number of transactions across property types in these suburbs

Task 10.1

Code:

# Select the suburbs of interest

suburbs <- c('Kew', 'South Yarra', 'Caulfield', 'Clayton', 'Glen Waverley', 'Burwood', 'Abbotsford')

# Filter data for property type 'house' and the selected suburbs

ptv\_house <- ptv\_data %>%

filter(property\_type == 'house' & suburb %in% suburbs)

# Remove the dollar and comma characters and convert them to numbers

ptv\_house$price <- gsub('\\$', "", ptv\_house$price)

ptv\_house$price <- gsub(",", "", ptv\_house$price)

ptv\_house$price <- as.numeric(ptv\_house$price)

# Delete rows that are missing values

ptv\_house\_new <- ptv\_house[, c("suburb", "price", "parking\_spaces", "bathrooms", "land\_size", "bedrooms")]

ptv\_house\_new <- na.omit(ptv\_house\_new)

# Calculate the mean and median of different variables

summary\_stats <- ptv\_house\_new %>%

group\_by(suburb) %>%

summarise(

mean\_bedrooms = mean(bedrooms),

mean\_bathrooms = mean(bathrooms),

mean\_parking\_spaces = mean(parking\_spaces),

mean\_land\_size = mean(land\_size),

mean\_price = mean(price),

median\_bedrooms = median(bedrooms),

median\_bathrooms = median(bathrooms),

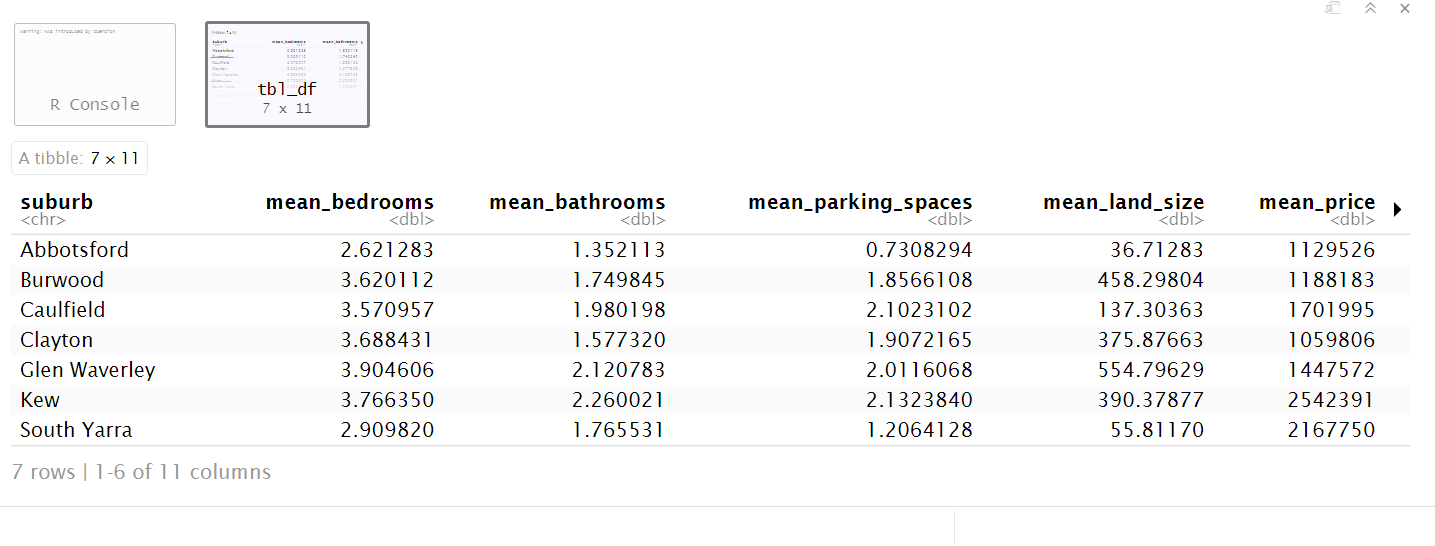
median\_parking\_spaces = median(parking\_spaces),

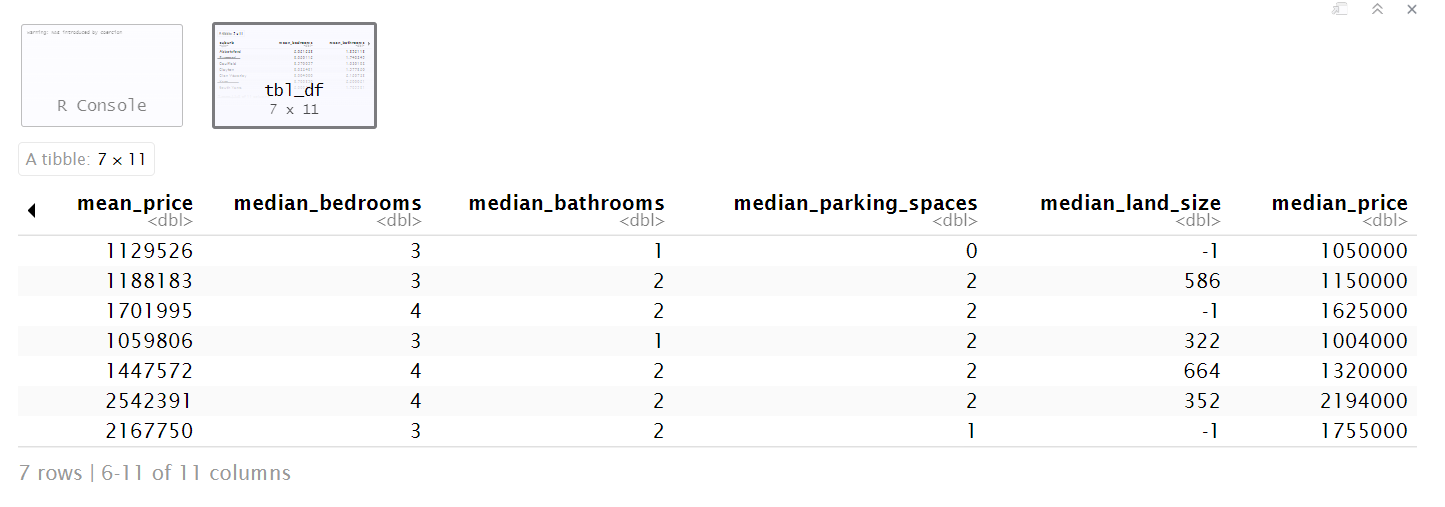
median\_land\_size = median(land\_size),

median\_price = median(price)

)

summary\_statsAnswer:





Explanation:

I selected some suburbs and filtered out the data with the attribute type "house". Then, we remove the dollar and comma characters from the price column and convert them to numeric types. Next, select the relevant columns and delete the rows with missing values. Finally, they were grouped by suburb, and the mean and median of the different variables were calculated.

Task 10.2

Code:

# Calculate correlations between different variables

ptv\_house\_cor <- ptv\_house %>%

group\_by(suburb) %>%

summarise(

cor\_bedrooms\_bathrooms = cor(bedrooms, bathrooms),

cor\_bathrooms\_price = cor(bathrooms, price),

cor\_bedrooms\_parking\_spaces = cor(bedrooms, parking\_spaces),

cor\_parking\_spaces\_land\_size = cor(parking\_spaces, land\_size),

cor\_bedrooms\_land\_size = cor(bedrooms, land\_size),

cor\_bedrooms\_price = cor(bedrooms, price),

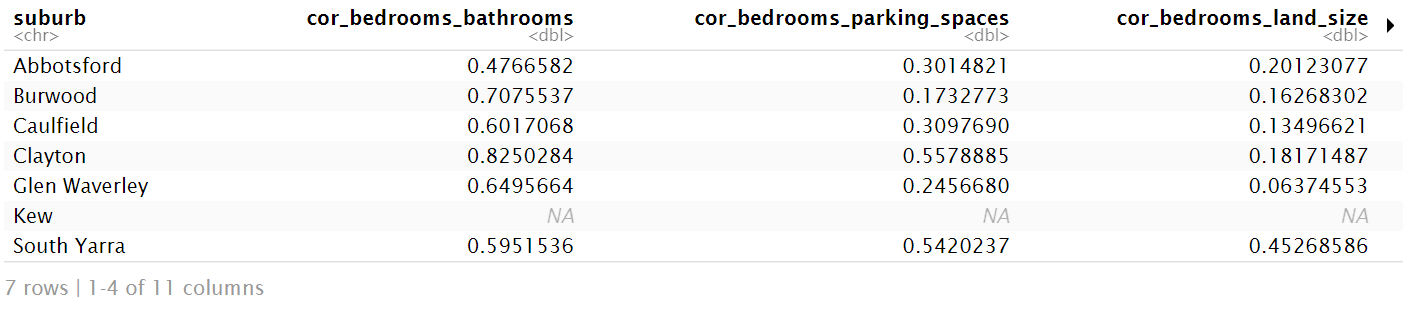
cor\_bathrooms\_parking\_spaces = cor(bathrooms, parking\_spaces),

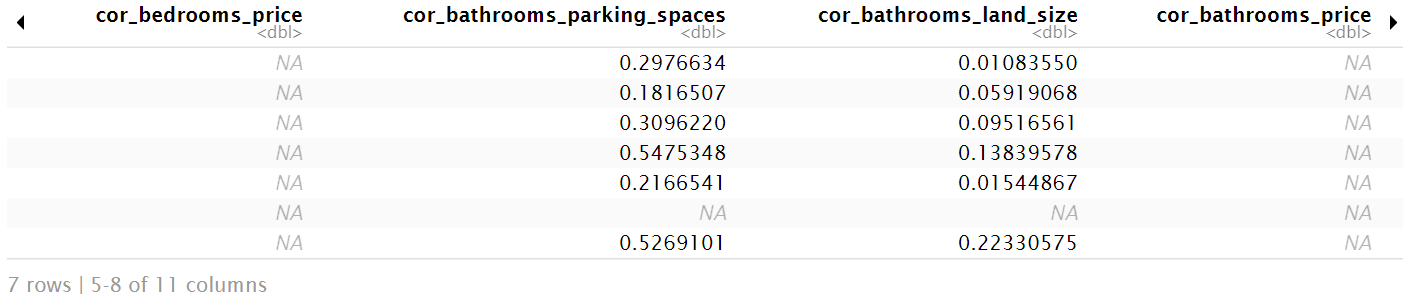
cor\_bathrooms\_land\_size = cor(bathrooms, land\_size),

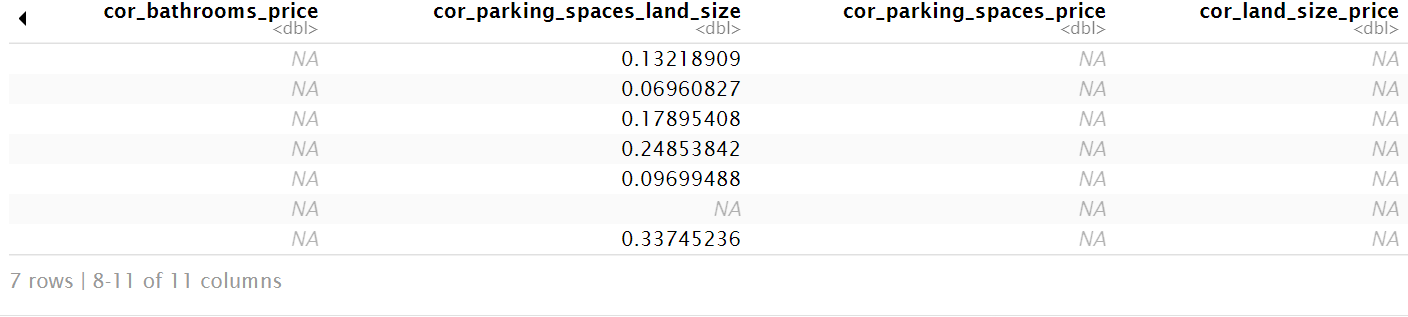
cor\_parking\_spaces\_price = cor(parking\_spaces, price),

cor\_land\_size\_price = cor(land\_size, price)

)

ptv\_house\_cor





Explanation:

Grouped by suburb, I calculated the correlation between the different variables

Task 11.1

Code:

# Removes missing values and calculates the length

ptv\_description <- ptv\_data\_vic %>%

na.omit() %>%

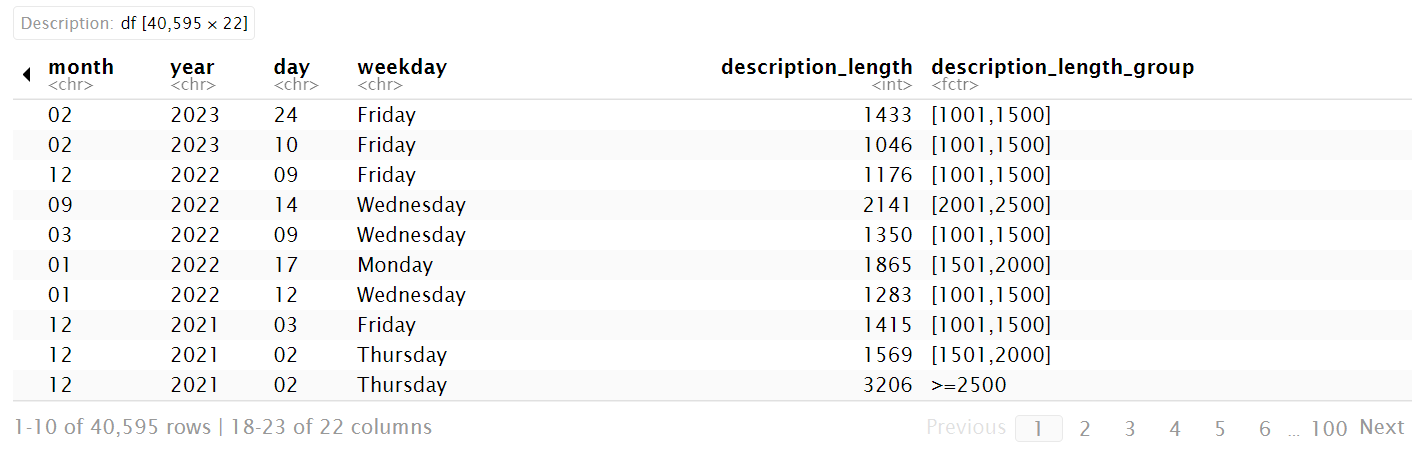
mutate(description\_length = nchar(gsub('<br/>', '', description)),

description\_length\_group = cut(description\_length, breaks = c(0, 500, 1000, 1500, 2000, 2500, Inf),

labels = c('[1,500]', '[501,1000]', '[1001,1500]', '[1501,2000]', '[2001,2500]', '>=2500')))

ptv\_description

Answer:



Explanation:

I first removed the line containing the missing value and calculated the length of the description text. Use the na.omit() function to remove missing values, and use the nchar() function to calculate the character length of the data. Before calculating the length, I use the gsub() function to replace the "<br />" tag in the description text with an empty string. Next, I use the cut() function to divide the descriptive text length into different groups, using predefined split points (0, 500, 1000, 1500, 2000, 2500, and infinity). And set a label for each group (" [1500] ", "[501100]", "[1001150]", "[1501200]", "[2001250]" and "& gt; = 2500 "). Finally, create a new data ptv\_description that contains information about the description text length and the description text length group.

Task 11.2

Code:

# Group the data and count the number of lists

ptv\_description\_length\_group <- ptv\_description %>%

group\_by(description\_length\_group) %>%

summarise(count = n(), .groups = 'drop')

# Draw bar charts

ggplot(ptv\_description\_length\_group, aes(x = description\_length\_group, y = count, fill = description\_length\_group)) +

geom\_bar(stat = 'identity') +

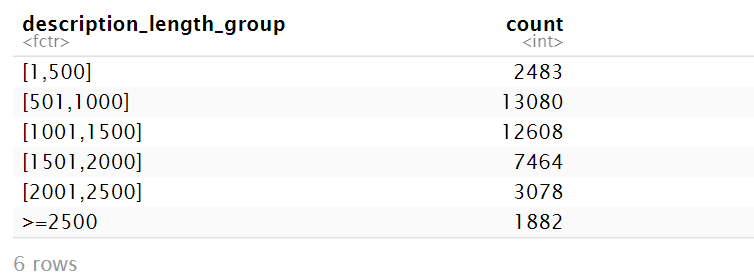
labs(x = 'Description Length Group', y = 'Count') +

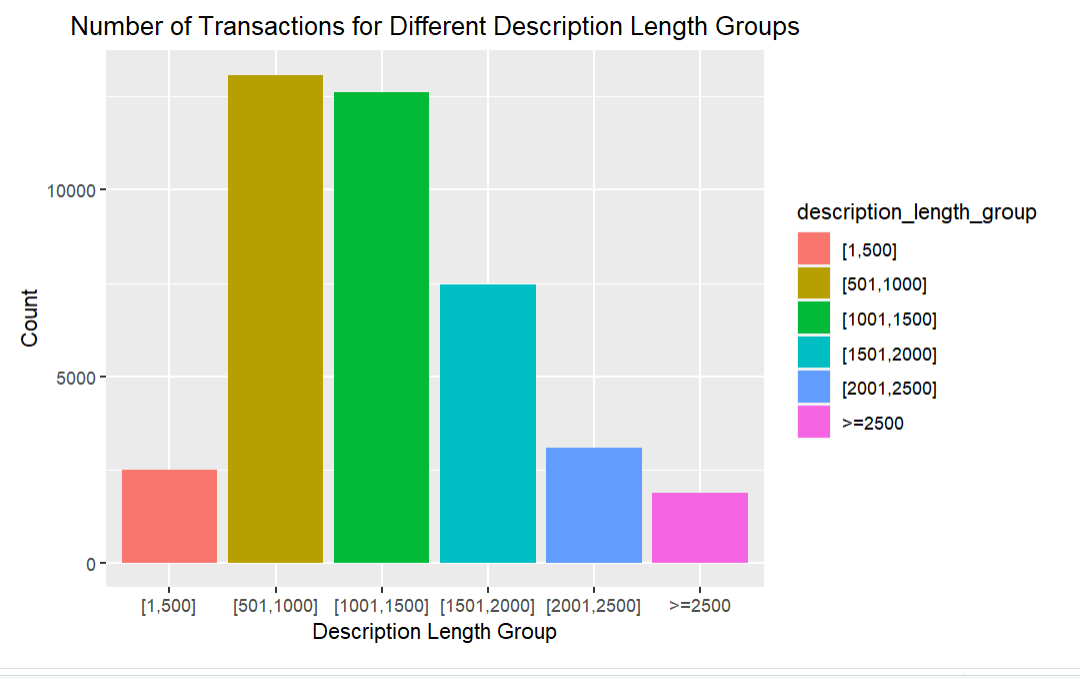
ggtitle('Number of Transactions for Different Description Length Groups') +

theme(plot.title = element\_text(hjust = 0.5))

ptv\_description\_length\_group

Answer:





Explanation:

I group the data and calculate the quantity. I use the group\_by() function to group the description text length groups, then the summarise() function to calculate the number of each group. Finally, I create a data named ptv\_description\_length\_group.

Finally, I use stacked bar charts to visualize the number of property transactions in groups of different descriptive text lengths.

Task D:

Code:

library(rpart)

library(dplyr)

library(rpart.plot)

library(tidyverse)

# Read data

train\_data <- read.csv('forum\_liwc\_train.csv')

test\_data <- read.csv('forum\_liwc\_test.csv')

# Remove rows with missing values

train\_data <- na.omit(train\_data)

test\_data <- na.omit(test\_data)

# Select independent variables

independent\_vars <- train\_data[, 2:90] # Select features from column 2 to 90 as independent variables

# Extract labels

labels <- train\_data$label

# Build Model 1

model1 <- rpart(labels ~ ., data = train\_data, method = 'class')

# The training data evaluates the performance of the model

train\_predictions <- predict(model1, train\_data, type = 'class')

train\_accuracy <- sum(train\_predictions == labels) / length(labels)

train\_accuracy

# Turn unit\_faculty and demographic\_sex into digital encodings

train\_data$unit\_faculty <- as.factor(train\_data$unit\_faculty)

train\_data$demographic\_sex <- as.factor(train\_data$demographic\_sex)

# Computational correlation

cor\_unit\_faculty <- cor(as.numeric(train\_data$unit\_faculty), as.numeric(train\_data$label))

cor\_unit\_faculty

# Computational correlation

cor\_demographic\_sex <- cor(as.numeric(train\_data$demographic\_sex), as.numeric(train\_data$label))

cor\_demographic\_sex

# Gets test\_independent\_vars and test\_labels

test\_independent\_vars <- test\_data[, 2:90]

test\_labels <- test\_data$label

# Prediction of test data

test\_predictions <- predict(model1, test\_data, type = 'class')

# Calculate the accuracy of the model

test\_accuracy <- sum(test\_predictions == test\_labels) / length(test\_labels)

test\_accuracy

# Evaluate model performance

confusion\_matrix <- table(test\_labels, test\_predictions)

precision <- confusion\_matrix[2, 2] / sum(confusion\_matrix[, 2])

recall <- confusion\_matrix[2, 2] / sum(confusion\_matrix[2, ])

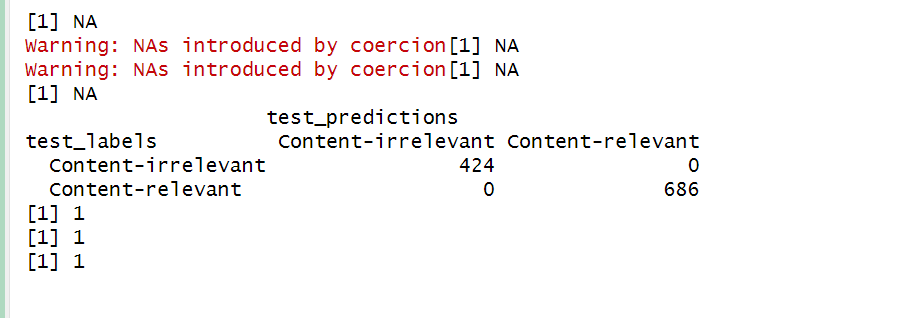
f1\_score <- 2 \* precision \* recall / (precision + recall)

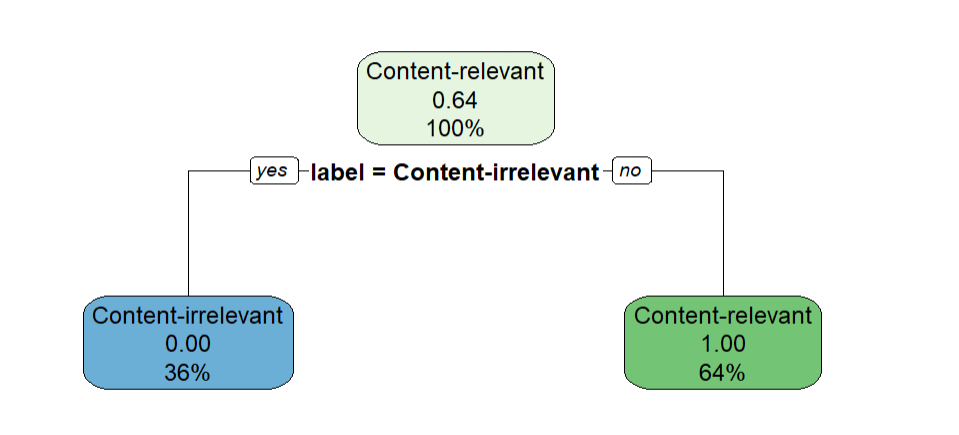
f1\_score

# Plot the decision tree model

rpart.plot(model1)

Answer:





Explanation:

Import the required R packages (rpart, dplyr, rpart.plot, and tidyverse).

Read CSV files for training and test data.

Select the independent variables in the training data and store them in the independent\_vars variable.

Extract the labels of the training data and store them in the labels variable.

Use the rpart function to build a decision tree model (Model 1) with labels and all independent variables (.) A link was established between the two.

The model was evaluated using training data. First, a prediction is made on the training data (the predict function), and then the train\_accuracy is calculated.

The representation of unit\_faculty and demographic\_sex are described. unit\_faculty may be a categorical variable, which can be represented by either unique heat coding or factorization;

Check how well the model fits the test data. First, the independent variables and labels of the test data are extracted, and then a prediction is made on the test data (the predict function).

Calculate the model's test\_accuracy on test data.

Evaluate model performance. First, create a confusion\_matrix to compare the consistency between actual and predicted labels. precision, recall, and F1 score (f1\_score) are then calculated to evaluate the performance of the model.

Finally, visualize the decision tree model using the rpart.plot function.

a. The code first reads the training and test data sets. It then selects all the independent variables as input features and uses these features to build a classification tree Model (Model 1). The performance of the model was evaluated on the training data set, and the accuracy rate was calculated as the evaluation index.

b. Analyze the correlation between features and target variables: By calculating the correlation coefficient between features and labels, you can understand the linear correlation between them. In the code, I calculated the correlation coefficient between unit\_faculty, demographic\_sex and label, because the calculation result is 1, indicating that there is a strong linear relationship between the feature and the target variable.

c. To improve the performance of Model 1

Explanation:

I used the na.omit() function to remove lines in the data that contain missing values. This ensures that the model is not disturbed by missing values during training and testing

References:

[1]<https://www.r-bloggers.com/2021/04/decision-trees-in-r/>

[2]<https://datatricks.co.uk/confusion-matrix-in-r-two-simple-methods>

[3]<https://www.digitalocean.com/community/tutorials/confusion-matrix-in-r>

[4]<https://rpubs.com/monuchacko/585317>

[5]<https://www.geeksforgeeks.org/confusion-matrix-machine-learning/>