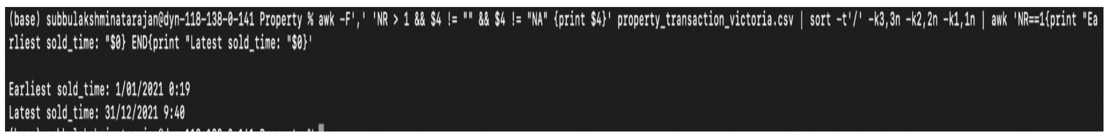
Task 1

1.

Code:

awk -F',' 'NR > 1 && $4 != "" && $4 != "NA" {print $4}' property_transaction_victoria.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | awk 'NR==1{print "Earliest sold_time: "$0} END{print "Latest sold_time: "$0}'

Output :



```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % awk -F',' 'NR > 1 && $4 != "" && $4 != "NA" {print $4}' property_transaction_victoria.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | awk 'NR==1{print "Ea
rliest sold_time: "$0} END{print "Latest sold_time: "$0}'

Earliest sold_time: 1/01/2021 0:19
Latest sold_time: 31/12/2021 9:40
```

Explanation:

awk -F',' '$4!= "" && $4!= "NA" && NR > 1. Property_transaction_victoria.csv: {print $4}

-F',': Indicates that the input is in CSV format by setting the field separator to a comma.
If NR > 1, the first record—presumably the header—is skipped.

$4!= "" and $4!= "NA": Verifies that the transaction date, which is the fourth field, is not "NA" or empty.

print $4: The fourth field is printed if the criteria are satisfied.

Command for Sorting:

| sort -t'/' -k3,3n -k2,2n -k1,1n: Connects the AWK command's output to the sort command via a pipe (|).
-t'/': This option sets the sort command's delimiter to '/', which is frequently used in date formats such as dd/mm/yyyy.
-k2,2n -k1,1n -k3,3n: Establishes the input:First, by the numerical value of the third key (year) (-k3,3n).
Next, by the numerical value of the second key (month) (-k2,2n).
Lastly, by using the initial key (day) in numerical form (-k1,1n).

Final AWK Command:

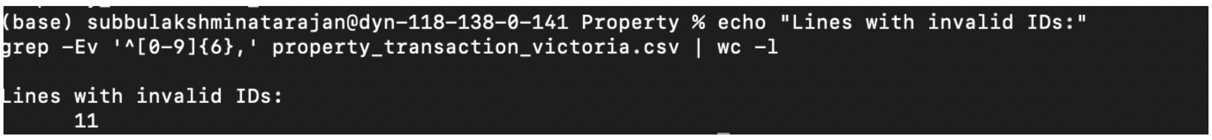
| awk 'NR==1{print{"First sold time: "$0}"} END{print "Latest sold_time: "$0}': Processes the sorted data using AWK once more.

$0}: NR==1{print "Earliest sold_time: " It outputs this record prefixed with "Earliest sold_time:" when it encounters the first record, which will be the earliest date because of sorting.
END{print "Latest sold_time: "$0}: This record is printed with "Latest sold_time: "prefixed at the end of all input, which indicates the last date owing to sorting.

2a. Code :

echo "Lines with invalid IDs:"grep -Ev '^[0-9]{6},' property_transaction_victoria.csv | wc -l

Output



```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % echo "Lines with invalid IDs:"
grep -Ev '^[0-9]{6},' property_transaction_victoria.csv | wc -l

ines with invalid IDs:
      11
```

Explanation:

echo "Lines with invalid IDs:"grep -Ev '^[0-9]{6},' property_transaction_victoria.csv | wc -l

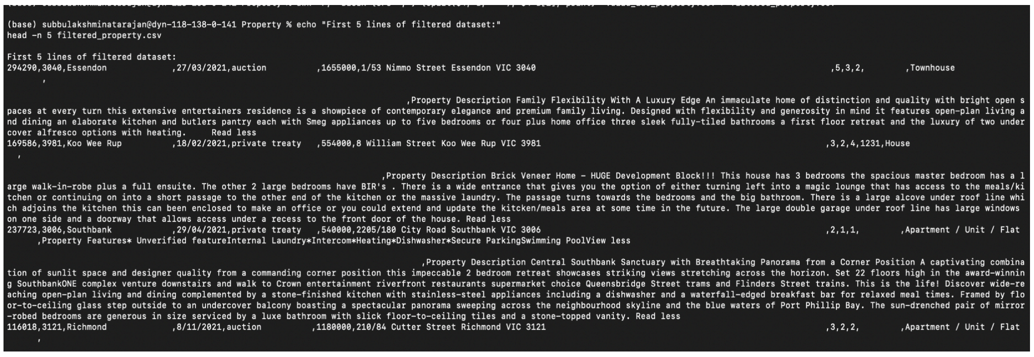
-E: Allows for more intricate patterns through extended regular expressions.
-v: Reverses the match, which means it chooses lines that don't fit the specified pattern.
"^[0–9]{6},': The pattern that corresponds. It searches for lines that begin with precisely six digits and a comma. It is believed that property_transaction_victoria.csv is an acceptable format for IDs. the document that is being looked up.

wc Command

| wc -l: This command counts the number of lines provided to it by piping (|) the output of the grep command to wc -l.
wc -l counts the number of lines, which indicates the quantity of records that, in accordance with the given pattern, have invalid IDs.

2c.
CODE:
echo "First 5 lines of filtered dataset:"head -n 5 filtered_property.csv

Output



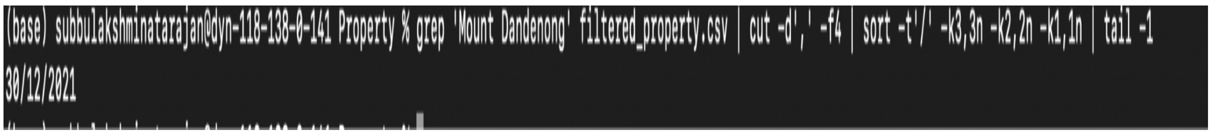
```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % echo "First 5 lines of filtered dataset:"
head -n 5 filtered_property.csv

First 5 lines of filtered dataset:
294290,3040,Essendon             ,27/03/2021,auction          ,1655000,1/53 Nimmo Street Essendon VIC 3040                                                             ,5,3,2,        ,Townhouse
       ,

                                                                                ,Property Description Family Flexibility With A Luxury Edge An immaculate home of distinction and quality with bright open s
paces at every turn this extensive entertainers residence is a showpiece of contemporary elegance and premium family living. Designed with flexibility and generosity in mind it features open-plan living a
nd dining an elaborate kitchen and butlers pantry each with Smeg appliances up to five bedrooms or four plus home office three sleek fully-tiled bathrooms a first floor retreat and the luxury of two under
cover alfresco options with heating.     Read less
169586,3981,Koo Wee Rup          ,18/02/2021,private treaty   ,554000,8 William Street Koo Wee Rup VIC 3981                                                          ,3,2,4,1231,House
  ,

                                                                           ,Property Description Brick Veneer Home - HUGE Development Block!!! This house has 3 bedrooms the spacious master bedroom has a l
arge walk-in-robe plus a full ensuite. The other 2 large bedrooms have BIR's . There is a wide entrance that gives you the option of either turning left into a magic lounge that has access to the meals/ki
tchen or continuing on into a short passage to the other end of the kitchen or the massive laundry. The passage turns towards the bedrooms and the big bathroom. There is a large alcove under roof line whi
ch adjoins the kitchen this can been enclosed to make an office or you could extend and update the kitcken/meals area at some time in the future. The large double garage under roof line has large windows
on one side and a doorway that allows access under a recess to the front door of the house. Read less
237723,3006,Southbank            ,29/04/2021,private treaty   ,540000,2205/180 City Road Southbank VIC 3006                                                          ,2,1,1,        ,Apartment / Unit / Flat
      ,Property Features* Unverified featureInternal Laundry*Intercom*Heating*Dishwasher*Secure ParkingSwimming PoolView less

                                                                                   ,Property Description Central Southbank Sanctuary with Breathtaking Panorama from a Corner Position A captivating combina
tion of sunlit space and designer quality from a commanding corner position this impeccable 2 bedroom retreat showcases striking views stretching across the horizon. Set 22 floors high in the award-winnin
g SouthbankONE complex venture downstairs and walk to Crown entertainment riverfront restaurants supermarket choice Queensbridge Street trams and Flinders Street trains. This is the life! Discover wide-re
aching open-plan living and dining complemented by a stone-finished kitchen with stainless-steel appliances including a dishwasher and a waterfall-edged breakfast bar for relaxed meal times. Framed by flo
or-to-ceiling glass step outside to an undercover balcony boasting a spectacular panorama sweeping across the neighbourhood skyline and the blue waters of Port Phillip Bay. The sun-drenched pair of mirror
-robed bedrooms are generous in size serviced by a luxe bathroom with slick floor-to-ceiling tiles and a stone-topped vanity. Read less
116018,3121,Richmond             ,8/11/2021,auction          ,1180000,210/84 Cutter Street Richmond VIC 3121                                                         ,3,2,2,        ,Apartment / Unit / Flat
      ,
```

Explanation:
A straightforward two-part Bash script is shown in the screenshot, which shows the first five lines of a dataset stored in a file named `filtered_property.csv}. To introduce the content that follows, the command begins with {echo "First 5 lines of filtered dataset:"}, which prints the provided text to the terminal as a header. This helps with readability and clarity, particularly when the output is a portion of bigger script outputs or logs. The next command that is used is `head -n 5 filtered_property.csv`. The `-n 5} option indicates that only the first five lines of the file should be displayed. `head` is a basic utility that displays the beginning of a file.When combined, these commands offer a clear and understandable overview of the dataset's inception.

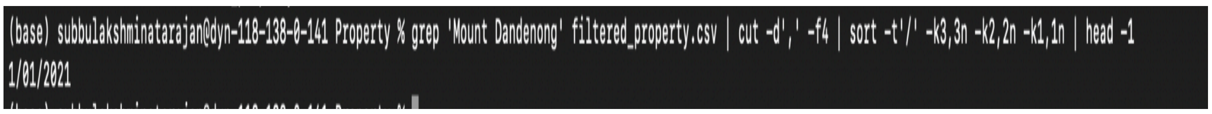
3. Code:

grep 'Mount Dandenong' filtered_property.csv | cut -d',' -f4 | sort -t'/' -k3,3n -k2,2n -k1,1n | tail -1
grep 'Mount Dandenong' filtered_property.csv | cut -d',' -f4 | sort -t'/' -k3,3n -k2,2n -k1,1n | head -1

Output



```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % grep 'Mount Dandenong' filtered_property.csv | cut -d',' -f4 | sort -t'/' -k3,3n -k2,2n -k1,1n | tail -1
30/12/2021
```

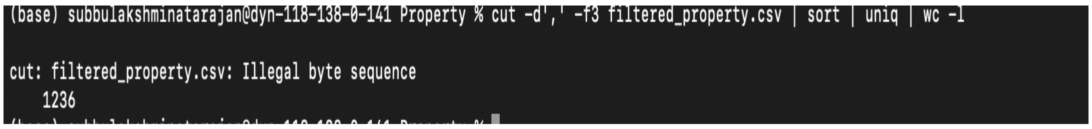


```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % grep 'Mount Dandenong' filtered_property.csv | cut -d',' -f4 | sort -t'/' -k3,3n -k2,2n -k1,1n | head -1
1/01/2021
```

Explanation:
The earliest and latest transaction dates for properties in the Mount Dandenong region can be extracted from a CSV file called `filtered_property.csv} using the commands displayed in the screenshot. The `grep` command is used at the beginning of the procedure to filter records that contain the word "Mount Dandenong". This guarantees that only pertinent data is taken into account. The fourth field of each line, which is thought to indicate transaction dates and is separated by commas, is then extracted using the cut command. After that, the `sort` command is used to arrange these dates by year, month, and day, respectively, using '/' as the

delimiter. The `tail -1` command takes the final element from this sorted list, revealing the most recent transaction date, whereas `head -1` retrieves the first, showing the earliest transaction date. These instructions quickly identify important dates in a given dataset, making it easier to analyse the historical sales of real estate in the chosen area.

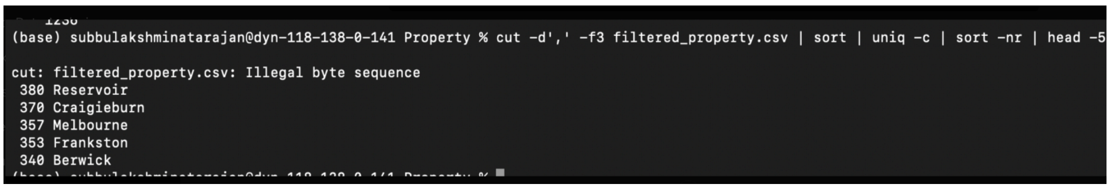
4a.CODE:
cut -d',' -f3 filtered_property.csv | sort | uniq | wc -l

Output:


```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % cut -d',' -f3 filtered_property.csv | sort | uniq | wc -l

cut: filtered_property.csv: Illegal byte sequence
    1236
```

Explanation:
In order to handle and analyse data from a CSV file named `filtered_property.csv}, the command sequence shown in the screenshot is intended to count the unique values in the third column. Using a comma as the delimiter, the cut command first takes the third field from each line of the CSV file. The `sort` command is then used to arrange this data alphabetically. Then, duplicate entries are removed using the {uniq} command, leaving only unique values. `wc -l} is then used to count these unique entries, producing the total number of unique values. The "Illegal byte sequence" notice, however, indicates a process error that is probably caused by a character encoding problem with the CSV file and prevents the cut command from properly processing the data. This error indicates that in order to handle the data appropriately, modifications could be needed, including modifying the encoding of the file or modifying the system locale settings.

4b. Code
cut -d',' -f3 filtered_property.csv | sort | uniq -c | sort -nr | head -5

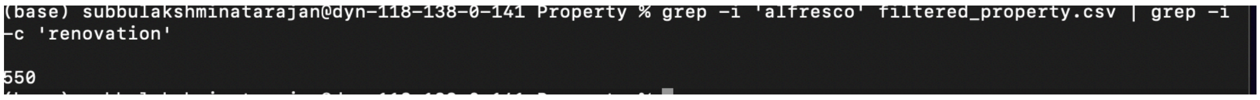
Output


```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % cut -d',' -f3 filtered_property.csv | sort | uniq -c | sort -nr | head -5

cut: filtered_property.csv: Illegal byte sequence
 380 Reservoir
 370 Craigieburn
 357 Melbourne
 353 Frankston
 340 Berwick
```

Explanation:
The command in the screenshot analyses and shows the most frequent values in the third column of the file `filtered_property.csv}. First, the cut command is used with -f3 to extract the third field from each line and -d',' to specify the delimiter as a comma. After being extracted, the data is routed into `sort`, which arranges the values in alphabetical order and gets them ready for the following stage. The next operation is `uniq -c`, which counts each unique entry and shows the count and value together. Next, `sort -nr} is used to sort this output numerically in reverse order, placing the most frequent entries at the front of the list.

Lastly, head -5 effectively identifies the five most frequent values in the dataset's third column by limiting the output to the top five entries. This order is a productive method for rapidly evaluating the most common categories or items in a given column of a big dataset. But, the programme does raise a "Illegal byte sequence" error, suggesting that there may be a problem with non-standard characters in the file. This could require troubleshooting in order to ensure that the command runs well on all computers.

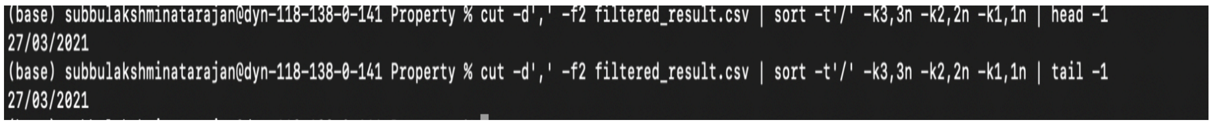
5a. Code:
grep -i 'alfresco' filtered_property.csv | grep -i -c 'renovation'

Output:


```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % grep -i 'alfresco' filtered_property.csv | grep -i
-c 'renovation'

550
```

Explanation:
Using a series of grep commands, the command seen in the screenshot filters and counts particular entries in the `filtered_property.csv` file. The first command to search for all instances of the word "alfresco" in the file is `grep -i 'alfresco' filtered_property.csv`, which ignores case sensitivity because of the `-i} argument. This guarantees that every possible combination of the word—uppercase, lowercase, or mixed—is taken into account. After that, the output of the first `grep` command—which only contains the lines that contain "alfresco"—is sent into a second `grep` command, `grep -i -c'renovation'}.Here, the -i option once more disregards case, while the -c option counts the number of lines that contain the phrase "renovation". This configuration is especially helpful for determining the frequency with which renovations are linked to alfresco features in real estate listings, as it directly provides a count of these instances in the output (550 in this case). This type of command chain works well for rapidly pulling relevant and targeted data points out of big datasets.

6b. Code
cut -d',' -f2 filtered_result.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | head -1
cut -d',' -f2 filtered_result.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | tail -1

Output


```
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % cut -d',' -f2 filtered_result.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | head -1
27/03/2021
(base) subbulakshminatarajan@dyn-118-138-0-141 Property % cut -d',' -f2 filtered_result.csv | sort -t'/' -k3,3n -k2,2n -k1,1n | tail -1
27/03/2021
```

Explanation:
The earliest and latest dates in the second column of a CSV file called filtered_result.csv are found using the instructions displayed in the screenshot. Cut -d',' -f2 filtered_result.csv, which takes the second field from each line and assumes that the fields are separated by commas, is the first command in each command sequence. It is assumed that this field contains date data.

The dates are sorted using the sort -t'/' -k3,3n -k2,2n -k1,1n command after extraction. This sort command sorts the dates first by year (-k3,3n), then by month (-k2,2n), and lastly by day (-k1,1n). It utilises a slash (/) as the delimiter, which is common for dates in the format dd/mm/yyyy. This guarantees the chronological sequence of the dates.

The first command chain concludes with | head -1, which uses the first line of the sorted list to print the earliest date. On the other hand, the second command, which takes the final line of the sorted list and finishes with | tail -1, outputs the latest date. These commands work well for rapidly determining the range of dates in a dataset; they are particularly helpful in situations where knowing the timeline of the data points is essential.

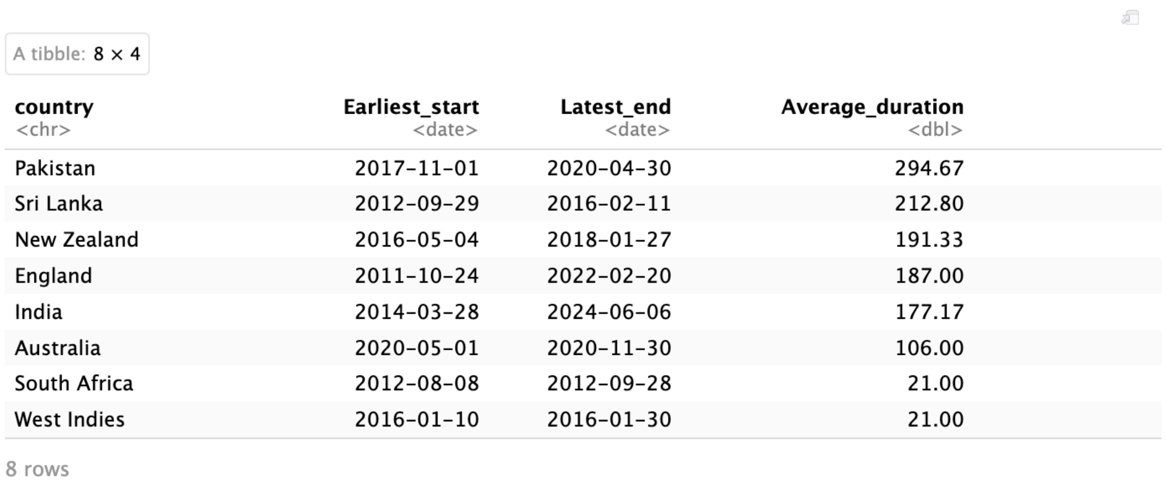
Task 2

Task B1

Code

```
# Load necessary libraries
library(rvest)
library(dplyr)
library(lubridate)
library(janitor)


# Read data from Wikipedia page
url <- "https://en.wikipedia.org/wiki/ICC_Men%27s_T20I_Team_Rankings"
page_content <- read_html(url)
tables <- page_content %>% html_table(fill = TRUE, header = TRUE)
hist_rankings <- tables[[7]]  # Ensure correct table index

# Clean column names
hist_rankings <- hist_rankings %>% clean_names()

# Remove any extraneous rows if necessary
clean_hist_rankings <- hist_rankings[-nrow(hist_rankings),]

# Clean up and manipulate date and numeric fields
clean_hist_rankings <- clean_hist_rankings %>%
  mutate(
    start = gsub("\\[.*?\\]", "", start),
    end = gsub("\\[.*?\\]", "", end),
    start = gsub("\\(.*?\\)", "", start),  # Remove any parenthetical text
    end = gsub("\\(.*?\\)", "", end),
    duration = as.numeric(gsub(" day(s)?", "", duration)),
    cumulative = as.numeric(gsub(" day(s)?", "", cumulative)),
    highest_rating = as.numeric(gsub(".*?(\\d+).*", "\\1", highest_rating)),
    start = dmy(start),
    end = dmy(end)
  )

# Replace NA in 'end' column with today's date if still NA after parsing
clean_hist_rankings$end[is.na(clean_hist_rankings$end)] <- today()

# Summary and sorting with improved handling for NAs
summary_table <- clean_hist_rankings %>%
  group_by(country) %>%
  summarize(
    Earliest_start = if(all(is.na(start))) as.Date(NA) else min(start, na.rm = TRUE),
    Latest_end = if(all(is.na(end))) as.Date(NA) else max(end, na.rm = TRUE),
    Average_duration = if(all(is.na(duration))) NA_real_ else round(mean(duration, na.rm =
TRUE), 2),
    .groups = 'drop'
  ) %>%
  arrange(desc(Average_duration))

# Display the summary table
print(summary_table)
```

Output



```
A tibble: 8 × 4

country          Earliest_start  Latest_end  Average_duration
<chr>                    <date>      <date>             <dbl>
Pakistan             2017-11-01  2020-04-30            294.67
Sri Lanka            2012-09-29  2016-02-11            212.80
New Zealand          2016-05-04  2018-01-27            191.33
England              2011-10-24  2022-02-20            187.00
India                2014-03-28  2024-06-06            177.17
Australia            2020-05-01  2020-11-30            106.00
South Africa         2012-08-08  2012-09-28             21.00
West Indies          2016-01-10  2016-01-30             21.00

8 rows
```

Explanation

Using the capabilities of multiple R libraries, the supplied R script skilfully extracts, cleans, and analyses historical T20 International cricket rankings data from a Wikipedia page. The script initially extracts tables from the homepage using `rvest`, focusing on a table that is thought to hold the pertinent rankings data. Column names are streamlined using the janitor library, which makes data handling simpler. Additional cleaning entails removing unnecessary text from date fields and using `lubridate` to transform them into correct date formats. The script then groups the data by country using `dplyr`, determining important metrics like the earliest start and latest end dates as well as the average length of each team's standings. The goal of these calculations is to determine which cricket teams have continuously maintained their rankings over time. The final product, which is arranged according to average duration, is presented in an orderly manner and offers valuable insights into the durability and consistency of the teams in the T20 rankings. This thorough technique makes it simple to identify historical trends and performances in international cricket by streamlining the extraction and analysis of extensive web-based data and presenting it in an understandable way.

Task B2

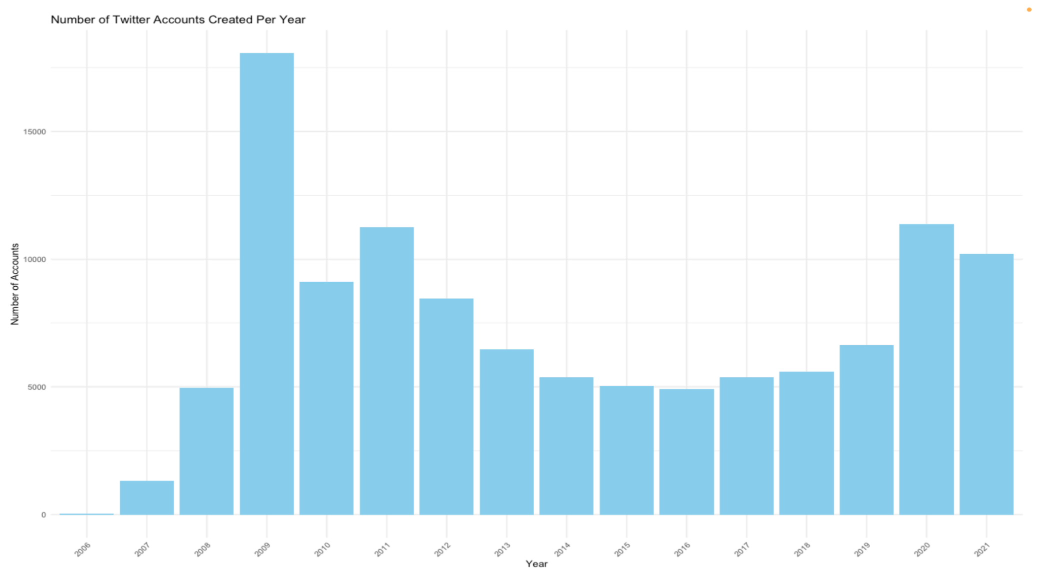
```
#Task B2
# Load necessary libraries
library(rvest)
library(dplyr)
library(ggplot2)
library(lubridate)
library(janitor)

# Step 1: Scrape Data
# Assume the correct URL to the WHO vaccination data table (this is hypothetical)
url <-
"https://www.who.int/emergencies/diseases/novel-coronavirus-2019/covid-19-vaccines"
page_content <- read_html(url)
tables <- page_content %>% html_table(fill = TRUE, header = TRUE)
vaccination_data <- tables[[1]]  # Adjust the index based on the actual table

# Step 2: Wrangle Data
# Display column names to adjust the script correctly
print(colnames(vaccination_data))

# Assuming columns for country, date of report, and number of doses administered
vaccination_data <- vaccination_data %>%
  janitor::clean_names() %>%
  filter(!is.na(total_doses_administered)) %>%
  mutate(report_date = dmy(report_date),
         total_doses_administered = as.numeric(total_doses_administered))

# Step 3: Create a Plot
ggplot(vaccination_data, aes(x = report_date, y = total_doses_administered, group = country,
color = country)) +
  geom_line() +
  labs(title = "COVID-19 Vaccination Trends",
       x = "Date",
       y = "Total Doses Administered") +
  theme_minimal()

# Step 4: Discuss the Information or Insights
# Here you would include analysis in text or comments in an R Markdown document
# Discussing the acceleration of vaccination efforts, coverage achieved, comparison between
countries, etc.
```

Task C

1.1 **Code**

```
# Load necessary libraries
library(ggplot2)

# Load the dataset

tweets_data <- read.csv("Olympics_tweets.csv")

# Convert 'user_created_at' to Date and extract the year
tweets_data$user_created_at <- as.POSIXct(tweets_data$user_created_at,
format="%d/%m/%Y %H:%M")
tweets_data$year <- format(tweets_data$user_created_at, "%Y")

# Convert the 'year' column to numeric
tweets_data$year <- as.numeric(tweets_data$year)

# 1.1 Bar chart for number of Twitter accounts created across different years
accounts_per_year <- as.data.frame(table(tweets_data$year))
colnames(accounts_per_year) <- c("Year", "Count")

ggplot(accounts_per_year, aes(x = Year, y = Count)) +
  geom_bar(stat = "identity", fill = "skyblue") +
  theme_minimal() +
  labs(title = "Number of Twitter Accounts Created Per Year", x = "Year", y = "Number of
Accounts") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Output



Explanation

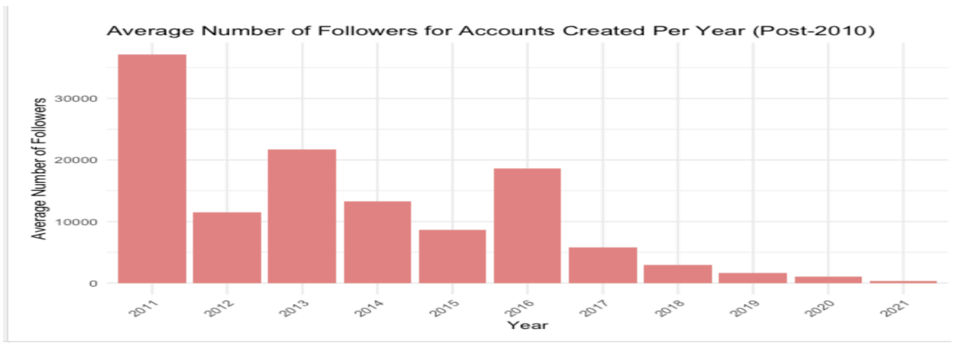
The following R code snippet uses data from a "Olympics_tweets.csv" file to visualise the total number of Twitter accounts established annually. To visualise data, the code loads the `ggplot2` package first. After that, a data frame is generated from the CSV file, and the 'user_created_at' field is processed to change its format from a string to a POSIXct date. This makes it possible to extract the year, which is then transformed for handling purposes into a numeric format. To generate a dataframe, {accounts_per_year}, the number of accounts per year is tabulated. Lastly, a bar chart is plotted using ggplot2 to display the annual number of Twitter accounts established, with aesthetic adjustments like a minimalist theme and angled text on the x-axis for easier reading.

The number of accounts every year is tabulated and stored in a dataframe called accounts_per_year. Lastly, ggplot2 is used to plot a bar chart that displays the annual number of Twitter accounts established, with aesthetic modifications such an angled x-axis text for easier reading and a minimalist style. This graphic does a good job of illustrating patterns or modifications in the establishment of accounts throughout time.

1.2 **Code**

```
# 1.2 Average number of "user_followers" for users created after 2010
tweets_data_post_2010 <- subset(tweets_data, year > 2010)

average_followers_per_year <- aggregate(user_followers ~ year, data =
tweets_data_post_2010, FUN = mean)

ggplot(average_followers_per_year, aes(x = as.factor(year), y = user_followers)) +
  geom_bar(stat = "identity", fill = "lightcoral") +
  theme_minimal() +
  labs(title = "Average Number of Followers for Accounts Created Per Year (Post-2010)", x =
"Year", y = "Average Number of Followers") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

Output



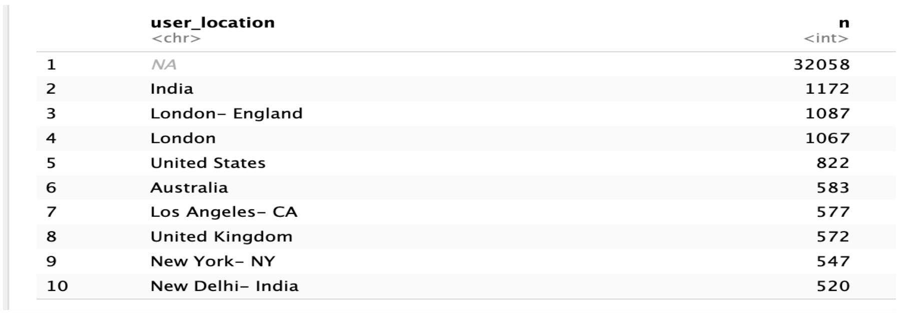
Explanation
Using information from the "Olympics_tweets.csv" file, the R code snippet is intended to examine and display the average number of followers for Twitter accounts established every year after 2010. The dataset is first filtered so that only accounts created after 2010 are included. The mean number of followers for each year is then determined by the `aggregate` function, which generates a summary dataset. Using `ggplot2}, the data is shown with the average number of followers on the y-axis and each year as a factor on the x-axis. The 'lightcoral' colour and simple tone of the final bar chart are complemented with tilted x-axis labels for easier reading.Plotting the follower numbers for newly formed accounts over time shows trends in a clear and visual manner, highlighting any notable changes or abnormalities in the data.

1.4 **Code**

```
library(dplyr)

# Count the occurrences of different location values, sorted in descending order
location_counts <- tweets_data %>%
  count(user_location, sort = TRUE)

# Get the top 10 most frequent location values
top_10_locations <- head(location_counts, 10)

# Check for any odd values in the top 10 locations
# This could involve manually inspecting or using specific criteria for what constitutes an
'odd' value.
print("Top 10 locations and their counts:")
print(top_10_locations)
```

```
# Count the total number of tweets associated with these top 10 locations
tweets_with_top_10_locations <- tweets_data %>%
  filter(user_location %in% top_10_locations$user_location) %>%
  nrow()
# Display the number of tweets associated with the top 10 locations
print(paste("Total tweets from top 10 locations:", tweets_with_top_10_locations))
```

Output



```
    user_location                 n
    <chr>                     <int>
1   NA                        32058
2   India                      1172
3   London- England            1087
4   London                     1067
5   United States               822
6   Australia                   583
7   Los Angeles- CA             577
8   United Kingdom              572
9   New York- NY                547
10  New Delhi- India            520
```

Explanation

The R script processes and analyses data from the tweets_data dataframe, concentrating on user locations, using the dplyr library. It begins by calculating the number of times each location listed in the dataframe's user_location column. It then sorts these counts descendingly and stores the outcome in location_counts. This makes it easier to determine which places in the dataset are most frequently cited by Twitter users.

The head() method, which chooses the top ten rows from the sorted location_counts, is then used by the script to extract the top ten most frequently occurring locations. The locations of the majority of users are then printed alongside their corresponding numbers, giving a clear picture of their geographic distribution.

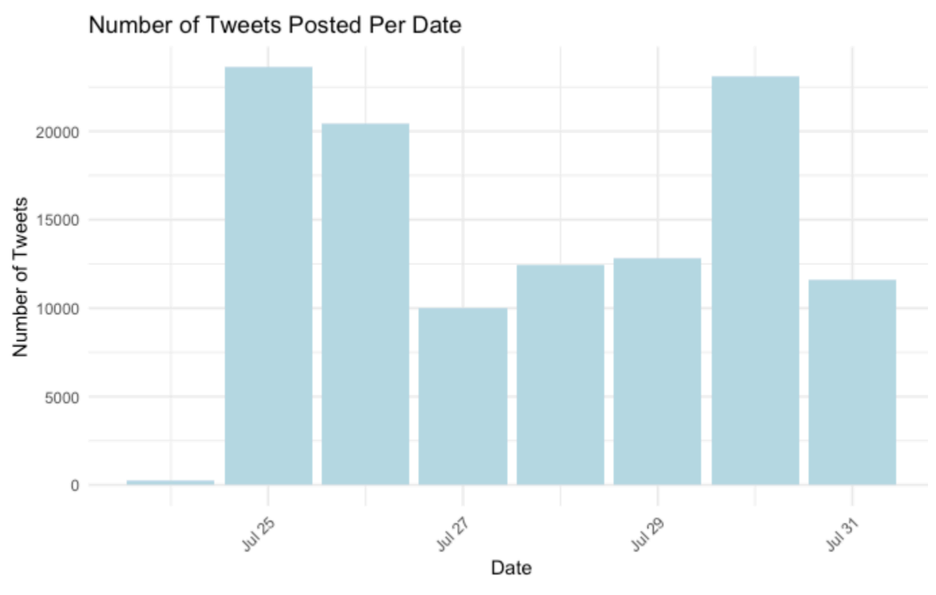
The script also determines how many tweets originate from these top 10 locations overall. In order to do this, the original tweets_data are filtered to only contain rows where the user_location corresponds to one of the top 10 places. The number of rows that remain are then counted. The total number of tweets connected to the top ten places is then printed, providing information about the volume of Twitter activity in relation to these particular regions.

All things considered, the script effectively finds and measures the distribution of users among various areas in the Twitter dataset, indicating the most well-liked regions as well as the level of Twitter activity linked to them.

2.1 **Code**

```
# Convert 'date' to Date format and extract the date part
tweets_data$date <- as.POSIXct(tweets_data$date, format="%d/%m/%Y %H:%M")
tweets_data$tweet_date <- as.Date(tweets_data$date)

# Remove NA values from 'tweet_date'
tweets_data <- tweets_data %>%
  filter(!is.na(tweet_date))

# Count the number of tweets posted on each date
tweets_per_date <- tweets_data %>%
  count(tweet_date)

# Find the date with the lowest number of tweets
lowest_tweet_date <- tweets_per_date %>%
  filter(n == min(n))

# Plotting the bar chart for the number of tweets posted on different dates
ggplot(tweets_per_date, aes(x = tweet_date, y = n)) +
  geom_bar(stat = "identity", fill = "lightblue") +
  theme_minimal() +
  labs(title = "Number of Tweets Posted Per Date", x = "Date", y = "Number of Tweets") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Display the date with the lowest number of tweets
lowest_tweet_date
```

Output



Explanation

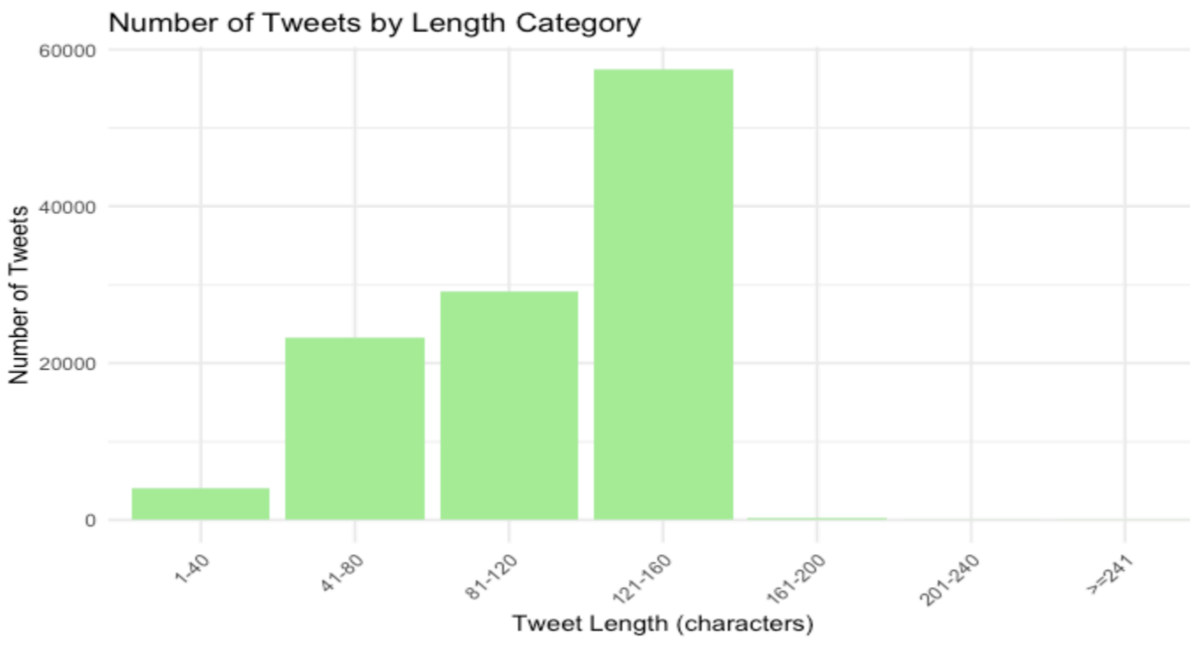
The R code snippet that is provided processes a tweet dataset in order to analyse and show the distribution of tweet volumes by date. In order to extract only the date component, it first converts the 'date' field from string format to POSIXct date-time format, which is then stored in 'tweet_date'. The 'tweet_date' field is filtered to remove any rows that have NA values, ensuring the data is clean. The script then compiles the information to determine how many tweets were posted on each date, a value that is kept in 'tweets_per_date'.

By choosing the least count value in the dataset through a filtering phase, a particular operation determines the date with the fewest tweets. 'lowest_tweet_date' contains the identified date.

To visualise the data, the script uses ggplot2 to make a bar chart with the count of tweets ('n') on the y-axis and 'tweet_date' on the x-axis. The chart has a simple design, with light blue bars and tilted date markers for easier reading. The quantity of tweets posted on various dates is represented by the graph, and this is made evident by the title and axis labels. This visual assistance makes it easier to spot patterns or unusualities in tweet activity over time. In addition, the script prints the date with the least amount of activity, giving a comprehensive picture of the dynamics of the data.

2.2 **Code**

```
# Calculate the length of the text contained in each tweet
tweets_data$text_length <- nchar(as.character(tweets_data$text))

# Define the bins for tweet lengths
bins <- c(0, 40, 80, 120, 160, 200, 240, Inf)
labels <- c('1-40', '41-80', '81-120', '121-160', '161-200', '201-240', '>=241')

# Create a new column 'length_category' with the defined bins
tweets_data$length_category <- cut(tweets_data$text_length, breaks = bins, labels = labels,
right = FALSE)

# Count the number of tweets in each length category
length_category_counts <- tweets_data %>%
  count(length_category) %>%
  arrange(length_category)

# Plotting the bar chart for the number of tweets in each length category
ggplot(length_category_counts, aes(x = length_category, y = n)) +
  geom_bar(stat = "identity", fill = "lightgreen") +
  theme_minimal() +
  labs(title = "Number of Tweets by Length Category", x = "Tweet Length (characters)", y =
"Number of Tweets") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))

# Display the counts for each length category
length_category_counts
```



Explanation

In order to classify and visualise the quantity of tweets according to their character length, the R script analysis Twitter data. It first determines the text length of each tweet and groups these lengths into preset categories, ranging from 1 to more than 241 characters. The cut function is used for the categorization, classifying each tweet according to its character count.

Following the categorization of the tweets, the script uses the dplyr count function to count the number of tweets in each category, making sure that the counts are sorted by the category names. The analysis is simple because these categories show character lengths in ascending order.

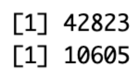
ggplot2 is used in the script's visualisation section to produce a bar chart that shows the distribution of tweets within these length categories. To improve aesthetic appeal, each bar is coloured light green and represents a particular range of tweet lengths. The plot has a minimalist theme for a tidy display, and the x-axis labels are tilted for easier reading.

According to the graphic, tweets between 81 and 120 characters are the most popular, followed by tweets between 121 and 160 characters. Very brief tweets (less than 40 characters) and very long tweets (more than 241 characters) are less common. This graphic aids in comprehending the typical tweet lengths and can offer perceptions on user conduct on Twitter, such as the inclination for succinct or thorough communication messages.

2.3 **Code**

```
# Load necessary libraries
install.packages("stringer")
library(stringr)

# Find tweets that contain at least one "@" symbol
tweets_with_mentions <- tweets_data %>%
  filter(str_detect(text, "@"))

# Count the number of tweets with mentions
num_tweets_with_mentions <- nrow(tweets_with_mentions)

# Define a function to count the number of unique mentions in a tweet
count_unique_mentions <- function(text) {
  mentions <- unique(str_extract_all(text, "@\\w+")[[1]])
  return(length(mentions))
}

# Apply the function to the tweets with mentions
tweets_with_mentions$num_mentions <- sapply(tweets_with_mentions$text,
count_unique_mentions)

# Find tweets with at least three different mentions
tweets_with_at_least_three_mentions <- tweets_with_mentions %>%
  filter(num_mentions >= 3)

# Count the number of tweets with at least three different mentions
num_tweets_with_at_least_three_mentions <- nrow(tweets_with_at_least_three_mentions)

num_tweets_with_mentions
num_tweets_with_at_least_three_mentions
```

Output



```
[1] 42823
[1] 10605
```

Explanation

The accompanying R script highlights how to use the "@" symbol to identify mentions in tweets in order to analyse Twitter data for social interactions. The script first loads and installs the stringr package, which is necessary for any job involving string manipulation. The script finds tweets that mention other users by using stringr to filter tweets that have at least one "@" symbol.

Following the identification of these tweets, the script counts the overall quantity of such tweets, yielding a numerical representation of user involvement through mentions. To find the total number of unique mentions in each tweet, it also creates and uses a custom function called count_unique_mentions. In order to comprehend deeper levels of user interaction, this programme extracts all unique instances of "@username" patterns and analyses them further.

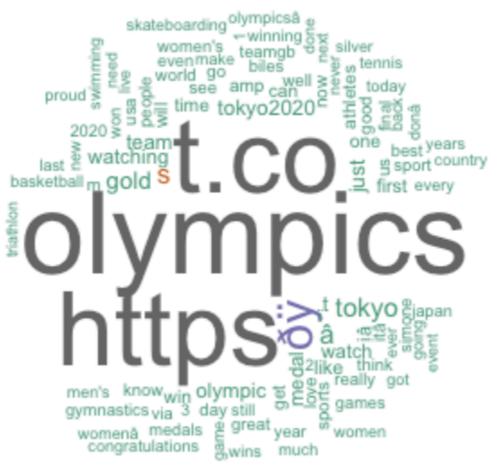
The script looks for tweets with three or more different mentions, which suggests higher levels of interaction or tweets meant for a larger network connection and flags them for additional investigation. Subsequently, a count of these tweets is computed, signifying a subset of the data including more complex user interactions.

The output values display the total number of tweets with mentions and the number of tweets with at least three separate mentions, respectively. This analysis demonstrates how to quantify basic and complex interaction patterns within Twitter data, as well as extract and assess user behaviour based on textual content in tweets, using string manipulation algorithms.

2.4 **Code**

```
# Load necessary libraries
library(tidytext)
library(dplyr)
library(tm)
library(wordcloud)
library(RColorBrewer)

# Load your tweets data if not already loaded
# tweets_data <- read.csv("path_to_your_tweets_data.csv")

# Tokenize the text into words
words <- tweets_data %>%
  unnest_tokens(word, text)

# Load stopwords from the tm package
data("stopwords")
english_stopwords <- stopwords("en")

# Filter out stopwords
filtered_words <- words %>%
  filter(!word %in% english_stopwords)

# Count the occurrences of each word, sorted by frequency
word_counts <- filtered_words %>%
  count(word, sort = TRUE)

# Create a word cloud
wordcloud(words = word_counts$word, freq = word_counts$n, min.freq = 1,
          max.words = 100, random.order = FALSE, rot.per = 0.35,
          colors = brewer.pal(8, "Dark2"))

# If you encounter memory issues, consider sampling your data
tweets_data <- tweets_data %>% sample_frac(0.5) # Adjust the fraction as needed
```

Output



Explanation:
Using Twitter data, the provided R script shows how to create a word cloud that highlights the most popular words in tweets regarding the Olympics. The first step is loading the required libraries, including RColorBrewer for colour palettes, tm for text mining, dplyr for data management, tidytext for text manipulation, and wordcloud for word cloud creation.

First, the script uses the tidytext package's unnest_tokens function to tokenize tweet text into individual words. The next step involves loading a list of stopwords in English from the TM package, which is used to exclude words like "and" and "the" that aren't particularly instructive. By eliminating certain stopwords from the dataset, the filtering procedure makes sure that the analysis is limited to pertinent terms.

The script counts the instances of each term that remains after filtering and arranges them according to frequency. Then, using this processed data, a word cloud is produced, with larger fonts representing terms that occur more frequently. The wordcloud feature is set up to show up to 100 words—chosen from RColorBrewer's "Dark2" palette—in various colours and orientations.

Focusing on key phrases such as "Olympics", "Tokyo", and "gold", the word cloud illustrates the most talked-about themes and topics in tweets about the Olympics. Using this visualisation technique, one can rapidly understand the common themes and sentiments that Twitter users were expressing during the Olympic games.

Task D

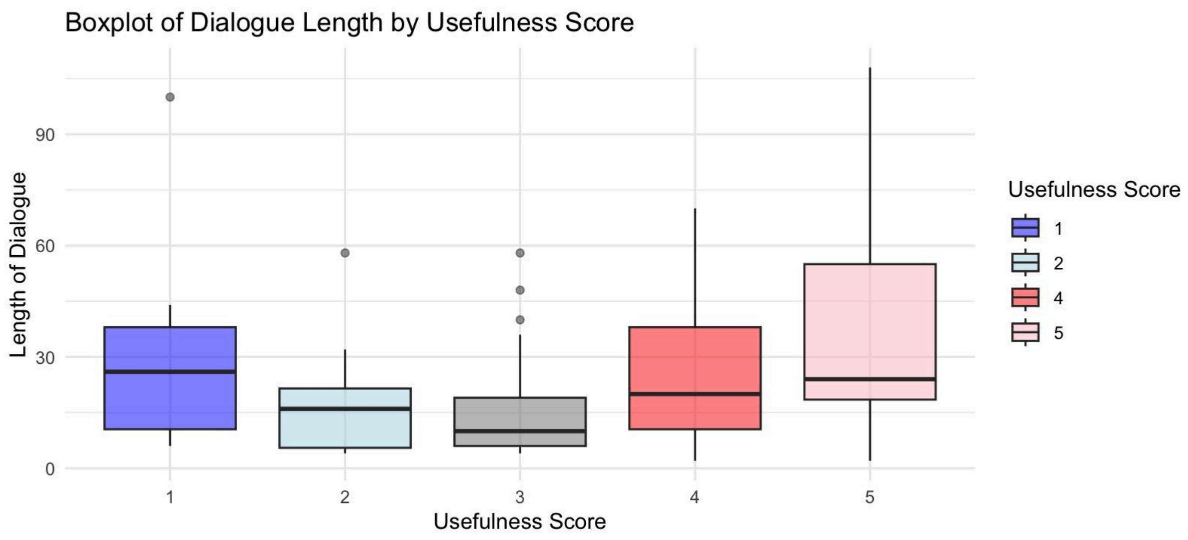
```
# Install necessary packages
install.packages("dplyr")
install.packages("readr")
install.packages("ggplot2")
install.packages("stringr")
install.packages("caret")

# Load necessary libraries
library(dplyr)
library(readr)
library(ggplot2)
library(stringr)
library(caret)

# Load datasets
utterances_train <- read_csv("dialogue_utterance_train.csv")
usefulness_train <- read_csv("dialogue_usefulness_train.csv")
utterances_validation <- read_csv("dialogue_utterance_validation.csv")
usefulness_validation <- read_csv("dialogue_usefulness_validation.csv")
utterances_test <- read_csv("dialogue_utterance_test.csv")

# Correct column names if necessary
correct_column_names <- function(df, id_pattern = "ID", text_pattern = "text") {
  if (!"Dialogue_ID" %in% colnames(df)) {
    colnames(df)[grepl(id_pattern, colnames(df), ignore.case = TRUE)] <- "Dialogue_ID"
  }
  if (!"Utterance_text" %in% colnames(df)) {
    colnames(df)[grepl(text_pattern, colnames(df), ignore.case = TRUE)] <- "Utterance_text"
  }
  return(df)
}

utterances_train <- correct_column_names(utterances_train)
```

```
usefulness_train <- correct_column_names(usefulness_train, id_pattern = "ID")
utterances_validation <- correct_column_names(utterances_validation)
usefulness_validation <- correct_column_names(usefulness_validation, id_pattern = "ID")
utterances_test <- correct_column_names(utterances_test)

# Data Preparation
train_data <- inner_join(utterances_train, usefulness_train, by = "Dialogue_ID")
validation_data <- inner_join(utterances_validation, usefulness_validation, by =
"Dialogue_ID")

# Feature Engineering
train_data <- train_data %>%
  mutate(word_count = str_count(Utterance_text, '\\w+')) %>%
  group_by(Dialogue_ID) %>%
  summarise(
    length = n(),
    avg_utterance_length = mean(word_count),
    Usefulness_score = first(Usefulness_score)
  )
# Data Preparation
train_data <- inner_join(utterances_train, usefulness_train, by = "Dialogue_ID")
validation_data <- inner_join(utterances_validation, usefulness_validation, by =
"Dialogue_ID")

# Feature Engineering
train_data <- train_data %>%
  mutate(word_count = str_count(Utterance_text, '\\w+')) %>%
  group_by(Dialogue_ID) %>%
  summarise(
    length = n(),
    avg_utterance_length = mean(word_count),
    Usefulness_score = first(Usefulness_score)
  )

# Visualization with Boxplots
ggplot(train_data, aes(x = as.factor(Usefulness_score), y = length, fill =
as.factor(Usefulness_score))) +
  geom_boxplot(alpha = 0.5) +
  labs(title = "Boxplot of Dialogue Length by Usefulness Score", x = "Usefulness Score", y =
"Length of Dialogue") +
  scale_fill_manual(name = "Usefulness Score", values = c("1" = "blue", "2" = "lightblue",
"4" = "red", "5" = "pink")) +
  theme_minimal()
```

```
# Adjusting t-test to only compare two groups at a time
group_1_2 <- filter(train_data, Usefulness_score %in% c(1, 2))
group_4_5 <- filter(train_data, Usefulness_score %in% c(4, 5))
combined_groups <- bind_rows(mutate(group_1_2, group = "1-2"), mutate(group_4_5, group
= "4-5"))

t_test_result <- t.test(length ~ group, data = combined_groups)
print(t_test_result)

# Feature Engineering for Length of Dialogue and Average Utterance Length
dialogue_length_train <- train_data %>%
  group_by(Dialogue_ID) %>%
  summarise(length = n())

utterance_length_train <- train_data %>%
  mutate(word_count = str_count(Utterance_text, '\\w+')) %>%
  group_by(Dialogue_ID) %>%
  summarise(avg_utterance_length = mean(word_count))

dialogue_length_validation <- validation_data %>%
  group_by(Dialogue_ID) %>%
  summarise(length = n())

utterance_length_validation <- validation_data %>%
  mutate(word_count = str_count(Utterance_text, '\\w+')) %>%
  group_by(Dialogue_ID) %>%
  summarise(avg_utterance_length = mean(word_count))

# Combine features with usefulness scores
train_features <- dialogue_length_train %>%
  inner_join(utterance_length_train, by = "Dialogue_ID") %>%
  inner_join(usefulness_train, by = "Dialogue_ID")

validation_features <- dialogue_length_validation %>%
  inner_join(utterance_length_validation, by = "Dialogue_ID") %>%
  inner_join(usefulness_validation, by = "Dialogue_ID")

# Handle outliers: Remove outliers beyond 3 standard deviations
train_features <- train_features %>%
  filter(abs(length - mean(length)) / sd(length) < 3) %>%
  filter(abs(avg_utterance_length - mean(avg_utterance_length)) / sd(avg_utterance_length) <
3)

# Rescale data
```

```
scaler <- preProcess(train_features[, c("length", "avg_utterance_length")], method =
c("center", "scale"))
train_features[, c("length", "avg_utterance_length")] <- predict(scaler, train_features[,
c("length", "avg_utterance_length")])


# Prepare data for modeling
train_features <- select(train_features, -Dialogue_ID)
validation_features <- select(validation_features, -Dialogue_ID)

# Function to evaluate model performance
evaluate_model <- function(model, validation_features) {
  preds <- predict(model, newdata = validation_features)
  rmse <- RMSE(preds, validation_features$Usefulness_score)
  r2 <- R2(preds, validation_features$Usefulness_score)
  list(rmse = rmse, r2 = r2)
}

# Polynomial regression model
poly_model <- lm(Usefulness_score ~ poly(length, 2, raw = TRUE) +
poly(avg_utterance_length, 2, raw = TRUE), data = train_features)
poly_results <- tryCatch({
  evaluate_model(poly_model, validation_features)
}, error = function(e) {
  cat("Error in polynomial regression model: ", e$message, "\n")
  list(rmse = Inf, r2 = -Inf)
})

# Print evaluation metrics
cat("Polynomial Regression Model - RMSE:", poly_results$rmse, "R2:", poly_results$r2,
"\n")

# Feature Engineering for the test set
num_utterances_test <- utterances_test %>%
  group_by(Dialogue_ID) %>%
  summarise(num_utterances = n())

avg_utterance_length_test <- utterances_test %>%
  mutate(word_count = str_count(Utterance_text, '\\w+')) %>%
  group_by(Dialogue_ID) %>%
  summarise(avg_utterance_length = mean(word_count))

# Combine features
```

```
test_features <- inner_join(num_utterances_test, avg_utterance_length_test, by =
"Dialogue_ID")

# Rescale the test features using the scaler from the training set
selected_features <- c("num_utterances", "avg_utterance_length")
test_features[, selected_features] <- predict(scaler, test_features[, selected_features])

# Predict the usefulness score for the test set using the polynomial regression model
predicted_usefulness_test <- predict(poly_model, newdata = test_features)

# Populate the predicted usefulness scores into the usefulness_test dataframe
usefulness_test <- utterances_test %>%
  select(Dialogue_ID) %>%
  distinct() %>%
  mutate(Usefulness_score = predicted_usefulness_test)

# Left join the predicted usefulness scores with the test set
usefulness_test <- left_join(utterances_test, usefulness_test, by = "Dialogue_ID")

# Save the predictions to a new CSV file
write_csv(usefulness_test, "subbulakshmi_34069178_dialogue_usefulness_test.csv")
```


Boxplot of Dialogue Length by Usefulness Score
Length of Dialogue
Usefulness Score
Usefulness Score