Task C: Exploratory Data Analysis Using R

Question 1.

Identify the top 3 suburbs with the highest number of property transactions over the years, and plot their monthly transaction counts for the year 2022. Include Toorak in the plot as well, if it is not among the top 3 suburbs.

```
# --- Question 1: Top 3 & Top 10 Suburbs with highest transactions & monthly plot for 2022
---
cat("--- Question 1: Top 3 & Top 10 Suburbs with Highest Transactions & Monthly Plot for 2
022 ---\n")
```

```
## --- Question 1: Top 3 & Top 10 Suburbs with Highest Transactions & Monthly Plot for 202
2 ---
```

```
# Identify the top suburbs with the highest number of property transactions
all_suburb_counts <- df %>%
    filter(!is.na(sold_date)) %>%
    count(suburb, sort = TRUE)

# Top 3 suburbs
top_3_suburbs <- all_suburb_counts %>%
    top_n(3, n) %>%
    pull(suburb)

cat("Top 3 Suburbs with Highest Transactions:\n")
```

```
## Top 3 Suburbs with Highest Transactions:
```

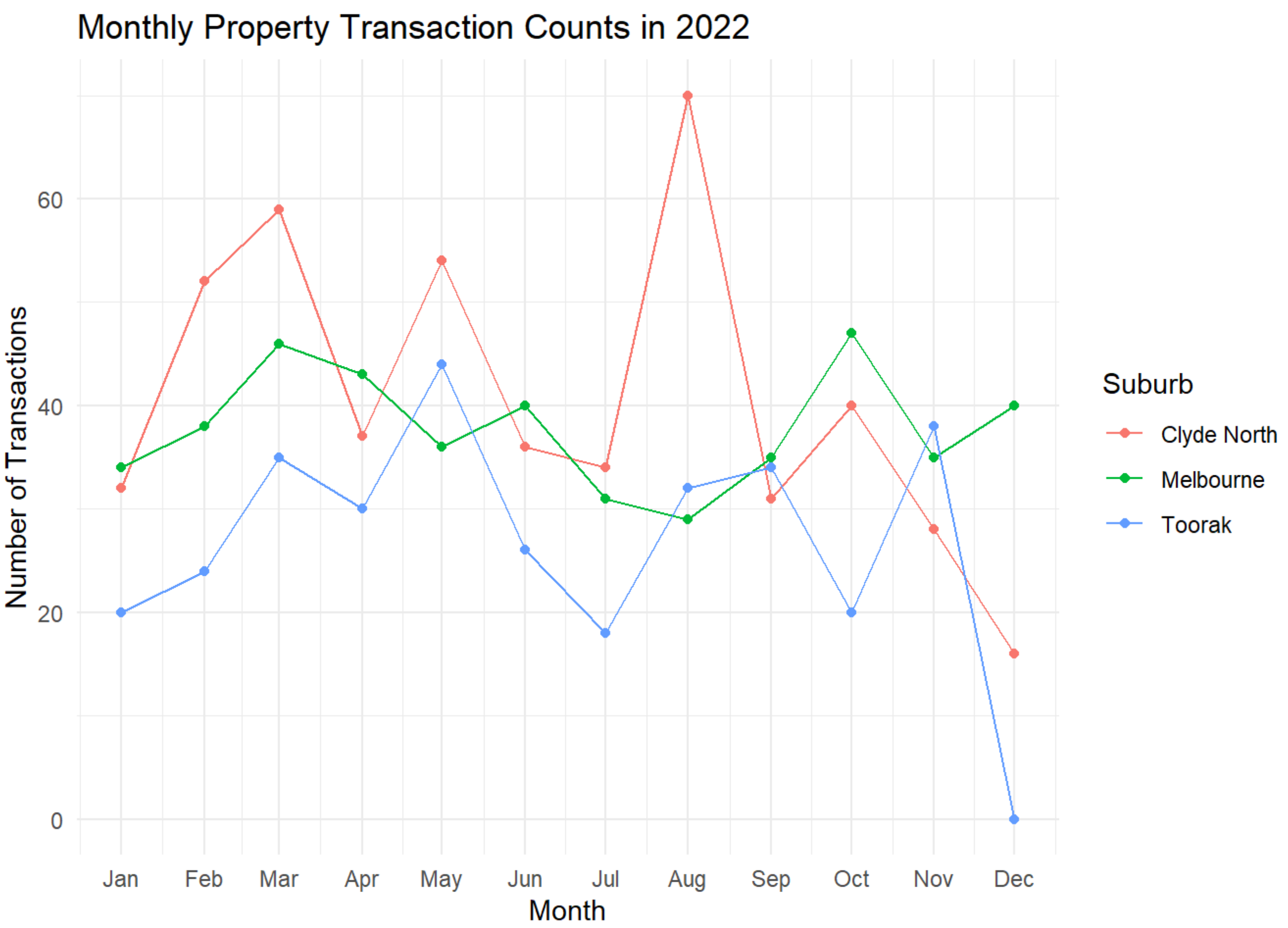
```
print(all_suburb_counts %>% top_n(3, n))
```

```
##        suburb    n
## 1      Toorak 3818
## 2   Melbourne 3530
## 3 Clyde North 2087
```

```
# Add 'Toorak' to the list for plotting if not already in top 3
suburbs_for_plot <- top_3_suburbs
if (!"Toorak" %in% suburbs_for_plot) {
    suburbs_for_plot <- c(suburbs_for_plot, "Toorak")
}

cat("Suburbs included in the 2022 monthly transaction plot:", paste(suburbs_for_plot, coll
apse = ", "), "\n")
```

```
## Suburbs included in the 2022 monthly transaction plot: Toorak, Melbourne, Clyde North
```

```
# Filter data for 2022 and selected suburbs
monthly_transactions_2022 <- df %>%
    filter(year(sold_date) == 2022, suburb %in% suburbs_for_plot) %>%
    mutate(month = floor_date(sold_date, "month")) %>%
    group_by(suburb, month) %>%
    summarise(transaction_count = n(), .groups = "drop") %>%
    # Fill in missing months with 0 transactions for complete lines in plot
    complete(suburb, month = seq.Date(min(.$month), max(.$month), by = "month"), fill = li
st(transaction_count = 0))


# Plot monthly transaction counts for 2022
q1_plot <- ggplot(monthly_transactions_2022, aes(x = month, y = transaction_count, color =
suburb)) +
    geom_line() +
    geom_point() +
    labs(
        title = "Monthly Property Transaction Counts in 2022",
        x = "Month",
        y = "Number of Transactions",
        color = "Suburb"
    ) +
    theme_minimal() +
    scale_x_date(date_breaks = "1 month", date_labels = "%b")

print(q1_plot)
```



Question 2 What are the 3 most important keywords in the description column that impact property prices? (Note: Since the description column contains a large volume of text, please extract a 10% sample from the original dataset to answer this.)

```
# --- Question 2: 3 Most Important Keywords in Description that Impact Property Prices ---
cat("--- Question 2: 3 Most Important Keywords in Description that Impact Property Prices
---\n")
```

```
## --- Question 2: 3 Most Important Keywords in Description that Impact Property Prices
---
```

```r
# Take a 10% sample from the original dataset
set.seed(123) # for reproducibility
sample_df_q2 <- df %>%
    filter(!is.na(description), !is.na(price)) %>%
    sample_frac(0.10) %>%
    # Ensure 'id' is unique for each row in the sample for dtm creation
    mutate(doc_id = row_number()) # Create a unique document ID for tidytext

# Text cleaning and tokenization
custom_stop_words <- tibble(word = c(
    "property", "house", "home", "apartment", "unit", "townhouse", "bedroom",
    "bathroom", "car", "space", "featuring", "boasting", "located", "close",
    "walk", "minutes", "access", "sqm", "m2", "m", "floor", "plan", "land",
    "size", "inspect", "open", "sale", "auction", "sold", "available", "date",
    "lister", "company", "listing", "image", "url", "agent", "phone", "enquire",
    "call", "email", "contact", "new", "private", "offers", "offer", "enquiries",
    "inspection", "view", "master", "en-suite", "ensuite", "kitchen", "living",
    "dining", "area", "balcony", "garage", "garden", "deck", "room", "family",
    "level", "street", "road", "park", "city", "cbd", "tram", "train", "bus",
    "east", "west", "north", "south", "central", "main", "top", "great", "good",
    "modern", "contemporary", "stylish", "spacious", "bright", "light", "perfect",
    "beautiful", "generous", "expansive", "exclusive", "stunning", "desirable",
    "prime", "premier", "luxury", "executive", "premium", "superb", "fabulous",
    "ample", "vast", "well", "designed", "built", "fitted", "appointed", "highly",
    "sought", "after", "ideal", "nestled", "set", "within", "enjoy", "featuring",
    "boasting", "providing", "offering", "delivering", "sure", "impress", "must",
    "see", "don't", "miss", "opportunity", "investment", "development", "build",
    "potential", "growth", "value", "capital", "gains", "secure", "solid", "returns",
    "first", "time", "buyer", "downsizer", "investor", "owner", "occupier", "vacant",
    "possession", "ready", "move", "immediate", "short", "settlement", "flexible",
    "terms", "conditions", "price", "guide", "contact", "agent", "today", "tomorrow",
    "yesterday", "next", "week", "month", "year", "make", "offer", "submit", "best",
    "offers", "expressions", "interest", "registration", "required", "online", "bidding",
    "phone", "bidders", "welcome", "circa", "plus", "minus", "approx", "approximate",
    "estimated", "from", "to", "over", "under", "up", "down", "around", "near",
    "beyond", "via", "through", "with", "and", "or", "but", "if", "then", "else", "is",
    "are", "was", "were", "be", "been", "being", "have", "has", "had", "do", "does",
    "did", "will", "would", "shall", "should", "can", "could", "may", "might", "must",
    "a", "an", "the", "this", "that", "these", "those", "my", "your", "his", "her",
    "its", "our", "their", "so", "as", "at", "by", "for", "from", "into", "of", "on",
    "to", "up", "down", "out", "off", "over", "under", "again", "further", "then",
    "once", "here", "there", "when", "where", "why", "how", "all", "any", "both",
    "each", "few", "more", "most", "other", "some", "such", "no", "nor", "not",
    "only", "own", "same", "so", "than", "too", "very", "s", # remove common possessive 's
    "d", "ll", "m", "t", "ve", "y" # common contractions remnants
))

cleaned_description_words <- sample_df_q2 %>%
    unnest_tokens(word, description) %>%
    anti_join(stop_words, by = "word") %>% # Remove common English stop words
    anti_join(custom_stop_words, by = "word") %>% # Remove custom stop words
    filter(str_detect(word, "^[a-z]+$")) # Keep only alphabetic words, no numbers/punctuat
ion remnants
```

```r
# Count word frequencies for each description
# Using term frequency (n) directly for DTM
word_counts_per_doc <- cleaned_description_words %>%
    count(doc_id, word, sort = TRUE)

# Cast to Document-Term Matrix
# Using control = list(weighting = tm::weightTf) to get term frequency
dtm <- word_counts_per_doc %>%
    cast_dtm(doc_id, word, n)

# Convert DTM to a data frame for regression
# Use as.matrix first then as.data.frame to ensure proper column names
dtm_df <- as.data.frame(as.matrix(dtm))

# Make column names valid and unique for R formulas
valid_colnames <- make.names(colnames(dtm_df), unique = TRUE)
colnames(dtm_df) <- valid_colnames

# Add 'doc_id' as a regular column from rownames
dtm_df$doc_id <- as.numeric(rownames(dtm_df))

# Merge with price data using the new 'doc_id'
regression_data_q2 <- sample_df_q2 %>%
    select(doc_id, price) %>%
    inner_join(dtm_df, by = "doc_id") %>%
    select(-doc_id) # Remove doc_id as it's not a predictor

# Build a linear regression model
# Identify top N most frequent words from the cleaned sample to reduce dimensionality
top_words_overall_q2 <- cleaned_description_words %>%
    count(word, sort = TRUE) %>%
    top_n(100, n) %>% # Increased to 100 for more potential features
    pull(word)

# Filter regression_data_q2 to include only these top words, plus 'price'
# Ensure column names match the DTM's valid names
top_words_valid_names <- make.names(top_words_overall_q2, unique = TRUE)
regression_data_filtered_q2 <- regression_data_q2 %>%
    select(price, intersect(top_words_valid_names, colnames(regression_data_q2)))

# Remove columns with zero variance (if any, after filtering) to avoid errors in lm()
regression_data_filtered_q2 <- regression_data_filtered_q2[, sapply(regression_data_filter
ed_q2, function(x) length(unique(x)) > 1)]

# Check if there are any predictor columns left after filtering
if (ncol(regression_data_filtered_q2) > 1) {
    # Create the formula dynamically
    formula_str_q2 <- paste("price ~", paste(colnames(regression_data_filtered_q2)[-1], co
llapse = " + "))
    model_q2 <- lm(as.formula(formula_str_q2), data = regression_data_filtered_q2)

    # Extract coefficients and identify important keywords
    coefficients_q2 <- as.data.frame(summary(model_q2)$coefficients)
    coefficients_q2$word <- rownames(coefficients_q2)
    colnames(coefficients_q2)[c(1, 4)] <- c("Estimate", "p_value")
```

```
    # Rank by absolute estimate, filter for significant p-values (e.g., < 0.05)
    # Show top 3 keywords with positive impact (positive estimate)
    important_keywords_q2 <- coefficients_q2 %>%
        filter(word != "(Intercept)", p_value < 0.05, Estimate > 0) %>% # Focus on positiv
e impact
        arrange(desc(abs(Estimate))) %>%
        head(3)

    if (nrow(important_keywords_q2) > 0) {
        cat("Top 3 Keywords positively impacting property prices (based on simplified regr
ession):\n")
        print(important_keywords_q2 %>% select(word, Estimate, p_value))
    } else {
        cat("No significant keywords with positive impact found with the current model and
data subset.\n")
    }
} else {
    cat("Not enough unique word features in the sample to build a regression model.\n")
}
```

```
## Top 3 Keywords positively impacting property prices (based on simplified regression):
##            word    Estimate      p_value
## steel     steel 43381968925 3.444202e-06
## natural natural 24362215057 5.460854e-10
## views     views  9746451832 3.350000e-04
```

Question 3

Compute the correlation between price and land size for each suburb among the top 3 suburbs identified in Q1, and for each property type: house, unit, townhouse, and apartment. Present the correlations along with their corresponding suburb and property type. (Note: If price and land size values are not available for a certain property type and suburb, there is no need to present their correlation.)

```
# --- Question 3: Correlation between Price and Land Size per Suburb & Property Type ---
cat("--- Question 3: Correlation between Price and Land Size per Suburb & Property Type
---\n")
```

```
## --- Question 3: Correlation between Price and Land Size per Suburb & Property Type ---
```

```
# Define target property types for this question (House, Unit, Townhouse, Apartment)
# Filter based on these specific types even if more are available in the dataset
q3_target_property_types <- c("house", "unit", "townhouse", "apartment")

cat("Calculating correlations for these property types:", paste(q3_target_property_types,
collapse = ", "), "\n")
```

```
## Calculating correlations for these property types: house, unit, townhouse, apartment
```

```
# Compute correlations for the top 3 suburbs (from Q1) and specified property types
correlations_q3 <- df %>%
    filter(
        suburb %in% top_3_suburbs,
        property_type %in% q3_target_property_types,
        !is.na(price), !is.na(land_size) # Ensure non-NA values for correlation
    ) %>%
    group_by(suburb, property_type) %>%
    summarise(
        correlation = cor(price, land_size, use = "pairwise.complete.obs"),
        .groups = "drop"
    ) %>%
    filter(!is.na(correlation)) # Remove combinations where correlation could not be compu
ted (e.g., all NA or constant values)

if (nrow(correlations_q3) > 0) {
    cat("Correlation between Price and Land Size for Top 3 Suburbs:\n")
    print(correlations_q3)
} else {
    cat("No correlations found for the specified suburbs and property types due to insuffi
cient data.\n")
}
```

```
## Correlation between Price and Land Size for Top 3 Suburbs:
## # A tibble: 8 × 3
##   suburb      property_type correlation
##   <chr>       <chr>               <dbl>
## 1 Clyde North house            0.000902
## 2 Clyde North townhouse        0.362
## 3 Clyde North unit             0.993
## 4 Melbourne   apartment       -0.00119
## 5 Melbourne   unit            -0.0409
## 6 Toorak      apartment       -0.00175
## 7 Toorak      house            0.301
## 8 Toorak      townhouse        0.0442
```

Question 4

Owning property has long been considered a reliable way to build personal wealth. Which properties have experienced the highest price increases since their first sale? Please exclude properties where the time between the first and last sale exceeds five years. List the top five properties along with their address, capital gain, and the duration between the first and last sale.

```
# --- Question 4: Properties with Highest Price Increases ---
cat("--- Question 4: Properties with Highest Price Increases ---\n")
```

```
## --- Question 4: Properties with Highest Price Increases ---
```

```r
# Group by a unique property identifier (assuming full_address is unique per property)
# Filter for properties that have at least two distinct sales dates to calculate gain
property_gains <- df %>%
    filter(!is.na(full_address), !is.na(sold_date), !is.na(price)) %>%
    group_by(full_address) %>%
    summarise(
        n_sales = n_distinct(sold_date), # Count distinct sale dates for the property
        first_sale_date = min(sold_date),
        last_sale_date = max(sold_date),
        first_sale_price = price[which.min(sold_date)], # Price corresponding to the earli
est date
        last_sale_price = price[which.max(sold_date)], # Price corresponding to the latest
date
        .groups = "drop"
    ) %>%
    # Filter for properties with at least two sales, and where first/last dates are truly
different
    # Also, check if prices are different to ensure actual 'gain'
    filter(n_sales >= 2, first_sale_date < last_sale_date, first_sale_price != last_sale_p
rice) %>%
    mutate(
        capital_gain = last_sale_price - first_sale_price,
        duration_days = as.numeric(last_sale_date - first_sale_date)
    ) %>%
    filter(duration_days <= (5 * 365)) %>% # Filter out properties where duration exceeds
five years
    arrange(desc(capital_gain)) %>%
    head(5)

if (nrow(property_gains) > 0) {
    cat("Top 5 Properties with Highest Capital Gains (within 5 years):\n")
    print(property_gains %>% select(full_address, capital_gain, duration_days))
} else {
    cat("No properties found with multiple sales records meeting the criteria (distinct da
tes, distinct prices, duration <= 5 years).\n")
}
```

```
## Top 5 Properties with Highest Capital Gains (within 5 years):
## # A tibble: 5 × 3
##   full_address                             capital_gain duration_days
##   <chr>                                           <dbl>         <dbl>
## 1 4 Anchor Place, Prahran, Vic 3181             1.67e13          1753
## 2 11 Charles Street, Selby, Vic 3159            7.55e11          1310
## 3 8 Harlington Street, Clayton, Vic 3168        6.00e11           655
## 4 69 Creek Road, Mitcham, Vic 3132              5.25e11            18
## 5 14 Banker Street, Kurunjang, Vic 3337         4.30e11          1278
```

Question 5

Property price trends can vary not only across suburbs but also across property types. Identify which suburb–property type combination exhibited the most volatility in median property prices over the months of 2022. Display the top 5 most volatile combinations and provide an appropriate plot. (Note: Consider only the following property types — house, unit, townhouse, and apartment.)

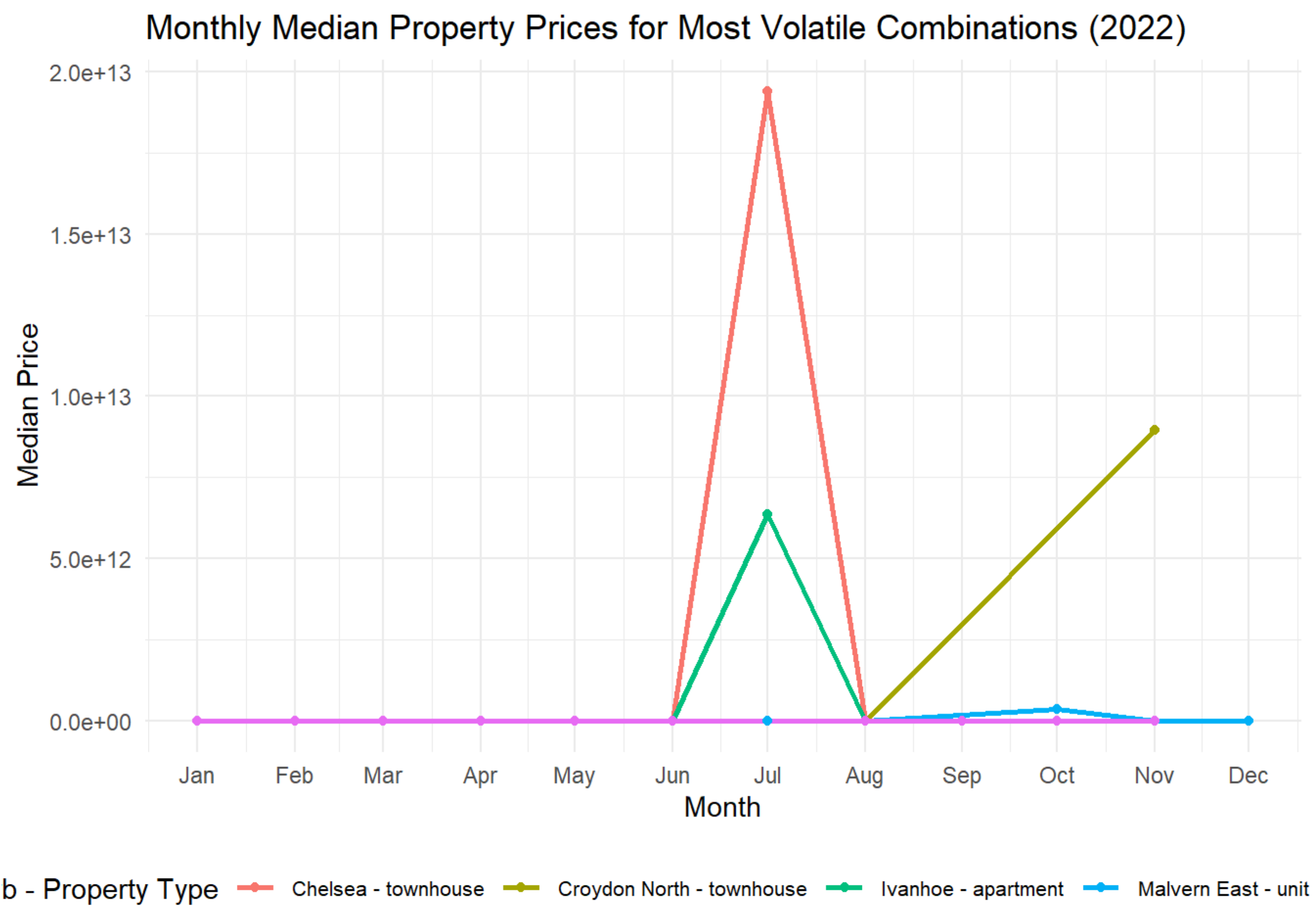
```
# --- Question 5: Most Volatile Suburb-Property Type Combination in 2022 ---
cat("--- Question 5: Most Volatile Suburb-Property Type Combination in 2022 ---\n")
```

```
## --- Question 5: Most Volatile Suburb-Property Type Combination in 2022 ---
```

```
# Define target property types for this question (House, Unit, Townhouse, Apartment)
q5_target_property_types <- c("house", "unit", "townhouse", "apartment")

# Calculate monthly median prices for 2022
monthly_median_prices_2022_q5 <- df %>%
    filter(
        year(sold_date) == 2022,
        property_type %in% q5_target_property_types,
        !is.na(price)
    ) %>%
    mutate(month = floor_date(sold_date, "month")) %>%
    group_by(suburb, property_type, month) %>%
    summarise(
        median_price = median(price, na.rm = TRUE),
        .groups = "drop"
    ) %>%
    filter(!is.na(median_price)) # Remove combinations where median price is NA (e.g., no
sales in a month)

# Calculate volatility (standard deviation of monthly median prices)
volatility_combinations_q5 <- monthly_median_prices_2022_q5 %>%
    group_by(suburb, property_type) %>%
    # Only calculate volatility if there's more than one monthly median price point
    filter(n() > 1) %>%
    summarise(
        volatility = sd(median_price, na.rm = TRUE),
        .groups = "drop"
    ) %>%
    filter(!is.na(volatility)) %>%
    arrange(desc(volatility)) %>%
    head(5)

if (nrow(volatility_combinations_q5) > 0) {
    cat("Top 5 Most Volatile Suburb-Property Type Combinations (2022):\n")
    print(volatility_combinations_q5)

    # Prepare data for plotting the top volatile combinations
    plot_data_volatility_q5 <- monthly_median_prices_2022_q5 %>%
        inner_join(volatility_combinations_q5, by = c("suburb", "property_type")) %>%
        mutate(combination = paste(suburb, property_type, sep = " - "))

    # Plot median prices over months for the most volatile combinations
    q5_plot <- ggplot(plot_data_volatility_q5, aes(x = month, y = median_price, color = co
mbination)) +
        geom_line(linewidth = 1) +
        geom_point(linewidth = 2) +
        labs(
            title = "Monthly Median Property Prices for Most Volatile Combinations (202
2)",
            x = "Month",
            y = "Median Price",
            color = "Suburb - Property Type"
        ) +
        theme_minimal() +
```

```
        scale_x_date(date_breaks = "1 month", date_labels = "%b") +
        theme(legend.position = "bottom", legend.text = element_text(size = 8))

    print(q5_plot)
} else {
    cat("No volatile combinations found for the specified criteria in 2022, or not enough 
data points per combination.\n")
}
```

```
## Top 5 Most Volatile Suburb-Property Type Combinations (2022):
## # A tibble: 5 × 3
##   suburb        property_type volatility
##   <chr>         <chr>              <dbl>
## 1 Chelsea       townhouse        5.85e12
## 2 Croydon North townhouse        4.00e12
## 3 Ivanhoe       apartment        1.92e12
## 4 Malvern East  unit             1.22e11
## 5 Toorak        house            1.46e 7
```



Question 6

Chris is looking for a renovated house to purchase. He wants the property to be close to a shopping centre, and since he has a 7-year-old son, proximity to a primary school is also important. He is looking for a home with 4 bedrooms and 2 bathrooms. Currently, he is considering six suburbs—Mulgrave, Vermont South, Doncaster East, Rowville, Glen Waverley, and Wheelers Hill—and plans to choose one from this list. Please provide the predicted price for September 2025 of a house that meets the above criteria in each of the six suburbs, and display the predicted price alongside the corresponding suburb name. (Note: When you build the prediction model, please use only the provided dataset and the period it covers.)

```
# --- Question 6: Predicted Price for September 2025 ---
cat("--- Question 6: Predicted Price for September 2025 ---\n")
```

```
## --- Question 6: Predicted Price for September 2025 ---
```

```r
# Define the target suburbs
chris_suburbs <- c("Mulgrave", "Vermont South", "Doncaster East", "Rowville", "Glen Waverley", "Wheelers Hill")

# Filter data for model training
model_data_q6 <- df %>%
    filter(
        property_type == "house",
        bedrooms == 4,
        bathrooms == 2,
        suburb %in% chris_suburbs,
        !is.na(price),
        !is.na(sold_date),
        !is.na(parking_spaces),
        !is.na(building_size),
        !is.na(land_size)
    ) %>%
    mutate(
        # Create a numerical time variable (days since the earliest date in the dataset)
        time_index = as.numeric(sold_date - min(.$sold_date, na.rm = TRUE)),
        # Convert suburb to a factor with levels based on the training data
        suburb = factor(suburb)
    )

# Check if there is enough data for modeling
if (nrow(model_data_q6) < 10) { # A heuristic for minimum data points, adjust as needed
    cat("Not enough data to build a reliable prediction model for the specified criteria. (Need at least 10 data points)\n")
} else {
    # Build a linear regression model
    # The formula `price ~ time_index + suburb + parking_spaces + building_size + land_size`
    # automatically handles 'suburb' as a factor due to the conversion above.
    tryCatch(
        {
            model_q6 <- lm(price ~ time_index + suburb + parking_spaces + building_size + land_size, data = model_data_q6)

            # Calculate median values for numerical features from the filtered training data
            median_parking_spaces_q6 <- median(model_data_q6$parking_spaces, na.rm = TRUE)
            median_building_size_q6 <- median(model_data_q6$building_size, na.rm = TRUE)
            median_land_size_q6 <- median(model_data_q6$land_size, na.rm = TRUE)

            # Prepare new data for prediction for September 2025
            prediction_date_q6 <- as.Date("2025-09-01")
            # Use the minimum sold_date from the training data for consistent time_index calculation
            min_sold_date_training_q6 <- min(model_data_q6$sold_date, na.rm = TRUE)
            prediction_time_index_q6 <- as.numeric(prediction_date_q6 - min_sold_date_training_q6)

            # Create a data frame for prediction
            predict_df_q6 <- data.frame(
```

```
            time_index = rep(prediction_time_index_q6, length(chris_suburbs)),
            # Ensure factor levels for suburb in prediction data match training data
            suburb = factor(chris_suburbs, levels = levels(model_data_q6$suburb)),
            parking_spaces = rep(median_parking_spaces_q6, length(chris_suburbs)),
            building_size = rep(median_building_size_q6, length(chris_suburbs)),
            land_size = rep(median_land_size_q6, length(chris_suburbs))
        )

        # Handle cases where a suburb in chris_suburbs was not in the training data
        # These will result in NA predictions, which we should filter or warn about.
        # Check if any suburb in predict_df_q6 is not present in model_data_q6 levels
        missing_suburbs_in_model <- chris_suburbs[!chris_suburbs %in% levels(model_dat
a_q6$suburb)]
        if (length(missing_suburbs_in_model) > 0) {
            cat(paste0("Warning: No training data for suburbs: ", paste(missing_suburb
s_in_model, collapse = ", "), ". Predictions for these suburbs will be NA or unreliable.
\n"))
        }

        # Predict prices
        predicted_prices_q6 <- predict(model_q6, newdata = predict_df_q6)
        predict_df_q6$predicted_price <- round(predicted_prices_q6, 2)

        cat("Predicted Prices for September 2025 (House, 4 Beds, 2 Baths, Median Parki
ng/Building/Land Size):\n")
        print(predict_df_q6 %>% select(suburb, predicted_price))
    },
    error = function(e) {
        cat("An error occurred during model building or prediction for Q6:\n")
        cat(e$message, "\n")
        cat("This might be due to insufficient variance in predictor variables or othe
r model fitting issues.\n")
    }
  )
}
```

```
## Predicted Prices for September 2025 (House, 4 Beds, 2 Baths, Median Parking/Building/La
nd Size):
##           suburb predicted_price
## 1       Mulgrave         1454205
## 2  Vermont South         1615660
## 3 Doncaster East         1956279
## 4       Rowville         1351177
## 5  Glen Waverley         1776813
## 6  Wheelers Hill         1574575
```

Task D: Predictive Data Analysis using R

This R Markdown document details the process of training a machine learning model to predict dialogue usefulness based on various engineered features.

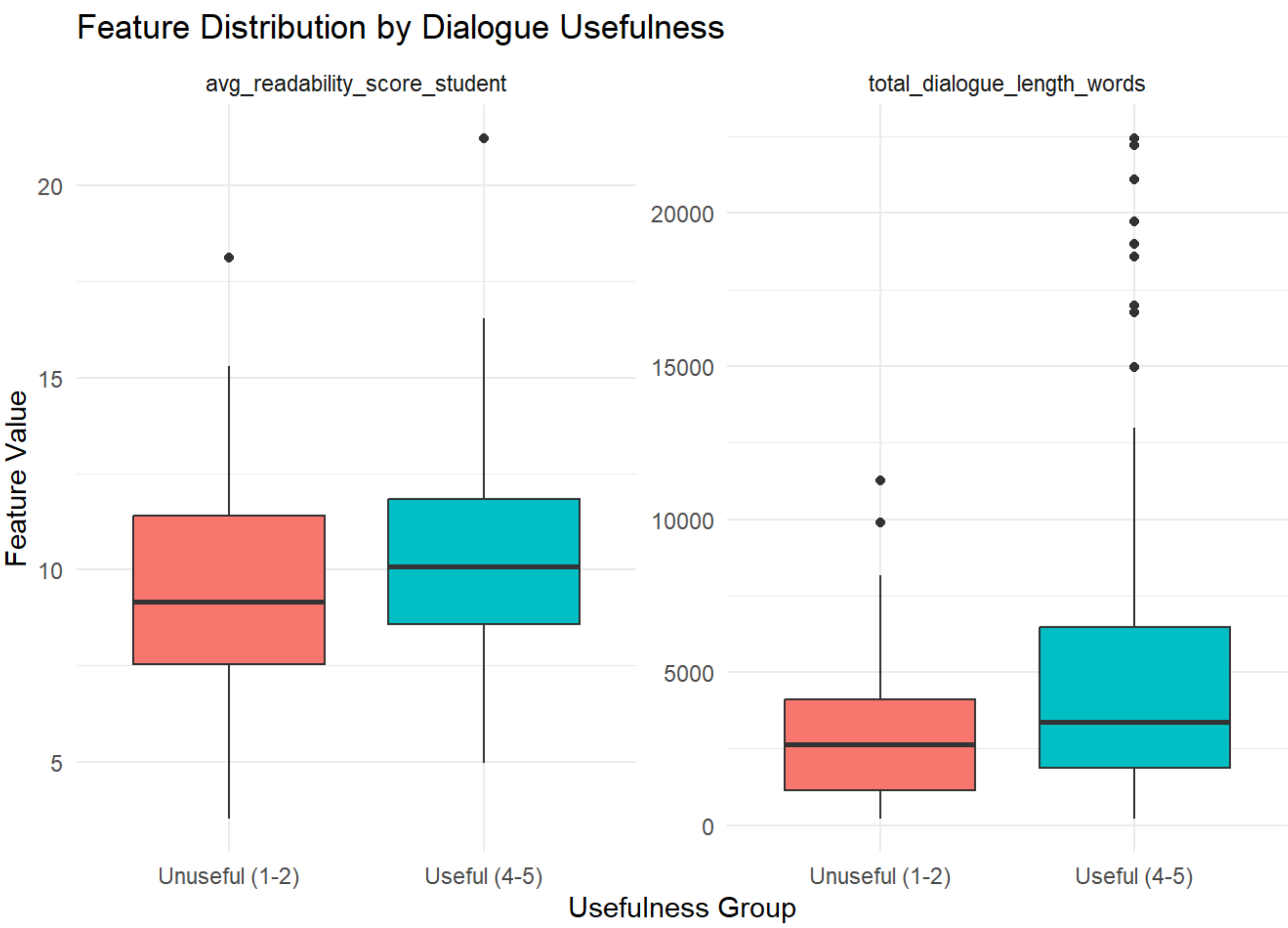
Step 1: Feature Engineering and Initial Visualization

This step involves loading the raw dialogue and usefulness data, engineering new features from the textual and temporal information, and then visualizing the distribution of two selected features across different usefulness score groups to assess potential differences.

```r
# Set working directory to the location of the Rmd file
library(this.path)
setwd(this.path::here())

# --- Load All Necessary Packages ---
# Check if packages are installed, if not, install them quietly
if (!requireNamespace("dplyr", quietly = TRUE)) install.packages("dplyr")
if (!requireNamespace("ggplot2", quietly = TRUE)) install.packages("ggplot2")
if (!requireNamespace("lubridate", quietly = TRUE)) install.packages("lubridate")
if (!requireNamespace("stringr", quietly = TRUE)) install.packages("stringr")
if (!requireNamespace("quanteda", quietly = TRUE)) install.packages("quanteda")
if (!requireNamespace("quanteda.textstats", quietly = TRUE)) install.packages("quanteda.te
xtstats")
if (!requireNamespace("tidyr", quietly = TRUE)) install.packages("tidyr") # For gather fun
ction

# Load required libraries
library(dplyr)
library(ggplot2)
library(lubridate)
library(stringr)
library(quanteda)
library(quanteda.textstats)
library(tidyr) # For gather

# --- Load Training Data ---
df_utterance_train <- read.csv("git_ignore/dialogue_utterance_train.csv")
df_usefulness_train <- read.csv("git_ignore/dialogue_usefulness_train.csv")

# Merge dataframes and sort by Dialogue_ID and Timestamp
df_merged_train <- left_join(df_utterance_train, df_usefulness_train, by = "Dialogue_ID")
%>%
    mutate(Timestamp = ymd_hms(Timestamp)) %>% # Convert Timestamp to datetime objects
    arrange(Dialogue_ID, Timestamp) # Sort for sequential processing

# --- Feature Engineering Functions ---

# Function to calculate readability scores per utterance using Flesch-Kincaid
calculate_readability <- function(df) {
    # Assign unique utterance IDs for quanteda processing
    df$utterance_id <- paste0("utt_", 1:nrow(df))
    # Create a quanteda corpus from the utterance text
    utterance_corpus <- corpus(df, text_field = "Utterance_text", docid_field = "utterance
_id")

    # Calculate readability scores if the corpus is not empty
    if (ndoc(utterance_corpus) > 0) {
        readability_scores <- textstat_readability(utterance_corpus, measure = "Flesch.Kin
caid") %>%
            select(document, Flesch.Kincaid) %>%
            rename(utterance_id = document, readability_score = Flesch.Kincaid)
        # Join readability scores back to the original dataframe
        df <- left_join(df, readability_scores, by = "utterance_id")
        # Replace NA readability scores with 0 (e.g., for empty utterances)
```

```
        df$readability_score[is.na(df$readability_score)] <- 0
    }
    return(df)
}

# Function to engineer all 17 dialogue-level features
engineer_features <- function(df_with_readability) {
    dialogue_features_df <- df_with_readability %>%
        group_by(Dialogue_ID) %>%
        summarise(
            num_utterances = n(), # Total number of utterances in a dialogue
            total_dialogue_length_words = sum(sapply(str_split(Utterance_text, "\\s+"), le
ngth), na.rm = TRUE), # Total words in dialogue
            dialogue_duration = as.numeric(difftime(max(Timestamp), min(Timestamp), units
= "secs")), # Duration of dialogue in seconds
            avg_len_student_utterance_words = mean(sapply(str_split(Utterance_text[Interlo
cutor == "Student"], "\\s+"), length), na.rm = TRUE), # Avg words in student utterances
            avg_len_chatbot_utterance_words = mean(sapply(str_split(Utterance_text[Interlo
cutor == "Chatbot"], "\\s+"), length), na.rm = TRUE), # Avg words in chatbot utterances
            num_student_questions = sum(str_detect(Utterance_text[Interlocutor == "Studen
t"], "\\?"), na.rm = TRUE), # Number of questions by student
            num_chatbot_questions = sum(str_detect(Utterance_text[Interlocutor == "Chatbo
t"], "\\?"), na.rm = TRUE), # Number of questions by chatbot
            # Store all words for TTR calculation
            all_student_words = list(unlist(str_split(paste(Utterance_text[Interlocutor ==
"Student"], collapse = " "), "\\s+"))),
            all_chatbot_words = list(unlist(str_split(paste(Utterance_text[Interlocutor ==
"Chatbot"], collapse = " "), "\\s+"))),
            avg_readability_score_student = mean(readability_score[Interlocutor == "Studen
t"], na.rm = TRUE), # Avg readability of student utterances
            avg_readability_score_chatbot = mean(readability_score[Interlocutor == "Chatbo
t"], na.rm = TRUE), # Avg readability of chatbot utterances
            time_diffs_raw = list(as.numeric(diff(Timestamp), units = "secs")) # Raw time
differences between utterances
        ) %>%
        mutate(
            # Calculate Type-Token Ratio (TTR) for student and chatbot
            num_unique_words_student = sapply(all_student_words, function(x) length(unique
(x[x != "" & !is.na(x)]))),
            num_unique_words_chatbot = sapply(all_chatbot_words, function(x) length(unique
(x[x != "" & !is.na(x)]))),
            total_words_student = sapply(all_student_words, function(x) length(x[x != "" &
!is.na(x)])),
            total_words_chatbot = sapply(all_chatbot_words, function(x) length(x[x != "" &
!is.na(x)])),
            ttr_student = ifelse(total_words_student > 0, num_unique_words_student / total
_words_student, 0),
            ttr_chatbot = ifelse(total_words_chatbot > 0, num_unique_words_chatbot / total
_words_chatbot, 0),
            # Calculate variance of time between utterances
            variance_time_between_utterances = sapply(time_diffs_raw, function(x) ifelse(l
ength(x) > 1, var(x, na.rm = TRUE), 0)),
            # Ratio of average utterance length
            ratio_student_chatbot_len_words = ifelse(avg_len_chatbot_utterance_words > 0,
avg_len_student_utterance_words / avg_len_chatbot_utterance_words, Inf)
```

```
        ) %>%
        # Remove intermediate list columns
        select(-all_student_words, -all_chatbot_words, -time_diffs_raw)

    # Add the Usefulness_score back to the features dataframe
    df_usefulness_scores_unique <- df_with_readability %>%
        select(Dialogue_ID, Usefulness_score) %>%
        distinct() # Ensure unique Dialogue_ID and Usefulness_score pairs
    dialogue_features_df <- left_join(dialogue_features_df, df_usefulness_scores_unique, b
y = "Dialogue_ID")
    return(dialogue_features_df)
}

# --- Execute Feature Engineering on Training Data ---
df_merged_train_readable <- calculate_readability(df_merged_train)
dialogue_features_train_raw <- engineer_features(df_merged_train_readable)

# --- Visualization & Statistical Tests ---
# Select two features for visualization: total dialogue length and average student readabi
lity
selected_features_vis <- c("total_dialogue_length_words", "avg_readability_score_student")

# Prepare data for boxplots: filter for extreme usefulness scores (1,2 and 4,5)
plot_data <- dialogue_features_train_raw %>%
    filter(Usefulness_score %in% c(1, 2, 4, 5)) %>%
    mutate(Score_Group = ifelse(Usefulness_score %in% c(1, 2), "Unuseful (1-2)", "Useful
(4-5)")) %>%
    select(all_of(selected_features_vis), Score_Group) %>%
    tidyr::gather(key = "Feature", value = "Value", -Score_Group) # Reshape data for ggplo
t

# Generate Boxplot
ggplot(plot_data, aes(x = Score_Group, y = Value, fill = Score_Group)) +
    geom_boxplot() +
    facet_wrap(~Feature, scales = "free_y") + # Create separate plots for each feature wit
h free y-scales
    labs(title = "Feature Distribution by Dialogue Usefulness", x = "Usefulness Group", y
= "Feature Value") +
    theme_minimal() + # Use a minimal theme for aesthetics
    theme(legend.position = "none") # Hide legend as fill explains groups
```




```
# Perform T-tests to check for statistical significance between the two groups
cat("--- Statistical Significance Tests ---\n")
```

```
## --- Statistical Significance Tests ---
```

```
for (feature in selected_features_vis) {
    group1_data <- dialogue_features_train_raw %>%
        filter(Usefulness_score %in% c(1, 2)) %>%
        pull(!!feature) # Extract feature values for unuseful group
    group2_data <- dialogue_features_train_raw %>%
        filter(Usefulness_score %in% c(4, 5)) %>%
        pull(!!feature) # Extract feature values for useful group

    # Perform t-test only if both groups have sufficient data points (more than 1 non-NA v
alue)
    if (length(na.omit(group1_data)) > 1 && length(na.omit(group2_data)) > 1) {
        ttest_result <- t.test(group1_data, group2_data)
        cat(paste("Feature:", feature, "- T-test p-value:", round(ttest_result$p.value,
4), "\n"))
    } else {
        cat(paste("Feature:", feature, "- Insufficient data for t-test.\n"))
    }
}
```

```
## Feature: total_dialogue_length_words - T-test p-value: 7e-04
## Feature: avg_readability_score_student - T-test p-value: 0.1623
```

Based on the boxplots, differences in feature values between “Unuseful (1-2)” and “Useful (4-5)” dialogues can be observed. The t-test p-values indicate the statistical significance of these differences. A p-value less than a chosen significance level (e.g., 0.05) suggests a statistically significant difference between the means of the two groups for that feature. For example, a small p-value for ‘total_dialogue_length_words’ would imply that useful dialogues tend to have a significantly different total word count than unuseful ones.

Step 2: Baseline Machine Learning Model Training and Evaluation

This section focuses on building baseline machine learning models using the full set of engineered features from Step 1 and evaluating their performance on the validation set. Four different regression models are trained: Linear Regression, Regression Tree, Random Forest, and Support Vector Regression (SVR). RMSE (Root Mean Squared Error) is used as the evaluation metric.

```r
# --- Load Packages for Modeling ---
if (!requireNamespace("caret", quietly = TRUE)) install.packages("caret")
if (!requireNamespace("rpart", quietly = TRUE)) install.packages("rpart")
if (!requireNamespace("randomForest", quietly = TRUE)) install.packages("randomForest")
if (!requireNamespace("e1071", quietly = TRUE)) install.packages("e1071")

library(caret)
library(rpart)
library(randomForest)
library(e1071)

# --- Load and Engineer Validation Data ---
df_utterance_validation <- read.csv("git_ignore/dialogue_utterance_validation.csv")
df_usefulness_validation <- read.csv("git_ignore/dialogue_usefulness_validation.csv")

# Merge and process validation data, similar to training data
df_merged_validation <- left_join(df_utterance_validation, df_usefulness_validation, by =
"Dialogue_ID") %>%
    mutate(Timestamp = ymd_hms(Timestamp)) %>%
    arrange(Dialogue_ID, Timestamp)

df_merged_validation_readable <- calculate_readability(df_merged_validation)
dialogue_features_validation_raw <- engineer_features(df_merged_validation_readable)

# --- Baseline Data Preparation: Remove rows with NA/Inf for initial models ---
# This creates a 'clean' subset for the baseline models, assuming a simple approach.
dialogue_features_train_baseline <- na.omit(dialogue_features_train_raw)
dialogue_features_validation_baseline <- na.omit(dialogue_features_validation_raw)

# Define the features to be used for modeling (all engineered features except ID and targe
t)
features_to_use <- setdiff(names(dialogue_features_train_baseline), c("Dialogue_ID", "Usef
ulness_score"))
# Create the formula for the models
formula_all <- as.formula(paste("Usefulness_score ~", paste(features_to_use, collapse = "
+ ")))

# Define RMSE function for evaluation
RMSE <- function(y_true, y_pred) sqrt(mean((y_true - y_pred)^2))

# --- Train All Four Baseline Models ---
cat("--- Training Baseline Models ---\n")
```

```
## --- Training Baseline Models ---
```

```
# Linear Regression Model
lm_model_baseline <- lm(formula_all, data = dialogue_features_train_baseline)

# Regression Tree Model
rt_model_baseline <- rpart(formula_all, data = dialogue_features_train_baseline, method =
"anova")

# Random Forest Model (set seed for reproducibility)
set.seed(123)
rf_model_baseline <- randomForest(formula_all, data = dialogue_features_train_baseline, nt
ree = 500)

# Support Vector Regression (SVR) Model
svr_model_baseline <- svm(formula_all, data = dialogue_features_train_baseline)

# --- Evaluate Baseline Models on Validation Set ---
models_baseline <- list(
    "Linear Regression" = lm_model_baseline,
    "Regression Tree" = rt_model_baseline,
    "Random Forest" = rf_model_baseline,
    "SVR" = svr_model_baseline
)

rmse_baseline <- sapply(models_baseline, function(model) {
    preds <- predict(model, newdata = dialogue_features_validation_baseline)
    RMSE(dialogue_features_validation_baseline$Usefulness_score, preds)
})

performance_df_baseline <- data.frame(Model = names(rmse_baseline), RMSE = rmse_baseline,
Stage = "Baseline")

cat("--- Baseline Model Performance (on NA-omitted data) ---\n")
```

```
## --- Baseline Model Performance (on NA-omitted data) ---
```

```
print(performance_df_baseline %>% arrange(RMSE))
```

```
##                               Model      RMSE    Stage
## SVR                             SVR 0.9883981 Baseline
## Random Forest         Random Forest 0.9917741 Baseline
## Linear Regression Linear Regression 1.1116204 Baseline
## Regression Tree     Regression Tree 1.1935477 Baseline
```

```
# Store the best baseline model for potential future use (though typically we aim for impr
oved models)
model1 <- performance_df_baseline %>%
    arrange(RMSE) %>%
    slice(1) %>%
    pull(Model)
```

The table above presents the RMSE values for each baseline model on the validation set. Model 1 is designated as the best-performing model from this baseline evaluation. The next step will aim to improve upon this performance.

Step 3: Model Improvement Through Advanced Data Processing and Feature Selection

This section explores methods to improve model performance, focusing on robust data processing (handling `NA` / `Inf` values) and feature selection. The chosen approach for `NA` / `Inf` imputation is to use mean imputation from the training set, and for feature selection, features with importance scores above the mean importance (from Random Forest) are retained.

```
# --- Improvement Stage 1: Advanced Data Processing ---
cat("--- Improvement Stage 1: Advanced Data Processing ---\n")
```

```
## --- Improvement Stage 1: Advanced Data Processing ---
```

```
# Start with the full raw engineered data from Step 1 for robust processing
train_processed <- dialogue_features_train_raw
validation_processed <- dialogue_features_validation_raw

# Mandatory: Impute NA and Inf values
# This method uses mean imputation for NA values and replaces Inf with a value
# slightly greater than the max finite value observed in the training set.
cat("Applying mandatory NA/Inf imputation...\n")
```

```
## Applying mandatory NA/Inf imputation...
```

```r
for (col in features_to_use) { # Iterate through all features identified for modeling
    # Handle Infinite values
    is_inf_train <- is.infinite(train_processed[[col]])
    if (any(is_inf_train)) {
        # Find the maximum finite value in the training column
        max_finite_train <- max(train_processed[[col]][!is_inf_train], na.rm = TRUE)
        # Replace Inf values in training set with a value slightly larger than max finite
        train_processed[[col]][is_inf_train] <- max_finite_train + 1
        # Apply the same logic to validation set using training set's max finite value
        validation_processed[[col]][is.infinite(validation_processed[[col]])] <- max_finit
e_train + 1
    }
    # Handle NA values
    if (any(is.na(train_processed[[col]]))) {
        # Calculate mean from the training set (excluding NAs)
        mean_val_train <- mean(train_processed[[col]], na.rm = TRUE)
        # Impute NA values in training set with its mean
        train_processed[[col]][is.na(train_processed[[col]])] <- mean_val_train
        # Impute NA values in validation set with the training set's mean
        validation_processed[[col]][is.na(validation_processed[[col]])] <- mean_val_train
    }
}

# --- Re-evaluate all models on the fully processed data ---
cat("--- Evaluating all models on Cleaned & Transformed Data ---\n")
```

```
## --- Evaluating all models on Cleaned & Transformed Data ---
```

```r
# Retrain models using the processed training data
lm_model_cleaned <- lm(formula_all, data = train_processed)
rt_model_cleaned <- rpart(formula_all, data = train_processed, method = "anova")
set.seed(123) # Set seed for reproducibility for Random Forest
rf_model_cleaned <- randomForest(formula_all, data = train_processed, ntree = 500, importa
nce = TRUE) # importance=TRUE to get feature importance
svr_model_cleaned <- svm(formula_all, data = train_processed)

models_cleaned <- list(
    "Linear Regression" = lm_model_cleaned,
    "Regression Tree" = rt_model_cleaned,
    "Random Forest" = rf_model_cleaned,
    "SVR" = svr_model_cleaned
)

# Evaluate cleaned models on the processed validation data
rmse_cleaned <- sapply(models_cleaned, function(model) {
    preds <- predict(model, newdata = validation_processed)
    RMSE(validation_processed$Usefulness_score, preds)
})

performance_df_cleaned <- data.frame(Model = names(rmse_cleaned), RMSE = rmse_cleaned, Sta
ge = "Cleaned")
print(performance_df_cleaned %>% arrange(RMSE))
```

```
##                                  Model      RMSE   Stage
## Random Forest         Random Forest 0.9882938 Cleaned
## SVR                             SVR 0.9883981 Cleaned
## Linear Regression Linear Regression 1.1116204 Cleaned
## Regression Tree     Regression Tree 1.1935477 Cleaned
```

```
# --- Improvement Stage 2: Feature Selection ---
cat("--- Improvement Stage 2: Feature Selection ---\n")
```

```
## --- Improvement Stage 2: Feature Selection ---
```

```
# Use the Random Forest model trained on cleaned data to determine feature importance
importance_scores <- importance(rf_model_cleaned, type = 1) # Type 1 for %IncMSE
importance_df <- data.frame(Feature = rownames(importance_scores), Importance = importance
_scores[, 1]) %>%
    arrange(desc(Importance)) # Sort features by importance

# Select features with importance greater than the mean importance
mean_importance <- mean(importance_df$Importance)
selected_features <- as.character(importance_df$Feature[importance_df$Importance > mean_im
portance])

cat(paste("Selected", length(selected_features), "features with importance > mean importan
ce:\n"))
```

```
## Selected 10 features with importance > mean importance:
```

```
print(selected_features)
```

```
##  [1] "total_dialogue_length_words"      "ttr_chatbot"
##  [3] "total_words_chatbot"              "total_words_student"
##  [5] "ttr_student"                      "num_unique_words_chatbot"
##  [7] "num_unique_words_student"         "dialogue_duration"
##  [9] "num_utterances"                   "variance_time_between_utterances"
```

```
# --- Re-evaluate all models on the selected feature set ---
# Create a new formula with only the selected features
formula_selected <- as.formula(paste("Usefulness_score ~", paste(selected_features, collap
se = " + ")))

cat("--- Evaluating all models on Selected Features ---\n")
```

```
## --- Evaluating all models on Selected Features ---
```

```
# Retrain models using the processed training data and selected features
lm_model_selected <- lm(formula_selected, data = train_processed)
rt_model_selected <- rpart(formula_selected, data = train_processed, method = "anova")
set.seed(123) # Set seed for reproducibility
rf_model_selected <- randomForest(formula_selected, data = train_processed, ntree = 500)
svr_model_selected <- svm(formula_selected, data = train_processed)

models_selected <- list(
    "Linear Regression" = lm_model_selected,
    "Regression Tree" = rt_model_selected,
    "Random Forest" = rf_model_selected,
    "SVR" = svr_model_selected
)

# Evaluate selected feature models on the processed validation data
rmse_selected <- sapply(models_selected, function(model) {
    preds <- predict(model, newdata = validation_processed)
    RMSE(validation_processed$Usefulness_score, preds)
})

performance_df_selected <- data.frame(Model = names(rmse_selected), RMSE = rmse_selected,
Stage = "Feat. Selected")
print(performance_df_selected %>% arrange(RMSE))
```

```
##                               Model     RMSE          Stage
## Linear Regression Linear Regression 1.009699 Feat. Selected
## Random Forest         Random Forest 1.030625 Feat. Selected
## SVR                             SVR 1.043454 Feat. Selected
## Regression Tree     Regression Tree 1.275154 Feat. Selected
```

```
# --- Final Model Selection ---
cat("--- Final Model Selection ---\n")
```

```
## --- Final Model Selection ---
```

```
# Combine performance from all stages to identify the overall best model
full_performance <- rbind(performance_df_baseline, performance_df_cleaned, performance_df_
selected)
best_run <- full_performance[which.min(full_performance$RMSE), ]
cat("The best overall model is:\n")
```

```
## The best overall model is:
```

```
print(best_run)
```

```
##                        Model      RMSE   Stage
## Random Forest1 Random Forest 0.9882938 Cleaned
```

```r
# Store the final best model object and its associated feature list
# This allows for consistent prediction in subsequent steps.
best_overall_model <- NULL
final_feature_list <- NULL

if (best_run$Stage == "Baseline") {
    best_overall_model <- models_baseline[[best_run$Model]]
    final_feature_list <- features_to_use # Use all features from the baseline stage
} else if (best_run$Stage == "Cleaned") {
    best_overall_model <- models_cleaned[[best_run$Model]]
    final_feature_list <- features_to_use # Use all features after cleaning
} else { # Feat. Selected
    best_overall_model <- models_selected[[best_run$Model]]
    final_feature_list <- selected_features # Use only selected features
}
```

Model improvement was attempted through two stages:
1. **Advanced Data Processing:** Instead of simply omitting rows with `NA` / `Inf` , a more robust imputation strategy was applied. `NA` values were filled with the mean of their respective features from the training set, and `Inf` values were replaced by a value slightly greater than the maximum finite value observed in the training set. This helps retain more data and potentially provide more stable models.
2. **Feature Selection:** The feature importance from the `RandomForest` model (trained on the cleaned data) was used to select a subset of features. Only features with an importance score greater than the mean importance were retained. This aims to reduce noise, prevent overfitting, and potentially improve model generalization by focusing on the most relevant predictors.

The combined performance results from all stages indicate whether these improvements were successful in reducing the RMSE compared to the baseline models. The best-performing model overall, along with its specific stage (Baseline, Cleaned, or Feature Selected), is identified.

We find that of all these, only basic data cleaning helped. While taking out non-meaningful features forces the model to concentrate on more important variables, it can only maintain good performance after pruning, not improve in performance. Log scaling the appropriate data, while attempting to help with model linearity, proved to be poor for model performance.

Step 4: Analysis of a Specific Dialogue and Feature Importance

This section demonstrates how to use the best-performing model to predict the usefulness score for a randomly selected dialogue from the validation set. It then compares this prediction to the ground truth score and analyzes the importance of features in the model's decision-making process.

```r
cat("--- Step 4: Analyze a Specific Dialogue ---\n")
```

```
## --- Step 4: Analyze a Specific Dialogue ---
```

```r
# Determine which processed validation data to use based on the best model's stage
# This ensures consistency with how the best_overall_model was trained.
validation_data_for_pred <- if (best_run$Stage == "Baseline") {
    dialogue_features_validation_baseline
} else {
    validation_processed
}

# Select a random dialogue ID from the validation set for analysis
set.seed(42) # Set seed for reproducibility of random selection
random_dialogue_id <- sample(df_usefulness_validation$Dialogue_ID, 1)
cat(paste("Selected Dialogue_ID:", random_dialogue_id, "\n"))
```

```
## Selected Dialogue_ID: 5980
```

```r
# Get the feature vector for this specific dialogue from the appropriate processed validat
ion data
dialogue_feature_vector <- validation_data_for_pred %>% filter(Dialogue_ID == random_dialo
gue_id)

# Make a prediction for the selected dialogue using the best overall model.
# Crucially, only supply the features that the final_feature_list specifies.
predicted_score <- predict(best_overall_model, newdata = dialogue_feature_vector[, final_f
eature_list, drop = FALSE])
# Get the ground truth score for comparison
ground_truth_score <- dialogue_feature_vector$Usefulness_score

cat("--- Prediction vs. Ground Truth ---\n")
```

```
## --- Prediction vs. Ground Truth ---
```

```r
cat(paste("Ground Truth Score:", ground_truth_score, "\n"))
```

```
## Ground Truth Score: 4
```

```r
cat(paste("Predicted Score:", round(predicted_score, 2), "\n"))
```

```
## Predicted Score: 3.66
```

```r
# Check if the prediction is "close" to the ground truth (within 0.5)
is_close <- abs(ground_truth_score - predicted_score) <= 0.5
cat(paste("Is the prediction close to the ground truth (within 0.5)?", is_close, "\n"))
```

```
## Is the prediction close to the ground truth (within 0.5)? TRUE
```

```
if (is_close) {
    cat("The model made a successful prediction.\n")
} else {
    cat("The prediction was not close to the ground truth. Possible reasons include:\n")
    cat("- The model may not capture the specific nuances of this conversation.\n")
    cat("- The dialogue might be an outlier or have unique characteristics not well-repres
ented in the training data.\n")
    cat("- The features, while generally useful, failed to describe what made this particu
lar dialogue useful or unuseful.\n\n")
}
```

```
## The model made a successful prediction.
```

```
cat("--- Feature Importance Analysis ---\n")
```

```
## --- Feature Importance Analysis ---
```

```
# Display the sorted feature importance table.
# This table was generated during the feature selection stage in Step 3.
importance_df_sorted <- importance_df %>% arrange(desc(Importance))
print(importance_df_sorted)
```

```
##                                                          Feature Importance
## total_dialogue_length_words          total_dialogue_length_words 11.9458648
## ttr_chatbot                                          ttr_chatbot 10.0849169
## total_words_chatbot                          total_words_chatbot  9.4560726
## total_words_student                          total_words_student  8.8272918
## ttr_student                                          ttr_student  8.7649075
## num_unique_words_chatbot                num_unique_words_chatbot  7.9448840
## num_unique_words_student                num_unique_words_student  6.6270364
## dialogue_duration                              dialogue_duration  6.4590941
## num_utterances                                    num_utterances  5.8024219
## variance_time_between_utterances variance_time_between_utterances  5.6732675
## ratio_student_chatbot_len_words   ratio_student_chatbot_len_words  4.7864840
## num_student_questions                      num_student_questions  3.8177256
## avg_len_student_utterance_words   avg_len_student_utterance_words  2.2314800
## num_chatbot_questions                      num_chatbot_questions  0.8920748
## avg_readability_score_chatbot       avg_readability_score_chatbot  0.4065591
## avg_len_chatbot_utterance_words   avg_len_chatbot_utterance_words -0.2416096
## avg_readability_score_student       avg_readability_score_student -0.6224998
```

Explanation of Feature Importance (%IncMSE): The table above displays the importance of each feature for our Random Forest model, quantified by '%IncMSE' (Percentage Increase in Mean Squared Error). This metric indicates how much the model's prediction error (MSE) would increase if a specific feature's values were randomly shuffled, breaking its relationship with the target variable (Usefulness_score).

- **Higher positive %IncMSE values:** Indicate more important features. If shuffling a feature significantly increases the MSE, it means the model relies heavily on that feature for accurate predictions.
- **Lower or negative %IncMSE values:** Suggest less important features. Features with values close to zero or negative may not be contributing positively to the model's performance, or their inclusion

might be adding noise.

If the prediction for the selected dialogue was close to the ground truth, the features with high importance scores (as shown in the `importance_df_sorted` table) are likely the ones that played significant roles in enabling the model to make that successful prediction. These features generally provide the most information for predicting dialogue usefulness across the dataset. If the prediction was not close, it could be due to this specific dialogue having characteristics not well-captured by the existing features or being an outlier not well-represented in the training data.

Step 5: Predict Usefulness on the Test Set and Generate Submission File

The final step is to apply the best-performing model (identified in Step 3) to predict the usefulness scores for dialogues in the test set. The predictions are then formatted and saved into a CSV file, ready for submission.

```
cat("--- Step 5: Predict on Test Set ---\n")
```

```
## --- Step 5: Predict on Test Set ---
```

```
# --- 1. Load and Process Test Data ---
df_utterance_test <- read.csv("git_ignore/dialogue_utterance_test.csv")
df_usefulness_test_orig <- read.csv("git_ignore/dialogue_usefulness_test.csv") # Original
structure for submission

# Merge and process test data, similar to training and validation data
df_merged_test <- left_join(df_utterance_test, df_usefulness_test_orig, by = "Dialogue_I
D") %>%
    mutate(Timestamp = ymd_hms(Timestamp)) %>%
    arrange(Dialogue_ID, Timestamp)

# Engineer features for the test set
df_merged_test_readable <- calculate_readability(df_merged_test)
test_processed <- engineer_features(df_merged_test_readable)

# --- 2. Apply the SAME processing pipeline as the best model ---
# This ensures the test data is preprocessed identically to how the best model was traine
d.
# The imputation values are derived from the training set statistics to prevent data leaka
ge.
cat("Applying imputation to test data...\n")
```

```
## Applying imputation to test data...
```

```
for (col in features_to_use) { # Iterate through the full set of features used in modeling
stages
    # Handle NA values using the mean from the training set
    if (any(is.na(test_processed[[col]]))) {
        # Check if train_processed (from Step 3) is available; if not, use baseline train
data
        if (exists("train_processed")) {
            mean_val_train <- mean(train_processed[[col]], na.rm = TRUE)
        } else {
            mean_val_train <- mean(dialogue_features_train_baseline[[col]], na.rm = TRUE)
        }
        test_processed[[col]][is.na(test_processed[[col]])] <- mean_val_train
    }
    # Handle Infinite values using the max finite value from the training set
    if (any(is.infinite(test_processed[[col]]))) {
        if (exists("train_processed")) {
            max_finite_train <- max(train_processed[[col]][!is.infinite(train_processed[[c
ol]])], na.rm = TRUE)
        } else {
            max_finite_train <- max(dialogue_features_train_baseline[[col]][!is.infinite(d
ialogue_features_train_baseline[[col]])], na.rm = TRUE)
        }
        test_processed[[col]][is.infinite(test_processed[[col]])] <- max_finite_train + 1
    }
}

# --- 3. Make Predictions on the Test Set ---
# Use the best_overall_model and its corresponding final_feature_list
test_predictions <- predict(best_overall_model, newdata = test_processed[, final_feature_l
ist, drop = FALSE])

# Ensure predictions are within the valid score range [1, 5]
test_predictions <- pmax(1, pmin(5, test_predictions))

# --- 4. Create and Save Submission File ---
# Create a dataframe with Dialogue_ID and predicted Usefulness_score
submission_df <- data.frame(
    Dialogue_ID = test_processed$Dialogue_ID,
    Usefulness_score = test_predictions
)

# Reorder the submission_df to match the original df_usefulness_test_orig order
submission_df <- submission_df[match(df_usefulness_test_orig$Dialogue_ID, submission_df$Di
alogue_ID), ]

# Define the output filename
output_filename <- "Leong_27030768_dialogue_usefulness_test.csv"

# Write the submission file to CSV without row names and without quoting strings
write.csv(submission_df, output_filename, row.names = FALSE, quote = FALSE)

cat(paste("Submission file saved as:", output_filename, "\n"))
```

```
## Submission file saved as: Leong_27030768_dialogue_usefulness_test.csv
```

```
print(head(submission_df))
```

```
##   Dialogue_ID Usefulness_score
## 1        5427         4.217600
## 2        5452         3.819000
## 3        5454         3.592400
## 4        5479         3.848867
## 5        5522         3.423333
## 6        5536         3.953133
```

The process ensures that the test data undergoes the same preprocessing steps (feature engineering, imputation) as the training data, applying parameters (like means for imputation) learned *only* from the training set. The best-performing model from Step 3 is then used to generate predictions. Finally, the predictions are constrained to the valid range of [1, 5] and formatted into the required CSV submission file.