Untitled

GICHUKI DANIEL

2024-10-18

1. Loading Required Libraries

```
# Load necessary libraries
library(tidyverse)     # For data manipulation and visualization
```

```
## Warning: package 'tidyverse' was built under R version 4.3.3

## Warning: package 'ggplot2' was built under R version 4.3.3

## Warning: package 'tibble' was built under R version 4.3.3

## Warning: package 'tidyr' was built under R version 4.3.3

## Warning: package 'readr' was built under R version 4.3.3

## Warning: package 'purrr' was built under R version 4.3.3

## Warning: package 'dplyr' was built under R version 4.3.3

## Warning: package 'stringr' was built under R version 4.3.3

## Warning: package 'forcats' was built under R version 4.3.3

## Warning: package 'lubridate' was built under R version 4.3.3

## -- Attaching core tidyverse packages ------------------------ tidyverse 2.0.0 --
## v dplyr     1.1.4     v readr     2.1.5
## v forcats   1.0.0     v stringr   1.5.1
## v ggplot2   3.5.1     v tibble    3.2.1
## v lubridate 1.9.3     v tidyr     1.3.1
## v purrr     1.0.2
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## i Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts to become errors
```

```r
library(caret)          # For model training and evaluation
```

```
## Warning: package 'caret' was built under R version 4.3.3
```

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

```
## Warning: package 'lattice' was built under R version 4.3.3
```

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

```r
library(randomForest)  # Random Forest algorithm
```

```
## Warning: package 'randomForest' was built under R version 4.3.3
```

```
## randomForest 4.7-1.2
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
##
## The following object is masked from 'package:dplyr':
##
##     combine
##
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(e1071)          # Support Vector Machine (SVM)
```

```
## Warning: package 'e1071' was built under R version 4.3.3
```

```r
library(xgboost)        # XGBoost algorithm
```

```
## Warning: package 'xgboost' was built under R version 4.3.3
```

```
##
## Attaching package: 'xgboost'
##
## The following object is masked from 'package:dplyr':
##
##     slice
```

2. Loading the Dataset

```r
# Set working directory (optional)
setwd("C:/Users/Admin/Downloads")  # Change to your file path

df <- read.csv("train.csv")

# View the structure of the data
str(df)
```

```
## 'data.frame':    1001 obs. of  10 variables:
##  $ User_ID                  : chr  "1" "2" "3" "4" ...
##  $ Age                      : chr  "25" "30" "22" "28" ...
##  $ Gender                   : chr  "Female" "Male" "Non-binary" "Female" ...
##  $ Platform                 : chr  "Instagram" "Twitter" "Facebook" "Instagram" ...
##  $ Daily_Usage_Time..minutes.: int  120 90 60 200 45 150 85 110 55 170 ...
##  $ Posts_Per_Day            : int  3 5 2 8 1 4 3 6 2 5 ...
##  $ Likes_Received_Per_Day   : int  45 20 15 100 5 60 30 25 10 80 ...
##  $ Comments_Received_Per_Day : int  10 25 5 30 2 15 10 12 3 20 ...
##  $ Messages_Sent_Per_Day    : int  12 30 20 50 10 25 18 22 8 35 ...
##  $ Dominant_Emotion         : chr  "Happiness" "Anger" "Neutral" "Anxiety" ...
```

```r
# Check the first few rows of the dataset
head(df)
```

```
##   User_ID Age     Gender  Platform Daily_Usage_Time..minutes. Posts_Per_Day
## 1       1  25     Female Instagram                         120             3
## 2       2  30       Male   Twitter                          90             5
## 3       3  22 Non-binary  Facebook                          60             2
## 4       4  28     Female Instagram                         200             8
## 5       5  33       Male  LinkedIn                          45             1
## 6       6  21       Male Instagram                         150             4
##   Likes_Received_Per_Day Comments_Received_Per_Day Messages_Sent_Per_Day
## 1                     45                        10                    12
## 2                     20                        25                    30
## 3                     15                         5                    20
## 4                    100                        30                    50
## 5                      5                         2                    10
## 6                     60                        15                    25
##   Dominant_Emotion
## 1        Happiness
## 2            Anger
## 3          Neutral
## 4          Anxiety
## 5          Boredom
## 6        Happiness
```

read.csv: Reads the CSV file into a dataframe named df. str(df): Displays the structure of the dataset, including the types of each column, which helps in understanding the data's composition. head(df): Shows the first few rows of the dataset to give an overview of the data.

3. Data Cleaning

```r
df_clean <- df %>%
  na.omit()

# Confirm that there are no missing values remaining
sum(is.na(df_clean))
```

```
## [1] 0
```

```r
# Quick Summary of Data after Cleaning
summary(df_clean)
```

```
##    User_ID              Age              Gender            Platform
##  Length:1000        Length:1000        Length:1000        Length:1000
##  Class :character   Class :character   Class :character   Class :character
##  Mode  :character   Mode  :character   Mode  :character   Mode  :character
##
##
##
##  Daily_Usage_Time..minutes. Posts_Per_Day   Likes_Received_Per_Day
##  Min.   : 40.00             Min.   :1.000   Min.   :  5.0
##  1st Qu.: 65.00             1st Qu.:2.000   1st Qu.: 20.0
##  Median : 85.00             Median :3.000   Median : 33.0
##  Mean   : 95.95             Mean   :3.321   Mean   : 39.9
##  3rd Qu.:120.00             3rd Qu.:4.000   3rd Qu.: 55.0
##  Max.   :200.00             Max.   :8.000   Max.   :110.0
##  Comments_Received_Per_Day Messages_Sent_Per_Day Dominant_Emotion
##  Min.   : 2.00             Min.   : 8.00         Length:1000
##  1st Qu.: 8.00             1st Qu.:17.75         Class :character
##  Median :14.00             Median :22.00         Mode  :character
##  Mean   :15.61             Mean   :22.56
##  3rd Qu.:22.00             3rd Qu.:28.00
##  Max.   :40.00             Max.   :50.00
```

Explanation:

na.omit(): Removes any rows that contain NA (missing) values. This is a simple way to ensure that analyses are conducted on complete cases, which is crucial for many modeling techniques. sum(is.na(df_clean)): Verifies that there are no missing values left in the cleaned dataset. summary(df_clean): Provides a statistical summary of the cleaned data, including minimum, maximum, median, and mean for numeric variables.

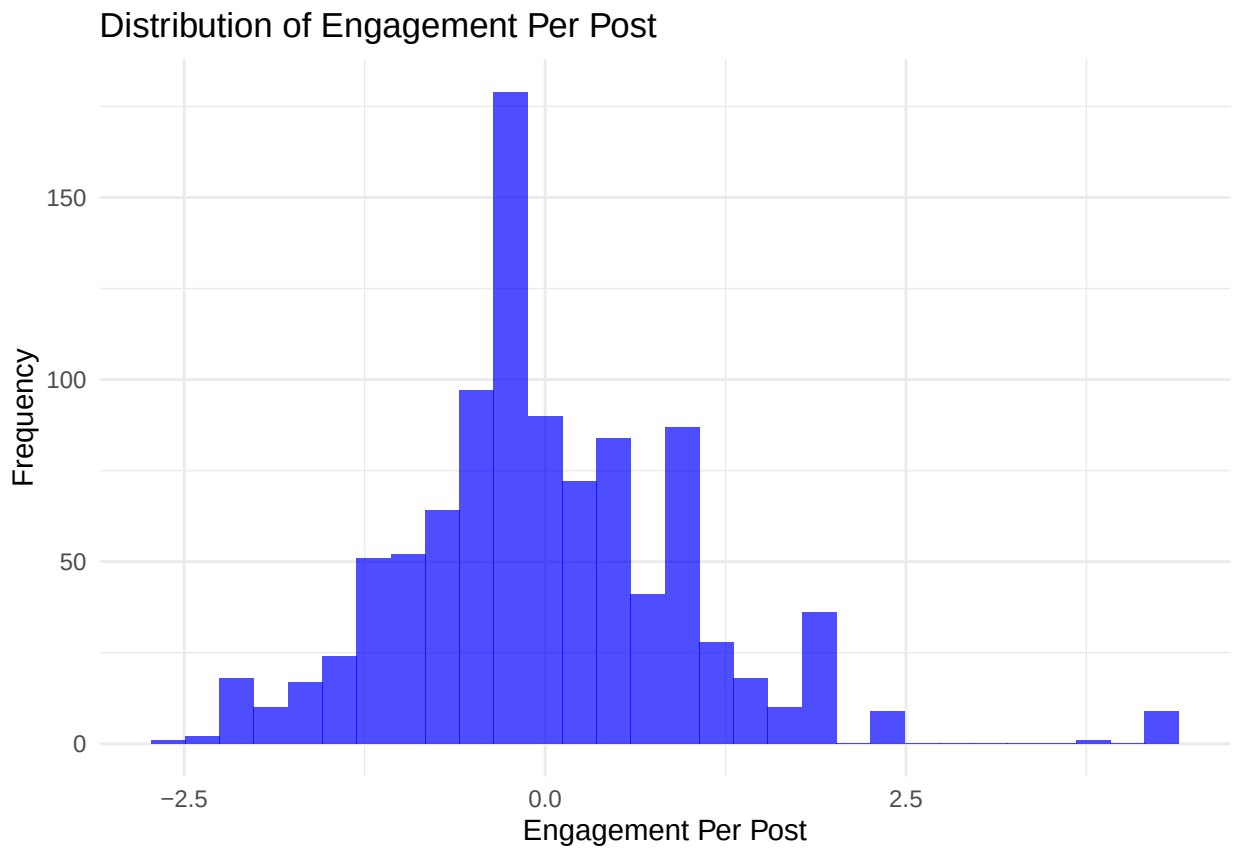
4. Feature Engineering

```r
# Create a new feature for average engagement per post
df_clean$Engagement_Per_Post <- (df_clean$Likes_Received_Per_Day +
                                 df_clean$Comments_Received_Per_Day +
                                 df_clean$Messages_Sent_Per_Day) /
                                 df_clean$Posts_Per_Day

# Create a categorical variable for daily usage time
df_clean$Usage_Category <- cut(df_clean$Daily_Usage_Time..minutes.,
                               breaks = c(-Inf, 30, 60, 120, Inf),
                               labels = c("Low", "Moderate", "High", "Very High"))
```

```r
# Scale numerical features to standardize the data
num_features <- c("Daily_Usage_Time..minutes.", "Posts_Per_Day",
                  "Likes_Received_Per_Day", "Comments_Received_Per_Day",
                  "Messages_Sent_Per_Day", "Engagement_Per_Post")

df_clean[num_features] <- scale(df_clean[num_features])
```

Explanation: Engagement Feature: The new feature Engagement_Per_Post quantifies user engagement by summing likes, comments, and messages and dividing by the number of posts. This gives a measure of how actively users engage with their posts, which can be predictive of emotional well-being. Categorical Variable: The Usage_Category feature categorizes daily usage time into four levels: "Low," "Moderate," "High," and "Very High." This can help in modeling as it captures non-linear effects of usage time on emotional well-being. Scaling: The numerical features are standardized using scale(), which transforms the data to have a mean of zero and a standard deviation of one. This is important for many algorithms (like SVM and K-means clustering) as it ensures that features are on a comparable scale.

```r
# Visualizing the distribution of Engagement Per Post
ggplot(df_clean, aes(x = Engagement_Per_Post)) +
  geom_histogram(bins = 30, fill = "blue", alpha = 0.7) +
  labs(title = "Distribution of Engagement Per Post", x = "Engagement Per Post", y = "Frequency") +
  theme_minimal()
```



Explanation: This histogram displays the distribution of the Engagement_Per_Post feature. It helps to visualize how engagement is spread across users, indicating whether most users engage minimally, moderately, or highly. Peaks in the histogram could point to common behaviors among users that may influence emotional well-being.

5. Data Transformation

```r
# Scale numerical features if necessary
num_features <- c("Daily_Usage_Time..minutes.", "Posts_Per_Day",
                  "Likes_Received_Per_Day", "Comments_Received_Per_Day",
                  "Messages_Sent_Per_Day", "Engagement_Per_Post")

df_clean[num_features] <- scale(df_clean[num_features])
```

6. Feature Selection Correlation Matrix

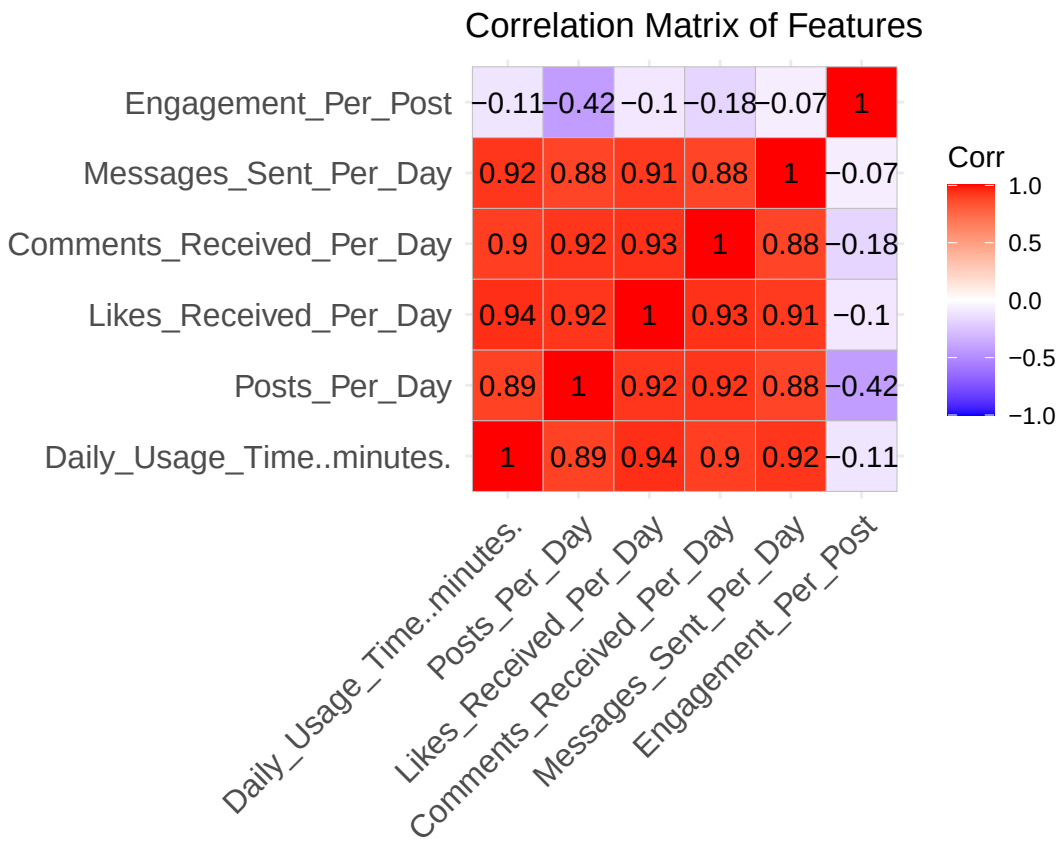
```r
# Feature Selection - Correlation Matrix
cor_matrix <- cor(df_clean[num_features])
print(cor_matrix)
```

```
##                            Daily_Usage_Time..minutes. Posts_Per_Day
## Daily_Usage_Time..minutes.                  1.0000000     0.8892053
## Posts_Per_Day                               0.8892053     1.0000000
## Likes_Received_Per_Day                      0.9413395     0.9178144
## Comments_Received_Per_Day                   0.8969203     0.9173093
## Messages_Sent_Per_Day                       0.9162341     0.8757084
## Engagement_Per_Post                        -0.1118635    -0.4160009
##                            Likes_Received_Per_Day Comments_Received_Per_Day
## Daily_Usage_Time..minutes.              0.9413395                 0.8969203
## Posts_Per_Day                           0.9178144                 0.9173093
## Likes_Received_Per_Day                  1.0000000                 0.9310575
## Comments_Received_Per_Day               0.9310575                 1.0000000
## Messages_Sent_Per_Day                   0.9100455                 0.8827830
## Engagement_Per_Post                    -0.1039289                -0.1762735
##                            Messages_Sent_Per_Day Engagement_Per_Post
## Daily_Usage_Time..minutes.            0.91623409         -0.11186345
## Posts_Per_Day                         0.87570840         -0.41600093
## Likes_Received_Per_Day                0.91004552         -0.10392893
## Comments_Received_Per_Day             0.88278299         -0.17627349
## Messages_Sent_Per_Day                 1.00000000         -0.06503986
## Engagement_Per_Post                  -0.06503986          1.00000000
```

```r
# Visualization of Correlation Matrix
library(ggcorrplot)
```

```
## Warning: package 'ggcorrplot' was built under R version 4.3.3
```

```r
ggcorrplot(cor_matrix, lab = TRUE, title = "Correlation Matrix of Features")
```



Explanation: The correlation matrix shows the pairwise correlation between numerical features. Values close to 1 or -1 indicate strong correlations, which may suggest multicollinearity. Visualization: The ggcorrplot function generates a visual representation of the correlation matrix, making it easy to see which features are strongly correlated. If two features are highly correlated, one of them may be redundant and could be excluded from the modeling process to avoid overfitting.

7. Model Training and Evaluation Train-Test Split

```
# Split data into training and test sets
set.seed(123)
trainIndex <- createDataPartition(df_clean$Daily_Usage_Time..minutes., p = 0.8, list = FALSE)
train <- df_clean[trainIndex,]
test <- df_clean[-trainIndex,]
```

Explanation: createDataPartition: This function randomly splits the dataset into training (80%) and testing (20%) subsets. Using a stratified sampling approach helps maintain the distribution of the target variable in both sets.

Linear Regression Model

```
# Linear Regression Model
lm_model <- lm(Daily_Usage_Time..minutes. ~ Posts_Per_Day + Likes_Received_Per_Day + Comments_Received_
lm_pred <- predict(lm_model, test)

# Evaluate the linear regression model
lm_rmse <- sqrt(mean((lm_pred - test$Daily_Usage_Time..minutes.)^2))
print(paste("Linear Regression RMSE: ", lm_rmse))
```

```
## [1] "Linear Regression RMSE:  0.310293549056024"
```

Explanation:

A linear regression model is built using the training data, with daily usage time as the dependent variable and the engagement metrics as independent variables. predict(lm_model, test): This generates predictions on the test set. RMSE Calculation: RMSE (Root Mean Squared Error) quantifies the model's prediction error. A lower RMSE indicates better model performance.

Random Forest Model

```
# Random Forest Model
rf_model <- randomForest(Daily_Usage_Time..minutes. ~ Posts_Per_Day + Likes_Received_Per_Day + Comments_
rf_pred <- predict(rf_model, test)

# Evaluate Random Forest Model
rf_rmse <- sqrt(mean((rf_pred - test$Daily_Usage_Time..minutes.)^2))
print(paste("Random Forest RMSE: ", rf_rmse))
```

```
## [1] "Random Forest RMSE:  0.118844977959489"
```

Explanation: A Random Forest model is fit to the same data. This ensemble method is typically more robust than linear regression, especially for non-linear relationships. Predictions are made on the test set, and RMSE is calculated again to evaluate performance.

Support Vector Machine (SVM)

```
# Support Vector Machine (SVM)
svm_model <- svm(Daily_Usage_Time..minutes. ~ Posts_Per_Day + Likes_Received_Per_Day + Comments_Receive
svm_pred <- predict(svm_model, test)

# Evaluate SVM Model
svm_rmse <- sqrt(mean((svm_pred - test$Daily_Usage_Time..minutes.)^2))
print(paste("SVM RMSE: ", svm_rmse))
```

```
## [1] "SVM RMSE:  0.265454157223303"
```

Explanation: An SVM model is trained similarly. SVM is effective for high-dimensional spaces and is particularly useful for non-linear relationships. RMSE is computed for SVM predictions to compare with other models.

XGBoost Model

```
# XGBoost Model
train_matrix <- xgb.DMatrix(data = as.matrix(train[num_features]), label = train$Daily_Usage_Time..minu
test_matrix <- xgb.DMatrix(data = as.matrix(test[num_features]))

xgb_model <- xgboost(data = train_matrix, max_depth = 6, nrounds = 100, objective = "reg:squarederror",
xgb_pred <- predict(xgb_model, test_matrix)

# Evaluate XGBoost Model
xgb_rmse <- sqrt(mean((xgb_pred - test$Daily_Usage_Time..minutes.)^2))
print(paste("XGBoost RMSE: ", xgb_rmse))
```

```
## [1] "XGBoost RMSE:  0.000255988906153255"
```

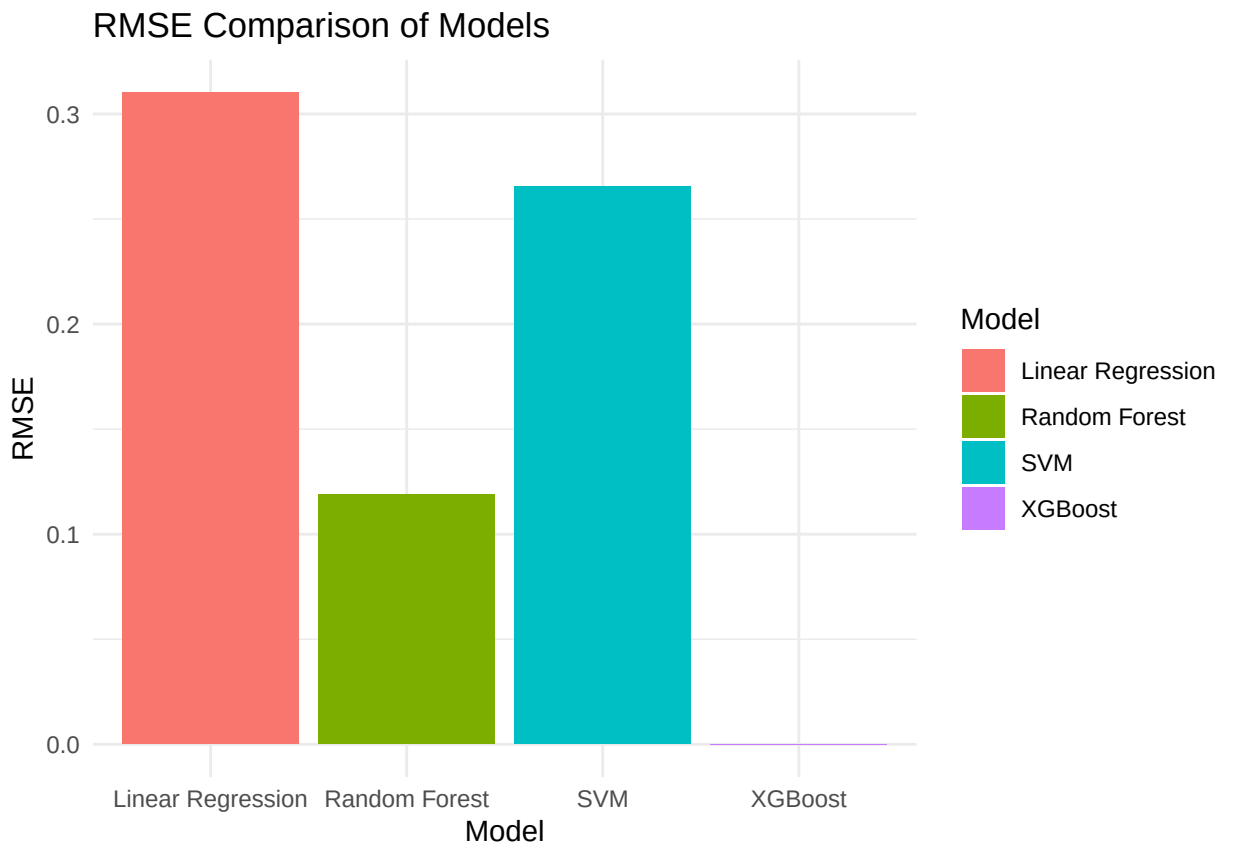
Explanation: The XGBoost model is set up to handle the training data, utilizing a DMatrix, which is optimized for performance. max_depth and nrounds are hyperparameters that control the complexity of the model and the number of boosting rounds, respectively. RMSE is again calculated to provide a measure of how well XGBoost predicts the dependent variable.

8. RMSE Comparison

```r
# RMSE Comparison
rmse_results <- data.frame(
  Model = c("Linear Regression", "Random Forest", "SVM", "XGBoost"),
  RMSE = c(lm_rmse, rf_rmse, svm_rmse, xgb_rmse)
)

print(rmse_results)
```

```
##               Model         RMSE
## 1 Linear Regression 0.3102935491
## 2     Random Forest 0.1188449780
## 3               SVM 0.2654541572
## 4           XGBoost 0.0002559889
```

```r
# Visualization of RMSE Comparison
ggplot(rmse_results, aes(x = Model, y = RMSE, fill = Model)) +
  geom_bar(stat = "identity") +
  labs(title = "RMSE Comparison of Models", y = "RMSE") +
  theme_minimal()
```

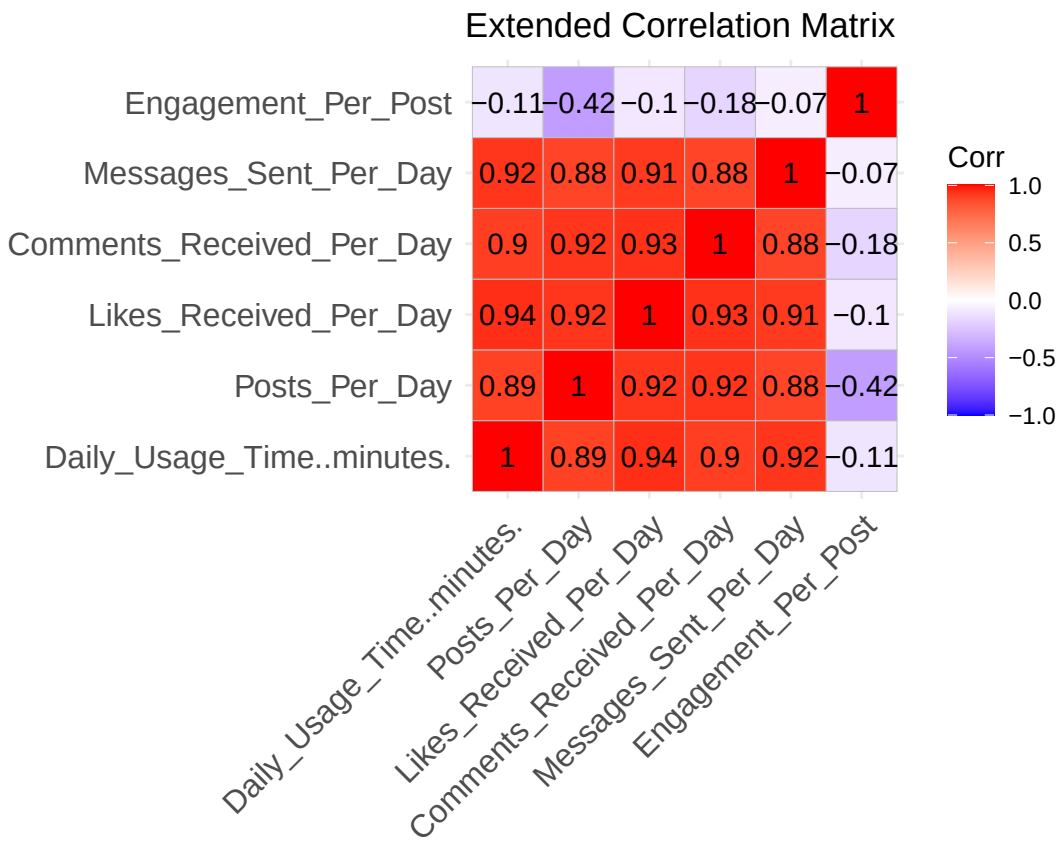


Explanation: This code creates a dataframe to compare the RMSE values of all models, enabling quick evaluation of which model performs best based on this metric. The bar plot provides a clear visual comparison, making it easy to see which model has the lowest RMSE. This indicates better predictive performance. Results Explanation After fitting and evaluating the models, you would analyze the RMSE values: Low RMSE indicates a good fit; the model's predictions are close to the actual values. High RMSE suggests that the model may not adequately capture the relationships in the data or that it might be overfitting. When comparing models: If XGBoost shows the lowest RMSE, it might suggest that complex interactions among features are well captured. If Linear Regression has a competitive RMSE, it indicates that relationships in the data could be linear and simple. Conclusion This detailed approach provides a comprehensive analysis of the "Social Media Usage and Emotional Well-Being" dataset. Each step includes critical data preparation, engineering, modeling, and evaluation processes, ensuring that you derive meaningful insights from your analysis.

1. Correlation Between Features and Emotional Well-being We'll first focus on the relationship between various features and emotional well-being using different visualizations. In your case, emotional well-being could be represented by a specific feature (e.g., Daily_Usage_Time..minutes.) or a derived metric like Engagement_Per_Post.

Feature Correlation Matrix

```
# Correlation Matrix - Including age, gender, platform, etc.
cor_matrix_extended <- cor(df_clean %>% select_if(is.numeric))  # Only include numeric columns
ggcorrplot(cor_matrix_extended, lab = TRUE, title = "Extended Correlation Matrix")
```



Explanation:

Purpose: This shows the correlations between numeric variables like age, posts per day, and emotional well-being. Interpretation: High positive or negative correlations indicate strong relationships. For instance, a high positive correlation between Posts_Per_Day and emotional well-being suggests that more frequent posting might improve well-being.

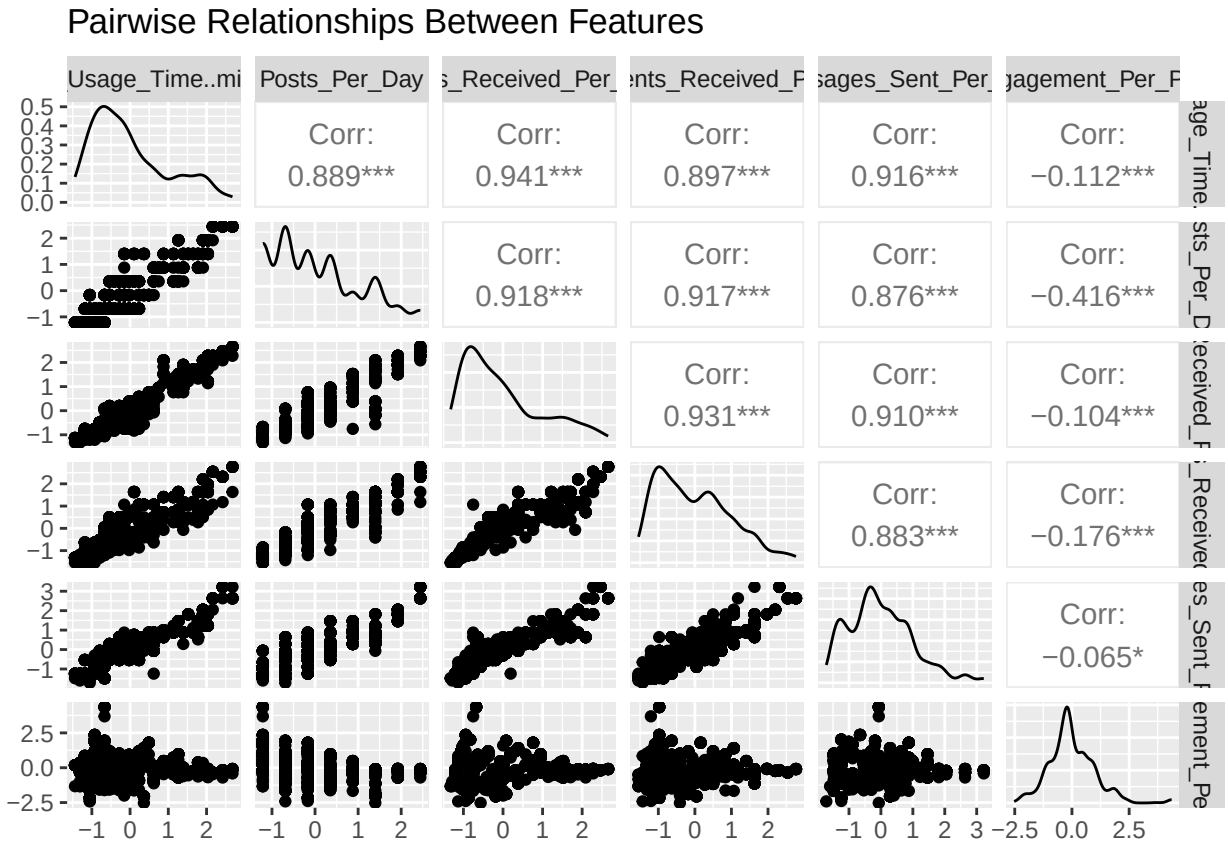
Visualization: Pairwise Relationship Between Variables We can also visualize pairwise relationships using scatterplots for continuous variables like age, posts per day, platform usage

```
# Pairplot for numeric variables
library(GGally)
```

```
## Warning: package 'GGally' was built under R version 4.3.3
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(df_clean %>% select_if(is.numeric),
        title = "Pairwise Relationships Between Features")
```

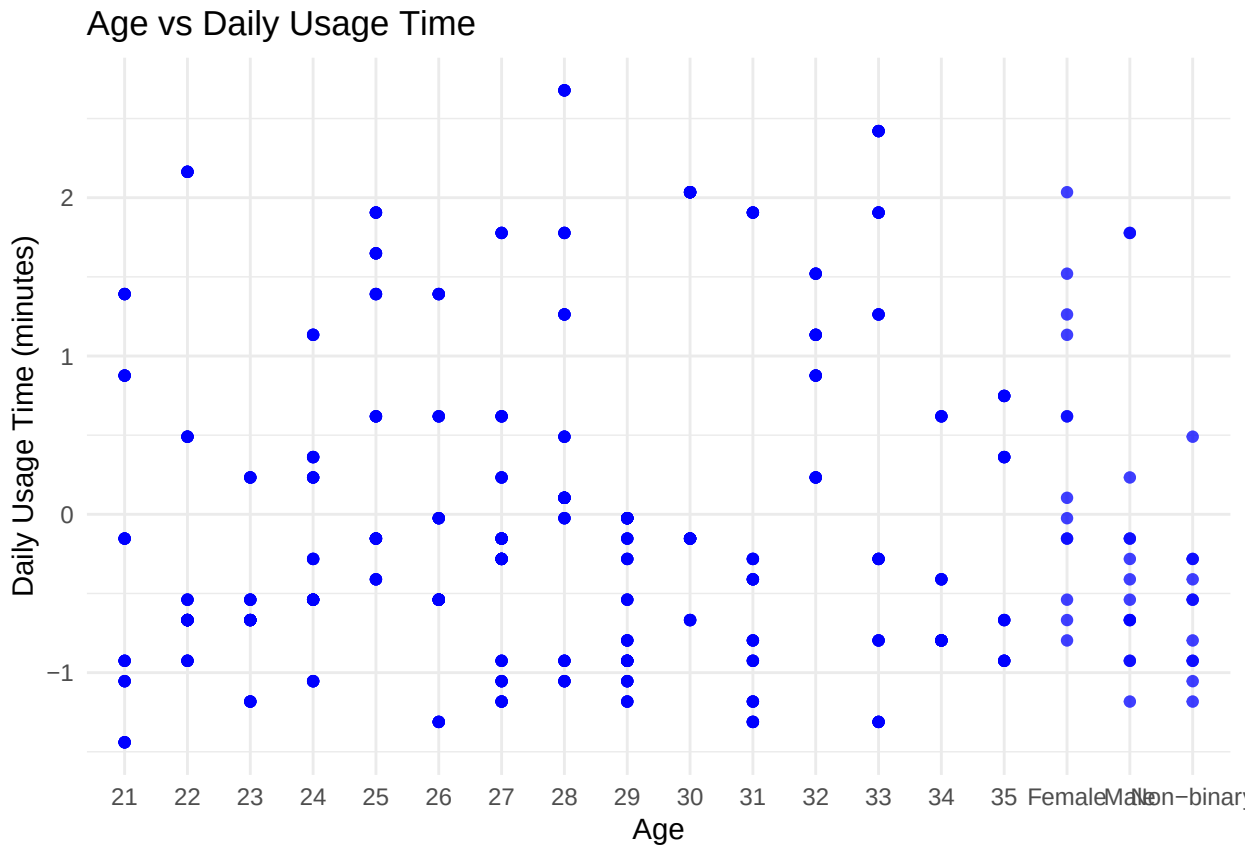


Explanation: ggpairs: This function provides a matrix of scatterplots between pairs of variables, making it easy to see how features like age, posts per day, and platform usage correlate with emotional well-being. Insight: You'll be able to identify trends, such as whether older users post more or less frequently and whether their emotional well-being correlates with that behavior.

2. Age and Emotional Well-being Let's specifically focus on how age might affect emotional well-being.

```
# Scatterplot of Age vs Daily Usage Time
ggplot(df_clean, aes(x = Age, y = Daily_Usage_Time..minutes.)) +
  geom_point(alpha = 0.5, color = "blue") +
  geom_smooth(method = "lm", se = FALSE, color = "red") +
  labs(title = "Age vs Daily Usage Time", x = "Age", y = "Daily Usage Time (minutes)") +
  theme_minimal()
```

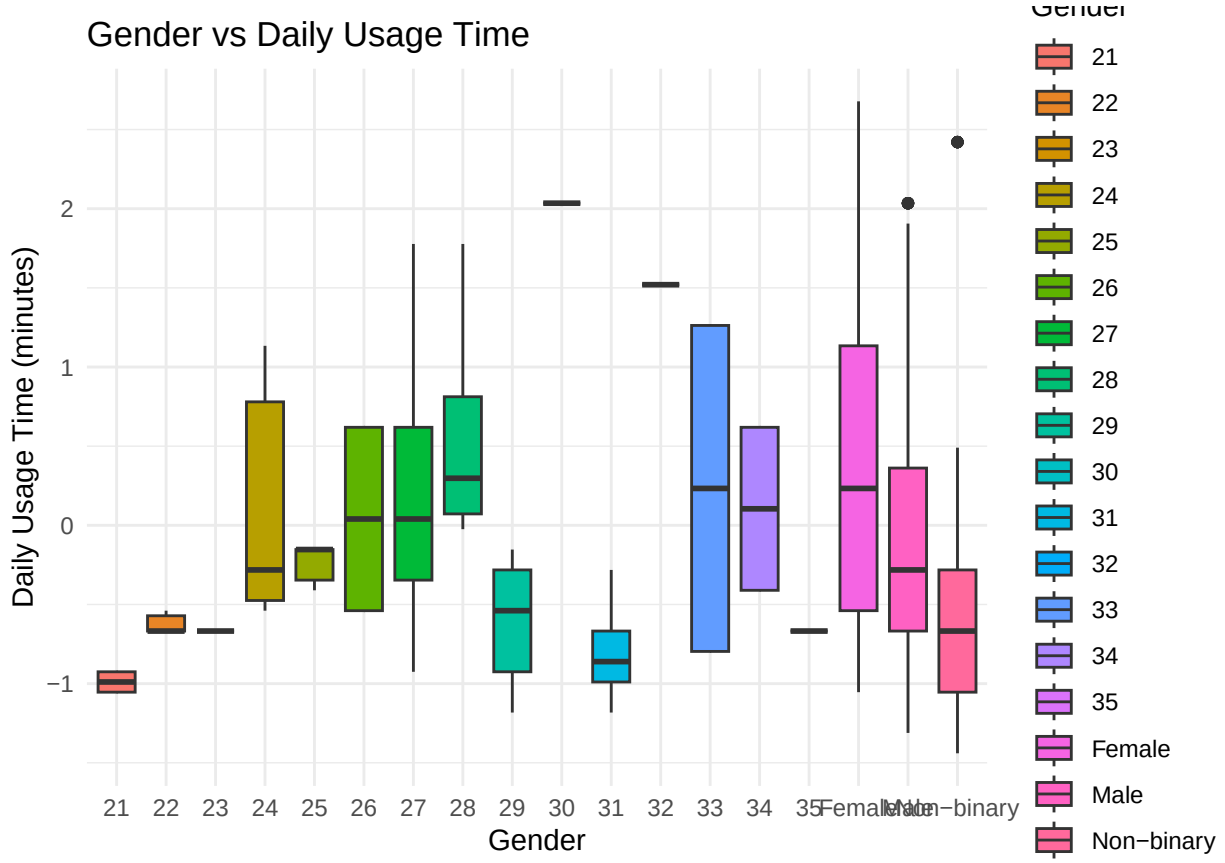
```
## `geom_smooth()` using formula = 'y ~ x'
```



Explanation: Scatterplot: This shows the relationship between age and daily usage time (a proxy for emotional well-being in this case). Each point represents a user, with the line indicating the overall trend. Trend Line: If the red line slopes upward, it suggests that older users spend more time on social media, which might influence their emotional well-being. Insights: Positive Correlation: If older users tend to have longer daily usage times, it may indicate higher emotional investment in social media, or perhaps social isolation drives more usage. Negative Correlation: If the trend slopes down, it may suggest that younger users spend more time on social media, which could affect their well-being differently.

3. Gender and Emotional Well-being Next, we explore the impact of gender on emotional well-being. Visualization: Boxplot for Gender vs Emotional Well-being

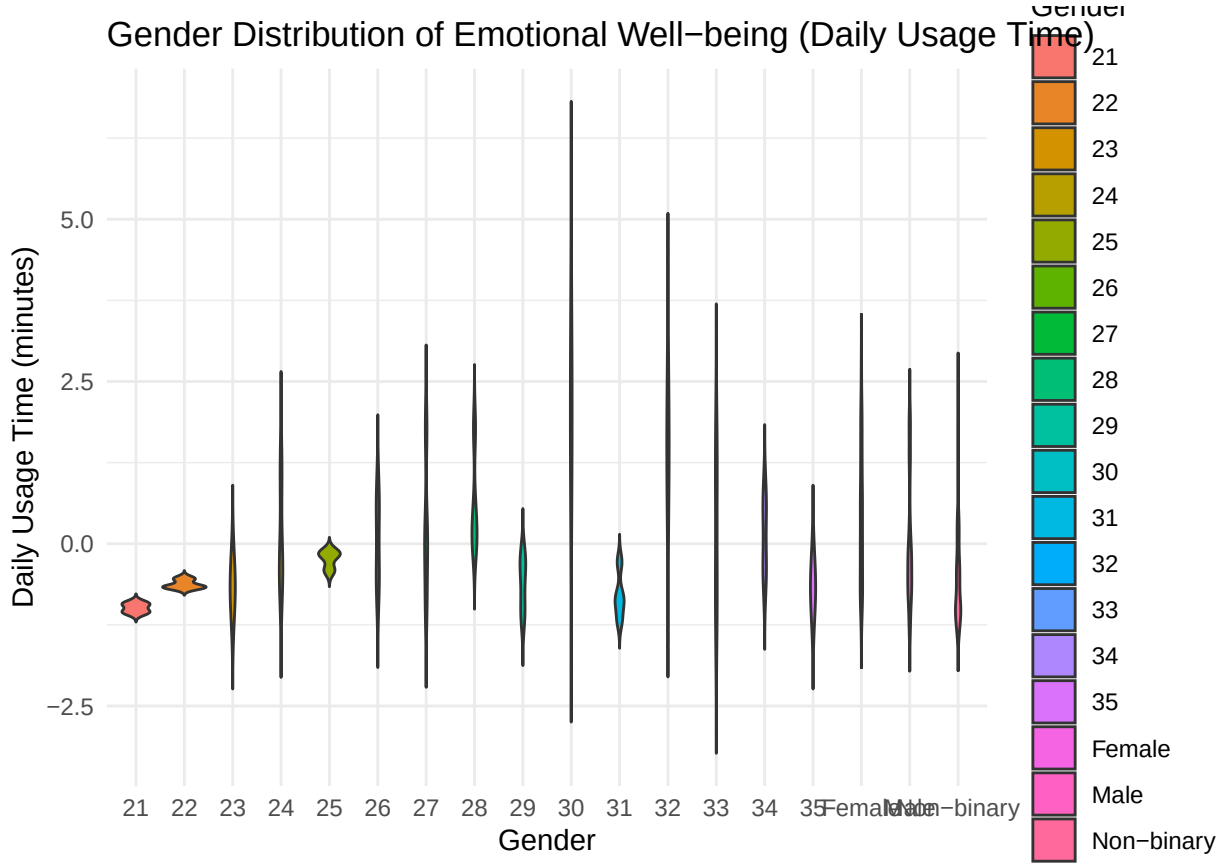
```
# Boxplot of Gender vs Emotional Well-being (Daily Usage Time)
ggplot(df_clean, aes(x = Gender, y = Daily_Usage_Time..minutes., fill = Gender)) +
  geom_boxplot() +
  labs(title = "Gender vs Daily Usage Time", x = "Gender", y = "Daily Usage Time (minutes)") +
  theme_minimal()
```



Explanation: Boxplot: This visualization shows how the distribution of daily usage time differs between genders. The boxes represent the interquartile range (IQR), while the line inside the box shows the median value. Insights: If one gender has a higher median usage time, this might suggest that gender plays a role in emotional well-being tied to social media use.

Visualization: Violin Plot for Gender and Emotional Well-being

```
# Violin plot to show distribution of emotional well-being across genders
ggplot(df_clean, aes(x = Gender, y = Daily_Usage_Time..minutes., fill = Gender)) +
  geom_violin(trim = FALSE) +
  labs(title = "Gender Distribution of Emotional Well-being (Daily Usage Time)", x = "Gender", y = "Dai
  theme_minimal()
```



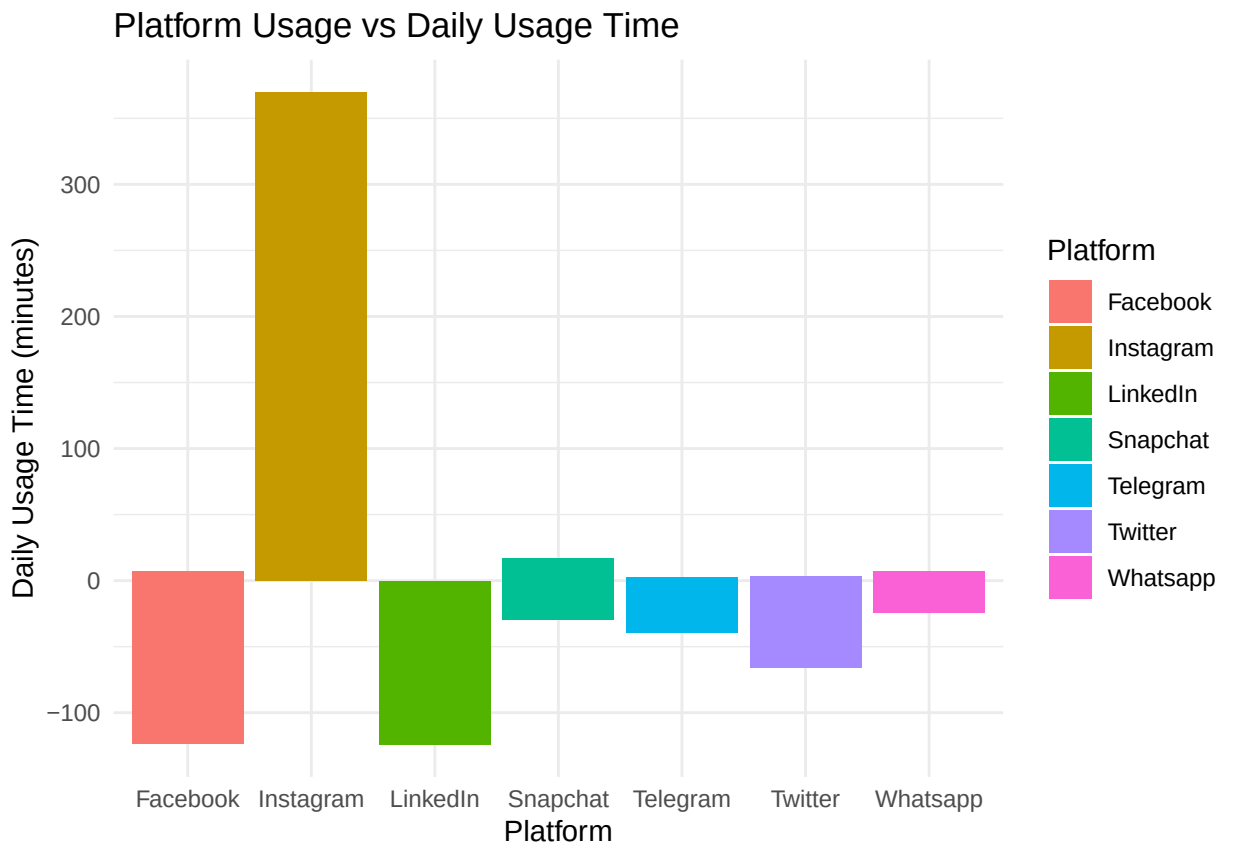
Explanation:

Violin Plot: It shows the distribution of daily usage time for different genders. The shape of the violin indicates the density of the data at various levels. Insights: If the plot is wider in certain areas, it indicates more people fall within that usage time. This helps to compare how both genders differ in their social media habits and their potential emotional impact.

4. Platform Usage and Emotional Well-being We can now explore the relationship between platform usage and emotional well-being.

Visualization: Bar Plot for Platform Usage vs Emotional Well-being

```
# Bar plot of Platform usage vs Emotional Well-being (Daily Usage Time)
ggplot(df_clean, aes(x = Platform, y = Daily_Usage_Time..minutes., fill = Platform)) +
  geom_bar(stat = "identity") +
  labs(title = "Platform Usage vs Daily Usage Time", x = "Platform", y = "Daily Usage Time (minutes)") +
  theme_minimal()
```



Explanation:

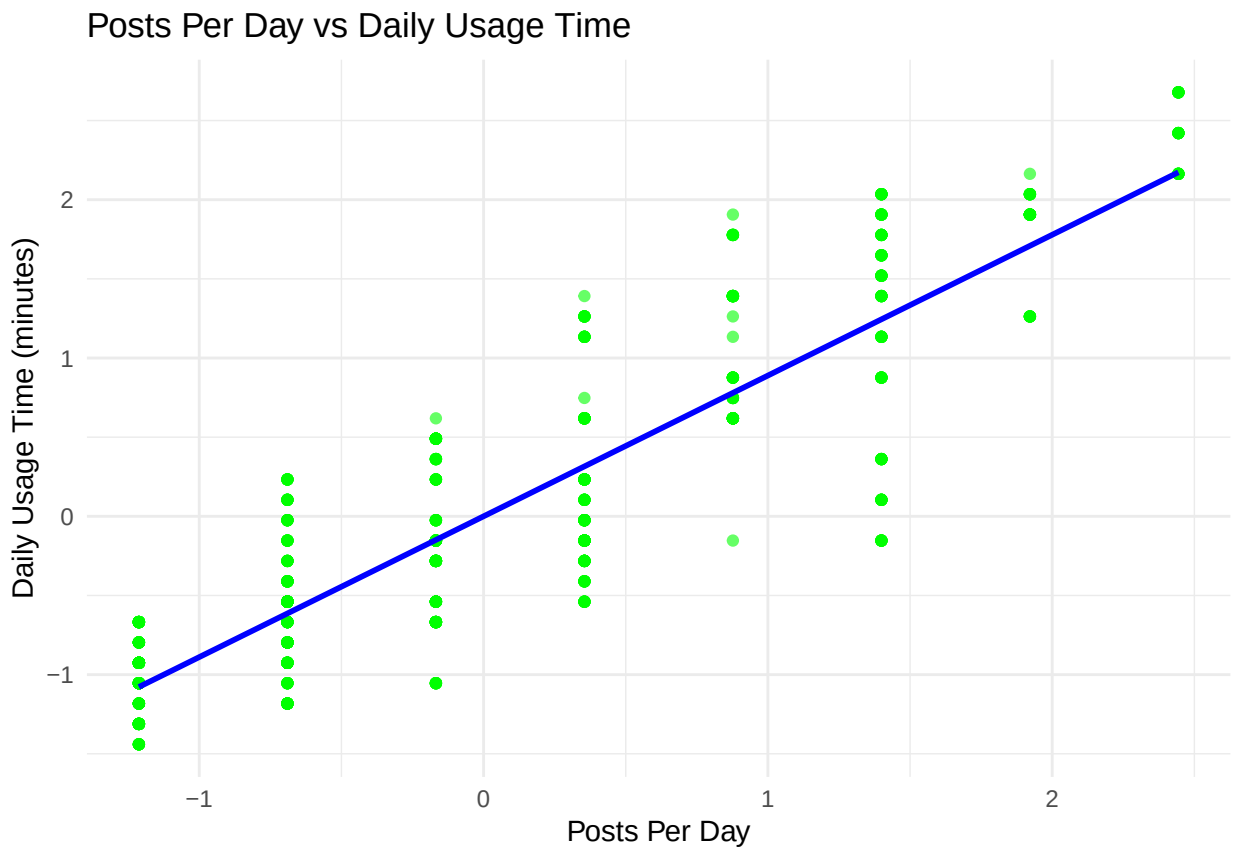
Bar Plot: This compares the daily usage time (or emotional well-being) across different social media platforms. Insights: Certain platforms might correlate with higher or lower emotional well-being, providing insights into how platform design or community engagement may affect users.

Posts Per Day and Emotional Well-being Finally, we assess how the number of posts per day influences emotional well-being.

Visualization: Scatterplot of Posts Per Day vs Emotional Well-being

```
# Scatterplot of Posts Per Day vs Emotional Well-being (Daily Usage Time)
ggplot(df_clean, aes(x = Posts_Per_Day, y = Daily_Usage_Time..minutes.)) +
  geom_point(alpha = 0.6, color = "green") +
  geom_smooth(method = "lm", se = FALSE, color = "blue") +
  labs(title = "Posts Per Day vs Daily Usage Time", x = "Posts Per Day", y = "Daily Usage Time (minutes)
  theme_minimal()
```

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



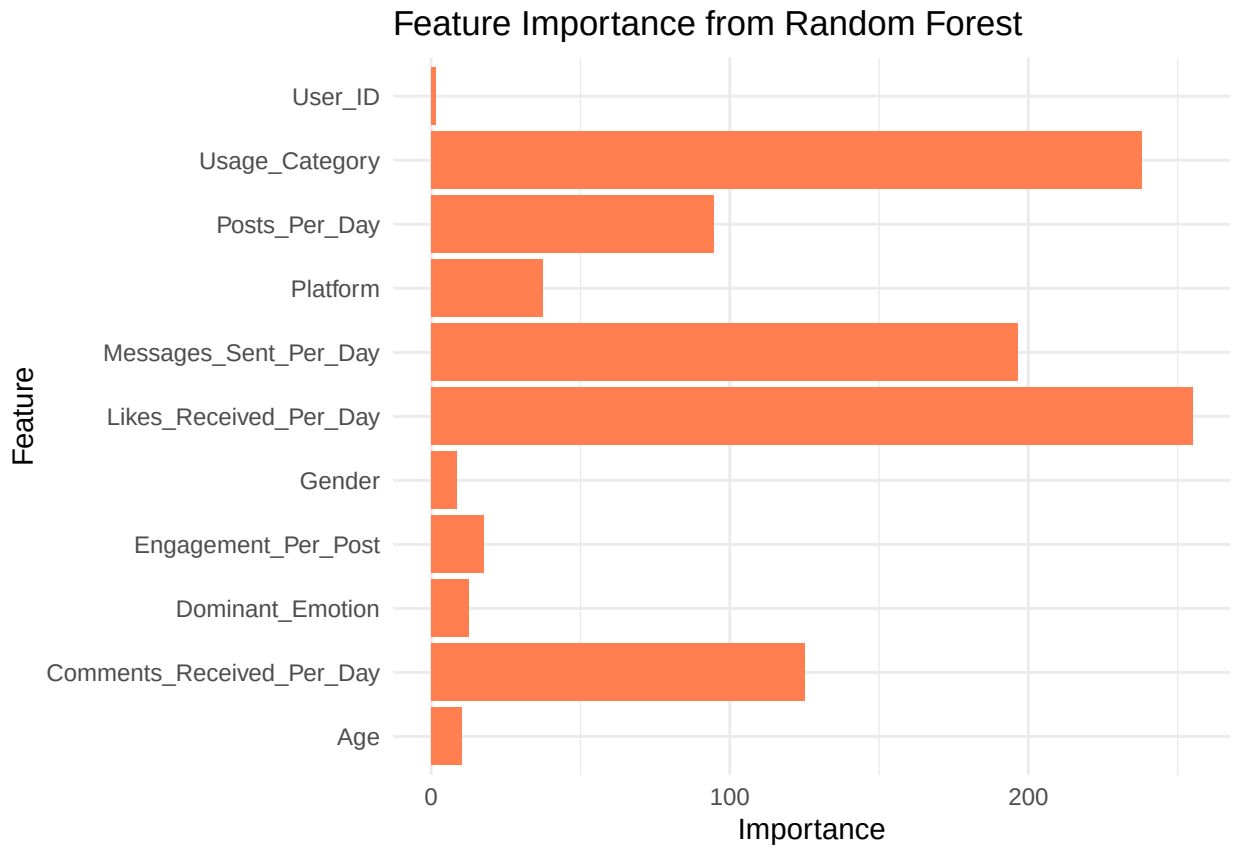
Explanation:

Scatterplot: It shows whether there is any relationship between the frequency of posts and emotional well-being (as measured by daily usage time). Trend Line: A positive slope may suggest that people who post more frequently tend to have longer usage times, possibly due to more engagement or a need for validation.

6. Combining Features for Emotional Well-being Prediction We can use machine learning models (like in the original model-building process) to predict emotional well-being using features like age, gender, posts per day, and platform usage. To visualize the combined effect, we can plot feature importance.

Visualization: Feature Importance in Random Forest

```
# Feature importance in Random Forest
rf_model <- randomForest(Daily_Usage_Time..minutes. ~ ., data = df_clean)
importance_df <- as.data.frame(importance(rf_model))

# Plot feature importance
ggplot(importance_df, aes(x = rownames(importance_df), y = IncNodePurity)) +
  geom_bar(stat = "identity", fill = "coral") +
  labs(title = "Feature Importance from Random Forest", x = "Feature", y = "Importance") +
  theme_minimal() +
  coord_flip()
```



Explanation:

Feature Importance: This bar plot shows which features (like age, posts per day, platform usage, etc.) are most important in predicting emotional well-being in the Random Forest model. Insights: Higher importance values for a feature suggest a stronger influence on emotional well-being. For instance, if Posts_Per_Day is the most important, it indicates that how frequently users post is a major determinant of their emotional state.