Final Project on Sleep Score Report

AUTHOR
Jatin Adya and Sharath Reddy Muthyala

```
rm(list = ls())
```

DATA IMPORT

```
library(dplyr)
```

```
Attaching package: 'dplyr'

The following objects are masked from 'package:stats':

    filter, lag

The following objects are masked from 'package:base':

    intersect, setdiff, setequal, union
```

```
library(tidyverse)
```

```
── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
✔ forcats   1.0.0     ✔ readr     2.1.4
✔ ggplot2   3.5.0     ✔ stringr   1.5.0
✔ lubridate 1.9.3     ✔ tibble    3.2.1
✔ purrr     1.0.2     ✔ tidyr     1.3.1

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

```
library(caret)
```

```
Loading required package: lattice

Attaching package: 'caret'

The following object is masked from 'package:purrr':

    lift
```

```r
library(ggplot2)
library(gridExtra) # Bivariate Analysis (Scatter Plots)
```

```
Attaching package: 'gridExtra'

The following object is masked from 'package:dplyr':

    combine
```

```r
# Read data
df <- read.csv("sleep_score.csv")

# Glimpse of data
glimpse(df)
```

```
Rows: 162
Columns: 9
$ sleep_log_entry_id    <dbl> 45343602090, 45325371774, 45299063506, 452881402…
$ timestamp             <chr> "2024-04-25T07:06:30Z", "2024-04-23T14:04:30Z", …
$ overall_score         <int> 70, 74, 71, 75, 54, 55, 61, 62, 62, 77, 58, 56, …
$ composition_score     <int> 16, 16, 20, 19, 16, 12, 15, 20, 14, 20, 17, 17, …
$ revitalization_score  <int> 21, 20, 10, 20, 14, 16, 20, 12, 21, 14, 18, 11, …
$ duration_score        <int> 33, 38, 41, 36, 24, 27, 26, 30, 27, 43, 23, 28, …
$ deep_sleep_in_minutes <int> 37, 51, 110, 25, 28, 30, 28, 109, 12, 124, 36, 5…
$ resting_heart_rate    <int> 63, 64, 64, 66, 65, 63, 65, 63, 65, 64, 71, 68, …
$ restlessness          <dbl> 0.11122770, 0.10490857, 0.20503331, 0.11210191, …
```

This dataset represents a sleep log, containing information about various sleep sessions. It consists of 162 rows and 9 columns.

Here's a breakdown of the columns:

1. **sleep_log_entry_id**: A unique identifier for each sleep log entry.
2. **timestamp**: The timestamp indicating when the sleep session occurred.
3. **overall_score**: An overall score representing the quality of the sleep session.
4. **composition_score**: A score indicating the composition quality of the sleep (e.g., how well it's structured).
5. **revitalization_score**: A score indicating how well the sleep session revitalized the individual.
6. **duration_score**: A score indicating the duration of the sleep session.
7. **deep_sleep_in_minutes**: The duration of deep sleep during the session, measured in minutes.
8. **resting_heart_rate**: The resting heart rate during the sleep session.

9. **restlessness**: A measure of restlessness during sleep, likely a decimal value representing a percentage or ratio.

Overall, this dataset is for analyzing sleep patterns and evaluating the quality of sleep sessions based on various metrics like overall score, composition, duration, deep sleep duration, resting heart rate, and restlessness.

```
# Summary of the data
summary(df)
```

```
 sleep_log_entry_id   timestamp         overall_score   composition_score
 Min.   :4.315e+10   Length:162         Min.   :44.00   Min.   :10.00
 1st Qu.:4.374e+10   Class :character   1st Qu.:61.25   1st Qu.:16.00
 Median :4.424e+10   Mode  :character   Median :68.50   Median :17.00
 Mean   :4.425e+10                      Mean   :67.70   Mean   :17.48
 3rd Qu.:4.475e+10                      3rd Qu.:75.00   3rd Qu.:19.00
 Max.   :4.534e+10                      Max.   :90.00   Max.   :23.00

 revitalization_score duration_score  deep_sleep_in_minutes resting_heart_rate
 Min.   : 8.00        Min.   :16.00   Min.   :  5.00        Min.   :59.00
 1st Qu.:15.00        1st Qu.:27.00   1st Qu.: 34.00        1st Qu.:64.00
 Median :18.00        Median :33.00   Median : 50.00        Median :67.00
 Mean   :17.49        Mean   :32.73   Mean   : 54.63        Mean   :66.65
 3rd Qu.:20.00        3rd Qu.:38.75   3rd Qu.: 74.00        3rd Qu.:69.00
 Max.   :23.00        Max.   :46.00   Max.   :125.00        Max.   :77.00
                                      NA's   :2

  restlessness
 Min.   :0.04714
 1st Qu.:0.09102
 Median :0.10559
 Mean   :0.10879
 3rd Qu.:0.12253
 Max.   :0.23274
```

This dataset summarizes sleep quality metrics for various sleep sessions. On average, the overall sleep scores fall around 67.70, indicating a decent quality of sleep. The composition and revitalization scores also show moderate values, suggesting reasonably structured and refreshing sleep. The duration of sleep sessions ranges from 16 to 46, with an average of 32.73, and deep sleep duration averages around 54.63 minutes. Resting heart rates range from 59 to 77 beats per minute, with an average of 66.65. Restlessness, a measure of sleep disturbance, averages around 0.10879, with values ranging between 0.04714 and 0.23274.

DATA CLEANING

```
# Missing Values
colSums(is.na(df))
```

```
   sleep_log_entry_id             timestamp         overall_score 
                    0                     0                     0 
    composition_score  revitalization_score        duration_score 
                    0                     0                     0 
deep_sleep_in_minutes    resting_heart_rate          restlessness 
                    2                     0                     0 
```

There seem to be 2 missing values in the deep_sleep_in_minutes column.

```
# omit rows with missing values
df_clean <- na.omit(df)
```

Those two rows are emitted from the data.

DATA TIDYING

```
# Convert Date column to date type if it is not already
df_clean <- df_clean %>%
  mutate(timestamp = as.Date(timestamp, format = "%Y-%m-%d"))
```

```
# Arranging data according to the date using timestamp
df_clean <- df_clean %>%
  arrange(timestamp)
```

```
# Rename column 'timestamp' to 'date' in df
names(df_clean)[names(df_clean) == "timestamp"] <- "date"
```

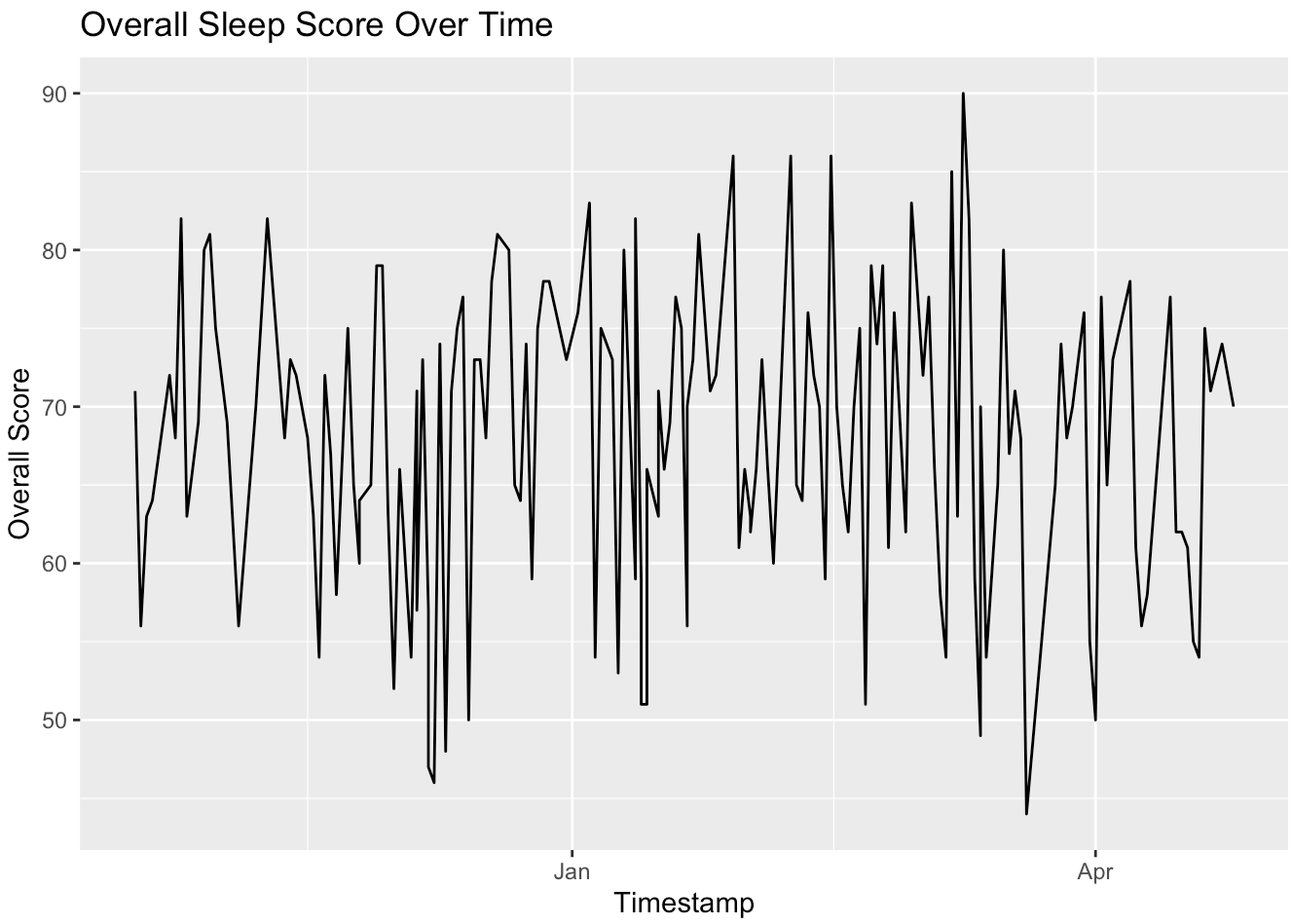
```
# First few rows of the data
head(df_clean)
```

```
  sleep_log_entry_id       date overall_score composition_score
1        43148572558 2023-10-17            71                18
2        43160869706 2023-10-18            56                19
3        43176502103 2023-10-19            63                15
4        43186718890 2023-10-20            64                17
5        43220788082 2023-10-23            72                17
6        43232084392 2023-10-24            68                19
  revitalization_score duration_score deep_sleep_in_minutes resting_heart_rate
1                   21             32                    52                 72
2                   18             19                    36                 67
3                   19             29                    24                 67
4                   14             33                    45                 64
5                   21             34                    38                 63
6                   14             35                    77                 61
  restlessness
1   0.11210191
2   0.11785095
```

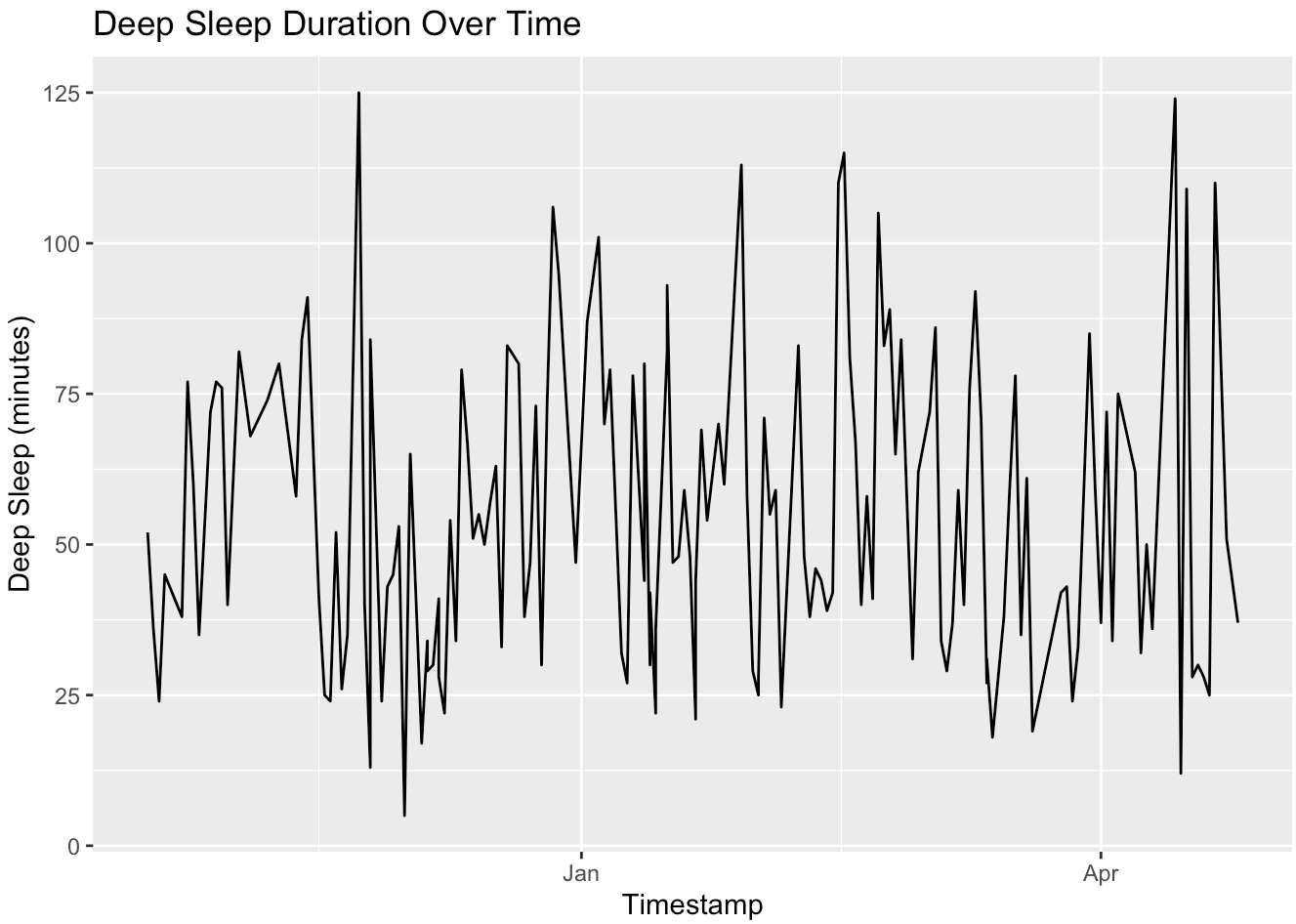
```
3   0.07925801
4   0.10671067
5   0.10532276
6   0.10408432
```

DATA EXPLORATION

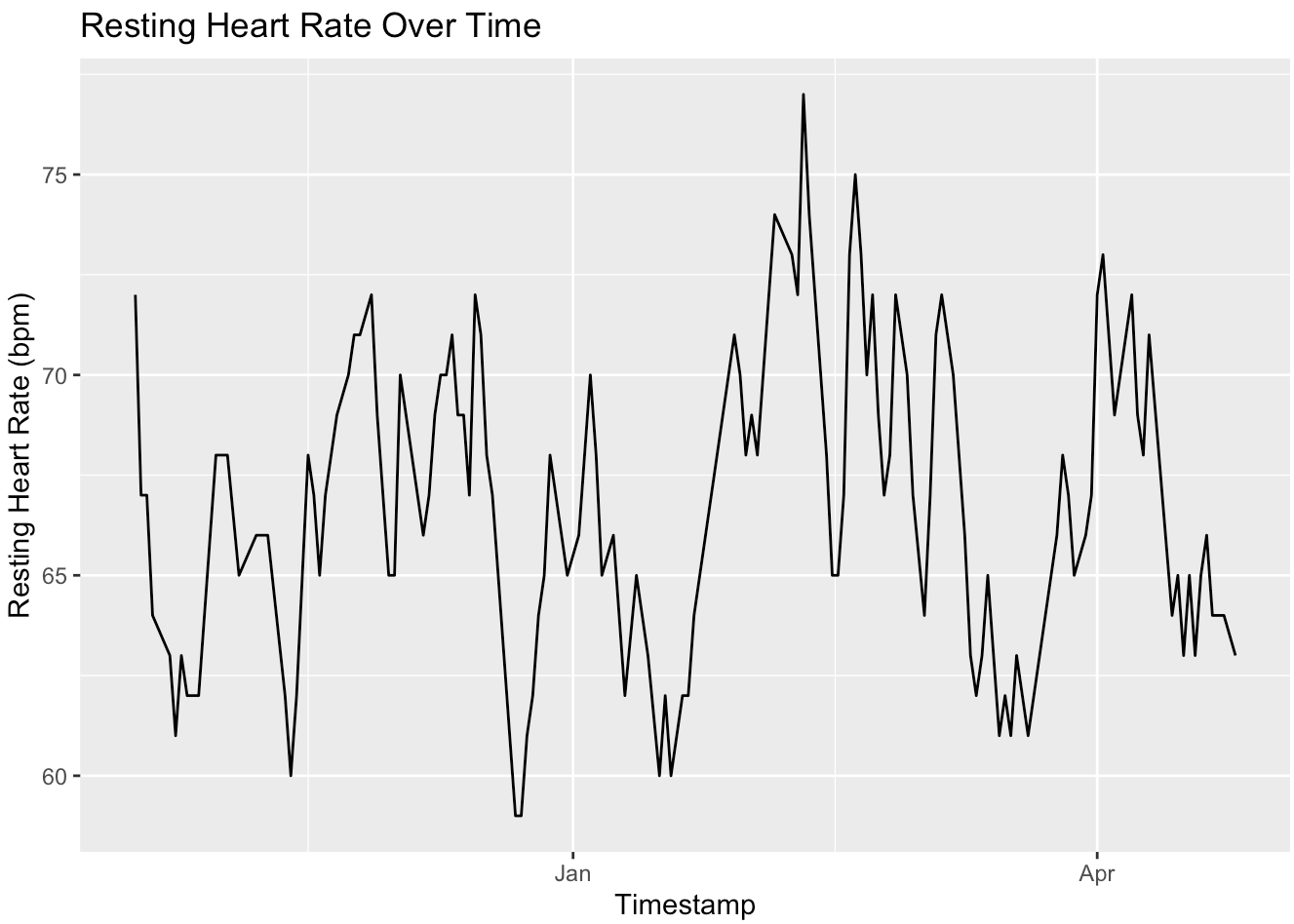
```r
# Trend of Overall Sleep Score over Time
ggplot(df_clean, aes(x=date, y=overall_score)) +
  geom_line() +
  labs(title="Overall Sleep Score Over Time", x="Timestamp", y="Overall Score")
```



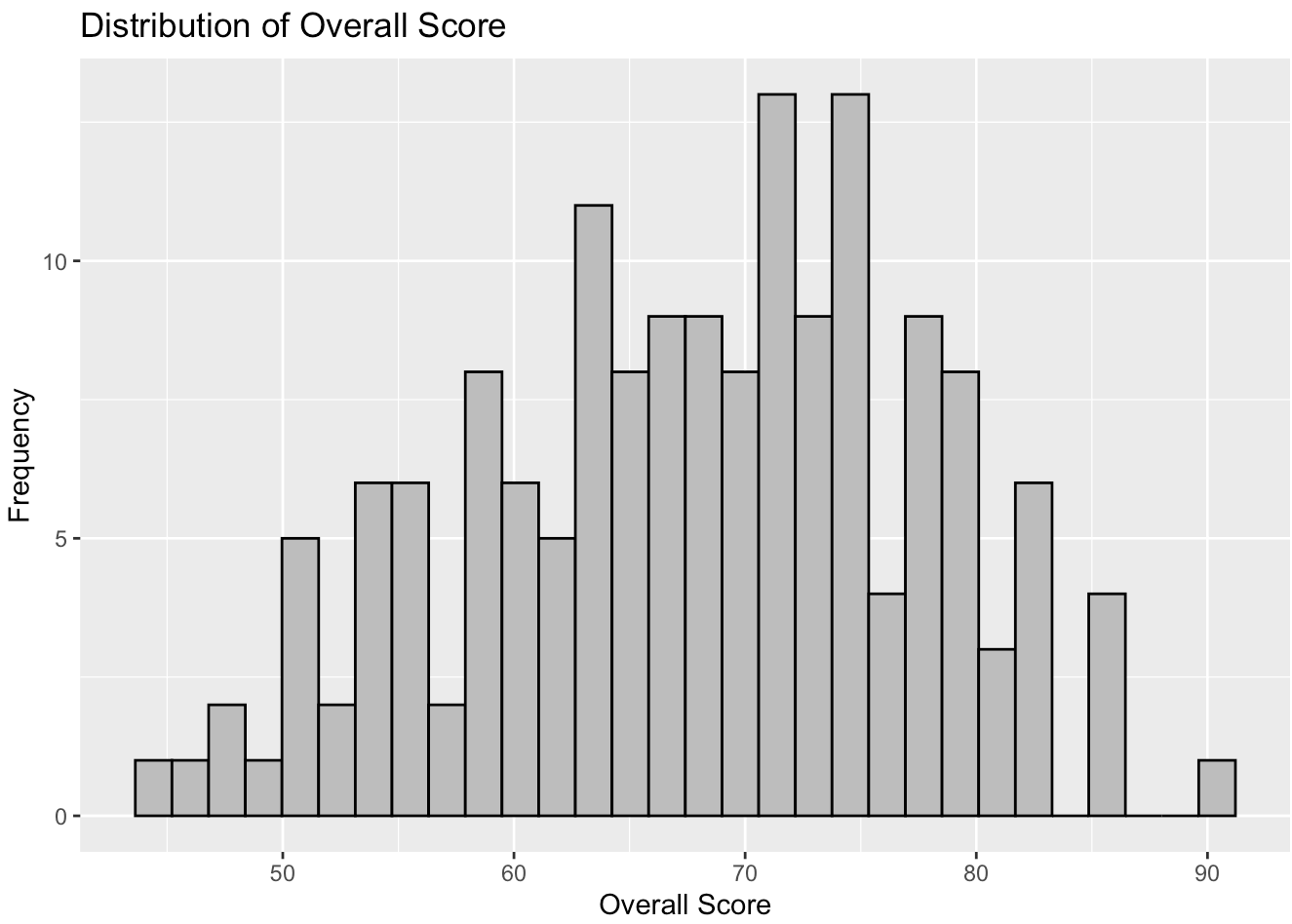

```r
# Trend of Deep Sleep Duration over Time
ggplot(df_clean, aes(x=date, y=deep_sleep_in_minutes)) +
  geom_line() +
  labs(title="Deep Sleep Duration Over Time", x="Timestamp", y="Deep Sleep (minutes)")
```




```
# Trend of Resting Heart Rate over Time
ggplot(df_clean, aes(x=date, y=resting_heart_rate)) +
  geom_line() +
  labs(title="Resting Heart Rate Over Time", x="Timestamp", y="Resting Heart Rate (bpm)")
```




```r
# Histogram of 'overall_score'
ggplot(df_clean, aes(x = overall_score)) +
  geom_histogram(bins = 30, fill = "grey", color = "black") +
  labs(title = "Distribution of Overall Score", x = "Overall Score", y = "Frequency")
```




```
# Summary Statistics
summary(df_clean$overall_score)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  44.00   61.75   69.00   67.86   75.00   90.00 
```

Distribution

- **Nearly Symmetrical:** The median (68.5) is close to the mean (67.7), suggesting a roughly symmetrical distribution. This aligns with the bell-shaped appearance of your density plot.
- **Slight Left Skew:** While mostly balanced, the 1st quartile (61.25) is farther from the median than the 3rd quartile (75). This indicates a very slight left skew, meaning there might be a slightly longer tail of lower sleep scores.

Spread

- **Range:** Your scores range from 44 to 90.
- **Interquartile Range (IQR):** The difference between the 3rd quartile (75) and the 1st quartile (61.25) is 13.75. This IQR represents the middle 50% of your sleep scores.

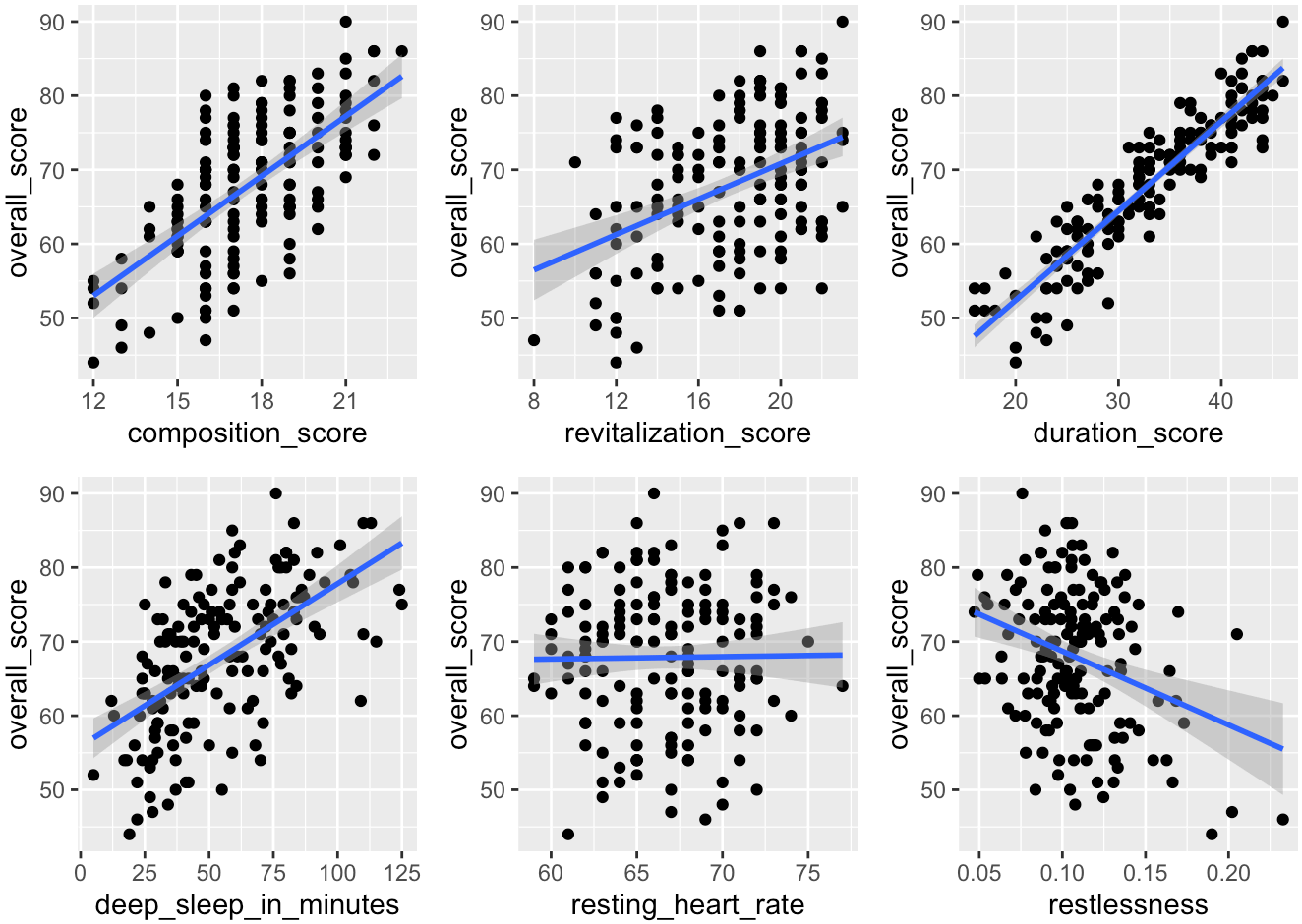
```r
# Shapiro-Wilk test to check the normality of the 'overall_score' distribution
shapiro.test(df_clean$overall_score)
```

```
    Shapiro-Wilk normality test

data:  df_clean$overall_score
W = 0.98713, p-value = 0.1481
```

Data is normal.

```r
# Create individual plots and store them in variables
p1 <- ggplot(df_clean, aes(x = composition_score, y = overall_score)) + geom_point() + ge
p2 <- ggplot(df_clean, aes(x = revitalization_score, y = overall_score)) + geom_point() +
p3 <- ggplot(df_clean, aes(x = duration_score, y = overall_score)) + geom_point() + geom_
p4 <- ggplot(df_clean, aes(x = deep_sleep_in_minutes, y = overall_score)) + geom_point()
p5 <- ggplot(df_clean, aes(x = resting_heart_rate, y = overall_score)) + geom_point() + g
p6 <- ggplot(df_clean, aes(x = restlessness, y = overall_score)) + geom_point() + geom_sm

# Arrange plots in a grid
grid.arrange(p1, p2, p3, p4, p5, p6, nrow = 2, ncol = 3)
```

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



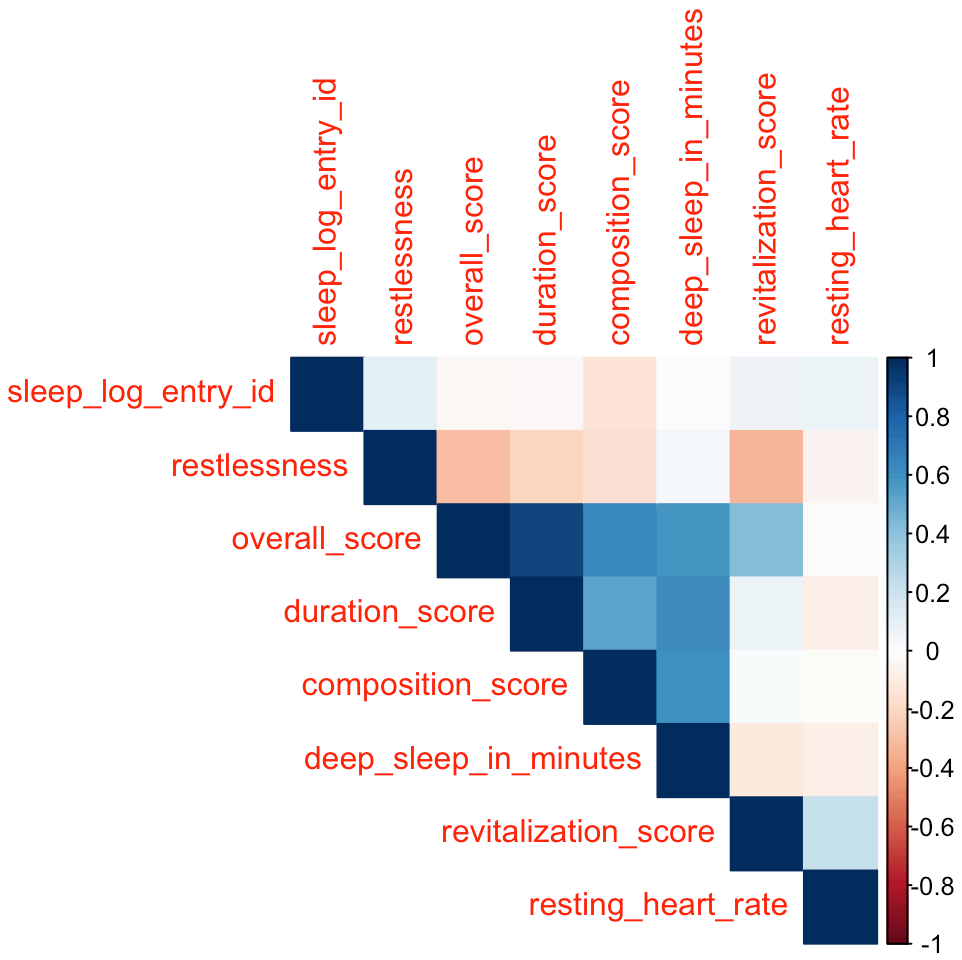

```
# Load the corrplot library
library(corrplot)
```

```
corrplot 0.92 loaded
```

```
# Identify and extract names of columns in df_clean that are numeric
numeric_cols <- names(df_clean)[sapply(df_clean, is.numeric)]
numeric_cols  # This line outputs the names of the numeric columns
```

```
[1] "sleep_log_entry_id"    "overall_score"         "composition_score"
[4] "revitalization_score"  "duration_score"        "deep_sleep_in_minutes"
[7] "resting_heart_rate"    "restlessness"
```

```
# Subset df_clean to only include numeric columns
df_numeric <- df_clean[numeric_cols]

# Compute the correlation matrix for the numeric columns
correlation_matrix <- cor(df_numeric)

# correlation plot from the correlation matrix
corrplot(correlation_matrix, method = "color", type = "upper", order = "hclust")
```



The correlation heatmap shows the relationships between various sleep measures in the data.

Here are some key observations:

Strong Correlations

- **Positive:** The strongest positive correlations seem to be between:
  - `overall_score` and `revitalization_score`
  - `composition_score` and `duration_score`
  - `deep_sleep_in_minutes` and `resting_heart_rate`

Weaker Correlations

- Weaker positive correlations are seen between:
  - `overall_score` and `duration_score`
  - `overall_score` and `composition_score`

Negative Correlations

- A weak negative correlation might be present between `overall_score` and `restlessness`.

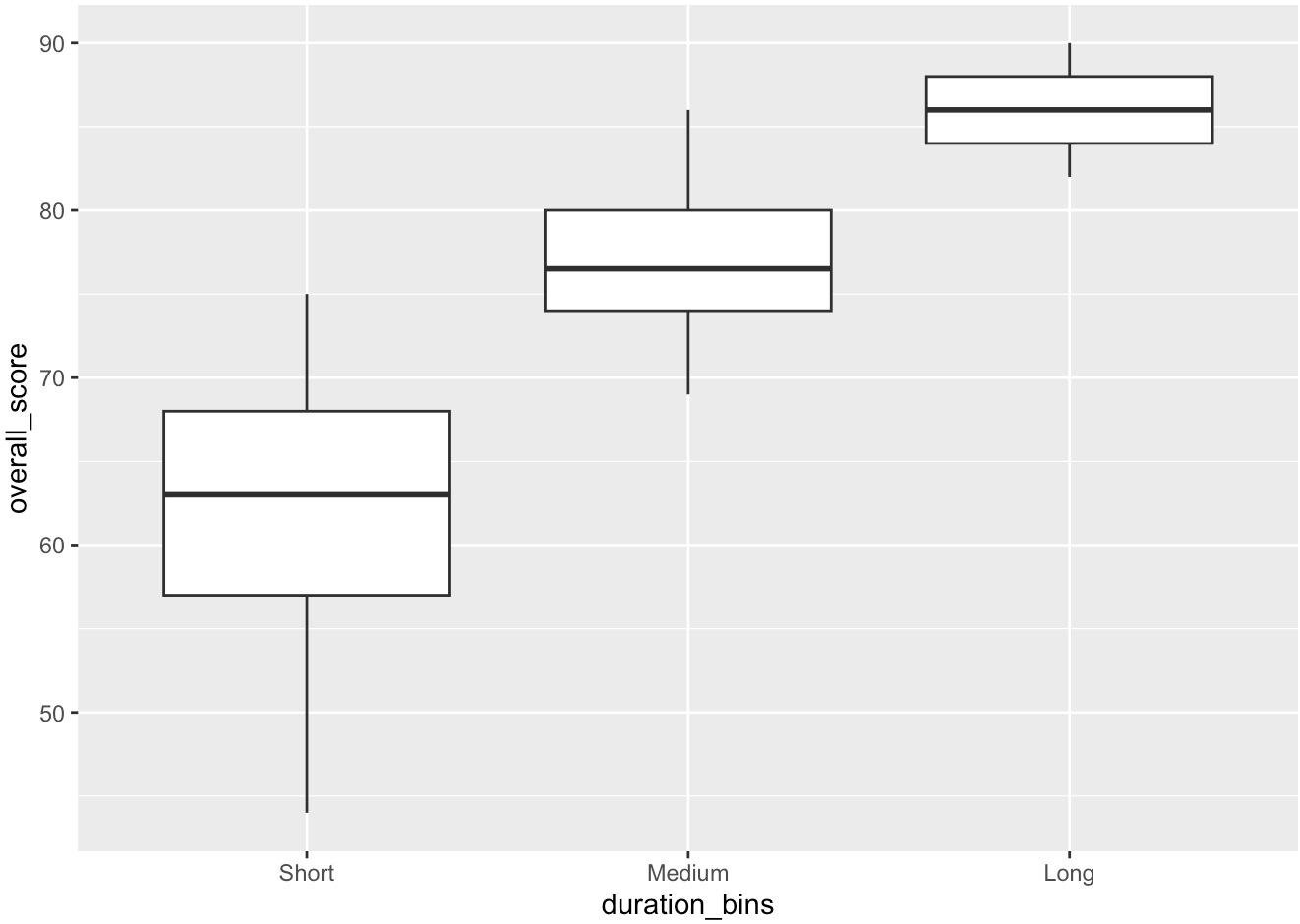
Interpretation

- **Supportive Sleep Factors:** Higher scores in `revitalization_score` and `deep_sleep` seem to be associated with better overall sleep scores. This makes sense as feeling more revitalized and getting more deep sleep are generally positive indicators of restful sleep.
- **Sleep Duration:** The positive correlation between `overall_score` and `duration_score` is weak, and the heatmap suggests a possible non-linearity. It might be that sufficient sleep duration is beneficial, but too much sleep could have the opposite effect.
- **Restlessness:** There's a weak negative correlation between `overall_score` and `restlessness`, indicating that less restlessness is associated with better sleep scores.

```r
# Create duration bins
df_clean$duration_bins <- cut(df_clean$duration_score,
                              breaks = c(0, 35, 45, 60, Inf),
                              labels = c("Short", "Medium", "Long", "Very Long"))

# Calculate mean overall score per bin
df_clean %>%
  group_by(duration_bins) %>%
  summarize(mean_overall_score = mean(overall_score))
```

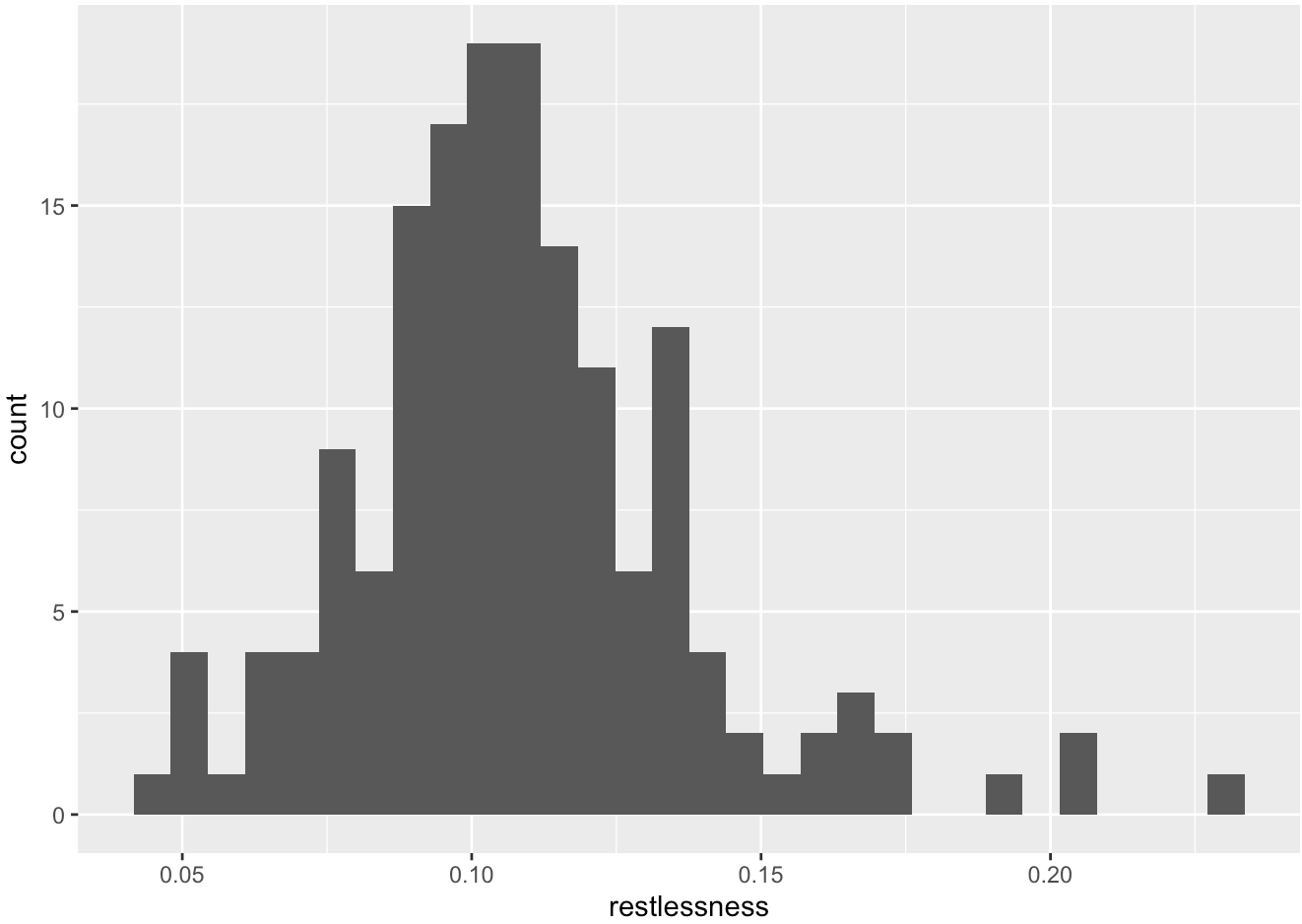
```
# A tibble: 3 × 2
  duration_bins mean_overall_score
  <fct>                      <dbl>
1 Short                       62.3
2 Medium                      76.9
3 Long                        86
```

```r
# Visualization
ggplot(df_clean, aes(x = duration_bins, y = overall_score)) +
  geom_boxplot()
```

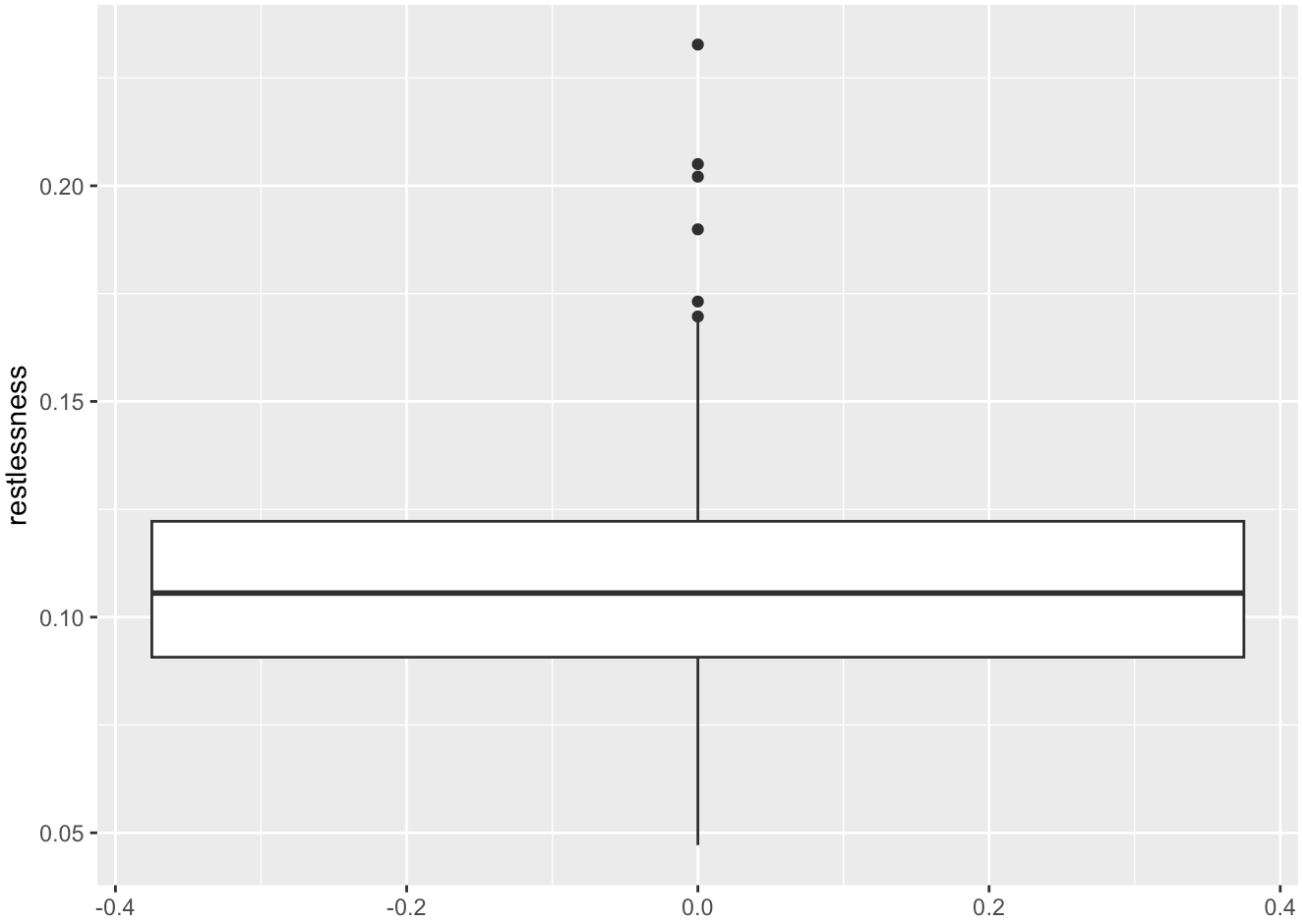



```
# Histogram of restlessness
ggplot(df_clean, aes(x = restlessness)) +
  geom_histogram()
```

```
`stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```




```
# Box plot of restlessness
ggplot(df_clean, aes(y = restlessness)) +
  geom_boxplot()
```




```
# Correlation with other features
cor(df_clean[ ,c("restlessness", "deep_sleep_in_minutes","composition_score","revitalizat
```

```
                    restlessness deep_sleep_in_minutes composition_score
restlessness          1.00000000            0.04822818       -0.15137518
deep_sleep_in_minutes 0.04822818            1.00000000        0.60013700
composition_score    -0.15137518            0.60013700        1.00000000
revitalization_score -0.32796536           -0.10104026        0.02075388
duration_score       -0.20409816            0.62457287        0.52710277
resting_heart_rate   -0.05637749           -0.07392913       -0.01301794
                    revitalization_score duration_score resting_heart_rate
restlessness                 -0.32796536    -0.20409816        -0.05637749
deep_sleep_in_minutes        -0.10104026     0.62457287        -0.07392913
composition_score             0.02075388     0.52710277        -0.01301794
revitalization_score          1.00000000     0.08460118         0.22940177
duration_score                0.08460118     1.00000000        -0.08615985
resting_heart_rate            0.22940177    -0.08615985         1.00000000
```

70+ is considered good, so create a binary variable for evaluation

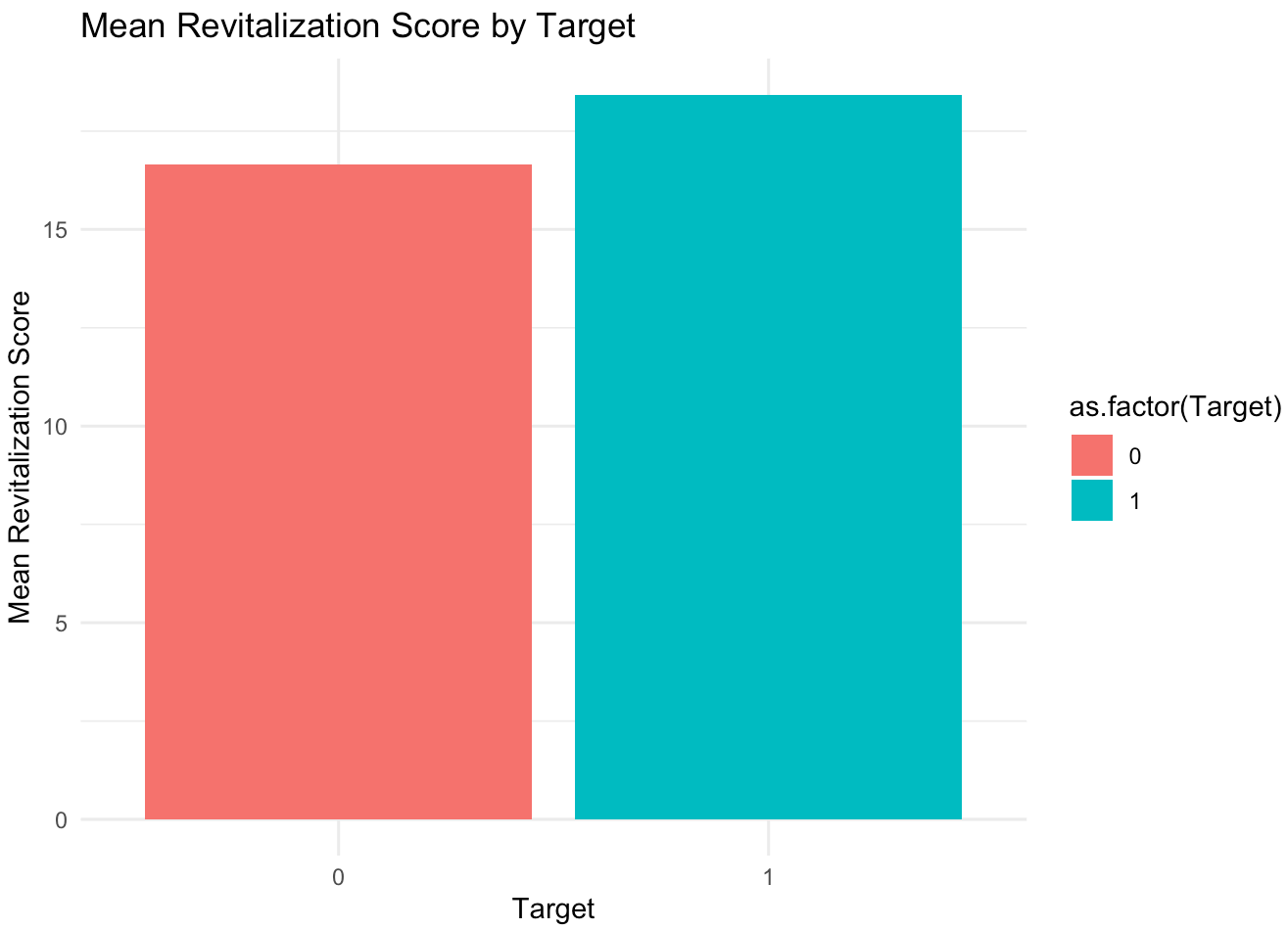
```
# A new binary column 'Target' where 1 indicates 'overall_score' is 70 or more, and 0 oth
df_clean <- df_clean %>%
  mutate(Target = ifelse(overall_score >= 70, 1, 0))
```

```r
# Calculate means for various metrics grouped by 'Evaluation'
metrics_means <- df_clean %>%
  group_by(Target) %>%
  summarise(
    Mean_Composition_Score = mean(composition_score, na.rm = TRUE),
    Mean_Revitalization_Score = mean(revitalization_score, na.rm = TRUE),
    Mean_Duration_Score = mean(duration_score, na.rm = TRUE),
    Mean_Deep_Sleep_Minutes = mean(deep_sleep_in_minutes, na.rm = TRUE),
    Mean_Resting_Heart_Rate = mean(resting_heart_rate, na.rm = TRUE),
    Mean_Restlessness = mean(restlessness, na.rm = TRUE)
  )
```

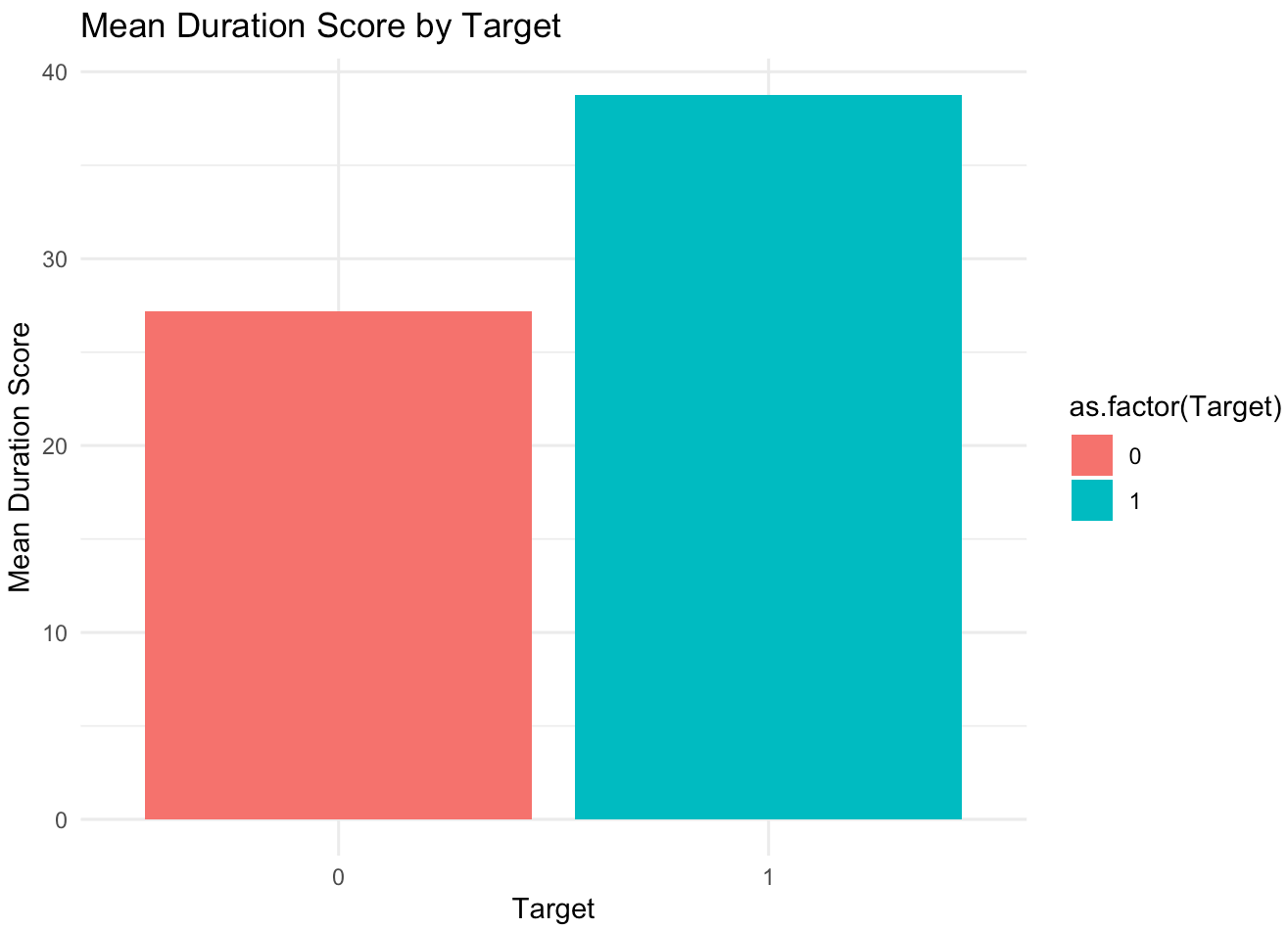
These following plots visually represent the mean values of various metrics categorized by the 'Target' variable, using bar graphs for clear, direct comparison. Each plot focuses on a different metric, such as Composition Score, Revitalization Score, Duration Score, Deep Sleep Minutes, Resting Heart Rate, and Restlessness, showing how each metric's average value differs across different target categories. This visualization helps to easily identify trends, patterns, and outliers in the data across different groups defined by the 'Target', facilitating a straightforward interpretation of how these metrics behave relative to each target category.

```r
library(ggplot2)

# Plotting the results
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Composition_Score, fill = as.fa
  geom_bar(stat = "identity") +
  labs(title = "Mean Composition Score by Target", x = "Target", y = "Mean Composition Sc
  theme_minimal()
```

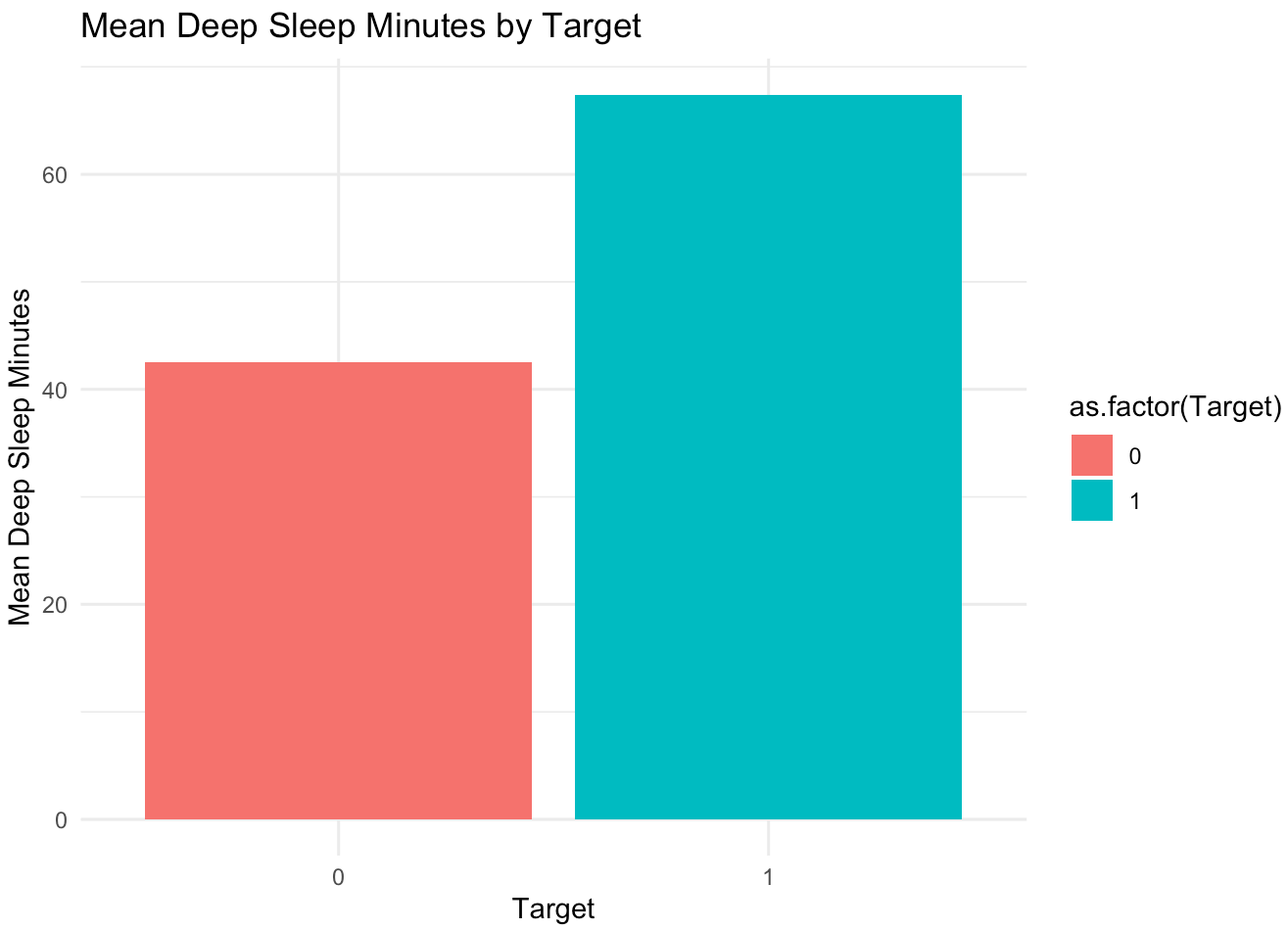

```
# Plot for Mean Revitalization Score
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Revitalization_Score, fill = as
  geom_bar(stat = "identity") +
  labs(title = "Mean Revitalization Score by Target", x = "Target", y = "Mean Revitalizat
  theme_minimal()
```



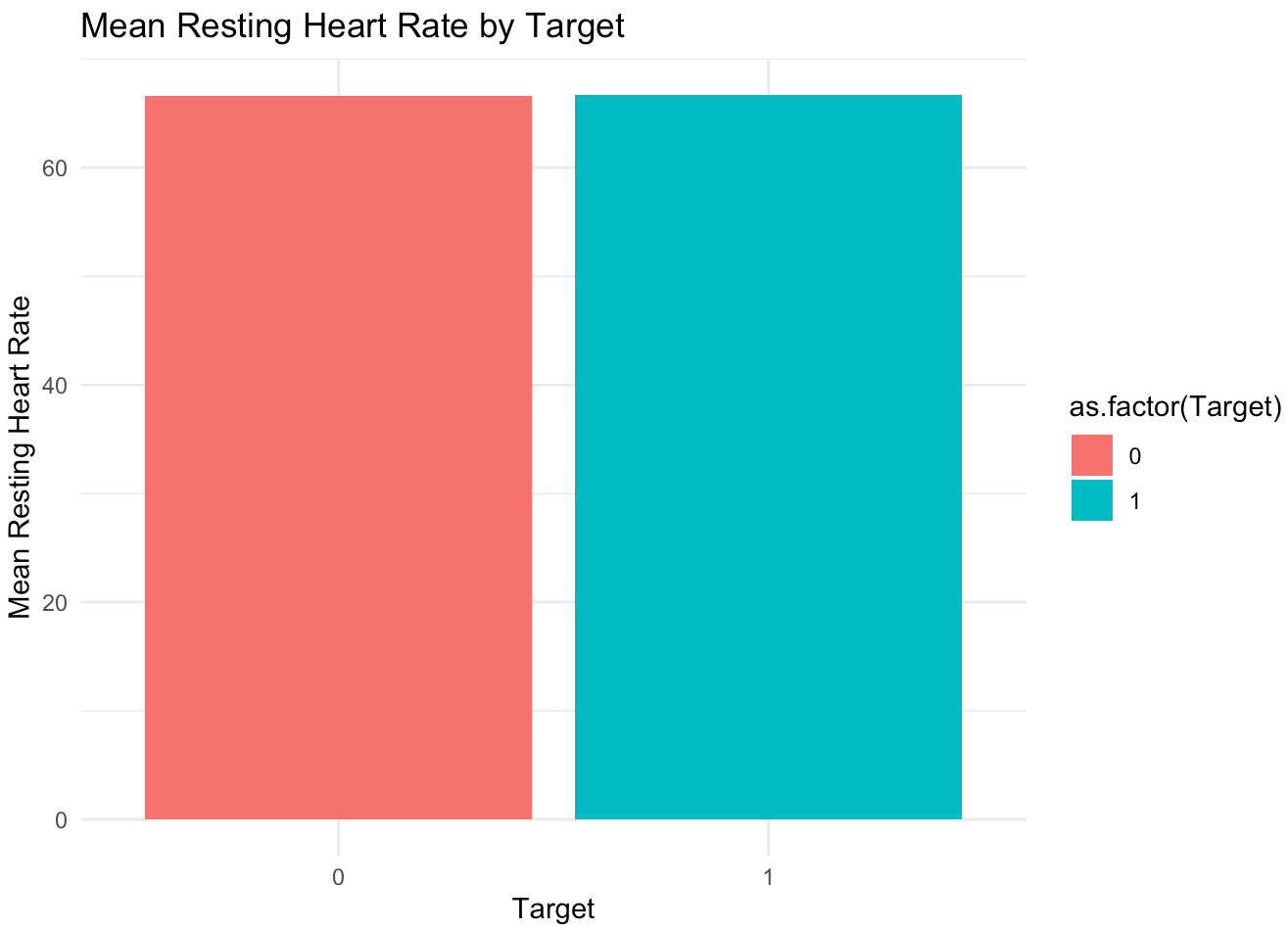

```
# Plot for Mean Duration Score
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Duration_Score, fill = as.facto
  geom_bar(stat = "identity") +
  labs(title = "Mean Duration Score by Target", x = "Target", y = "Mean Duration Score")
  theme_minimal()
```



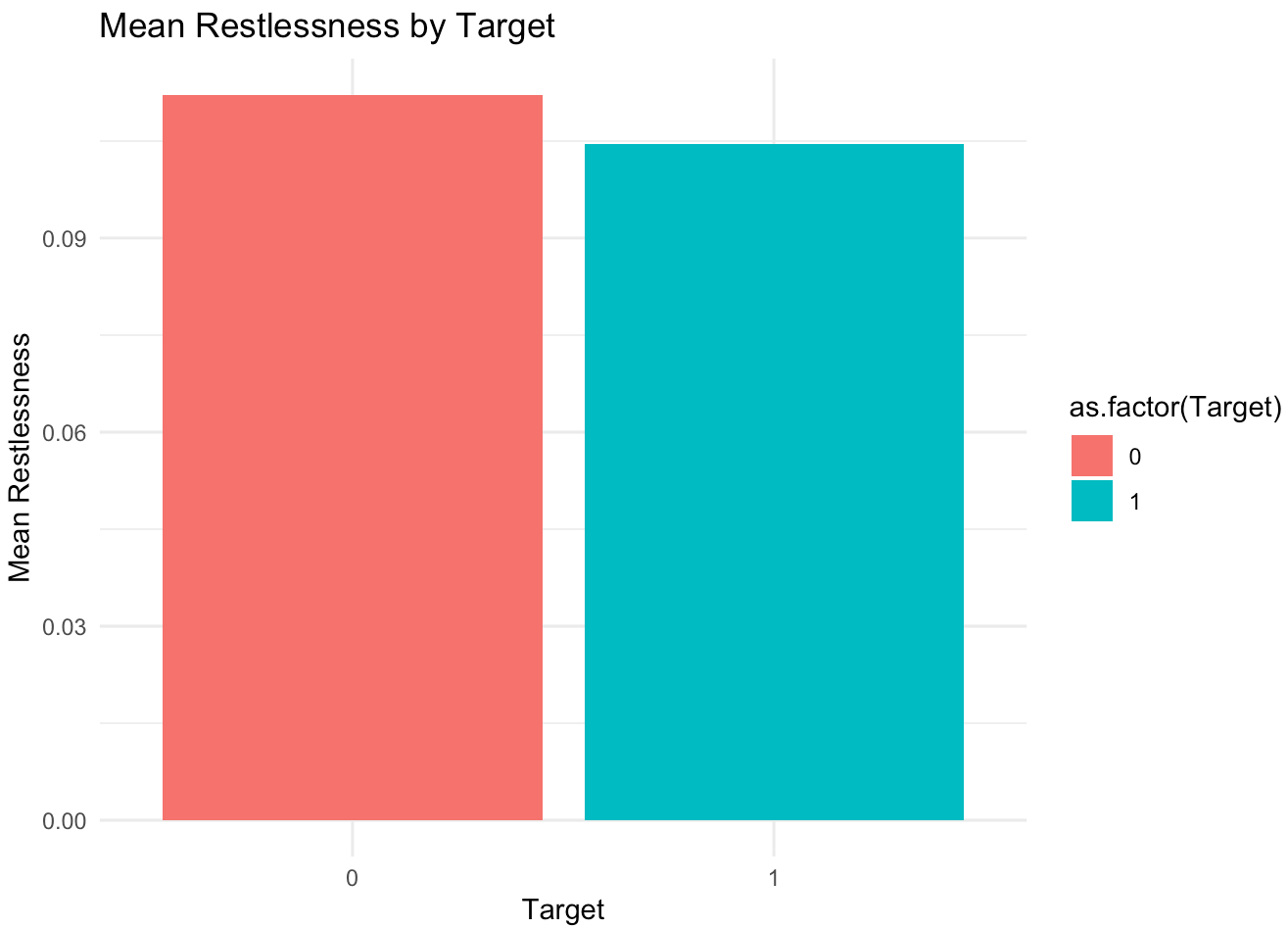

```
# Plot for Mean Deep Sleep Minutes
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Deep_Sleep_Minutes, fill = as.f
  geom_bar(stat = "identity") +
  labs(title = "Mean Deep Sleep Minutes by Target", x = "Target", y = "Mean Deep Sleep Mi
  theme_minimal()
```




```
# Plot for Mean Resting Heart Rate
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Resting_Heart_Rate, fill = as.f
  geom_bar(stat = "identity") +
  labs(title = "Mean Resting Heart Rate by Target", x = "Target", y = "Mean Resting Heart
  theme_minimal()
```




```r
# Plot for Mean Restlessness
ggplot(metrics_means, aes(x = as.factor(Target), y = Mean_Restlessness, fill = as.factor(
  geom_bar(stat = "identity") +
  labs(title = "Mean Restlessness by Target", x = "Target", y = "Mean Restlessness") +
  theme_minimal()
```



MODELING

```
# Load the necessary library for data partitioning
library(caret)

# Define a vector of names for the independent (predictor) variables
independent_vars <- c('composition_score', 'revitalization_score', 'duration_score', 'dee

# Specify the dependent (response) variable
dependent_var <- "overall_score"

# Remove rows with missing values in specified columns to ensure data quality for modelin
df_clean <- na.omit(df_clean[, c(independent_vars, dependent_var)])

# Create a new binary column 'target' in df_clean where 1 indicates 'overall_score' is 70
# This binary column is typically used for classification purposes
df_clean$target <- ifelse(df_clean$overall_score >= 70, 1, 0)

# Set a seed for random number generation to ensure reproducibility in data splitting
set.seed(123)

# Use caret's createDataPartition function to create a split index for 75% training data
# This ensures that approximately 75% of the data is used for training the model
```

```r
split_index <- createDataPartition(df_clean$overall_score, p = 0.75, list = FALSE)

# Subset df_clean to create a training dataset using the split index
train <- df_clean[split_index, ]

# Subset df_clean to create a testing dataset using the rows not included in the split in
test <- df_clean[-split_index, ]
```

Linear Regression

```r
library(tidymodels)
```

```
── Attaching packages ─────────────────────────────────────── tidymodels 1.2.0 ──

✔ broom        1.0.5      ✔ rsample      1.2.1
✔ dials        1.2.1      ✔ tune         1.2.0
✔ infer        1.0.6      ✔ workflows    1.1.4
✔ modeldata    1.3.0      ✔ workflowsets 1.1.0
✔ parsnip      1.2.1      ✔ yardstick    1.3.1
✔ recipes      1.0.10

── Conflicts ────────────────────────────────────────── tidymodels_conflicts() ──
✖ gridExtra::combine()     masks dplyr::combine()
✖ scales::discard()        masks purrr::discard()
✖ dplyr::filter()          masks stats::filter()
✖ recipes::fixed()         masks stringr::fixed()
✖ dplyr::lag()             masks stats::lag()
✖ caret::lift()            masks purrr::lift()
✖ yardstick::precision()   masks caret::precision()
✖ yardstick::recall()      masks caret::recall()
✖ yardstick::sensitivity() masks caret::sensitivity()
✖ yardstick::spec()        masks readr::spec()
✖ yardstick::specificity() masks caret::specificity()
✖ recipes::step()          masks stats::step()
• Search for functions across packages at https://www.tidymodels.org/find/
```

```r
# Define the model specification
linear_spec <- linear_reg() |>
  set_engine("lm")

# Define the recipe
linear_recipe <- recipe(overall_score ~ ., data = train) |>
  step_normalize(all_predictors())

# Create the workflow
linear_workflow <- workflow() |>
  add_model(linear_spec) |>
  add_recipe(linear_recipe)
```

```r
# Fit the model
linear_fit <- fit(linear_workflow, data = train)

# Make predictions
linear_predictions <- predict(linear_fit, new_data = test) |>
  bind_cols(test)

# Calculate metrics
linear_metrics <- linear_predictions |>
  metrics(truth = overall_score, estimate = .pred)
linear_metrics
```

```
# A tibble: 3 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard    1.56e-14
2 rsq     standard    1   e+ 0
3 mae     standard    1.53e-14
```

Ridge Regression

```r
# Define the model specification for Ridge Regression
ridge_spec <- linear_reg(penalty = 0.1, mixture = 0) %>%
  set_engine("glmnet")

# Define the recipe
ridge_recipe <- recipe(overall_score ~ ., data = train) %>%
  step_normalize(all_predictors())

# Create the workflow
ridge_workflow <- workflow() %>%
  add_model(ridge_spec) %>%
  add_recipe(ridge_recipe)

# Fit the model
ridge_fit <- fit(ridge_workflow, data = train)

# Make predictions
ridge_predictions <- predict(ridge_fit, new_data = test) %>%
  bind_cols(test)

# Calculate metrics
ridge_metrics <- ridge_predictions %>%
  metrics(truth = overall_score, estimate = .pred)
ridge_metrics
```

```
# A tibble: 3 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       0.923
2 rsq     standard       0.994
3 mae     standard       0.750
```

Lasso Regression

```
library(tidymodels)
library(glmnet)
```

```
Loading required package: Matrix


Attaching package: 'Matrix'

The following objects are masked from 'package:tidyr':

    expand, pack, unpack

Loaded glmnet 4.1-8
```

```
# Define the model specification for Lasso with tuning
lasso_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

# Define the recipe
lasso_recipe <- recipe(target ~ ., data = train) %>%
  step_normalize(all_predictors())

# Create the workflow
lasso_workflow <- workflow() %>%
  add_model(lasso_spec) %>%
  add_recipe(lasso_recipe)

# Define a grid of hyperparameters
lasso_grid <- grid_regular(
  penalty(range = c(-3, 0)),
  levels = 20
)


# Fit the model using cross-validation
cv_folds <- vfold_cv(train, v = 5)
lasso_fit <- tune_grid(
  lasso_workflow,
  resamples = cv_folds,
```

```
  grid = lasso_grid
)
```

```
→ A | warning: A correlation computation is required, but `estimate` is constant and has
0
                standard deviation, resulting in a divide by 0 error. `NA` will be
returned.

There were issues with some computations   A: x1

There were issues with some computations   A: x5
```

```
# Collect metrics
lasso_results <- collect_metrics(lasso_fit)

# Select the best settings using RMSE
best_lasso <- lasso_results %>%
   filter(.metric == "RMSE") %>%
   arrange(mean) %>%
   head(1)

# Refit the model with the best penalty
final_lasso <- finalize_workflow(
  lasso_workflow,
  select_best(lasso_fit, metric = "rmse")
)

# Fit the final model
final_lasso_fit <- fit(final_lasso, data = train)

# Make predictions
lasso_predictions <- predict(final_lasso_fit, new_data = test) %>%
  bind_cols(test)

# Calculate metrics
lasso_metrics <- lasso_predictions %>%
  metrics(truth = target, estimate = .pred)
lasso_metrics
```

```
# A tibble: 3 × 3
  .metric .estimator .estimate
  <chr>   <chr>          <dbl>
1 rmse    standard       0.277
2 rsq     standard       0.705
3 mae     standard       0.236
```

Decision Tree

```r
# Load necessary libraries
library(tidymodels)

# Convert the target variable to a factor
train$target <- factor(train$target)


# Decision Tree Workflow
tree_spec <- decision_tree() %>%
  set_engine("rpart") %>%
  set_mode("classification")

tree_recipe <- recipe(target ~ ., data = train)

tree_workflow <- workflow() %>%
  add_model(tree_spec) %>%
  add_recipe(tree_recipe)

# 5. Fit and Predict
tree_fit <- fit(tree_workflow, data = train)
tree_predictions <- predict(tree_fit, new_data = test) %>%
  bind_cols(data.frame(actual = test$target))

# Ensure correct column names!
tree_predictions$actual <- factor(tree_predictions$actual, levels = c("0", "1"))


#Confusion Matrix
conf_matrix <- confusionMatrix(data = tree_predictions$.pred_class,
                               reference = tree_predictions$actual)
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 20  0
         1  0 18

               Accuracy : 1
                 95% CI : (0.9075, 1)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : 2.555e-11

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
```

```
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5263
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

SVM

```
library(e1071)
```

```
Attaching package: 'e1071'

The following object is masked from 'package:tune':

    tune

The following object is masked from 'package:rsample':

    permutations

The following object is masked from 'package:parsnip':

    tune
```

```
library(caret)

# Prepare the data
train$target <- as.factor(train$target)
test$target <- as.factor(test$target)

# Fit the SVM model
svm_model <- svm(target ~ ., data = train,
                 kernel = "linear",
                 cost = 1,
                 scale = FALSE)

# Make predictions
svm_predictions <- predict(svm_model, newdata = test)

# Convert to factors (for confusion matrix)
svm_predictions_factor <- as.factor(svm_predictions)
test_target_factor <- as.factor(test$target)

conf_matrix <- confusionMatrix(svm_predictions_factor, test_target_factor)
```

```r
accuracy_conf_matrix <- conf_matrix$overall['Accuracy']
print(accuracy_conf_matrix)
```

```
Accuracy
       1
```

```r
print(conf_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 20  0
         1  0 18

               Accuracy : 1
                 95% CI : (0.9075, 1)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : 2.555e-11

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.5263
         Detection Rate : 0.5263
   Detection Prevalence : 0.5263
      Balanced Accuracy : 1.0000

       'Positive' Class : 0
```

```r
# Save predictions in a dataframe
svm_predictions <- data.frame(
    actual = test$target,
    predicted = svm_predictions_factor
)
```

NaiveBayes

```r
# Fit the Naive Bayes model
nb_model <- naiveBayes(target ~ ., data = train)

# Make predictions on the test set
```

```r
nb_predictions <- predict(nb_model, newdata = test)

# Store predictions and actual values in a dataframe
nb_predictions <- data.frame(
    actual = test$target,
    predicted = as.factor(nb_predictions)
)

# Confusion matrix and overall accuracy
nb_confusion_matrix <- confusionMatrix(nb_predictions$predicted, nb_predictions$actual)
nb_accuracy <- nb_confusion_matrix$overall['Accuracy']
print(nb_accuracy)
```

```
 Accuracy
0.9736842
```

```r
# Additional metrics (optional)
print(nb_confusion_matrix)
```

```
Confusion Matrix and Statistics

          Reference
Prediction  0  1
         0 19  0
         1  1 18

               Accuracy : 0.9737
                 95% CI : (0.8619, 0.9993)
    No Information Rate : 0.5263
    P-Value [Acc > NIR] : 8.993e-10

                  Kappa : 0.9474

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9500
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9474
             Prevalence : 0.5263
         Detection Rate : 0.5000
   Detection Prevalence : 0.5000
      Balanced Accuracy : 0.9750

       'Positive' Class : 0
```

MODEL SELECTION

To determine the best model, we need to examine the metrics from each model. Let's review the key metrics across the different modeling approaches:

1. **Linear Regression**

- **RMSE:** ~0 (1.563534e-14)
- **R^2:** 1.000000e+00
- **MAE:** ~0 (1.533276e-14)

2. **Ridge Regression**

- **RMSE:** 0.9228239
- **R^2:** 0.9938190
- **MAE:** 0.7498178

3. **Lasso Regression**

- **RMSE:** 0.2769829
- **R^2:** 0.7048158
- **MAE:** 0.2362407

4. **Decision Tree (Classification)**

- **Accuracy:** 1
- **Kappa:** 1
- **Sensitivity:** 1.0000
- **Specificity:** 1.0000

5. **SVM (Classification)**

- **Accuracy:** 1
- **Kappa:** 1
- **Sensitivity:** 1.0000
- **Specificity:** 1.0000

6. **Naive Bayes (Classification)**

- **Accuracy:** 0.9737
- **Kappa:** 0.9474
- **Sensitivity:** 0.9500
- **Specificity:** 1.0000

Model Selection Based on Metrics

Here are considerations based on model metrics:

Regression Models

- **Linear Regression** shows a perfect fit with $R^2 = 1$ and virtually zero RMSE and MAE. This suggests a possible overfitting scenario, especially if this perfection persists across various splits of data.
- **Ridge Regression** also shows high performance but not perfect, indicating some regularization effect that may generalize better than a plain linear model.
- **Lasso Regression** shows the worst performance among the regression models in terms of RMSE and R^2, but it might be incorporating more regularization, potentially avoiding overfitting and providing better generalization on unseen data.

Classification Models

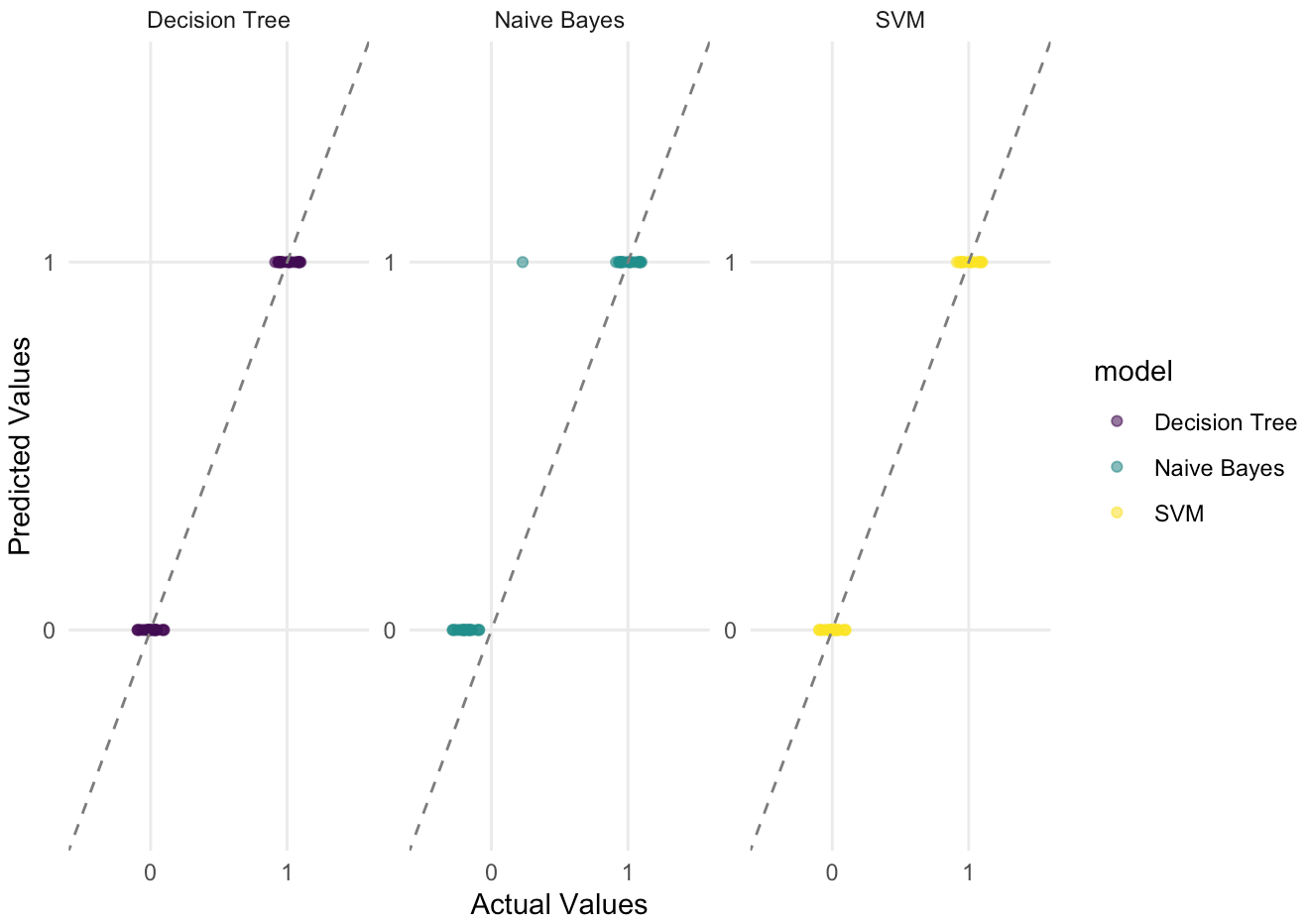
- Both the **Decision Tree** and **SVM** show perfect metrics. This perfection might suggest overfitting, particularly if your dataset is not large or complex enough to justify such results.
- **Naive Bayes** shows slightly less than perfect metrics, which might indicate better generalization than the Decision Tree and SVM.

MODEL ASSESMENT AND INTERPRETATION

```
combined_predictions <- data.frame(
    actual = lasso_predictions$.pred,
    lasso = lasso_predictions$.pred,
    linear = linear_predictions$.pred,
    ridge = ridge_predictions$.pred
)
```

```
# Ensure your prediction dataframes have the same structure and a column for actual value
combined_predictions <- bind_rows(
  svm_predictions %>% mutate(model = "SVM"),  # No renaming needed for SVM
  nb_predictions %>% mutate(model = "Naive Bayes"), # No renaming needed for NB
  tree_predictions %>% rename(predicted = .pred_class) %>% mutate(model = "Decision Tree"
)
```

```
ggplot(combined_predictions, aes(x = actual, y = predicted, color = model)) +
  geom_point(position = position_jitterdodge(jitter.width = 0.2), alpha = 0.6) +  # Jitte
  geom_abline(intercept = 0, slope = 1, linetype = "dashed", color = "gray50") +
  facet_wrap(~ model, scales = "free_y") + # Create panels for each model
  labs(x = "Actual Values", y = "Predicted Values") +
  theme_minimal() +
  scale_color_viridis_d()  # Example of a color-blind friendly palette
```



The plot compares the predicted values from three different classification models (Decision Tree, Naive Bayes, SVM) against the actual values, which are binary (0 or 1). Each model's predictions are represented by colored dots along lines that correspond to the actual values of 0 or 1.

Here's what we can see from the plot: - **Decision Tree**: The model correctly predicts both classes (0 and 1), as shown by the purple dots aligning perfectly with the actual values. - **Naive Bayes**: Similar to the Decision Tree, it also correctly predicts both classes, as shown by the cyan dots. - **SVM**: The model shows correct predictions as well, with the yellow dots aligning with both actual 0 and 1 values.

Overall, all three models appear to perform very well, correctly classifying the actual outcomes without any visible errors in this plot.

```
levels(tree_predictions$.pred_class)
```

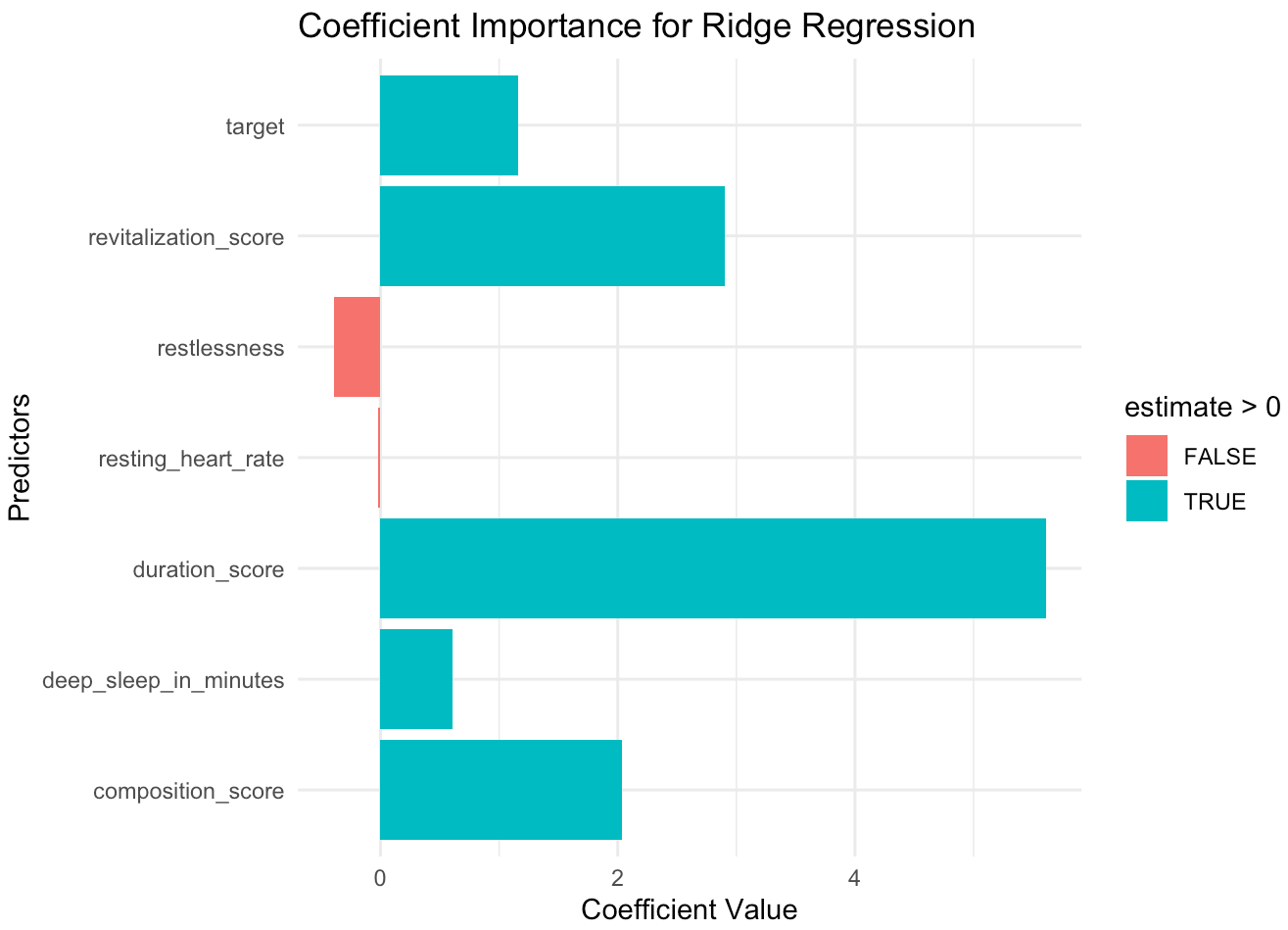
```
[1] "0" "1"
```

```
levels(tree_predictions$actual)
```

```
[1] "0" "1"
```

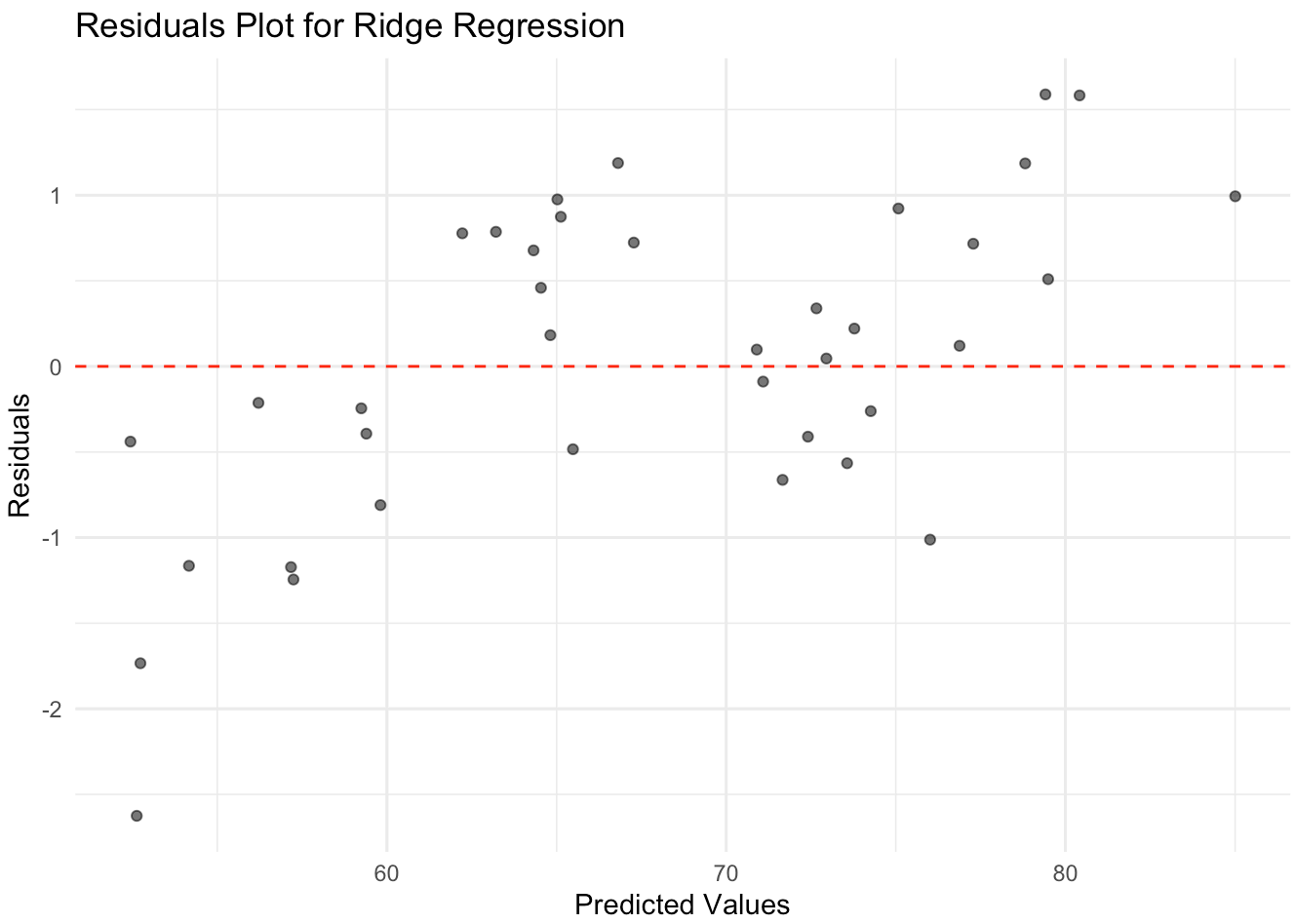
```
# Extract coefficients from the Ridge Regression model (adjust based on your model fittin
ridge_model_fit <- extract_fit_parsnip(ridge_fit) %>%
                   tidy() %>%
```

```
                  filter(term != "(Intercept)")

# Plotting
ggplot(ridge_model_fit, aes(x = term, y = estimate, fill = estimate > 0)) +
    geom_col() +
    coord_flip() +  # flips the axes for easier reading
    labs(x = "Predictors", y = "Coefficient Value") +
    ggtitle("Coefficient Importance for Ridge Regression") +
    theme_minimal()
```



The chart shows the impact of various predictors in a Ridge Regression model. Most predictors positively influence the model's outcome, with "duration_score" having the most substantial positive impact. "Restlessness" negatively affects the outcome, indicating an inverse relationship with the model's response variable.

```
ridge_predictions$residuals <- ridge_predictions$overall_score - ridge_predictions$.pred

ggplot(ridge_predictions, aes(x = .pred, y = residuals)) +
    geom_point(alpha = 0.6) +
    geom_hline(yintercept = 0, linetype = "dashed", color = "red") +
    labs(x = "Predicted Values", y = "Residuals") +
    ggtitle("Residuals Plot for Ridge Regression") +
    theme_minimal()
```



The residuals plot for Ridge Regression displays the residuals (errors) on the y-axis against the predicted values on the x-axis. The plot shows a random scatter of points around the zero line, which is good as it indicates no obvious pattern or systematic error. However, there are a few outliers, particularly one point with a residual near -2, suggesting some predictions are less accurate than others. Overall, the model seems to be performing adequately.

RESULTS COMMUNICATION

Project Overview

This project focused on analyzing sleep data to determine what influences good sleep quality. We worked with a dataset that included several sleep-related metrics such as composition score, revitalization score, duration score, deep sleep minutes, resting heart rate, and restlessness. The main goal was to classify sleep quality as either good or poor based on these factors.

Challenges Faced

A significant challenge was the small size of the dataset, which only had 160 entries. This limitation made it difficult to build effective models, leading to issues like overfitting across different machine

learning techniques, including logistic regression, decision trees, SVMs, and methods like random forests and XGBoost.

Issues with Overfitting

Overfitting was a major issue throughout our analysis. Despite using methods like cross-validation and regularization, and pruning decision trees, the models often showed very high accuracy. This wasn't due to the models being highly effective but because they were memorizing the data instead of learning to predict new data accurately.

Limitations of the Data

The small number of data points meant that models often learned from the noise in the data rather than true underlying patterns. This problem was made worse by the dataset having many variables compared to the number of data points. Even after we removed variables that were closely related to each other to make the models simpler, overfitting was still a problem.

Final Thoughts

The extremely high accuracies we saw in our models are misleading and shouldn't be trusted to predict new, unseen data accurately. This project showed how important it is to have a large and varied dataset for machine learning projects. Without enough data, even the most advanced models can fail to give useful results.