# Predicting Rental Bike Count Summary

Derex Wangmang

## Contents



### Introduction

My project focuses on a regression analysis of the number of bikes rented in Seoul, South Korea. As part of my project, I retrieved and processed the dataset, *Seoul Bike Sharing Demand Data Set*, from the University of California at Irvine's Machine Learning Repository, found here.

My data processing led to 8465 observations with 10 predictor variables and 1 target variable: the rental bike count. Using these factors, I sought to predict the number of rental bikes at a given time. In the future, this analysis can be used by cities or companies to accurately assess demand for rental bikes and provide them where they are needed the most.

### Exploratory Data Analysis

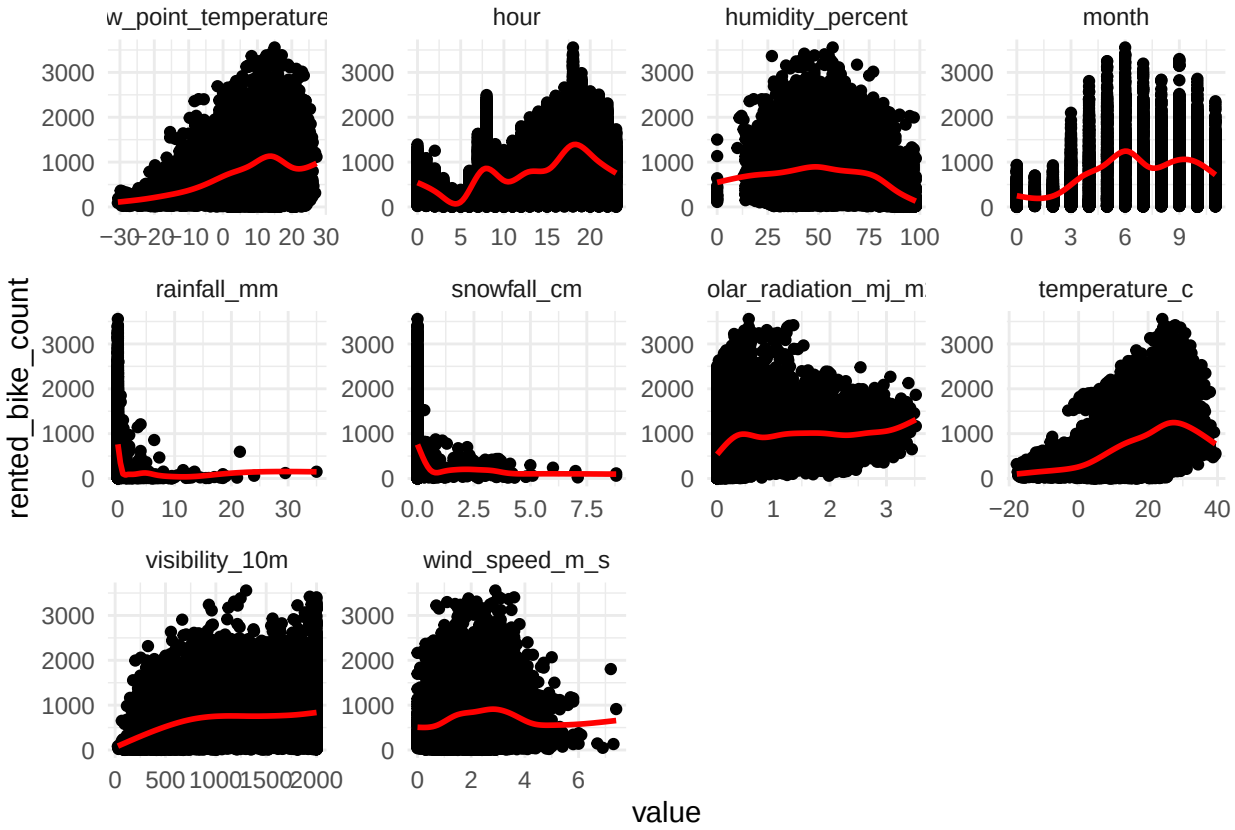
#### Packages

```
library(tidyverse)
library(ggcorrplot)

load("data/processed/intermediate_seoul_bike_data.rda")
```

#### Plots

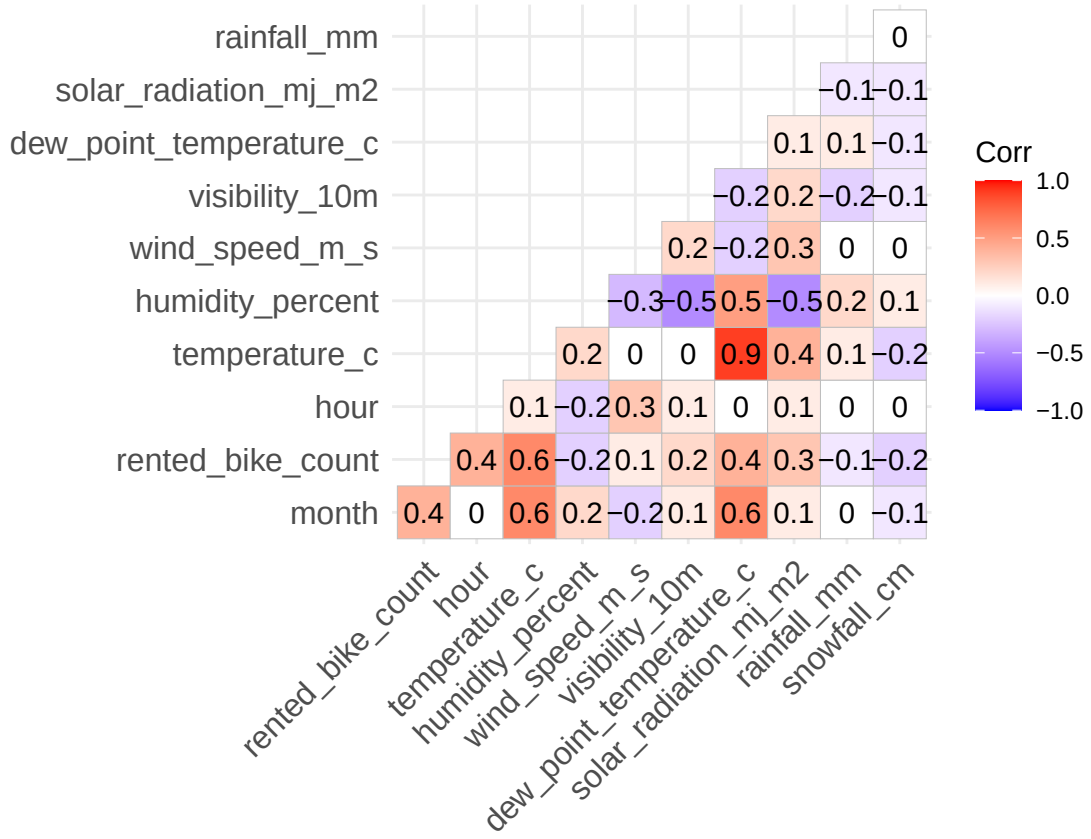
```
bike %>%
  select(where(is.numeric)) %>%
  gather(-rented_bike_count, key = "var", value = "value") %>%
  ggplot(aes(x = value, y = rented_bike_count)) +
  geom_point() +
  geom_smooth(se = F, color = "red") +
  facet_wrap(~ var, scales = "free") +
  theme_minimal()
```




Selecting all the numeric variables and plotting the target variable against them provides an overview of the distributions in the dataset.

I then examined the variables for collinearity. Collinearity presents an issue, as it indicates that predictor variables may have strong linear relationships, cause such variables to no longer be independent. Thus, small changes in the data can have large effects on the model.

```
corr <- round(cor(bike %>% select(where(is.numeric))), 1)
ggcorrplot(corr, type = "lower", lab = T)
```




The Pearson Correlation plot indicated that there are two features strongly correlated with each other: the dew point temperature and the temperature.

```
fit <- lm(rented_bike_count ~ ., bike)
car::vif(fit)
```

```
##                    month                     hour          temperature_c
##                 1.665685                 1.181179              88.295453
##        humidity_percent          wind_speed_m_s          visibility_10m
##               20.205397                 1.295215                1.640993
## dew_point_temperature_c   solar_radiation_mj_m2             rainfall_mm
##              116.471474                 2.012877                1.084470
##             snowfall_cm                  holiday
##                1.097139                 1.008544
```

The collinearity is confirmed by the variance inflation factor.

```
fit2 <- lm(rented_bike_count ~ . - dew_point_temperature_c, bike)
car::vif(fit2)
```

```
##                    month                     hour          temperature_c
##                 1.665589                 1.178699                2.166928
##        humidity_percent          wind_speed_m_s          visibility_10m
##                 2.562210                 1.293253                1.629828
## solar_radiation_mj_m2             rainfall_mm             snowfall_cm
##                 1.912811                 1.071168                1.092256
##                  holiday
##                 1.008526
```

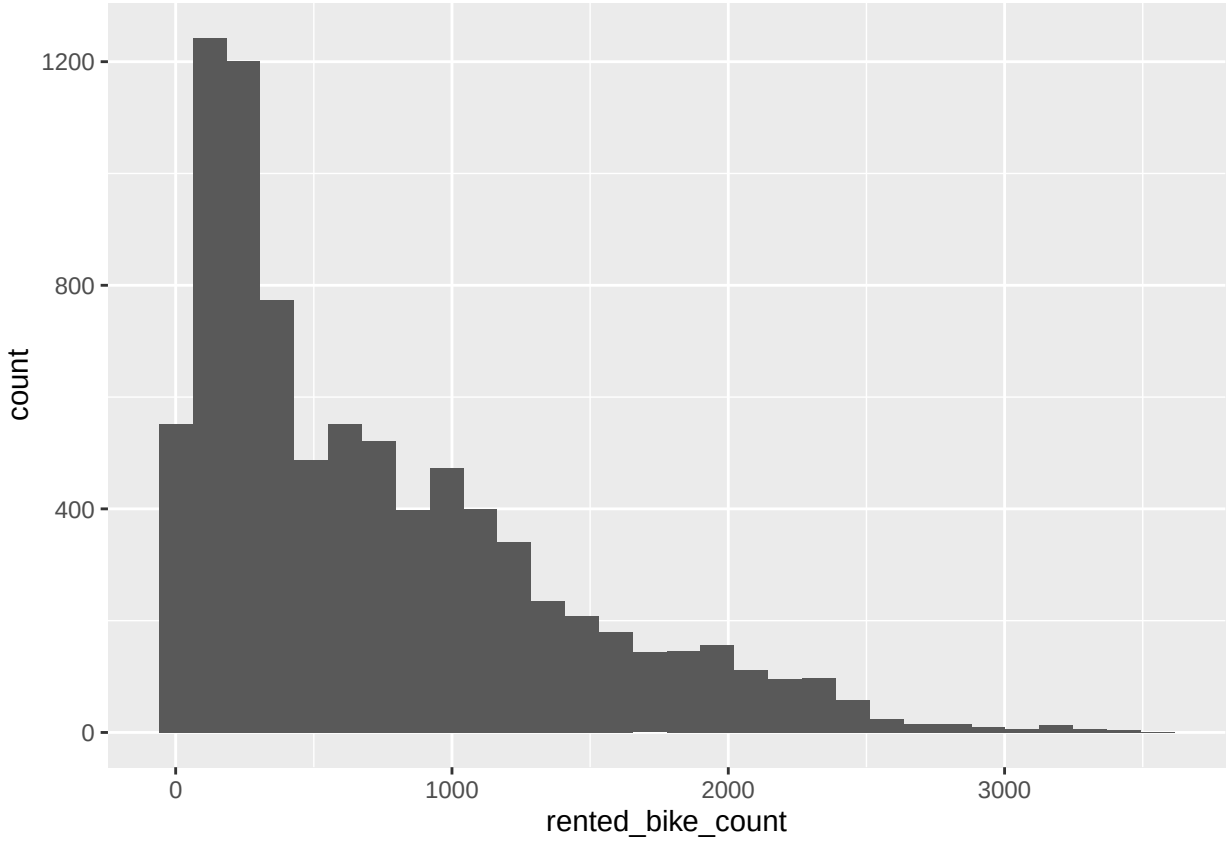
Removing `dew_point_temperature_c` led to a large reduction in the variance inflation factor.

```
bike <- bike %>%
  select(-dew_point_temperature_c)
save(bike, file = "data/processed/seoul_bike_data.rda")
```

Thus, I removed the variable from the processed dataset and saved the updated dataset for future use.

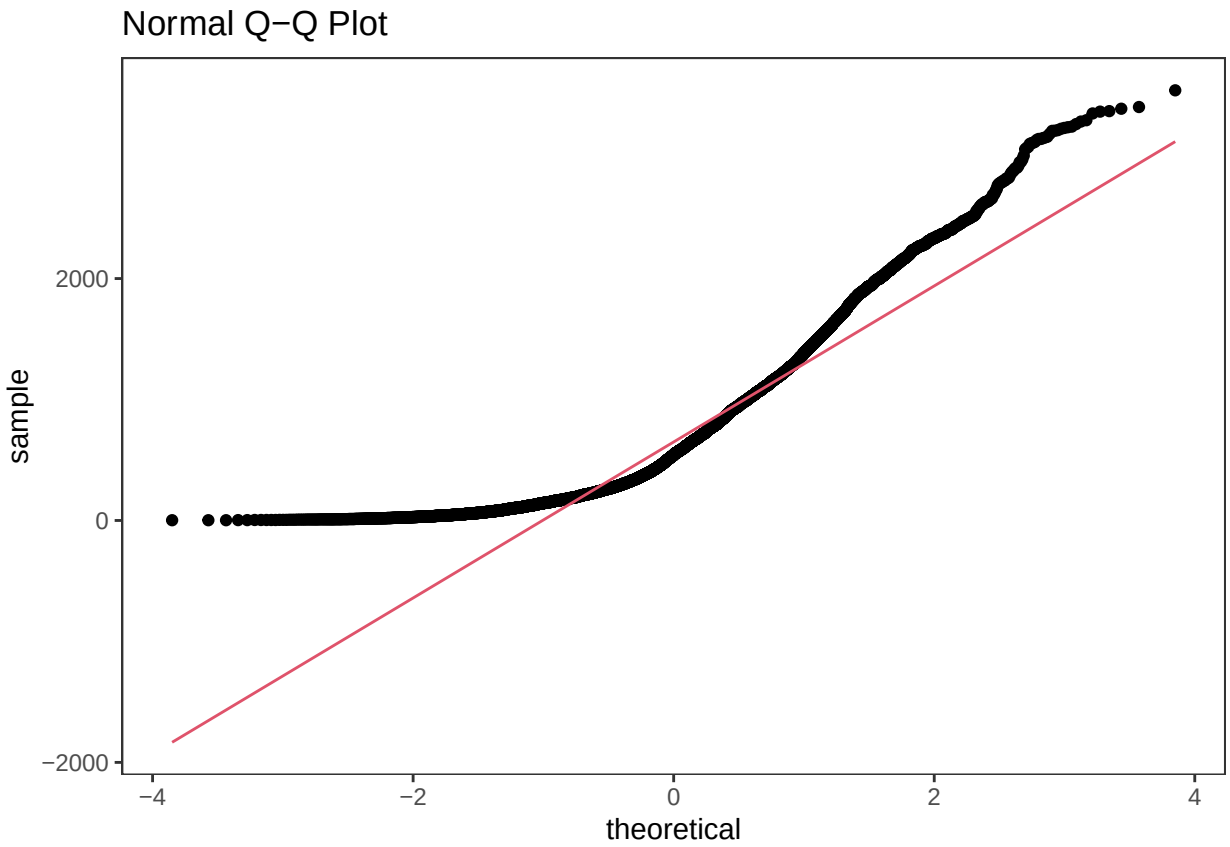
### Rental Bike Count Distribution

```
ggplot(bike, aes(rented_bike_count)) +
  geom_histogram()
```




The rented bike count has a unimodal distribution with a center around 100 and a right tail.

```
ggplot(bike, aes(sample = rented_bike_count)) +
  stat_qq() +
  stat_qq_line(color = 2) +
  labs(title = "Normal Q-Q Plot") +
  theme_bw() +
  theme(panel.grid = element_blank())
```




This Q-Q plot confirms that the distribution is non-normal. The red line represents the normal line.

## Data Modeling

### Packages, Seed, and Data

```
library(tidyverse)
library(tidymodels)

set.seed(7985)

load("data/processed/seoul_bike_data.rda")
```

I load the required packages, set the seed, and load the processed data.

### Data Splitting

```
bike_split <- initial_split(bike, prop = 0.7, strata = rented_bike_count)
bike_train <- training(bike_split)
bike_test <- testing(bike_split)
```

I split the data, reserving 70% of the data for training and 30% of the data for testing.

### Data Folds

```
bike_folds <- vfold_cv(bike_train, v = 10, repeats = 5)
```

I utilize k-fold repeated cross-validation, setting the number of folds to be 10 and repeating 5 times. Cross-validation splits the data into groups, trains the model on k-1 folds, and measures its performance on the last fold. This process is iterated until the last fold. In this case, the data uses 90% of the training set to train, then 10% of the training set to measure performance. The performance is then averaged, leading to an estimate of its performance on unseen data, without using up the testing data. Repeated cross-validation repeats the process a number of times. This technique produces performance metrics and allows comparisons between different models, so that later a model can be selected based on the best metric.

**Models**

```
bike_recipe_non_hot <- recipe(rented_bike_count ~ ., bike_train) %>%
  step_dummy(all_nominal()) %>%
  step_normalize(all_predictors())

bike_recipe <- recipe(rented_bike_count ~ ., bike_train) %>%
  step_dummy(all_nominal()) %>%
  step_normalize(all_predictors())
```

**Recipes** I developed two models, one without one hot encoding and one with it. The recipe without one hot encoding is used for linear regression models, whereas the recipe with one hot encoding is used for the tree-based models and the nearest neighbor model.

```
lm_fit <- read_rds("output/modeling/lm_fit.Rds")
ridge_tuned <- read_rds("output/modeling/ridge_tuned.Rds")
lasso_tuned <- read_rds("output/modeling/lasso_tuned.Rds")
elastic_tuned <- read_rds("output/modeling/elastic_tuned.Rds")
rf_tuned <- read_rds("output/modeling/rf_tuned.Rds")
bt_tuned <- read_rds("output/modeling/bt_tuned.Rds")
nn_tuned <- read_rds("output/modeling/nn_tuned.Rds")
```

**Loading Models** Here I trained and loaded 6 models. Nearly all of these were tuned. They are below.

- Linear regression: This is the quintessential ordinary least squares model. It minimizes the sum of the differences between the observed and the predicted output.
- Elastic net: Elastic net models adds bias towards certain values. In Ridge and Lasso regression, the penalty is set whereas in the generalized elastic net, the Ridge and Lasso is combined.
  - Ridge regression: alpha = 0, tuned penalty
  - Lasso regression: alpha = 1, tuned penalty
  - Generalized Elastic Net: tuned alpha, penalty
- Random forest: The random forest model is built from decision trees, which have a habit of overfitting, and averages the trees. Additionally, it randomly selects predictor variables.
  - tuned min_n, mtry
- Boosted tree: The boosted tree model builds an original model, then builds further models sequentially to reduce error.
  - tuned min_n, mtry, learn_rate
- Nearest neighbor: The nearest neighbor model averages values of the nearest observations, with similar observations being closer in distance than different observations.
  - tuned number of neighbors

**Tuning Metrics** I reviewed the metrics for different models.

```
collect_metrics(lm_fit)
```

```
## # A tibble: 2 x 6
```

```
##   .metric .estimator    mean     n std_err .config
##   <chr>   <chr>        <dbl> <int>   <dbl> <chr>
## 1 rmse    standard   438.       50 2.57    Preprocessor1_Model1
## 2 rsq     standard     0.531    50 0.00395 Preprocessor1_Model1
```

The metrics for the linear model.

```
show_best(ridge_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001   rmse    standard    438.    50    2.63 Preprocessor1_Model01
## 2 0.0000139 rmse    standard    438.    50    2.63 Preprocessor1_Model02
## 3 0.0000193 rmse    standard    438.    50    2.63 Preprocessor1_Model03
## 4 0.0000268 rmse    standard    438.    50    2.63 Preprocessor1_Model04
## 5 0.0000373 rmse    standard    438.    50    2.63 Preprocessor1_Model05
```

The metrics for the Ridge regression model.

```
show_best(lasso_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 7
##     penalty .metric .estimator  mean     n std_err .config
##       <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.720     rmse    standard    437.    50    2.58 Preprocessor1_Model35
## 2 1         rmse    standard    437.    50    2.58 Preprocessor1_Model36
## 3 0.00001   rmse    standard    437.    50    2.58 Preprocessor1_Model01
## 4 0.0000139 rmse    standard    437.    50    2.58 Preprocessor1_Model02
## 5 0.0000193 rmse    standard    437.    50    2.58 Preprocessor1_Model03
```

The metrics for the Lasso regression model.

```
show_best(elastic_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   0.720   0.878 rmse    standard    437.    50    2.58 Preprocessor1_Model2185
## 2   0.720   0.918 rmse    standard    437.    50    2.58 Preprocessor1_Model2285
## 3   1       0.837 rmse    standard    437.    50    2.58 Preprocessor1_Model2086
## 4   1       0.816 rmse    standard    437.    50    2.58 Preprocessor1_Model2036
## 5   1       0.796 rmse    standard    437.    50    2.58 Preprocessor1_Model1986
```

The metrics for the generalized Elastic net model.

```
show_best(rf_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 8
##    mtry min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5     2 rmse    standard    228.    50    1.83 Preprocessor1_Model03
## 2     7     2 rmse    standard    229.    50    1.91 Preprocessor1_Model04
## 3     7    11 rmse    standard    229.    50    1.82 Preprocessor1_Model09
## 4     5    11 rmse    standard    229.    50    1.69 Preprocessor1_Model08
## 5     9     2 rmse    standard    230.    50    1.95 Preprocessor1_Model05
```

The metrics for the random forest model.

```
show_best(bt_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 9
##    mtry min_n learn_rate .metric .estimator  mean     n std_err .config
##   <int> <int>      <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     9     2      0.282 rmse    standard    233.    50    1.82 Preprocessor1_M~
## 2     9    21      0.282 rmse    standard    233.    50    1.76 Preprocessor1_M~
## 3     9    11      0.282 rmse    standard    234.    50    1.80 Preprocessor1_M~
## 4     9    30      0.282 rmse    standard    234.    50    1.77 Preprocessor1_M~
## 5     9    40      0.282 rmse    standard    235.    50    1.73 Preprocessor1_M~
```

The metrics for the boosted tree model.

```
show_best(nn_tuned, metric = "rmse")
```

```
## # A tibble: 5 x 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        11 rmse    standard    288.    50    1.83 Preprocessor1_Model4
## 2         8 rmse    standard    288.    50    1.80 Preprocessor1_Model3
## 3        15 rmse    standard    290.    50    1.89 Preprocessor1_Model5
## 4         4 rmse    standard    303.    50    1.90 Preprocessor1_Model2
## 5         1 rmse    standard    363.    50    2.63 Preprocessor1_Model1
```

The metrics for the nearest neighbor regression model.

The linear models all have nearly the same RMSE at ~437. These models perform poorly in comparison to the empirical models. The tree-based models perform the best: the random forest has a RMSE value of 228, and the boosted tree has a RMSE value of 233. The neural network performs slightly worse, with a RMSE of 288.

### Finalizing Workflow

```
rf_model <- rand_forest(mode = "regression",
                        min_n = tune(),
                        mtry = tune()) %>%
  set_engine("ranger")

rf_workflow <- workflow() %>%
  add_recipe(bike_recipe) %>%
  add_model(rf_model)

rf_workflow_tuned <- rf_workflow %>%
  finalize_workflow(select_best(rf_tuned, metric = "rmse"))

rf_results <- fit(rf_workflow_tuned, bike_train)
```

```
bt_model <- boost_tree(mode = "regression",
                       mtry = tune(),
                       min_n = tune(),
                       learn_rate = tune()) %>%
  set_engine("xgboost")

bt_workflow <- workflow() %>%
  add_model(bt_model) %>%
  add_recipe(bike_recipe)
```

```
bt_workflow_tuned <- bt_workflow %>%
  finalize_workflow(select_best(bt_tuned, metric = "rmse"))

bt_results <- fit(bt_workflow_tuned, bike_train)
```

I finalize the workflows based on the parameters that minimize RMSE and fit the two best models, the random forest model and the boosted tree, to the entire training dataset.

### Model Evaluation

```
bike_metric <- metric_set(rmse, mae, rsq)
```

```
predict(rf_results, new_data = bike_test) %>%
  bind_cols(bike_test %>% select(rented_bike_count)) %>%
  bike_metric(truth = rented_bike_count, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard     228.
## 2 mae     standard     141.
## 3 rsq     standard       0.879
```

The performance metrics on the testing dataset for the random forest model.

```
predict(bt_results, new_data = bike_test) %>%
  bind_cols(bike_test %>% select(rented_bike_count)) %>%
  bike_metric(truth = rented_bike_count, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard     229.
## 2 mae     standard     147.
## 3 rsq     standard       0.878
```

The performance metrics on the testing dataset for the boosted tree model.

The mean absolute value (MAE) is slightly smaller for the random forest, but in other aspects, the two models are similar in performance.

## Debrief and Next Steps

The tree-based models performed the best. Due to their empirical nature, they fits non-linear data well. Additionally, trees deal with high dimensional data well.

From the EDA, it was clear that the target variable does not have linear correlation with any of the variables. Furthermore, the assumptions of linear regression, such as homoscedasticity or normality of residuals, were not assessed prior to the fitting. These assumptions appear to have been violated. Thus, none of the linear regression models predicted the rental bike count well, leading to much higher RMSE values.

The dataset was comprehensive in providing weather and time information. It seems unlikely that other datasets would improve model performance. In my approach, I chose to reduce the overall complexity by simplifying the original date variable to the month instead. If the date was kept so that a time series approach could be taken, that could provide a different approach to predicting the rental bike count.

Future research questions are included below.

- Which variables have the most effect on the rental bike count?
- Which variables have the least effect on the rental bike count?
- How do changes in temperature correlate with the rental bike count?
- Would a time-series approach provide better performance?