Modeling Run Performance

Joe Martin

11/26/2021

Garmin Data Modeling

The target variable is average pace (avg_pace), but I will also compare Average Speed (avg_speed) in miles per hour. A higher average speed and a lower average pace are the desired outcome when measuring performance over time. Reviewing the results of the two preliminary linear regression models, the more desirable variable is average pace, as it has stronger relationships with other variables.

The target variable is average pace (avg_pace_sec), measured in seconds. Average pace is the best variable to use because it can predict race times, but be applied to different race lengths. Additionally, it is an actionable measure. A runner can easily monitor and control their pace using a fitness watch. It is important that this model has low error. Even a small amount of error could amount to a dramatic difference in final race time. For example, if a person runs a marathon at a 7:00 pace, their final time is 3:03:32. If a second athlete runs a marathon at a 7:05 pace, they would achieve a 3:05:43 marathon. From this example, we can see that a runner hoping to qualify for a race like the Boston Marathon with a 3:05:00 time would be at risk if they are off pace by just 5 seconds per mile. This will be the benchmark for RMSE values - an estimated value of less than 5 seconds per mile.

```r
# Create preliminary test
prelim_spd <- lm(avg_spd ~ ., df)
prelim_spd_table <- summary(prelim_spd)

#write.table(prelim_spd_table, file = here::here("figures","prelim_spd_table"))
```

```r
prelim_pace <- lm(avg_pace_sec ~ ., df)
summary(prelim_pace)
```

```
##
## Call:
## lm(formula = avg_pace_sec ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -32.387  -3.624  -0.653   2.608  52.722
##
## Coefficients: (1 not defined because of singularities)
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)               1.185e+03  6.408e+01  18.494  < 2e-16 ***
## distance                  4.099e-01  4.657e-01   0.880 0.379371
## avg_hr                    5.530e-01  1.535e-01   3.603 0.000355 ***
## max_hr                    7.341e-02  9.892e-02   0.742 0.458427
## avg_run_cadence          -1.874e+00  3.921e-01  -4.779 2.49e-06 ***
```

```
## max_run_cadence            2.944e-02  3.537e-02   0.832 0.405775    
## total_ascent              -5.631e-03  1.151e-02  -0.489 0.625032    
## total_decent               5.189e-03  1.094e-02   0.474 0.635448    
## avg_stride                -2.402e+02  4.152e+01  -5.785 1.49e-08 ***
## min_elevation             -1.252e-02  1.574e-02  -0.796 0.426668    
## max_elevation              2.196e-02  1.627e-02   1.350 0.177895    
## best_pace_sec              2.860e-02  1.546e-02   1.851 0.064987 .  
## aerobic_TE                -1.290e+01  2.379e+00  -5.421 1.04e-07 ***
## aerobic_fctImpacting      -4.099e+00  1.363e+00  -3.007 0.002806 ** 
## aerobic_fctMaintaining     5.204e+00  2.631e+00   1.978 0.048604 *  
## aerobic_fctOverreaching    5.849e+00  2.138e+00   2.735 0.006513 ** 
## anaerobic_value           -1.262e+00  1.642e+00  -0.768 0.442659    
## anaerobic_fctMaintaining   2.128e+00  2.605e+00   0.817 0.414383    
## anaerobic_fctNo Benefit    1.159e+00  5.160e+00   0.225 0.822431    
## anaerobic_fctSome Benefit  2.494e+00  3.889e+00   0.641 0.521699    
## avg_spd                   -2.330e+01  7.450e+00  -3.127 0.001896 ** 
## max_spd                   -7.389e-03  3.834e-01  -0.019 0.984635    
## short_distanceY           -1.454e+00  2.842e+00  -0.512 0.609177    
## middle_distanceY          -9.990e-01  2.110e+00  -0.474 0.636096    
## long_distanceY                    NA         NA      NA       NA    
## rhr                       -1.885e-02  1.052e-01  -0.179 0.857941    
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 6.751 on 392 degrees of freedom
## Multiple R-squared:  0.9894, Adjusted R-squared:  0.9888 
## F-statistic:  1532 on 24 and 392 DF,  p-value: < 2.2e-16
```

The ultimate goal of this model is to utilize data leading up to a performance event. Thinking about the purpose of the model (predicting how well I can perform given a set of racing conditions), the best target variable to choose is Average Pace (using only seconds as the unit). This variable is easier to work with than total time (which is in an HMS format) while having the same outcome. It is also something I can know in real-time on runs through my watch and has actionable meaning, compared to the average speed variable. Going forward, all models will use average pace (in seconds) as the target variable and use a linear regression for prediction.

```r
set.seed(456)
# Split data into training and testing sets
df_split <- initial_split(df, prop = 3/4)

train_df <- training(df_split)
test_df <- testing(df_split)

# Create recipe
pace_rec <- recipe(avg_pace_sec ~ ., data = train_df)

summary(pace_rec)
```

```
## # A tibble: 22 x 4
##    variable        type    role      source  
##    <chr>           <chr>   <chr>     <chr>   
##  1 distance        numeric predictor original
##  2 avg_hr          numeric predictor original
```

```
##   3 max_hr          numeric predictor original
##   4 avg_run_cadence numeric predictor original
##   5 max_run_cadence numeric predictor original
##   6 total_ascent    numeric predictor original
##   7 total_decent    numeric predictor original
##   8 avg_stride      numeric predictor original
##   9 min_elevation   numeric predictor original
##  10 max_elevation   numeric predictor original
## # ... with 12 more rows
```

```
lm_pace <- linear_reg() %>%
  set_engine("lm")

pace_wflow <- workflow()%>%
  add_model(lm_pace) %>%
  add_recipe(pace_rec)

pace_fit <- pace_wflow %>%
  fit(data = train_df)

tidy(pace_fit)
```

```
## # A tibble: 26 x 5
##    term              estimate std.error statistic  p.value
##    <chr>                <dbl>     <dbl>     <dbl>    <dbl>
##  1 (Intercept)     1157.        76.0       15.2   1.02e-38
##  2 distance           0.820      0.562      1.46  1.45e- 1
##  3 avg_hr             0.596      0.183      3.26  1.23e- 3
##  4 max_hr             0.0970     0.122      0.793 4.29e- 1
##  5 avg_run_cadence   -1.71       0.463     -3.70  2.61e- 4
##  6 max_run_cadence    0.0152     0.0434     0.350 7.27e- 1
##  7 total_ascent      -0.00545    0.0146    -0.374 7.09e- 1
##  8 total_decent       0.00390    0.0137     0.285 7.76e- 1
##  9 avg_stride      -228.        49.7       -4.59  6.54e- 6
## 10 min_elevation     -0.0109     0.0201    -0.541 5.89e- 1
## # ... with 16 more rows
```

```
predict(pace_fit, test_df)
```

```
## Warning in predict.lm(object = object$fit, newdata = new_data, type =
## "response"): prediction from a rank-deficient fit may be misleading
```

```
## # A tibble: 105 x 1
##    .pred
##    <dbl>
##  1  447.
##  2  447.
##  3  436.
##  4  440.
##  5  397.
##  6  414.
##  7  428.
```

```
## 8  435.
## 9  448.
## 10  434.
## # ... with 95 more rows
```

```
pace_aug <- augment(pace_fit, test_df)
```

```
## Warning in predict.lm(object = object$fit, newdata = new_data, type =
## "response"): prediction from a rank-deficient fit may be misleading
```

```
pace_aug %>% select(avg_pace_sec, .pred)
```

```
## # A tibble: 105 x 2
##    avg_pace_sec .pred
##           <dbl> <dbl>
##  1          447  447.
##  2          449  447.
##  3          432  436.
##  4          438  440.
##  5          391  397.
##  6          414  414.
##  7          419  428.
##  8          432  435.
##  9          444  448.
## 10          430  434.
## # ... with 95 more rows
```

The R Mean-Squared Error for this model is 5.64. In other words, this model can predict average pace within 5.24 seconds.

```
pace_error <- pace_aug %>%
  rmse(truth = avg_pace_sec, .pred)

pace_error
```

```
## # A tibble: 1 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard        5.65
```

These analyses provide a good starting point for building a more complex model that can predict good performance. The possible next step is to use v-fold cross validation to enhance the quality of my training set. In this section, the random forest model will use v-fold cross validation and train with all variables.

```
pacman::p_load(tidymodels, ranger, parallel)

cores <- parallel::detectCores()

set.seed(456)

# Split data into training and testing sets
```

```r
df_split <- initial_split(df, prop = 3/4)

train_df <- training(df_split)
test_df <- testing(df_split)

# Create recipe
rf_rec <- recipe(avg_pace_sec ~ ., data = train_df) %>%
          step_dummy(all_nominal_predictors())

folds <- vfold_cv(train_df, v = 10, repeats = 5, strata = avg_pace_sec)

summary(rf_rec)
```

```
## # A tibble: 22 x 4
##    variable        type    role      source
##    <chr>           <chr>   <chr>     <chr>
##  1 distance        numeric predictor original
##  2 avg_hr          numeric predictor original
##  3 max_hr          numeric predictor original
##  4 avg_run_cadence numeric predictor original
##  5 max_run_cadence numeric predictor original
##  6 total_ascent    numeric predictor original
##  7 total_decent    numeric predictor original
##  8 avg_stride      numeric predictor original
##  9 min_elevation   numeric predictor original
## 10 max_elevation   numeric predictor original
## # ... with 12 more rows
```

```r
rf_mod <- rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger", num.threads = cores) %>%
  set_mode("regression")

rf_wf <- workflow() %>%
  add_model(rf_mod) %>%
  add_recipe(rf_rec)

rf_res <- rf_wf %>%
  tune_grid(folds,
            grid = 25,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(rmse))
```

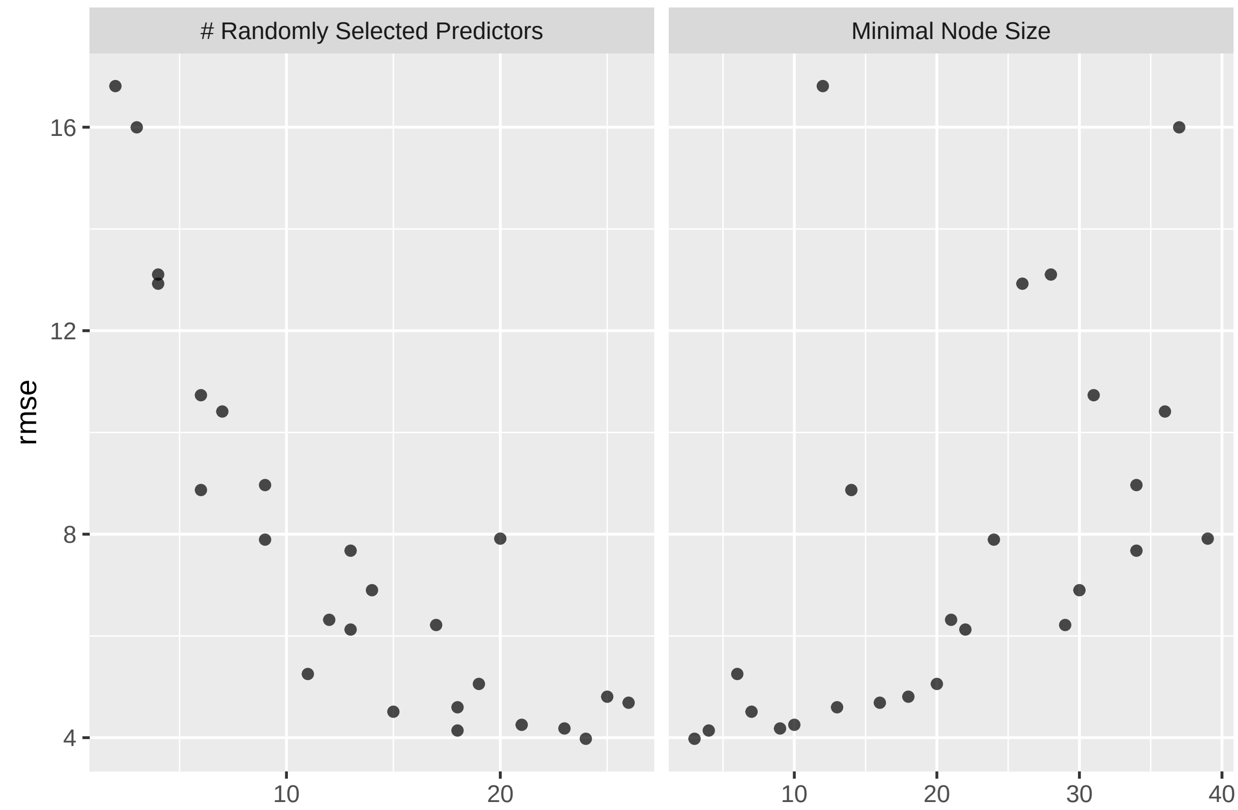
```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```r
rf_res %>%
  show_best(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    24     3 rmse    standard    3.98    50   0.310 Preprocessor1_Model01
## 2    18     4 rmse    standard    4.14    50   0.317 Preprocessor1_Model25
```

```
## 3    23      9 rmse    standard     4.18    50   0.362 Preprocessor1_Model07
## 4    21     10 rmse    standard     4.25    50   0.371 Preprocessor1_Model08
## 5    15      7 rmse    standard     4.51    50   0.343 Preprocessor1_Model15
```

```r
autoplot(rf_res)
```




```r
rf_best <- rf_res %>%
  select_best(metric = "rmse")
rf_res %>% collect_predictions()
```

```
## # A tibble: 39,000 x 8
##    id      id2    .pred  .row  mtry min_n avg_pace_sec .config
##    <chr>   <chr>  <dbl> <int> <int> <int>        <dbl> <chr>
##  1 Repeat1 Fold01  455.    17    24     3          459 Preprocessor1_Model01
##  2 Repeat1 Fold01  485.    19    24     3          485 Preprocessor1_Model01
##  3 Repeat1 Fold01  581.    26    24     3          577 Preprocessor1_Model01
##  4 Repeat1 Fold01  624.    29    24     3          624 Preprocessor1_Model01
##  5 Repeat1 Fold01  441.    33    24     3          436 Preprocessor1_Model01
##  6 Repeat1 Fold01  610.    79    24     3          613 Preprocessor1_Model01
##  7 Repeat1 Fold01  529.    93    24     3          539 Preprocessor1_Model01
##  8 Repeat1 Fold01  561.    98    24     3          563 Preprocessor1_Model01
##  9 Repeat1 Fold01  468.   104    24     3          470 Preprocessor1_Model01
## 10 Repeat1 Fold01  511.   117    24     3          512 Preprocessor1_Model01
## # ... with 38,990 more rows
```

```
final_rf_wf <- rf_wf %>%
  finalize_workflow(rf_best)

final_fit_rf <- final_rf_wf %>%
  last_fit(df_split)

final_fit_rf %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       2.49  Preprocessor1_Model1
## 2 rsq     standard       0.998 Preprocessor1_Model1
```

```
rf_rmse <-
  rf_res %>%
  collect_predictions(parameters = rf_best) %>%
  rmse(avg_pace_sec, .pred) %>%
  mutate(model = "Random Forest")
rf_rmse
```

```
## # A tibble: 1 x 4
##   .metric .estimator .estimate model
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard        4.56 Random Forest
```

Next, tune the parameters. mtry = 24, min_n=3

```
tuned_rf <- rand_forest(mtry = 24, min_n = 3, trees = 1000) %>%
  set_engine("ranger", num.threads = cores, importance = "impurity") %>%
  set_mode("regression")

tuned_wf <- rf_wf %>%
  update_model(tuned_rf)

tuned_rf_fit <- tuned_wf %>%
  last_fit(df_split)
tuned_rf_fit
```

```
## # Resampling results
## # Manual resampling
## # A tibble: 1 x 6
##   splits            id               .metrics    .notes   .predictions   .workflow
##   <list>            <chr>            <list>      <list>   <list>         <list>
## 1 <split [312/105]> train/test split <tibble [~ <tibble~ <tibble [105~ <workflo~
```

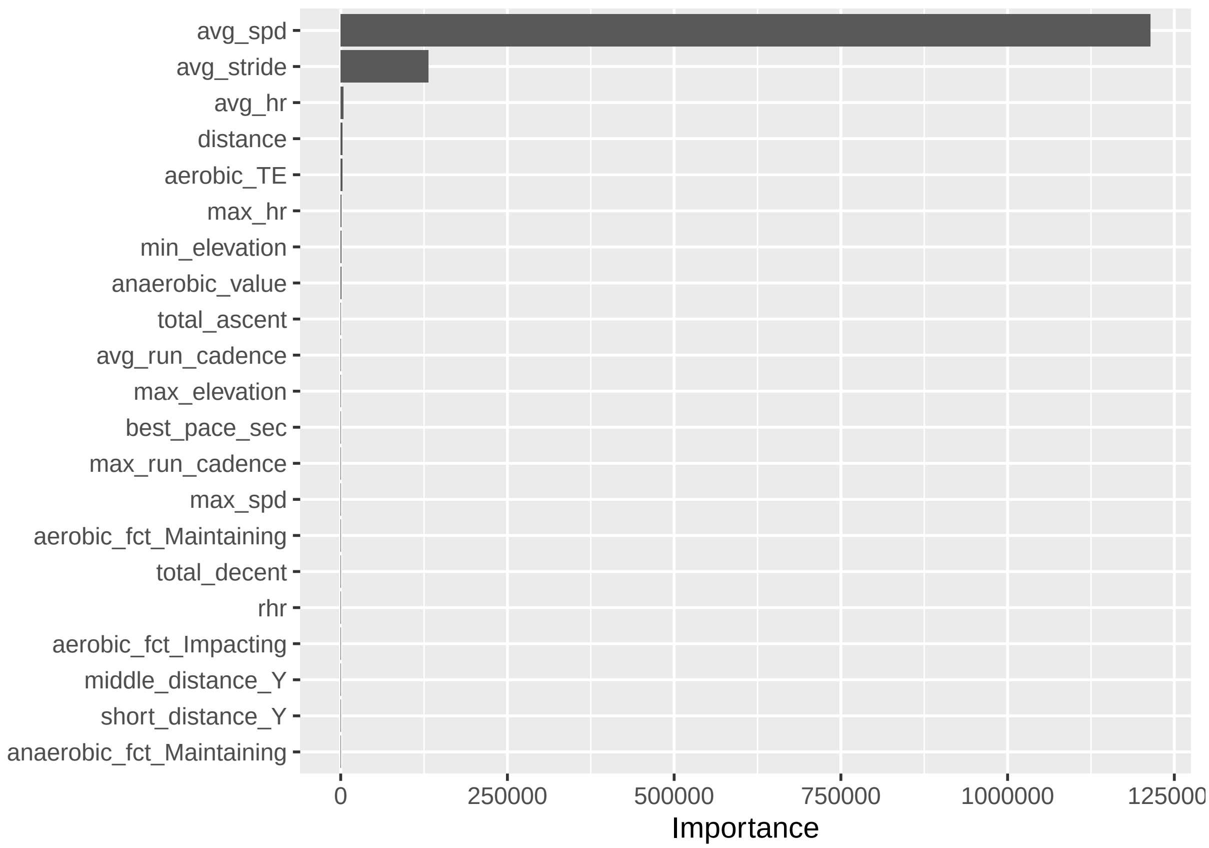
```
tuned_rf_fit %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       2.46  Preprocessor1_Model1
## 2 rsq     standard       0.998 Preprocessor1_Model1
```

This model predicts average pace within 2.45 seconds. This is an excellent error value, given the constraints defined earlier. Because this error is lower than 5 seconds per mile, it would work well as a final model.

One more consideration to try improving this model is that there is a large number of features, a total of 21 predictors. The following figure shows how which are most relevant to predicting average pace:

```
pacman::p_load(vip)
tuned_rf_fit %>%
  pluck(".workflow", 1) %>%
  extract_fit_parsnip() %>%
  vip(num_features = 21)
```



Reviewing these relevance of each variable, it seems that the variable with the greatest impact is average speed (avg_spd). When building the model, the `importance = "impurity"` argument sets the importance measurement to variance by default for regression models. This figure is problematic because it the avg_spd variable may constitute data leakage. Technically, this value is not known until the conclusion of a run and it is directly related to the target variable. The model should be re-run without avg_spd.

```
pacman::p_load(tidymodels, ranger, parallel)

cores <- parallel::detectCores()

set.seed(456)

no_spd <- df %>% select(-avg_spd, -max_spd)

# Split data into training and testing sets
```

```r
df_split2 <- initial_split(no_spd, prop = 3/4)

train_df2 <- training(df_split2)
test_df2 <- testing(df_split2)

# Create recipe
rf_rec2 <- recipe(avg_pace_sec ~ ., data = train_df2) %>%
          step_dummy(all_nominal_predictors())

folds <- vfold_cv(train_df2, v = 10, repeats = 5, strata = avg_pace_sec)

summary(rf_rec2)
```

```
## # A tibble: 20 x 4
##    variable        type    role      source
##    <chr>           <chr>   <chr>     <chr>
##  1 distance        numeric predictor original
##  2 avg_hr          numeric predictor original
##  3 max_hr          numeric predictor original
##  4 avg_run_cadence numeric predictor original
##  5 max_run_cadence numeric predictor original
##  6 total_ascent    numeric predictor original
##  7 total_decent    numeric predictor original
##  8 avg_stride      numeric predictor original
##  9 min_elevation   numeric predictor original
## 10 max_elevation   numeric predictor original
## 11 best_pace_sec   numeric predictor original
## 12 aerobic_TE      numeric predictor original
## 13 aerobic_fct     nominal predictor original
## 14 anaerobic_value numeric predictor original
## 15 anaerobic_fct   nominal predictor original
## 16 short_distance  nominal predictor original
## 17 middle_distance nominal predictor original
## 18 long_distance   nominal predictor original
## 19 rhr             numeric predictor original
## 20 avg_pace_sec    numeric outcome   original
```

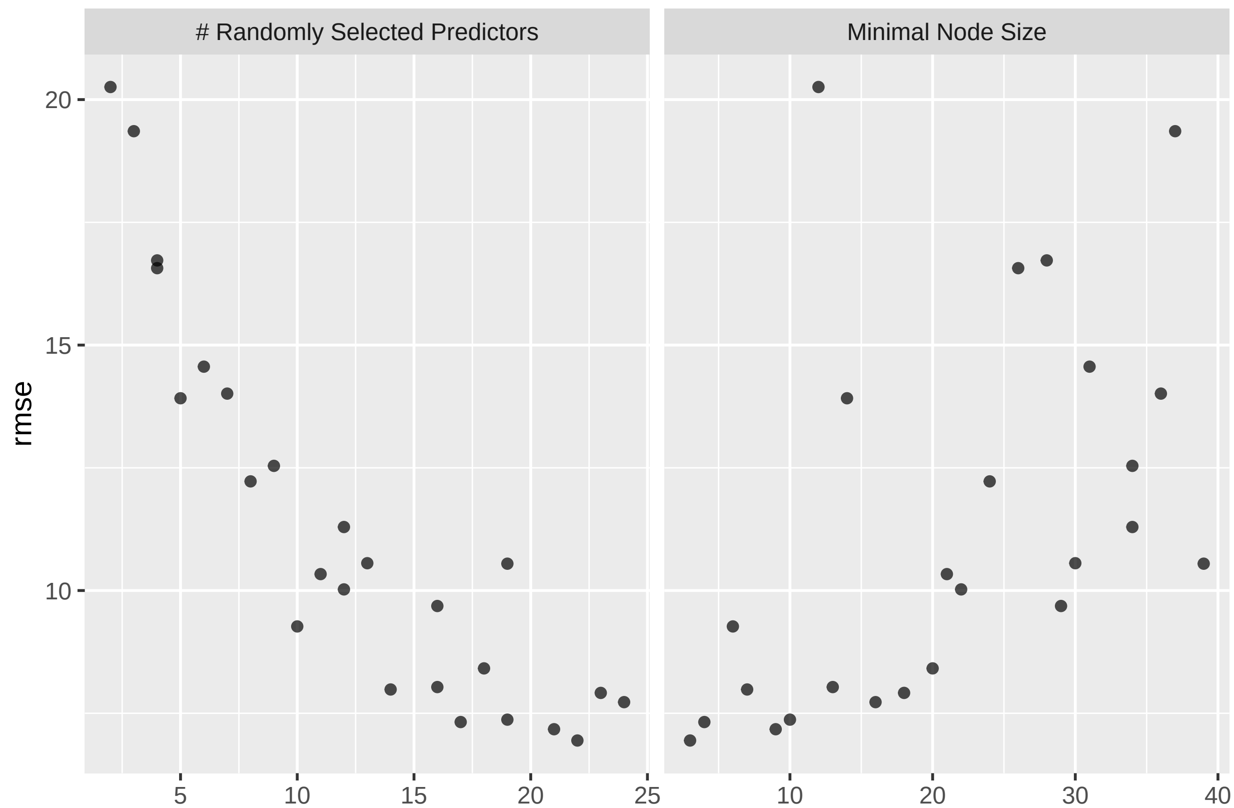
```r
rf_mod2 <- rand_forest(mtry = tune(), min_n = tune(), trees = 1000) %>%
  set_engine("ranger", num.threads = cores) %>%
  set_mode("regression")

rf_wf2 <- workflow() %>%
  add_model(rf_mod2) %>%
  add_recipe(rf_rec2)

rf_res2 <- rf_wf2 %>%
  tune_grid(folds,
            grid = 25,
            control = control_grid(save_pred = TRUE),
            metrics = metric_set(rmse))
```

```
## i Creating pre-processing data to finalize unknown parameter: mtry
```

```
rf_res2 %>%
  show_best(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    22     3 rmse    standard    6.95    50   0.276 Preprocessor1_Model01
## 2    21     9 rmse    standard    7.18    50   0.307 Preprocessor1_Model07
## 3    17     4 rmse    standard    7.32    50   0.261 Preprocessor1_Model25
## 4    19    10 rmse    standard    7.37    50   0.310 Preprocessor1_Model08
## 5    24    16 rmse    standard    7.73    50   0.369 Preprocessor1_Model22
```

```
autoplot(rf_res2)
```




```
rf_best2 <- rf_res2 %>%
  select_best(metric = "rmse")
rf_res2 %>% collect_predictions()
```

```
## # A tibble: 39,000 x 8
##    id      id2    .pred  .row  mtry min_n avg_pace_sec .config
##    <chr>   <chr>  <dbl> <int> <int> <int>        <dbl> <chr>
##  1 Repeat1 Fold01  451.    17    22     3          459 Preprocessor1_Model01
##  2 Repeat1 Fold01  483.    19    22     3          485 Preprocessor1_Model01
##  3 Repeat1 Fold01  575.    26    22     3          577 Preprocessor1_Model01
```

```
##   4 Repeat1 Fold01  621.    29    22     3          624 Preprocessor1_Model01
##   5 Repeat1 Fold01  436.    33    22     3          436 Preprocessor1_Model01
##   6 Repeat1 Fold01  611.    79    22     3          613 Preprocessor1_Model01
##   7 Repeat1 Fold01  515.    93    22     3          539 Preprocessor1_Model01
##   8 Repeat1 Fold01  569.    98    22     3          563 Preprocessor1_Model01
##   9 Repeat1 Fold01  470.   104    22     3          470 Preprocessor1_Model01
##  10 Repeat1 Fold01  503.   117    22     3          512 Preprocessor1_Model01
## # ... with 38,990 more rows
```

```
rf_best2
```

```
## # A tibble: 1 x 3
##    mtry min_n .config
##   <int> <int> <chr>
## 1    22     3 Preprocessor1_Model01
```

```
final_rf_wf2 <- rf_wf2 %>%
  finalize_workflow(rf_best2)

final_fit_rf2 <- final_rf_wf2 %>%
  last_fit(df_split2)

final_fit_rf2 %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       6.48  Preprocessor1_Model1
## 2 rsq     standard       0.986 Preprocessor1_Model1
```

```
rf_rmse2 <-
  rf_res2 %>%
  collect_predictions(parameters = rf_best2) %>%
  rmse(avg_pace_sec, .pred) %>%
  mutate(model = "Random Forest")
rf_rmse2
```

```
## # A tibble: 1 x 4
##   .metric .estimator .estimate model
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard        7.22 Random Forest
```

Tune parameters. mtry = 22, min_n = 3

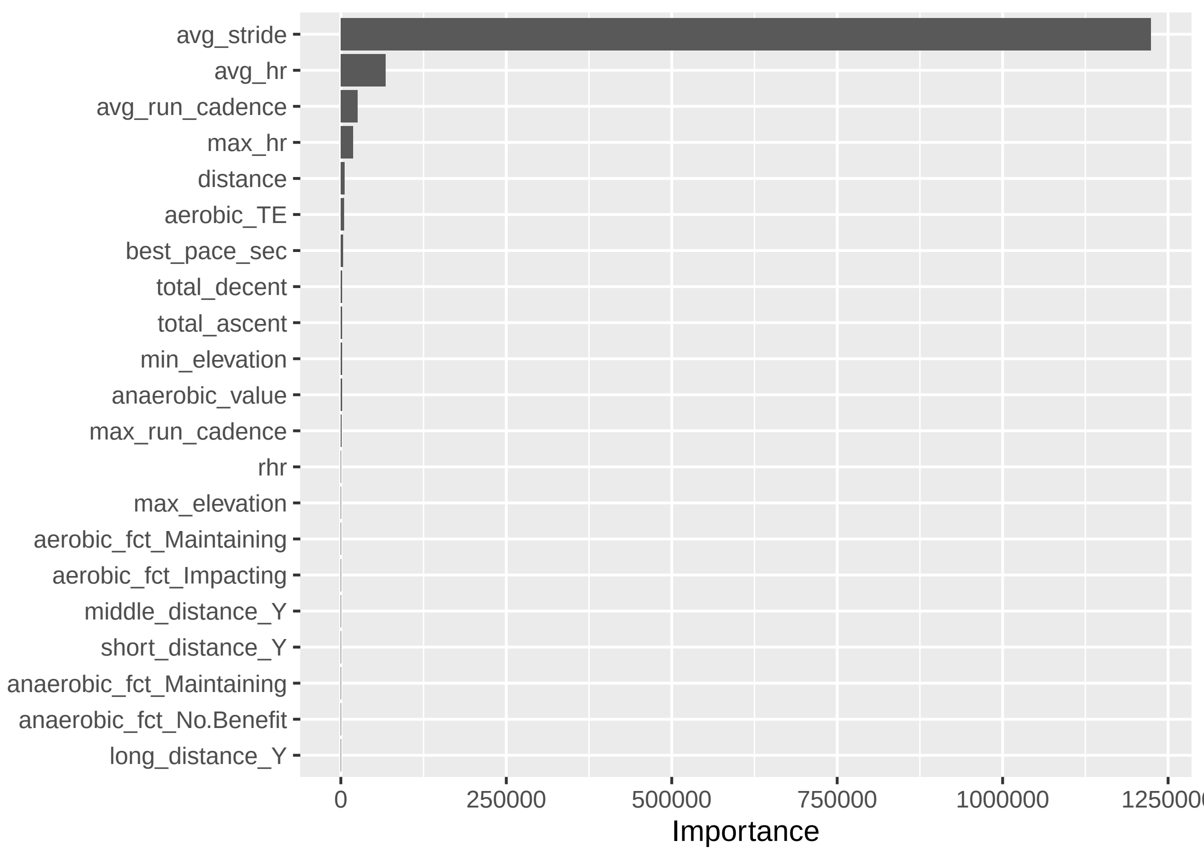
```
tuned_rf2 <- rand_forest(mtry = 22, min_n = 2, trees = 1000) %>%
  set_engine("ranger", num.threads = cores, importance = "impurity") %>%
  set_mode("regression")

tuned_wf2 <- rf_wf2 %>%
  update_model(tuned_rf2)

tuned_rf_fit2 <- tuned_wf2 %>%
  last_fit(df_split2)
tuned_rf_fit2
```

```
## # Resampling results
## # Manual resampling
## # A tibble: 1 x 6
##   splits            id               .metrics   .notes   .predictions   .workflow
##   <list>            <chr>            <list>     <list>   <list>         <list>
## 1 <split [312/105]> train/test split <tibble [~ <tibble~ <tibble [105~ <workflo~
```

```
tuned_rf_fit2 %>% collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       6.39  Preprocessor1_Model1
## 2 rsq     standard       0.987 Preprocessor1_Model1
```

```
tuned_rf_fit2 %>%
  pluck(".workflow", 1) %>%
  extract_fit_parsnip() %>%
  vip(num_features = 21)
```



The RMSE clearly is not as good for this model, but this shows a clearer picture of which variables are most important. It seems that many of the variables have little affect on the model. With the relatively large number of variables in this model, a LASSO regression may be a good option to automate feature selection.

```r
# Excellent tidymodels LASSO tutorial from Julia Silge: https://www.youtube.com/watch?v=R32AsuKICAY
set.seed(456)
# Split data into training and testing sets
# use no_spd splits called df_split2, train_df2, and test_df2

#new dataframe
final_df <- no_spd %>% select(-anaerobic_fct,-aerobic_fct,-max_elevation,-rhr,-max_run_cadence, -anaerob

final_split <- initial_split(final_df, prop = 4/5, strata = avg_pace_sec)

train_fin <- training(final_split)
testing_fin <- testing(final_split)

# Create recipe
lasso_rec <- recipe(avg_pace_sec ~ ., data = train_fin) %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) #center and scale
lasso_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         10
##
## Operations:
##
## Zero variance filter on all_numeric(), -all_outcomes()
## Centering and scaling for all_numeric(), -all_outcomes()
```

```r
# create folds
folds <- vfold_cv(train_df2, v = 10, repeats = 5, strata = avg_pace_sec)

# create validation set
val_set <- validation_split(train_df2,
                            strata = avg_pace_sec,
                            prop = 0.80)
val_set
```

```
## # Validation Set Split (0.8/0.2)  using stratification
## # A tibble: 1 x 2
##   splits           id
##   <list>           <chr>
## 1 <split [248/64]> validation
```

```r
summary(lasso_rec)
```

```
## # A tibble: 11 x 4
##    variable        type    role      source
##    <chr>           <chr>   <chr>     <chr>
```

```
##  1 distance        numeric predictor original
##  2 avg_hr          numeric predictor original
##  3 max_hr          numeric predictor original
##  4 avg_run_cadence numeric predictor original
##  5 total_ascent    numeric predictor original
##  6 total_decent    numeric predictor original
##  7 avg_stride      numeric predictor original
##  8 min_elevation   numeric predictor original
##  9 best_pace_sec   numeric predictor original
## 10 aerobic_TE      numeric predictor original
## 11 avg_pace_sec    numeric outcome   original
```

```
lasso_spec <- linear_reg(penalty = 0.1, mixture = 1) %>%
  set_engine("glmnet")
```

```
lasso_wkfl <- workflow() %>%
  add_recipe(lasso_rec)

lasso_fit <- lasso_wkfl %>%
  add_model(lasso_spec) %>%
  fit(data = train_fin)

lasso_fit %>%
  pull_workflow_fit() %>%
  tidy()
```

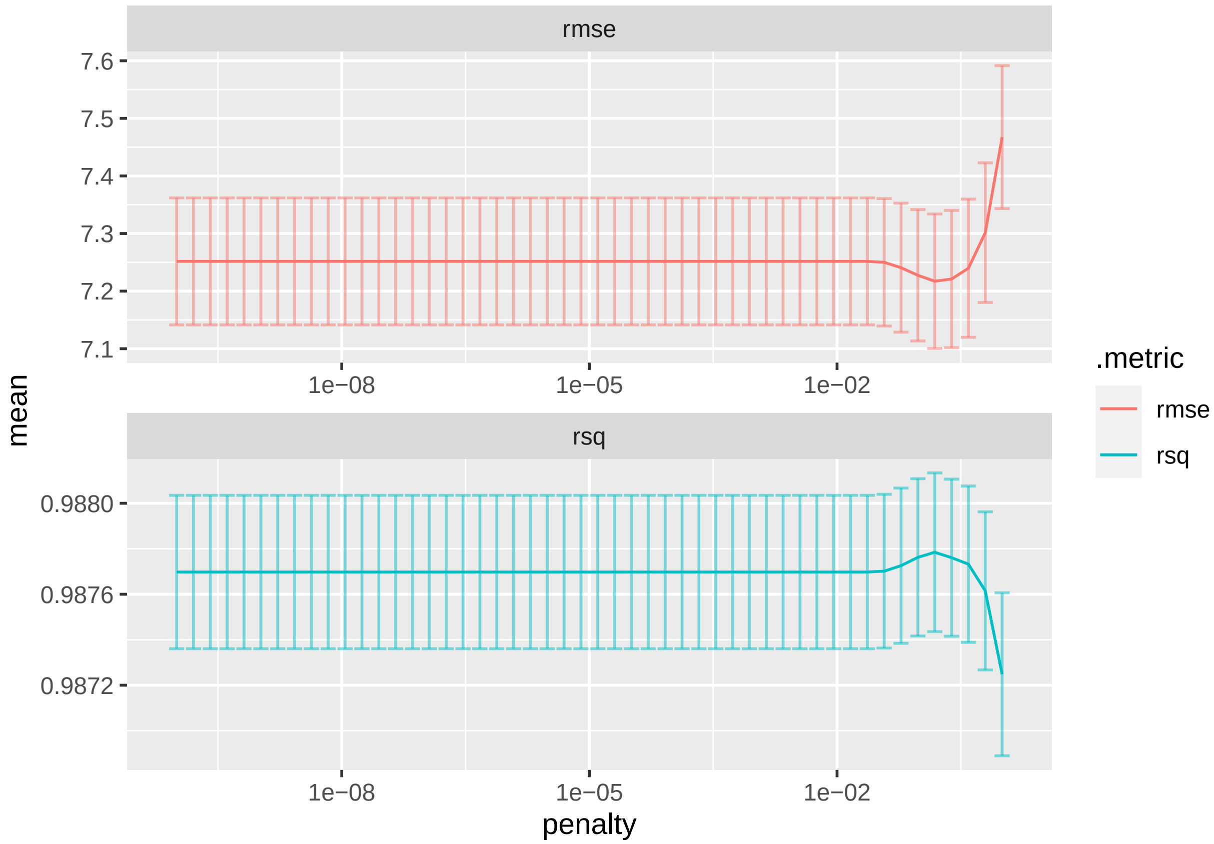
```
## Warning: `pull_workflow_fit()` was deprecated in workflows 0.2.3.
## Please use `extract_fit_parsnip()` instead.

## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 4.1-2

## # A tibble: 11 x 3
##    term            estimate penalty
##    <chr>              <dbl>   <dbl>
##  1 (Intercept)      483.        0.1
##  2 distance          -0.381     0.1
##  3 avg_hr             0         0.1
##  4 max_hr             0         0.1
##  5 avg_run_cadence  -12.2       0.1
##  6 total_ascent       0         0.1
##  7 total_decent       0         0.1
##  8 avg_stride       -51.4       0.1
##  9 min_elevation     -1.91      0.1
## 10 best_pace_sec      2.41      0.1
## 11 aerobic_TE        -4.16      0.1
```

```r
# pick the penalty value with resampling and tuning
# when running models, I keep getting the warning "! Bootstrap11: preprocessor 1/1, model 1/1 (predicti
# Upon further research, it seems that I need to remove variables for this to work well. I'm going to g


set.seed(456)
garmin_boot <- bootstraps(train_fin, strata = avg_pace_sec)

tune_spec <- linear_reg(penalty = tune(), mixture = 1) %>%
  set_engine("glmnet")

lambda_grid <- grid_regular(penalty(),
                            levels = 50)

doParallel::registerDoParallel()

set.seed(2020)
lasso_grid <- tune_grid(
  lasso_wkfl %>%
    add_model(tune_spec),
  resamples = garmin_boot,
  grid = lambda_grid
)
```

```r
lasso_grid_plot <- lasso_grid %>%
  collect_metrics() %>%
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_errorbar(aes(ymin = mean - std_err,
                    ymax = mean + std_err),
                alpha = .5)+
  geom_line(show.legent = FALSE) +
  facet_wrap(~.metric, scales = "free", nrow = 2) +
  scale_x_log10()
```

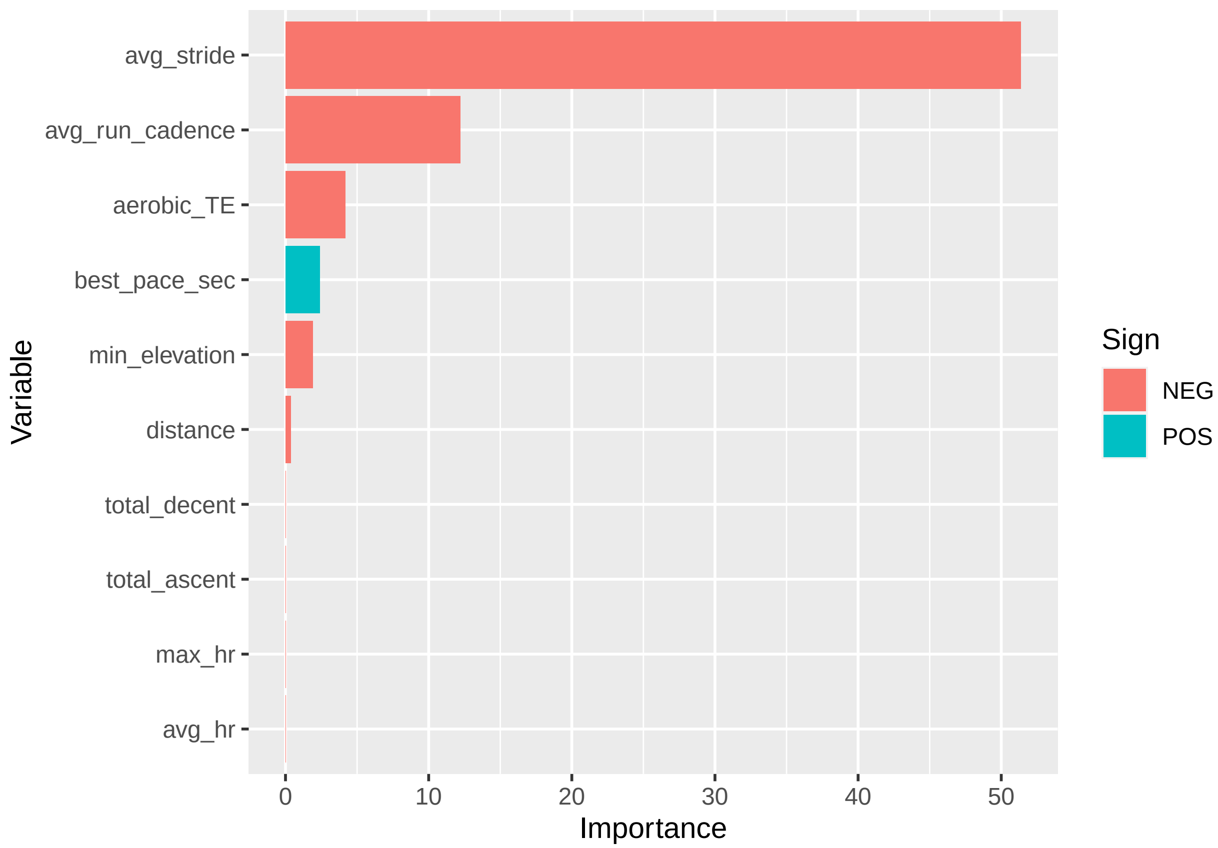
```
## Warning: Ignoring unknown parameters: show.legent
```

```r
lasso_grid_plot
```




```
low_rmse <- lasso_grid %>%
  select_best("rmse") #best metric is model 23

#create final workflow
final_lasso <- finalize_workflow(lasso_wkfl %>%
                    add_model(tune_spec),
                  low_rmse)
pacman::p_load(vip)

#train best model
final_lasso %>%
  fit(train_fin) %>%
  pull_workflow_fit() %>%
  vi(lambda = low_rmse$penalty) %>%
  mutate(Importance = abs(Importance),
         Variable = fct_reorder(Variable, Importance)) %>%
  ggplot(aes(x = Importance, y = Variable, fill = Sign))+
  geom_col()
```

```
## Warning: ‘pull_workflow_fit()‘ was deprecated in workflows 0.2.3.
## Please use ‘extract_fit_parsnip()‘ instead.
```




```
last_fit(final_lasso,
         final_split) %>%
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard      10.3   Preprocessor1_Model1
## 2 rsq     standard       0.977 Preprocessor1_Model1
```

After creating a grid for this LASSO model, I found that my RMSE actually did worse. I want it to be under 5, but this resulted in a value greater than 10. The other important lesson learned with this model is that most of the variables with high importance have negative importance values. This means these variables could be irrelevant, or it could mean that my model is underfitting based on these variables. Since LASSO models are used to regularize, I'm going to try another model to see if I can improve my results. Since I had better luck with my random forest, I'm going to return to that and follow a different process laid out by Julia Silge in this tidy tuesday: https://juliasilge.com/blog/intro-tidymodels/

Previously, I used `glmnet` as my model engine. I'm planning now to use `lm`.

```
#create a simple linear model. This will be used to compare to random forest values
set.seed(456)
rf_split <- final_df %>%
  initial_split(strata = avg_pace_sec)
```

```
rf_train <- training(rf_split)
rf_test <- testing(rf_split)

# Create recipe
rf_rec <- recipe(avg_pace_sec ~ ., data = train_fin) %>%
  step_zv(all_numeric(), -all_outcomes()) %>%
  step_normalize(all_numeric(), -all_outcomes()) #center and scale
rf_rec
```

```
## Recipe
##
## Inputs:
##
##       role #variables
##    outcome          1
##  predictor         10
##
## Operations:
##
## Zero variance filter on all_numeric(), -all_outcomes()
## Centering and scaling for all_numeric(), -all_outcomes()
```

```
#initiate model
lm_spec <- linear_reg() %>%
  set_engine(engine = "lm")

#fit model
lm_fit <- lm_spec %>%
  fit(avg_pace_sec ~ .,
    data = rf_train
  )
lm_fit
```

```
## parsnip model object
##
## Fit time:  0ms
##
## Call:
## stats::lm(formula = avg_pace_sec ~ ., data = data)
##
## Coefficients:
##     (Intercept)         distance           avg_hr           max_hr
##       1.452e+03        1.078e-01        5.711e-01       -1.078e-01
## avg_run_cadence     total_ascent     total_decent       avg_stride
##      -3.054e+00       -6.306e-03        1.063e-02       -3.853e+02
##   min_elevation    best_pace_sec       aerobic_TE
##      -6.142e-02        3.039e-02       -1.047e+01
```

```
#set engine
rf_spec <- rand_forest(mode = "regression") %>%
  set_engine("ranger")
rf_spec
```

```
## Random Forest Model Specification (regression)
##
## Computational engine: ranger
```

```
#create fit without recipe.
rf_fit <- rf_spec %>%
  fit(avg_pace_sec ~ .,
    data = rf_train
  )

rf_fit
```

```
## parsnip model object
##
## Fit time:  260ms
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, num.threads = 1,      verbose = FALSE, seed = sample
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      312
## Number of independent variables:  10
## Mtry:                             3
## Target node size:                 5
## Variable importance mode:         none
## Splitrule:                        variance
## OOB prediction error (MSE):       114.3259
## R squared (OOB):                  0.9718918
```

```
results_train <- lm_fit %>%
  predict(new_data = rf_train) %>%
  mutate(
    truth = rf_train$avg_pace_sec,
    model = "lm"
  ) %>%
  bind_rows(rf_fit %>%
    predict(new_data = rf_train) %>%
    mutate(
      truth = rf_train$avg_pace_sec,
      model = "rf"
    ))

results_test <- lm_fit %>%
  predict(new_data = rf_test) %>%
  mutate(
    truth = rf_test$avg_pace_sec,
    model = "lm"
  ) %>%
  bind_rows(rf_fit %>%
    predict(new_data = rf_test) %>%
    mutate(
```

```
      truth = rf_test$avg_pace_sec,
      model = "rf"
    ))
```

This model meets the standard of predicting with an RMSE lower than 5. Will the testing data work, as well?

```
results_train %>%
  group_by(model) %>%
  rmse(truth = truth, estimate = .pred)
```

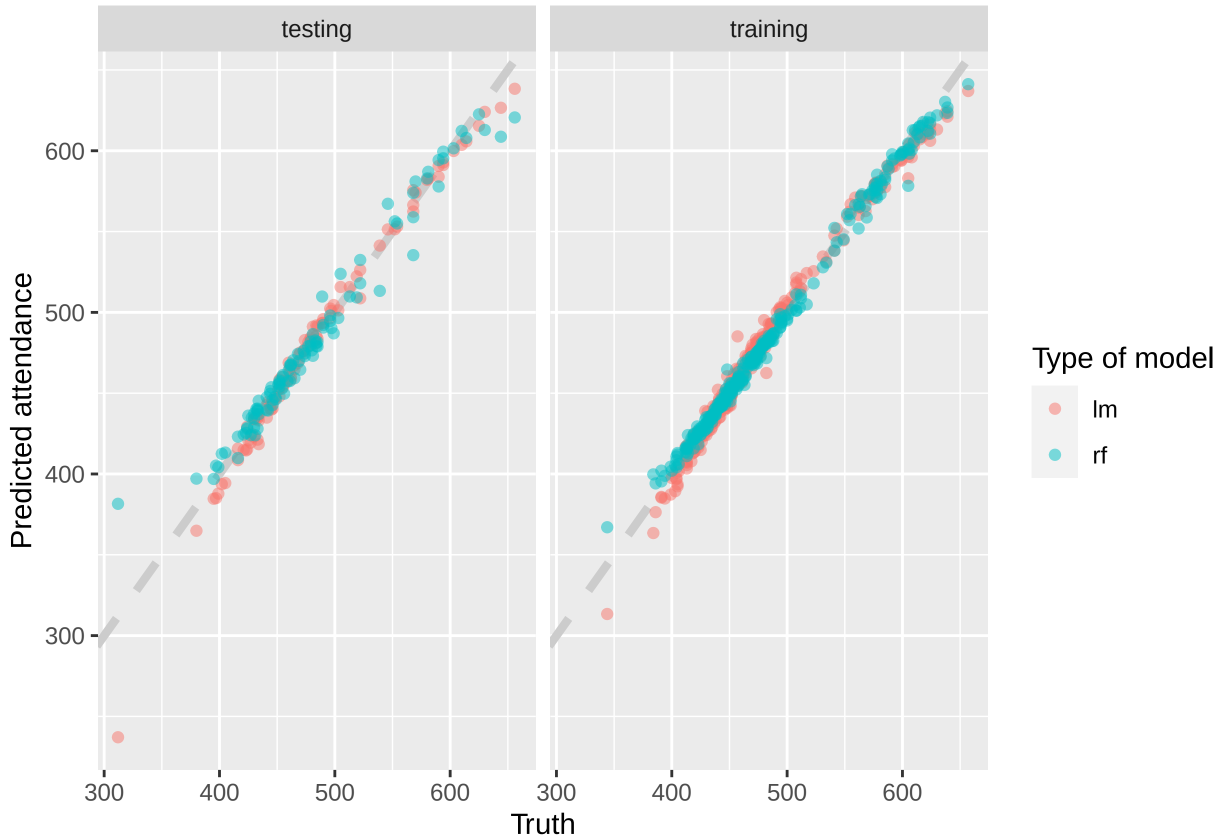
```
## # A tibble: 2 x 4
##   model .metric .estimator .estimate
##   <chr> <chr>   <chr>          <dbl>
## 1 lm    rmse    standard        6.57
## 2 rf    rmse    standard        4.61
```

This model still is not a great choice based on the RMSE value for the testing data. The next step is to try resampling.

```
results_test %>%
  group_by(model) %>%
  rmse(truth = truth, estimate = .pred)
```

```
## # A tibble: 2 x 4
##   model .metric .estimator .estimate
##   <chr> <chr>   <chr>          <dbl>
## 1 lm    rmse    standard        9.58
## 2 rf    rmse    standard       11.5
```

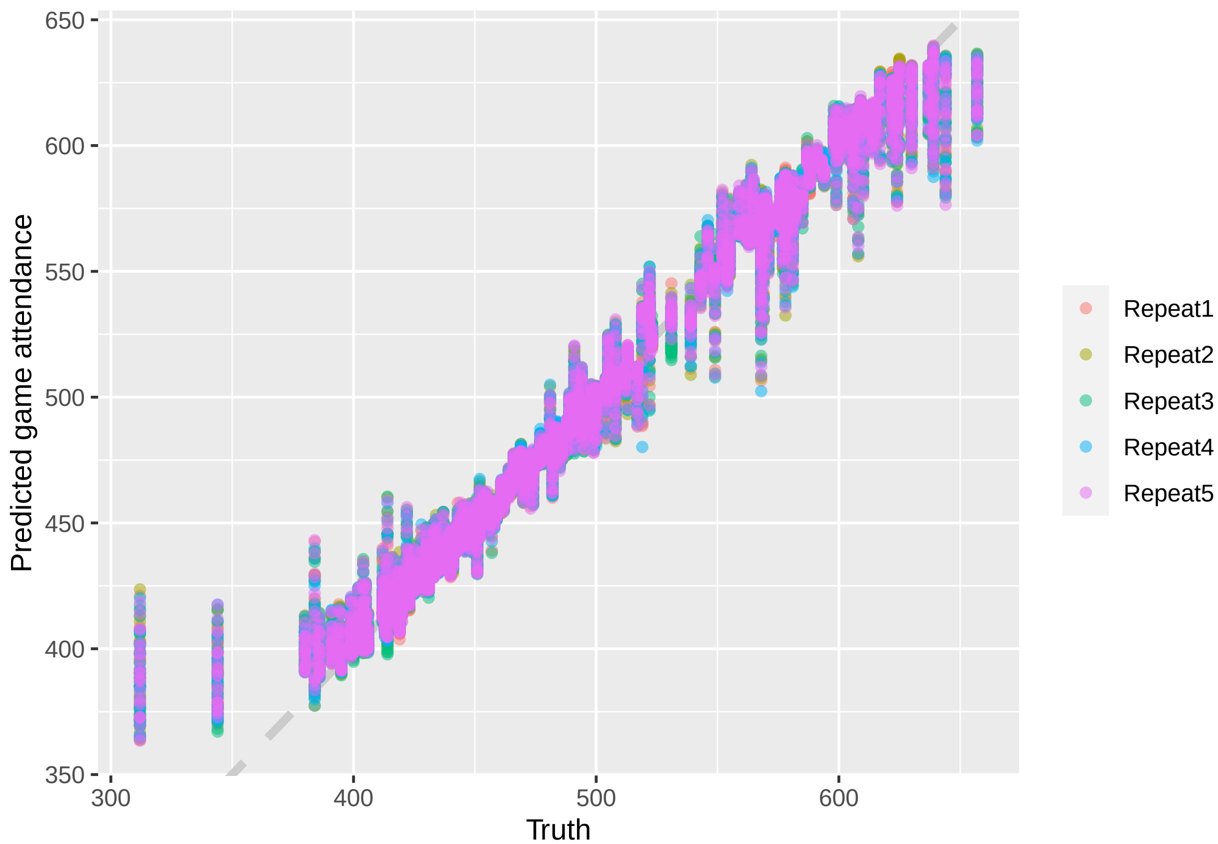
```
results_test %>%
  mutate(train = "testing") %>%
  bind_rows(results_train %>%
    mutate(train = "training")) %>%
  ggplot(aes(truth, .pred, color = model)) +
  geom_abline(lty = 2, color = "gray80", size = 1.5) +
  geom_point(alpha = 0.5) +
  facet_wrap(~train) +
  labs(
    x = "Truth",
    y = "Predicted attendance",
    color = "Type of model"
  )
```




```r
# training
set.seed(456)
rf_folds <- rsample::vfold_cv(rf_train)

rf_wf <- workflow() %>%
  add_model(rf_spec) %>%
  add_recipe(rf_rec)


rf_res <- rf_wf %>% fit_resamples(
  resamples = (rf_folds),
  control = control_resamples(save_pred = TRUE)
)

rf_res %>%
  collect_metrics()

#testing
rf_testing_fit <- predict(rf_wf, testing_fin)

rf_final <- rf_wf %>%
  last_fit(final_split)
rf_final #fitting test data using resampled results

rf_final %>% collect_metrics()
```

Once again, the final fit did not do as well as the initial fit.

```
rf_res %>%
  unnest(.predictions) %>%
  ggplot(aes(avg_pace_sec, .pred, color = id)) +
  geom_abline(lty = 2, color = "gray80", size = 1.5) +
  geom_point(alpha = 0.5) +
  labs(
    x = "Truth",
    y = "Predicted game attendance",
    color = NULL
  )
```



As it turns out, the best model built is a simple linear regression. Resampling and other types of models like LASSO regressions and random forests simply resulted in greater error. The initial model will be deployed with an error of 5.64 seconds. When more data is available (there should be more than 700 observations in one year), then a new model will be evaluated.