HW 8 & 9 -Modeling Practice

This homework is meant to give you a chance to do some structured practice with fitting linear models in R.

Data

We will use a dataset from the UCI Machine Learning Repository. This data set is about bike sharing rentals and is available at the assignment link. You can learn more about the data here. The data is available at https://www4.stat.ncsu.edu/~online/datasets/SeoulBikeData.csv

The data description describes the following variables:

- Date : **day/month/year**
- Rented Bike count - Count of bikes rented at each hour
- Hour - Hour of the day
- Temperature-Temperature in Celsius
- Humidity - %
- Windspeed - m/s
- Visibility - 10m
- Dew point temperature - Celsius
- Solar radiation - MJ/m2
- Rainfall - mm
- Snowfall - cm
- Seasons - Winter, Spring, Summer, Autumn
- Holiday - Holiday/No holiday
- Functional Day - NoFunc(Non Functional Hours), Fun(Functional hours)

Reading Data

- First read in the data
- When using `readr::read_csv()` I got an error `Error in nchar(x, "width") : invalid multibyte string, element 1`
- Google this and it is a quick fix!

```
library(tidyverse)
library(tidymodels)
bike_data <- read_csv("https://www4.stat.ncsu.edu/~online/datasets/SeoulBikeData.csv",
                      local = locale(encoding = "latin1"))
bike_data
```

```
## # A tibble: 8,760 x 14
##    Date       `Rented Bike Count`  Hour `Temperature(°C)` `Humidity(%)`
##    <chr>                    <dbl> <dbl>             <dbl>         <dbl>
##  1 01/12/2017                 254     0              -5.2            37
##  2 01/12/2017                 204     1              -5.5            38
```

```
## 3 01/12/2017          173     2              -6             39
## 4 01/12/2017          107     3              -6.2           40
## 5 01/12/2017           78     4              -6             36
## 6 01/12/2017          100     5              -6.4           37
## 7 01/12/2017          181     6              -6.6           35
## 8 01/12/2017          460     7              -7.4           38
## 9 01/12/2017          930     8              -7.6           37
## 10 01/12/2017         490     9              -6.5           27
## # i 8,750 more rows
## # i 9 more variables: `Wind speed (m/s)` <dbl>, `Visibility (10m)` <dbl>,
## #   `Dew point temperature(°C)` <dbl>, `Solar Radiation (MJ/m2)` <dbl>,
## #   `Rainfall(mm)` <dbl>, `Snowfall (cm)` <dbl>, Seasons <chr>, Holiday <chr>,
## #   `Functioning Day` <chr>
```

EDA

Checking the Data

First, let's check for missingness (there's a `tidy` way to do this but this is just a `baseR` way).

```
bike_data |>
  is.na() |>
  colSums()
```

```
##                      Date         Rented Bike Count                      Hour
##                         0                         0                         0
##           Temperature(°C)               Humidity(%)          Wind speed (m/s)
##                         0                         0                         0
##          Visibility (10m) Dew point temperature(°C)   Solar Radiation (MJ/m2)
##                         0                         0                         0
##              Rainfall(mm)             Snowfall (cm)                   Seasons
##                         0                         0                         0
##                   Holiday           Functioning Day
##                         0                         0
```

No apparent missingness. Let's check for column type and values.

```
attributes(bike_data)$spec
```

```
## cols(
##   Date = col_character(),
##   `Rented Bike Count` = col_double(),
##   Hour = col_double(),
##   `Temperature(°C)` = col_double(),
##   `Humidity(%)` = col_double(),
##   `Wind speed (m/s)` = col_double(),
##   `Visibility (10m)` = col_double(),
##   `Dew point temperature(°C)` = col_double(),
##   `Solar Radiation (MJ/m2)` = col_double(),
##   `Rainfall(mm)` = col_double(),
##   `Snowfall (cm)` = col_double(),
##   Seasons = col_character(),
```

```
##   Holiday = col_character(),
##   `Functioning Day` = col_character()
## )
```

All columns seem reasonable except the `Date` column. Let's turn that into a real date.

```
bike_data <- bike_data |>
  mutate(date = lubridate::dmy(Date)) |>
  select(-Date)
```

Now briefly summarize each column to see if there are any weird values.

```
summary(bike_data)
```

```
##  Rented Bike Count      Hour       Temperature(°C)   Humidity(%)
##  Min.   :   0.0    Min.   : 0.00   Min.   :-17.80   Min.   : 0.00
##  1st Qu.: 191.0    1st Qu.: 5.75   1st Qu.:  3.50   1st Qu.:42.00
##  Median : 504.5    Median :11.50   Median : 13.70   Median :57.00
##  Mean   : 704.6    Mean   :11.50   Mean   : 12.88   Mean   :58.23
##  3rd Qu.:1065.2    3rd Qu.:17.25   3rd Qu.: 22.50   3rd Qu.:74.00
##  Max.   :3556.0    Max.   :23.00   Max.   : 39.40   Max.   :98.00
##  Wind speed (m/s) Visibility (10m) Dew point temperature(°C)
##  Min.   :0.000    Min.   :  27     Min.   :-30.600
##  1st Qu.:0.900    1st Qu.: 940     1st Qu.: -4.700
##  Median :1.500    Median :1698     Median :  5.100
##  Mean   :1.725    Mean   :1437     Mean   :  4.074
##  3rd Qu.:2.300    3rd Qu.:2000     3rd Qu.: 14.800
##  Max.   :7.400    Max.   :2000     Max.   : 27.200
##  Solar Radiation (MJ/m2)  Rainfall(mm)     Snowfall (cm)      Seasons
##  Min.   :0.0000          Min.   : 0.0000   Min.   :0.00000   Length:8760
##  1st Qu.:0.0000          1st Qu.: 0.0000   1st Qu.:0.00000   Class :character
##  Median :0.0100          Median : 0.0000   Median :0.00000   Mode  :character
##  Mean   :0.5691          Mean   : 0.1487   Mean   :0.07507
##  3rd Qu.:0.9300          3rd Qu.: 0.0000   3rd Qu.:0.00000
##  Max.   :3.5200          Max.   :35.0000   Max.   :8.80000
##    Holiday           Functioning Day         date
##  Length:8760        Length:8760        Min.   :2017-12-01
##  Class :character   Class :character   1st Qu.:2018-03-02
##  Mode  :character   Mode  :character   Median :2018-06-01
##                                        Mean   :2018-06-01
##                                        3rd Qu.:2018-08-31
##                                        Max.   :2018-11-30
```

I don't know much about weather but things seem ok. `Visibility (10m)` is likely truncated at 2000.

Check the character columns

```
bike_data$Seasons |>
  unique()
```

```
## [1] "Winter" "Spring" "Summer" "Autumn"
```

```
bike_data$Holiday |>
  unique()
```

```
## [1] "No Holiday" "Holiday"
```

```
bike_data$`Functioning Day` |>
  unique()
```

```
## [1] "Yes" "No"
```

Ok, no worries there! Let's turn these into factor variables.

```
bike_data <- bike_data |>
  mutate(seasons = factor(Seasons),
         holiday = factor(Holiday),
         fn_day = factor(`Functioning Day`)) |>
  select(-Seasons, -Holiday, -`Functioning Day`)
```

Lastly, I think renaming the rest of the variables will be beneficial.

```
bike_data <- bike_data |>
  rename('bike_count' = `Rented Bike Count`,
         'hour' = "Hour",
         "temp" = `Temperature(°C)`,
         "wind_speed" = `Wind speed (m/s)`,
         "humidity" = `Humidity(%)`,
         "vis" = `Visibility (10m)`,
         "dew_point_temp" = `Dew point temperature(°C)`,
         "solar_radiation" = `Solar Radiation (MJ/m2)`,
         "rainfall" = "Rainfall(mm)",
         "snowfall" = `Snowfall (cm)`)
```

- Seems like the `fn_day` variable implies they were out of commission sometimes. Let's remove those observations and that variable.

```
bike_data <- bike_data |>
  filter(fn_day == "Yes") |>
  select(-fn_day)
```

To simplify our analysis, we'll summarize across the hours so that each day has one observation associated with it. Let's `group_by()` the `date`, `seasons`, and `holiday` variables and find the sum of the `bike_count`, `rainfall`, and `snowfall` variables and the mean of all the weather related variables.

```
bike_data <- bike_data |>
  group_by(date, seasons, holiday) |>
  summarize(bike_count = sum(bike_count),
            temp = mean(temp),
            humidity = mean(humidity),
            wind_speed = mean(wind_speed),
            vis = mean(vis),
```

```
            dew_point_temp = mean(dew_point_temp),
            solar_radiation = mean(solar_radiation),
            rainfall = sum(rainfall),
            snowfall = sum(snowfall)) |>
  ungroup()
```

```
bike_data
```

```
## # A tibble: 353 x 12
##    date       seasons holiday    bike_count    temp humidity wind_speed   vis
##    <date>     <fct>   <fct>           <dbl>   <dbl>    <dbl>      <dbl> <dbl>
##  1 2017-12-01 Winter  No Holiday       9539 -2.45       45.9      1.54  1871.
##  2 2017-12-02 Winter  No Holiday       8523  1.32       62.0      1.71  1471.
##  3 2017-12-03 Winter  No Holiday       7222  4.88       81.5      1.61   456.
##  4 2017-12-04 Winter  No Holiday       8729 -0.304      52.5      3.45  1363.
##  5 2017-12-05 Winter  No Holiday       8307 -4.46       36.4      1.11  1959.
##  6 2017-12-06 Winter  No Holiday       6669  0.0458     70.8      0.696 1187.
##  7 2017-12-07 Winter  No Holiday       8549  1.09       67.5      1.69   949.
##  8 2017-12-08 Winter  No Holiday       8032 -3.82       41.8      1.85  1872.
##  9 2017-12-09 Winter  No Holiday       7233 -0.846      46        1.08  1861.
## 10 2017-12-10 Winter  No Holiday       3453  1.19       69.7      2.00  1043.
## # i 343 more rows
## # i 4 more variables: dew_point_temp <dbl>, solar_radiation <dbl>,
## #   rainfall <dbl>, snowfall <dbl>
```

Summary Stats & Graphs

Some quick summary stats. We're going to focus on modeling the `bike_count` so let's focus there.

Numeric summaries first. Let's produce the mean, median, sd, IQR, min, and max for this variable. Then do the same across levels of the categorical variables.

```
bike_data |>
  summarize(across(`bike_count`,
                   .fns = c("mean" = mean,
                            "median" = median,
                            "sd" = sd,
                            "IQR" = IQR,
                            "min" = min,
                            "max" = max),
                   .names = "{.col}_{.fn}"))
```

```
## # A tibble: 1 x 6
##   bike_count_mean bike_count_median bike_count_sd bike_count_IQR bike_count_min
##             <dbl>             <dbl>         <dbl>          <dbl>          <dbl>
## 1          17485.             18563         9937.          19318            977
## # i 1 more variable: bike_count_max <dbl>
```

- Looks to be right skewed with a pretty large standard deviation.

```r
bike_data |>
  group_by(holiday) |>
  summarize(across(`bike_count`,
                   .fns = c("mean" = mean,
                            "median" = median,
                            "sd" = sd,
                            "IQR" = IQR,
                            "min" = min,
                            "max" = max),
                   .names = "{.col}_{.fn}"))
```

```
## # A tibble: 2 x 7
##   holiday    bike_count_mean bike_count_median bike_count_sd bike_count_IQR
##   <fct>                <dbl>             <dbl>         <dbl>          <dbl>
## 1 Holiday             12700.              7184        10504.          16576
## 2 No Holiday          17727.             19104.        9862.          19168.
## # i 2 more variables: bike_count_min <dbl>, bike_count_max <dbl>
```

```r
bike_data |>
  group_by(seasons) |>
  summarize(across(`bike_count`,
                   .fns = c("mean" = mean,
                            "median" = median,
                            "sd" = sd,
                            "IQR" = IQR,
                            "min" = min,
                            "max" = max),
                   .names = "{.col}_{.fn}"))
```

```
## # A tibble: 4 x 7
##   seasons bike_count_mean bike_count_median bike_count_sd bike_count_IQR
##   <fct>             <dbl>             <dbl>         <dbl>          <dbl>
## 1 Autumn           22099.             23350         6711.          10733
## 2 Spring           17910.             17590         8357.          14362.
## 3 Summer           24818.             25572.        7297.           9308.
## 4 Winter            5413.              5498         1808.           2634.
## # i 2 more variables: bike_count_min <dbl>, bike_count_max <dbl>
```

- Strong differences depending on `holiday` and `seasons`.

```r
bike_data |>
  group_by(seasons, holiday) |>
  summarize(across(`bike_count`,
                   .fns = c("mean" = mean,
                            "median" = median,
                            "sd" = sd,
                            "IQR" = IQR,
                            "min" = min,
                            "max" = max),
                   .names = "{.col}_{.fn}"))
```

```
## # A tibble: 8 x 8
```

```
## # Groups:   seasons [4]
##   seasons holiday bike_count_mean bike_count_median bike_count_sd bike_count_IQR
##   <fct>   <fct>             <dbl>             <dbl>         <dbl>          <dbl>
## 1 Autumn  Holiday          22754.             21705          5642.          5740
## 2 Autumn  No Hol~          22065.             23472          6792.         10734
## 3 Spring  Holiday          15247.             13790         10917.         10844
## 4 Spring  No Hol~          18002.             17730          8322.         14224.
## 5 Summer  Holiday          24532.             24532.         8438.          5966.
## 6 Summer  No Hol~          24824.             25572.         7324.          9165
## 7 Winter  Holiday           3759               3454.         1561.          1060.
## 8 Winter  No Hol~           5574.              5609          1757.          2564
## # i 2 more variables: bike_count_min <dbl>, bike_count_max <dbl>
```

- Differences are pretty big in the Winter and Autumn but not the other seasons. Perhaps an interaction between these two variables is important.

Now let's do some correlation.

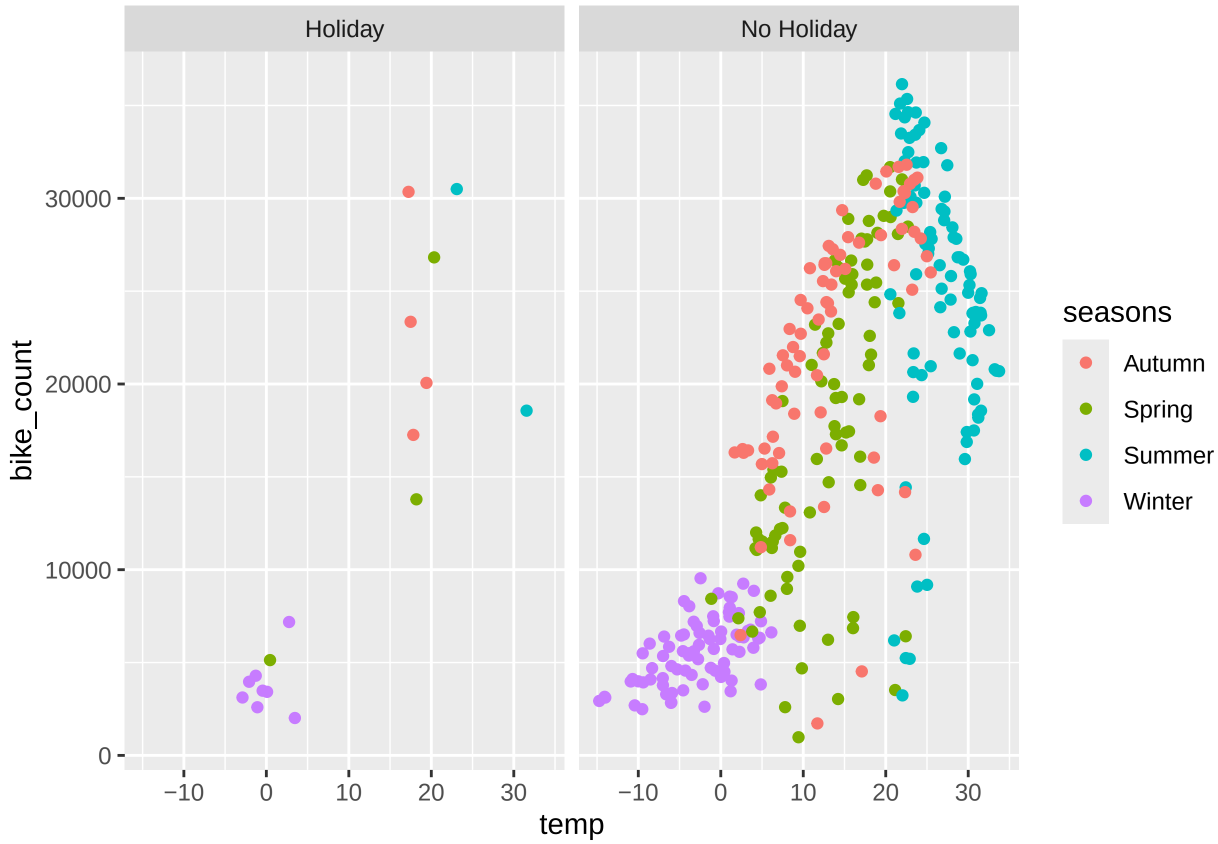
```
bike_data |>
  select(where(is.numeric)) |>
  cor() |>
  round(3)
```

```
##                 bike_count   temp humidity wind_speed    vis dew_point_temp
## bike_count           1.000  0.753    0.036     -0.193  0.166          0.650
## temp                 0.753  1.000    0.404     -0.261  0.002          0.963
## humidity             0.036  0.404    1.000     -0.234 -0.559          0.632
## wind_speed          -0.193 -0.261   -0.234      1.000  0.206         -0.288
## vis                  0.166  0.002   -0.559      0.206  1.000         -0.154
## dew_point_temp       0.650  0.963    0.632     -0.288 -0.154          1.000
## solar_radiation      0.736  0.550   -0.274      0.096  0.271          0.383
## rainfall            -0.239  0.145    0.529     -0.102 -0.222          0.265
## snowfall            -0.265 -0.267    0.065      0.021 -0.102         -0.210
##                 solar_radiation rainfall snowfall
## bike_count                0.736   -0.239   -0.265
## temp                      0.550    0.145   -0.267
## humidity                 -0.274    0.529    0.065
## wind_speed                0.096   -0.102    0.021
## vis                       0.271   -0.222   -0.102
## dew_point_temp            0.383    0.265   -0.210
## solar_radiation           1.000   -0.323   -0.233
## rainfall                 -0.323    1.000   -0.023
## snowfall                 -0.233   -0.023    1.000
```

- Definitely a few moderate relationships with `bike_count` here (`temp` and `solar_radiation`). `temp` and `dew_point_temp` are obviously pretty related. `humidity` and `vis` along with `humidity` and `dew_point_temp` as well.

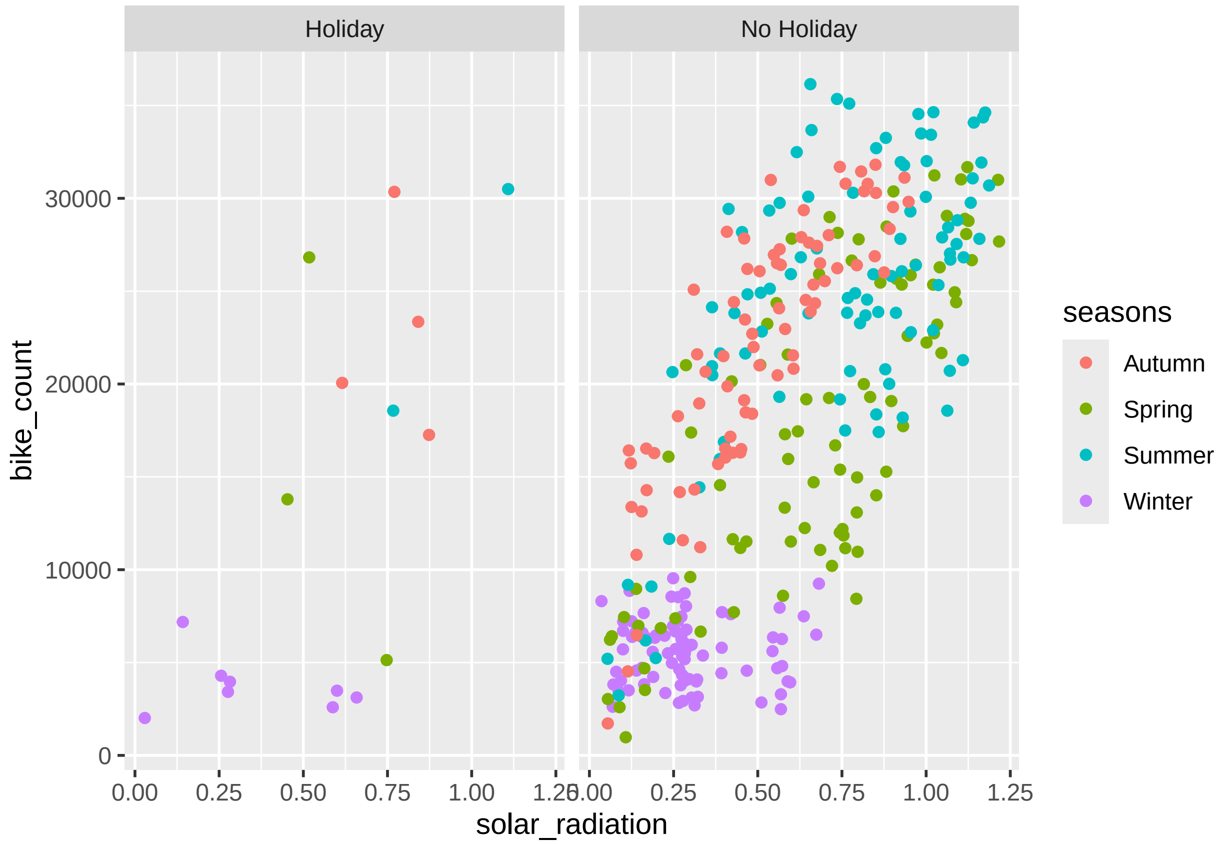
Let's do some visualizations.

```
ggplot(bike_data, aes(x = temp, y = bike_count)) +
  geom_jitter(aes(color = seasons)) +
  facet_grid(~holiday)
```



Some expected trends here and we can see that once it gets pretty hot, bike rentals slow.

```
ggplot(bike_data, aes(x = solar_radiation, y = bike_count)) +
  geom_point(aes(color = seasons)) +
  facet_grid(~holiday)
```



More solar radiation is associated with more bike rentals. (Let's just remove the ozone!)

Ok, we could keep going with these types of plots to help us understand our data but I'm going to stop there.

Split the Data

- Use functions from `tidymodels` to split the data into a training and test set (75/25 split). Use the `strata` argument to stratify the split on the `seasons`.
- On the training set, create a 10 fold CV split

```
set.seed(11)
bike_split <- initial_split(bike_data, prop = 0.75, strata = seasons)
bike_train <- training(bike_split)
bike_test <- testing(bike_split)
bike_10_fold <- vfold_cv(bike_train, 10)
```

Fitting MLR Models

First, let's create some recipes.

For the 1st recipe:

- Let's ignore the `date` variable (so we'll need to remove that or give it a different ID) but use it to create a weekday/weekend (factor) variable. (See step 2 of the `shinymodels` tutorial! You can use

step_date() then step_mutate() with an factor(if_else(...)) to create the variable. I then had to remove the intermediate variable created.)
- Let's standardize the numeric variables since their scales are pretty different.
- Let's create dummy variables for the seasons, holiday, and our new day type variable

```
MLR_rec1 <- recipe(bike_count ~ ., data = bike_train) |>
  step_date(date, features = "dow") |>
  step_mutate(day_type = factor(if_else(date_dow %in% c("Sat", "Sun"), "Weekend", "Weekday"))) |>
  step_rm(date, date_dow) |>
  step_dummy(seasons, holiday, day_type) |>
  step_normalize(all_numeric(), -bike_count)
```

For the 2nd recipe:

- Do the same steps as above.
- Add in interactions between seasons and holiday, seasons and temp, temp and rainfall. For the seasons interactions, you can use starts_with() to create the proper interactions.

```
MLR_rec2 <- MLR_rec1 |>
  step_interact(terms = ~starts_with("seasons_")*starts_with("holiday_") +
                starts_with("seasons_W")*temp +
                temp*rainfall)
MLR_rec2 <- MLR_rec1 |>
  step_interact(terms = ~starts_with("seasons_")*starts_with("holiday_") + temp*rainfall) |>
  step_interact(terms = ~seasons_Spring*temp + seasons_Winter*temp) |>
  step_interact(terms = ~seasons_Summer*temp)
```

For the 3rd recipe:

- Do the same as the 2nd recipe.
- Add in quadratic terms for each numeric predictor

```
MLR_rec3 <- MLR_rec2 |>
  step_poly(temp,
            wind_speed,
            vis,
            dew_point_temp,
            solar_radiation,
            rainfall,
            snowfall,
            degree = 2, keep_original_cols = FALSE)
```

Now we can set up our linear model fit.

```
MLR_spec <- linear_reg() |>
  set_engine("lm")
```

Fit the models using 10 fold CV and consider the training set CV error to choose a best model.

```r
MLR_CV_fit1 <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(MLR_spec) |>
  fit_resamples(bike_10_fold)
MLR_CV_fit2 <- workflow() |>
  add_recipe(MLR_rec2) |>
  add_model(MLR_spec) |>
  fit_resamples(bike_10_fold)
MLR_CV_fit3 <- workflow() |>
  add_recipe(MLR_rec3) |>
  add_model(MLR_spec) |>
  fit_resamples(bike_10_fold)
```

Get our metrics:

```r
rbind(MLR_CV_fit1 |> collect_metrics(),
      MLR_CV_fit2 |> collect_metrics(),
      MLR_CV_fit3 |> collect_metrics())
```

```
## # A tibble: 6 x 6
##   .metric .estimator     mean     n  std_err .config
##   <chr>   <chr>         <dbl> <int>    <dbl> <chr>
## 1 rmse    standard   4284.       10 165.     Preprocessor1_Model1
## 2 rsq     standard      0.822    10   0.0151 Preprocessor1_Model1
## 3 rmse    standard   3156.       10 267.     Preprocessor1_Model1
## 4 rsq     standard      0.898    10   0.0176 Preprocessor1_Model1
## 5 rmse    standard   3070.       10 213.     Preprocessor1_Model1
## 6 rsq     standard      0.903    10   0.0142 Preprocessor1_Model1
```

The last model appears to be the best! Let's fit that to the entire training set and then see how it performs on the test set.

```r
final_fit <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(MLR_spec) |>
  last_fit(bike_split)
final_fit |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard    3980.    Preprocessor1_Model1
## 2 rsq     standard       0.846 Preprocessor1_Model1
```

Obtain the final model (fit on the entire training set) coefficient table using `tidy()`.

```r
final_fit |>
  extract_fit_parsnip() |>
  tidy()
```

```
## # A tibble: 14 x 5
##    term               estimate std.error statistic   p.value
##    <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)          17446.      252.    69.3   9.38e-165
##  2 temp                 -2439.     5215.    -0.468 6.40e-  1
##  3 humidity             -1927.     1904.    -1.01  3.13e-  1
##  4 wind_speed            -523.      286.    -1.83  6.86e-  2
##  5 vis                    -63.7     361.    -0.177 8.60e-  1
##  6 dew_point_temp        7143.     6143.     1.16  2.46e-  1
##  7 solar_radiation       4088.      473.     8.64  6.74e- 16
##  8 rainfall             -1779.      333.    -5.35  2.00e-  7
##  9 snowfall              -317.      276.    -1.15  2.50e-  1
## 10 seasons_Spring       -2528.      355.    -7.12  1.14e- 11
## 11 seasons_Summer       -1670.      442.    -3.78  1.98e-  4
## 12 seasons_Winter       -3684.      501.    -7.35  2.88e- 12
## 13 holiday_No.Holiday    835.      256.     3.26  1.28e-  3
## 14 day_type_Weekend     -1050.      256.    -4.10  5.56e-  5
```

Fitting LASSO

We'll use the first recipe here.

Let's set up our model specification.

```
LASSO_spec <- linear_reg(penalty = tune(), mixture = 1) |>
  set_engine("glmnet")
```

Now we can create our workflow.

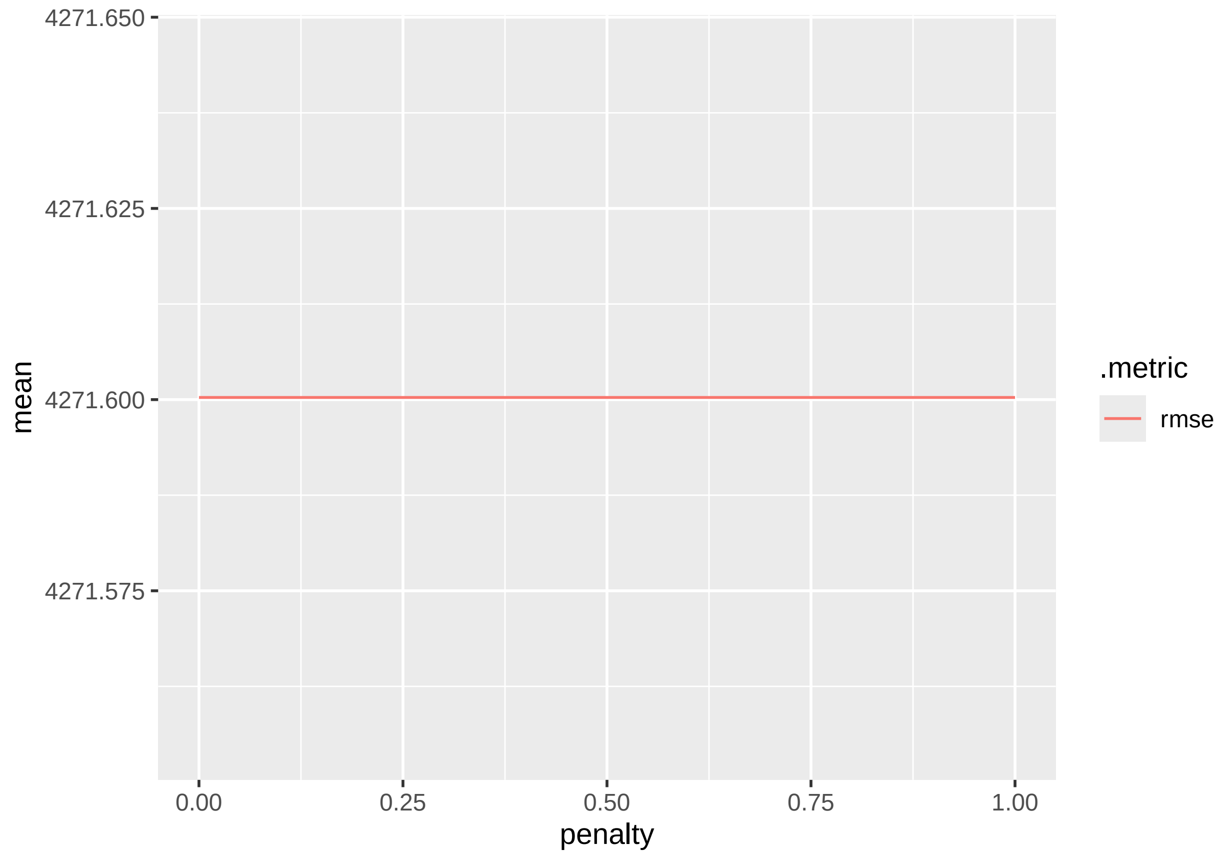
```
LASSO_wkf <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(LASSO_spec)
```

Now we'll fit the model with `tune_grid()` and `grid_regular()`.

```
LASSO_grid <- LASSO_wkf |>
  tune_grid(resamples = bike_10_fold,
            grid = grid_regular(penalty(), levels = 400))
```

Let's see which tuning parameter is the best.

```
LASSO_grid |>
  collect_metrics() |>
  filter(.metric == "rmse") |>
  ggplot(aes(penalty, mean, color = .metric)) +
  geom_line()
```



Looks like all the penalty values give the same `rmse`. That means we'll select the smallest penalty.

```
lowest_rmse <- LASSO_grid |>
  select_best(metric = "rmse")
lowest_rmse
```

```
## # A tibble: 1 x 2
##        penalty .config
##          <dbl> <chr>
## 1 0.0000000001 Preprocessor1_Model001
```

Fitting Regression Tree

Let's set up our model specification.

```
tree_mod <- decision_tree(tree_depth = tune(),
                          min_n = 20,
                          cost_complexity = tune()) |>
  set_engine("rpart") |>
  set_mode("regression")
```

Now we can create our workflow.

```r
tree_wkf  <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(tree_mod)
```

Now we'll fit the model with `tune_grid()` and `grid_regular()`.

```r
tree_grid <- grid_regular(cost_complexity(),
                          tree_depth(),
                          levels = c(10, 5))
tree_fits <- tree_wkf |>
  tune_grid(resamples = bike_10_fold,
            grid = tree_grid)
```

Let's see which tuning parameter is the best.

```r
tree_fits |>
  collect_metrics() |>
  filter(.metric == "rmse") |>
  arrange(mean)
```

```
## # A tibble: 50 x 8
##    cost_complexity tree_depth .metric .estimator  mean     n std_err .config
##              <dbl>      <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1    0.001                11 rmse    standard   3817.    10    285. Preprocess~
##  2    0.001                15 rmse    standard   3817.    10    285. Preprocess~
##  3    0.0000000001         11 rmse    standard   3837.    10    302. Preprocess~
##  4    0.000000001          11 rmse    standard   3837.    10    302. Preprocess~
##  5    0.00000001           11 rmse    standard   3837.    10    302. Preprocess~
##  6    0.0000001            11 rmse    standard   3837.    10    302. Preprocess~
##  7    0.000001             11 rmse    standard   3837.    10    302. Preprocess~
##  8    0.00001              11 rmse    standard   3837.    10    302. Preprocess~
##  9    0.0001               11 rmse    standard   3837.    10    302. Preprocess~
## 10    0.0000000001         15 rmse    standard   3837.    10    302. Preprocess~
## # i 40 more rows
```

Grab that tuning specification.

```r
lowest_rmse_tree <- tree_fits |>
  select_best(metric = "rmse")
lowest_rmse_tree
```

```
## # A tibble: 1 x 3
##   cost_complexity tree_depth .config
##             <dbl>      <int> <chr>
## 1           0.001         11 Preprocessor1_Model38
```

Fitting Bagged Tree

Let's set up our model specification.

```r
bag_spec <- bag_tree(tree_depth = 5, min_n = 10, cost_complexity = tune()) |>
  set_engine("rpart") |>
  set_mode("regression")
```

Now we can create our workflow.

```r
library(baguette)
bag_wkf  <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(bag_spec)
```

Now we'll fit the model with `tune_grid()` and `grid_regular()`.

```r
bag_fit <- bag_wkf |>
  tune_grid(resamples = bike_10_fold,
            grid = grid_regular(cost_complexity(),
                                levels = 15))
```

Let's see which tuning parameter is the best.

```r
bag_fit |>
  collect_metrics() |>
  filter(.metric == "rmse") |>
  arrange(mean)
```

```
## # A tibble: 15 x 7
##    cost_complexity .metric .estimator  mean     n std_err .config
##              <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
##  1        1.64e- 7 rmse    standard   3210.    10    168. Preprocessor1_Model06
##  2        1.39e- 5 rmse    standard   3247.    10    194. Preprocessor1_Model09
##  3        8.48e- 9 rmse    standard   3265.    10    173. Preprocessor1_Model04
##  4        7.20e- 7 rmse    standard   3271.    10    179. Preprocessor1_Model07
##  5        6.11e- 5 rmse    standard   3279.    10    182. Preprocessor1_Model10
##  6        3.16e- 6 rmse    standard   3338.    10    185. Preprocessor1_Model08
##  7        4.39e-10 rmse    standard   3341.    10    168. Preprocessor1_Model02
##  8        1.18e- 3 rmse    standard   3353.    10    192. Preprocessor1_Model12
##  9        2.68e- 4 rmse    standard   3364.    10    168. Preprocessor1_Model11
## 10        3.73e- 8 rmse    standard   3371.    10    188. Preprocessor1_Model05
## 11        1.93e- 9 rmse    standard   3413.    10    202. Preprocessor1_Model03
## 12        1   e-10 rmse    standard   3471.    10    182. Preprocessor1_Model01
## 13        5.18e- 3 rmse    standard   3489.    10    172. Preprocessor1_Model13
## 14        2.28e- 2 rmse    standard   3980.    10    135. Preprocessor1_Model14
## 15        1   e- 1 rmse    standard   4961.    10    173. Preprocessor1_Model15
```

Get the best one

```r
lowest_rmse_bag <- bag_fit |>
  select_best(metric = "rmse")
lowest_rmse_bag
```

```
## # A tibble: 1 x 2
##   cost_complexity .config
##             <dbl> <chr>
## 1     0.000000164 Preprocessor1_Model06
```

Fitting Random Forest

```
rf_spec <- rand_forest(mtry = tune()) |>
  set_engine("ranger",importance = "impurity") |>
  set_mode("regression")
```

Now we can create our workflow.

```
rf_wkf  <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(rf_spec)
```

Now we'll fit the model with `tune_grid()` and `grid_regular()`.

```
rf_fit <- rf_wkf |>
  tune_grid(resamples = bike_10_fold,
            grid = 12)
```

Let's see which tuning parameter is the best.

```
rf_fit |>
  collect_metrics() |>
  filter(.metric == "rmse") |>
  arrange(mean)
```

```
## # A tibble: 9 x 7
##    mtry .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    13 rmse    standard   2979.    10    196. Preprocessor1_Model1
## 2    11 rmse    standard   3006.    10    200. Preprocessor1_Model2
## 3     9 rmse    standard   3019.    10    196. Preprocessor1_Model6
## 4    10 rmse    standard   3030.    10    192. Preprocessor1_Model5
## 5     7 rmse    standard   3045.    10    193. Preprocessor1_Model3
## 6     6 rmse    standard   3057.    10    196. Preprocessor1_Model9
## 7     4 rmse    standard   3145.    10    188. Preprocessor1_Model4
## 8     2 rmse    standard   3477.    10    171. Preprocessor1_Model8
## 9     1 rmse    standard   4784.    10    172. Preprocessor1_Model7
```

Get the best one

```
lowest_rmse_rf <- rf_fit |>
  select_best(metric = "rmse")
lowest_rmse_rf
```

```
## # A tibble: 1 x 2
##    mtry .config
##   <int> <chr>
## 1    13 Preprocessor1_Model1
```

Getting the `mae` for the Final Models and Other Summaries

Refit the MLR model and get all the `rmse` and `mae`.

```
final_fit <- workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(MLR_spec) |>
  last_fit(bike_split, metrics = metric_set(rmse, mae))
final_fit |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3980. Preprocessor1_Model1
## 2 mae     standard       3039. Preprocessor1_Model1
```

Let's get the coefficient table too.

```
workflow() |>
  add_recipe(MLR_rec1) |>
  add_model(MLR_spec) |>
  fit(bike_train) |>
  tidy()
```

```
## # A tibble: 14 x 5
##    term               estimate std.error statistic   p.value
##    <chr>                 <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)          17446.      252.    69.3   9.38e-165
##  2 temp                 -2439.     5215.    -0.468 6.40e-  1
##  3 humidity             -1927.     1904.    -1.01  3.13e-  1
##  4 wind_speed            -523.      286.    -1.83  6.86e-  2
##  5 vis                    -63.7     361.    -0.177 8.60e-  1
##  6 dew_point_temp        7143.     6143.     1.16  2.46e-  1
##  7 solar_radiation       4088.      473.     8.64  6.74e- 16
##  8 rainfall             -1779.      333.    -5.35  2.00e-  7
##  9 snowfall              -317.      276.    -1.15  2.50e-  1
## 10 seasons_Spring       -2528.      355.    -7.12  1.14e- 11
## 11 seasons_Summer       -1670.      442.    -3.78  1.98e-  4
## 12 seasons_Winter       -3684.      501.    -7.35  2.88e- 12
## 13 holiday_No.Holiday     835.      256.     3.26  1.28e-  3
## 14 day_type_Weekend     -1050.      256.    -4.10  5.56e-  5
```

Let's fit our LASSO model to our training data and test on the test set, specifying we want `rmse` and `mae`.

```
LASSO_final <- LASSO_wkf |>
  finalize_workflow(lowest_rmse) |>
  last_fit(bike_split, metrics = metric_set(rmse, mae))
LASSO_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3999. Preprocessor1_Model1
## 2 mae     standard       3063. Preprocessor1_Model1
```

Get the coefficient table too.

```r
LASSO_wkf |>
  finalize_workflow(lowest_rmse) |>
  fit(bike_train) |>
  tidy()
```

```
## # A tibble: 14 x 3
##    term               estimate      penalty
##    <chr>                 <dbl>        <dbl>
##  1 (Intercept)          17446. 0.0000000001
##  2 temp                   389. 0.0000000001
##  3 humidity              -887. 0.0000000001
##  4 wind_speed            -522. 0.0000000001
##  5 vis                      0  0.0000000001
##  6 dew_point_temp        3752. 0.0000000001
##  7 solar_radiation       4065. 0.0000000001
##  8 rainfall             -1841. 0.0000000001
##  9 snowfall              -336. 0.0000000001
## 10 seasons_Spring       -2505. 0.0000000001
## 11 seasons_Summer       -1607. 0.0000000001
## 12 seasons_Winter       -3653. 0.0000000001
## 13 holiday_No.Holiday     820. 0.0000000001
## 14 day_type_Weekend     -1060. 0.0000000001
```

Let's fit our regression tree model to our training data and test on the test set, specifying we want `rmse` and `mae`.

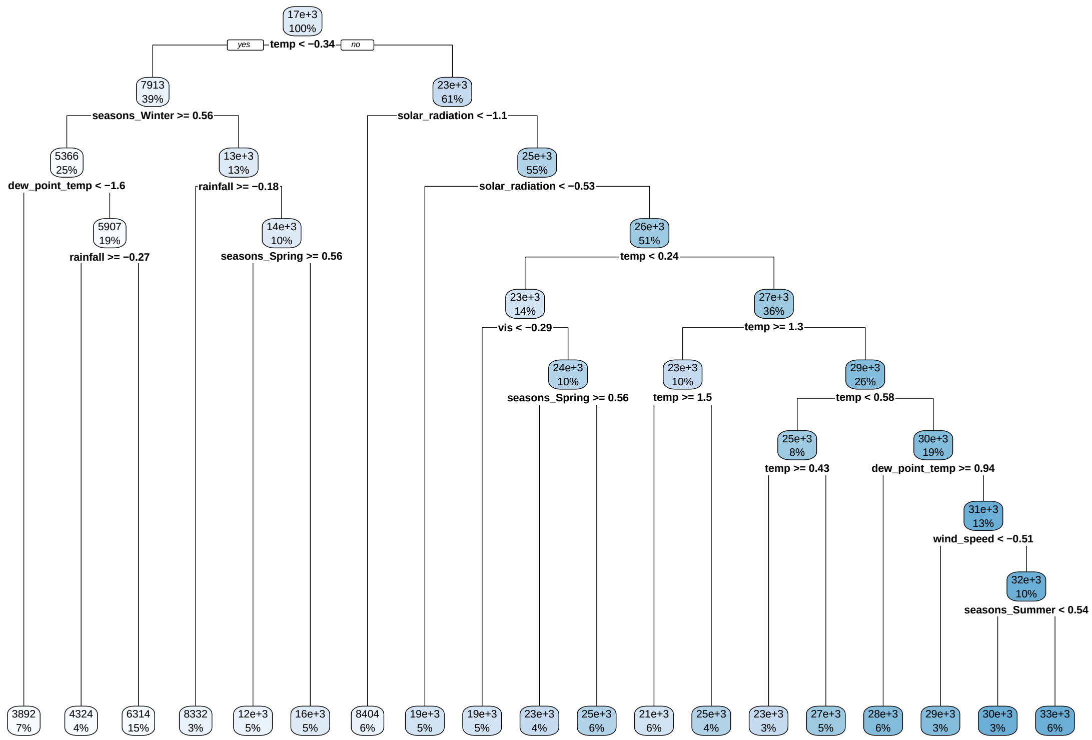
```r
tree_final <- tree_wkf |>
  finalize_workflow(lowest_rmse_tree) |>
  last_fit(bike_split, metrics = metric_set(rmse, mae))
tree_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3096. Preprocessor1_Model1
## 2 mae     standard       2362. Preprocessor1_Model1
```

Plot the final fit.

```r
tree_final_train_fit <- extract_workflow(tree_final)

tree_final_train_fit %>%
  extract_fit_engine() %>%
  rpart.plot::rpart.plot(roundint = FALSE)
```



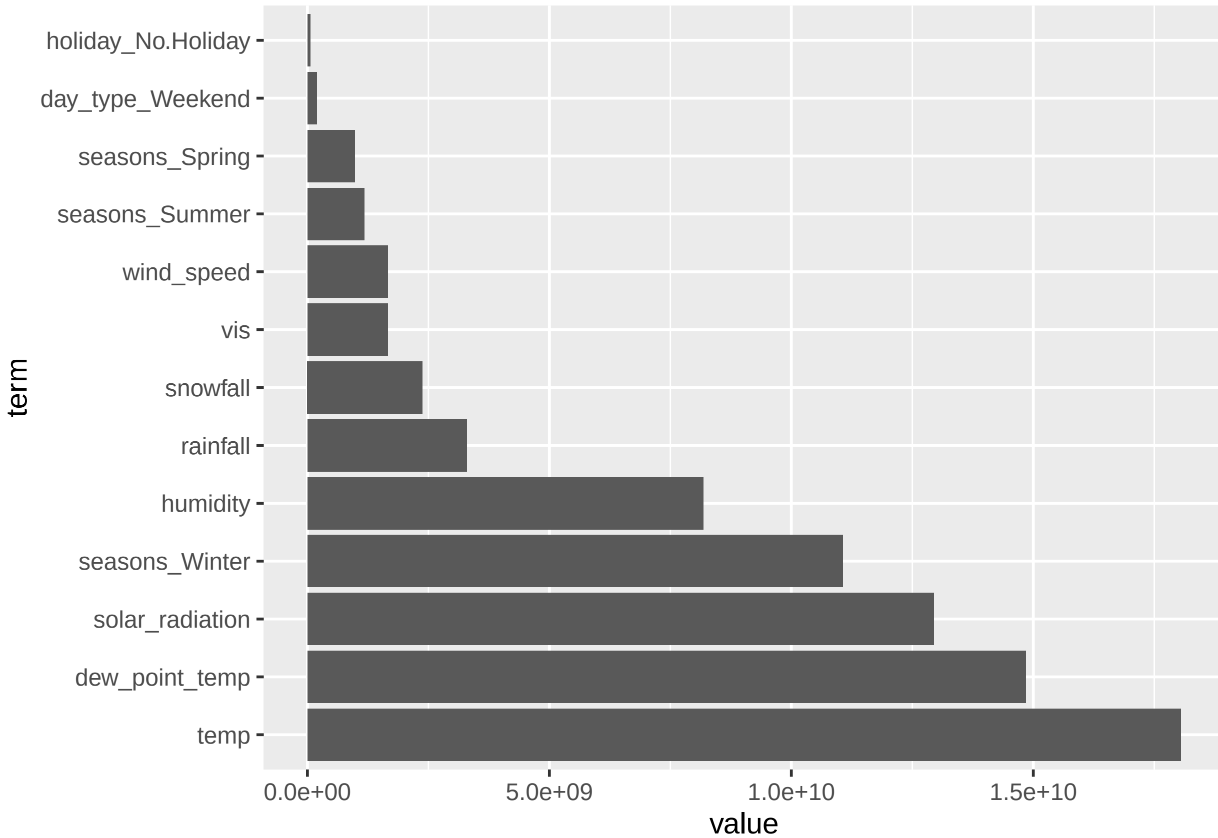
Let's fit our bagged tree model to our training data and test on the test set, specifying we want `rmse` and `mae`.

```
bag_final <- bag_wkf |>
  finalize_workflow(lowest_rmse_bag) |>
  last_fit(bike_split, metrics = metric_set(rmse, mae))
bag_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3221. Preprocessor1_Model1
## 2 mae     standard       2516. Preprocessor1_Model1
```

Plot the variable importance.

```
bag_final_model <- extract_fit_engine(bag_final)
bag_final_model$imp |>
  mutate(term = factor(term, levels = term)) |>
  ggplot(aes(x = term, y = value)) +
  geom_bar(stat ="identity") +
  coord_flip()
```



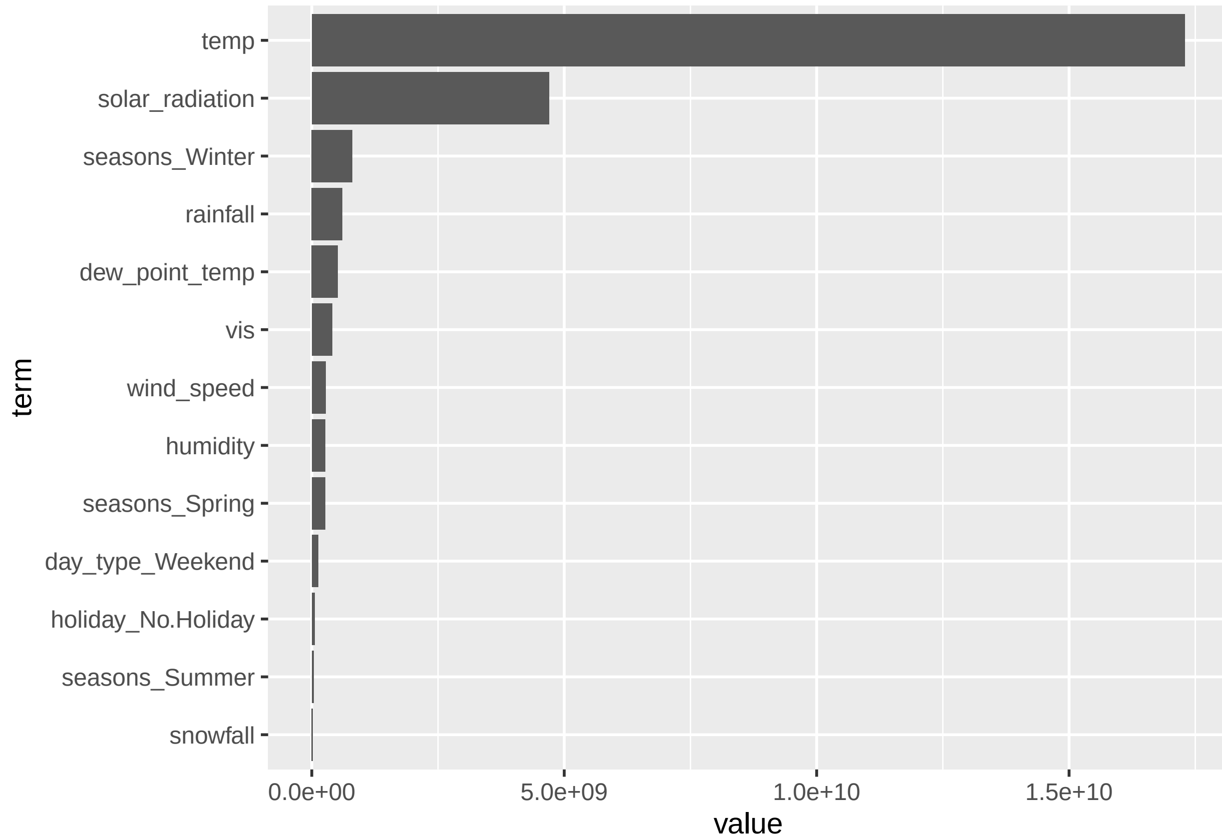
Let's fit our rf model to our training data and test on the test set, specifying we want `rmse` and `mae`.

```
rf_final <- rf_wkf |>
  finalize_workflow(lowest_rmse_rf) |>
  last_fit(bike_split, metrics = metric_set(rmse, mae))
rf_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       2621. Preprocessor1_Model1
## 2 mae     standard       2109. Preprocessor1_Model1
```

Plot the variable importance.

```
rf_final_model <- extract_fit_engine(rf_final)
tibble(term = names(rf_final_model$variable.importance),
       value = rf_final_model$variable.importance) |>
  arrange(value) |>
  mutate(term = factor(term, levels = term)) |>
  ggplot(aes(x = term, y = value)) +
  geom_bar(stat ="identity") +
  coord_flip()
```



Wrap up

Just to have all the test set metrics together:

```
LASSO_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3999. Preprocessor1_Model1
## 2 mae     standard       3063. Preprocessor1_Model1
```

```
tree_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3096. Preprocessor1_Model1
## 2 mae     standard       2362. Preprocessor1_Model1
```

```r
bag_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       3221. Preprocessor1_Model1
## 2 mae     standard       2516. Preprocessor1_Model1
```

```r
rf_final |>
  collect_metrics()
```

```
## # A tibble: 2 x 4
##   .metric .estimator .estimate .config
##   <chr>   <chr>          <dbl> <chr>
## 1 rmse    standard       2621. Preprocessor1_Model1
## 2 mae     standard       2109. Preprocessor1_Model1
```

Random forest is best on the test set. Let's fit it to the entire data set.

```r
best_model <- rf_wkf |>
  finalize_workflow(lowest_rmse_rf) |>
  fit(bike_data)
```

```r
best_model
```

```
## == Workflow [trained] ==========================================================
## Preprocessor: Recipe
## Model: rand_forest()
##
## -- Preprocessor ----------------------------------------------------------------
## 5 Recipe Steps
##
## * step_date()
## * step_mutate()
## * step_rm()
## * step_dummy()
## * step_normalize()
##
## -- Model -----------------------------------------------------------------------
## Ranger result
##
## Call:
##  ranger::ranger(x = maybe_data_frame(x), y = y, mtry = min_cols(~13L,      x), importance = ~"impuri
##
## Type:                             Regression
## Number of trees:                  500
## Sample size:                      353
## Number of independent variables:  13
## Mtry:                             13
## Target node size:                 5
```

```
## Variable importance mode:         impurity
## Splitrule:                        variance
## OOB prediction error (MSE):       7591965
## R squared (OOB):                  0.9231172
```