101_wk_6_regression_models

Seung Hyun Sung

11/21/2021

DS4B 101-R: R FOR BUSINESS ANALYSIS —-

REGRESSION MODELS —-

GOAL: BUILD PREDICTION MODEL FOR PRICING ALGORITHM

Regression: A technique to predict numeric values of a feature in a dataset

parsnip: A package that provides an API to many powerful modeling algorithms in R (modeling package). An interface that standardizes the making of models in R.

- Idea: One issue with different functions available in R that do the same thing is that they can have different interfaces and arguments. The parsnip interface solves this issue by providing consistency.
- engines: an engine is the underlying algorithm that you are connecting parsnip to

recipes: Preprocessing

- making statistical transformations piror to modeling

rsample: Sampling & Cross Validation

- Includes various sampling methods

yardstick: Model Metrics (e.g. MAE, RMSE)

- Get model metrics for model comparison

LIBRARIES & DATA —-

```r
# Standard
library(readxl)
library(tidyverse)
library(tidyquant)

# Modeling
library(parsnip)

# Preprocessing & Sampling
library(recipes)
library(rsample)

# Modeling Error Metrics
library(yardstick)

# Plotting Decision Trees
library(rpart.plot)

library(tidymodels)
```

Source Scripts

```r
source("01_scripts/separate_bikes_and_outlier_detection.R")
```

Read Data

```r
bike_orderlines_tbl <- read_rds("~/Desktop/University_business_science/DS4B_101/00_data/bike_sales/data_

glimpse(bike_orderlines_tbl)
```

```
## Rows: 15,644
## Columns: 13
## $ order_date     <dttm> 2011-01-07, 2011-01-07, 2011-01-10, 2011-01-10, 2011-0~
## $ order_id       <dbl> 1, 1, 2, 2, 3, 3, 3, 3, 3, 4, 5, 5, 5, 5, 6, 6, 6, 6, 7~
## $ order_line     <dbl> 1, 2, 1, 2, 1, 2, 3, 4, 5, 1, 1, 2, 3, 4, 1, 2, 3, 4, 1~
## $ quantity       <dbl> 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 1, 1, 1, 1, 1, 1, 1~
## $ price          <dbl> 6070, 5970, 2770, 5970, 10660, 3200, 12790, 5330, 1570,~
## $ total_price    <dbl> 6070, 5970, 2770, 5970, 10660, 3200, 12790, 5330, 1570,~
## $ model          <chr> "Jekyll Carbon 2", "Trigger Carbon 2", "Beast of the Ea~
## $ category_1     <chr> "Mountain", "Mountain", "Mountain", "Mountain", "Road",~
## $ category_2     <chr> "Over Mountain", "Over Mountain", "Trail", "Over Mounta~
## $ frame_material <chr> "Carbon", "Carbon", "Aluminum", "Carbon", "Carbon", "Ca~
## $ bikeshop_name  <chr> "Ithaca Mountain Climbers", "Ithaca Mountain Climbers",~
## $ city           <chr> "Ithaca", "Ithaca", "Kansas City", "Kansas City", "Loui~
## $ state          <chr> "NY", "NY", "KS", "KS", "KY", "KY", "KY", "KY", "KY", "~
```

1.0 PROBLEM DEFINITION —-

- Which Bike Categories are in high demand?
- Which Bike Categories are under represented?
- GOAL: Use a pricing algorithm to determine a new product price in a category gap

```
model_sales_tbl <- bike_orderlines_tbl %>%
    select(total_price, model, category_2, frame_material) %>%

    group_by(model, category_2, frame_material) %>%
    summarise(total_sales = sum(total_price)) %>%
    ungroup() %>%

    arrange(desc(total_sales))
```

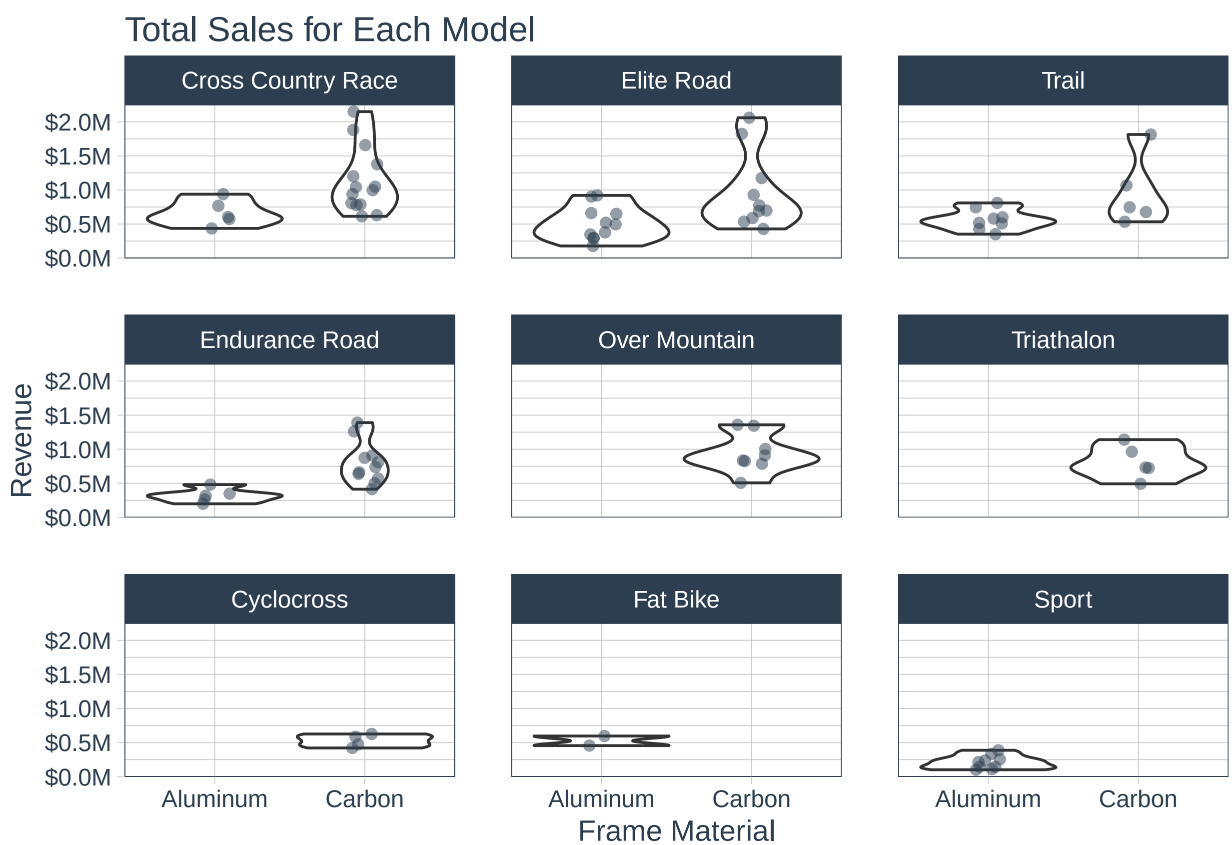
```
## ‘summarise()‘ has grouped output by ’model’, ’category_2’. You can override using the ‘.groups‘ argu
```

Comment:

- Not all categories have an Aluminum Option
- As Sport model is most basic model, lunching a new Carbon framed Sport model probability would not be an effective business opportunity. Conversely, products with "High-End" (Carbon Option) only could be missing out (e.g. Over Mountain model with only Carbon frame model). This might be a logical areas for us to focus on.
- **Product Gap!** By all means, learning to identify product gaps and convert them into **business opportunities** which will help the business growth. Simply is a market segment that existing businesses are not yet serving. One could find this opportunity & discover the not served or future demand internally or externally via market research and by analysing google trends - **Market Gap**
    - Using Google Trends to Find Niches
    - Find Relevant Product Categories in Related Topics
    - Using Google Trends for Keyword Research
    - Promote Your Store Around Seasonal Trends
    - Using Google Trends for Content Freshness
    - Create Content About Current Trends
    - Find Niche Topics by Region
    - Monitor Competitors' Position with Google Trends Compare
    - Google Trends YouTube
    - Google Trends Google Shopping
- Once one found the product gap, in here, we can focus on product like "Over Mountain" and "Triathalon" for new product development, its profitability needs to be carefully verified and modeling using the pricing algorithm. The basic statistics of the out put price, e.g. mean, variance, and pairwise comparison, intra-cluster correlation will all need to take account for consideration. When done right, one could potentially generate significant revenue by filling in the gaps.

```
model_sales_tbl %>%
    mutate(category_2 = as_factor(category_2) %>%
               # reorder by max revenue generating model
               fct_reorder(total_sales, .fun = max) %>%
               fct_rev()) %>%
```

```
ggplot(aes(frame_material, total_sales)) +
geom_violin() +
geom_jitter(width = 0.1, alpha = 0.5, color = "#2c3e50") +
#coord_flip() +
facet_wrap(~ category_2) +
scale_y_continuous(labels = scales::dollar_format(scale = 1e-6, suffix = "M", accuracy = 0.1)) +
theme_tq() +
labs(
    title = "Total Sales for Each Model",
    x = "Frame Material", y = "Revenue"
)
```



2.0 TRAINING & TEST SETS —-

Comment:

ID: Tells us which rows are being sampled

Target (price): What we are trying to predict

Predictors (all other columns): Data that our algorithms will use to build relationships to target.

- Data manipulation:
  - Adding row_number - ID prior to data partitioning.

  - Operate separate_bike_model pre-made function - Data wrangled. Our custom function for adding engineered features built from the "model" column.
  - It can be effective to make feature engineered custom function as wrangling specific parts of the column is organised approach + more interpretable.

```
bike_features_tbl <- bike_orderlines_tbl %>%
    # category_1 is not needed because category_2 comprises it
    select(price, model, category_2, frame_material) %>%

    distinct() %>%

    mutate(id = row_number()) %>%

    select(id, everything()) %>%
    # separate_bike_model: Data frame function
    # both argument set TRUE, as we want to keep all the columns including model
    separate_bike_model(keep_model_column = T, append = T)

bike_features_tbl
```

```
## # A tibble: 97 x 14
##       id price model     category_2  frame_material model_base model_tier  black
##    <int> <dbl> <chr>     <chr>       <chr>          <chr>      <chr>       <dbl>
##  1     1  6070 Jekyll C~ Over Mount~ Carbon         Jekyll     Carbon 2        0
##  2     2  5970 Trigger ~ Over Mount~ Carbon         Trigger    Carbon 2        0
##  3     3  2770 Beast of~ Trail       Aluminum       Beast of ~ 1               0
##  4     4 10660 Supersix~ Elite Road  Carbon         Supersix ~ Hi-Mod Team     0
##  5     5  3200 Jekyll C~ Over Mount~ Carbon         Jekyll     Carbon 4        0
##  6     6 12790 Supersix~ Elite Road  Carbon         Supersix ~ Black Inc.      1
##  7     7  5330 Supersix~ Elite Road  Carbon         Supersix ~ Hi-Mod Dur~     0
##  8     8  1570 Synapse ~ Endurance ~ Aluminum       Synapse    Disc 105        0
##  9     9  4800 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Dis~     0
## 10    10   480 Catalyst~ Sport       Aluminum       Catalyst   3               0
## # ... with 87 more rows, and 6 more variables: hi_mod <dbl>, team <dbl>,
## #   red <dbl>, ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

Pro Tip: [1] Splitting into training & test sets helps to **prevent over-fitting** and improve model **generalization**, the ability for a model to perform well on future data. Even better is **cross-validation**.

[2] Generalisation is the goal, as we do not want the algorithm to train only on the data we see, but also with new data that can be generated for the future.

[3] The model_base feature has 18 levels. A random split may not have all levels in the training set, which is bad. We can try to prevent this by adding as model_base as a stratafication variable.

Comment:

- strata: The strata argument causes the random sampling to be conducted within the stratification variable. The can help ensure that the number of data points in the analysis data is equivalent to the proportions in the original data set.

```
set.seed(42)

split_obj <- rsample::initial_split(bike_features_tbl, prop = 0.8, strata = "model_base")
```

```r
bike_features_tbl %>% distinct(model_base)
```

```
## # A tibble: 18 x 1
##    model_base
##    <chr>
##  1 Jekyll
##  2 Trigger
##  3 Beast of the East
##  4 Supersix Evo
##  5 Synapse
##  6 Catalyst
##  7 F-Si
##  8 SuperX
##  9 Slice
## 10 Habit
## 11 CAAD12
## 12 Scalpel 29
## 13 Scalpel-Si
## 14 CAAD8
## 15 Trail
## 16 Bad Habit
## 17 Fat CAAD1
## 18 Fat CAAD2
```

```r
# Have all 18 objects from model_base column -> managed to avoid concern!
split_obj %>% training() %>% distinct(model_base)
```

```
## # A tibble: 18 x 1
##    model_base
##    <chr>
##  1 Trigger
##  2 Beast of the East
##  3 Supersix Evo
##  4 Jekyll
##  5 Slice
##  6 CAAD12
##  7 Scalpel-Si
##  8 CAAD8
##  9 SuperX
## 10 Habit
## 11 Trail
## 12 Catalyst
## 13 Scalpel 29
## 14 Synapse
## 15 F-Si
## 16 Bad Habit
## 17 Fat CAAD1
## 18 Fat CAAD2
```

```r
# Have all 12 objects from model_base column -> having less object on the testing set is not as severe
split_obj %>% testing() %>% distinct(model_base)
```

```
## # A tibble: 14 x 1
##    model_base
##    <chr>
##  1 Jekyll
##  2 Supersix Evo
##  3 Catalyst
##  4 F-Si
##  5 SuperX
##  6 Scalpel 29
##  7 Scalpel-Si
##  8 Slice
##  9 Synapse
## 10 CAAD12
## 11 Trigger
## 12 CAAD8
## 13 Trail
## 14 Habit
```

```
train_tbl <- split_obj %>% training()
test_tbl <- split_obj %>% testing()
```

*** FIX 1 *** —-

Error: factor model_base has new levels Fat CAAD2

- Need to move Fat CAAD2 from test to training set because model doesn't know how to handle

a new category that is unseen in the training data

```
train_tbl <- train_tbl %>%
    bind_rows(
        test_tbl %>% filter(model_base %>% str_detect("Fat CAAD2"))
    )

test_tbl <- test_tbl %>%
    filter(!model_base %>% str_detect("Fat CAAD2"))
```

*** END FIX 1 *** —-

linear_reg: Creates a parsnip specification for a linear regression * engines: + lm (default) + glmnet + stan + spark + keras

- The penalty & mixture argument (tunable variable)
  - lm: Does not have any adjustable parameters

3.0 LINEAR METHODS —-

```
?linear_reg
?set_engine
?fit
```

```
## Help on topic 'fit' was found in the following packages:
##
##   Package               Library
##   parsnip               /Library/Frameworks/R.framework/Versions/4.1-arm64/Resources/library
##   generics              /Library/Frameworks/R.framework/Versions/4.1-arm64/Resources/library
##
##
## Using the first match ...
```

```
?predict.model_fit
?metrics
```

Parsnip API: Three Steps

- [1] Create a model linear_reg()
- [2] Set a engine set_engine()
- [3] Fit the model to data fit()

3.1 LINEAR REGRESSION - NO ENGINEERED FEATURES —-

Comment:

- Model Metrics: we calculate model metrics comparing the test data predictions with the actual values to get a baseline model performance
- Residuals: Difference between actual (truth) and prediction 9estimate from the model)
- MAE: Mean Absolute Error
  - Absolute value of residuals generates the magnitude of error
  - Take the averages to get the average error

Model Interpretability for Linear Model - How it works

- Dealing with Categorical Predictors

Making a Linear Regression interpretable is very easy provided all of the predictors are categorical (meaning any numeric predictors need to be binned so their values are interpreted in the model as low/med/high).

- Dealing with Numeric Predictors

For numeric features, we just need to bin them prior to performing the regression to make them categorical in bins like High, Medium, Low. Otherwise, the coefficients will be in the wrong scale for the numeric predictors. We saw in the course how to do this - You have a few options, and my preference is using a mutate() with case_when(). You can find the price at the 33th and 66th quantile with quantile(probs = c(0.33, 0.66)). Then just use those values to create low, medium, high categories.

Our models in the course have categorical data, but in the future just remember that numeric predictors will need to be converted to categorical (i.e. binned) prior to making an interpretable plot like the plot above.

3.1.1 Model —-

```
# leave the penalty and mixture NULL for now: as our operation focus is lm
model_01_linear_lm_simple <- linear_reg(mode = "regression") %>%
    set_engine("lm") %>%
    fit(price ~ category_2 + frame_material, data = train_tbl)

model_01_linear_lm_simple %>% class()
```

```
## [1] "_lm"       "model_fit"
```

```
# [1] "_lm"       "model_fit"

model_01_linear_lm_simple %>%
    predict(new_data = test_tbl) %>%

    bind_cols(test_tbl %>% select(price)) %>%
    # residuals = actual(Y) - predict(Y^)
    mutate(residuals = price - .pred) %>%
    summarise(
        # abs: absolute value as we only want the magnitude not direction
        mae = abs(residuals) %>% mean(),
        # 1. square. 2. average. 3. taking the sqaure root
        rmse = mean(residuals^2)^0.5
    )
```

```
## # A tibble: 1 x 2
##     mae  rmse
##   <dbl> <dbl>
## 1 2248. 3108.
```

```
model_01_linear_lm_simple %>%
    predict(new_data = test_tbl) %>%

    bind_cols(test_tbl %>% select(price)) %>%
    yardstick::metrics(truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    3108.
## 2 rsq     standard       0.492
## 3 mae     standard    2248.
```

3.1.2 Feature Importance —-

Model Terms & Coefficients (Estimates) In a linear model, the predicitors become terms. The algorithm then assigns coefficients (estimate) to the terms.

These estimates are:

- If term is categorical, estimate is in units of the output
- If term is numeric, need to multiply by the term's state
- Prediction = c1* Term1 + c2* Term2 + c3* Term3 + … + Intercept (the value that all predictions start with)

Example: Price of Carbon Triathalon Bike Price = intercept + c1* frame_materialCarbon + c2* category2_Triathalon = $2,639 + $3,659* (1) + (-$2,427)* (1) = $3,871

Model coefficient for Numeric VS categorical: categorical data is always converted to binary. This makes interpretation very easy because coefficients are in terms of the output.

Numeric data will be in values of the numeric feature. If numeric feature was weight, the coefficient will be in dollars per unit weight.

Comment:

- fit: the model from the stats::lm() function is stored in the "fit" element
- parsnip as a n interface: The parsnip functions provide consistent wrappers around modelling functions in the R modelling package ecosystem.
- The broom package for lm has all available methods: tidy & glance & augment

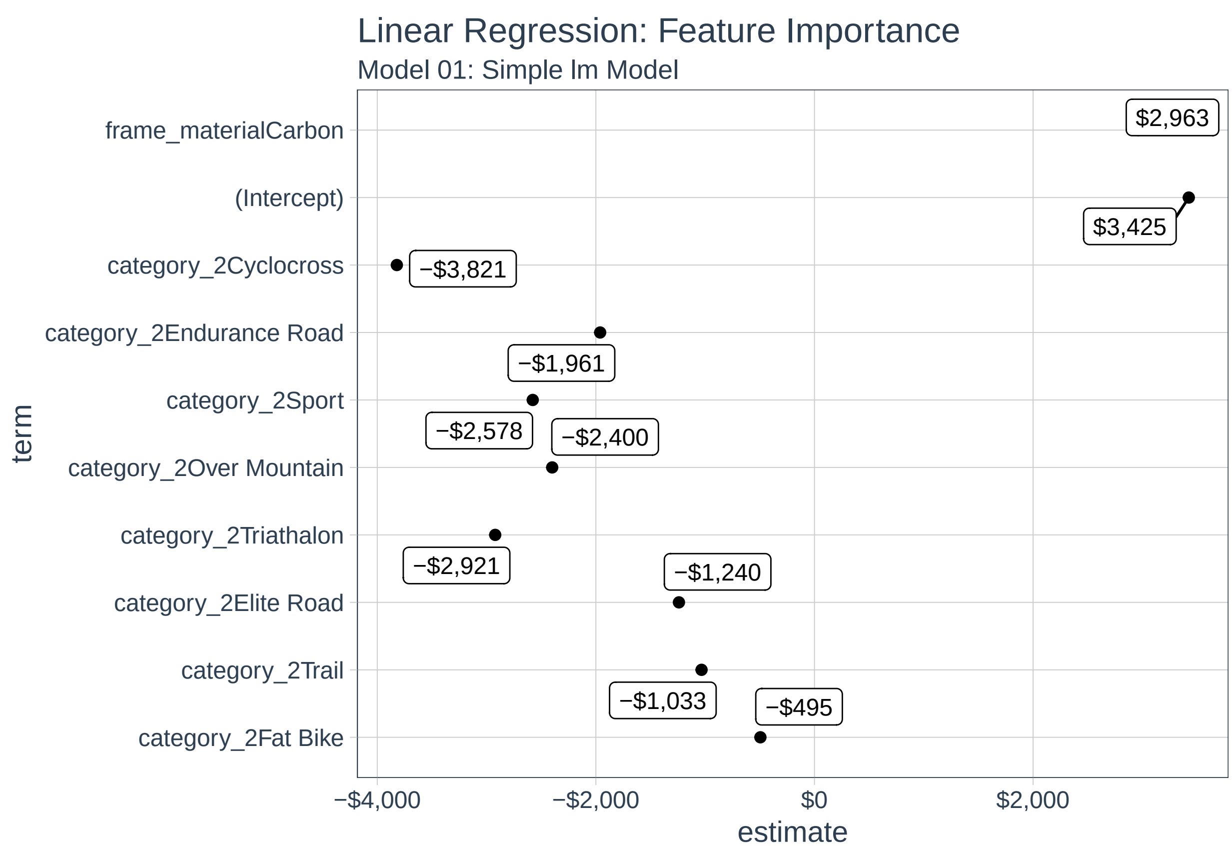
```
model_01_linear_lm_simple
```

```
## parsnip model object
##
## Fit time:  2ms
##
## Call:
## stats::lm(formula = price ~ category_2 + frame_material, data = data)
##
## Coefficients:
##              (Intercept)      category_2Cyclocross      category_2Elite Road
##                   3424.7                   -3820.9                   -1239.9
## category_2Endurance Road        category_2Fat Bike   category_2Over Mountain
##                  -1960.8                    -494.7                   -2399.5
##         category_2Sport           category_2Trail      category_2Triathalon
##                  -2577.6                   -1033.2                   -2920.9
##     frame_materialCarbon
##                   2962.8
```

```
model_01_linear_lm_simple$fit %>% class()
```

```
## [1] "lm"
```

```r
model_01_linear_lm_simple$fit %>%
    broom::tidy() %>%
    arrange(p.value) %>%
    mutate(term = as_factor(term) %>% fct_rev()) %>%

    ggplot(aes(x = estimate, y = term)) +
    geom_point() +
    # accuracy = 1, removes cents (decimal palces)
    ggrepel::geom_label_repel(aes(label = scales::dollar(estimate, accuracy = 1)),
                              size = 3,
                              max.overlaps = getOption("ggrepel.max.overlaps", default = 100)) +
    scale_x_continuous(labels = scales::dollar_format()) +
    labs(title = "Linear Regression: Feature Importance",
         subtitle = "Model 01: Simple lm Model") +
    theme_tq()
```



This is our baseline model. We will try to improve the performance of the model and see how the feature importance and their weights changes in the process of adding complexity.

3.1.3 Function to Calculate Metrics —-

comments:

- The feature importance will change between the model to model

- The process of the metrics to predict the model performance (bind_cols and yardsticks). Hence we will make this metrics process into function to compress the workflow and reduce repeatiable work.

```
model_01_linear_lm_simple %>%
    predict(new_data = test_tbl) %>%

    bind_cols(test_tbl %>% select(price)) %>%
    yardstick::metrics(truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    3108.
## 2 rsq     standard       0.492
## 3 mae     standard    2248.
```

```
calc_metrics <- function(model, new_data = test_tbl){
    model %>%
    predict(new_data = new_data) %>%

    bind_cols(new_data %>% select(price)) %>%
    yardstick::metrics(truth = price, estimate = .pred)
}

model_01_linear_lm_simple %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    3108.
## 2 rsq     standard       0.492
## 3 mae     standard    2248.
```

3.2 LINEAR REGRESSION - WITH ENGINEERED FEATURES —-

3.2.1 Model —-

Pro Tip: number 1 way to improve model performance is: Including better features! Spend max time here. Moving to advanced models won't help you if you dont have a good features. As a rule of thumb, supplying appropriate additional features that trains the model will always out perform models that selected from bunch of different algorithm. -> without changing the algorithm

```
model_02_linear_lm_complex <- linear_reg(mode = "regression") %>%
    set_engine("lm") %>%
    fit(price ~., data = train_tbl %>% select(-c(id, model, model_tier)))

# Goodness of fit (RSS) metrics on Training Set
model_01_linear_lm_simple$fit %>% glance()
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic     p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>       <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.489         0.420 1939.      7.13 0.000000362     9  -687. 1396. 1421.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
model_02_linear_lm_complex$fit %>% glance()
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.squared sigma statistic  p.value    df logLik   AIC   BIC
##       <dbl>         <dbl> <dbl>     <dbl>    <dbl> <dbl>  <dbl> <dbl> <dbl>
## 1     0.902         0.854  973.      18.8 7.56e-18    25  -623. 1300. 1364.
## # ... with 3 more variables: deviance <dbl>, df.residual <int>, nobs <int>
```

```
# Goodness of fit metrics on Testing Set
model_02_linear_lm_complex %>% calc_metrics(new_data = test_tbl)
```

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

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2009.
## 2 rsq     standard       0.758
## 3 mae     standard    1335.
```

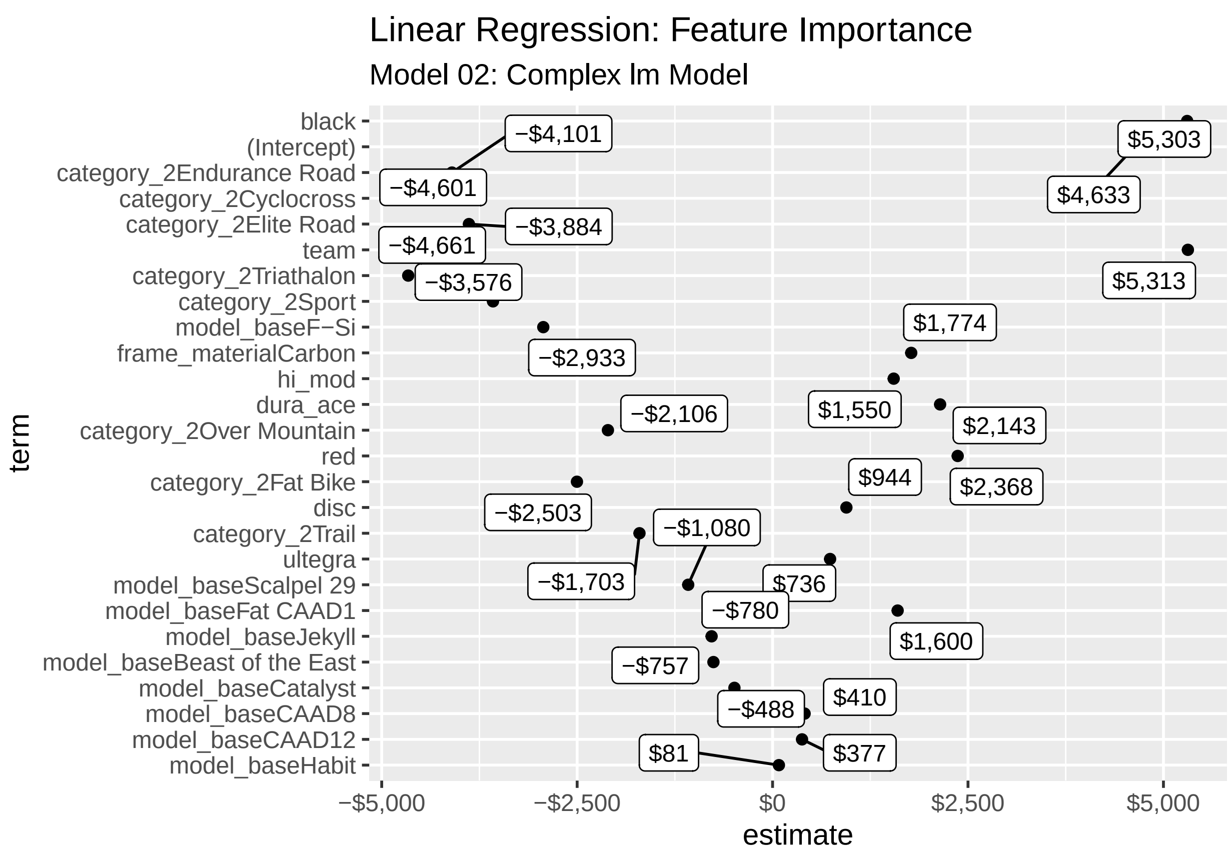
3.2.2 Feature importance —-

```
model_02_linear_lm_complex$fit %>%
    broom::tidy() %>% head(20)
```

```
## # A tibble: 20 x 5
##    term                         estimate std.error statistic   p.value
##    <chr>                           <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)                     4633.      578.     8.02   1.35e-10
##  2 category_2Cyclocross           -4601.      713.    -6.45   3.97e- 8
##  3 category_2Elite Road           -3884.      612.    -6.35   5.76e- 8
##  4 category_2Endurance Road       -4101.      603.    -6.80   1.13e- 8
##  5 category_2Fat Bike             -2503.     1131.    -2.21   3.15e- 2
##  6 category_2Over Mountain        -2106.      698.    -3.02   3.96e- 3
##  7 category_2Sport                -3576.      755.    -4.74   1.78e- 5
##  8 category_2Trail                -1703.      898.    -1.90   6.37e- 2
##  9 category_2Triathalon           -4661.      804.    -5.80   4.25e- 7
## 10 frame_materialCarbon            1774.      432.     4.11   1.44e- 4
## 11 model_baseBeast of the East     -757.      888.    -0.852  3.98e- 1
## 12 model_baseCAAD12                 377.      733.     0.514  6.10e- 1
## 13 model_baseCAAD8                  410.      794.     0.516  6.08e- 1
## 14 model_baseCatalyst              -488.      743.    -0.657  5.14e- 1
```

```
## 15 model_baseF-Si              -2933.       650.   -4.51    3.82e- 5
## 16 model_baseFat CAAD1          1600.       1376.    1.16    2.50e- 1
## 17 model_baseFat CAAD2            NA          NA    NA        NA
## 18 model_baseHabit                80.8       818.    0.0988  9.22e- 1
## 19 model_baseJekyll              -780.       888.   -0.878   3.84e- 1
## 20 model_baseScalpel 29         -1080.       698.   -1.55    1.28e- 1
```

```
model_02_linear_lm_complex$fit %>%
    broom::tidy() %>%
    filter(complete.cases(.)) %>%
    arrange(p.value) %>%
    mutate(term = as_factor(term) %>% fct_rev()) %>%

    ggplot(aes(x = estimate, y = term)) +
    geom_point() +
    ggrepel::geom_label_repel(aes(label = scales::dollar(estimate, accuracy = 1)),
                              size = 3) +
    scale_x_continuous(labels = scales::dollar_format()) +
    labs(title = "Linear Regression: Feature Importance",
         subtitle = "Model 02: Complex lm Model")
```



3.3 PENALIZED REGRESSION —-

3.3.1 Model —-

Regularization: A penalty factor that is applied to columns that are present but that have lower predictive value.

Model parameter:

- penalty: A non-negative number representing the total amount of regularization (specific engines only).
- mixture: A number between zero and one (inclusive) that is the proportion of L1 regularization (i.e. lasso) in the model. When mixture = 1, it is a pure lasso model while mixture = 0 indicates that ridge regression is being used (specific engines only).

```r
?linear_reg
?glmnet::glmnet

model_03_linear_glmnet <- linear_reg(mode = "regression", penalty = 200, mixture = 0.1) %>%
    set_engine("glmnet") %>%
    fit(price ~ ., data = train_tbl %>% select(-id, -model, -model_tier))


model_03_linear_glmnet %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2074.
## 2 rsq     standard       0.749
## 3 mae     standard    1405.
```

3.3.2 Feature Importance —-

Hyper Parameter Tuning: Systematically adjusting the model parameters to optimize the performance

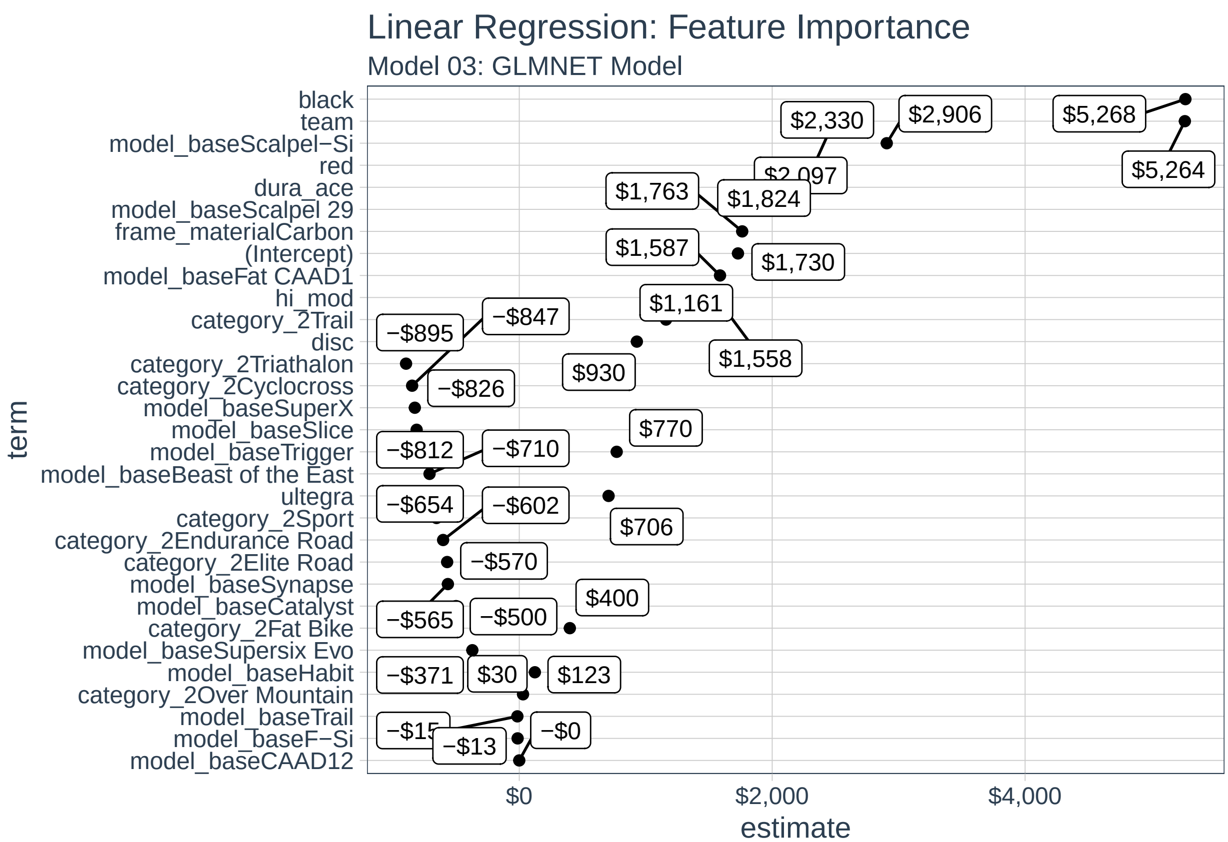
Grid Search: A popular hyperparameter tuning method of producing a "grid" that has many combinations of parameters

In statistics, deviance is a goodness-of-fit statistic for a statistical model; it is often used for statistical hypothesis testing. It is a generalization of the idea of using the sum of squares of residuals (RSS) in ordinary least squares to cases where model-fitting is achieved by maximum likelihood. It plays an important role in exponential dispersion models and generalized linear models.

https://www.youtube.com/watch?v=UaU-WZIf8c4 https://www.youtube.com/watch?v=9lRv01HDU0s

```r
model_03_linear_glmnet$fit %>%
    broom::tidy() %>%
    # *** FIX 2 *** ----
    # Problem: glmnet returns all lambda (penalty) values assessed
    # Solution: Filter to the max dev.ratio, which is the "optimal" lambda according to glmnet
    filter(dev.ratio == max(dev.ratio)) %>%
```

```r
    # *** END FIX 2 *** ----

    arrange(desc(abs(estimate))) %>%
    mutate(term = as_factor(term) %>% fct_rev()) %>%

    ggplot(aes(x = estimate, y = term)) +
    geom_point() +
    ggrepel::geom_label_repel(aes(label = scales::dollar(estimate, accuracy = 1)),
                              size = 3) +
    scale_x_continuous(labels = scales::dollar_format()) +
    labs(title = "Linear Regression: Feature Importance",
         subtitle = "Model 03: GLMNET Model") +
    theme_tq()
```



4.0 TREE-BASED METHODS —-

4.1 DECISION TREES —-

- Tree-based model acquire minimal pre-processing
- Random Forest & XGBoost:
  - Pro: High performance
  - Con: Less Explainability

4.1.1 Model —-

Avoid Overfitting: Tree-based methods can beome over-fit if we let the nodes contain too few data points or the trees to grow too large. (aka. high variance)

Avoid Underfitting: if tree is too shallow or too many data points are required per node, tree becomes under-fit (aka. high bias)

purrr: should be used for hypertuning optimization

```
?decision_tree
?rpart::rpart

model_04_tree_decision_tree <- decision_tree(mode = "regression",
              cost_complexity = 0.001,
              tree_depth      = 7,
              min_n           = 6) %>%
    set_engine("rpart") %>%
    fit(price ~ ., data = train_tbl %>% select(-id, -model, -model_tier))

model_04_tree_decision_tree %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    1890.
## 2 rsq     standard       0.777
## 3 mae     standard    1346.
```

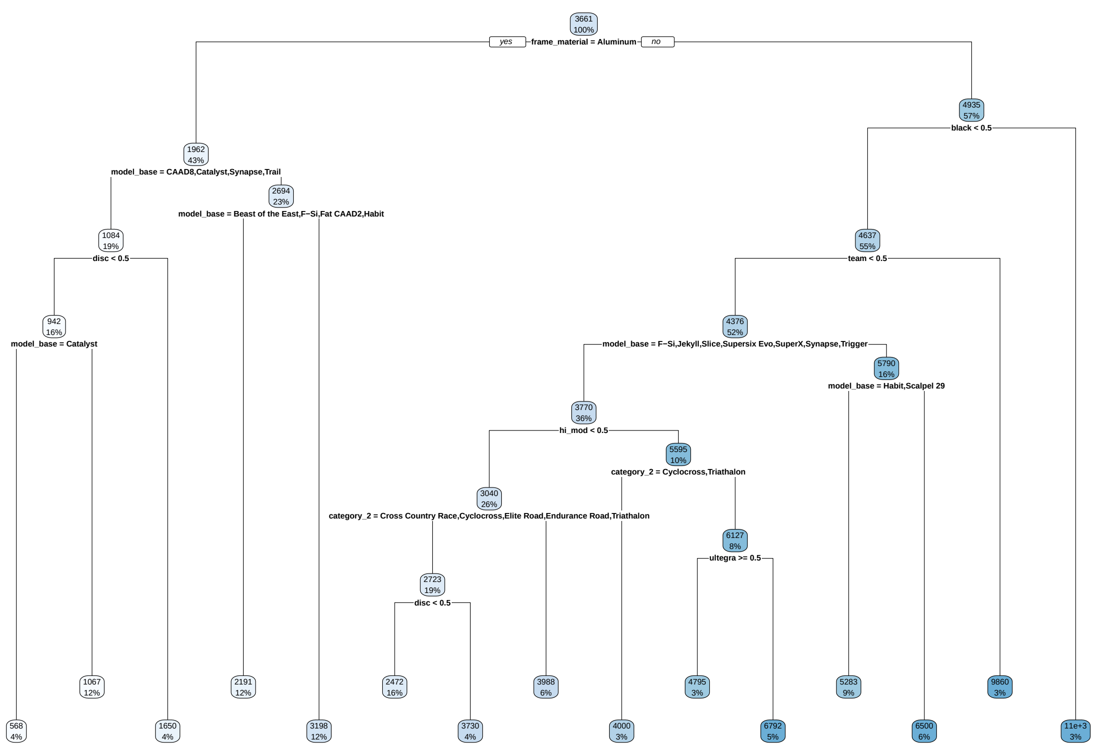
4.1.2 Decision Tree Plot —-

```
?rpart.plot()

model_04_tree_decision_tree$fit
```

```
## n= 77
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##   1) root 77 492967600.00  3660.8440
##     2) frame_material=Aluminum 33  43325950.00  1962.2730
##       4) model_base=CAAD8,Catalyst,Synapse,Trail 15   2606923.00  1083.6670
##         8) disc< 0.5 12   1007373.00   942.0833
##          16) model_base=Catalyst 3     42466.67   568.3333 *
##          17) model_base=CAAD8,Synapse,Trail 9    406150.00  1066.6670 *
##         9) disc>=0.5 3    396800.00  1650.0000 *
##       5) model_base=Bad Habit,Beast of the East,CAAD12,F-Si,Fat CAAD1,Fat CAAD2,Habit 18  19490440.0
##        10) model_base=Beast of the East,F-Si,Fat CAAD2,Habit 9   1364089.00  2191.1110 *
##        11) model_base=Bad Habit,CAAD12,Fat CAAD1 9  13566160.00  3197.7780 *
##     3) frame_material=Carbon 44 283024500.00  4934.7730
```

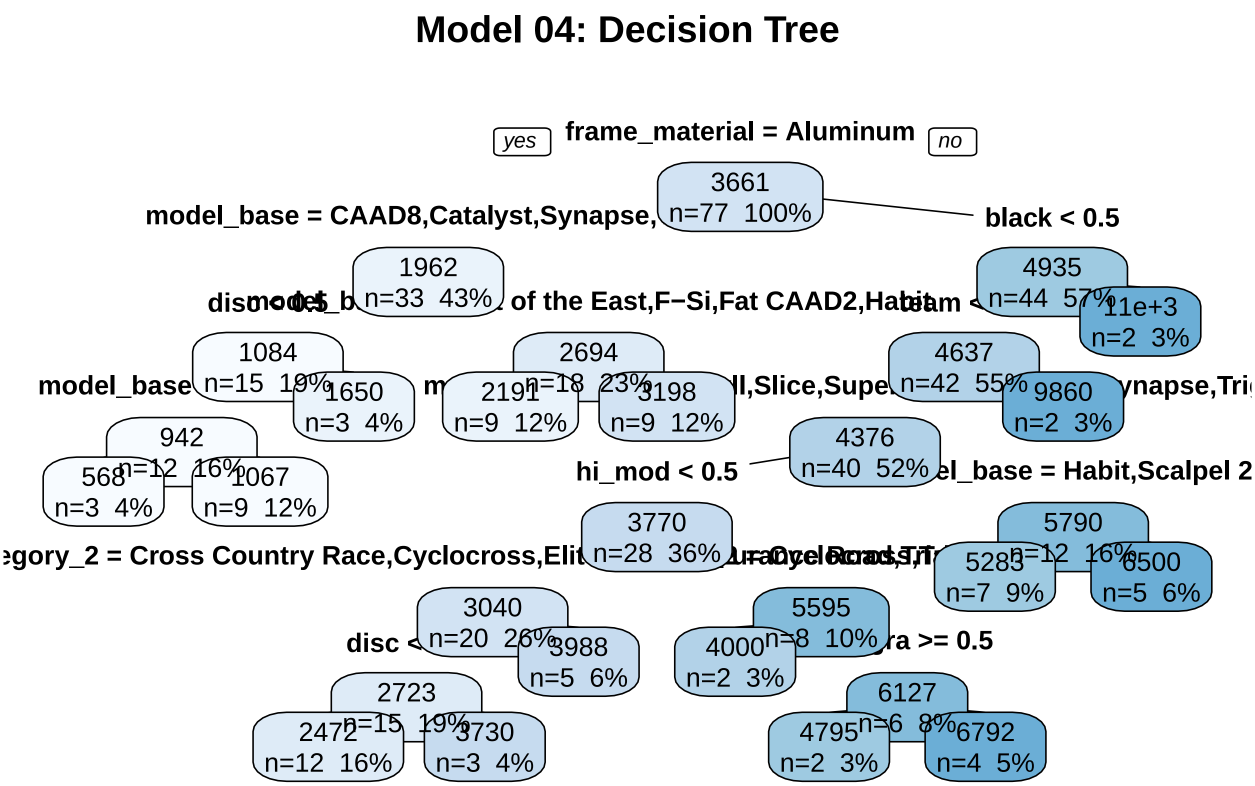
```
##       6) black< 0.5 42 195922300.00  4636.9050
##        12) team< 0.5 40 137352800.00  4375.7500
##          24) model_base=F-Si,Jekyll,Slice,Supersix Evo,SuperX,Synapse,Trigger 28  74798300.00  3769.
##            48) hi_mod< 0.5 20  20123700.00  3039.5000
##              96) category_2=Cross Country Race,Cyclocross,Elite Road,Endurance Road,Triathalon 15   8
##               192) disc< 0.5 12   2777367.00  2471.6670 *
##               193) disc>=0.5 3   2289800.00  3730.0000 *
##              97) category_2=Over Mountain 5   5258680.00  3988.0000 *
##            49) hi_mod>=0.5 8  17357000.00  5595.0000
##              98) category_2=Cyclocross,Triathalon 2    500000.00  4000.0000 *
##              99) category_2=Cross Country Race,Elite Road,Endurance Road 6  10072930.00  6126.6670
##               198) ultegra>=0.5 2    572450.00  4795.0000 *
##               199) ultegra< 0.5 4   4180475.00  6792.5000 *
##          25) model_base=Habit,Scalpel 29,Scalpel-Si 12  28267000.00  5790.0000
##            50) model_base=Habit,Scalpel 29 7  10072340.00  5282.8570 *
##            51) model_base=Scalpel-Si 5  13873800.00  6500.0000 *
##        13) team>=0.5 2   1280000.00  9860.0000 *
##       7) black>=0.5 2   5120000.00 11190.0000 *
```

```
model_04_tree_decision_tree$fit %>%
    # roundint = FALSE to avoid warning: model=TRUE needed to include y(price) object to the model
    rpart.plot(roundint = FALSE)
```

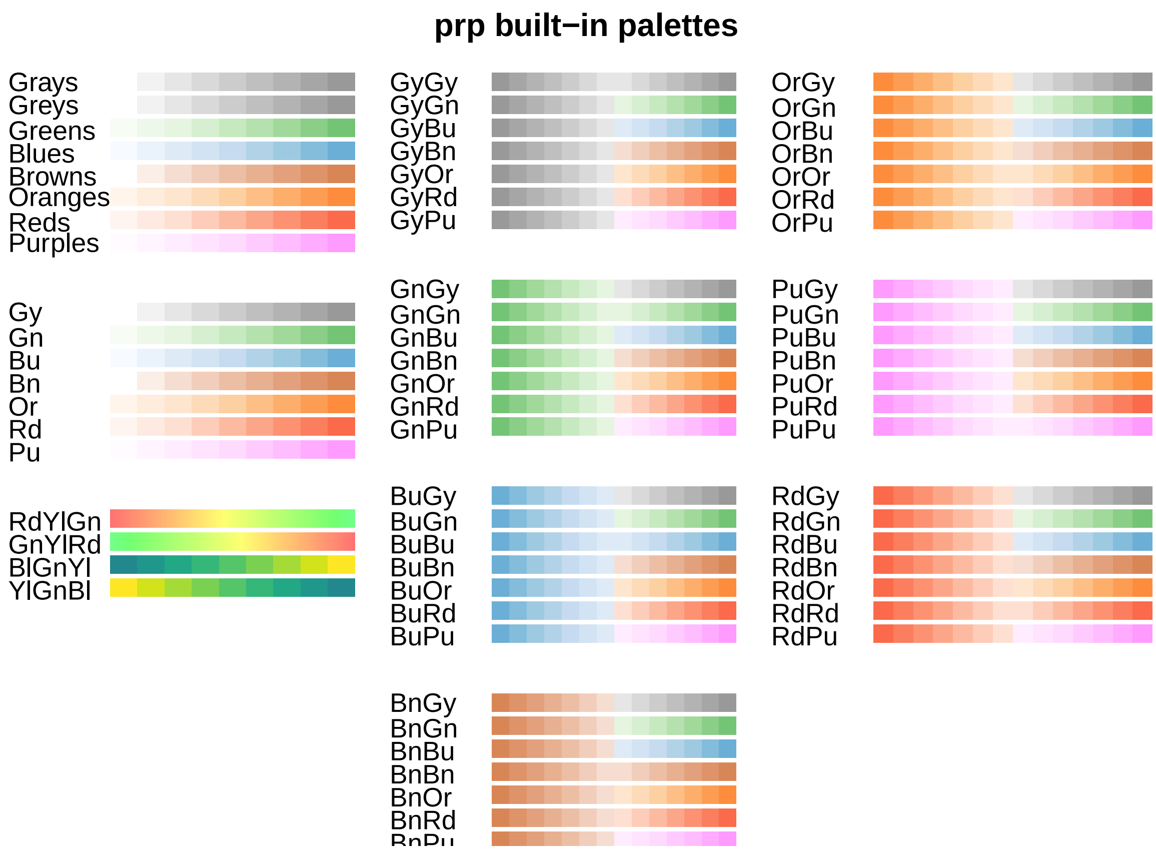



```
model_04_tree_decision_tree$fit %>%
    rpart.plot(
```

```
        roundint = FALSE,
        # type of node display
        type = 1,
        # number of percentage display on node
        extra = 101,
        fallen.leaves = FALSE,
        cex = 0.8,
        main = "Model 04: Decision Tree",
        box.palette = "Blues"
        )
```




```
show.prp.palettes()
```



4.2 RANDOM FOREST —-

4.2.1 Model: ranger —-

Reproducibility the results will be different as it is a random forest proess, and we did not make this reproducible. Don't worry we will show you how to make the random forest reproducible in a couple lectures

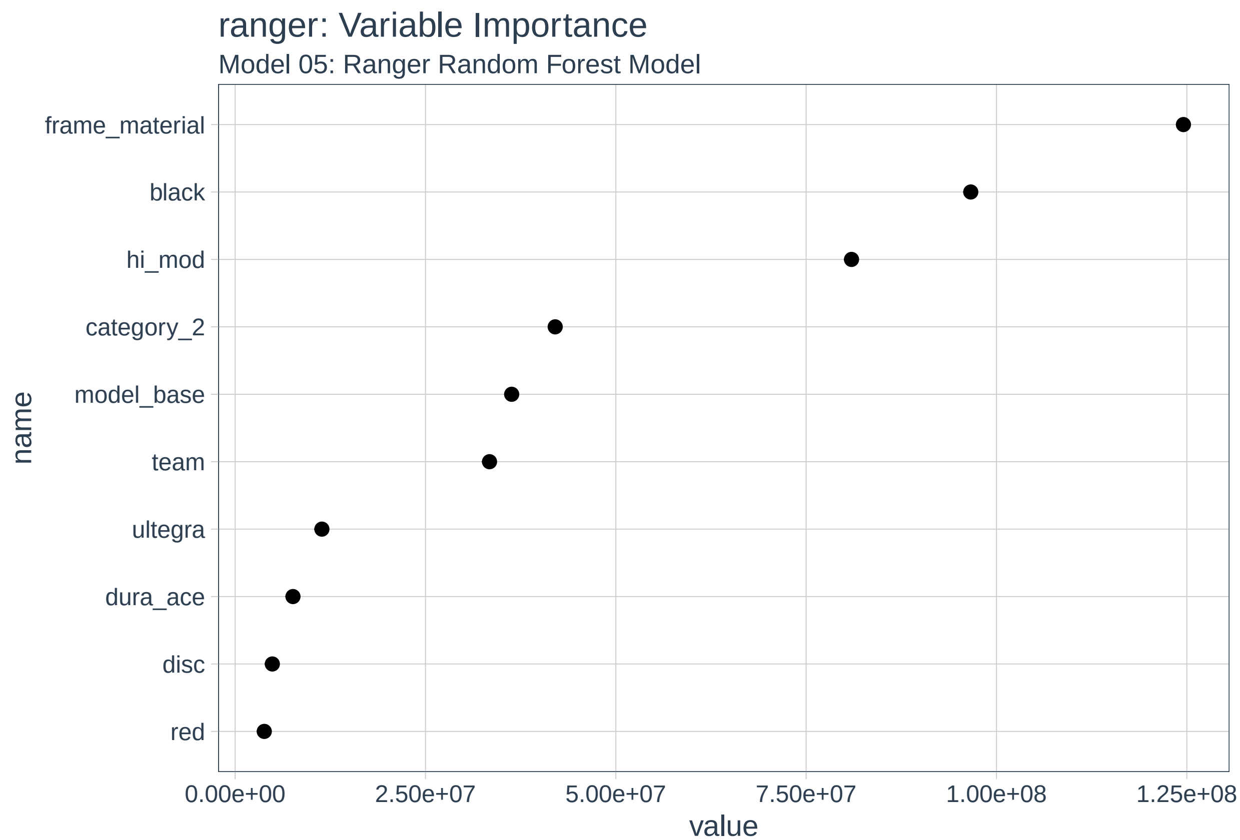
```
set.seed(1234)
model_05_rand_forest_ranger <- rand_forest(
    mode = "regression",
    mtry = 8,
    trees = 2000,
    min_n = 5
    ) %>%
    set_engine("ranger",
               replace = TRUE,
               splitrule = "extratrees",
               importance = "impurity") %>%
    fit(price ~ .,
        data = train_tbl %>% select(-id, -model, -model_tier) %>%
            mutate_if(is.character, factor))

model_05_rand_forest_ranger %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2110.
## 2 rsq     standard       0.769
## 3 mae     standard    1580.
```

4.2.2 ranger: Feature Importance —-

enframe(): turns a list or vector into a tibble with the names = names of the list and the values = the contents of the list

```
model_05_rand_forest_ranger$fit %>%
    ranger::importance() %>%
    enframe() %>%
    arrange(desc(value)) %>%
    mutate(name = as_factor(name) %>% fct_rev()) %>%

    ggplot(aes(value, name)) +
    geom_point(size = 2) +
    labs(title = "ranger: Variable Importance",
         subtitle = "Model 05: Ranger Random Forest Model") +
    theme_tq()
```



4.2.3 Model randomForest —-

*** FIX 3 *** —-

- Encodings have changed for randomForest. Must use factors for character data

- Error: New factor levels not present in the training data

```
set.seed(38)
model_06_rand_forest_randomForest <- rand_forest("regression") %>%
    set_engine("randomForest") %>%
    fit(price ~ ., data = train_tbl %>% dplyr::select(-id, -model, -model_tier))
```

Solution:

```
set.seed(1234)
model_06_rand_forest_randomForest <- rand_forest("regression") %>%
    set_engine("randomForest") %>%
    fit(price ~ .,
        data = train_tbl %>%
            select(-id, -model, -model_tier) %>%
            mutate_if(is.character, factor) # <- This internally tells randomForest to convert characte
        )

model_06_rand_forest_randomForest %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2661.
## 2 rsq     standard       0.741
## 3 mae     standard    1711.
```

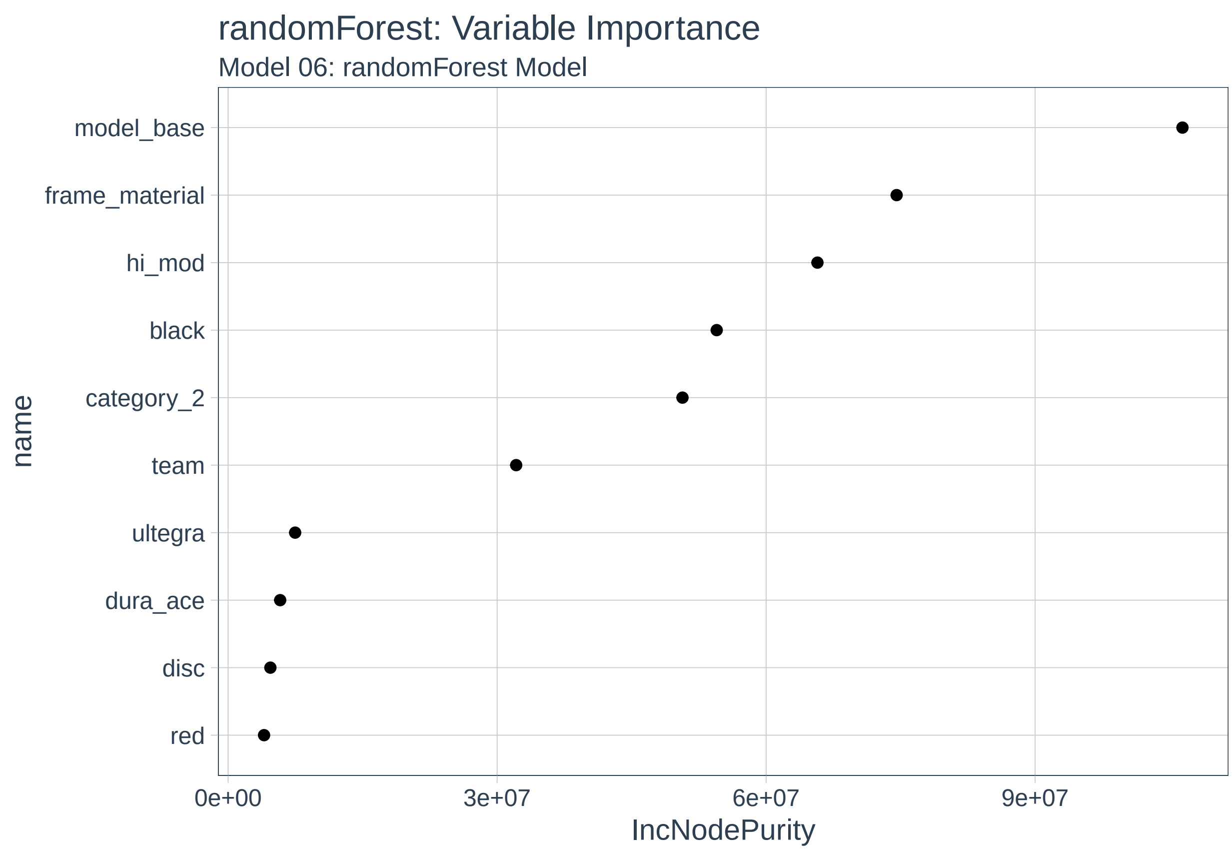
*** END FIX 3 *** —-

4.2.4 randomForest: Feature Importance —-

as_tibble(rownames = "name")

```
model_06_rand_forest_randomForest$fit %>%
    randomForest::importance() %>%
    as_tibble(rownames = "name") %>%
    arrange(desc(IncNodePurity)) %>%
```

```
mutate(name = as_factor(name) %>% fct_rev()) %>%

ggplot(aes(IncNodePurity, name)) +
geom_point() +
labs(
    title = "randomForest: Variable Importance",
    subtitle = "Model 06: randomForest Model"
) +
theme_tq()
```



> ProTips: Always make your researh reproducible. This makes it easier for others to heck your work, getting the same result as you. (model robustness)

4.3 XGBOOST —-

?boost_tree ?xgboost::xgboost

Learning Rate Thye gradient decent learning rate is used to find the global optima that reduces the model error (loss function). Too low, and algorithm gets stuck in a local optima. Too high, and it misses the global optima.

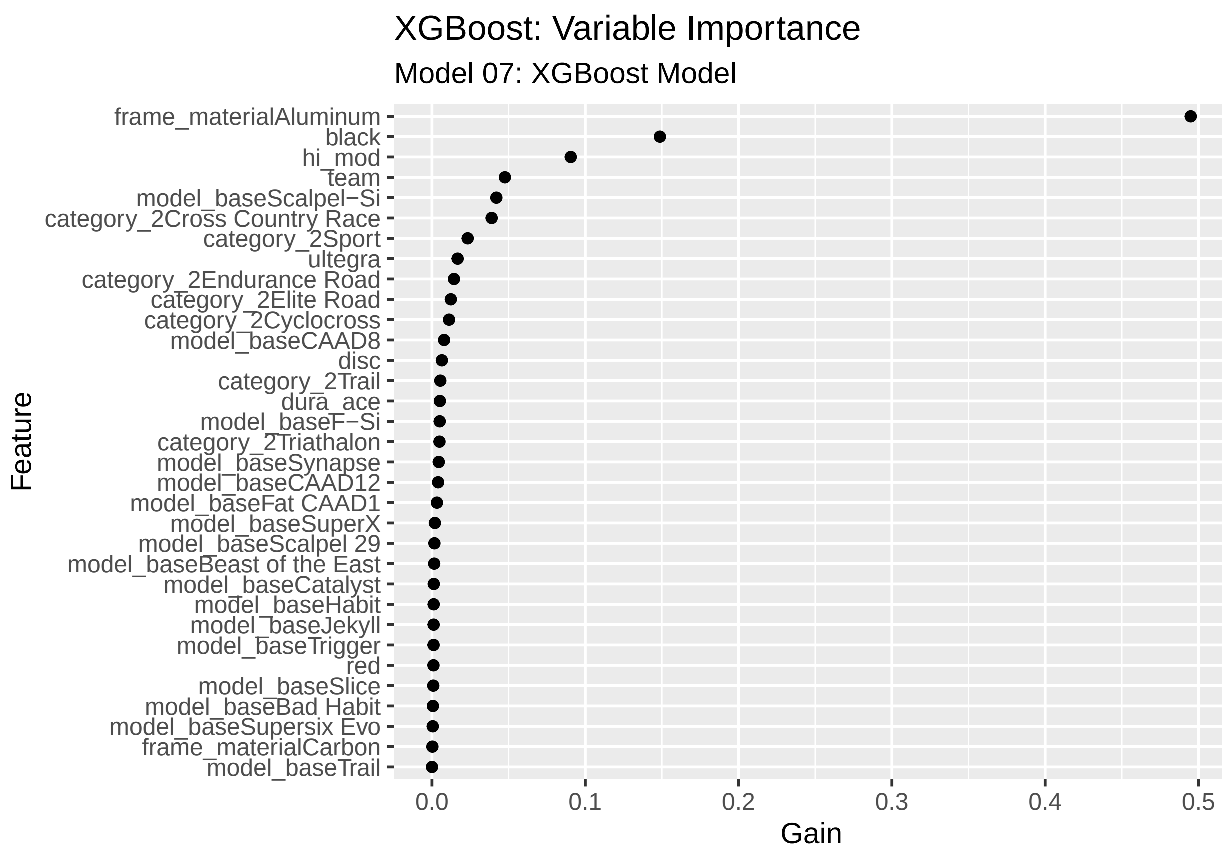
4.3.1 Model —-

```
set.seed(42)
# Linear Booster
# Tree Booster

model_07_boost_tree_xgboost <- boost_tree(
    # unknown, regression, classification
    mode = "regression",
    # Number for the number of precditors that will be randomly sampled at each split when reating the
    mtry = 30,
    learn_rate = 0.25,
    tree_depth = 7
    ) %>%
    set_engine("xgboost") %>%
    fit(price ~ ., data = train_tbl %>% select(-id, -model, -model_tier))

model_07_boost_tree_xgboost %>% calc_metrics(test_tbl)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2219.
## 2 rsq     standard       0.783
## 3 mae     standard    1597.
```

4.3.2 Feature Importance —-

```
model_07_boost_tree_xgboost$fit %>%
    xgboost::xgb.importance(model = .) %>%
    as_tibble() %>%
    arrange(desc(Gain)) %>%
    mutate(Feature = as_factor(Feature) %>% fct_rev()) %>%

    ggplot(aes(Gain, Feature)) +
    geom_point() +
    labs(
        title = "XGBoost: Variable Importance",
        subtitle = "Model 07: XGBoost Model"
    )
```



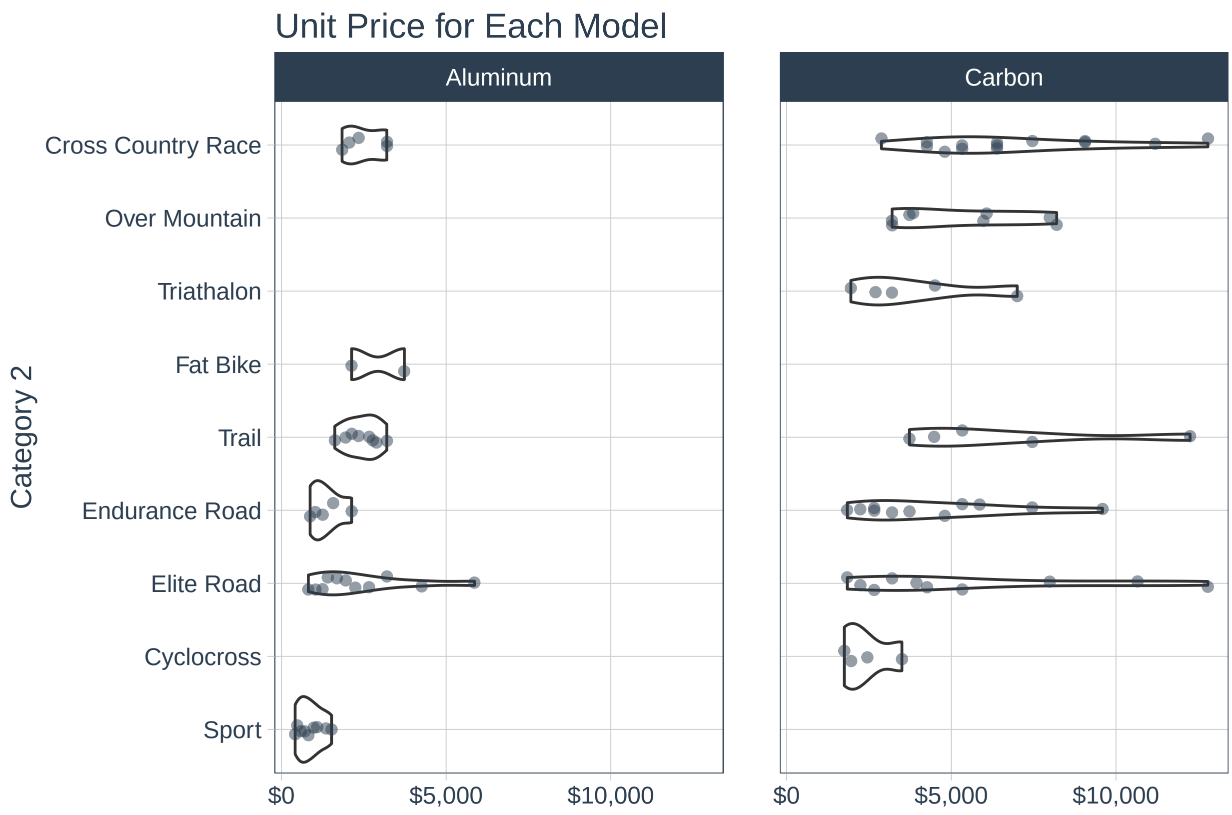
5.0 TESTING THE ALGORITHMS OUT —-

> Protip: It is always a good idea to set up an experiemnt to determine if your models are predicting how you think they should predict. In this session, we will see how we can use data visualisation to help determine model effectivness and performnce!

Taks: Predict Price for New Bike Models

```
g1 <- bike_features_tbl %>%
    mutate(category_2 = as_factor(category_2) %>%
               fct_reorder(price)) %>%

    ggplot(aes(category_2, y = price)) +
    geom_violin() +
    geom_jitter(width = 0.1, alpha = 0.5, color = "#2c3e50") +
    coord_flip() +
    facet_wrap(~ frame_material) +
    scale_y_continuous(labels = scales::dollar_format()) +
    theme_tq() +
    labs(
        title = "Unit Price for Each Model",
        y = "", x = "Category 2"
    )
```

```
g1
```



5.1 NEW JEKYLL MODEL —-

```
new_over_mountain_jekyll <- tibble(
    model = "Jekyll Al 1",
    frame_material = as.factor("Aluminum"),
    category_2 = as.factor("Over Mountain"),
    model_base = as.factor("Jekyll"),
    model_tier = as.factor("Aluminum 1"),
    black       = 0,
    hi_mod      = 0,
    team        = 0,
    red         = 0,
    ultegra     = 0,
    dura_ace    = 0,
    disc        = 0
)

new_over_mountain_jekyll
```

```
## # A tibble: 1 x 12
```

```
##   model frame_material category_2 model_base model_tier black hi_mod  team   red
##   <chr> <fct>          <fct>      <fct>      <fct>      <dbl>  <dbl> <dbl> <dbl>
## 1 Jeky~ Aluminum       Over Moun~ Jekyll     Aluminum 1     0      0     0     0
## # ... with 3 more variables: ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

Linear Methods —-

```
predict(model_03_linear_glmnet, new_data = new_over_mountain_jekyll)
```

```
## # A tibble: 1 x 1
##   .pred
##   <dbl>
## 1 1952.
```

Tree-Based Methods —-

```
predict(model_07_boost_tree_xgboost, new_data = new_over_mountain_jekyll)
```

```
## # A tibble: 1 x 1
##   .pred
##   <dbl>
## 1 2151.
```

Iteration

> ProTips: Data Frames can be a very useful way to keep models organised. Just put them in a "list-column"

```
models_tbl <- tibble(
    model_id = str_c("Model 0", 1:7),
    model = list(
        model_01_linear_lm_simple,
        model_02_linear_lm_complex,
        model_03_linear_glmnet,
        model_04_tree_decision_tree,
        model_05_rand_forest_ranger,
        model_06_rand_forest_randomForest,
        model_07_boost_tree_xgboost
    )
)

models_tbl
```

```
## # A tibble: 7 x 2
##   model_id model
```

```
##   <chr>    <list>
## 1 Model 01 <fit[+]>
## 2 Model 02 <fit[+]>
## 3 Model 03 <fit[+]>
## 4 Model 04 <fit[+]>
## 5 Model 05 <fit[+]>
## 6 Model 06 <fit[+]>
## 7 Model 07 <fit[+]>
```

Add Predictions

```
# map(.x, .f, ...)
# The dots (...) are arguments you can specify in the mapping function (.f)
# .f = predict(object, new_data)
# The object is the models in our model column
# The new_data is our new over Mountain Bike

predictions_new_over_mountain_tbl <- models_tbl %>%
    mutate(predictions = map(model,
                             predict,
                             new_data = new_over_mountain_jekyll)) %>%
    unnest(predictions) %>%
    mutate(category_2 = "Over Mountain") %>%
    left_join(new_over_mountain_jekyll, by = "category_2")
```

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

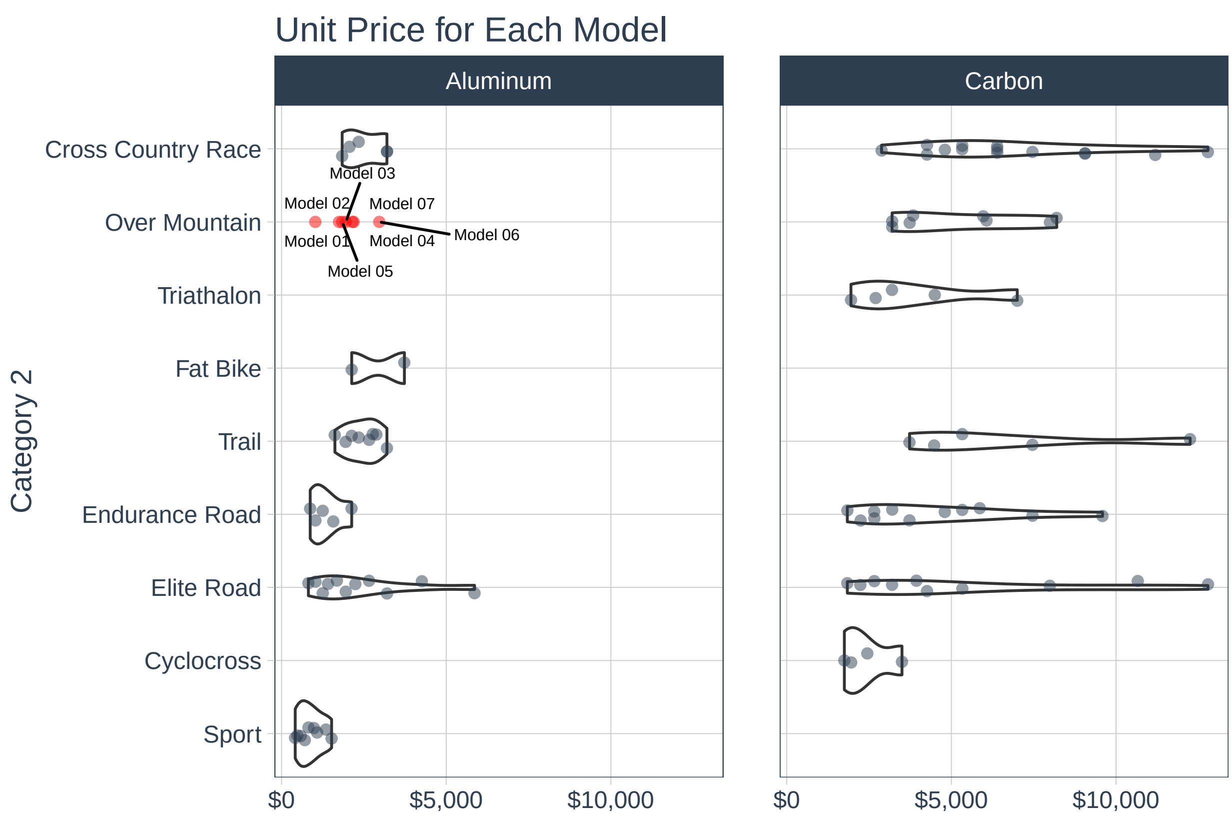
```
predictions_new_over_mountain_tbl
```

```
## # A tibble: 7 x 15
##   model_id model.x .pred category_2 model.y frame_material model_base model_tier
##   <chr>    <list>  <dbl> <chr>      <chr>   <fct>          <fct>      <fct>
## 1 Model 01 <fit[+~ 1025. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 2 Model 02 <fit[+~ 1746. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 3 Model 03 <fit[+~ 1952. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 4 Model 04 <fit[+~ 2191. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 5 Model 05 <fit[+~ 1843. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 6 Model 06 <fit[+~ 2968. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 7 Model 07 <fit[+~ 2151. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## # ... with 7 more variables: black <dbl>, hi_mod <dbl>, team <dbl>, red <dbl>,
## #   ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

Update plot

```
g2 <- g1 +
    geom_point(
```

```
        aes(y = .pred), color = "red", alpha = 0.5,
              data = predictions_new_over_mountain_tbl
        ) +
    ggrepel::geom_text_repel(
        aes(label = model_id, y = .pred),
            size = 2,
        max.overlaps = 100,
            data = predictions_new_over_mountain_tbl
        )

g2
```



5.2 NEW TRIATHALON MODEL —-

```
new_triathalon_slice_tbl <- tibble(
    model = "Slice Al 1",
    frame_material = as.factor("Aluminum"),
    category_2 = as.factor("Triathalon"),
    model_base = as.factor("Slice"),
    model_tier = as.factor("Ultegra"),
    black      = 0,
    hi_mod     = 0,
```

```
    team       = 0,
    red        = 0,
    ultegra    = 0,
    dura_ace   = 0,
    disc       = 0
)
```

Red Flag: Linear Models have an issue with predicting Triathlon bikes

We actually only need category_2 and frame_material. This simplifies the code quite a bit

```
new_triathalon_slice_tbl
```

```
## # A tibble: 1 x 12
##   model frame_material category_2 model_base model_tier black hi_mod  team   red
##   <chr> <fct>          <fct>      <fct>      <fct>      <dbl>  <dbl> <dbl> <dbl>
## 1 Slic~ Aluminum       Triathalon Slice      Ultegra        0      0     0     0
## # ... with 3 more variables: ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

```
predictions_new_triathalon_tbl <- models_tbl %>%
    mutate(
        predictions = map(model, predict, new_data = new_triathalon_slice_tbl)
        ) %>% unnest(predictions) %>%
    mutate(category_2 = "Triathalon", frame_material = "Aluminum")
```

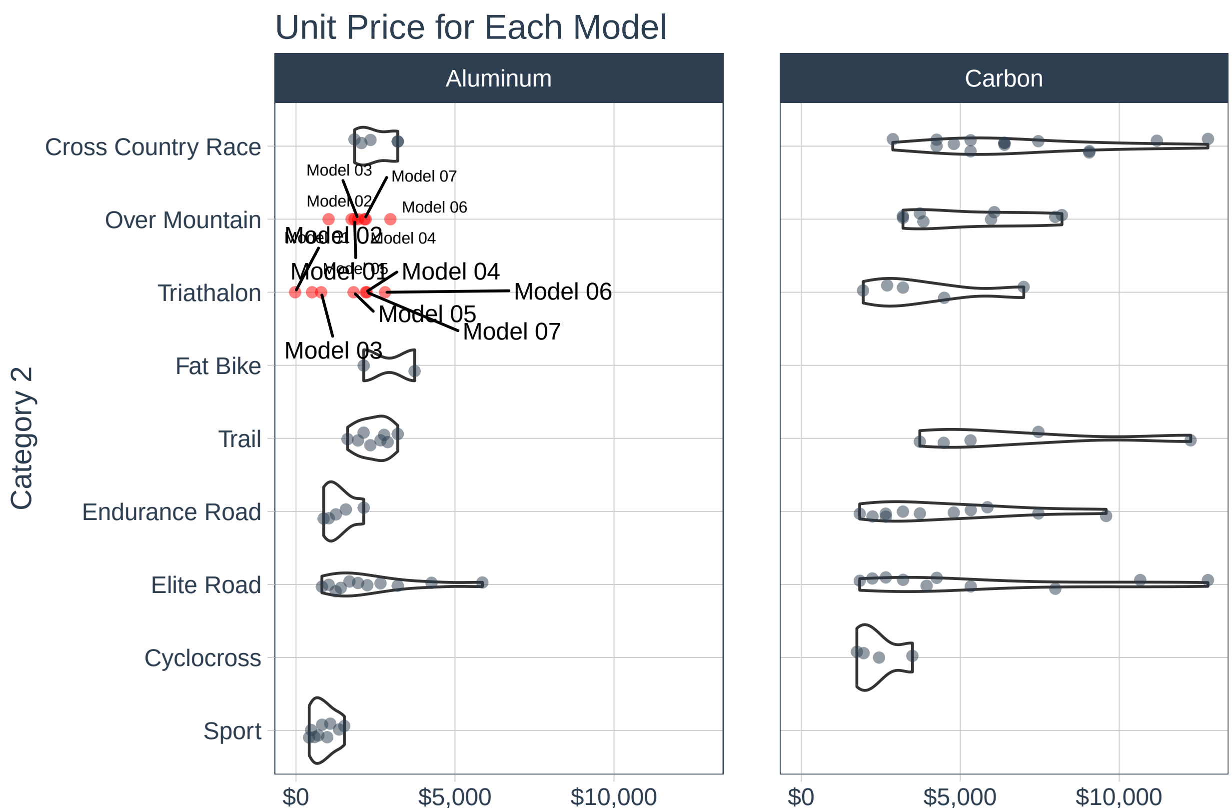
```
## Warning in predict.lm(object = object$fit, newdata = new_data, type =
## "response"): prediction from a rank-deficient fit may be misleading
```

```
predictions_new_triathalon_tbl
```

```
## # A tibble: 7 x 5
##   model_id model     .pred category_2 frame_material
##   <chr>    <list>    <dbl> <chr>      <chr>
## 1 Model 01 <fit[+]>  504.  Triathalon Aluminum
## 2 Model 02 <fit[+]>  -28.3 Triathalon Aluminum
## 3 Model 03 <fit[+]>  791.  Triathalon Aluminum
## 4 Model 04 <fit[+]> 2191.  Triathalon Aluminum
## 5 Model 05 <fit[+]> 1810.  Triathalon Aluminum
## 6 Model 06 <fit[+]> 2800.  Triathalon Aluminum
## 7 Model 07 <fit[+]> 2225.  Triathalon Aluminum
```

Update Plot

```
g3 <- g2 +
    geom_point(aes(y = .pred), color = "red", alpha = 0.5,
               data = predictions_new_triathalon_tbl) +
    ggrepel::geom_text_repel(aes(y = .pred, label = model_id),
                             size = 3,
                             max.overlaps = 500,
                             data = predictions_new_triathalon_tbl)
g3
```



6.0 ADDITIONAL ADVANCED CONCEPTS —-

- CLASSIFICATION - Binary & Multi-Class
- ADVANCED ALGORITHMS
  - SVMs - svm_poly() and svm_rbf() - Must be normalized
  - Neural Networks - keras - Must be normalized
  - Stacking Models
- PREPROCESSING - recipes
- HYPERPARAMETER TUNING - purrr
- SAMPLING & CROSS VALIDATION - rsample
- AUTOMATIC MACHINE LEARNING - H2O

7.0 BONUS - PREPROCESSING & SVM-Regression —-

In step_dummy() process ProTips: Set one_hot = TRUE to get all of the categories as columns. By default, step_dummy() returns one less column than number of categories

*** FIX 4 *** —- Error: Assigned data `log(new_data[[col_names[i]]] + object$offset, base = object$base)` must be compatible with existing data. - Recipe interface has changed when applying recipes to the target - Need to use skip = TRUE

```r
library(recipes)


recipe_obj <- recipe(price ~ ., data = train_tbl) %>%
    # step_rm: remove variables
    step_rm(id, model, model_tier) %>%

    # step_dummy:
    # converts categorical data to bianry columns (os and 1s)
    # all_nominal(): selects any variables that are categorical
    step_dummy(all_nominal(), one_hot = TRUE) %>%

    # *** END FIX 4 *** ----
    # step_log(price) %>%
    # step_center(price) %>%
    # step_scale(price) %>%

    # step_center(): substracts the mean of the numeric feature
    # step_log(): Applies a logarithmic transformation. Great for
    # normalising skew (making distribution a bell curve shape)
    step_log(price, skip = TRUE) %>%
    step_center(price, skip = TRUE) %>%
    step_scale(price, skip = TRUE) %>%

    # *** END FIX 4 *** ----

    # prep():
    # Once steps have been added, prep() will perform initial calculations
    # prior to applying the recipe
    prep()

bake(recipe_obj, train_tbl) %>% glimpse()
```

```
## Rows: 77
## Columns: 37
## $ black                       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1~
## $ hi_mod                      <dbl> 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ team                        <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ red                         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ultegra                     <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0~
## $ dura_ace                    <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ disc                        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ price                       <dbl> 5970, 2770, 10660, 3200, 5330, 4500, 224~
## $ category_2_Cross.Country.Race <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ category_2_Cyclocross       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Elite.Road       <dbl> 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1~
## $ category_2_Endurance.Road   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Fat.Bike         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Over.Mountain    <dbl> 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Sport            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Trail            <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Triathalon       <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ frame_material_Aluminum     <dbl> 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1~
```

```
## $ frame_material_Carbon         <dbl> 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0~
## $ model_base_Bad.Habit          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Beast.of.the.East  <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_CAAD12             <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1~
## $ model_base_CAAD8              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0~
## $ model_base_Catalyst           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_F.Si               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Fat.CAAD1          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Fat.CAAD2          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Habit              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Jekyll             <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Scalpel.29         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Scalpel.Si         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ model_base_Slice              <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Supersix.Evo       <dbl> 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0~
## $ model_base_SuperX             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Synapse            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Trail              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Trigger            <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
bake(recipe_obj, test_tbl) %>% glimpse()
```

```
## Rows: 20
## Columns: 37
## $ black                         <dbl> 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0~
## $ hi_mod                        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1~
## $ team                          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ red                           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ultegra                       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0~
## $ dura_ace                      <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1~
## $ disc                          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ price                         <dbl> 6070, 12790, 480, 11190, 1960, 3200, 320~
## $ category_2_Cross.Country.Race <dbl> 0, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 0, 0, 0~
## $ category_2_Cyclocross         <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Elite.Road         <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ category_2_Endurance.Road     <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1~
## $ category_2_Fat.Bike           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Over.Mountain      <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0~
## $ category_2_Sport              <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Trail              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Triathalon         <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0~
## $ frame_material_Aluminum       <dbl> 0, 0, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0~
## $ frame_material_Carbon         <dbl> 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1~
## $ model_base_Bad.Habit          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Beast.of.the.East  <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_CAAD12             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ model_base_CAAD8              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Catalyst           <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_F.Si               <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0~
## $ model_base_Fat.CAAD1          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Fat.CAAD2          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Habit              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Jekyll             <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Scalpel.29         <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
```

```
## $ model_base_Scalpel.Si         <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Slice              <dbl> 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0~
## $ model_base_Supersix.Evo       <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_SuperX             <dbl> 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Synapse            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1~
## $ model_base_Trail              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Trigger            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0~
```

```
# tidy(recipe): tidying a recipe returns information on all of the steps!!!
tidy(recipe_obj)
```

```
## # A tibble: 5 x 6
##   number operation type   trained skip  id
##    <int> <chr>     <chr>  <lgl>   <lgl> <chr>
## 1      1 step      rm     TRUE    FALSE rm_IeqAq
## 2      2 step      dummy  TRUE    FALSE dummy_zMHf4
## 3      3 step      log    TRUE    TRUE  log_95NzG
## 4      4 step      center TRUE    TRUE  center_W6e83
## 5      5 step      scale  TRUE    TRUE  scale_SqpGM
```

```
# Tidying a recipe and including a step number returns information on the step
scale  <- tidy(recipe_obj, 5)
center <- tidy(recipe_obj, 4)
```

*** FIX *** —-

Why the error occurs:

Any time that we apply a recipe step to an “outcome” (e.g. the price column) that is not present in new data Solution 1 - Use `skip = TRUE` in the recipe step and `juice()` instead of `bake()` to transform the training data set Solution 2 - Include an artificial outcome column in the new_data that allows the steps to be performed instead of causing an error.

```
# - Recipe interface has changed when applying recipes to the target (when skip = TRUE)
# - Need to juice() instead of bake()

# train_transformed_tbl <- bake(recipe_obj, train_tbl)
train_transformed_tbl <- juice(recipe_obj) %>%
  select(price, everything())

# *** END FIX *** ----

test_transformed_tbl  <- bake(recipe_obj, test_tbl) %>%
  select(price, everything())

tidy(recipe_obj)
```

```
## # A tibble: 5 x 6
##   number operation type   trained skip  id
##    <int> <chr>     <chr>  <lgl>   <lgl> <chr>
## 1      1 step      rm     TRUE    FALSE rm_IeqAq
```

```
## 2      2 step      dummy  TRUE    FALSE dummy_zMHf4
## 3      3 step      log    TRUE    TRUE  log_95NzG
## 4      4 step      center TRUE    TRUE  center_W6e83
## 5      5 step      scale  TRUE    TRUE  scale_SqpGM
```

```
scale  <- tidy(recipe_obj, 5)
center <- tidy(recipe_obj, 4)
tidy(recipe_obj, 3)
```

```
## # A tibble: 1 x 3
##   terms  base id
##   <chr> <dbl> <chr>
## 1 price  2.72 log_95NzG
```

> Protips: Whenever we transform data we need to reverse the transformations after making predictions

> likewise to glm, svm has scaling built into the model. Hence once we have pre-processed feature scaling, we need to avoid double scaling by scale = FALSE. Double scaling with reduce the model performance

SVM: Radial Basis

```
train_transformed_tbl %>% glimpse()
```

```
## Rows: 77
## Columns: 37
## $ price                       <dbl> 1.00543448, -0.05704399, 1.80758814, 0.1~
## $ black                       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1~
## $ hi_mod                      <dbl> 0, 0, 1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ team                        <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ red                         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ ultegra                     <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0~
## $ dura_ace                    <dbl> 0, 0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ disc                        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0~
## $ category_2_Cross.Country.Race <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ category_2_Cyclocross       <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Elite.Road       <dbl> 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 1, 1, 1, 1~
## $ category_2_Endurance.Road   <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Fat.Bike         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Over.Mountain    <dbl> 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Sport            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Trail            <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ category_2_Triathalon       <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ frame_material_Aluminum     <dbl> 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1, 1, 1~
## $ frame_material_Carbon       <dbl> 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0~
## $ model_base_Bad.Habit        <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Beast.of.the.East <dbl> 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_CAAD12           <dbl> 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 1~
```

```
## $ model_base_CAAD8              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0~
## $ model_base_Catalyst           <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_F.Si               <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Fat.CAAD1          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Fat.CAAD2          <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Habit              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Jekyll             <dbl> 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Scalpel.29         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Scalpel.Si         <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0~
## $ model_base_Slice              <dbl> 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Supersix.Evo       <dbl> 0, 0, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0~
## $ model_base_SuperX             <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Synapse            <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Trail              <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
## $ model_base_Trigger            <dbl> 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0~
```

```r
model_08_svm_rbf <- svm_rbf("regression", cost = 10, rbf_sigma = 0.1, margin = 0.25) %>%
    set_engine("kernlab", scaled = FALSE) %>%
    fit(price ~ ., data = train_transformed_tbl)

model_08_svm_rbf %>%
    predict(new_data = test_transformed_tbl) %>%
    mutate(
        .pred = .pred * scale$value,
        .pred = .pred + center$value,
        # reverse the log
        .pred = exp(.pred)
        ) %>%
    bind_cols(test_tbl %>% select(price)) %>%
    yardstick::metrics(truth = price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard    2667.
## 2 rsq     standard       0.638
## 3 mae     standard    1621.
```

Predictions

```r
# fix are done for this part: new data without the y values(price)
bake(recipe_obj, new_data = new_over_mountain_jekyll) %>%
    predict(object = model_08_svm_rbf, new_data = .) %>%
    mutate(
        .pred = .pred * scale$value,
        .pred = .pred + center$value,
        .pred = exp(.pred)
    )
```

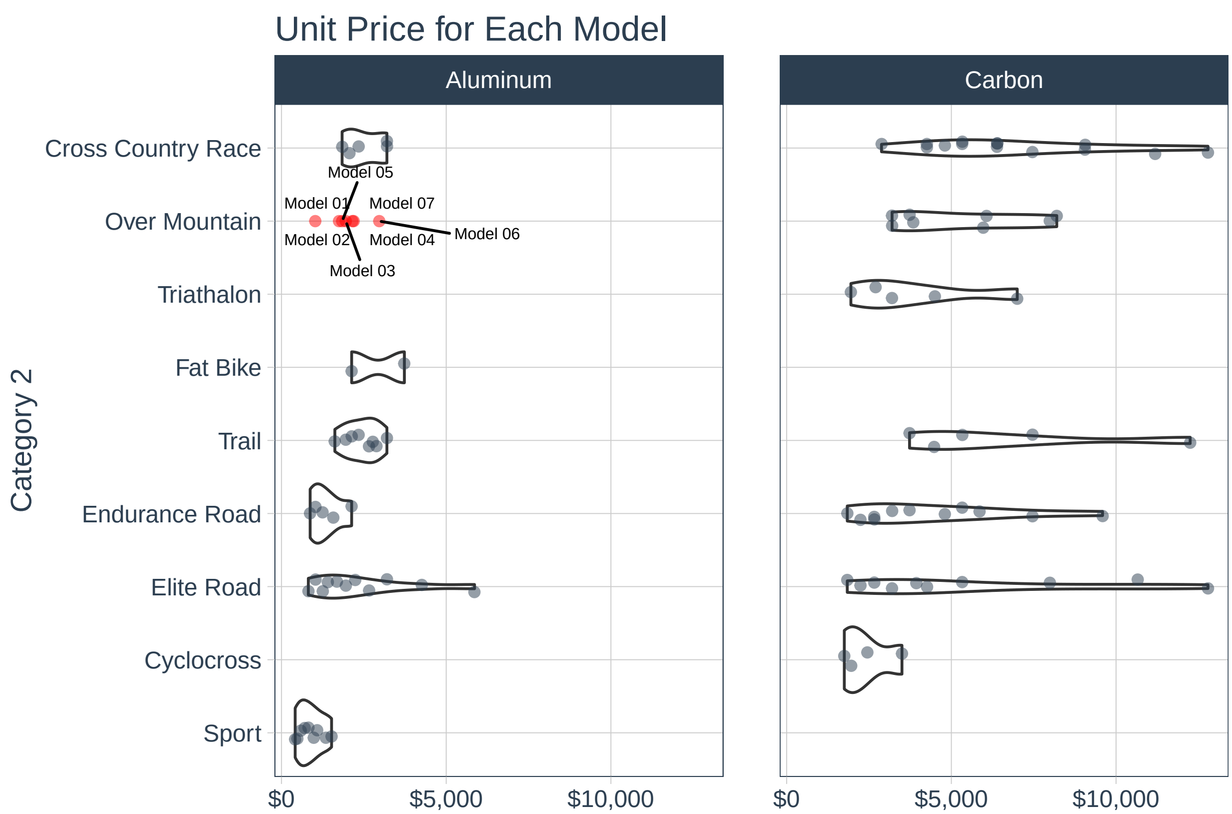
```
## # A tibble: 1 x 1
```

```
##    .pred
##    <dbl>
## 1 1904.
```

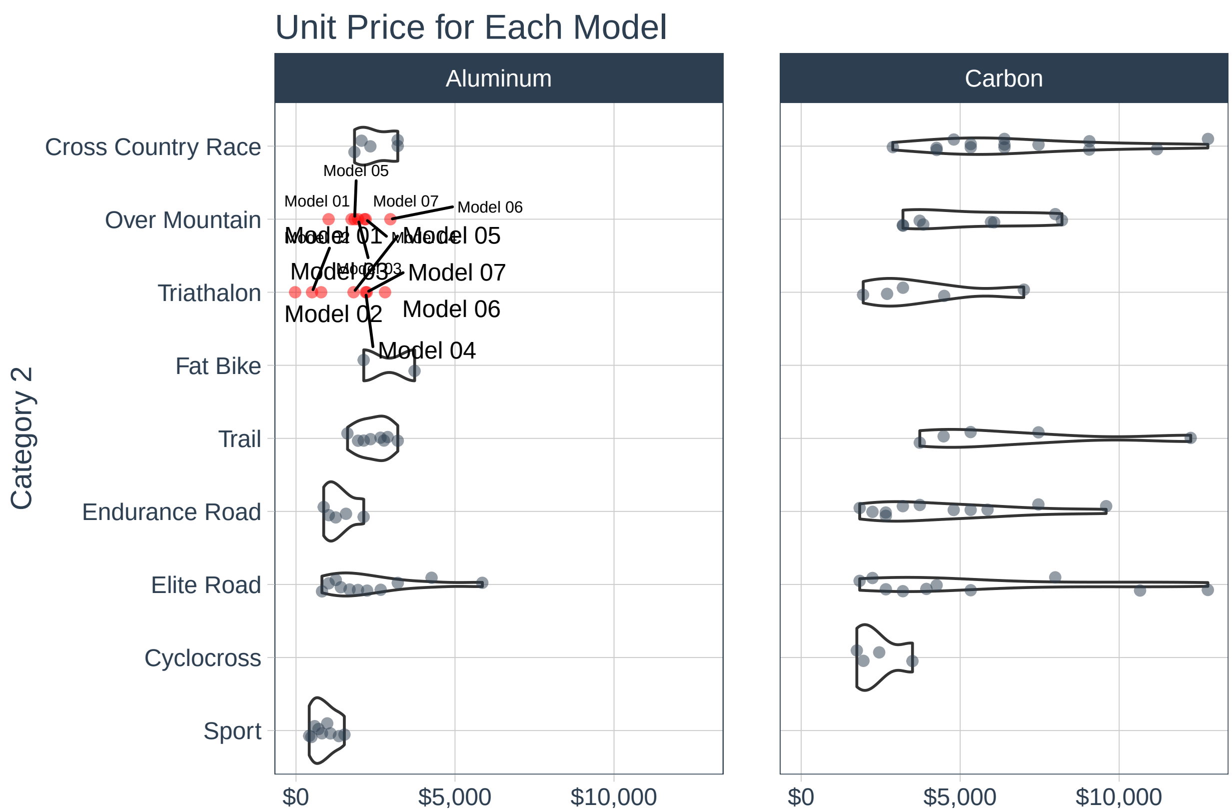
```
predictions_new_over_mountain_tbl
```

```
## # A tibble: 7 x 15
##   model_id model.x .pred category_2 model.y frame_material model_base model_tier
##   <chr>    <list>  <dbl> <chr>      <chr>   <fct>          <fct>      <fct>
## 1 Model 01 <fit[+~ 1025. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 2 Model 02 <fit[+~ 1746. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 3 Model 03 <fit[+~ 1952. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 4 Model 04 <fit[+~ 2191. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 5 Model 05 <fit[+~ 1843. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 6 Model 06 <fit[+~ 2968. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## 7 Model 07 <fit[+~ 2151. Over Moun~ Jekyll~ Aluminum       Jekyll     Aluminum 1
## # ... with 7 more variables: black <dbl>, hi_mod <dbl>, team <dbl>, red <dbl>,
## #   ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

```
g2
```




```
g3
```




```
bake(recipe_obj, new_data = new_triathalon_slice_tbl) %>%
    predict(object = model_08_svm_rbf, new_data = .) %>%
    mutate(
        .pred = .pred * scale$value,
        .pred = .pred + center$value,
        .pred = exp(.pred)
    )
```

```
## # A tibble: 1 x 1
##    .pred
##    <dbl>
## 1 1569.
```

```
predictions_new_triathalon_tbl
```

```
## # A tibble: 7 x 5
##   model_id model      .pred category_2 frame_material
##   <chr>    <list>     <dbl> <chr>      <chr>
## 1 Model 01 <fit[+]>  504.  Triathalon Aluminum
## 2 Model 02 <fit[+]>  -28.3 Triathalon Aluminum
## 3 Model 03 <fit[+]>  791.  Triathalon Aluminum
## 4 Model 04 <fit[+]> 2191.  Triathalon Aluminum
## 5 Model 05 <fit[+]> 1810.  Triathalon Aluminum
## 6 Model 06 <fit[+]> 2800.  Triathalon Aluminum
## 7 Model 07 <fit[+]> 2225.  Triathalon Aluminum
```

```r
bike_features_tbl %>%
    filter(category_2 == "Endurance Road") %>%
    arrange(price)
```

```
## # A tibble: 16 x 14
##       id price model     category_2  frame_material model_base model_tier  black
##    <int> <dbl> <chr>     <chr>       <chr>          <chr>      <chr>       <dbl>
##  1    44   870 Synapse ~ Endurance ~ Aluminum       Synapse    Claris          0
##  2    32  1030 Synapse ~ Endurance ~ Aluminum       Synapse    Sora            0
##  3    12  1250 Synapse ~ Endurance ~ Aluminum       Synapse    Disc Tiagra     0
##  4     8  1570 Synapse ~ Endurance ~ Aluminum       Synapse    Disc 105        0
##  5    40  1840 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Tia~     0
##  6    68  2130 Synapse ~ Endurance ~ Aluminum       Synapse    Disc Adven~     0
##  7    17  2240 Synapse ~ Endurance ~ Carbon         Synapse    Carbon 105      0
##  8    29  2660 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Ult~     0
##  9    73  2660 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Dis~     0
## 10    39  3200 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Ult~     0
## 11    37  3730 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Dis~     0
## 12     9  4800 Synapse ~ Endurance ~ Carbon         Synapse    Carbon Dis~     0
## 13    83  5330 Synapse ~ Endurance ~ Carbon         Synapse    Hi-Mod Dis~     0
## 14    70  5860 Synapse ~ Endurance ~ Carbon         Synapse    Hi-Mod Dur~     0
## 15    35  7460 Synapse ~ Endurance ~ Carbon         Synapse    Hi-Mod Dis~     0
## 16    41  9590 Synapse ~ Endurance ~ Carbon         Synapse    Hi-Mod Dis~     1
## # ... with 6 more variables: hi_mod <dbl>, team <dbl>, red <dbl>,
## #   ultegra <dbl>, dura_ace <dbl>, disc <dbl>
```

8.0 SAVING & LOADING MODELS —-

```r
fs::dir_create("00_models")

models_tbl <- list(
    "MODEL_01__LM_SIMPLE"  = model_01_linear_lm_simple,
    "MODEL_02__LM_COMPLEX" = model_02_linear_lm_complex,
    "MODEL_03__GLMNET"     = model_03_linear_glmnet,
    "MODEL_04__DECISION_TREE"    = model_04_tree_decision_tree,
    "MODEL_05__RF_RANGER"        = model_05_rand_forest_ranger,
    "MODEL_06__RF_RANDOMFOREST" = model_06_rand_forest_randomForest,
    "MODEL_07__XGBOOST" = model_07_boost_tree_xgboost,
    "MODEL_08__SVM"     = model_08_svm_rbf) %>%
    enframe(name = "model_id", value = "model")

models_tbl
```

```
## # A tibble: 8 x 2
##   model_id                  model
##   <chr>                     <list>
## 1 MODEL_01__LM_SIMPLE       <fit[+]>
## 2 MODEL_02__LM_COMPLEX      <fit[+]>
## 3 MODEL_03__GLMNET          <fit[+]>
## 4 MODEL_04__DECISION_TREE   <fit[+]>
```

```
## 5 MODEL_05__RF_RANGER       <fit[+]>
## 6 MODEL_06__RF_RANDOMFOREST <fit[+]>
## 7 MODEL_07__XGBOOST         <fit[+]>
## 8 MODEL_08__SVM             <fit[+]>
```

```
models_tbl %>% write_rds("00_models/parsnip_models_tbl.rds")

recipes_tbl <- list("RECIPE_01" = recipe_obj) %>%
    enframe(name = "recipe_id", value = "recipe")

recipes_tbl %>% write_rds("00_models/recipes_tbl.rds")

calc_metrics %>% write_rds("01_scripts/calc_metrics.rds")
```

Reading

```
models_tbl <- read_rds("00_models/parsnip_models_tbl.rds")

recipes_tbl <- read_rds("00_models/recipes_tbl.rds")

calc_metrics <- read_rds("01_scripts/calc_metrics.rds")
```