Project 2: Linear Regression and Its Cousins

Amanda Rose Knudsen

2025-05-04

Project 2 (Team) Assignment Prompt:

This is role playing. I am your new boss. I am in charge of production at ABC Beverage and you are a team of data scientists reporting to me. My leadership has told me that new regulations are requiring us to understand our manufacturing process, the predictive factors and be able to report to them our predictive model of PH.

Please use the historical data set I am providing. Build and report the factors in BOTH a technical and non-technical report. I like to use Word and Excel. Please provide your non-technical report in a business friendly readable document and your predictions in an Excel readable format. The technical report should show clearly the models you tested and how you selected your final approach.

Please submit both Rpubs links and .rmd files or other readable formats for technical and non-technical reports. Also submit the excel file showing the prediction of your models for pH.

```
library(tidyverse)
library(knitr)
library(caret)
library(pls)
library(glmnet)
library(readxl)
library(corrplot)
library(RANN)
library(e1071)
```

Linear Regression Model

Our objective is to evaluate several linear modeling approaches to predict pH in a beverage production process, with a focus on accuracy, interpretability, and practical application for business stakeholders.

Explore and Preprocess

We'll explore and then pre-process the data. First, we must load the data files.

The "StudentData" file is our training data, and the "StudentEvaluation" file is our testing data.

If our data had not been provided in separate training and test files, we would subsequently perform a split of the 'main' data file into a training and a test set. In this case, since it's already been provided, we do not need to segment.

```r
train_data <- read_excel("StudentData.xlsx")
test_data <- read_excel("StudentEvaluation.xlsx")
```

Since we'll be training our model on the training data, let's look at the first few rows of the data to see what we've got.

```r
head(train_data)
```

```
# A tibble: 6 x 33
  `Brand Code` `Carb Volume` `Fill Ounces` `PC Volume` `Carb Pressure`
  <chr>                <dbl>         <dbl>       <dbl>           <dbl>
1 B                     5.34          24.0       0.263            68.2
2 A                     5.43          24.0       0.239            68.4
3 B                     5.29          24.1       0.263            70.8
4 A                     5.44          24.0       0.293            63
5 A                     5.49          24.3       0.111            67.2
6 A                     5.38          23.9       0.269            66.6
# i 28 more variables: `Carb Temp` <dbl>, PSC <dbl>, `PSC Fill` <dbl>,
#   `PSC CO2` <dbl>, `Mnf Flow` <dbl>, `Carb Pressure1` <dbl>,
#   `Fill Pressure` <dbl>, `Hyd Pressure1` <dbl>, `Hyd Pressure2` <dbl>,
#   `Hyd Pressure3` <dbl>, `Hyd Pressure4` <dbl>, `Filler Level` <dbl>,
#   `Filler Speed` <dbl>, Temperature <dbl>, `Usage cont` <dbl>,
#   `Carb Flow` <dbl>, Density <dbl>, MFR <dbl>, Balling <dbl>,
#   `Pressure Vacuum` <dbl>, PH <dbl>, `Oxygen Filler` <dbl>, ...
```

```r
str(train_data)
```

```
tibble [2,571 x 33] (S3: tbl_df/tbl/data.frame)
 $ Brand Code       : chr [1:2571] "B" "A" "B" "A" ...
 $ Carb Volume      : num [1:2571] 5.34 5.43 5.29 5.44 5.49 ...
 $ Fill Ounces      : num [1:2571] 24 24 24.1 24 24.3 ...
 $ PC Volume        : num [1:2571] 0.263 0.239 0.263 0.293 0.111 ...
 $ Carb Pressure    : num [1:2571] 68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...
 $ Carb Temp        : num [1:2571] 141 140 145 133 137 ...
 $ PSC              : num [1:2571] 0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...
 $ PSC Fill         : num [1:2571] 0.26 0.22 0.34 0.42 0.16 ...
 $ PSC CO2          : num [1:2571] 0.04 0.04 0.16 0.04 0.12 ...
 $ Mnf Flow         : num [1:2571] -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
 $ Carb Pressure1   : num [1:2571] 119 122 120 115 118 ...
 $ Fill Pressure    : num [1:2571] 46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...
 $ Hyd Pressure1    : num [1:2571] 0 0 0 0 0 0 0 0 0 0 ...
 $ Hyd Pressure2    : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...
 $ Hyd Pressure3    : num [1:2571] NA NA NA 0 0 0 0 0 0 0 ...
 $ Hyd Pressure4    : num [1:2571] 118 106 82 92 92 116 124 132 90 108 ...
 $ Filler Level     : num [1:2571] 121 119 120 118 119 ...
 $ Filler Speed     : num [1:2571] 4002 3986 4020 4012 4010 ...
 $ Temperature      : num [1:2571] 66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...
 $ Usage cont       : num [1:2571] 16.2 19.9 17.8 17.4 17.7 ...
 $ Carb Flow        : num [1:2571] 2932 3144 2914 3062 3054 ...
 $ Density          : num [1:2571] 0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...
 $ MFR              : num [1:2571] 725 727 735 731 723 ...
```

```
 $ Balling          : num [1:2571] 1.4 1.5 3.14 3.04 3.04 ...
 $ Pressure Vacuum  : num [1:2571] -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...
 $ PH               : num [1:2571] 8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...
 $ Oxygen Filler    : num [1:2571] 0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...
 $ Bowl Setpoint    : num [1:2571] 120 120 120 120 120 120 120 120 120 120 ...
 $ Pressure Setpoint: num [1:2571] 46.4 46.8 46.6 46 46 46 46 46 46 46 ...
 $ Air Pressurer    : num [1:2571] 143 143 142 146 146 ...
 $ Alch Rel         : num [1:2571] 6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...
 $ Carb Rel         : num [1:2571] 5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...
 $ Balling Lvl      : num [1:2571] 1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...
```

We can see there are 33 columns in total and 2,571 rows in total. The 33 columns includes the "PH" column which is what we will be aiming to predict – PH will be the response variable in our linear regression exploration.

We can see that we have mostly numeric values. The only character or categorical (non-numeric) is the `Brand Code`. Based on information in our team's preferred guidance on predictive modeling, to deal with non-numeric values (a.k.a. categorical values, which Brand Code is the only one) the recommendation is to either convert into dummy variables or remove if not informative or the value has too many categories.

Since "Brand Code" may be an important predictor, we will choose to convert to a dummy variable before training (since models such as Lasso requires input predictors to be numeric).

We can also see there is an assortment of null values even just in the first few rows of data, across various columns. This gives us a sense that we will need to impute missing values. This is a preferable approach to our exploratory predictive work because it enables us to keep predictors.

We can also see that there are negative values which means we won't be able to apply the BoxCox method for processing our data. Our guidance and standards, from Applied Predictive Modeling, state that if the data includes negatives we should use the YeoJohnson method instead.

```
colSums(is.na(train_data))
```

```
       Brand Code       Carb Volume       Fill Ounces         PC Volume 
              120                10                38                39 
   Carb Pressure         Carb Temp               PSC          PSC Fill 
               27                26                33                23 
         PSC CO2          Mnf Flow    Carb Pressure1     Fill Pressure 
               39                 2                32                22 
    Hyd Pressure1     Hyd Pressure2     Hyd Pressure3     Hyd Pressure4 
               11                15                15                30 
     Filler Level      Filler Speed       Temperature        Usage cont 
               20                57                14                 5 
        Carb Flow           Density               MFR           Balling 
                2                 1               212                 1 
  Pressure Vacuum                PH     Oxygen Filler     Bowl Setpoint 
                0                 4                12                 2 
Pressure Setpoint     Air Pressurer          Alch Rel          Carb Rel 
               12                 0                 9                10 
      Balling Lvl 
                1 
```

Once we build our training and test sets, we will continue exploration and pre-processing, including by dealing with our null values and looking at the relationships among predictors, among other steps.

We also notice there are 4 missing values for PH, which will be problematic. Guidance to handle this scenario is to remove the rows where the PH is null from the training and test sets so that would include removing the rows in the predictor and response sets. We will explain more on this when we get to that step prior to training our model.

Prepare response and predictor sets

We'll prepare response and predictor sets (separating out the predictors and response variable – pH is our "response".) We'll also remove "Brand Code" from both the training and test sets.

```
train_y <- train_data$PH
train_x <- train_data |> select(-PH)
test_y <- test_data$PH
test_x <- test_data |> select(-PH)
```

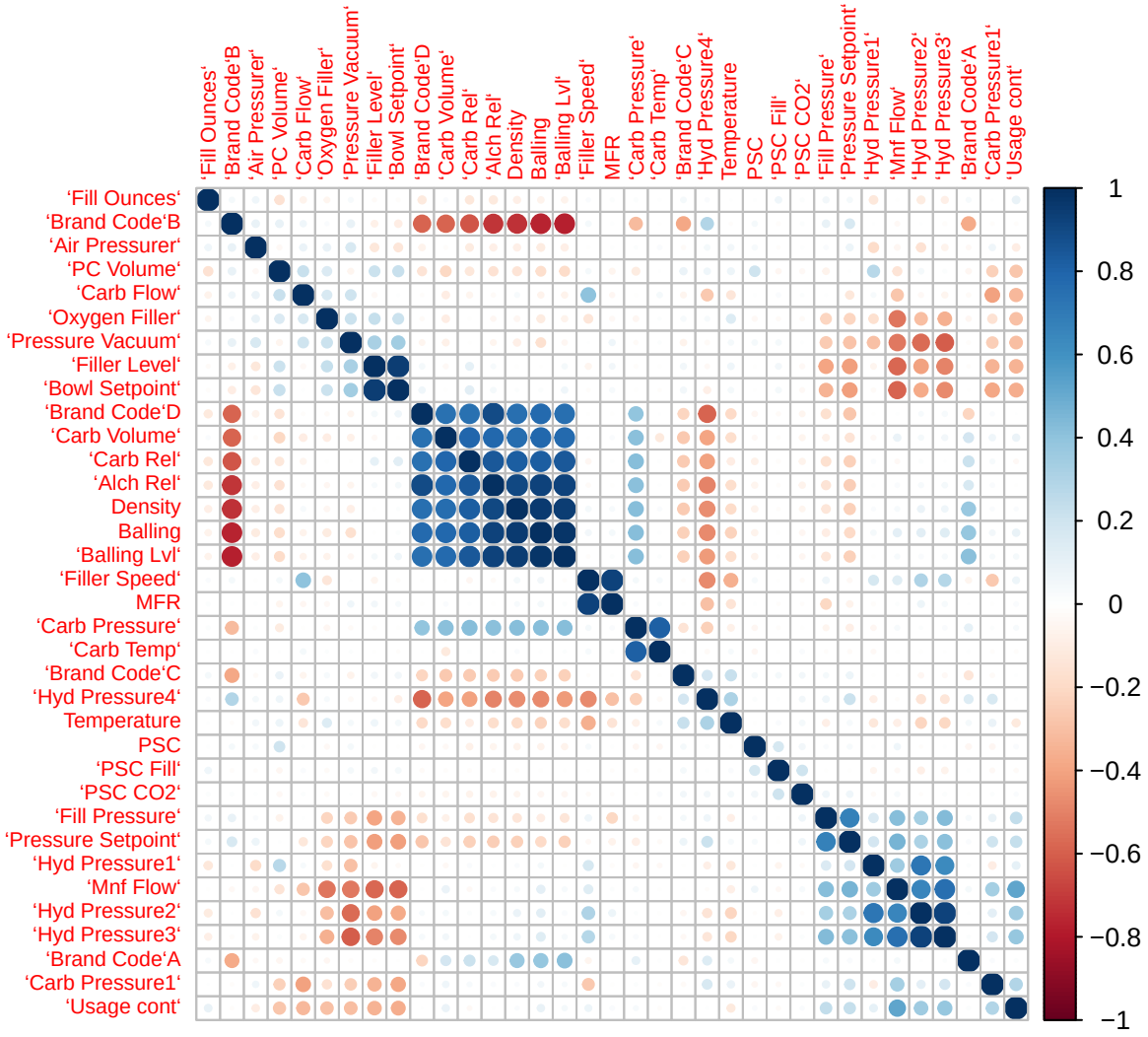
Next we'll convert all to numeric (dummy encoding for non-numeric variable Brand Code)

```
dummies <- dummyVars(~ ., data = train_x)
train_x <- predict(dummies, newdata = train_x)
test_x  <- predict(dummies, newdata = test_x)
```

Correlation matrix

A correlation matrix is used to understand the relationships among predictor variables.

```
cor_matrix <- cor(train_x, use = "pairwise.complete.obs")
corrplot(cor_matrix, order = "hclust", tl.cex = 0.7)
```



We removed highly redundant predictors using a correlation threshold of 0.95. While 0.75 is a common starting point, retaining more features helps maintain signal for models like PLS, which can internally handle correlated predictors by extracting latent factors.

```
high_corr <- findCorrelation(cor_matrix, cutoff = 0.95)
train_x <- train_x[, -high_corr]
test_x <- test_x[, -high_corr]
```

We removed only near-duplicate predictors with a correlation threshold of 0.95 to preserve interpretability for linear models while retaining useful variance for PLS.

Impute, transform, center and scale

To accommodate both traditional linear regression and PLS models, we will apply a preprocessing pipeline that includes transformations, centering, scaling, and imputation. Again based on our expert guidance from Applied Predictive Modeling, we will use the training set to fit preprocessing, and apply it to both training and test sets. We're setting the "seed" here to ensure reproducibility of results.

We will be imputing values using "knnImpute" to handle our NULL (missing) values; we will be using "YeoJohnson" for handling skewness and negative values (which we observed in our dataset – hence BoxCox method would not be appropriate), and we will center and scale the values, which is important for applied predictive modeling.

```
train_x <- as.data.frame(train_x)
test_x <- as.data.frame(test_x)
```

We coerced the entire dataset to a dataframe to ensure that our preprocessing steps will work.

However, before we move on we remember there are missing values in our outcome variable.

```
sum(is.na(train_y))
```

```
[1] 4
```

```
sum(is.na(test_y))
```

```
[1] 267
```

All 267 rows of the test set are missing values for PH because that is what we are trying to predict. This means that we can't remove all the corresponding predictor variables in the test set because that would literally remove all the values in our test set.

We'll remove rows from the training set where the outcome is missing. Modeling cannot proceed with missing values in the outcome.

```
complete_rows <- complete.cases(train_y)
train_y <- train_y[complete_rows]
train_x <- train_x[complete_rows, ]
```

To ensure that KNN imputation of missing values will work, we'll coerce the entire dataset to avoid hidden issues and then preprocess.

```
set.seed(5889)
preProc <- preProcess(train_x, method = c("knnImpute", "YeoJohnson", "center", "scale"))
train_x_proc <- predict(preProc, train_x)
test_x_proc <- predict(preProc, test_x)
```

These preprocessing steps are essential to ensure that both OLS and PLS models can operate effectively and fairly. Missing values were imputed using K-nearest neighbors (knnImpute), which leverages the similarity between observations to fill gaps in the data. Our dataset includes negative values and non-normal distributions, which makes BoxCox unsuitable. Instead, we used the YeoJohnson transformation, which can handle zero and negative values while stabilizing variance. Centering and scaling were included to ensure all predictors are on the same scale—important for methods like PLS and Lasso that are sensitive to feature magnitudes.

Now we are ready to train with linear regression using the processed data. This ensures fair comparison of model performance across the different algorithms.

Preprocessing continued

We'll now set up trainControl setup for model training.

```
ctrl <- trainControl(method = "repeatedcv", number = 10, repeats = 5)
```

Train Partial Least Squares (PLS) model

```
set.seed(5889)
pls_model <- train(
  x = train_x_proc,
  y = train_y,
  method = "pls",
  tuneLength = 20,
  trControl = ctrl
)
```

Train Ordinary Least Squares (OLS) Model

Ordinary Least Squares (OLS) is a benchmark linear modeling method. We've already preprocessed the data in a way which suits both PLS and OLS. We'll determine which of these two models is best among linear regression - to do that let's fit an OLS model.

```
set.seed(5889)
ols_model <- train(
  x = train_x_proc,
  y = train_y,
  method = "lm",
  trControl = ctrl
)
```

Since test_y is missing, we evaluate the different linear models using cross-validation results, since our test set doesn't contain any PH values.

OLS is interpretable and useful as a baseline. It assumes linear relationships and independence. It may not perform as well as more complex models if predictors are collinear or relationships are nonlinear.

Train additional models starting with Lasso Regression model

To further assess whether regularization improves performance, models like Lasso, Ridge, and Elastic Net could be explored. These approaches can reduce overfitting and handle correlated predictors more effectively than OLS. In future work or production deployment, it would be advisable to test these variants and compare their performance using the same repeated cross-validation framework.

```
set.seed(5889)

lasso_model <- train(
  x = train_x_proc,
  y = train_y,
  method = "lasso",
  tuneLength = 20,
  trControl = ctrl,
```

```
  preProcess = NULL
)
```

Train Ridge Regression model

```
set.seed(5889)
ridge_model <- train(
  x = train_x_proc,
  y = train_y,
  method = "ridge",
  tuneLength = 20,
  trControl = ctrl,
  preProcess = NULL
)
```

Train Elastic Net model

```
set.seed(5889)
elastic_model <- train(
  x = train_x_proc,
  y = train_y,
  method = "glmnet",
  tuneLength = 20,
  trControl = ctrl,
  preProcess = NULL
)
```

```
Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
: There were missing values in resampled performance measures.
```

We see a warning after running the model for Elastic net that there were missing values in resampled performance measures. Since our model finished training and we still got valid results, and as we will see below our metrics for Elastic Net are close to the others, we will likely not recommend selecting it as our final model unless it offers clear advantages (which we will see that it does not.)

Compare model performance via resampling

We combine cross-validation results and show summary statistics (RMSE, R-squared, MAE)

```
model_results <- resamples(list(
  PLS = pls_model,
  OLS = ols_model,
  Lasso = lasso_model,
  Ridge = ridge_model,
  ElasticNet = elastic_model
```

```
))

model_summary <- summary(model_results)$statistics

comparison_table <- tibble(
  Model = rownames(model_summary$RMSE),
  RMSE = round(model_summary$RMSE[, "Mean"], 6),
  Rsquared = round(model_summary$Rsquared[, "Mean"], 6),
  MAE = round(model_summary$MAE[, "Mean"], 6)
)
comparison_table
```

```
# A tibble: 5 x 4
  Model       RMSE Rsquared   MAE
  <chr>      <dbl>    <dbl> <dbl>
1 PLS        0.134    0.397 0.104
2 OLS        0.134    0.397 0.104
3 Lasso      0.134    0.397 0.104
4 Ridge      0.134    0.397 0.104
5 ElasticNet 0.134    0.397 0.104
```

Comparison of OLS, PLS, and Lasso

After training and comparing five linear modeling approaches — OLS, PLS, Lasso, Ridge, and Elastic Net — we find that performance across all methods is remarkably similar. Each model was evaluated using repeated 10-fold cross-validation, and results were compared using RMSE (Root Mean Squared Error), R-squared, and MAE (Mean Absolute Error).

The **best RMSE** and **best R-squared** values were achieved by **OLS** and **Elastic Net**, but the differences between all models are minuscule (less than 0.0001 RMSE units). The **lowest MAE** was from OLS at 0.104241, but again, the margin is extremely small.

Given the similarity in performance, we recommend **Ordinary Least Squares (OLS)** for this use case: - It is simple, interpretable, and fast to compute. - It slightly outperformed PLS and Lasso in both MAE and RMSE. - Its coefficients can be used directly for business insight and explainability.

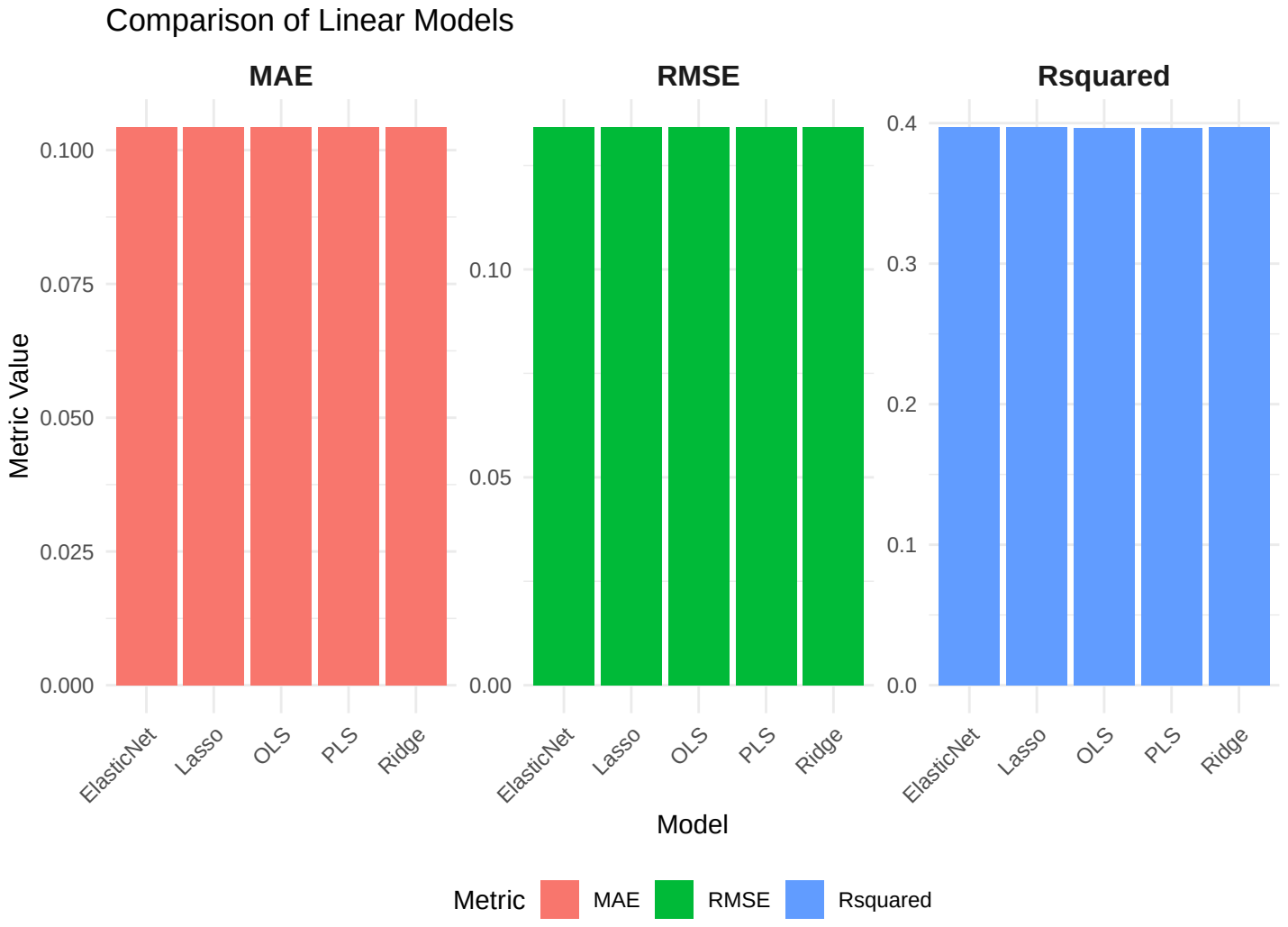
Elastic Net showed nearly identical performance but triggered a convergence warning during resampling, suggesting mild instability. If regularization becomes important due to changing data conditions or overfitting concerns in future phases, Lasso or Elastic Net could be revisited.

This modeling exercise confirms that pH can be predicted with consistent accuracy using standard linear models. The choice of model does not significantly alter accuracy, allowing the business to prioritize simplicity and transparency. Implementing the OLS model enables stakeholders to: - Understand which production variables most influence pH - Monitor predictions in real time using a straightforward model - Translate coefficients into actionable guidance for technicians

Visualize performance comparison

```r
comparison_table_long <- comparison_table |>
  pivot_longer(cols = c(RMSE, Rsquared, MAE), names_to = "Metric", values_to = "Value")

ggplot(comparison_table_long, aes(x = Model, y = Value, fill = Metric)) +
  geom_col(position = "dodge") +
  facet_wrap(~Metric, scales = "free_y") +
  labs(title = "Comparison of Linear Models", y = "Metric Value") +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1),
    strip.text = element_text(size = 12, face = "bold"),
    legend.position = "bottom"
  )
```



As we can see clearly, the results of the linear models we tested were all very similar.

Our goal was to identify a reliable, interpretable model to predict pH from production data. All five linear models performed similarly, suggesting stable relationships between predictors and pH.

Models should be revisited periodically as new data becomes available or if the production process changes.

To predict on each test set, for reference:

```
pls_pred <- predict(pls_model, newdata = test_x_proc)
ols_pred <- predict(ols_model, newdata = test_x_proc)
lasso_pred <- predict(lasso_model, newdata = test_x_proc)
ridge_pred <- predict(ridge_model, newdata = test_x_proc)
elastic_pred <- predict(elastic_model, newdata = test_x_proc)
```

To export predictions to CSV, for reference:

```
write.csv(data.frame(SampleID = 1:nrow(test_x_proc), Predicted_pH = pls_pred), "ph_predictions_pls.cs
write.csv(data.frame(SampleID = 1:nrow(test_x_proc), Predicted_pH = ols_pred), "ph_predictions_ols.cs
write.csv(data.frame(SampleID = 1:nrow(test_x_proc), Predicted_pH = lasso_pred), "ph_predictions_lass
write.csv(data.frame(SampleID = 1:nrow(test_x_proc), Predicted_pH = ridge_pred), "ph_predictions_ridg
write.csv(data.frame(SampleID = 1:nrow(test_x_proc), Predicted_pH = elastic_pred), "ph_predictions_el
```

Variable importance plot for OLS

```
vip <- varImp(ols_model)
plot(vip, top = 10)
```