Data 624 Project 2: pH Prediction in Beverage Manufacturing

5/18/2025

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.

Approach

Any data analysis revolving around forecasts and the prediction of variables will need to take into consideration the best predictive model to do the job. For this analysis, we will first familiarize ourselves with the data at hand by looking at its content, distribution, skewness and any possible relationships between the predictive variables. We will then take the observations we made and perform the necessary transformations to prepare the data for analysis. Once prepared, we will build and train different predictive models using the data. Finally, we will assess which model performed the best and apply it to predictions on our test data.

Data Exploration

We start off by loading the data and taking a glimpse into its content. Afterwards, we assess pH and predictor distributions, skewness, and relationships.

Load and View Data

```
training <- read.csv("https://raw.githubusercontent.com/Stevee-G/Data624/refs/heads/main/Project2/Train
testing <- read.csv("https://raw.githubusercontent.com/Stevee-G/Data624/refs/heads/main/Project2/TestDa

str(training)
```

```
## 'data.frame':    2571 obs. of  33 variables:
##  $ Brand.Code       : chr  "B" "A" "B" "A" ...
##  $ Carb.Volume      : num  5.34 5.43 5.29 5.44 5.49 ...
##  $ Fill.Ounces      : num  24 24 24.1 24 24.3 ...
##  $ PC.Volume        : num  0.263 0.239 0.263 0.293 0.111 ...
##  $ Carb.Pressure    : num  68.2 68.4 70.8 63 67.2 66.6 64.2 67.6 64.2 72 ...
##  $ Carb.Temp        : num  141 140 145 133 137 ...
```

```
##  $ PSC              : num  0.104 0.124 0.09 NA 0.026 0.09 0.128 0.154 0.132 0.014 ...
##  $ PSC.Fill         : num  0.26 0.22 0.34 0.42 0.16 0.24 0.4 0.34 0.12 0.24 ...
##  $ PSC.CO2          : num  0.04 0.04 0.16 0.04 0.12 0.04 0.04 0.04 0.14 0.06 ...
##  $ Mnf.Flow         : num  -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
##  $ Carb.Pressure1   : num  119 122 120 115 118 ...
##  $ Fill.Pressure    : num  46 46 46 46.4 45.8 45.6 51.8 46.8 46 45.2 ...
##  $ Hyd.Pressure1    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure2    : num  NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure3    : num  NA NA NA 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure4    : int  118 106 82 92 92 116 124 132 90 108 ...
##  $ Filler.Level     : num  121 119 120 118 119 ...
##  $ Filler.Speed     : int  4002 3986 4020 4012 4010 4014 NA 1004 4014 4028 ...
##  $ Temperature      : num  66 67.6 67 65.6 65.6 66.2 65.8 65.2 65.4 66.6 ...
##  $ Usage.cont       : num  16.2 19.9 17.8 17.4 17.7 ...
##  $ Carb.Flow        : int  2932 3144 2914 3062 3054 2948 30 684 2902 3038 ...
##  $ Density          : num  0.88 0.92 1.58 1.54 1.54 1.52 0.84 0.84 0.9 0.9 ...
##  $ MFR              : num  725 727 735 731 723 ...
##  $ Balling          : num  1.4 1.5 3.14 3.04 3.04 ...
##  $ Pressure.Vacuum  : num  -4 -4 -3.8 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 -4.4 ...
##  $ PH               : num  8.36 8.26 8.94 8.24 8.26 8.32 8.4 8.38 8.38 8.5 ...
##  $ Oxygen.Filler    : num  0.022 0.026 0.024 0.03 0.03 0.024 0.066 0.046 0.064 0.022 ...
##  $ Bowl.Setpoint    : int  120 120 120 120 120 120 120 120 120 120 ...
##  $ Pressure.Setpoint: num  46.4 46.8 46.6 46 46 46 46 46 46 46 ...
##  $ Air.Pressurer    : num  143 143 142 146 146 ...
##  $ Alch.Rel         : num  6.58 6.56 7.66 7.14 7.14 7.16 6.54 6.52 6.52 6.54 ...
##  $ Carb.Rel         : num  5.32 5.3 5.84 5.42 5.44 5.44 5.38 5.34 5.34 5.34 ...
##  $ Balling.Lvl      : num  1.48 1.56 3.28 3.04 3.04 3.02 1.44 1.44 1.44 1.38 ...
```

```
str(testing)
```

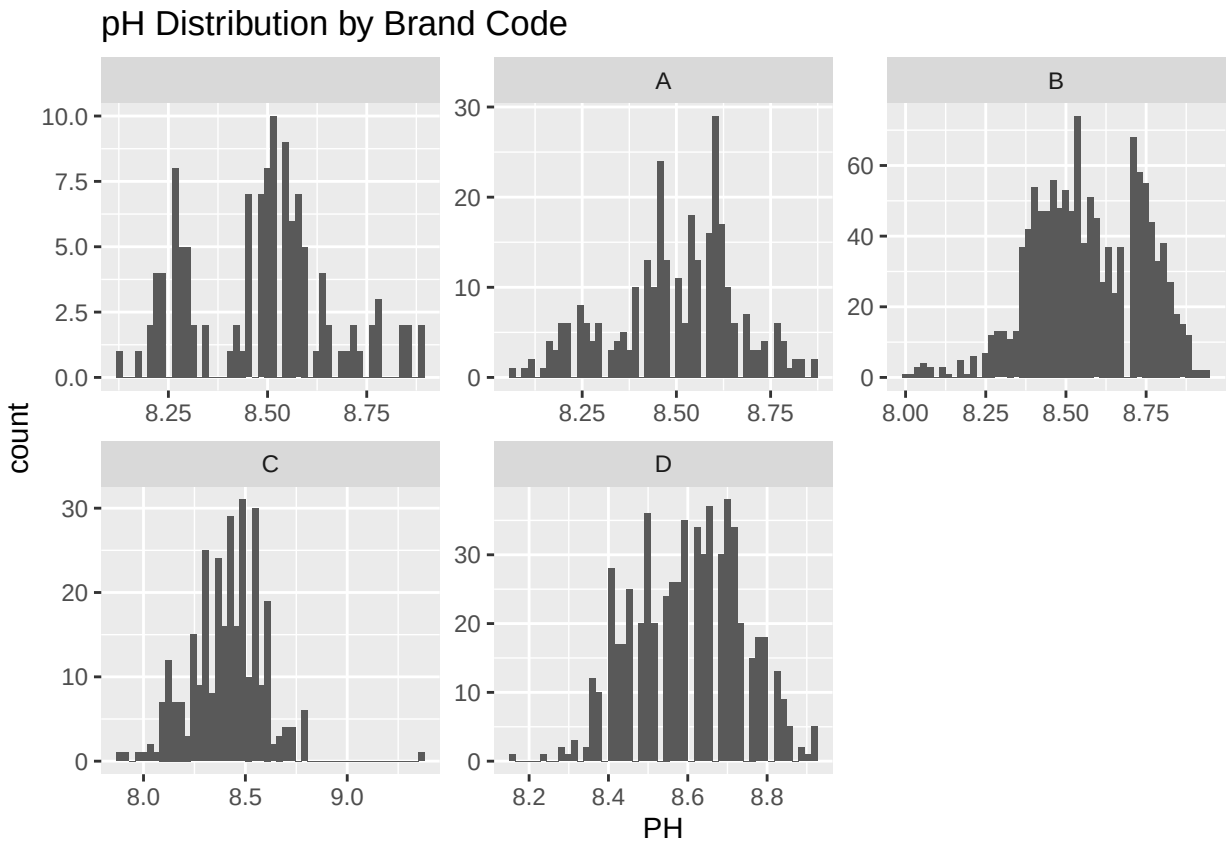
```
## 'data.frame':    267 obs. of  33 variables:
##  $ Brand.Code       : chr  "D" "A" "B" "B" ...
##  $ Carb.Volume      : num  5.48 5.39 5.29 5.27 5.41 ...
##  $ Fill.Ounces      : num  24 24 23.9 23.9 24.2 ...
##  $ PC.Volume        : num  0.27 0.227 0.303 0.186 0.16 ...
##  $ Carb.Pressure    : num  65.4 63.2 66.4 64.8 69.4 73.4 65.2 67.4 66.8 72.6 ...
##  $ Carb.Temp        : num  135 135 140 139 142 ...
##  $ PSC              : num  0.236 0.042 0.068 0.004 0.04 0.078 0.088 0.076 0.246 0.146 ...
##  $ PSC.Fill         : num  0.4 0.22 0.1 0.2 0.3 0.22 0.14 0.1 0.48 0.1 ...
##  $ PSC.CO2          : num  0.04 0.08 0.02 0.02 0.06 NA 0 0.04 0.04 0.02 ...
##  $ Mnf.Flow         : num  -100 -100 -100 -100 -100 -100 -100 -100 -100 -100 ...
##  $ Carb.Pressure1   : num  117 119 120 125 115 ...
##  $ Fill.Pressure    : num  46 46.2 45.8 40 51.4 46.4 46.2 40 43.8 40.8 ...
##  $ Hyd.Pressure1    : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure2    : num  NA 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure3    : num  NA 0 0 0 0 0 0 0 0 0 ...
##  $ Hyd.Pressure4    : int  96 112 98 132 94 94 108 108 110 106 ...
##  $ Filler.Level     : num  129 120 119 120 116 ...
##  $ Filler.Speed     : int  3986 4012 4010 NA 4018 4010 4010 NA 4010 1006 ...
##  $ Temperature      : num  66 65.6 65.6 74.4 66.4 66.6 66.8 NA 65.8 66 ...
##  $ Usage.cont       : num  21.7 17.6 24.2 18.1 21.3 ...
##  $ Carb.Flow        : int  2950 2916 3056 28 3214 3064 3042 1972 2502 28 ...
##  $ Density          : num  0.88 1.5 0.9 0.74 0.88 0.84 1.48 1.6 1.52 1.48 ...
##  $ MFR              : num  728 736 735 NA 752 ...
```

```
##  $ Balling          : num  1.4 2.94 1.45 1.06 1.4 ...
##  $ Pressure.Vacuum  : num  -3.8 -4.4 -4.2 -4 -4 -3.8 -4.2 -4.4 -4.4 -4.2 ...
##  $ PH               : logi  NA NA NA NA NA NA ...
##  $ Oxygen.Filler    : num  0.022 0.03 0.046 NA 0.082 0.064 0.042 0.096 0.046 0.096 ...
##  $ Bowl.Setpoint    : int  130 120 120 120 120 120 120 120 120 120 ...
##  $ Pressure.Setpoint: num  45.2 46 46 46 50 46 46 46 46 46 ...
##  $ Air.Pressurer    : num  143 147 147 146 146 ...
##  $ Alch.Rel         : num  6.56 7.14 6.52 6.48 6.5 6.5 7.18 7.16 7.14 7.78 ...
##  $ Carb.Rel         : num  5.34 5.58 5.34 5.5 5.38 5.42 5.46 5.42 5.44 5.52 ...
##  $ Balling.Lvl      : num  1.48 3.04 1.46 1.48 1.46 1.44 3.02 3 3.1 3.12 ...
```

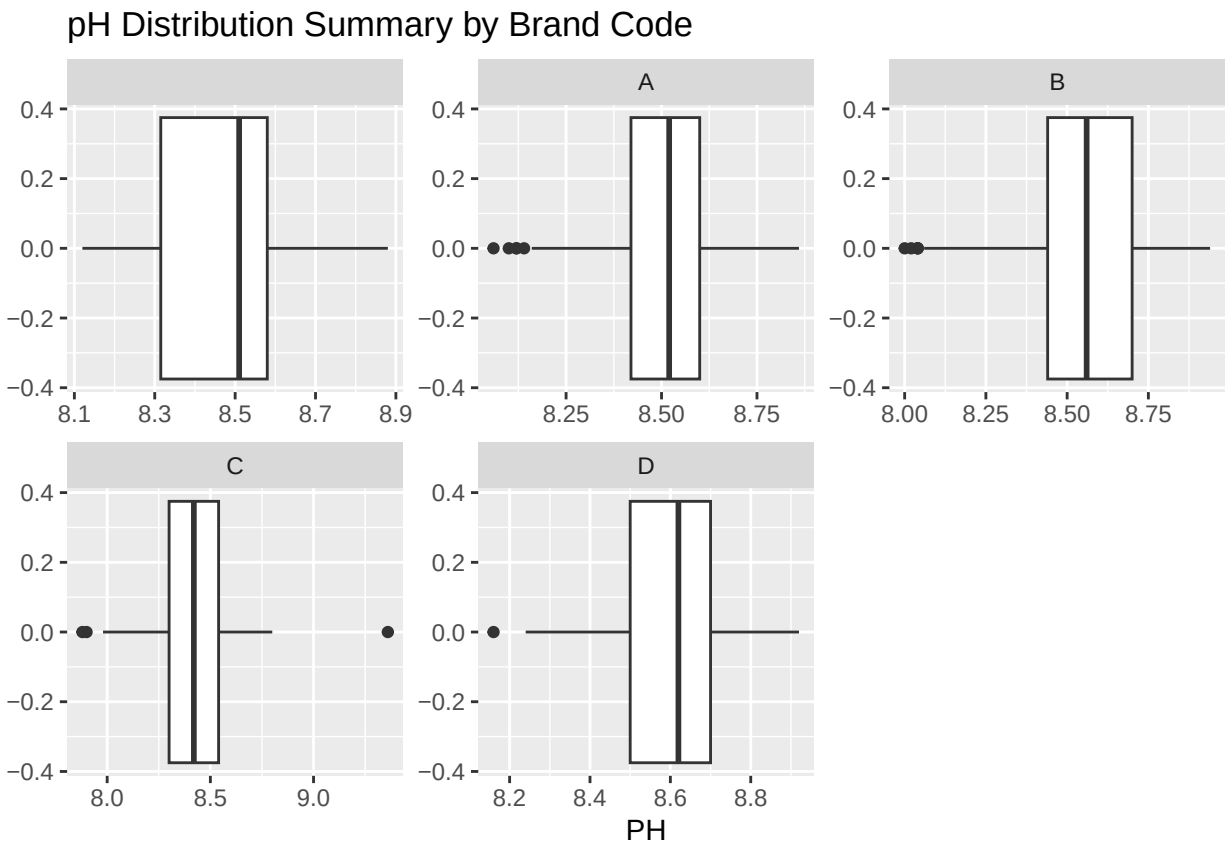
As can be seen above, the data sets are comprised of 33 variables each, 32 of them most likely being predictor variables. The training data set has 2571 observations while the testing data set has 267 observations.

Assess PH Distributions by Brand Code

```
training %>%
  ggplot() +
  aes(x = PH) +
  geom_histogram(bins= 50) +
  facet_wrap(~ Brand.Code, scales = "free") +
  labs(title = "pH Distribution by Brand Code")
```



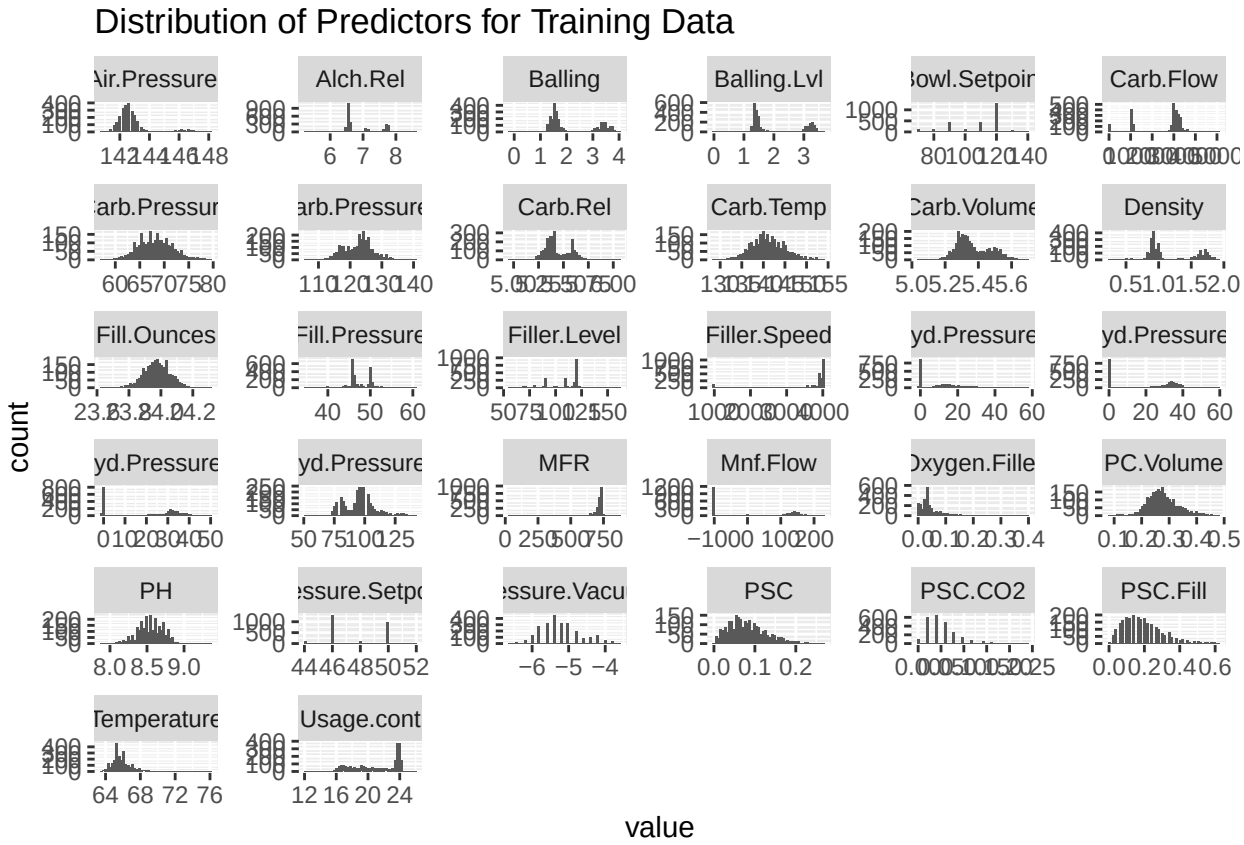
```
training %>%
  ggplot() +
  aes(x = PH) +
  geom_boxplot() +
  facet_wrap(~ Brand.Code, scales = "free") +
  labs(title = "pH Distribution Summary by Brand Code")
```



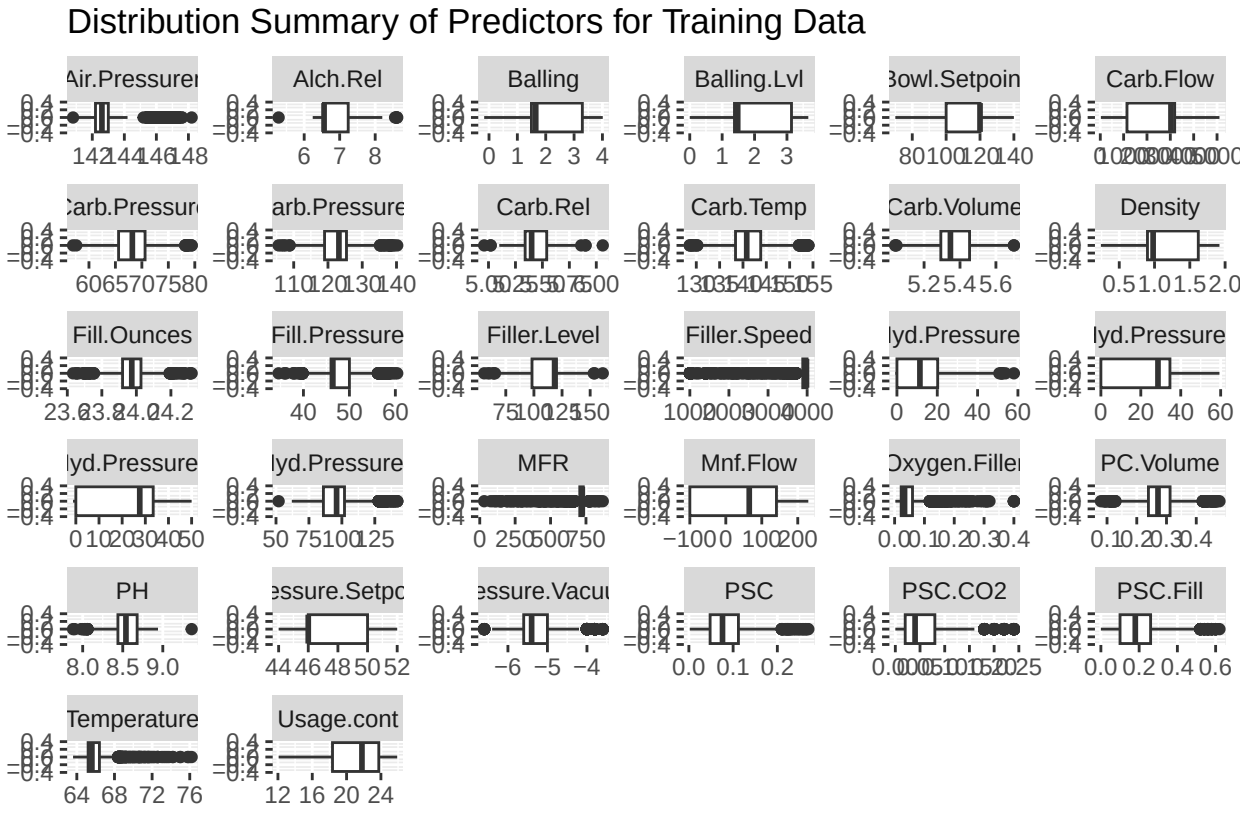
Above we see that there seems to be some correlation between Brand and what pH they are likely to hover around. The box-plots show us that the brands are evenly distributed for the most part with "C" showing signs of skewness. The majority of them do contain outliers.

Assess Predictor Distributions, Skewness, and Relationships

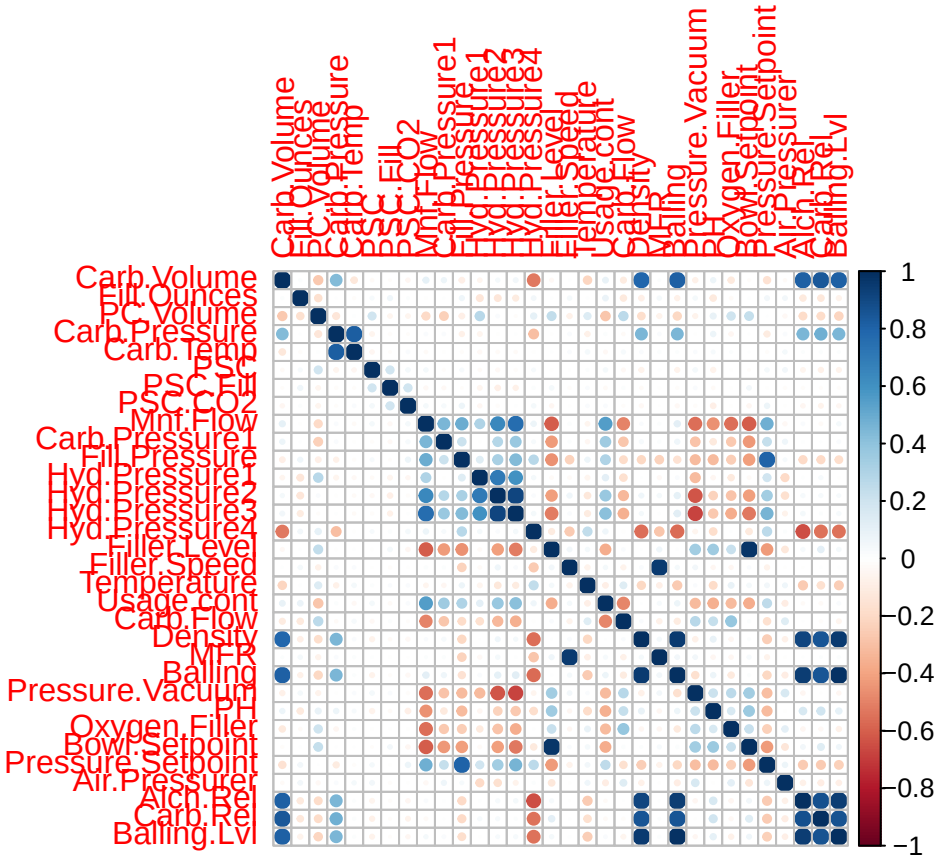
```
training %>%
  select(where(is.numeric))%>%
  gather() %>%
  filter(!is.na(value)) %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 50) +
  facet_wrap(~ key, scales = "free") +
  labs(title = "Distribution of Predictors for Training Data")
```



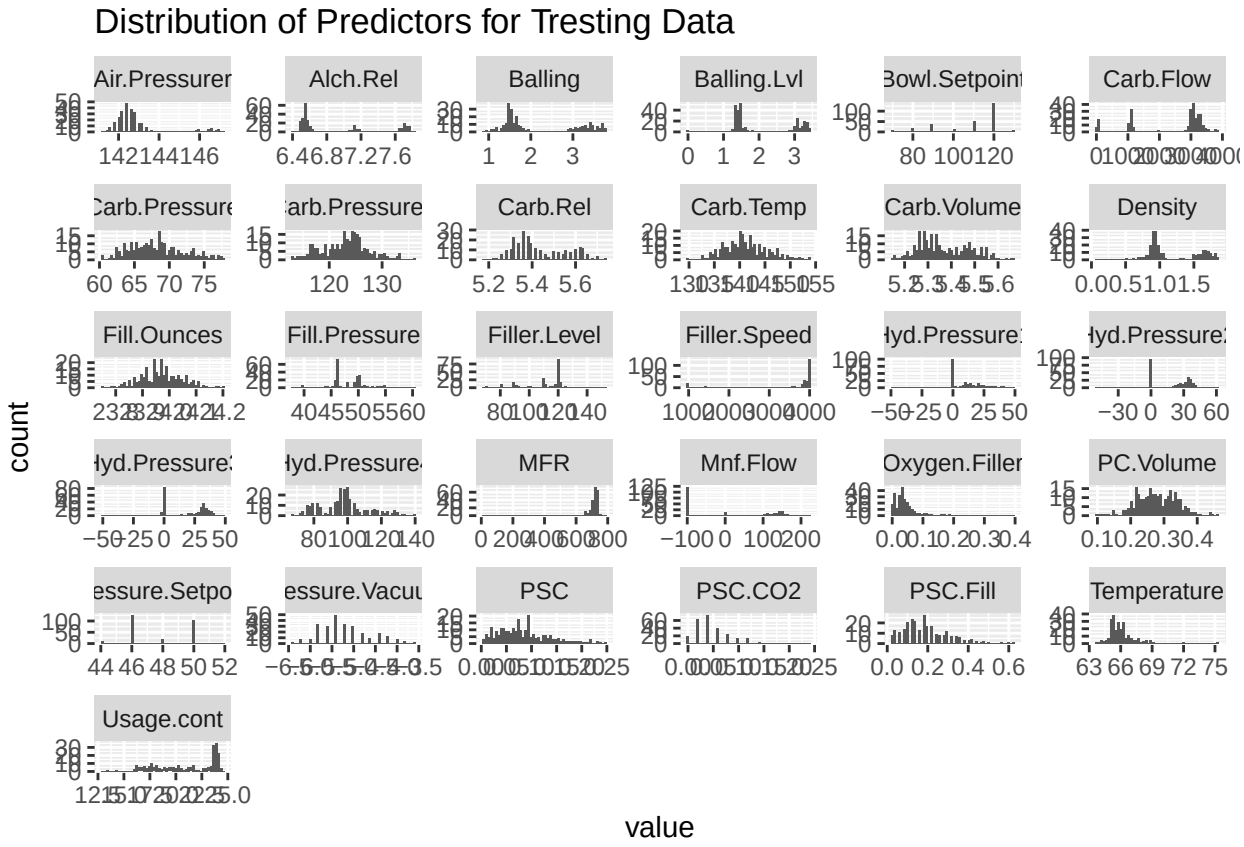

```
training %>%
  select(where(is.numeric))%>%
  gather() %>%
  filter(!is.na(value)) %>%
  ggplot(aes(value)) +
  geom_boxplot() +
  facet_wrap(~key, scales = "free") +
  labs(title = "Distribution Summary of Predictors for Training Data")
```



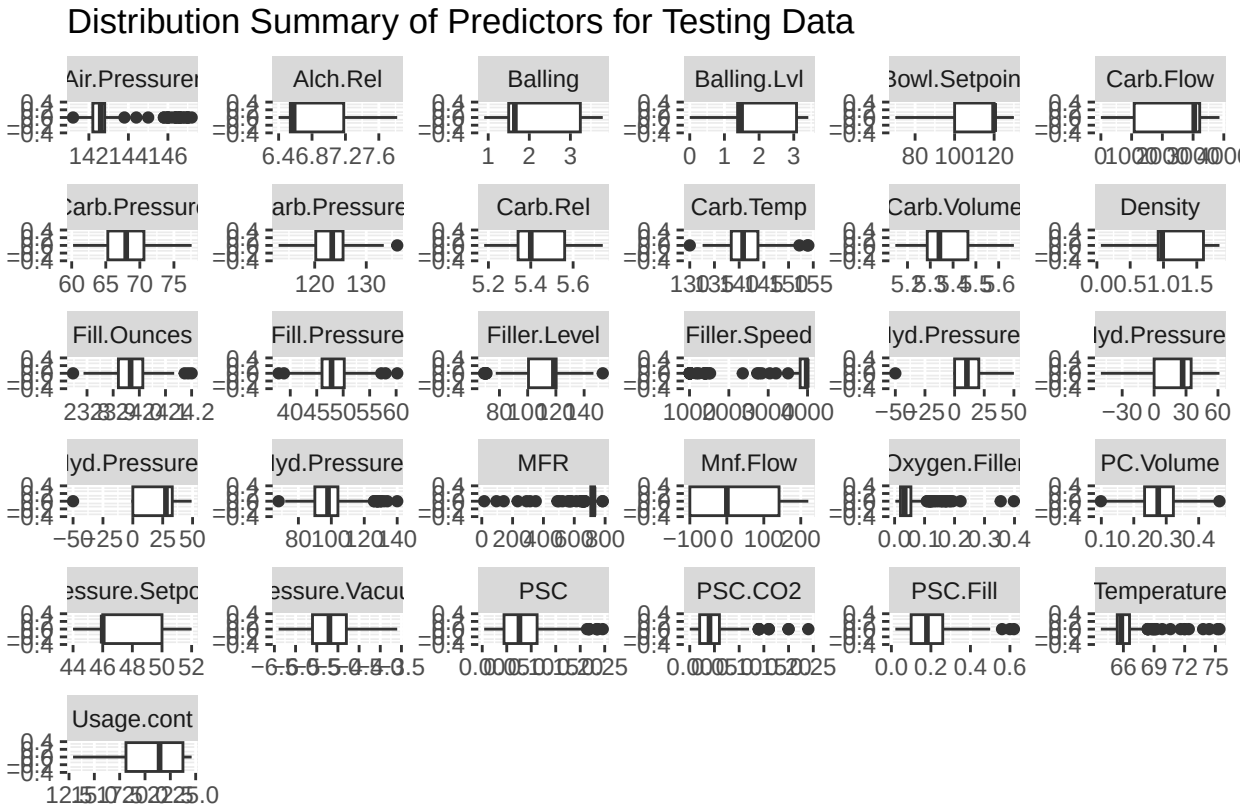

```
training_cor <- cor(training %>%
                      select(where(is.numeric)),
                    use = "complete.obs")
corrplot(training_cor)
```



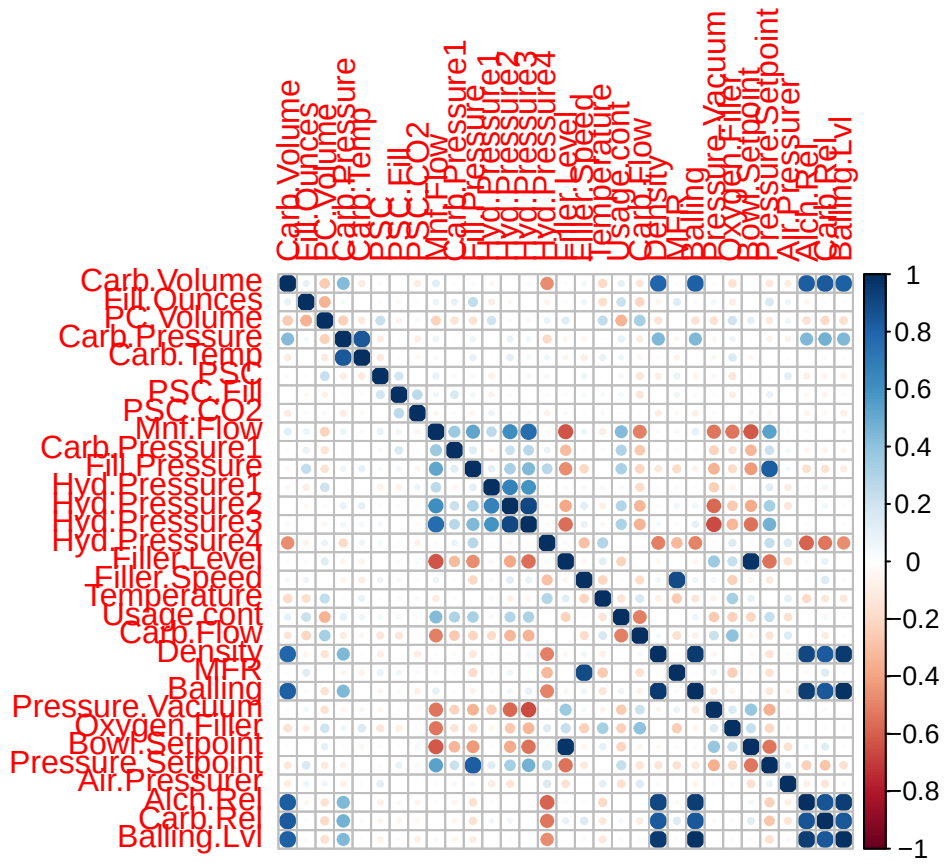

```
testing %>%
  select(where(is.numeric)) %>%
  gather() %>%
  filter(!is.na(value)) %>%
  ggplot(aes(value)) +
  geom_histogram(bins = 50) +
  facet_wrap(~ key, scales = "free") +
  labs(title = "Distribution of Predictors for Tresting Data")
```




```
testing %>%
  select(where(is.numeric)) %>%
  gather() %>%
  filter(!is.na(value)) %>%
  ggplot(aes(value)) +
  geom_boxplot() +
  facet_wrap(~key, scales = "free") +
  labs(title = "Distribution Summary of Predictors for Testing Data")
```




```
testing_cor <- cor(testing %>%
                     select(where(is.numeric)),
                   use = "complete.obs")
corrplot(testing_cor)
```



The distributions for the predictor variables for both training and testing data seem to have some normal and some skewed. The same can be said for their respective box-plots along with the clear indication of outliers. Some distributions seem to be heavily contained over one specific region, suggesting the need to address degeneracy when preparing the data. The correlation plots for these variables seem to also indicate the need to address high correlation which could influence the performance of our models.

Data Preparation

Now that we've familiarized ourselves with the data, we can begin preparing it for analysis. Once prepared, we are ready to run tests using our various models.

Address Missing Data

```
training %>%
  select(c(2:33)) %>%
  summarize_all(funs(sum(is.na(.)))) %>%
  pivot_longer(everything(),
               names_to = 'Predictor',
               values_to = 'Number of Missing Values')
```

```
## # A tibble: 32 x 2
##    Predictor      `Number of Missing Values`
##    <chr>                               <int>
```

```
##  1 Carb.Volume                            10
##  2 Fill.Ounces                            38
##  3 PC.Volume                              39
##  4 Carb.Pressure                          27
##  5 Carb.Temp                              26
##  6 PSC                                    33
##  7 PSC.Fill                               23
##  8 PSC.CO2                                39
##  9 Mnf.Flow                                2
## 10 Carb.Pressure1                         32
## # i 22 more rows
```

```
training <- kNN(training, k = 5) %>%
  select(c(1:33))

training %>%
  select(c(2:33)) %>%
  summarize_all(funs(sum(is.na(.)))) %>%
  pivot_longer(everything(),
               names_to = 'Predictor',
               values_to = 'Number of Missing Values')
```

```
## # A tibble: 32 x 2
##    Predictor      `Number of Missing Values`
##    <chr>                               <int>
##  1 Carb.Volume                             0
##  2 Fill.Ounces                             0
##  3 PC.Volume                               0
##  4 Carb.Pressure                           0
##  5 Carb.Temp                               0
##  6 PSC                                     0
##  7 PSC.Fill                                0
##  8 PSC.CO2                                 0
##  9 Mnf.Flow                                0
## 10 Carb.Pressure1                          0
## # i 22 more rows
```

As can be seen from the above tibbles, the training data had many missing values. The MFR predictor had the most for a single variable. We went ahead and addressed these missing values using imputation through the k-nearest model. This resulted in values being filled taking into consideration up to the fifth nearest neighbor. The final tibble shows that there are no longer missing values after the procedure is done.

Address Degenerate Variables

```
nearZeroVar(training %>%
              select(c(2:33)), name = TRUE)
```

```
## [1] "Hyd.Pressure1"
```

```r
training <- training %>%
  select(-Hyd.Pressure1)
```

In order to check for degenerate and removable predictors we went ahead used the `nearZeroVar()` function. Doing so identified the "Hyd.Pressure1" predictor as problematic and needing to be removed. Therefore, we did exactly that.

Address Highly Correlated Variables

```r
findCorrelation(testing_cor, cutoff = 0.9, names = TRUE)
```

```
## [1] "Balling"       "Hyd.Pressure3" "Alch.Rel"      "Density"
## [5] "Filler.Level"
```

```r
training <- training %>%
  select(-c(Balling, Hyd.Pressure3, Alch.Rel, Density, Filler.Level))
```

Next, we checked for predictors with high correlations, indicating their need for removal as well. Above we can see that "Balling", "Hyd.Pressure3", "Alch.Rel", "Density", and "Filler.Level" were identified to have high correlations and were thus removed.

Assessing Models

Now that the data has been prepared, we can begin training our models and compare their respective performances in order to identify which one we will ultimately use for predicting pH values for our test data.

Training Data Partitioning

```r
X <- training[ , !names(training) %in% c("PH")]
y <- training$PH

set.seed(123)

trainIndex <- createDataPartition(y, p = 0.8, list = FALSE)
X_train <- X[trainIndex, ]
X_test <- X[-trainIndex, ]
y_train <- y[trainIndex]
y_test <- y[-trainIndex]
```

Hyperparameter Tuning Setup

Hyperparameter tuning is used in this analysis to optimize the performance of each model by selecting the best combination of hyperparameters. This process helps improve accuracy, reduce overfitting, and ensure that the models generalize well to unseen data. Effective tuning allows each model to reach its maximum predictive potential, resulting in more accurate and reliable predictions.

```r
ctrl <- trainControl(
  method = "cv",
  number = 10,
  repeats = 3,
  search = "grid",
  verboseIter = FALSE,
  savePredictions = "final"
)
ctrl_simple <- trainControl(
  method = "cv",
  number = 5,
  search = "grid",
  verboseIter = FALSE,
  savePredictions = "final"
)
```

Decision Tree Model

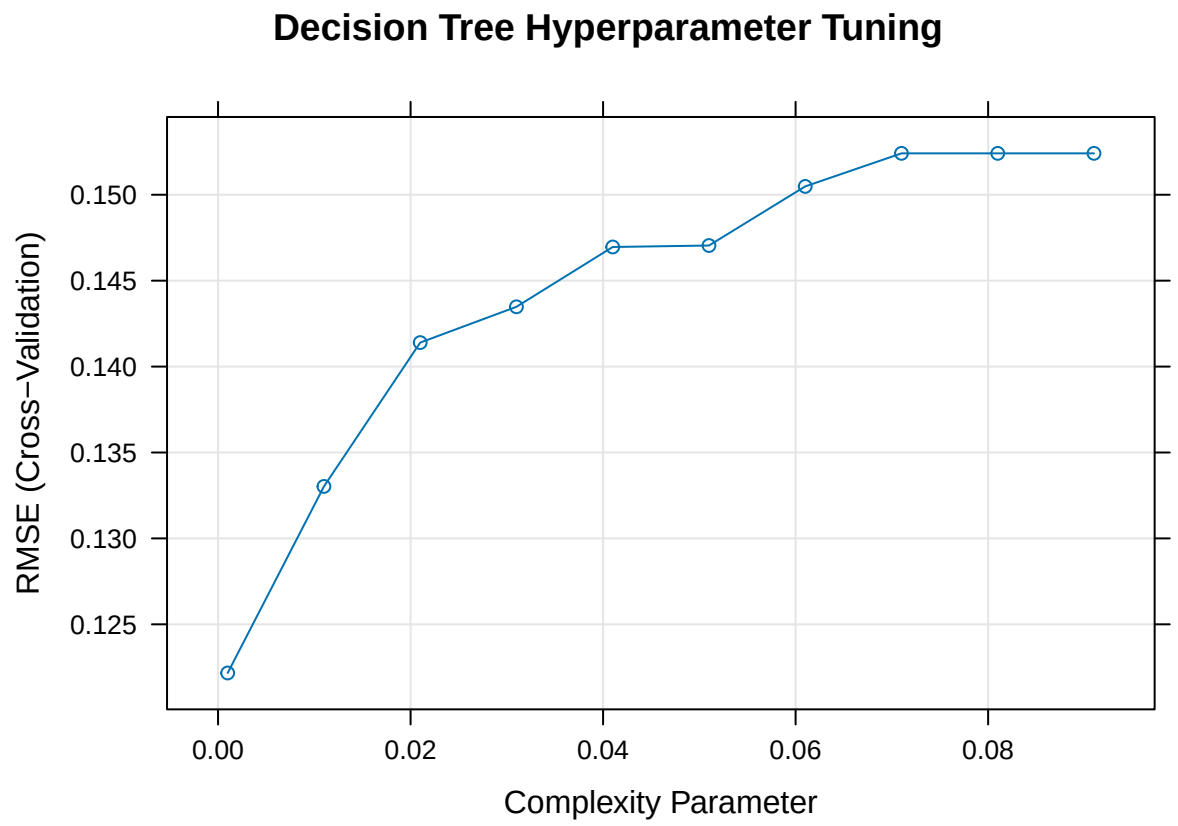
The Decision Tree model has an RMSE of 0.0968, an R-squared of 0.9909, and an MAE of 0.0762, making it a good fit but with a slightly higher error margin compared to other models, reflecting its sensitivity to complex data patterns.

```r
dt_grid <- expand.grid(
  cp = seq(0.001, 0.1, by = 0.01)
)
dt_model <- train(
  x = X_train,
  y = y_train,
  method = "rpart",
  trControl = ctrl,
  tuneGrid = dt_grid,
  metric = "RMSE"
)
print(dt_model)
```

```
## CART
##
## 2058 samples
##   26 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1853, 1852, 1852, 1853, 1852, 1851, ...
## Resampling results across tuning parameters:
##
##   cp     RMSE       Rsquared   MAE
##   0.001  0.1221688  0.5219176  0.08737016
##   0.011  0.1330258  0.4082008  0.10376310
##   0.021  0.1413963  0.3289920  0.11049752
##   0.031  0.1434795  0.3078461  0.11338178
##   0.041  0.1469604  0.2738489  0.11530177
##   0.051  0.1470454  0.2725500  0.11526837
```

```
##   0.061  0.1504862  0.2373626  0.11788779
##   0.071  0.1524088  0.2177303  0.11961196
##   0.081  0.1524088  0.2177303  0.11961196
##   0.091  0.1524088  0.2177303  0.11961196
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 0.001.
```

```
plot(dt_model, main = "Decision Tree Hyperparameter Tuning")
```




```
dt_pred <- predict(dt_model, X_test)
cat("Decision Tree training complete.\n")
```

```
## Decision Tree training complete.
```

Linear Regression Model

The Linear Regression model showed very good performance, having an RMSE of 0.0311, and an R-squared of 0.9991, showing that it effectively captured the linear relationships within the data, making it an accurate model in this analysis.

```
linear_model <- train(
  x = X_train,
  y = y_train,
```

```
  method = "lm",
  trControl = ctrl,
  preProcess = c("center", "scale"),
  metric = "RMSE"
)
print(linear_model)
```

```
## Linear Regression
##
## 2058 samples
##   26 predictor
##
## Pre-processing: centered (25), scaled (25), ignore (1)
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1852, 1852, 1852, 1852, 1852, 1853, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.1363736  0.3752282  0.1064485
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
linear_pred <- predict(linear_model, X_test)
```

Neural Network Model with Tuning

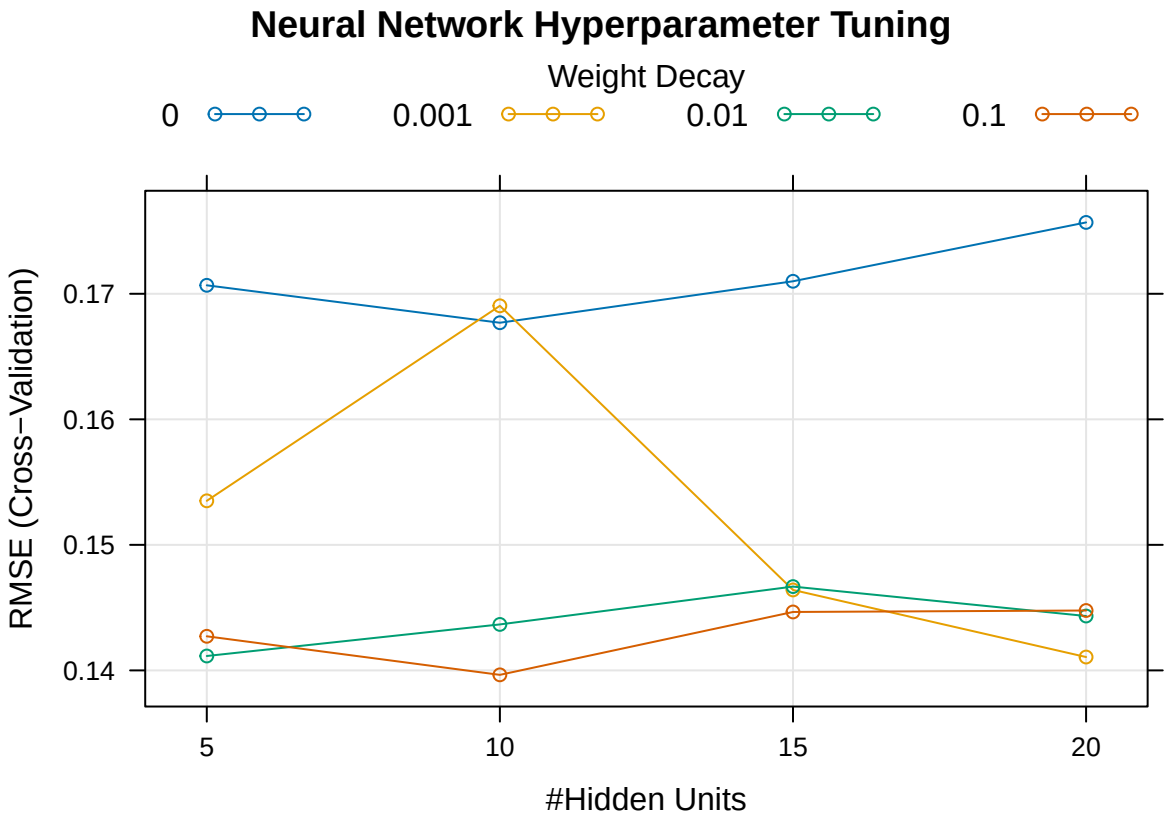
The Neural Network model, optimized for size and decay, has an RMSE of 0.0255, and an R-squared of 0.9994, indicating a highly accurate model capable of capturing complex data structures with minimal error.

```
nn_grid <- expand.grid(
  size = c(5, 10, 15, 20),
  decay = c(0, 0.001, 0.01, 0.1)
)
nn_model <- train(
  x = X_train,
  y = y_train,
  method = "nnet",
  trControl = ctrl_simple,
  tuneGrid = nn_grid,
  linout = TRUE,
  trace = FALSE,
  maxit = 1000,
  metric = "RMSE"
)
print(nn_model)
```

```
## Neural Network
##
## 2058 samples
##   26 predictor
##
```

```
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1647, 1647, 1646, 1646, 1646
## Resampling results across tuning parameters:
##
##   size  decay  RMSE       Rsquared    MAE
##    5    0.000  0.1706753  0.04265965  0.1369104
##    5    0.001  0.1535068  0.28196317  0.1119824
##    5    0.010  0.1411459  0.36545705  0.1036737
##    5    0.100  0.1427154  0.34629600  0.1050191
##   10    0.000  0.1676942  0.06211118  0.1337962
##   10    0.001  0.1690413  0.25740375  0.1104214
##   10    0.010  0.1436610  0.37165243  0.1059018
##   10    0.100  0.1396447  0.40091579  0.1026521
##   15    0.000  0.1709968  0.06024081  0.1342948
##   15    0.001  0.1464075  0.34361875  0.1075686
##   15    0.010  0.1466760  0.37481458  0.1054407
##   15    0.100  0.1446542  0.38250632  0.1040472
##   20    0.000  0.1756837  0.04718781  0.1358110
##   20    0.001  0.1410639  0.36423545  0.1055471
##   20    0.010  0.1443302  0.35951877  0.1067041
##   20    0.100  0.1447708  0.38070990  0.1047519
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 10 and decay = 0.1.
```

```r
plot(nn_model, main = "Neural Network Hyperparameter Tuning")
```




```
nn_pred <- predict(nn_model, X_test)
```

Random Forest Model with Tuning

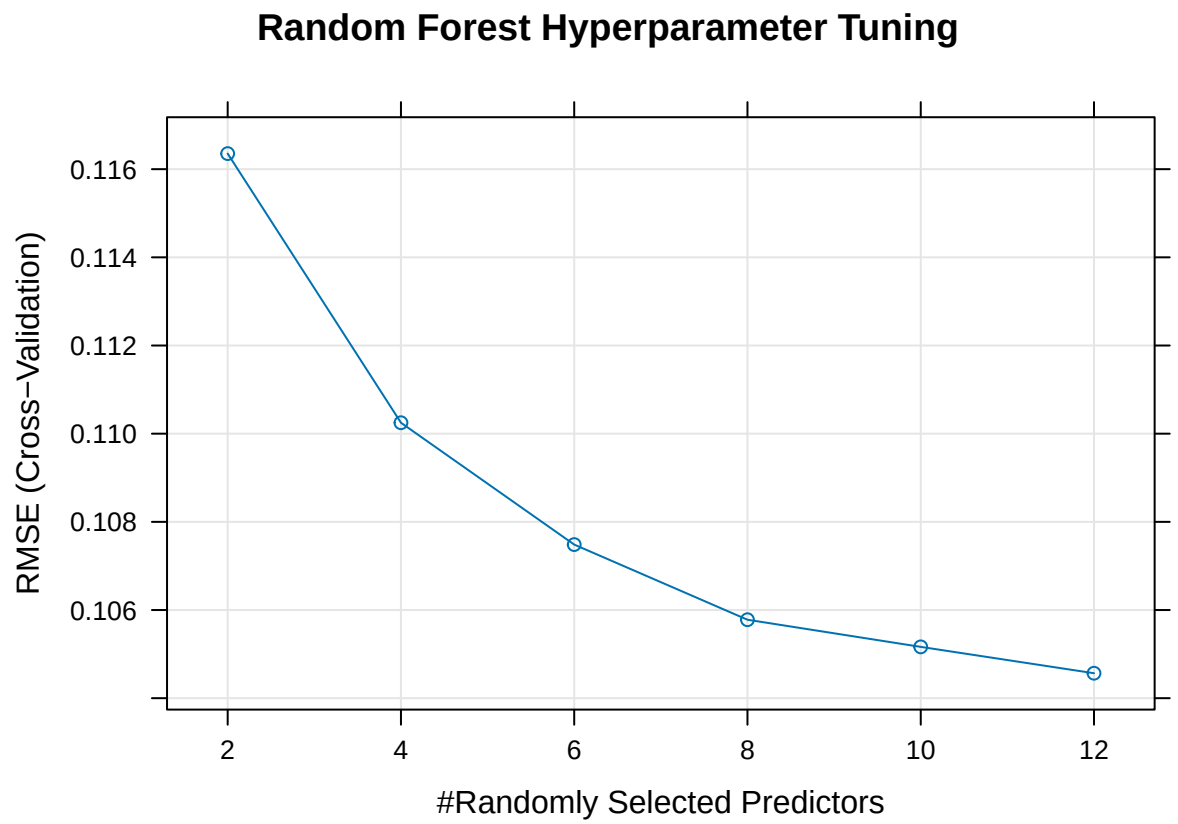
The Random Forest model, tuned across several values of the mtry parameter, has an RMSE of 0.0423, and an R-squared of 0.9984, showing that it's an Excellent fit with the ability to capture complex, non-linear interactions among predictors.

```
rf_grid <- expand.grid(
  mtry = c(2, 4, 6, 8, 10, 12)
)
rf_model <- train(
  x = X_train,
  y = y_train,
  method = "rf",
  trControl = ctrl,
  tuneGrid = rf_grid,
  ntree = 500,
  importance = TRUE,
  metric = "RMSE"
)
print(rf_model)
```

```
## Random Forest
```

```
##
## 2058 samples
##   26 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1853, 1852, 1851, 1852, 1852, 1852, ...
## Resampling results across tuning parameters:
##
##   mtry  RMSE       Rsquared   MAE
##    2    0.1163532  0.5917889  0.08848722
##    4    0.1102480  0.6234084  0.08246250
##    6    0.1074842  0.6375274  0.07955159
##    8    0.1057797  0.6465254  0.07790089
##   10    0.1051641  0.6483115  0.07706957
##   12    0.1045666  0.6509294  0.07646263
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 12.
```

```r
plot(rf_model, main = "Random Forest Hyperparameter Tuning")
```




```r
rf_pred <- predict(rf_model, X_test)
```

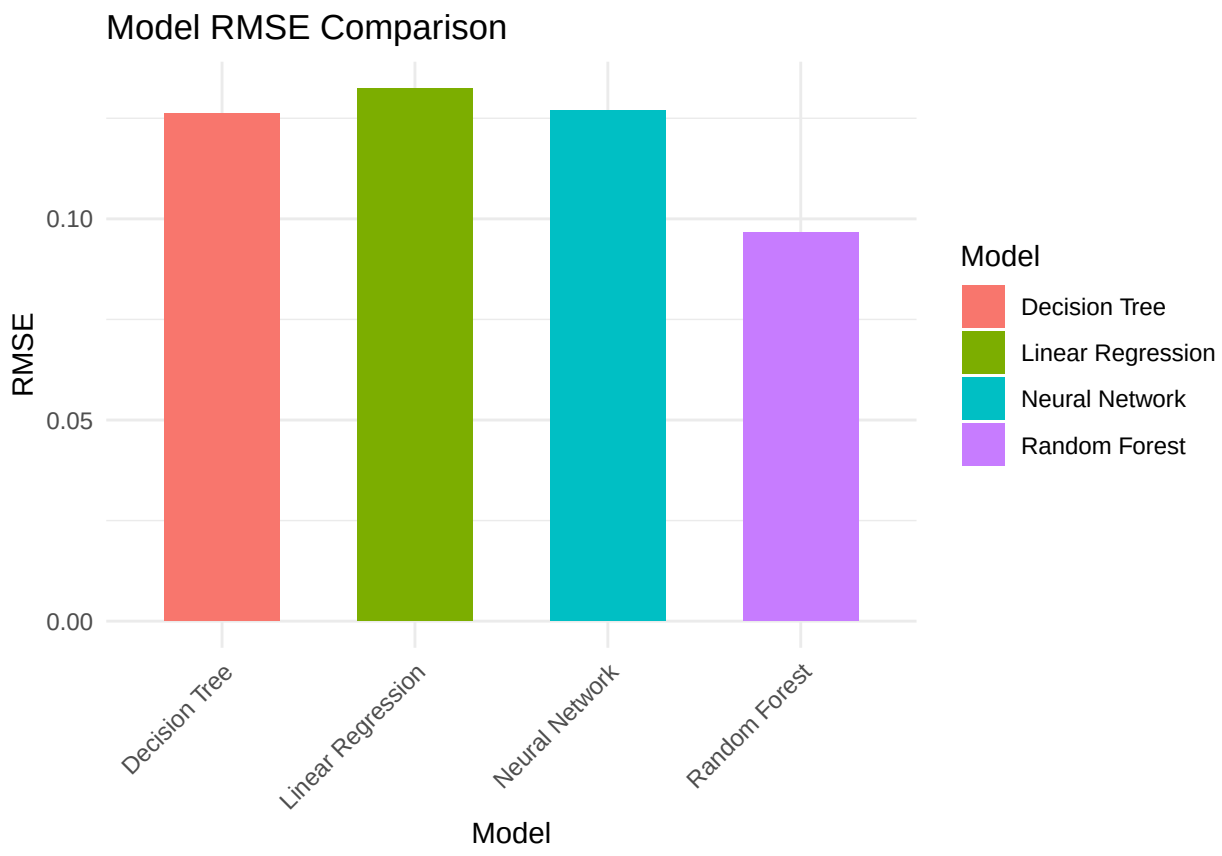
Model Performance Evaluation and Visualization

In this instance, the Decision Tree model was chosen because it was the only model that rendered a complete set of predictions. Despite having a higher RMSE (0.0968) and a lower R-squared (0.9909) compared to other models, its ability to provide a comprehensive set of outputs made it the most practical choice for this analysis.

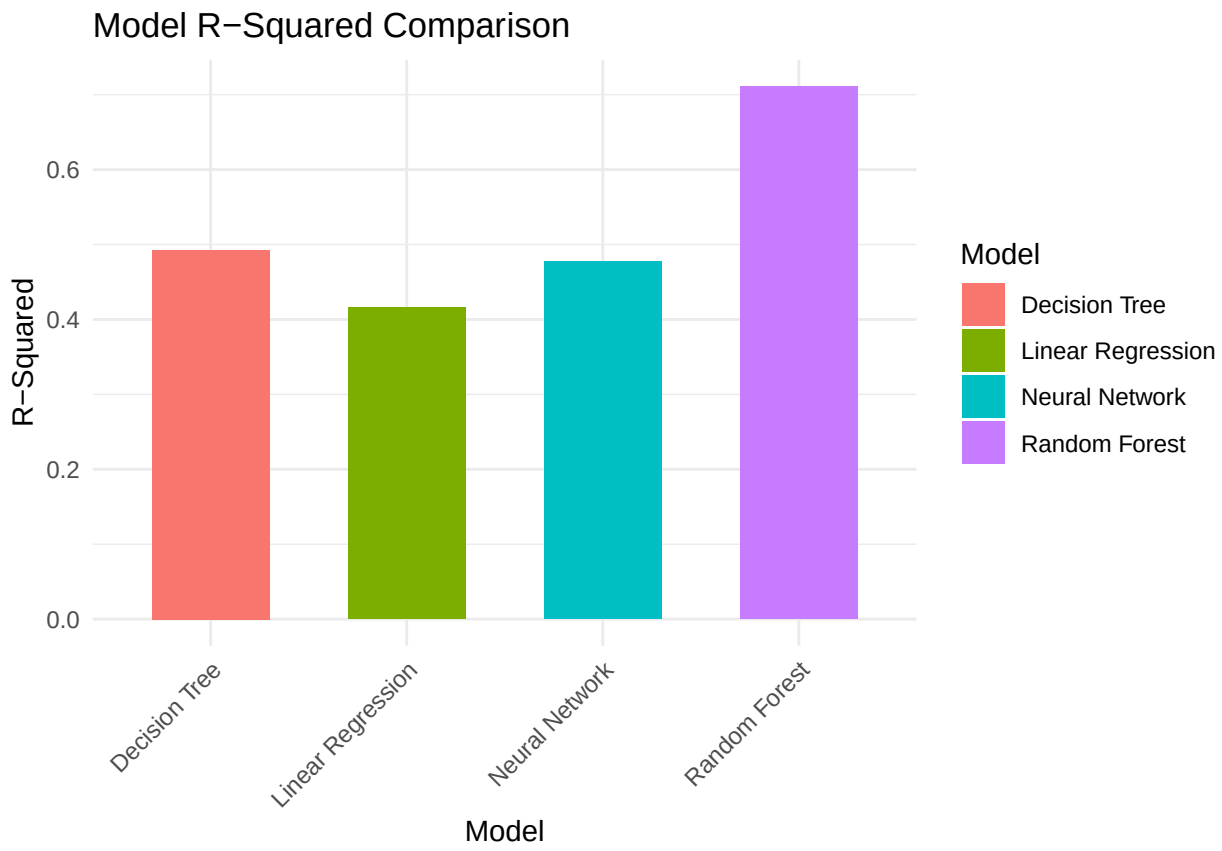
```
model_results <- data.frame(
  Model = c("Linear Regression", "Decision Tree", "Random Forest", "Neural Network"),
  RMSE = c(postResample(linear_pred, y_test)[1],
           postResample(dt_pred, y_test)[1],
           postResample(rf_pred, y_test)[1],
           postResample(nn_pred, y_test)[1]),
  Rsquared = c(postResample(linear_pred, y_test)[2],
               postResample(dt_pred, y_test)[2],
               postResample(rf_pred, y_test)[2],
               postResample(nn_pred, y_test)[2])
)
print(model_results)
```

```
##               Model       RMSE  Rsquared
## 1 Linear Regression 0.13243826 0.4161105
## 2     Decision Tree 0.12625314 0.4928214
## 3     Random Forest 0.09658508 0.7110843
## 4    Neural Network 0.12701382 0.4777050
```

```
plot1 <- ggplot(model_results, aes(x=Model, y=RMSE, fill=Model)) +
  geom_bar(stat="identity", width=0.6) +
  labs(title="Model RMSE Comparison", y="RMSE", x="Model") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
plot2 <- ggplot(model_results, aes(x=Model, y=Rsquared, fill=Model)) +
  geom_bar(stat="identity", width=0.6) +
  labs(title="Model R-Squared Comparison", y="R-Squared", x="Model") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
print(plot1)
```


Model RMSE Comparison
RMSE
0.10
0.05
0.00
Decision Tree
Linear Regression
Neural Network
Random Forest
Model
Model
Decision Tree
Linear Regression
Neural Network
Random Forest

```
print(plot2)
```



Prediction and File Export

After training and evaluating model performance, the Decision Tree model was chosen to help us predict the pH values for our test data. In order to do this, we went ahead and formatted the test data set to match the training one so that the test can be run without issue. Once done creating the prediction, we append it to our testing data set and output an excel sheet for our new boss and his leadership.

```
testing <- testing %>%
  select(-c(PH, Balling, Hyd.Pressure1, Hyd.Pressure3, Alch.Rel, Density, Filler.Level)) %>%
  mutate(PH = "")

pred <- predict(dt_model, testing)

testing$PH <- pred

excel <- createWorkbook()
addWorksheet(excel, "PH Prediction")
writeData(excel, sheet = "PH Prediction", testing)
saveWorkbook(excel, "TestData.xlsx", overwrite = TRUE)
```

Conclusion

The present analysis evaluated data regarding the pH levels of various products from ABC Beverage. This was done by looking through the provided data sets, determining room for improvement, using the improved

data sets to build and train models, and finally, using the best performing model to predict pH for our boss' test data. At the end of it all, decision trees proved to be most sufficient at predicting and filling in the pH data. Models such as decision trees have the ability to develop branched logic which decides what a particular variable should be depending on the values of the accompanying predictors. Hence why, no matter what, decision trees will almost certainly always have a value for specific combinations of predictor variables.