Propose a data analysis that sounds interesting to you

STAT 5610 *Statistical Learning*
Phebe Chen

Forecasting NHL Team Points using Statistical Learning

Background

In this data analysis, I aim to forecast NHL team points using various statistical learning methods. NHL team points are calculated based on wins and overtime losses. Specifically, teams earn two points for a win (whether in regulation, overtime, or a shootout), one point for an overtime loss (whether in overtime or a shootout), and zero points for a loss in regulation. These points contribute to a team's standing in the league, and projecting how many points a team will accumulate over the course of a full 82-game season can provide valuable insights into a team's potential performance.

To make these predictions, I plan on using various team performance metrics that capture different aspects of a team's play. These metrics include offensive and defensive statistics, special teams performance (e.g., power play percentage and penalty kill percentage), shot metrics, possession statistics, and scoring chance data. Statistical learning methods, including machine learning algorithms like random forests, bagging, and boosting will be applied to model these relationships and forecast the points that teams are likely to earn throughout the season.

Dataset

The data used for this analysis comes from hockey-reference.com, a reputable source for historical NHL statistics. The dataset includes five years of team performance data from the 2017-2019 and 2021-2024 seasons. I excluded the 2019-2020 and 2020-2021 seasons due to the irregularities caused by the COVID-19 pandemic. Specifically, the 2019-2020 season had an abbreviated schedule due to the pandemic, and the 2020-2021 season was affected by a lockout, with teams playing only 56 games instead of the usual 82. These disruptions made it difficult to maintain consistent year-to-year comparisons, so I decided to focus on seasons where teams played a full 82-game schedule. These five years of team performance data totalled to 158 observations.

The dataset contains a wide range of variables, covering multiple aspects of team performance. These variables include:

- Per Game Metrics: Team goals scored, goals against, and per game statistics.
- Special Teams: Power play percentage (a situation in which the team has a temporary numerical advantage because an opposing player/s are in the penalty box), penalty kill percentage (opposite of power play, situation in which the team has a temporary numerical disadvantage because a player/s are in the penalty box), and other related metrics.
- Shot Data: Shots on goal, shooting percentage, shots against, save percentage
- Possession Metrics: Corsi and Fenwick (5v5), both of which measure shot attempts, and other possession-related statistics.
- Scoring Chances: Data related to scoring opportunities, scoring chances for and against and high danger scoring for and against

Before modeling, I analyzed the relationship between the performance metrics and the target variable—points (PTS). I used a correlation matrix to identify how each of these performance metrics was related to the total points teams accumulated in a season. This analysis aided in determining which variables had the strongest associations with team points, helping to guide the selection of features for predictive modeling.

Table 1: Variables Used in the Analysis

Variable	**Interpretation**
PTS	Points
AvAge	Average age of team
SOS	Strength of schedule: A rating of strength of schedule
PPp	Power play percentage: power play goals/power play opportunities
PKp	Penatly kill percentage: power play goals against/power play opportunities against
oPIMpG	Opponent penalties in minutes per game
Sp	Shooting %: goals for/shots on goal
SVp	Save %: goals against/shots against
PDO	Shooting % + Save %
CFp	Corsi for % at 5 on 5: CF/(CF + CA) CF: Corsi for (shots + blocks + misses) CA: Corsi against (shots + blocks + misses)
FFp	Fenwick for % at 5 on 5: FF/(FF + FA) FF: Fenwick for (shots + misses) FA: Fenwick against (shots + misses)
axDiff	Actual goal differential minus expected goal differential. A positive differential would indicate a team is converting or stopping an inordinate amount of good chances compared to league average. A negative differential would indicate a team is getting more good chances, but not converting or is allowing more than league norms.
SCFp	Percentage of scoring chances in this team's favor
HDFp	Percentage of high-danger scoring chances in this team's favor
HDCp	Percentage of high-danger scoring chances that are converted to goals, for this team.
HDCOp	Percentage of high-danger scoring chances that are converted to goals, for this team's opponents.

Figure 1: Correlation Matrix using Points

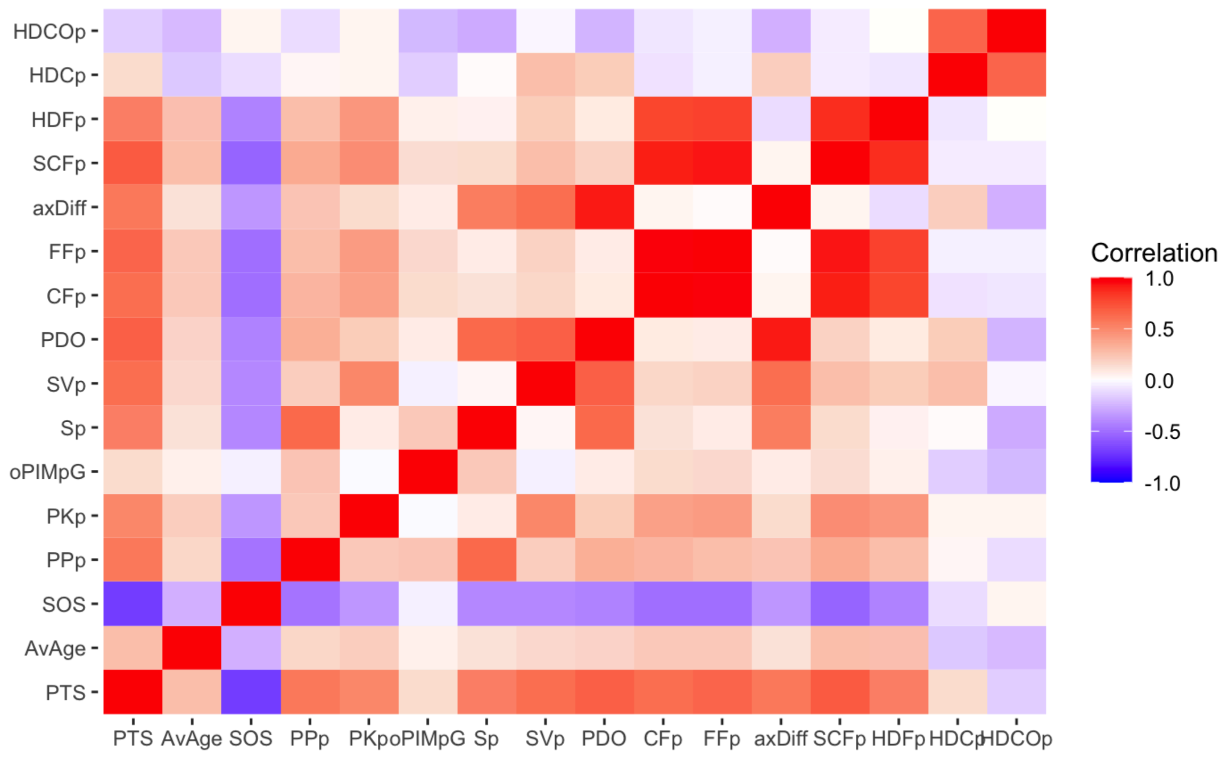


Figure 1: displays a correlation plot between various variables and points. The plot indicates that variables such as SCFp (percentage of scoring chances in the team's favor), PDO (shooting% + sv%), FFp (fenwick for percentage), CFp (corsi for percentage), and SVp (sv%) have the strongest correlations with points. These findings suggest that teams with higher values in these areas tend to accumulate more points in an 82-game season. Furthermore, SOS (strength of schedule) shows a negative correlation, implying that teams facing tougher opponents might accumulate fewer points over a season.

Modeling goal and Methods

The goal of this project is to develop a predictive model that accurately forecasts the total points that NHL teams will accumulate over a full 82-game season based on a variety of team performance metrics. The objective is to identify and quantify the relationship between key statistics (such as shooting percentage, save percentage, special teams performance, and advanced metrics like Corsi and Fenwick) and the number of points a team earns in a season.

Model development:

1. Identify Key Predictors of Team Points: Using correlation analysis, I identified the most important performance metrics that are strongly correlated with a team's success in terms of total points accumulated during the regular season. These metrics include both basic and advanced statistics.
2. Build a Predictive Model: The data was split into training and testing sets using an 80-20 split, with 80% of the data used to train the models and 20% reserved for testing. This partition was done randomly to ensure a representative test set. Using a structured approach to statistical learning, tree-based models such as random forests was employed to capture non-linear relationships and feature interactions. Furthermore, ensemble methods, including bagging and boosting, were implemented for predictive accuracy. The most significant features identified during exploratory analysis were utilized, with the ultimate goal of minimizing error and maximizing the proportion of variance explained by the model.
3. Model training and Hyperparameter Tuning: To optimize the Random Forest model, the tuneRF function was used to perform hyperparameter tuning, specifically targeting the mtry parameter, which controls the number of features considered at each split. The tuning process involved testing different values of mtry to minimize the Out-Of-Bag (OOB) error, with the best result found at mtry = 7, which was selected for the final model. The model was trained using 300 trees (ntree = 300) to reduce overfitting, considering the small size of the dataset (158 observations). For the Bagging model, a Random Forest was used with all available predictors and the target variable PTS (points). 200 trees (ntree = 200) were used, and mtry = ncol(train_data) - 1, allowing all

features to be considered at each split. The Boosting model was implemented using the Gradient Boosting Machine (GBM) algorithm. The GBM model was trained with 300 trees (n.trees = 300), with key hyperparameters such as interaction depth (set to 5) and shrinkage (set to 0.01) optimized to balance model complexity and performance. The model's performance was evaluated using 5-fold cross-validation to prevent overfitting and ensure reliable results. The optimal number of trees was selected using the gbm.perf() function, which identifies the best tree count based on cross-validation results.

4. Evaluate Model Performance: The performance of the predictive model was assessed using several key metrics, including R-squared (R^2) and Root Mean Squared Error (RMSE). R-squared measures the proportion of the variance in the target variable (PTS) that is explained by the model, with values closer to 1 indicating a better fit. RMSE, on the other hand, quantifies the average magnitude of the prediction errors, providing a measure of how far off the model's predictions are from the actual values. Lower RMSE values indicate more accurate predictions. Together, these metrics help assess how well the model generalizes to new data. Additionally, residual plots were used to visually inspect the differences between predicted and actual values, helping to identify any patterns or systematic errors that could suggest model weaknesses.
5. Cross-Validation: To enhance the model's reliability and reduce overfitting, 10-fold cross-validation was applied to evaluate the performance of three models: Random Forest, Gradient Boosting Machine (GBM), and Bagging. For the Random Forest model, cross-validation was implemented using the train() function from the caret package, with the number of trees set to 300 and mtry tuned to 7. The 10-fold cross-validation process involved splitting the data into 10 subsets, training the model on 9 of them, and

evaluating it on the remaining fold. The same cross-validation approach was used for the GBM model, where the hyperparameters such as the number of trees, tree depth, learning rate, and minimum observations in a leaf node were tuned over a grid search. For Bagging, which used Random Forest as the base model, cross-validation ensured that the model's performance was consistent and robust across different data splits. The final models were evaluated using performance metrics such as RMSE and R^2, providing a more accurate and stable assessment of their predictive capabilities by averaging the results across all folds. This process ensured that the models were not overfitting to a particular subset of the data and provided a more reliable estimate of their performance.

6. Model Comparison: The model comparison was performed by evaluating the predictions from three different algorithms—Random Forest (RF), Gradient Boosting Machine (GBM), and Bagging—along with their respective cross-validation (CV) counterparts. A data frame was created to display the actual points (PTS) from the test data alongside the predicted values from each model.

Results

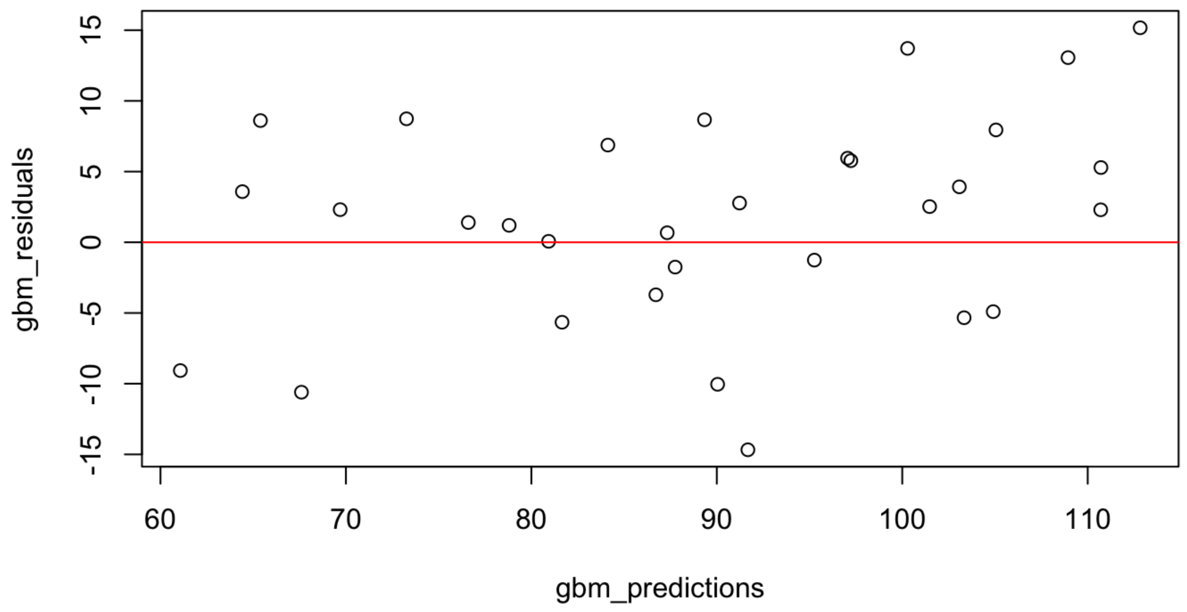
Table 2: Performance Metrics for the Tuned Models

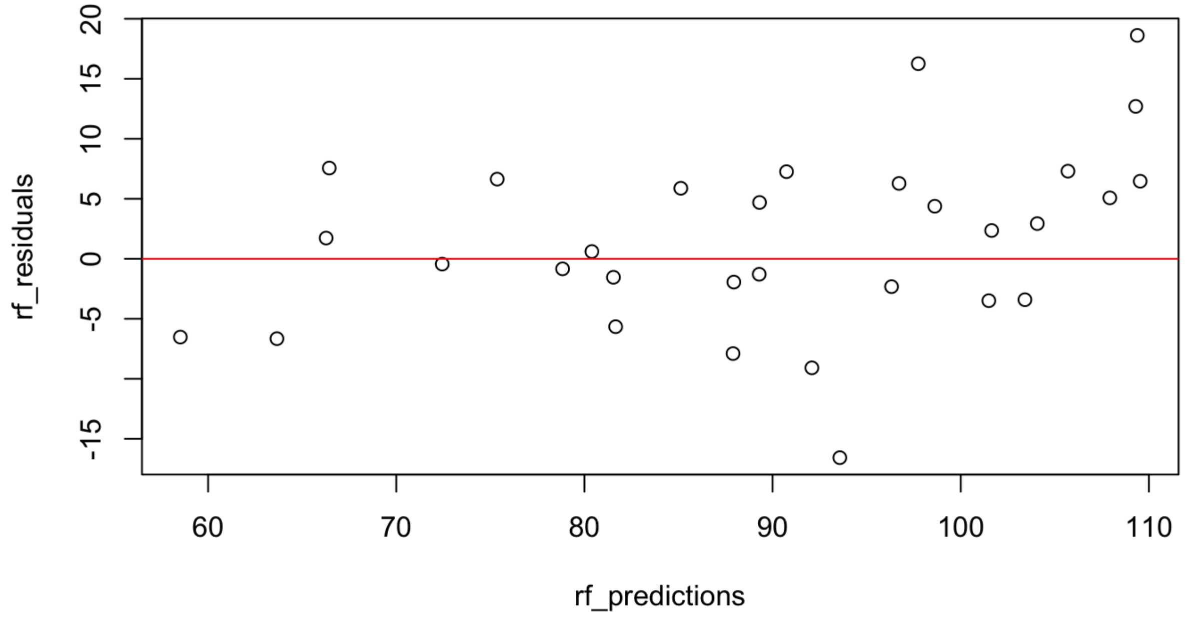
Model	RMSE	R-Squared
Random Forest	7.515	0.857
Bagging	7.973	0.833
Boosting	7.547	0.852

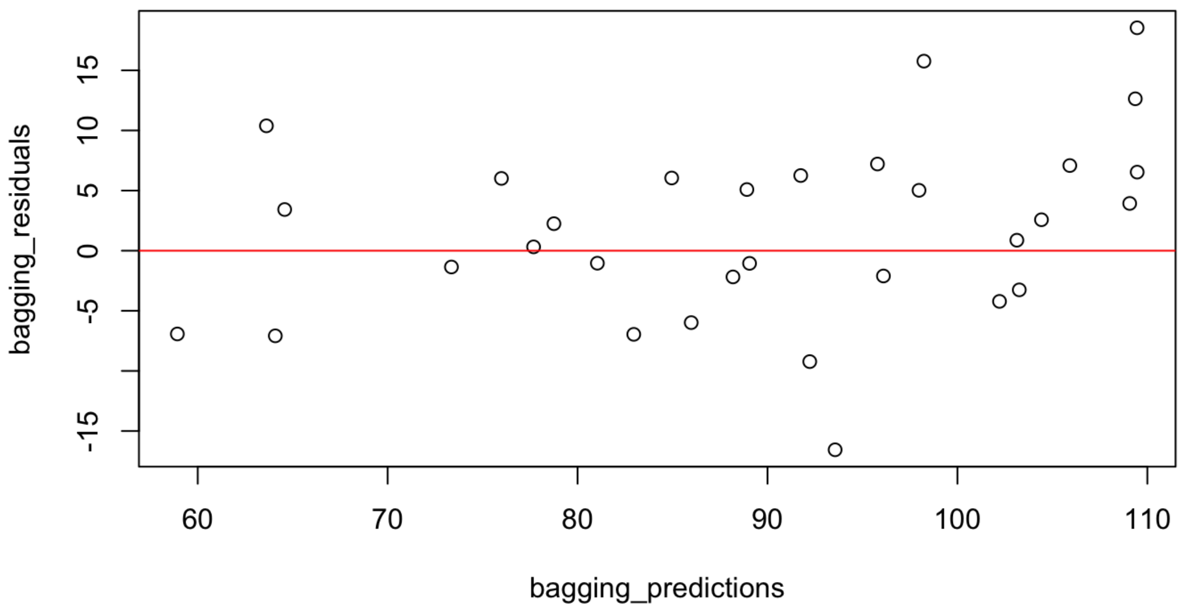
The performance of the three tuned models—Random Forest, Bagging, and Boosting—was assessed using Root Mean Squared Error (RMSE) and R-squared (R^2) values, as shown in Table 2. The Random Forest model exhibited the best performance with an RMSE of 7.515 and an R^2

of 0.857, indicating high predictive accuracy and the ability to explain 85.7% of the variance in the target variable. The Bagging model had an RMSE of 7.973 and an R^2 of 0.833, showing slightly lower predictive accuracy but still accounting for 83.3% of the variance. The Boosting model yielded an RMSE of 7.547 and an R^2 of 0.852, performing similarly to Random Forest. While all three models demonstrated strong predictive capabilities, Random Forest slightly outperformed the others in terms of both RMSE and R^2. These results suggest that each model is effective, with Random Forest offering the best balance between accuracy and explanatory power.

Figure 2: Residual Plots for Random Forest, Bagging, and Boosting


gbm_residuals
gbm_predictions
-15
-10
-5
0
5
10
15
60
70
80
90
100
110


rf_residuals
rf_predictions
-15
-5
0
5
10
15
20
60
70
80
90
100
110



The residual plots for the three models—Random Forest (RF), Gradient Boosting Machine (GBM), and Bagging—reveal some key insights into their performance. For Random Forest, the residuals show a noticeable pattern, with larger errors occurring as the predicted values increase, particularly beyond 100 points. This suggests that the model may not be fully capturing the complexity of higher predicted values, leading to increasing residuals. In contrast, the Gradient Boosting Machine (GBM) residual plot displays a more scattered distribution of residuals, though there is still some spread both above and below the zero line. This indicates that the GBM model is slightly better at handling varying prediction ranges than Random Forest, but there is still room for improvement. The Bagging model shows a pattern similar to that of Random Forest, where residuals increase with higher predicted values, though this trend is less pronounced. Overall, the spread of residuals increases with predicted values, pointing to heteroscedasticity. However, the residuals' mean (indicated by the red line at zero) suggests no significant bias in any of the models. Despite this, all models appear to have difficulty in capturing the variability at higher predicted values, suggesting they may underestimate the complexity of predictions at the upper end of the scale.

Table 3: Cross-Validation Results

Model	RMSE	R-Squared
Random Forest	7.603	0.855
Bagging	7.845	0.840
Boosting	7.325	0.857

The performance of the models was further assessed using 10-fold cross-validation to evaluate their robustness and generalizability. The Random Forest model achieved an RMSE of 7.603 and an R-squared of 0.855, indicating strong predictive accuracy and a good fit to the data. The Bagging model, while still performing well, produced a slightly higher RMSE of 7.845 and a lower R-squared of 0.840, suggesting that it was marginally less effective than Random Forest in capturing the variance in the target variable. The Gradient Boosting (Boosting) model outperformed both Random Forest and Bagging with the lowest RMSE of 7.325 and the highest R-squared of 0.857. This indicates that the Boosting model provided the best balance of predictive accuracy and explained variance in the data, making it the most reliable model when evaluated via cross-validation. Overall, all models demonstrated strong performance, with Boosting slightly outperforming the others in terms of both RMSE and R-squared.

Table 4: Predicting the Target Variable (Points) on Test Data

Actual	RF	GBM	Bagging	RF_CV	GBM_CV	Bagging_CV

114	98.02	99.18	96.72	98.21	106.82	98.11
113	108.1	110.77	108.79	107.97	114.25	108.07
107	104.42	104.32	105.38	104.02	102.36	104.96
91	84.63	84.61	85.39	84.68	81.85	83.39
88	89.29	87.35	87.78	89.89	85.78	89.32
78	78.53	74.52	76.79	78.69	75.77	80
77	93.51	91.67	95.21	93.61	88.62	94.12
52	58.49	61.76	59.92	58.89	58.38	58.59
103	97.1	97.86	95.78	96.77	98.34	96.37
98	100.45	103.24	101.91	101.43	101.06	102.17
86	88.2	88.44	87.71	88.69	89.45	88.42
81	79.73	80.69	79.39	80.78	75.24	78.96
122	109.06	108.43	109.7	108.63	109.05	109.18
116	109.85	111.18	108.19	110.06	115.45	108.64
113	105.49	104.25	105.75	105.1	106.99	105.74
94	97.03	95.06	96.63	95.95	98.64	96.46
76	82.35	82.09	82.02	81.73	85.25	83.26
74	64.53	65.28	63.61	64.85	59.72	64.55
68	64.82	64.42	64.74	64.86	58.05	65.62
57	64.32	67.78	64.39	63.93	62.39	64.49
128	110.1	112.78	108.75	109.77	116.38	109.45
104	101.09	100.82	102.75	101.4	101.59	103.11
103	98.45	96.94	98.09	97.67	96.5	96.53
100	103.24	105.5	103.5	103.03	102.2	103
82	75.4	72.48	75.06	75.25	68.66	76.15
80	81.39	78.23	80.14	81.01	80.98	80.14
72	72.07	69.96	73.45	72.27	67.9	73.06
98	91.5	89.31	91.75	90.24	85.56	91.99
94	89.87	91.55	88.55	90.05	89.88	86.69
83	91.16	85.83	91.75	91.88	85.77	93.05
80	88	89.32	88.1	87.11	86.09	87.26

The table displays the predictions for the target variable "PTS" from three models—Random Forest (RF), Gradient Boosting Machine (GBM), and Bagging—along with their respective cross-validation (CV) predictions. It shows the actual values versus the predicted values for each model. The predictions from the models and their cross-validation counterparts are compared, illustrating how well each model approximates the actual data.

Figure 3: Predicted vs Actual Values (Tuned Models)

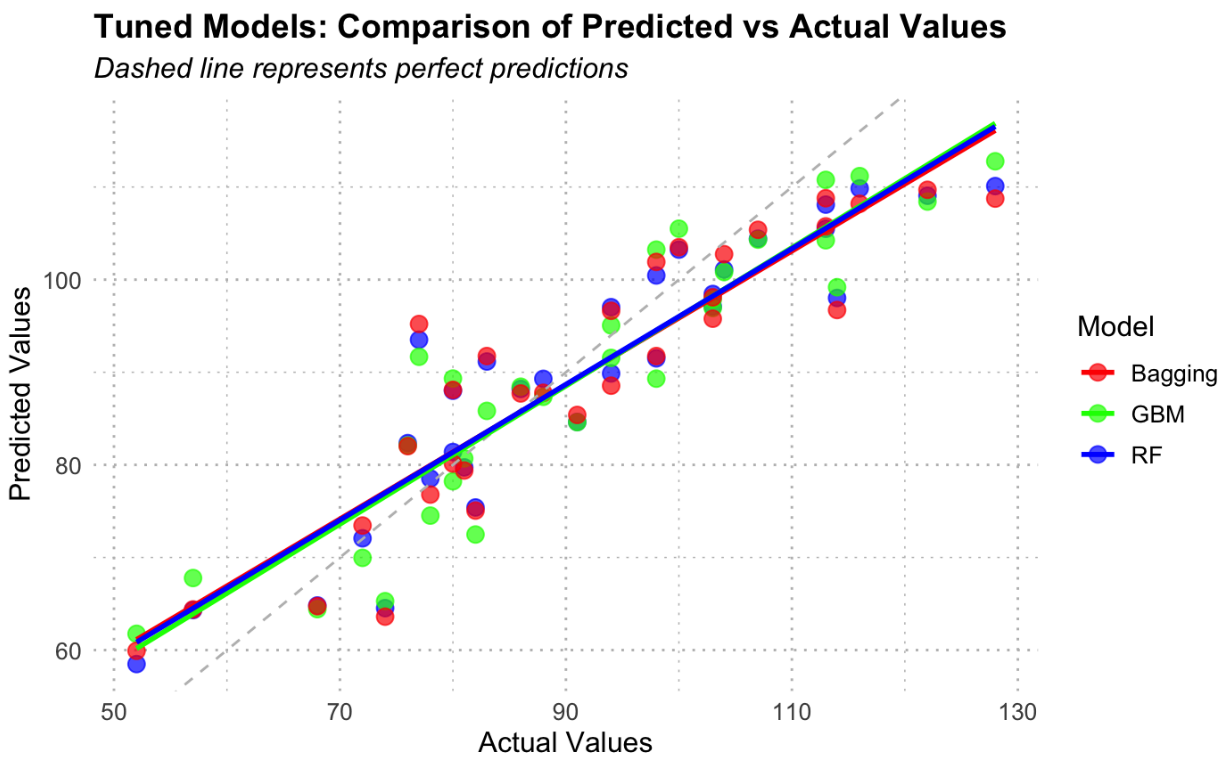


Figure 3 visualizes the results from table 4 and compares predicted versus actual values for three the tune models—Bagging, Gradient Boosting Machine (GBM), and Random Forest (RF). The dashed gray diagonal line represents the ideal scenario where predicted values perfectly match actual values. All three models show a relatively tight clustering of points around the dashed line, indicating strong predictive performance. Furthermore, there doesn't seem to be significant visual differences in prediction accuracy among the models, as the points for Bagging, GBM, and RF overlap considerably.

Figure 4: Predicted vs Actual Values (Cross-Validated Models)

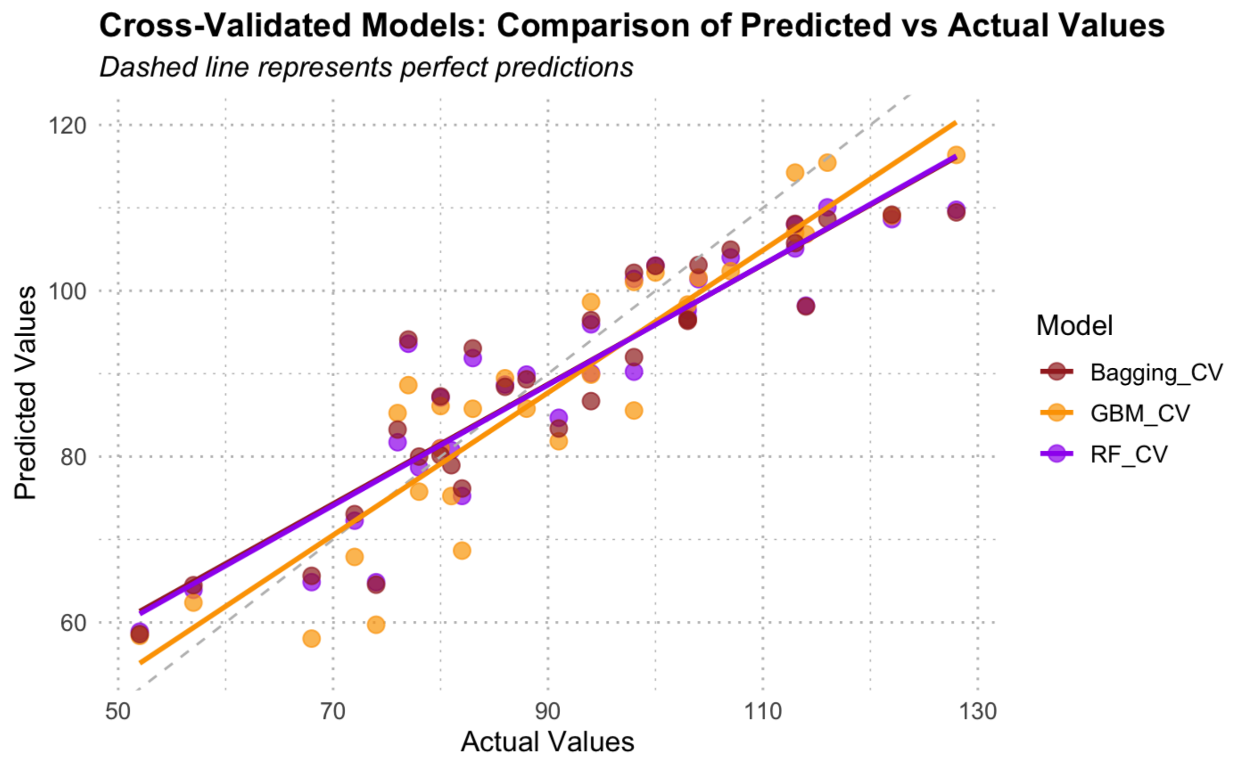


Figure 4 visualizes the results from Table 4, comparing predicted versus actual values for three cross-validated models: Bagging_CV, Gradient Boosting Machine (GBM_CV), and Random Forest (RF_CV). Similar to Figure 3, it demonstrates the relationship between predictions and actual values but emphasizes the improved performance of the GBM_CV model. The orange line, representing GBM_CV, has the slope closest to the dashed line, indicating its higher predictive accuracy compared to the other models.

Conclusion

This study aimed to develop a model to predict NHL teams' total points over an 82-game season using performance metrics. Tree-based models—Random Forest, Bagging, and Gradient

Boosting Machine (GBM)—demonstrated strong predictive capabilities. The Random Forest model achieved the best performance (out of the tuning models) on test data with an RMSE of 7.515 and R^2 of 0.857, explaining 85.7% of the variance. However, cross-validation results showed GBM_CV slightly outperformed Random Forest with the lowest RMSE (7.325) and same R^2 (0.857), demonstrating robust generalizability. Despite these results, all models showed limitations. Residuals increased for higher predicted values, suggesting difficulty in modeling high-performing teams. The study's small sample size (158 observations) and omission of contextual factors such as injuries, trades, varying coaching strategies limit the models' applicability. Future work should expand the dataset across many more seasons, incorporate additional predictors, and explore advanced modeling techniques like neural networks to better capture complex patterns.

Appendix

Phebe Chen

2024-11-15

```r
# Load Libraries
library(readxl)
library(tidyjson)
```

```
##
## Attaching package: 'tidyjson'

## The following object is masked from 'package:stats':
##
##     filter
```

```r
library(corrplot)
```

```
## corrplot 0.95 loaded
```

```r
library(reshape2)
library(ggplot2)
library(caret)
```

```
## Loading required package: lattice
```

```r
library(randomForest)
```

```
## randomForest 4.7-1.2

## Type rfNews() to see new features/changes/bug fixes.

##
## Attaching package: 'randomForest'

## The following object is masked from 'package:ggplot2':
##
##     margin
```

```r
library(gbm)
```

```
## Loaded gbm 2.2.2

## This version of gbm is no longer under development. Consider transitioning to gbm3, https://github.c
```

```r
library(Metrics)
```

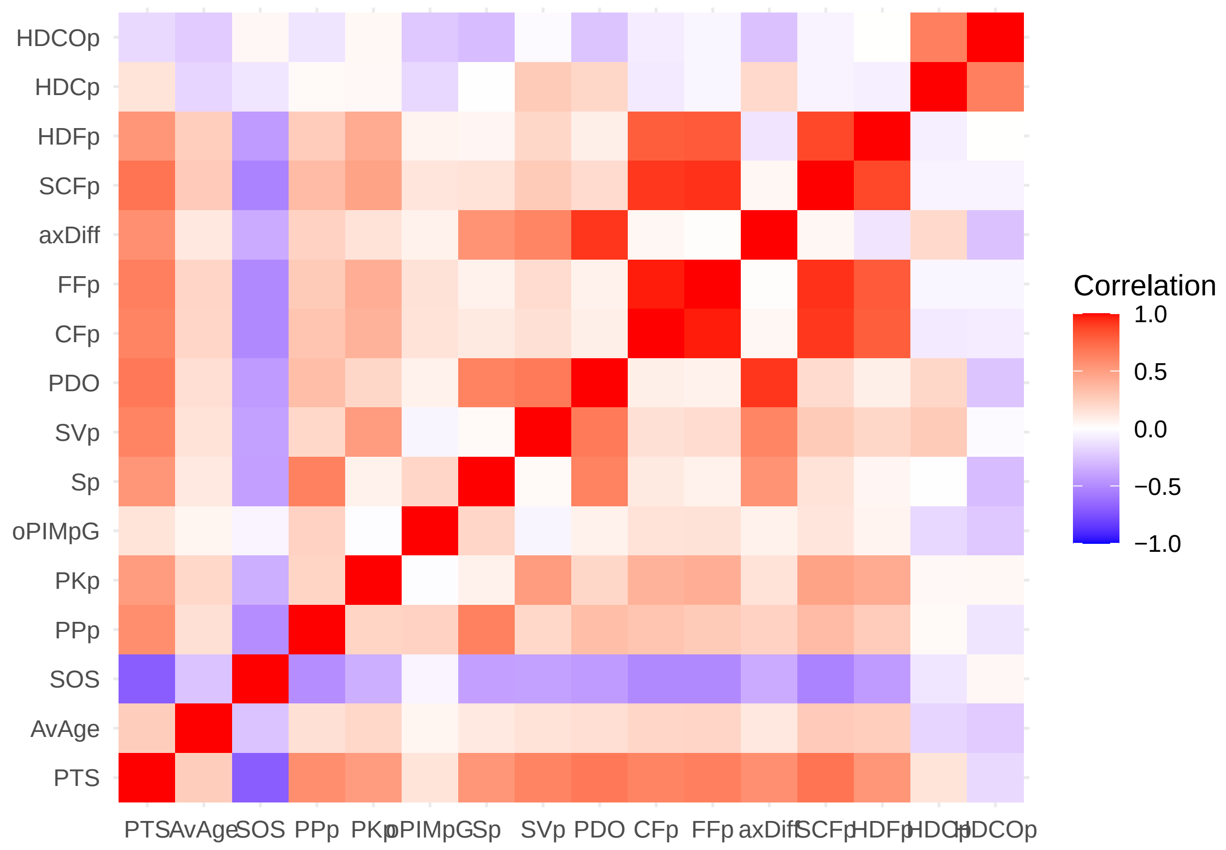
```
##
## Attaching package: 'Metrics'

## The following objects are masked from 'package:caret':
##
##     precision, recall
```

```r
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':
##
##     smiths
```

```r
# Section 1: Data Cleaning
file_path <- "seasonDataset.xlsx"
sheets <- excel_sheets(file_path)
data <- lapply(sheets, function(sheet) read_excel(file_path, sheet = sheet))
nhl_data <- bind_rows(data)

# Select only numeric columns and drop "GP"
numeric_data <- nhl_data[, sapply(nhl_data, is.numeric)]
numeric_data <- numeric_data[, setdiff(names(numeric_data), "GP")]
```

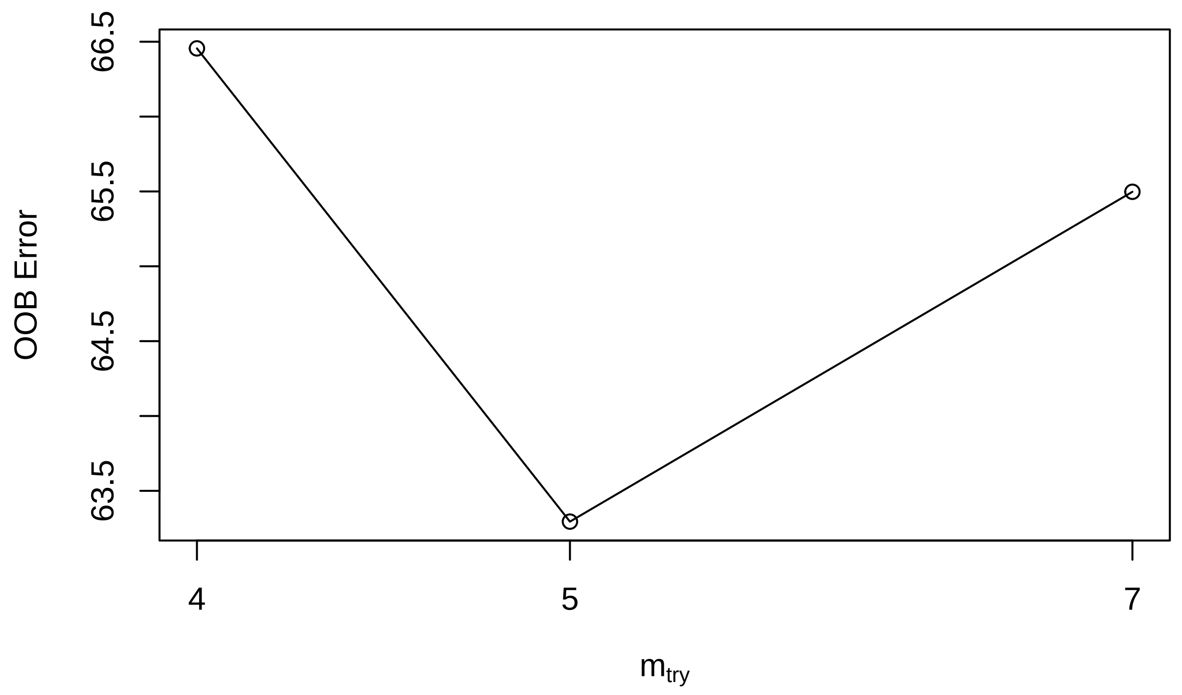
```r
# Section 2: Correlation Analysis
cor_matrix <- cor(numeric_data)
cor_pts <- cor_matrix[, "PTS"]

# Drop redundant variables & filter
columns_to_keep <- c("PTS", "AvAge", "SOS", "PPp", "PKp", "oPIMpG", "Sp", "SVp",
                     "PDO", "CFp", "FFp", "axDiff", "SCFp", "HDFp", "HDCp", "HDCOp")
nhl_data_filtered <- nhl_data[, columns_to_keep]

# Visualize Correlation Matrix
cor_matrix_filtered <- cor(nhl_data_filtered)
melted_cor <- melt(cor_matrix_filtered)
ggplot(data = melted_cor, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient2(low = "blue", high = "red", limit = c(-1, 1), name = "Correlation") +
  theme_minimal() +
  theme(axis.title.x = element_blank(), axis.title.y = element_blank())
```



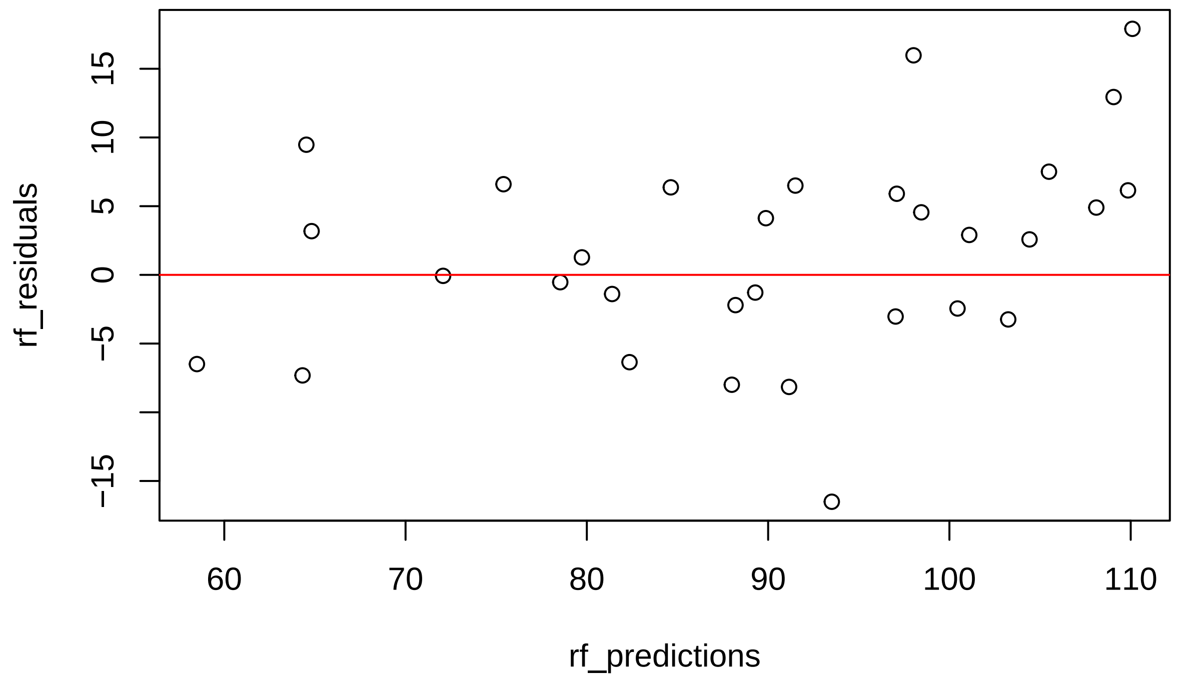
```r
# Section 3: Train-Test Split
set.seed(123)
train_index <- createDataPartition(nhl_data_filtered$PTS, p = 0.8, list = FALSE)
train_data <- nhl_data_filtered[train_index, ]
test_data <- nhl_data_filtered[-train_index, ]
```

```r
# Section 4: Random Forest Model
tune_rf <- tuneRF(train_data[, -which(names(train_data) == "PTS")], train_data$PTS, stepFactor = 1.5, i
```

```
## mtry = 5  OOB error = 63.29442
## Searching left ...
## mtry = 4     OOB error = 66.45564
## -0.04994471 0.01
## Searching right ...
## mtry = 7     OOB error = 65.49779
## -0.03481151 0.01
```



```
#300 trees to reduce overfitting (small dataset)
set.seed(123)
rf_model <- randomForest(PTS ~ ., data = train_data, mtry = 7, ntree = 300, importance = TRUE)
rf_predictions <- predict(rf_model, newdata = test_data)

rf_residuals <- test_data$PTS - rf_predictions
plot(rf_predictions, rf_residuals)
abline(h = 0, col = "red")
```



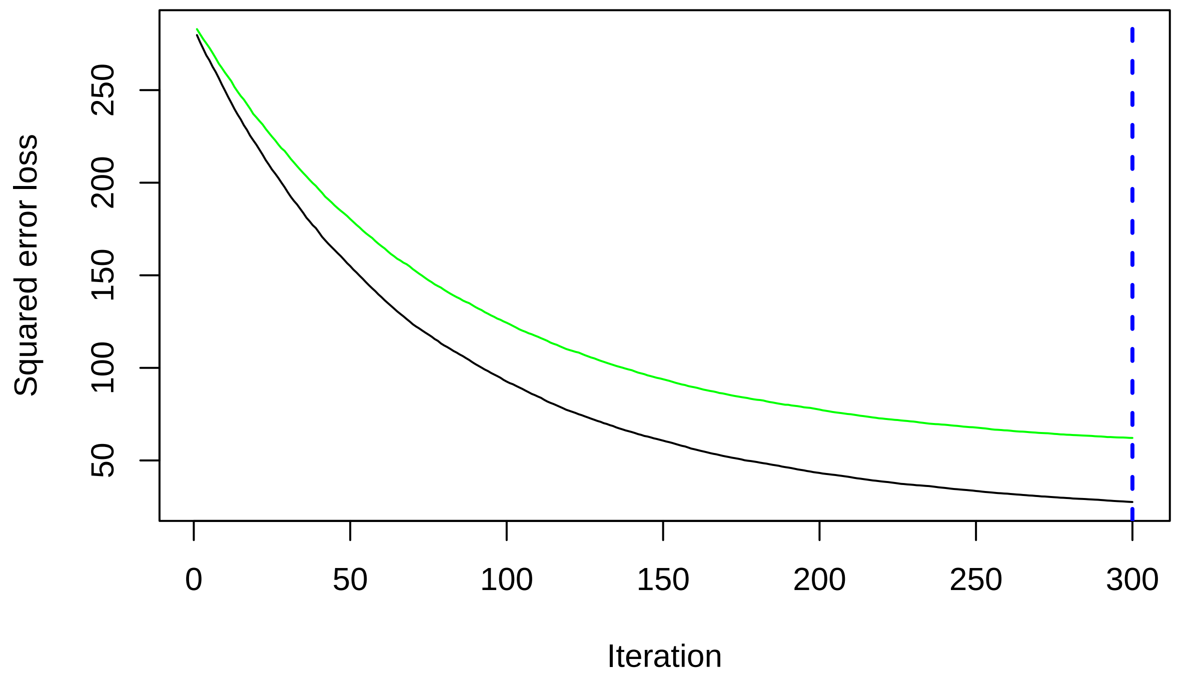
```r
# Evaluate RF Model
rmse_rf <- rmse(test_data$PTS, rf_predictions)
rsq_rf <- cor(test_data$PTS, rf_predictions)^2
cat("Random Forest - RMSE:", rmse_rf, "R-squared:", rsq_rf, "\n")
```

```
## Random Forest - RMSE: 7.514855 R-squared: 0.8568228
```

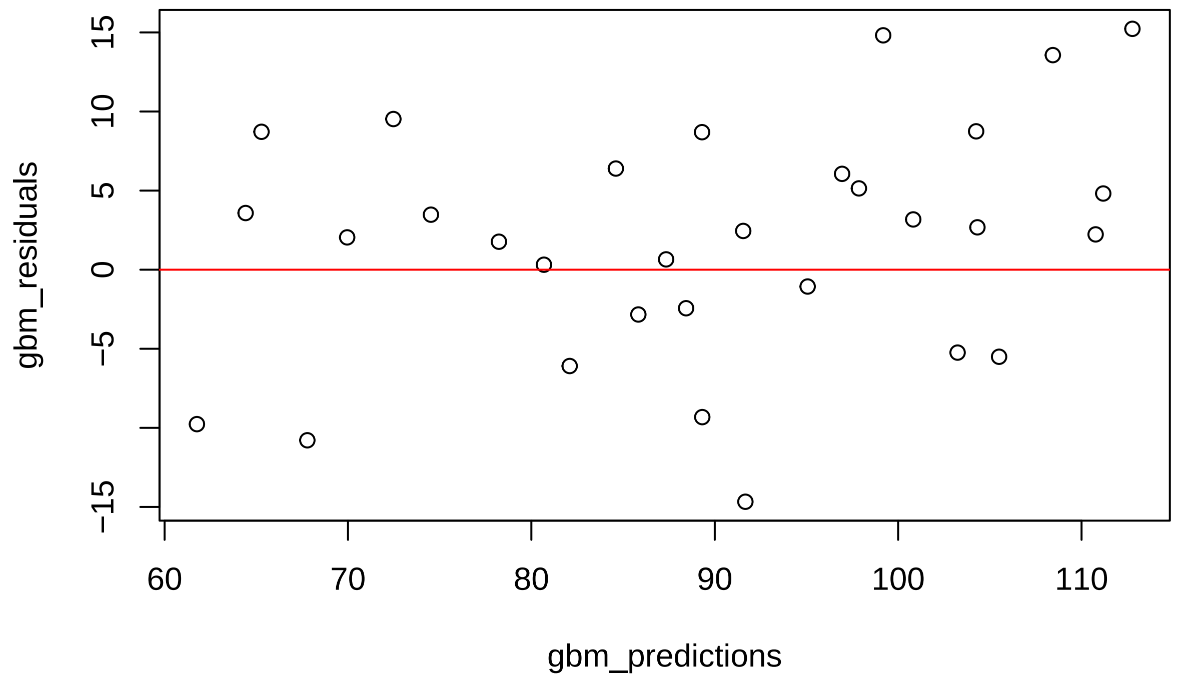
```r
# Cross-validation for Random Forest
set.seed(123)
rf_cv <- train(PTS ~ ., data = train_data,
               method = "rf",
               trControl = trainControl(method = "cv", number = 10),
               tuneGrid = expand.grid(mtry = 7),
               ntree = 300)

rf_predictions_cv <- predict(rf_cv, newdata = test_data)

# Evaluate Random Forest model
rmse_rf_cv <- rmse(test_data$PTS, rf_predictions_cv)
rsq_rf_cv <- cor(test_data$PTS, rf_predictions_cv)^2
cat("Random Forest - RMSE:", rmse_rf_cv, "R-squared:", rsq_rf_cv, "\n")
```

```
## Random Forest - RMSE: 7.602834 R-squared: 0.8545144
```

```r
# Section 5: GBM Model
set.seed(123)
gbm_model <- gbm(PTS ~ ., data = train_data, distribution = "gaussian",
                 n.trees = 300, interaction.depth = 5, shrinkage = 0.01, cv.folds = 5)
best_trees <- gbm.perf(gbm_model, method = "cv")
```



```r
gbm_predictions <- predict(gbm_model, test_data, n.trees = best_trees)

gbm_residuals <- test_data$PTS - gbm_predictions
plot(gbm_predictions, gbm_residuals)
abline(h = 0, col = "red")
```



```
# Evaluate GBM Model
rmse_gbm <- rmse(test_data$PTS, gbm_predictions)
rsq_gbm <- cor(test_data$PTS, gbm_predictions)^2
cat("GBM - RMSE:", rmse_gbm, "R-squared:", rsq_gbm, "\n")
```

```
## GBM - RMSE: 7.547409 R-squared: 0.8520441
```

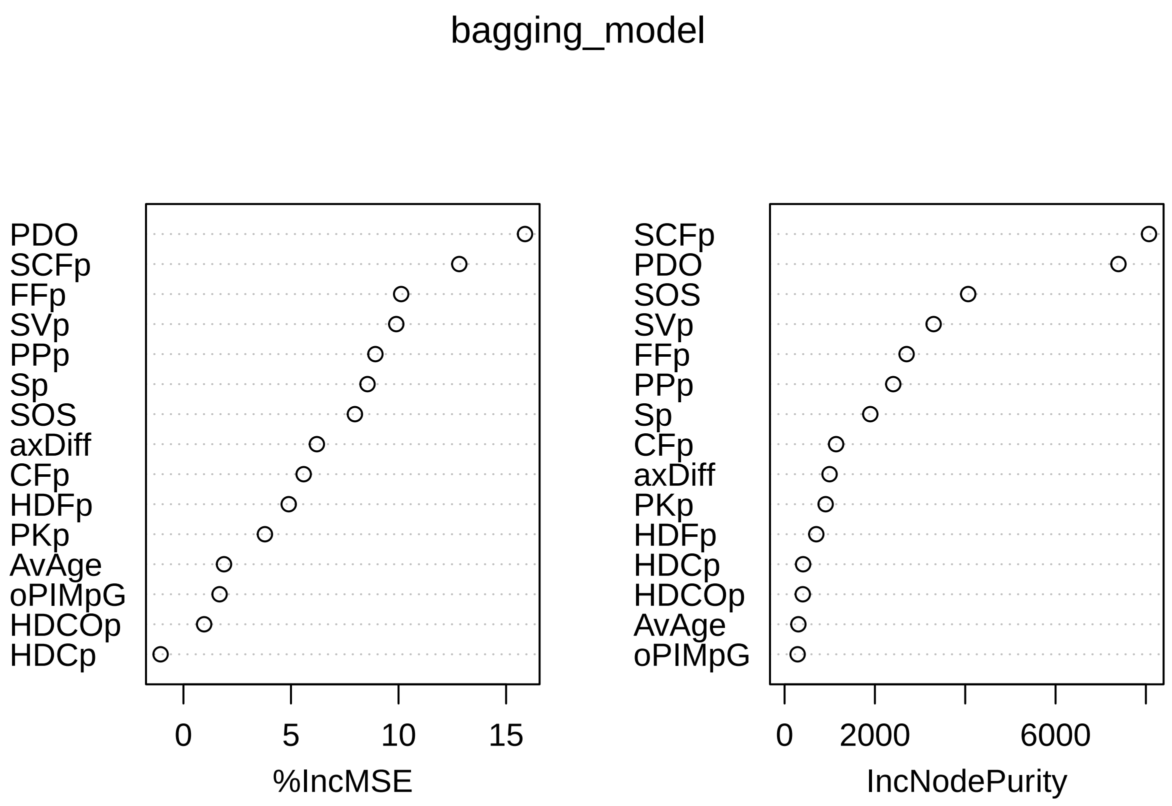
```
# Cross-validation for GBM
set.seed(123)

# Define tuning grid with correct parameters for GBM
gbm_grid <- expand.grid(
  n.trees = c(100, 200, 300),
  interaction.depth = c(3, 5, 7),
  shrinkage = c(0.01, 0.1),
  n.minobsinnode = c(10, 20)
)

# Train the GBM model using caret with 10-fold cross-validation
gbm_cv <- train(
  PTS ~ .,
  data = train_data,
  method = "gbm",
  trControl = trainControl(method = "cv", number = 10),
  tuneGrid = gbm_grid,
  verbose = FALSE
)

# Get the best model and make predictions
gbm_predictions_cv <- predict(gbm_cv, newdata = test_data)

# Evaluate the GBM model
rmse_gbm_cv <- rmse(test_data$PTS, gbm_predictions_cv)
rsq_gbm_cv <- cor(test_data$PTS, gbm_predictions_cv)^2
cat("GBM - RMSE:", rmse_gbm_cv, "R-squared:", rsq_gbm_cv, "\n")
```

```
## GBM - RMSE: 7.324712 R-squared: 0.8574602
```

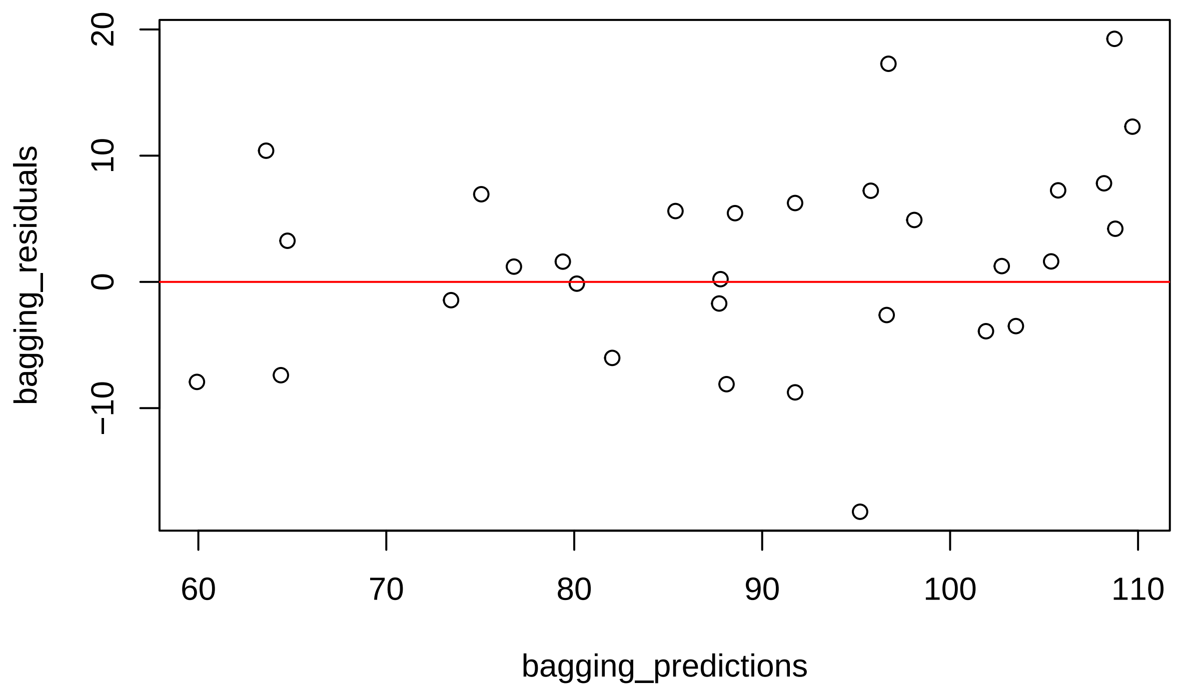
```
# Section 6: Bagging
set.seed(123)
bagging_model <- randomForest(PTS ~ ., data = train_data, mtry = ncol(train_data) - 1,
                              importance = TRUE, ntree = 200)
importance(bagging_model)
```

```
##            %IncMSE IncNodePurity
## AvAge    1.8857131      305.3603
## SOS      7.9708661     4065.2239
## PPp      8.9236187     2405.9426
## PKp      3.7873342      909.1050
## oPIMpG   1.6795372      289.1515
## Sp       8.5562809     1897.8392
## SVp      9.8943423     3300.1974
## PDO     15.8799504     7392.7431
## CFp      5.5904913     1141.0801
## FFp     10.1227632     2701.4835
## axDiff   6.2022986      996.1368
## SCFp    12.8233852     8069.3328
## HDFp     4.8950224      701.9289
## HDCp    -1.0627580      410.4966
## HDCOp    0.9619299      402.9820
```

```
varImpPlot(bagging_model)
```



```r
bagging_predictions <- predict(bagging_model, test_data)
bagging_residuals <- test_data$PTS - bagging_predictions
plot(bagging_predictions, bagging_residuals)
abline(h = 0, col = "red")
```



```r
#Evaluate Bagging Model
rmse_bagging <- rmse(test_data$PTS, bagging_predictions)
rsq_bagging <- cor(test_data$PTS, bagging_predictions)^2
cat("Bagging - RMSE:", rmse_bagging, "R-squared:", rsq_bagging, "\n")
```

```
## Bagging - RMSE: 7.972839 R-squared: 0.8327265
```

```r
# Cross-validation for Bagging (Random Forest)
set.seed(123)  # Set seed for reproducibility
bagging_cv <- train(PTS ~ ., data = train_data,
                    method = "rf",
                    trControl = trainControl(method = "cv", number = 10),
                    tuneGrid = expand.grid(mtry = ncol(train_data) - 1),
                    ntree = 200)  # Set number of trees

bagging_predictions_cv <- predict(bagging_cv, newdata = test_data)

# Evaluate Bagging model
rmse_bagging_cv <- rmse(test_data$PTS, bagging_predictions_cv)
rsq_bagging_cv <- cor(test_data$PTS, bagging_predictions_cv)^2
cat("Bagging - RMSE:", rmse_bagging_cv, "R-squared:", rsq_bagging_cv, "\n")
```

```
## Bagging - RMSE: 7.845305 R-squared: 0.8403278
```

```r
# Section 7: Model Comparison

# Create a data frame with predictions from the original models
predictions <- data.frame(
  Actual = test_data$PTS,
  RF = round(rf_predictions, 2),
  GBM = round(gbm_predictions, 2),
  Bagging = round(bagging_predictions, 2),
  RF_CV = round(rf_predictions_cv, 2),
  GBM_CV = round(gbm_predictions_cv, 2),
  Bagging_CV = round(bagging_predictions_cv, 2)
)

# Print the predictions for comparison
print(predictions)
```

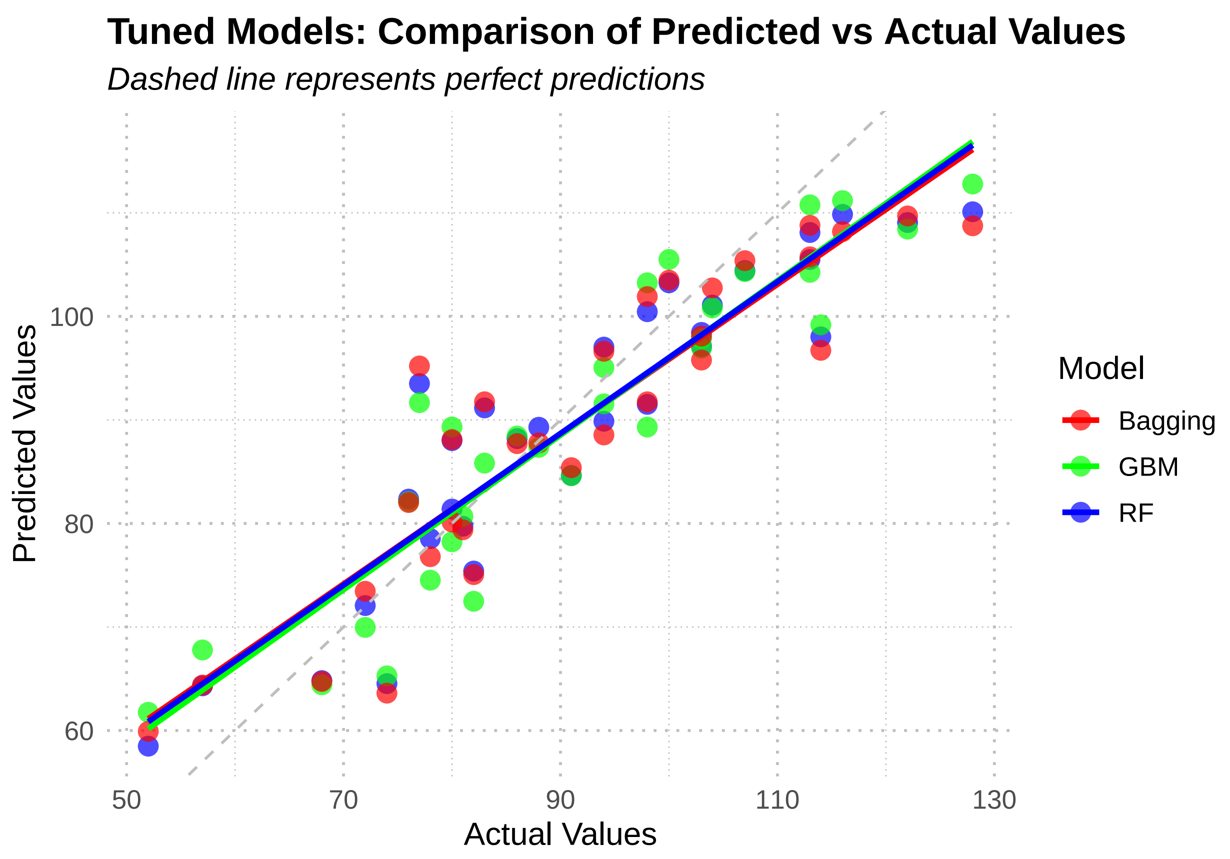
```
##    Actual     RF    GBM Bagging  RF_CV GBM_CV Bagging_CV
## 1     114  98.02  99.18   96.72  98.21 106.82      98.11
## 2     113 108.10 110.77  108.79 107.97 114.25     108.07
## 3     107 104.42 104.32  105.38 104.02 102.36     104.96
## 4      91  84.63  84.61   85.39  84.68  81.85      83.39
## 5      88  89.29  87.35   87.78  89.89  85.78      89.32
## 6      78  78.53  74.52   76.79  78.69  75.77      80.00
## 7      77  93.51  91.67   95.21  93.61  88.62      94.12
## 8      52  58.49  61.76   59.92  58.89  58.38      58.59
## 9     103  97.10  97.86   95.78  96.77  98.34      96.37
## 10     98 100.45 103.24  101.91 101.43 101.06     102.17
## 11     86  88.20  88.44   87.71  88.69  89.45      88.42
## 12     81  79.73  80.69   79.39  80.78  75.24      78.96
## 13    122 109.06 108.43  109.70 108.63 109.05     109.18
## 14    116 109.85 111.18  108.19 110.06 115.45     108.64
## 15    113 105.49 104.25  105.75 105.10 106.99     105.74
## 16     94  97.03  95.06   96.63  95.95  98.64      96.46
## 17     76  82.35  82.09   82.02  81.73  85.25      83.26
## 18     74  64.53  65.28   63.61  64.85  59.72      64.55
## 19     68  64.82  64.42   64.74  64.86  58.05      65.62
## 20     57  64.32  67.78   64.39  63.93  62.39      64.49
## 21    128 110.10 112.78  108.75 109.77 116.38     109.45
## 22    104 101.09 100.82  102.75 101.40 101.59     103.11
## 23    103  98.45  96.94   98.09  97.67  96.50      96.53
## 24    100 103.24 105.50  103.50 103.03 102.20     103.00
## 25     82  75.40  72.48   75.06  75.25  68.66      76.15
## 26     80  81.39  78.23   80.14  81.01  80.98      80.14
## 27     72  72.07  69.96   73.45  72.27  67.90      73.06
## 28     98  91.50  89.31   91.75  90.24  85.56      91.99
## 29     94  89.87  91.55   88.55  90.05  89.88      86.69
## 30     83  91.16  85.83   91.75  91.88  85.77      93.05
## 31     80  88.00  89.32   88.10  87.11  86.09      87.26
```

```r
# Save the rounded predictions to a CSV file
write.csv(predictions, "predictions.csv", row.names = FALSE)
```

```r
# Separate the predictions into two datasets: Tuned Models and Cross-Validated Models
predictions_long <- predictions %>%
  gather(key = "Model", value = "Prediction", -Actual)

predictions_tuned <- predictions_long %>%
  filter(Model %in% c("RF", "GBM", "Bagging"))

predictions_cv <- predictions_long %>%
  filter(Model %in% c("RF_CV", "GBM_CV", "Bagging_CV"))

# Create the plot for Tuned Models
ggplot(predictions_tuned, aes(x = Actual, y = Prediction, color = Model)) +
  geom_point(alpha = 0.7, size = 3) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray") +
  geom_smooth(method = "lm", se = FALSE, aes(color = Model), linetype = "solid") +
  labs(
    title = "Tuned Models: Comparison of Predicted vs Actual Values",
    subtitle = "Dashed line represents perfect predictions",
    x = "Actual Values",
    y = "Predicted Values"
  ) +
  scale_color_manual(values = c("RF" = "blue", "GBM" = "green", "Bagging" = "red")) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12, face = "italic"),
    axis.text = element_text(size = 10),
    axis.title = element_text(size = 12),
    legend.title = element_text(size = 12),
    legend.text = element_text(size = 10),
    panel.grid.major = element_line(color = "gray", size = 0.5, linetype = "dotted"),
    panel.grid.minor = element_line(color = "gray", size = 0.25, linetype = "dotted")
  )
```

```
## Warning: The `size` argument of `element_line()` is deprecated as of ggplot2 3.4.0.
## i Please use the `linewidth` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## `geom_smooth()` using formula = 'y ~ x'
```

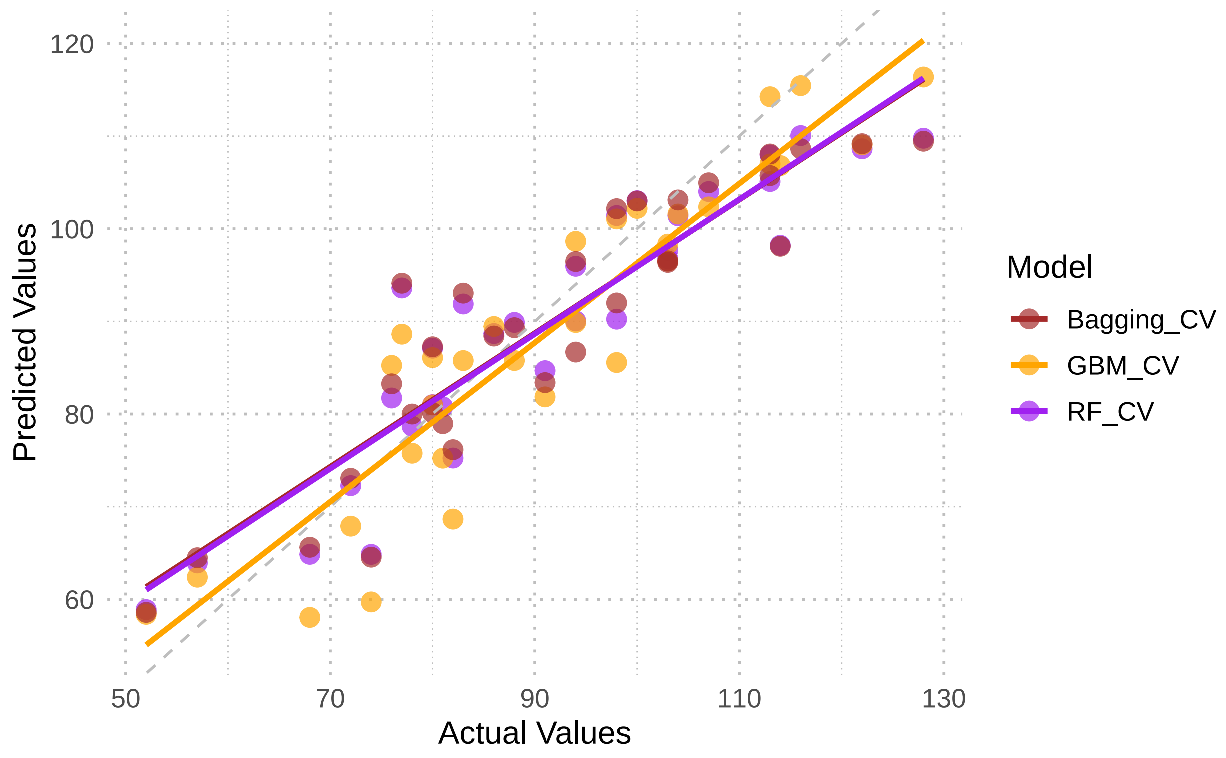


```r
# Create the plot for Cross-Validated Models
ggplot(predictions_cv, aes(x = Actual, y = Prediction, color = Model)) +
  geom_point(alpha = 0.7, size = 3) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "gray") +
  geom_smooth(method = "lm", se = FALSE, aes(color = Model), linetype = "solid") +
  labs(
    title = "Cross-Validated Models: Comparison of Predicted vs Actual Values",
    subtitle = "Dashed line represents perfect predictions",
    x = "Actual Values",
    y = "Predicted Values"
  ) +
  scale_color_manual(values = c("RF_CV" = "purple", "GBM_CV" = "orange", "Bagging_CV" = "brown")) +
  theme_minimal() +
  theme(
    plot.title = element_text(size = 14, face = "bold"),
    plot.subtitle = element_text(size = 12, face = "italic"),
    axis.text = element_text(size = 10),
    axis.title = element_text(size = 12),
    legend.title = element_text(size = 12),
    legend.text = element_text(size = 10),
    panel.grid.major = element_line(color = "gray", size = 0.5, linetype = "dotted"),
    panel.grid.minor = element_line(color = "gray", size = 0.25, linetype = "dotted")
  )
```

```
## ‘geom_smooth()‘ using formula = ’y ~ x’
```

Cross−Validated Models: Comparison of Predicted vs Actual Val

Dashed line represents perfect predictions


Predicted Values
Actual Values
Model
Bagging_CV
GBM_CV
RF_CV