Stat 184 Final

Patrick Erickson, Aiden Shindel, Jordan Brophy

Exploratory Predictive Data Analysis for Basketball Teams: Can we Create a Model to Predict if one Basketball Team will Win over Another?

Introduction

Intuition Behind the Choice of Data Set

In basketball, a number of statistics can be used in order to predict the winner of a game. For instance, a team's field goal percentage often correlates to the number of points that a team will score in any given game. Given these metrics, there has to be some combination of statistics that can ultimately help us predict if one team will win a game based on these statistics. In order to be able to perform this prediction, we must first find a data set that fits the parameters of the following:

- The data set must be large to be able to get a good testing pool.
- The data set has meaningful predictors are practically relevant to our test case.
- The data set contains a winning/losing team variable
- The data set follows **F.A.I.R.** and **C.A.R.E.** principles.

We ultimately landed on the *NCAA Regular Season Basketball Dataset* by Nate Duncan [1], due to its extremely high case to variable ratio (90902:40), the fact that the data set had many predictors that would practically contribute to the winning or losing of a team, and that it had included a winning and losing team, which is our testing parameter.

[1] [1]

Ethical Considerations in the Choice of our Data Set

In order to ensure the ethical feasibility of the data set, we also scrutinized our selection to follow the **F.A.I.R** and **C.A.R.E** principles as discussed in class.

Since we had obtained our data from the open-source data hub Kaggle, we can ensure that anyone can **find** and **access** this data set through Kaggle's use guidelines, stating for a free distribution of data sets that are published for public use to practice ML or data analysis on. The data is also highly versatile, taking a csv format. This is one of the most **interoperable** formats there are for data sets, especially for our use-case in R, where reading in a csv is already built into the base. Lastly, the data set can be **reused** for multiple types of data analysis, and will always remain relevant for its time period.

In terms of **collective benefit**, the data set contributes to an overall understanding of basketball without harming any individuals or teams, due to the fact that it is simply raw data. Secondly, as I had stated the author's name, there is clearly ownership over the data set coming from sports-reference.com as Duncan mentioned, showing proper **Authority to Control**. Since proper attributions are maintained, it also falls under the **Responsibility and Ethics** guidelines of the C.A.R.E principles.

Initial Cleaning

In order for use to perform data analysis, we must first do some cleaning to better understand our data variables. Luckily, Kaggle offers us a a short summary of each of these variables we can use to clean our data. Upon closer inspection, we have found that the dataset also includes opponent data, which is not information that we wish to keep, as we are only interested in a specific team's performance, regardless of the opponent's. Null values that may skew calculations in the latter processes of our analysis were also present within the data set. Additionally, we can also see that it is extremely hard to determine what each statistic is based on their respective abbreviations. As a result, we devised a multi-step process to clean and ready the data for further analysis and testing:

- Reject any null values that may appear in the data set, since we can not interpret them.
- Reject any negative values from the data.
- Rename every variable within the basketball data set to match that of the subscripted value beneath.
- the name of the column on the website for better readability.
- Change wins and losses to a binary 1 or 0.
- 1-hot encode site to 3 separate variables: Home, Away, or Neutral

Particular motivations for 1-hot and binary encoding arose due to the fact that a lot of solutions to our proposed problems may require categorical variables to be represented numerically. Therefore, in order to capture categorical variables that may have a significant effect on our

data we 1-hot encode them (the site variable in our case) capture the influence a particular portion of the model. Secondly, since win or loss can be represented in a binary format, we chose also mutate the game decision variable (categorical) into a numerical value as well (numerical). Lastly, you may notice that there are some variables that may not be able to be one-hot encoded, (such as team_name and date) because doing so would give our dataset hundreds of extra dimensions, making it infeasible to do any sort of calculations on.

Based on the following information, we can perform the following operations to achieve a new, cleaned dataframe, using tidyr api:

By looking at the head, we can see that this model is much easier to interpret. The following gives us a little more information that we can use for our data analysis:

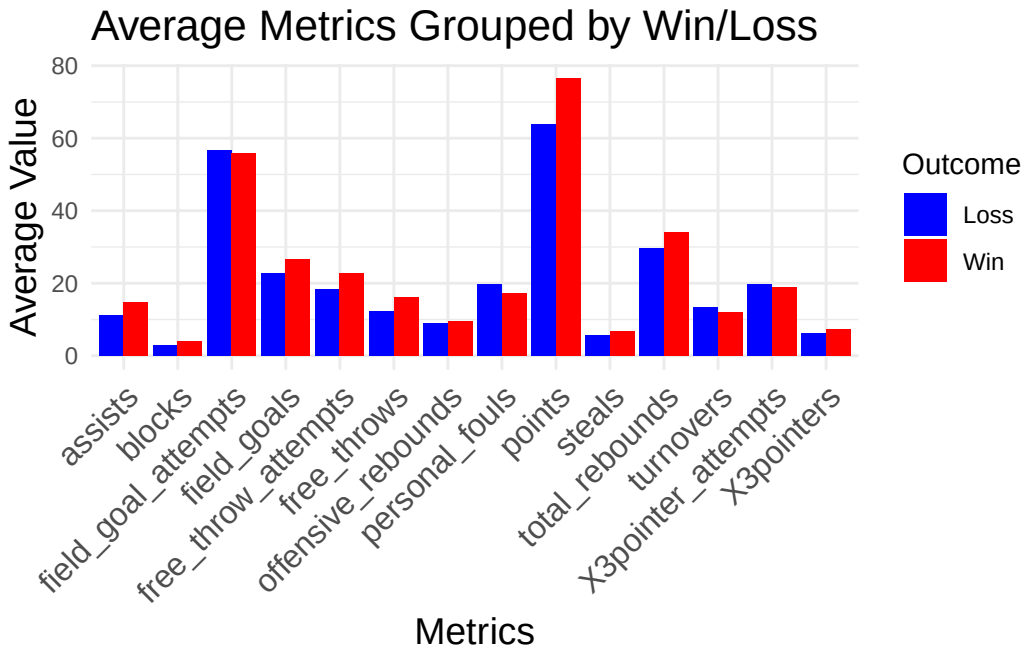
Initial Exploratory Data Analysis

Let us first look at some of the average statistics for our model to help us better understand our data:

Game Statistic	Min	25%-tile	Median	75%-tile	Max	mean
points	25.00	61.00	70.00	79.00	151.00	70.44
field_goals	6.00	21.00	24.00	28.00	56.00	24.68
field_goal_attempts	27.00	51.00	56.00	61.00	106.00	56.17
field_goal_percentage	0.12	0.39	0.44	0.49	0.78	0.44
X3pointers	0.00	5.00	6.00	9.00	26.00	6.71
X3pointer_attempts	1.00	15.00	19.00	23.00	58.00	19.35
X3pointer_percentage	0.00	0.27	0.34	0.42	1.00	0.35
free_throws	0.00	10.00	14.00	18.00	54.00	14.37
free_throw_attempts	1.00	15.00	20.00	26.00	78.00	20.59
free_throw_percentage	0.00	0.62	0.70	0.78	1.00	0.69
offensive_rebounds	0.00	6.00	9.00	12.00	37.00	9.22
total_rebounds	6.00	27.00	32.00	36.00	71.00	31.90
assists	0.00	10.00	13.00	16.00	40.00	13.10
steals	0.00	4.00	6.00	8.00	37.00	6.34
blocks	0.00	2.00	3.00	5.00	18.00	3.47
turnovers	0.00	10.00	12.00	15.00	39.00	12.68
personal_fouls	3.00	15.00	18.00	21.00	44.00	18.48

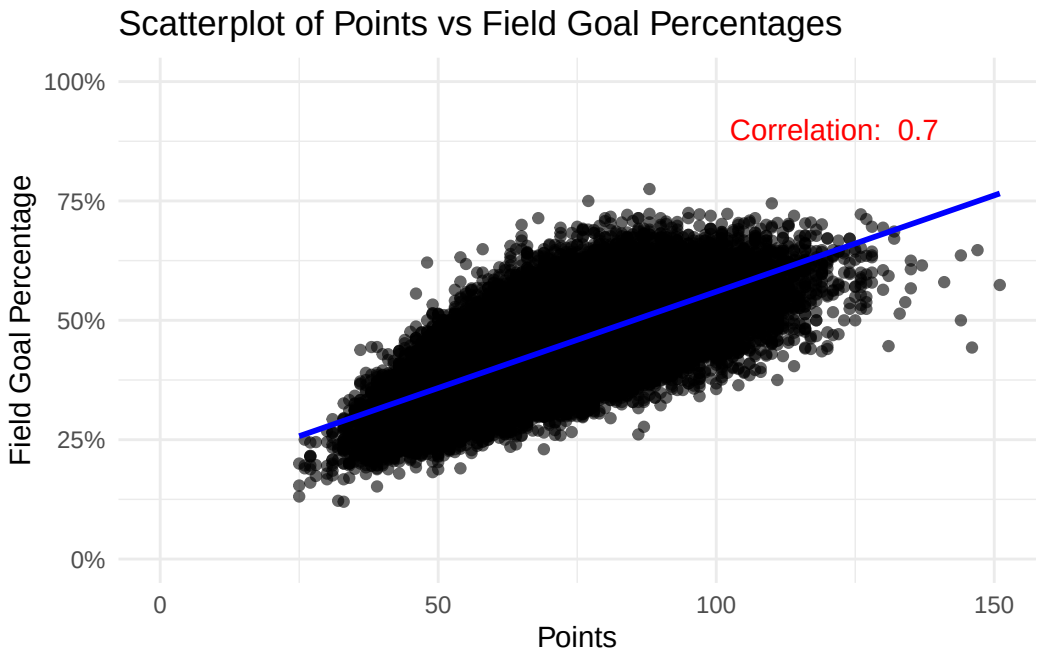
Based on the following, we can see that there is a lot of variability within the dataset. In particular, we want to explore the mean further and see if we can obtain any sort of relationships from the data.

As Kosslyn had stated in his book of *Graphic Design for the Eye and Mind*, "two properties must differ by a large enough proportion or they will not be distinguished."[2] In order to build up on the notion of prediction we must therefore distinguish the difference margins of data. In particular, we are interested on how average statistics differ based on wins and losses. Winning or losing is generally the metric of greatest interest for every basketball team every season. Therefore, our data focus will be around **wins and losses**, and the potential relationships every variable has on this metric. We can therefore view a simple bar graph to see if there is an imbalance between wins and losses, and start to begin to explore trends in our data.

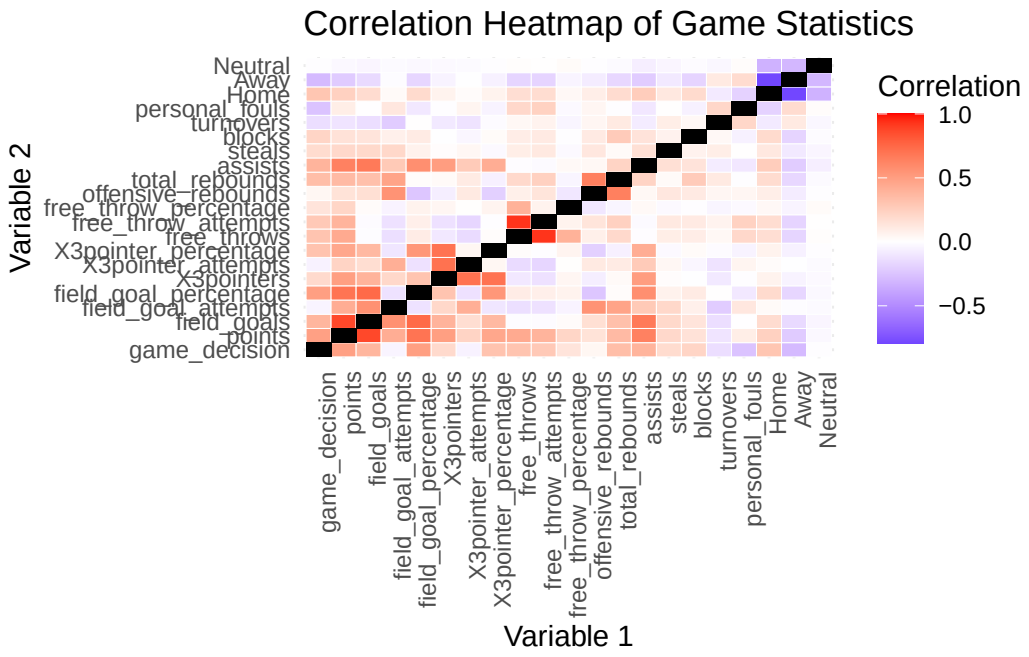


We can see here that there are clear differences in the averages of the teams that won and the averages of the teams that lost. Based on this difference, we can **infer** there might be some effect a higher or lower value of one of these variables can have, on the win rate of a game, respectively. We can see however, that based on the names, there might also be correlations between the variables. Let us model a scatterplot for the correlation between points and field

[2] [4]



We can see that some of this variable therefore have strong pairwise correlations between each other. This indicates that changing one of the variables may have some sort of influence on changing the other. In order to better visualize all of these pairwise relationships, we can use a heatmap map of correlations to get a better idea of of these pairwise correlations:



The model depicts some weak correlation between all pairwise variables, with the diagonal signifying a variable's pairwise relationship with itself, which will obviously be 1. Since the heatmap has at least some weak correlation in all squares, it can therefore suggested that there are many relationships between multiple values of the dataset, since if multiple variables are pairwise dependent with multiple variables, then there may also be some colinearity in greater dimensions. However, this becomes much harder to visualize as we go up in dimensions. According to Tufte , "reasoning about evidence should not be stuck in 2 dimensions, for the world we seek to understand is profoundly multivariate."[3] It is therefore crucial that we analyze many aspects of our statistical parameters to find objects of interest. In order to continue interpreting the data, we need a new method to understand the data.

Given that these variables all have some sort of correlation with each other, curiosity drove us to the possibility of explaining these variables and their respective relationships with some sort of regression model. In explaining multiple variables with one regression function, while visualizations can remain difficult, the influences can be understood. Conveniently, there exists a method to be able to answer our original question of being able to predict a the binary outcome of a game's decision, effectively solving both propositions. After consulting Hyungsuk Tak, a professor of the Eberly College Statistics Department [4], we deduced that this can indeed be approached as a binary classification problem, where we can explore the data via **logistic regression** model as a means of predicting a win or loss, which will help us both explain relationships and predict if a team will win or lose based on their statistics.

Introduction to Logistical Regression

Logistical regression is a form of regression analysis in which, unlike normal regression where $\widehat{\beta}_{MLE}$ (the regression coefficient) be used to explain a direct linear relationship between two variables (conventionally represented as $x_i^T\widehat{\beta} + \epsilon_i$ in multiple linear regression), we have such that a logistic regression model is a regression model that can be represented as the log of the odds, or functionally,

$$log(\frac{\theta_i}{1-\theta_i}) = x_i^T\widehat{\beta} = logit(\theta_i)$$

We can obtain the odds of any one team winning or losing by exponentiating $logit(\theta_i)$, where $\widehat{\beta}$ is fixed and x_i is any valid input of a vector of statistics. This will give us some probability of a team winning based on all the statistical metrics we initially input based on the functional form $\theta_i = logit^{-1}(x_i^T\widehat{\beta})$. We then have some sort of prediction threshold, where we "predict" whether a team will win or lose based on said threshold. This is therefore a possible solution to our proposed question, where we wish to determine the binary probability of a win or a loss (this is our respective θ_i in terms of the logistic regression model) based on some combination of the factors in our data set. We can therefore use our data set to "train" a logistic regression

[3] [7]
[4] [5]

model, where every individual case of some combination of specified statistics can lead to a more accurate prediction. For the sake of simplicity, we will be using Base R's glm() function to perform our logistic regression.

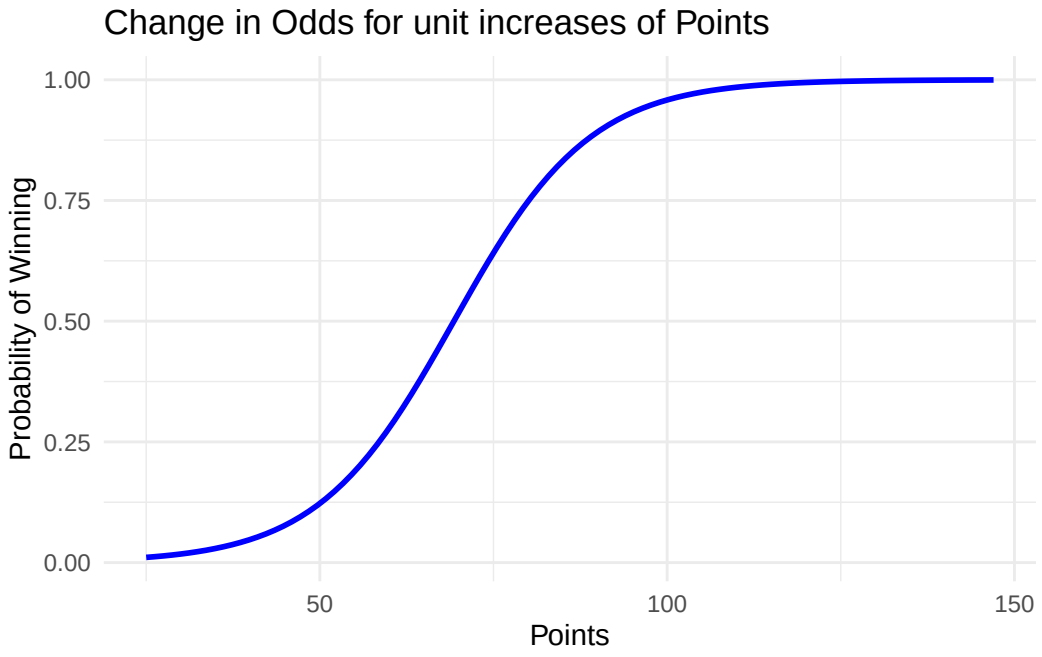
Readying the Data for Exploratory Logistic Regression Analysis

We had shown earlier some of the choices we had made to manipulate our dataset to ensure that logistic regression can be run with Base R's glm() function. We will therefore create a table that is able to be fitted to a logistic regression model by performing the following transformations:

Here we make the choice to use single train/test sets to make reproducibility easier, as well as reduce our computational overhead. The head of the model is shown as the final result for our logistic regression model fitting.

EDA for Logistic Regression (Simple Logistic Regression)

In order to help us interpret what regression will do, we need will first fit a single variable into a logistic regression model. We chose points to look at to see if it can help us increase the odds to correctly predict the number of odds in our data set. The probability graph is then plotted for further understanding of the concept.



This graph essentially shows what happens for every unit increase in points. Notice how the relationship predicts that basketball teams that score close to 0 points have an EXTREMELY low probability of winning, while teams that score 100 points are predicted to have over a 90% chance of winning.

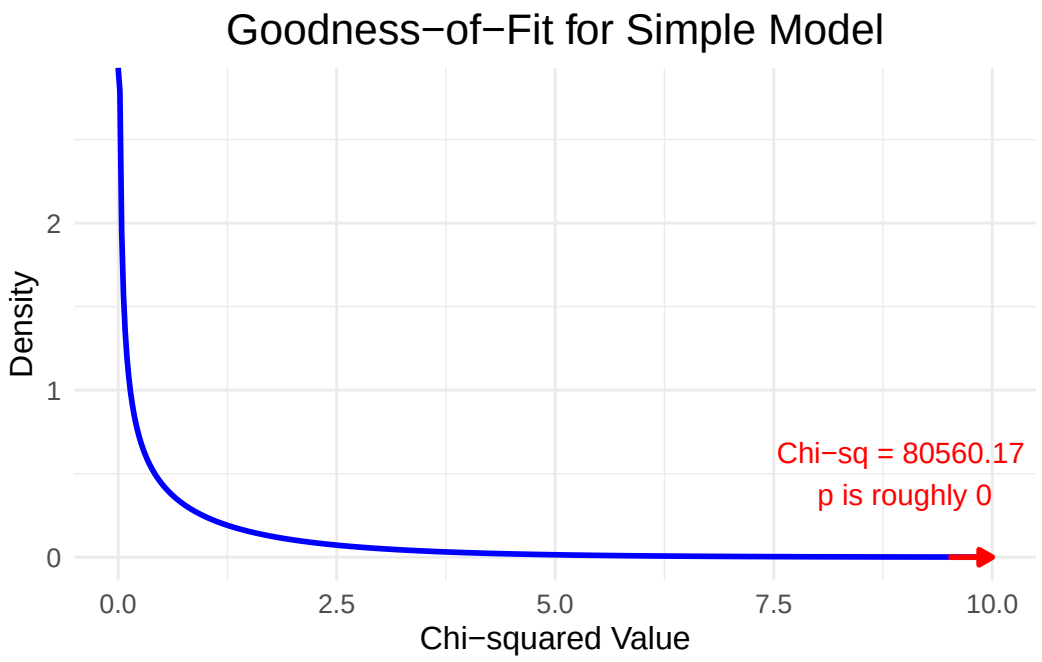
Hypothesis Test for Model Significance

Despite the fact that our logistic regression model can produce some prediction of our binary outcome, it may turn out that the predictor used to construct the model might be completely garbage. In other words, it could either have too weak of a relationship with the binary outcome (game decision), meaning the overall power of this predictor might not be statistically significant. Luckily, Base R's logistic regression models allow us to extract the Null and Residual Deviances (D_N and D_R for our use case), which can be used as a metric to perform a goodness of fit test. The differences in their degrees of freedom df follows a χ^2_{df} distribution, or more intuitively the number of predictors within our model (1). We can therefore construct the following hypothesis test to test for model significance:

$$H_0 : \beta_1 = 0$$

$$H_1 : \beta_1 \neq 0$$

Based on the following distribution, we will plot both the p-value and the distribution to either reject or fail to reject our null hypothesis:

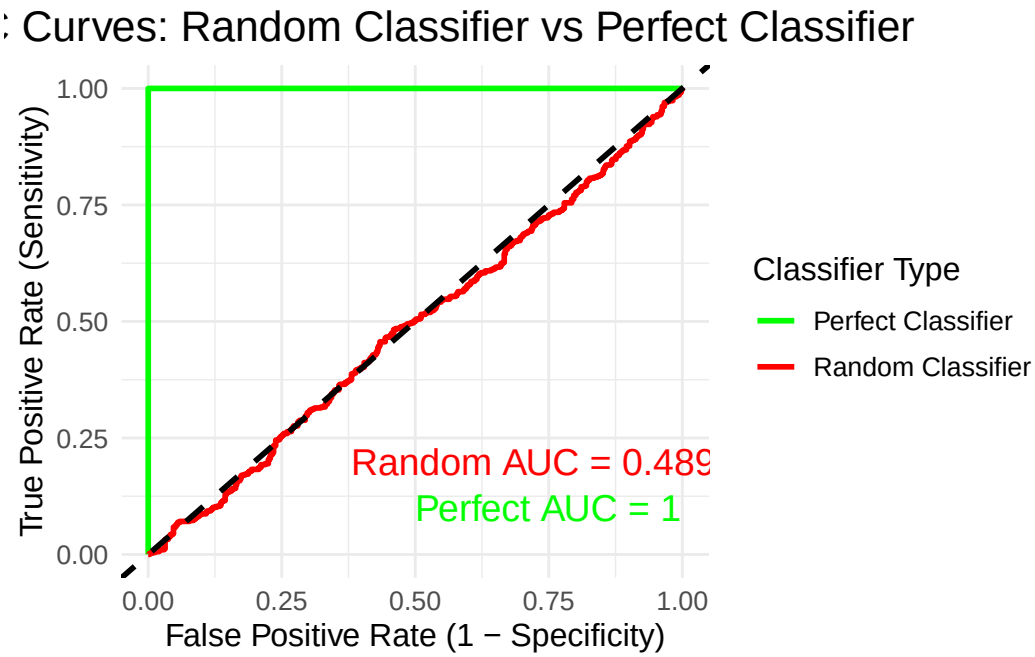


We can see that the p-value is so significant that it can't be feasibly plotted to the y-axis. Based on the following derived information (p-value essentially being 0), we can therefore reject the null hypothesis, and state our model is at least significant.

Can we Improve our Model?

Testing the Efficacy of our Model with AUC of ROC

We now want to construct a metric to be able to compare our logistical regression model to others and compare their efficacies to see if we can predict our binary outcome better. In practice, we can use our testing set to be able to plot the rate of identifying True positives (the true value in the testing set is a win, and our model predicts a win) over the false positive (the true value is positive, but our model predicts negative). The Area Under the Curve (AUC) of the ROC ultimately gives us a rating on model efficacy, and is widely used in practice.

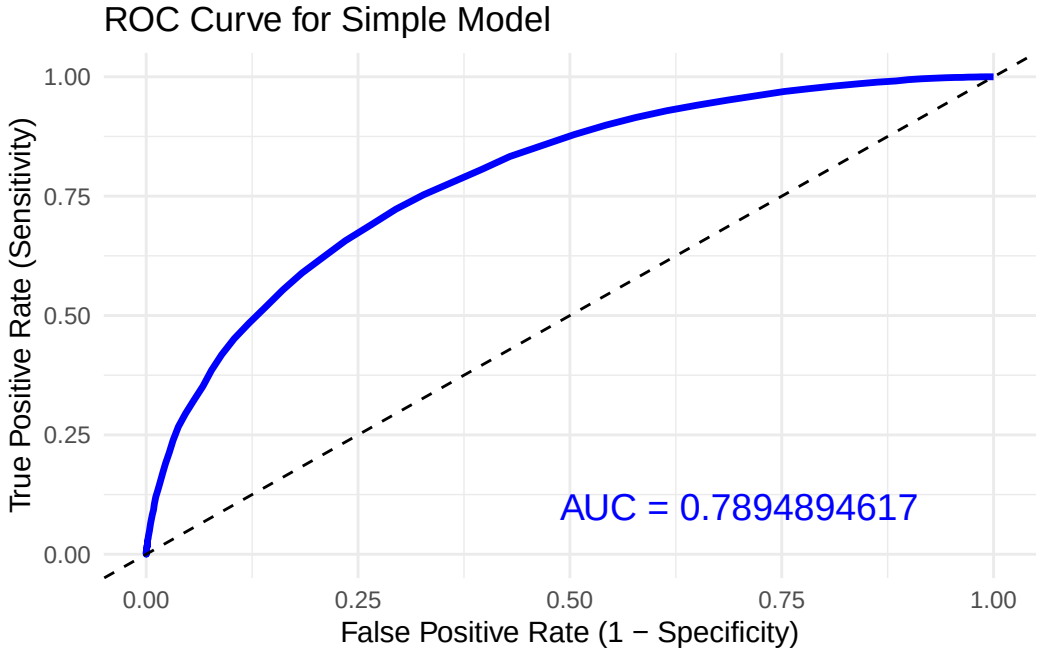


We consulted the ggplot api to help us construct this graph. [5]

Here we can see the baseline AUC in blue, specifying that a model would not do any better than randomly guessing the outcome, and so we get a curve that mimics the probability of our initial ratio of wins to losses is near .5. Meanwhile, marked in green is the perfect case, where a model's efficacy is perfect. We are aiming to achieve the green with our model.

[5] [6]

We can therefore construct our ROC curve and compute the AUC for our simple regression model to see if it does any better than random sampling (.52 AUC). We hand previously created a test set of 20% of our dataset, which we will use to compute the following.



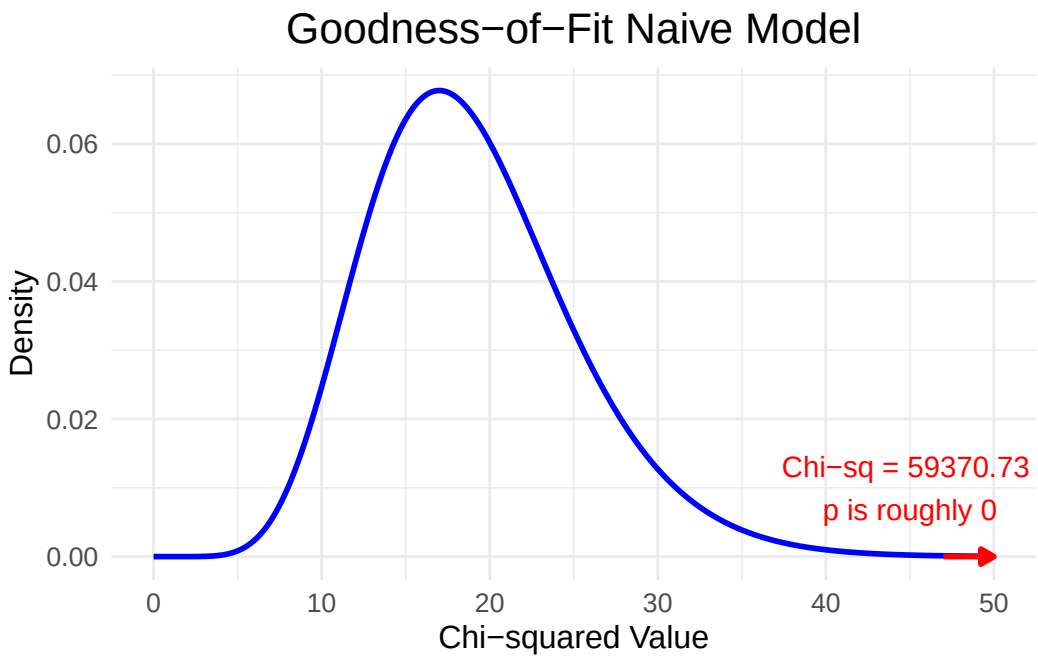
We can therefore infer that our model does improve our odds of predicting the binary classifier. **Keep in mind that this is not a measure of accuracy, rather a tool to be able to compare the efficacy some models with others.**

Naively Fitting all of our predictors of interest (Make improvements to the models)

We can now compare two models' efficacies with AUC of ROC. We will now construct a new model, computing the following tests to see how well it does with models that we have already tested. We will also test the model for model significance. We will construct our null and alternatives as:

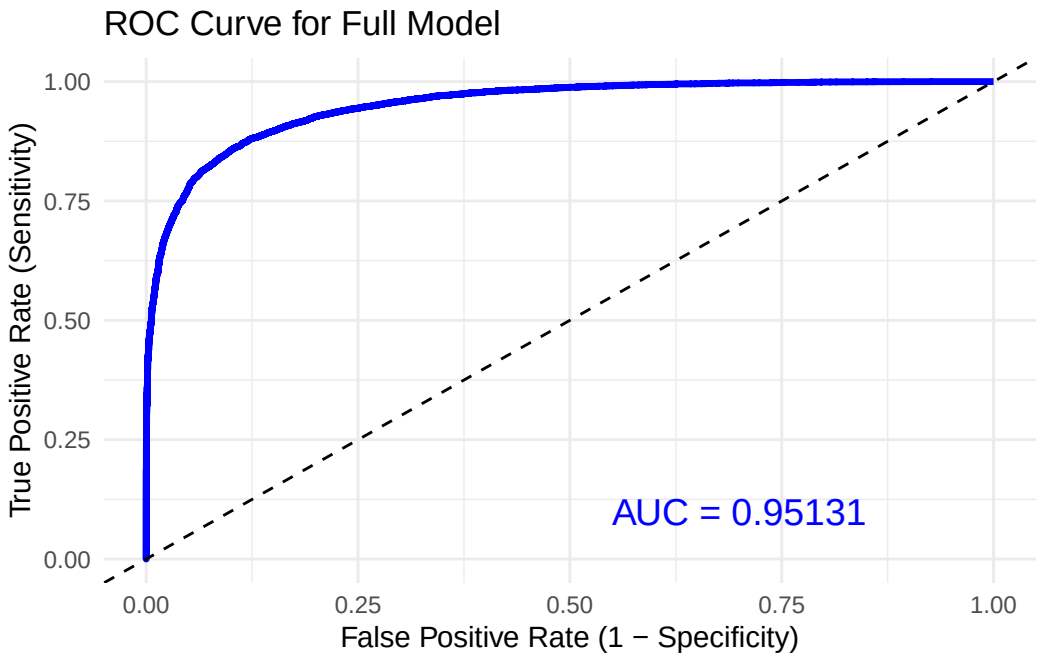
$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_i = 0$$
$$H_1 : \beta_1 = \beta_2 = \cdots = \beta_i \neq 0$$

essentially expanding our simple regression goodness of fit test for significance to include all of our parameters of interest. This is a total of 20 variables, which follows χ^2_{20}.



∴ Our new model is at least significant. Notice how we had inputted 20 variables, but only got 19 degrees of freedom. Upon closer inspection, we notice that the predictor neutral was some combination of home + away, and was therefore perfectly colinear with our 1-hot encoding. Our glm ultimately decided to expel it from our deviance calculations.

Testing for our new AUC, we get:



We can see that our model now performs significantly better than our initial model, with a new AUC of .9514.

High Multicolinearity and Fallibilities of Logistic Regression

It is important to note that logistic regression does not come without its downsides. For example, when two predictors are highly colinear (the change in one of the variables can be explained by the change in the other), the regression model has trouble discerning the overall effect that these predictors may have in tandem, even when they may be statistically significant to predictions. This problem can compound, where many variables are multicolinear, meaning there is some set $S \in i, i = \{1, \cdots, n\}$ such that each variable in S can be explained by the combination of all of the other variables in S. Furthermore, the computational method that Base R uses is iterative numerical optimization, which is used to estimate $\hat{\beta}_{MLE}$. Multicollinearity with this method can therefore lead to instability in coefficient estimates. This is caused by the fact that when values are highly colinear, rounding errors can worsen the overall prediction by causing the calculations propagated through the decision matrix to be absurdly high or low, effectively incorrectly reporting a variable's log odd influence. This incorrect fixing of the log odd influence coefficient can lead to such a strong prediction of the original training data that whenever you try to input new data for the model to try and "predict", we get that the model will perform more poorly. This is called overfitting, where too many unnecessary variables can dampen the prediction power of the model. We therefore value a more **parsimonious model**, or in other words a simpler model with less noise to reduce the possibility of overfitting. In

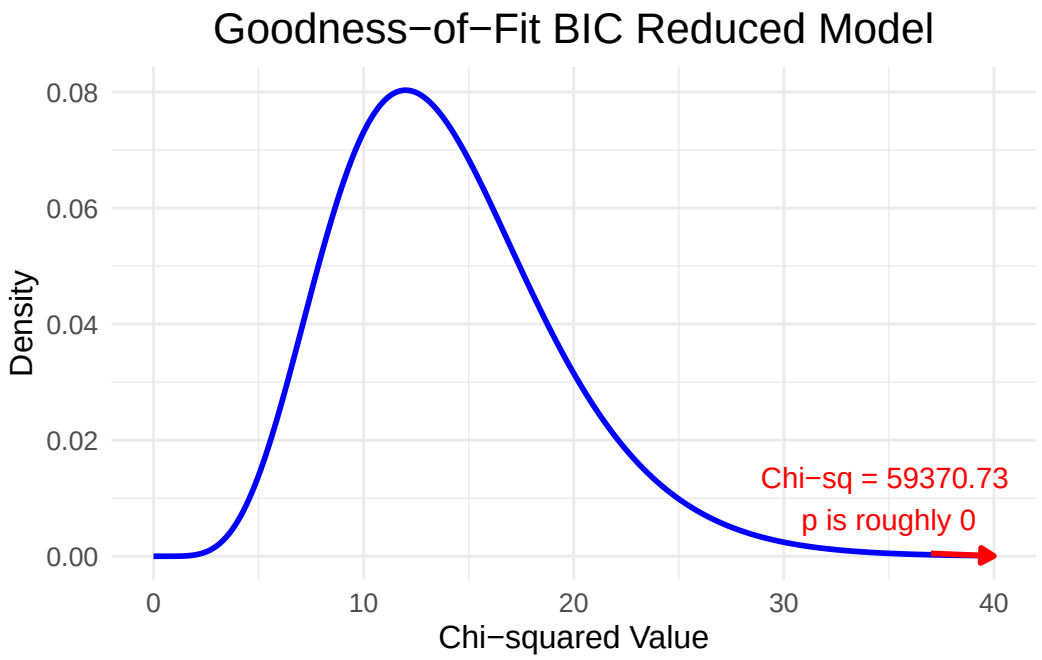
order to achieve this model, we will be employing some well-established methods to shave off excess noise from our model to avoid such clashes.

Methods to obtain a more Parsimonious Model: BIC

In order to achieve a more parsimonious model, we can employ a method known as BIC backwards elimination, where we penalize poorer fitting predictors that are highly multicolinear by a large amount. BIC then picks the greatest calculated values per predictor to eliminate, effectively expelling them from our model. In general, BIC is a more rigorous elimination method than AIC, since BIC makes use the log of the number of predictors multiplied with some penalty, where AIC uses a constant to perform the same metric. Theoretically, we can obtain a simpler model that performs just as well, if not better due to reduced overfitting, by shaving off these "garbage" predictors via this method.

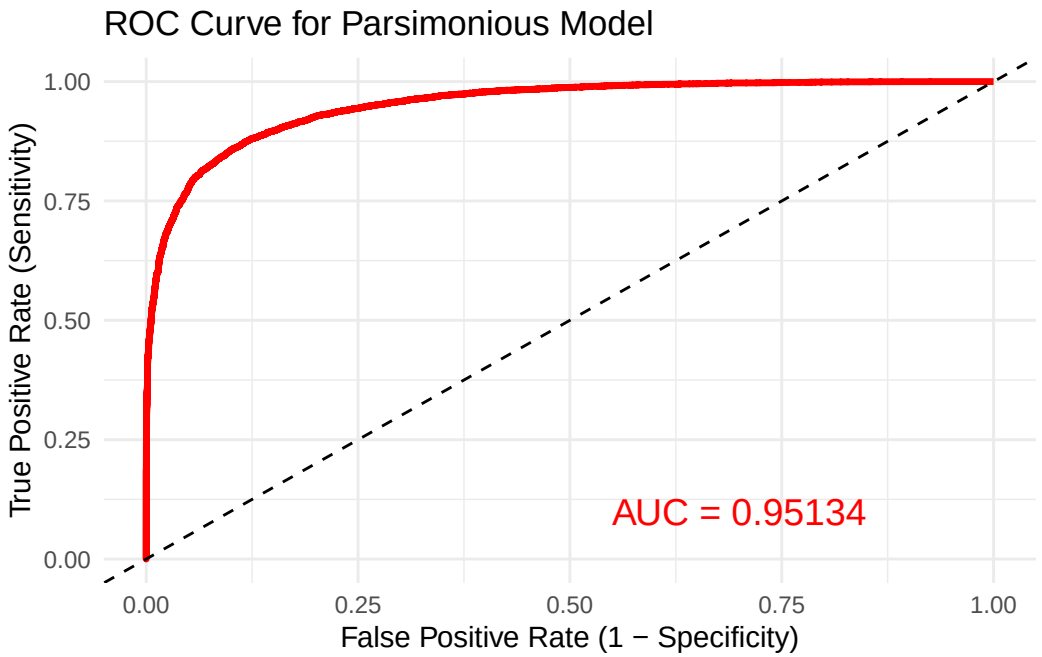
We will also construct the same tests for our new model to compare its efficacy with the other two: We will construct our null and alternatives as:

$$H_0 : \beta_1 = \beta_2 = \cdots = \beta_i = 0$$
$$H_1 : \beta_1 = \beta_2 = \cdots = \beta_i \neq 0$$



∴ Our new model is at least significant.

Testing for our new AUC, we get:

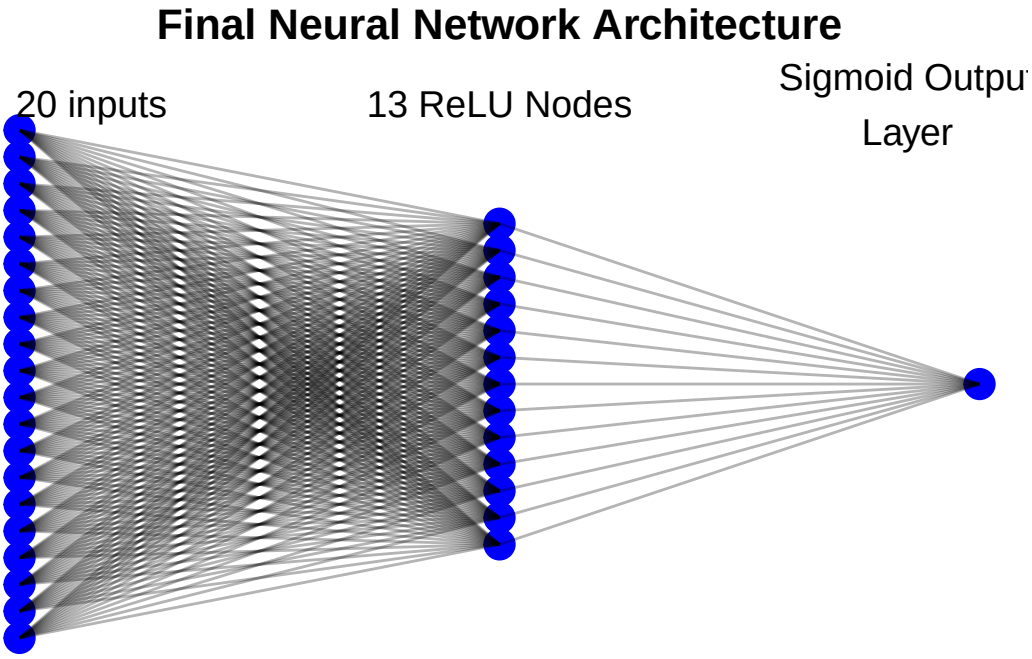


We can see the difference between the AUC for the BIC model and our original model is almost negligible, with a .00004 difference between the two.

Final Approach: Construction of a Neural Network

In a final effort to push the bounds of prediction, we can construct a neural network in order to be able to predict a game win or loss. We will therefore trade simplicity for computing power, with the sole objective to compute accurate predictions. Based on the conventions commonly used in the construction of a classifier neural network, we follow an architecture that includes an actuation function to realize any sort of non-linearity, then plug our data into a sigmoid function (which is just a logistic regression layer) to receive our final output. After some careful and rigorous hyperparameter tuning, I have ended up with the following architecture using h2o [6].

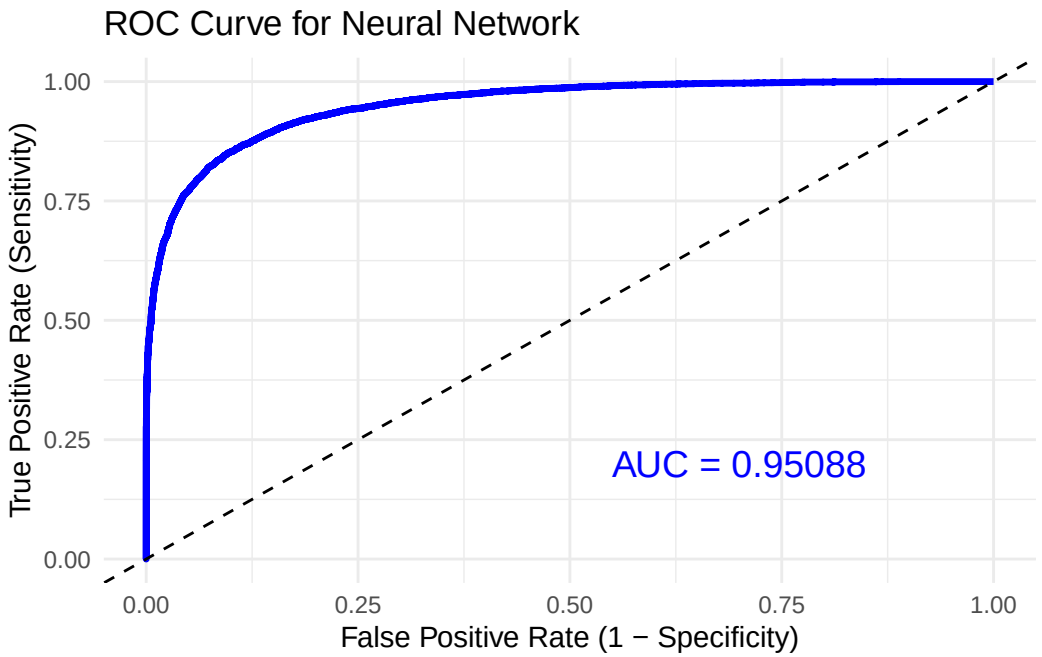
[6] [2]



If you are interested in hyperparameter tuning, here is a useful guide as to how to tune them with a fair amount of efficiency: [7]

Due to the black-box nature of neural networks, such tests that we have used previously have become almost impossible to implement, and by the graph above, we have demonstrated that understanding of the mechanisms of the neural network become increasingly obscure. Therefore, the sole purpose of this is to see if we can achieve a better prediction margin. We can therefore compare the Neural Network's AUC of the ROC against our other two models:

[7] [3]



Based on the generated ROC from our neural network We can see that the overall efficacy of the model is close to those of the logistic regression models we had generated previously. But why does this fall short

Theory: Ockham's Razor

Ockham's Razor is a convention that states that if we have multiple models that all produce roughly same efficacy margins, the simplest model that can be used to explain the data is generally preferred. While this is not set in stone, the intuition behind it makes sense, especially considering how our models may be subject to overfitting. Especially in the case of our neural network, it is many times more complex than logistic alone, where logistic regression is just one proponent of it. Because of this, the neural network is much more prone to overfitting, which could be why no matter what choice of hyperparameters I chose, I could not reach the efficacy of the logistic regression models. In terms of the BIC-reduced model, we can ultimately see that the most robust, parsimonious model is our BIC reduced model, since they all perform similarly.

Conclusion

Areas for improvement

Throughout this document, we have run into a plethora of issues including as insufficient computing power, the dimensionality of categorical variables, and time constraints. Some things to try in the future to improve the efficacy of our model analysis are:

- Obtain permission to use Neural Network Processing that is not run on a Virtual Machine to be able to implement more rigorous hyperparameter sweeps
- Obtain more computing to be able to represent all of our categorical variables in 1-hot and implement k-fold cross-validation
- Expand our exploratory research question to be able to include opponent information in some combinatory statistic with teams

Based on the convention of Ockham's Razor, we are ultimately able our most robust, yet parsimonious model, the BIC - reduced logistic regression model, to be able to accurately make predictions on whether we can obtain a win or a loss based on a team's statistics. The implications are that given any team, we can take their averages for a given time frame (for example, a season, a couple of games, all-time), and input it into our model. Our model, based on these statistics, can then "predict" based on these statistics of their overall performance, whether they are a "winning" team or a "losing" team. This can be useful to validate power ratings, for example, and extrapolate better predictions for future games.

References

[1] Nate Duncan. *2011 Current NCAA Basketball Games Dataset*. https://www.kaggle.com/datasets/nateduncan/2011current-ncaa-basketball-games. Accessed 15 Dec. 2024. 2011.

[2] H2O.ai. *Deep Learning — H2O 3.42.0.1 Documentation*. https://docs.h2o.ai/h2o/latest-stable/h2o-docs/data-science/deep-learning.html. Accessed 15 Dec. 2024. 2024.

[3] Jeremy Jordan. *A Guide to Hyperparameter Tuning*. https://www.jeremyjordan.me/hyperparameter-tuning/. Accessed 15 Dec. 2024. 2024.

[4] Stephen Michael Kosslyn. *Graph Design for the Eye and Mind*. Oxford University Press, 2006.

[5] Hyungsuk Tak. *Professor of the Eberly College of Science, Pennsylvania State University*. 2024.

[6] Tidyverse. *ggplot2 Package Reference*. https://ggplot2.tidyverse.org/reference/. Accessed 15 Dec. 2024. 2024.

[7] Edward R. Tufte. *The Visual Display of Quantitative Information*. 2nd. Graphics Press, 2001.

Code Appendix

```
library(dplyr)
library(tidyr)
basketball_data <- read.csv("Stat184ProjectTables/games.csv")
basketball_data <- na.omit(basketball_data)


# Rename specific columns
basketball_data <- basketball_data %>%
  rename(
    game_decision = w_l,
    points = pts,
    opp_points = opp_pts,
    field_goals = fg,
    field_goal_attempts = fga,
    field_goal_percentage = fg_per,
    `3pointers` = `X3p`,
    `3pointer_attempts` = `X3pa`,
    `3pointer_percentage` = `X3p_per`,
    free_throws = ft,
    free_throw_attempts = fta,
    free_throw_percentage = ft_per,
    offensive_rebounds = orb,
    total_rebounds = trb,
    assists = ast,
    steals = stl,
    blocks = blk,
    turnovers = tov,
    personal_fouls = pf,
    opp_fieldgoals = opp_fg,
    opp_fieldgoal_attempts = opp_fga,
    opp_fieldgoal_percentage = opp_fg_per,
    `opp_3pointers` = `opp_3p`,
    `opp_3pointer_attempts`= `opp_3pa`,
    `opp_3pointer_percentage` = `opp_3p_per`,
    opp_freethrows = opp_ft,
    opp_freethrow_attempts = opp_ft_per,
    opp_offensive_rebounds = opp_orb,
    opp_assists = opp_ast,
    opp_steals = opp_stl,
```

```
    opp_blocks = opp_blk,
    opp_turnovers = opp_tov,
    opp_personal_fouls = opp_pf
  )

# From all number values, remove numbers < 0
basketball_data <- basketball_data %>%
  filter(if_all(everything(), ~ . >= 0)) %>%
  filter(
    field_goal_percentage <= 1,
    `3pointer_percentage` <= 1,
    free_throw_percentage <= 1
  )

# Make it so that the opposing team values are removed, and categorical variables that can't
basketball_data <- basketball_data %>%
  select(-contains("opp")) %>%
  filter(site %in% c("Home", "Away", "Neutral")) %>%
  mutate(
    Home = ifelse(site == "Home", 1, 0),
    Away = ifelse(site == "Away", 1, 0),
    Neutral = ifelse(site == "Neutral", 1, 0)
  ) %>%
  select(-site)  # Drop the original site column

write.csv(basketball_data, "Stat184ProjectTables/renamed_games.csv", row.names = FALSE)

library(dplyr)
library(knitr)
library(kableExtra)

data <- read.csv("Stat184ProjectTables/fullmodel.csv")
data <- data %>%
  select(-game_decision,-Home,-Away,-Neutral)


stats_list <- lapply(data, function(x) {
  c(
    Min = min(x, na.rm = TRUE),
    `25%-tile` = quantile(x, 0.25, na.rm = TRUE),
    Median = median(x, na.rm = TRUE),
    `75%-tile` = quantile(x, 0.75, na.rm = TRUE),
```

```
    Max = max(x, na.rm = TRUE),
    mean = mean(x, na.rm = TRUE)
  )
})

summary_stats <- as.data.frame(do.call(rbind, stats_list))
summary_stats <- cbind(`Game Statistic` = rownames(summary_stats), summary_stats)
rownames(summary_stats) <- NULL
p <- summary_stats %>%
  kable(
    digits = 2,
    col.names = c("Game Statistic", "Min", "25%-tile", "Median",
                  "75%-tile", "Max", "mean"),
    format.args = list(big.mark = ","),
    align = "lccccccccc"
  ) %>%
  kable_classic() %>%
  add_footnote(
    notation = "none",
    threeparttable = TRUE
  )
p
library(dplyr)
library(tidyr)
library(ggplot2)

# Load the dataset
basketball.data <- read.csv("Stat184ProjectTables/renamed_games.csv")

# Convert w_l to numeric (1 for Win, 0 for Loss)
basketball.data <- basketball.data %>%
  mutate(
    game_decision = ifelse(game_decision == "W", 1,0)
  ) %>%
  na.omit(basketball.data)

# Select and summarize the data
grouped_data <- basketball.data %>%
  select(
    points,
    field_goals,
    field_goal_attempts,
```

```
    `X3pointers`,
    `X3pointer_attempts`,
    free_throws,
    free_throw_attempts,
    offensive_rebounds,
    total_rebounds,
    assists,
    steals,
    blocks,
    turnovers,
    personal_fouls,
    game_decision) %>%
  mutate(across(-game_decision, as.numeric)) %>%  # Convert all except w_l to numeric
  group_by(game_decision) %>%
  summarize(across(everything(), mean, na.rm = TRUE)) %>%
  pivot_longer(-game_decision, names_to = "Metric", values_to = "Average")

# Map w_l values to labels
grouped_data <- grouped_data %>%
  mutate(Outcome = ifelse(game_decision == 1, "Win", "Loss"))

# Create the bar plot
ggplot(grouped_data, aes(x = Metric, y = Average, fill = Outcome)) +
  geom_bar(stat = "identity", position = "dodge") +
  scale_fill_manual(values = c("blue", "red")) +
  labs(
    title = "Average Metrics Grouped by Win/Loss",
    x = "Metrics",
    y = "Average Value",
    fill = "Outcome"
  ) +
  theme_minimal() +
  theme(
    axis.text.x = element_text(angle = 45, hjust = 1, size = 12),
    axis.title = element_text(size = 14),
    plot.title = element_text(size = 16)
  )

library(dplyr)
library(tidyr)
library(ggplot2)
```

```r
basketball.data <- read.csv("Stat184ProjectTables/renamed_games.csv")

# most of this dataset is not in terms of numeric. We will generally have to turn it into nu
# for certain visualizations
basketball.data <- basketball.data %>%
  mutate(
    points = as.numeric(as.character(points)),
    field_goals = as.numeric(as.character(field_goal_percentage))
  )

#construct scatterplot
ggplot(basketball.data, aes(x = points, y = field_goal_percentage)) +
  geom_point(alpha = 0.6) +
  geom_smooth(method = "lm", color = "blue", se = FALSE) +
  theme_minimal() +
  labs(
    title = "Scatterplot of Points vs Field Goal Percentages",
    x = "Points",
    y = "Field Goal Percentage"
  ) +
  scale_y_continuous(labels = scales::percent_format()) +
  coord_cartesian(xlim = c(0, 150), ylim = c(0, 1)) +  # Manually set bounds
  annotate(
    "text",
    x = 140,
    y = 0.9,
    label = paste(
      "Correlation: ",
      round(cor(basketball.data$field_goal_percentage, basketball.data$points, use = "comple
    ),
    color = "red",
    hjust = 1
  )


library(dplyr)

basketball.data <- read.csv("Stat184ProjectTables/renamed_games.csv")

# Changes 0 and 1 so we can create some sort of numerical correlation coefficient
basketball.data <- basketball.data %>%
  mutate(game_decision = ifelse(game_decision == "W", 1, 0)) %>%
```

```r
  mutate(across(everything(), ~ as.numeric(as.character(.))))

# remove categorical variables
numeric_data <- basketball.data %>%
  select(-team_name, -game_num, -date) %>%
  select(where(is.numeric))


# Computing correlations
cor_matrix <- cor(numeric_data, use = "pairwise.complete.obs")
cor_data <- as.data.frame(as.table(cor_matrix))
colnames(cor_data) <- c("Var1", "Var2", "Correlation")

# Add a correlation greater than .95 or less than -.995
cor_data <- cor_data %>%
  mutate(Color = ifelse(abs(Correlation) >= 0.95, "black", NA))

#construct heatmap
ggplot(cor_data, aes(Var1, Var2, fill = Correlation)) +
  #baseline color is white
  geom_tile(color = "white") +
  # approaching -1 is blue, approaching 1 is red
  scale_fill_gradient2(low = "blue", high = "red", mid = "white", midpoint = 0) +
  geom_tile(data = cor_data %>% filter(!is.na(Color)), aes(Var1, Var2), fill = "black") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(
    title = "Correlation Heatmap of Game Statistics",
    x = "Variable 1",
    y = "Variable 2",
    fill = "Correlation"
  )
library(tidyr)
library(ggplot2)
library(dplyr)

basketball_data <- read.csv("Stat184ProjectTables/renamed_games.csv")
basketball_data <- na.omit(basketball_data)

# turn our binary variable of interest to 1's and 0's
fullmodel <- basketball_data %>%
  select(-team_name,-date,-game_num) %>%
```

```
  mutate(
    game_decision = ifelse(game_decision == "W", 1, 0)
  )
fullmodel <- fullmodel %>%
  mutate(across(everything(), ~ as.numeric(as.character(.))))

# Set a seed for reproducibility random sampling
# We do NOT use k-fold due to computational limitations
set.seed(112911)
train_index <- sample(1:nrow(fullmodel), 0.8 * nrow(fullmodel))
trainingset <- fullmodel[train_index, ]  # 80% training data

#Create and SAVE test and training sets, as well as the full model for some comaparisons in t
testset <- fullmodel[-train_index, ] # 20%
write.csv(fullmodel, "Stat184ProjectTables/fullmodel.csv", row.names = FALSE)
write.csv(trainingset,"Stat184ProjectTables/trainingset.csv",row.names = FALSE)
write.csv(testset,"Stat184ProjectTables/testset.csv",row.names = FALSE)

library(ggplot2)
trainingset <- read.csv("Stat184ProjectTables/trainingset.csv")

#needed in order to vectorize the variables, otherwise glm will not work
points <- trainingset$points

#logistic regression model
singleregression <- glm(game_decision ~
                          points,
                        data = trainingset,
                        family = binomial)

#creation of dataframe with min and max values for a graph to plug into ggplot
points_seq <- seq(min(trainingset$points, na.rm = TRUE), max(trainingset$points, na.rm = TRU

#exponentiate as talked about above
predicted_odds <- exp(predict(singleregression, newdata = data.frame(points = points_seq)))

#logit then convert to dataframe
predicted_probabilities <- predicted_odds / (1 + predicted_odds)
prediction_df <- data.frame(points = points_seq, probability = predicted_probabilities)

# Plot the change in odds
ggplot(prediction_df, aes(x = points, y = probability)) +
```

```
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(
    title = "Change in Odds for unit increases of Points",
    x = "Points",
    y = "Probability of Winning"
  )

library(ggplot2)


# Extract null and residual deviance
null_deviance <- singleregression$null.deviance
residual_deviance <- singleregression$deviance

# Compute degrees of freedom
df_null <- singleregression$df.null
df_residual <- singleregression$df.residual
df_diff <- df_null - df_residual
p_value <- 1-pchisq(df_residual,df_diff)

# Generate x values
x_vals <- seq(0, 10, length.out = 500)

# Compute the chi-squared density with 19 degrees of freedom
y_vals <- dchisq(x_vals, df = df_diff)

# Create a data frame for plotting
chi_sq_data <- data.frame(x = x_vals, density = y_vals)

# Plot the chi-squared distribution
ggplot(chi_sq_data, aes(x = x, y = density)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(
    title = "Goodness-of-Fit for Simple Model",
    x = "Chi-squared Value",
    y = "Density"
  ) +
  annotate(
    "text",
    x = 9,
```

```
    y = .5,  # Adjust label height
    label = paste("Chi-sq =", round(residual_deviance, 2), "\np is roughly", signif(p_value,
    color = "red",
    hjust = 0.5
  ) +
    annotate(
    "segment",
    x = 9.5, xend = 10,  # Starting and ending x-coordinates for the arrow
    y = 0, yend = 0,    # Starting and ending y-coordinates for the arrow
    arrow = arrow(length = unit(0.2, "cm"), type = "closed"),  # Arrow settings
    color = "red",       # Arrow color
    size = 1             # Arrow thickness
  ) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10)
  )


library(ggplot2)
library(pROC)

# Create random predictions and true labels for the random classifier
set.seed(112911)
n <- 1000
true_labels <- sample(c(0, 1), size = n, replace = TRUE)
random_predictions <- runif(n)  # Random predictions between 0 and 1

# Compute the ROC curve for random predictions
roc_random <- roc(true_labels, random_predictions)
auc_random <- auc(roc_random)

# Extract data for the random ROC curve
roc_data_random <- data.frame(
  specificity = rev(roc_random$specificities),  # X-axis (1 - specificity)
  sensitivity = rev(roc_random$sensitivities)   # Y-axis
)

# Create a perfect ROC curve matrix
roc_data_perfect <- data.frame(
  specificity = c(1, 1, 0),
```

```
  sensitivity = c(0, 1, 1)
)
auc_perfect <- 1.0  # Perfect classifier has AUC = 1


# Combine the data and add a 'Type' column for ggplot
roc_data_random$Type <- "Random Classifier"
roc_data_perfect$Type <- "Perfect Classifier"
roc_data_combined <- rbind(roc_data_random, roc_data_perfect)

# Plot the ROC curves with AUC annotations
ggplot(roc_data_combined, aes(x = 1 - specificity, y = sensitivity, color = Type)) +
  geom_line(size = 1) +
  scale_color_manual(values = c("Random Classifier" = "red", "Perfect Classifier" = "green"))
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "black", size = 1, show
  theme_minimal() +
  labs(
    title = "ROC Curves: Random Classifier vs Perfect Classifier",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)",
    color = "Classifier Type"
  ) +
  annotate(
    "text",
    x = 0.38, y = 0.2,
    label = paste("Random AUC =", round(.489, 3)),
    color = "red",
    size = 5,
    hjust = 0
  ) +
  annotate(
    "text",
    x = 0.5, y = 0.1,
    label = paste("Perfect AUC =", round(auc_perfect, 10)),
    color = "green",
    size = 5,
    hjust = 0
  ) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10),
```

```
    legend.title = element_text(size = 12),
    legend.text = element_text(size = 10)
  )
library(ggplot2)
library(dplyr)

# calculate ROC and AUC values
testset <- read.csv("Stat184ProjectTables/testset.csv")
random_probs <- predict(singleregression, newdata = testset, type = "response")
random_auc <- auc(testset$game_decision, random_probs)
random_roc <- roc(testset$game_decision, random_probs)

#construct df for ggplot w values from prev
random_roc_df <- data.frame(
  FPR = 1 - random_roc$specificities,
  TPR = random_roc$sensitivities
)

ggplot(random_roc_df, aes(x = FPR, y = TPR)) +
  geom_line(color = "blue", size = 1.2) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "black") +
  labs(
    title = "ROC Curve for Simple Model",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +
  annotate("text", x = 0.7, y = 0.1, label = paste("AUC =", round(random_auc, 10)), size = 5
library(pROC)
library(ggplot2)
ts <- read.csv("Stat184ProjectTables/trainingset.csv")


# need to vectorize all columns of the training set
# courtesy of Dr. Hyungusk Tak
game_decision <- ts$game_decision
points <- ts$points
field_goals<- ts$field_goals
field_goal_attempts<- ts$field_goal_attempts
field_goal_percentage<- ts$field_goal_percentage
`X3pointers`<- ts$`X3pointers`
`X3pointer_attempts`<- ts$`X3pointer_attempts`
```

```
`X3pointer_percentage`<- ts$`X3pointer_percentage`
free_throws<- ts$free_throws
free_throw_attempts<- ts$free_throw_attempts
free_throw_percentage<- ts$free_throw_percentage
offensive_rebounds<- ts$offensive_rebounds
total_rebounds<- ts$total_rebounds
assists<- ts$assists
steals<- ts$steals
blocks<- ts$blocks
turnovers<- ts$turnovers
personal_fouls<- ts$personal_fouls
Home<- ts$Home
Away<- ts$Away
Neutral<- ts$Neutral


#logisitic regression
naivemodel <- glm(game_decision ~
                    points +
                    field_goals +
                    field_goal_attempts +
                    field_goal_percentage +
                    `X3pointers` +
                    `X3pointer_attempts` +
                    `X3pointer_percentage`+
                    free_throws+
                    free_throw_attempts+
                    free_throw_percentage+
                    offensive_rebounds+
                    total_rebounds+
                    assists+
                    steals+
                    blocks+
                    turnovers+
                    personal_fouls+
                    Home+
                    Away+
                    Neutral,
                  data = trainingset,
                  family = binomial
)
```

```r
# Extract null and residual deviance
null_deviance <- naivemodel$null.deviance
residual_deviance <- naivemodel$deviance

# Compute degrees of freedom
df_null <- naivemodel$df.null
df_residual <- naivemodel$df.residual
df_diff <- df_null - df_residual

# Compute chi-squared statistic
chisq_stat <- null_deviance - residual_deviance

# Compute p-value
p_value <- 1 - pchisq(chisq_stat, df_diff)
library(ggplot2)

# Generate x values
x_vals <- seq(0, 50, length.out = 500)

# Compute the chi-squared density with 19 degrees of freedom
y_vals <- dchisq(x_vals, df = df_diff)

# Create a data frame for plotting
chi_sq_data <- data.frame(x = x_vals, density = y_vals)

# Plot the chi-squared distribution
ggplot(chi_sq_data, aes(x = x, y = density)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(
    title = "Goodness-of-Fit Naive Model",
    x = "Chi-squared Value",
    y = "Density"
  ) +
  annotate(
    "text",
    x = 45,
    y = 0.01,  # Adjust label height
    label = paste("Chi-sq =", round(chisq_stat, 2), "\np is roughly", signif(p_value, 3)),
    color = "red",
    hjust = 0.5
  ) +
```

```
    annotate(
    "segment",
    x = 47, xend = 50,
    y = 0, yend = 0,    # Starting and ending coords for arrow
    arrow = arrow(length = unit(0.2, "cm"), type = "closed"),  # Arrow settings
    color = "red",       # Arrow color
    size = 1             # Arrow thickness
  ) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10)
  )

testset <- read.csv("Stat184ProjectTables/testset.csv")

naivemodel_probs <- predict(naivemodel, newdata = testset, type = "response")
naivemodel_auc <- auc(testset$game_decision, naivemodel_probs)
naivemodel_roc <- roc(testset$game_decision, naivemodel_probs)

# Data frame cast for correct ggplot usage
naivemodel_roc_df <- data.frame(
  FPR = 1 - naivemodel_roc$specificities,
  TPR = naivemodel_roc$sensitivities
)

# naivemodel plot
roc_plot_naivemodel <- ggplot(naivemodel_roc_df, aes(x = FPR, y = TPR)) +
  geom_line(color = "blue", size = 1.2) +
  geom_abline(slope = 1,
              intercept = 0,
              linetype = "dashed",
              color = "black") +
  labs(
    title = "ROC Curve for Full Model",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +
  annotate("text", x = 0.7, y = 0.1, label = paste("AUC =", round(naivemodel_auc, 5)), size =

roc_plot_naivemodel
```

```r
#runs code for BIC backwards elim. WARNING: computationally expensive, takes time
parsimoniousmodel <- step(naivemodel, k=log(20))

# Extract null and residual deviance
null_deviance <- parsimoniousmodel$null.deviance
residual_deviance <- parsimoniousmodel$deviance

# Compute degrees of freedom
df_null <- parsimoniousmodel$df.null
df_residual <- parsimoniousmodel$df.residual
df_diff <- df_null - df_residual

# Generate x values
x_vals <- seq(0, 40, length.out = 500)

# Compute the chi-squared density with 19 degrees of freedom
y_vals <- dchisq(x_vals, df = df_diff)

# Create a data frame for plotting
chi_sq_data <- data.frame(x = x_vals, density = y_vals)

# Plot the chi-squared distribution
ggplot(chi_sq_data, aes(x = x, y = density)) +
  geom_line(color = "blue", size = 1) +
  theme_minimal() +
  labs(
    title = "Goodness-of-Fit BIC Reduced Model",
    x = "Chi-squared Value",
    y = "Density"
  ) +
  annotate(
    "text",
    x = 35,
    y = 0.01,  # Adjust label height
    label = paste("Chi-sq =", round(chisq_stat, 2), "\np is roughly", signif(p_value, 3)),
    color = "red",
    hjust = 0.5
  ) +
    annotate(
    "segment",
    x = 37, xend = 40,  # Starting and ending x-coordinates for the arrow
    y = 0.0005, yend = 0.00005,    # Starting and ending y-coordinates for the arrow
```

```
    arrow = arrow(length = unit(0.2, "cm"), type = "closed"),  # Arrow settings
    color = "red",       # Arrow color
    size = 1             # Arrow thickness
  ) +
  theme(
    plot.title = element_text(size = 16, hjust = 0.5),
    axis.title = element_text(size = 12),
    axis.text = element_text(size = 10)
  )

testset <- read.csv("Stat184ProjectTables/testset.csv")
parsimonious_probs <- predict(parsimoniousmodel, newdata = testset, type = "response")
parsimonious_auc <- auc(testset$game_decision, parsimonious_probs)
parsimonious_roc <- roc(testset$game_decision, parsimonious_probs)
parsimonious_roc_df <- data.frame(
  FPR = 1 - parsimonious_roc$specificities,
  TPR = parsimonious_roc$sensitivities
)
# BIC model plot
roc_plot_parsimonious <- ggplot(parsimonious_roc_df, aes(x = FPR, y = TPR)) +
  geom_line(color = "red", size = 1.2) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "black") +
  labs(
    title = "ROC Curve for Parsimonious Model",
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +
  annotate("text", x = 0.7, y = 0.1, label = paste("AUC =", round(parsimonious_auc, 5)), size
roc_plot_parsimonious
library(ggplot2)
library(dplyr)

# Define the structure of the neural network
input_nodes <- 20
hidden_nodes <- 13
output_nodes <- 1

# Calculate vertical midpoint for centering layers
max_nodes <- max(input_nodes,
                 hidden_nodes,
                 output_nodes)
```

```
input_y <- seq(1,
               max_nodes,
               length.out = input_nodes)
hidden_y <- seq(1 + (max_nodes - hidden_nodes) / 2,
                hidden_nodes + (max_nodes - hidden_nodes) / 2,
                length.out = hidden_nodes)
output_y <- seq(1 + (max_nodes - output_nodes) / 2,
                output_nodes + (max_nodes - output_nodes) / 2,
                length.out = output_nodes)

# Create a data frame for nodes
nodes <- data.frame(
  layer = c(rep("Input", input_nodes),
            rep("Hidden (ReLU Activation)", hidden_nodes),
            rep("Output (Sigmoid)", output_nodes)),
  node_id = c(1:input_nodes, 1:hidden_nodes, 1:output_nodes),
  x = c(rep(1, input_nodes), rep(2, hidden_nodes), rep(3, output_nodes)),
  y = c(input_y, hidden_y, output_y)
)

# Create a data frame for connections
connections <- expand.grid(
  from = 1:input_nodes,
  to = (1:hidden_nodes) + input_nodes
) %>%
  mutate(x_from = 1, y_from = input_y[from], x_to = 2, y_to = hidden_y[to - input_nodes]) %>%
  bind_rows(expand.grid(
    from = (1:hidden_nodes) + input_nodes,
    to = (1:output_nodes) + input_nodes + hidden_nodes
  ) %>% mutate(x_from = 2, y_from = hidden_y[from - input_nodes], x_to = 3, y_to = output_y[t

# Plot the neural network
ggplot() +
  # Plot nodes
  geom_point(data = nodes, aes(x = x, y = y), size = 5, color = "blue") +
  geom_segment(data = connections,
               aes(x = x_from,
                   y = y_from,
                   xend = x_to,
                   yend = y_to),
               alpha = 0.3) +
  # Add labels for layers
```

```
  annotate("text", x = 1.15,
           y = max_nodes + 2,
           label = "\n20 inputs",
           size = 5,
           hjust = 0.5) +
  annotate("text",
           x = 2,
           y = max_nodes + 2,
           label = "\n13 ReLU Nodes",
           size = 5,
           hjust = 0.5) +
  annotate("text",
           x = 2.85,
           y = max_nodes + 2,
           label = "\nSigmoid Output\nLayer",
           size = 5,
           hjust = 0.5) +
  # Customize the plot
  theme_void() +
  labs(title = "Final Neural Network Architecture") +
  theme(
    plot.title = element_text(face = "bold", size = 16, hjust = 0.5)
  )


# It is necessary to have JVM 18.0.2.1 installed to be able to render this
# Neural Net correctly.
library(h2o)
library(ggplot2)

testset <- read.csv("Stat184ProjectTables/testset.csv")
trainingset <- read.csv("Stat184ProjectTables/trainingset.csv")

h2o.init()

trainingset$game_decision <- as.factor(trainingset$game_decision)
testset$game_decision <- as.factor(testset$game_decision)

training_h2o <- as.h2o(trainingset)
test_h2o <- as.h2o(testset)

predictors <- colnames(training_h2o)[-which(colnames(training_h2o) == "game_decision")]
```

```r
target <- "game_decision"

##NN model with tuned hyperparameters after sweeps based on intuition
model <- h2o.deeplearning(
  x = predictors,
  y = target,
  training_frame = training_h2o,
  nfolds = 5,                  #This runs on VM, so can do cross-validation
  validation_frame = test_h2o,
  hidden = c(13),              # One hidden layer with 13 nodes
  epochs = 39,
  activation = "Rectifier",    # ReLU activation
  loss = "CrossEntropy",
  stopping_metric = "logloss",
  stopping_rounds = 3,         # Stop training if no improvement for 3 rounds
  stopping_tolerance = 0.001,
  seed = 112911,
)

perf <- h2o.performance(model, newdata = test_h2o)
# Make predictions on the test set
predictions <- h2o.predict(model, test_h2o)

nn_probs <- as.data.frame(predictions)$p1

nn_true_labels <- as.vector(test_h2o$game_decision)

# Compute AUC
nn_roc <- roc(nn_true_labels, nn_probs)
nn_auc <- auc(nn_roc)

# Generate ROC data for ggplot
nn_roc_df <- data.frame(
  FPR = 1 - nn_roc$specificities,
  TPR = nn_roc$sensitivities
)

nn_plot <- ggplot(nn_roc_df, aes(x = FPR, y = TPR)) +
  geom_line(color = "blue", size = 1.2) +
  geom_abline(slope = 1, intercept = 0, linetype = "dashed", color = "black") +
  labs(
    title = "ROC Curve for Neural Network",
```

```
    x = "False Positive Rate (1 - Specificity)",
    y = "True Positive Rate (Sensitivity)"
  ) +
  theme_minimal() +
  annotate(
    "text", x = 0.7, y = 0.2,
    label = paste("AUC =", round(nn_auc, 5)),
    size = 5, color = "blue"
  )

nn_plot
```