Drew Whitaker, Ben Winbinger, Cole Erusha, Leo Piscitello

Final Report

To start our report, we would like to explain why we chose a cybersecurity angle in today's world. Cybersecurity has become a massive concern across organizations and all sectors of information as sensitive data can be breached and stolen in seconds and without a trace if the proper securities are not in place. Cyber threats are constantly evolving in complexity and frequency, as well as the different ways of targeting systems, user behavior, and networks. This report's goal and plan is to explore a comprehensive cybersecurity intrusion detection dataset, designed to analyze and help with development of intrusion detection systems.

Our cybersecurity intrusion detection dataset was picked to demonstrate a real world representation of network activity. It has both network-level and user behavior features, which is useful for learning models aimed at distinguishing between harmless and malicious activities. With a total of 9,537 records, each corresponding to a unique session ID, the dataset includes important attributes such as packet size, login attempts, encryption protocol, failed login attempts, session duration, browser types, and a binary label indicating whether an attack was detected. Intrusion detection plays an important role in cybersecurity defense mechanisms and strategy. Traditional rule-based detection systems often struggle to adapt to new and sophisticated attack patterns. The use of machine learning allows systems to learn from historical attack data and use patterns in a way to detect novel threats easier. The motivation behind exploring this dataset is the need to evaluate various data-driven methods that can enhance the efficiency and accuracy of IDS. We can apply supervised learning models to classify future sessions and aid us in stopping different threats. Furthermore, the dataset’s diversity in feature types, ranging from numeric to categorical, gives the opportunity to practice comprehensive preprocessing and modeling strategies.

The dataset includes the following ten features, along with a target label attack\_detected:

network\_packet\_size: Numeric representation of the packet size in bytes, ranging from small control packets (~64 bytes) to large data-bearing packets (~1500 bytes). Unusual sizes can be indicative of reconnaissance or exploitation attempts.

protocol\_type: A categorical feature indicating the communication protocol (TCP, UDP, ICMP). Each protocol has distinct characteristics, with ICMP often abused in DoS attacks.

encryption\_used: Specifies the type of encryption (AES, DES, or None). The absence or weakness of encryption can be exploited by attackers to intercept or manipulate data.

login\_attempts: A count of how many login attempts were made during the session. Excessive attempts may point to brute-force attacks.

session\_duration: The time, in seconds, that the session was active. Extremely long durations may indicate unauthorized access or an attempt to maintain persistence.

failed\_logins: Number of unsuccessful login attempts. High numbers may signal credential stuffing or password-guessing attacks.

unusual\_time\_access: A binary flag (0 or 1) that highlights whether access occurred during non-standard hours, which can be a tactic used by attackers to avoid detection.

ip\_reputation\_score: A float value between 0 and 1 indicating the risk level associated with the IP address. Higher scores correspond to lower trustworthiness.

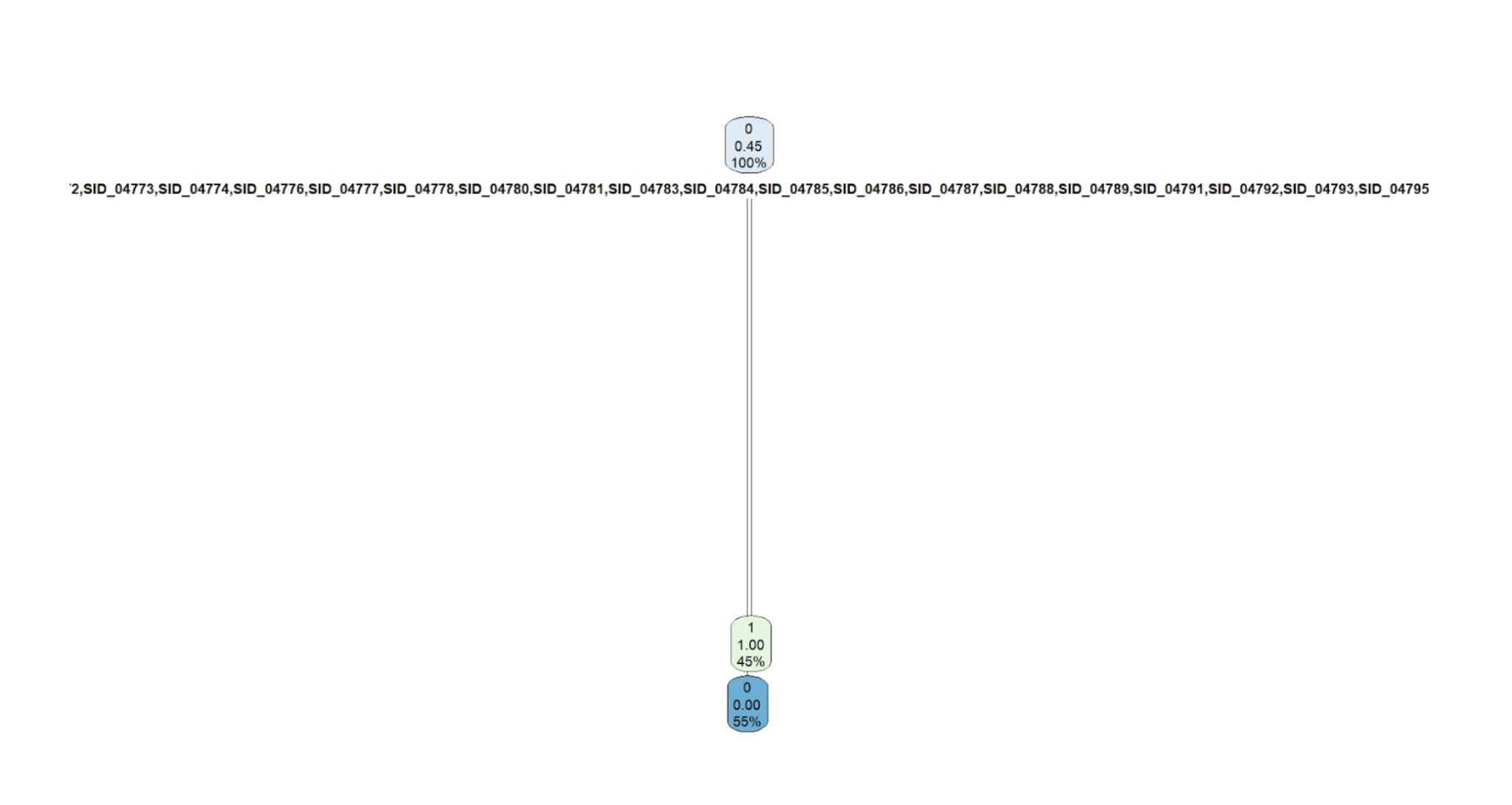
browser\_type: Categorical variable representing the user’s browser. While common browsers are typically benign, unknown or uncommon types may hint at automated scripts or bot activity.

attack\_detected: A binary label (0 for normal, 1 for attack), serving as the target variable for classification.

The dataset also contains a session\_id, a unique identifier that helps distinguish individual activity records.

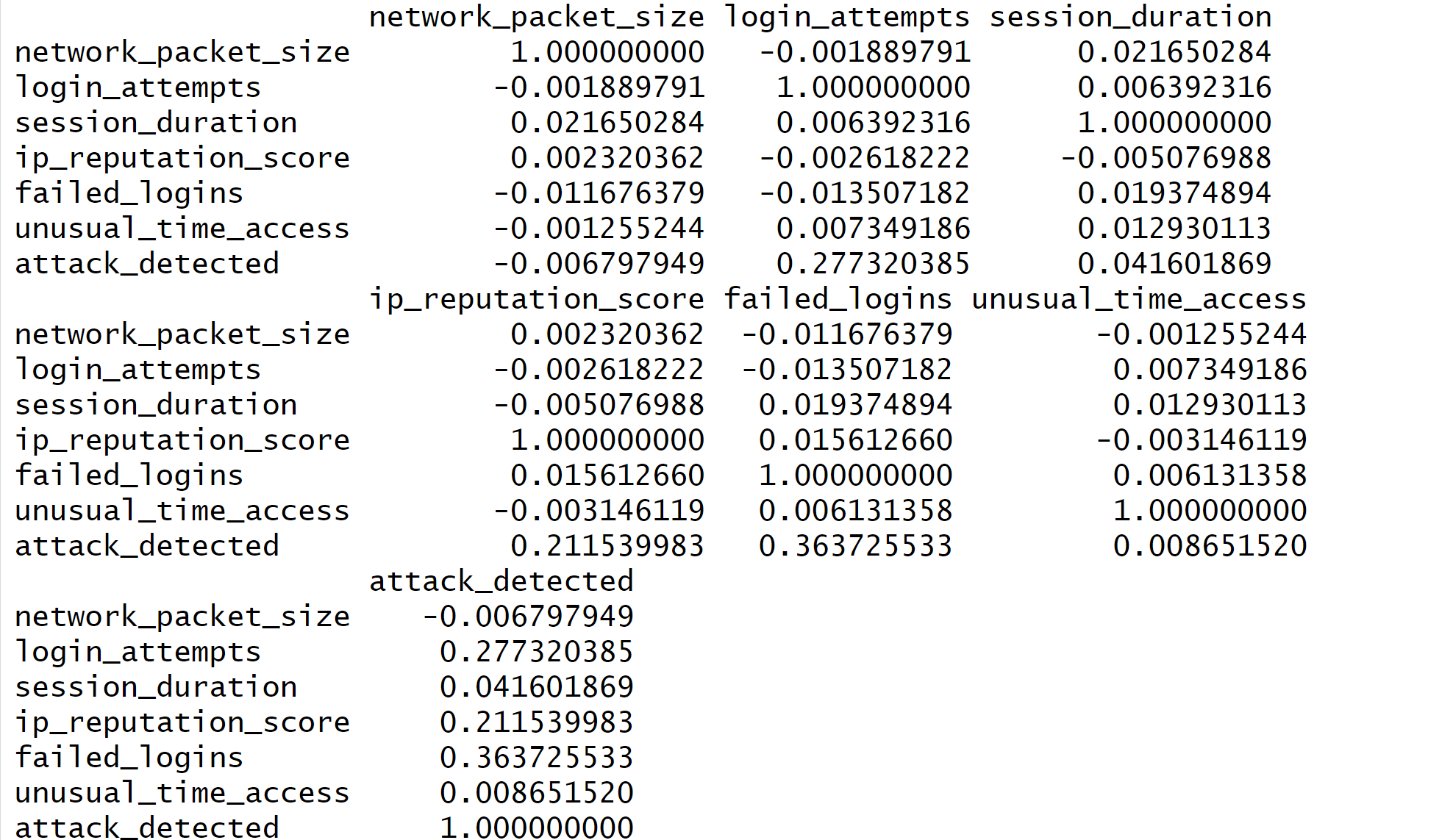
Our first step after we loaded our dataset into R Studio was to clean the dataset by checking for duplicates and null variables, but none were found.

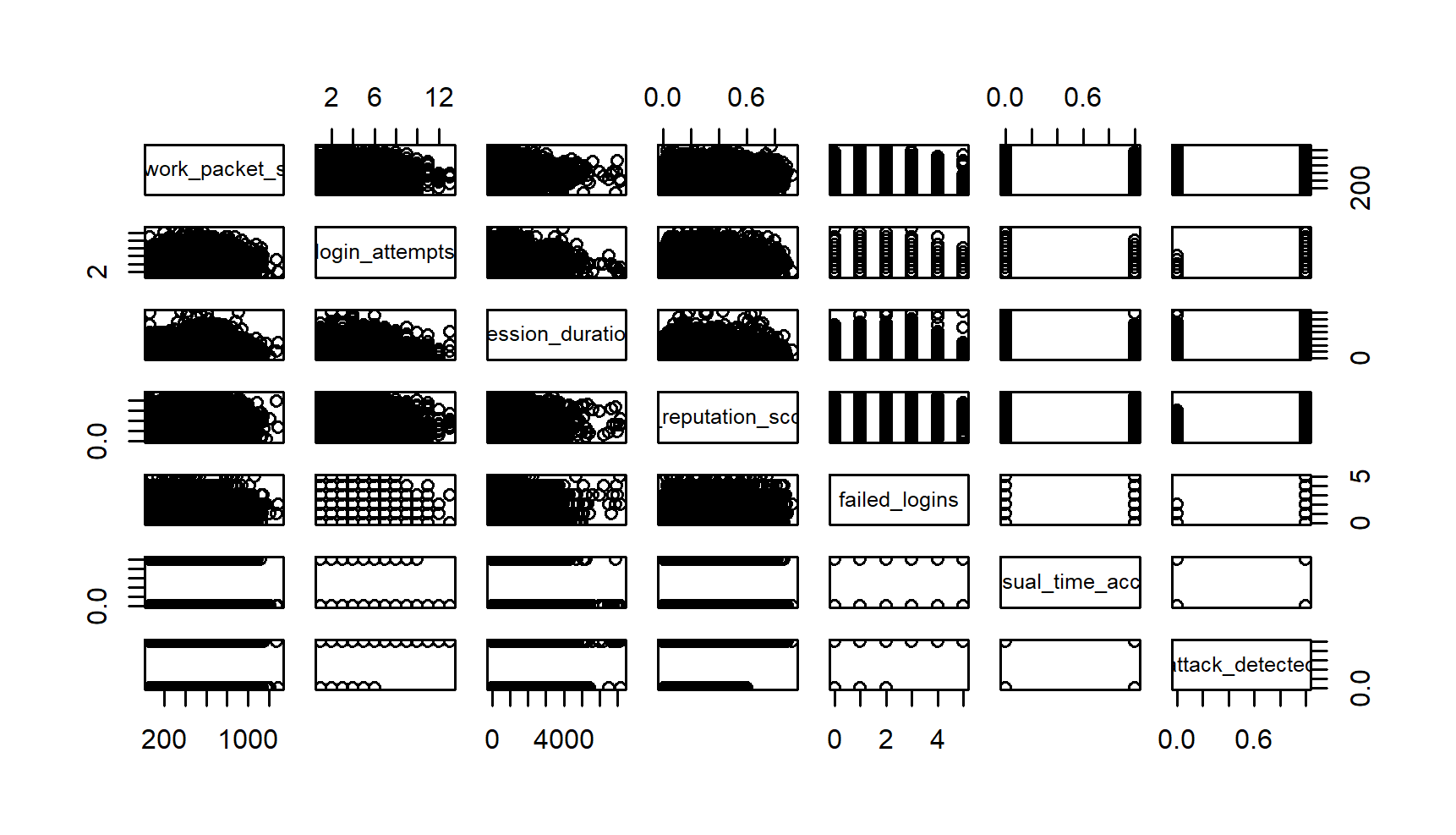
Our original idea was to create a classification tree to attempt to predict whether a cyber-attack will happen based on our 11 variables. This tree model was trained on the entire dataset with just under 10,000 objects with attack\_detected as the target variable. The decision tree model was easy to implement, but we noticed some limitations that led us to look into other models. The tree was built with (minsplit = 0), which allowed it to grow excessively deep, which would lead to overfitting by tailoring itself too closely to the training data. While the tree achieved near-perfect splits on the training data, it lacked generalization, which could lead to significantly lower performance on unseen data. Our main deterrent with the tree model was the fact that each data point was classified into either 1 (attack detected) or 0 (no attack detected). This meant that the tree didn’t offer any probability estimates for predictions. For example, logistical regression and random forest models could tell us a specific percentage as the confidence level. But with the tree each prediction was either attack or no attack.



As you can see, the tree has very few splits and limited depth which results in minimal branching. The visual structure doesn’t clearly show how different features influence predictions. The variable names at the top are also very cluttered and difficult to interpret as many even overlap.

Due to these issues, we decided to use a linear regression model instead to narrow it down to a few predictors and how they influence the odds of an attack. We ran a correlation matrix and a scatterplot matrix to find relationships between variables in order to narrow our search down to a few predictors that might lead to positive relationships.





Each off-diagonal cell shows a scatterplot between the variable in the row and the variable in the column. For example, the second row, first column shows login attempts vs network packet size. In order to read this, we must look at the dots and if they form a clear line or curve there is likely a relationship. If the dots are spread out randomly, there is likely no correlation. The vertical lines or striped bars indicate the variable is binary or discrete. Some key observations from this are that session duration is inversely related to login attempts, failed logins, and reputation score. Since our data set included roughly 10,000 records, the scatter plot can be difficult to read, but strong diagonal lines indicate strong relationships.

While this data and scatter plot can be confusing to interpret at first, we were first able to select our three predictors, **Failed Logins, Login Attempts, IP Reputation Score** because their correlation values were the highest with attack\_detected.

* failed\_logins 0.36 - The strongest correlation with attack\_detected. More failed login attempts are associated with attacks.
* login\_attempts 0.28 - Moderate correlation. More login attempts could indicate attack attempts.
* Ip\_reputation\_score 0.21 - A suspicious IP reputation is somewhat linked to attacks.

After selecting these predictors, we ran our linear regression model.

model <- glm(attack\_detected ~ failed\_logins + login\_attempts + ip\_reputation\_score,

data = data, family = binomial)

Since our target variable, attack\_detected, is binary meaning it results in 0 or 1, we ran this to predict the probability of our three independent variables on an attack being detected. A positive value indicates an increased chance of attack, and negative indicates decreased chance.

exp(coef(model))

This was run to convert the log odds into ratios to better represent change.

odds of increasing attack after each variable

* 0.0137 when all predictors are 0
* 2.55 for every addition failed login (positive relationship)
* 1.48 for each addition login attempt (positive relationship)
* 22.71 for each increase in 1 unit of Ip reputation score (very strong relationship)

Due to the range of IP\_Reputation\_Score, and strong increase we changed the unit to increase by 0.1 instead of 1.

odds\_ratios["ip\_reputation\_score"] <- exp(coef(model)["ip\_reputation\_score"] \* 0.1)

We then converted those odds to percentages to more clearly demonstrate how each variable affects our target variable.

percentage\_changes <- (odds\_ratios - 1) \* 100

failed logins - 155%

login attempts - 48%

ip reputation score - 37%

Failed logins and login attempts represent the percent increase with one additional log, ip reputation score represents a 0.1 increase.

Next, we ran a confusion matrix to see what predictions are correct, and where the model was making possible mistakes. Since we used binary classification with 0 = no attack and 1 = attack the matrix looked like this.

Predicted = 0, Actual = 0 - True negative

Predicted = 0, Actual = 1 - False negative

Predicted = 1, Actual = 0 - False positive

Predicted = 1, Actual = 1 - True positive

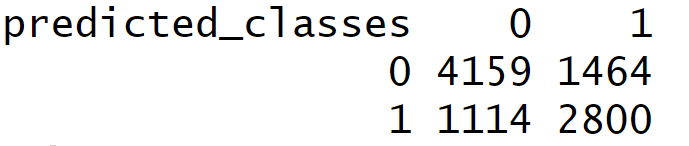
True negative - Model predicted no attack, and there was no attack

False negative - Model predicted no attack, but there was an attack (most dangerous in cybersecurity)

False positive - Model predicted an attack, but there was no attack

True positive - Model predicted an attack, and there was an attack

Here were our findings:



We then defined our confusion matrix values to calculate accuracy, precision, recall, and an F1 score

# Define confusion matrix values

TN <- 4500 # True Negatives

FP <- 1100 # False Positives

FN <- 800 # False Negatives

TP <- 3300 # True Positives

Accuracy: Overall correctness of the model

accuracy <- (TP + TN) / (TP + TN + FP + FN)

print(paste("Accuracy:", round(accuracy, 4)))

Accuracy - 0.8041

Precision: How many predicted attacks were correct

precision <- TP / (TP + FP)

print(paste("Precision:", round(precision, 4)))

Precision - 0.75

Recall: How many actual attacks were correctly identified

recall <- TP / (TP + FN)

print(paste("Recall:", round(recall, 4)))

Recall - 0.8049

F1 Score: Mean of precision & recall

f1\_score <- 2 \* (precision \* recall) / (precision + recall)

print(paste("F1 Score:", round(f1\_score, 4)))

F1 score - 0.7765

Accuracy 80% Model correctly predicts about 80% of cases

Precision 75% When it predicts an attack, it’s right 75% of the time

Recall 80% The model catches 80% of actual attacks

F1 Score 78% Balance between precision & recall

While our linear regression model did a lot of things well, we would have liked to see higher accuracy, so the model could overall be more trustworthy, and higher precision so we wouldn’t have to worry so much about false alarms.

We decided to run another model, and we chose a random forest to see if we could improve some of these scores to better understand the relationship between variables in our dataset and a potential cyberattack. Unlike the linear regression model, a random forest allows us to explore non-linear relationships because it builds a decision tree with splits at different thresholds to detect any jumps in variable relationships.

We trained the model using 500 trees and considered 2 features at each split, meaning at each split in the decision tree the model selected 2 available features and chose the best one for the split. This adds randomness and reduces overfitting.

In Random Forests, each tree is trained on a random subset of the data and about one-third of the data is left out from that tree's training set. These left-out observations are called out-of-bag samples. The OOB error rate is calculated by testing each tree on its out-of-bag data, providing a built-in cross-validation to estimate the model's performance.  
The 12.99% OOB error rate means that when testing the model on these out-of-bag samples, it made a wrong prediction 12.99% of the time.  
That also means it was correct 87.01% of the time, which is the estimated accuracy of the model even before testing it on new data.

We ran a Random Forest confusion matrix and found that there were 5248 true negatives, 3050 true positives, 1214 false negatives, and 25 false positives.

5248 TN - cases correctly identified as no attack

3050 TP - real attacks correctly identified

1214 FN - real attacks missed by the model

25 FP - cases incorrectly flagged as attacks

Here are our percentages:

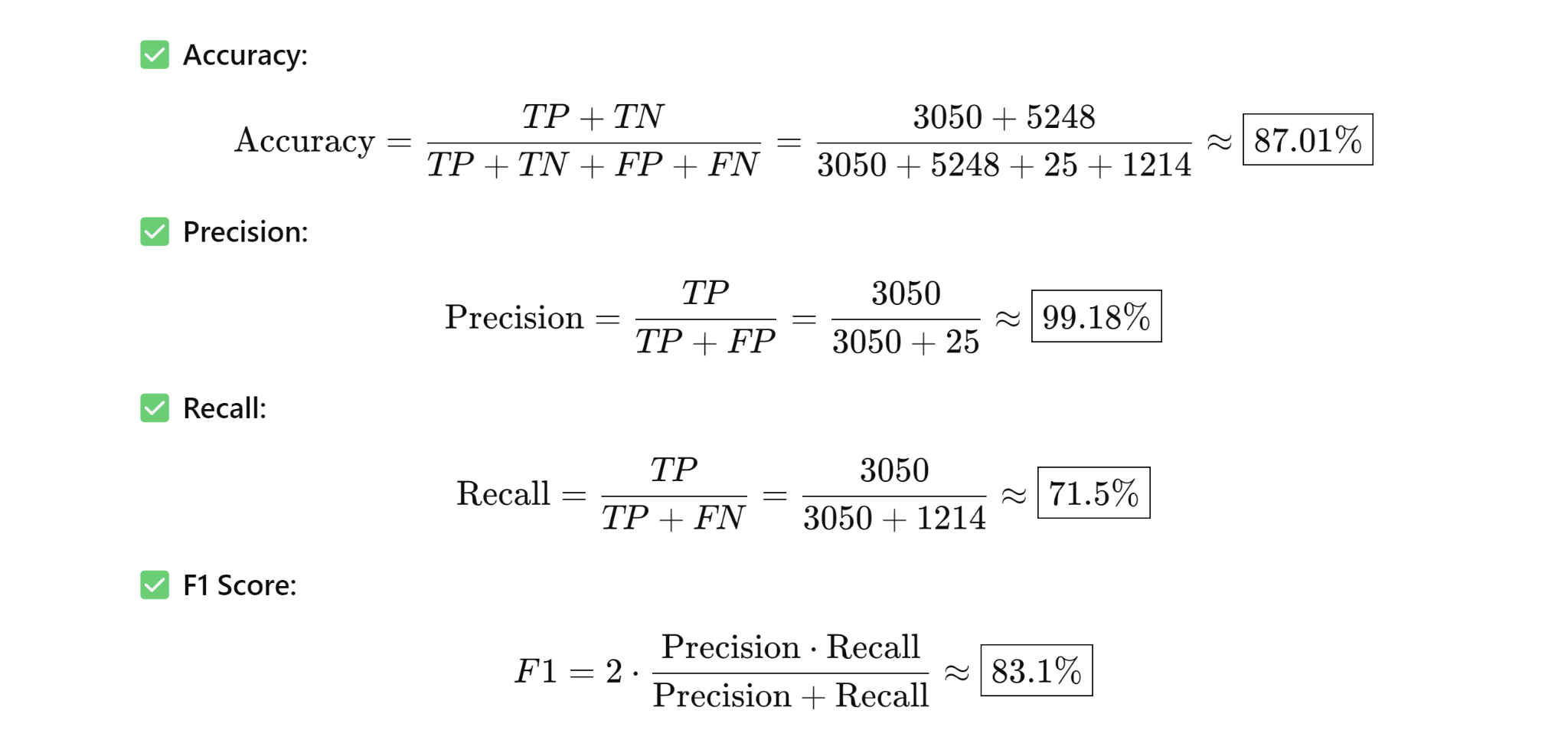
Accuracy 87.01% Model correctly predicts about 87% of cases

Precision 99.18% When it predicts an attack, it’s right 99% of the time

Recall 71.5% The model catches 71% of actual attacks

F1 83.1% Balance between precision & recall

Here are the formulas to find the four values above (Used ChatGPT to help with formulas)



In cybersecurity false negatives, where a real attack goes undetected, are particularly dangerous. They mean that malicious behavior slips past the system undetected, which can lead to breaches, data loss, and severe security consequences. Due to this, we prioritized improving the recall score as it is the percentage of attacks actually detected.

In order to improve the recall score we lowered the default threshold of greater than 0.5 to 0.3 in order, meaning the model would flag an incident even if it was only 30% positive there was an attack. A slightly higher rate of precision, the likelihood of an attack when one is predicted, is something we are willing to risk because a false alarm is much less dangerous than a false negative in cybersecurity. A false positive might waste time, but it is better to be safe than sorry when sensitive data is at stake.

After lowering the threshold to 0.3…

Recall improved to 79%, an 8% increase

Precision dropped slightly but remained at 99%

F1 score increased to 88%, meaning a raised balance between recall and precision

Patterns aren’t always linear in a cybersecurity environment as one variable won’t have a strictly positive relationship with another, so a random forest model was a strategic choice to best demonstrate the relationships between our three selected variables and an attack happening.

Comparison between Logistic regression and Random Forest model:

Accuracy: LR 80% RF 87%

Precision: LR 75% RF 99%

Recall: LR 80% RF 79%

F1: 78% RF 89%

In Conclusion, the Cybersecurity Intrusion dataset, found on Kaggle, allowed us to explore relationships between variables. We chose attack\_detected as our target variable because our ultimate goal was to determine how different log data and characteristics lead to attacks more than others. Our initial thought process was to use a classification tree but we encountered limitations on interpretability and prediction. We were also turned away by the initial tree plot that fit 10,000 objects into two groupings and created quite the eye sore. We then used a logistic regression model to quantify the probability of an attack based on failed logins, login attempts, and ip reputation score. After this we decided to run a random forest model to dive into non-linear relationships and patterns in order to better understand how one of our three variables affects the odds of an attack. Our findings showed that failed login attempts had the strongest positive correlation with detected attacks, followed by login attempts and IP reputation score. The logistic model produced good performance metrics with an accuracy of 80%, precision of 75%, recall of 80%, and an F1 score of 78%, suggesting that even a relatively simple model can provide meaningful predictive power. However, a random forest model may offer an advantage in handling real-world cybersecurity threats due to the ability to generalize better and pull more randomized data. The Random Forest model produced strong performance metrics as well with an accuracy of 87%, precision of 99%, recall of 79%, and an F1 score of 89%. This model increased the models accuracy in predicting attacks and had a near perfect precision meaning when an attack was predicted it was almost always right.

Overall, these models provided excellent demonstrations of how we can use datasets to find relationships to better understand real-world problems. Cyberattacks, especially with today's growing reliance on technology, pose a very dangerous threat to companies and individuals as sensitive data can be breached and stolen in an instant. Models like this can help companies identify what specific red flags correlate the highest with attacks, which will allow them to adjust firewall rules and other security protocols to block malicious activity before it can cause damage.