2022-12-07

The Silent Killer: Underlying Factors in Soccer Injuries

Final Report

Name	**Email**
Osman Sultan	osman.sultan@mail.utoronto.ca
Denzel Berko	denzel.berko@mail.utoronto.ca
Pascal Kardjian	pascal.kardjian@mail.utoronto.ca
Adam Tibi	adam.tibi@mail.utoronto.ca

1.0 Problem Statement

Brazil, favourites to win the 2014 FIFA World Cup, saw their prospects diminish following the injuries of star player Neymar. With their subsequent 7-1 loss to Germany, Brazil became a key example of how injuries can alter the outcome of a soccer match and dictate team success. Many predictive models in the realm of soccer make assumptions when it comes to the injury status or proneness of the involved players. The predicted statistical output of a player usually neglects their ability to stay on the field due to the complexity associated with injury risk. This investigation will attempt to address the complex nature of soccer injuries and predict whether players are at risk of injury using various binary classifications models. We also hope to also discover some of the underlying factors that may or may not contribute to soccer injuries. Ultimately, prediction can provide insight for coaching staff and help in deploying appropriate measures to mitigate injury risk, whether it be by altering their workload or through increased physiotherapy. It can even assist management staff in making more informed decisions when signing new players who are injury prone. Further insight on key indicators of injury can help coaches identify player tendencies contributing to their injury proneness. Altogether, we hope these insights will result in less injuries suffered amongst a team's players and enable them to perform at full strength throughout their season.

2.0 Data

Determining whether a player is at risk or not can be represented by a binary outcome, where 1 represents a player being injured and 0 indicates the player is not injured. The output probability between 0 and 1, along with a defined threshold, will tell us if a player is at risk of injury. Given the nature of this problem, we need datasets which have information regarding the current injury status of a player (target) along with other general player information that we believe would be relevant in predicting a player's risk of injury.

2.1 Data Collection

After searching dataset repositories, we were unable to find anything that suited our needs and so the focus shifted towards website scraping. All retrieved data came from two websites: Transfermarkt.com and FBRef.com [1][2]. The former possesses data regarding player injury status and history of past injuries while the latter holds data such as general player information and in-game statistics. Two separate web scrapers were developed in Python for each of the aforementioned websites. The development process was an iterative one— many rows were being dropped in the initial runs of both web scrapers due to missing data. The team did not want to resort to imputing as most of the rows being dropped had null values for over 90% of columns. Imputing these rows would make them almost entirely synthetic. The team also believed the missing data was a result of the logic implemented in the web scraper rather than a lack of information from the websites. Therefore, testing and debugging was done to obtain a more robust scraper. In the end, the team managed to retain nearly 100% of rows between both datasets.

Next, we merged the two datasets that we obtained through our web scraper (Appendix A). To accomplish this, we joined the datasets on the name column and dropped duplicates beforehand (~50 rows across both). Although we initially believed that formatting was consistent throughout both websites, we lost a total of 300 rows in the process. After investigating, we realized that many names were being stored differently across the websites. For example, Tottenham and Richarlison de Andrade were being stored simply as Richarlison on one of the websites. This, and other problems were difficult to automate. Finally, we were left with 1924 rows in our dataset (Appendix A).

Our final dataset comprises key feature variables including age, position, player usage (minutes & appearances), and past injury history for players across top five European leagues using their country coefficient [3]. Additionally, there are 35+ columns pertaining to in-game statistics recorded by FBRef which have also been included in the dataset. The target variable is the current injury status of the player, which is binary. Further, since the output of our predictive model was the probability of being injured, we needed all columns to be either integer or float types. Thus, a variety of data cleansing techniques were used to convert non-numeric columns into something digestible for our chosen models. This included array unpacking, string manipulation and one-hot encoding for categorical data.

2.2 Defining Testing, Training, and Validation Split

To prepare the data for fitting, we created a train-test-validation split of 85-7.5-7.5, respectively. Although a validation set is not always necessary, we needed it for the purpose of model evaluation and preventing overfitting. Through the validation set, we can tune the model's hyperparameters and configurations accordingly. That is, while the model trains on the training set, the model is also being evaluated through the validation set after every iteration or epoch. As for why the training set is large; one of our models, the artificial neural network (ANN), requires a large amount of data to train on. Via trial and error, we found that precision and recall became sufficient with an 85% size training set while also taking measures to ensure no overfitting. This will be discussed further in Section 4.2.

3.0 Exploratory Data Analysis (EDA)

To provide insight into our data and influence modeling decisions, we conducted EDA on both training and testing data. The results of our EDA would help us answer questions relating to model parameters, model performance, model engineering techniques, and further tuning of the dataset.

3.1 Class Imbalance

Upon initial analysis, it can be seen that the training data contained 1154 rows for non-injured players and only 94 rows for injured players (See Figure 1). This showed a major class imbalance between the majority and minority class, being non-injured players and injured players respectively.

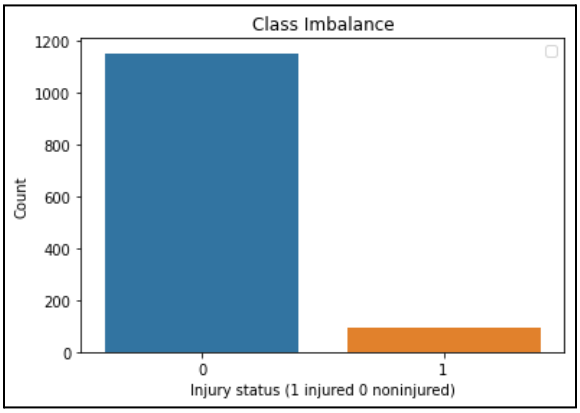

Figure 1. Training set class imbalance

If unaddressed, the negative impacts of the class imbalance could potentially lead to low precision as the models would not have many positive data points to train off. It can also lead to models being significantly better at predicting non-injured players (0) due to the larger amount of data that is provided for this class. To address the class imbalance, we used the SMOTE oversampling technique. Through

SMOTE, we balance a dataset to include more positive samples [4]. Specifically, SMOTE uses a k-nearest neighbor algorithm to create synthetic rows for the minority class by taking samples of the feature space and combining them to make one row. To avoid extreme oversampling, SMOTE was chosen instead of normal random over sampling since it copies existing instances of rows in the minority class. Because of the extremely large imbalance, the dataset would have to be populated with many instances from the initial dataset if this standard technique was used. After SMOTE implementation, 1060 rows were added to the training set for a total of 1154 rows, eliminating the class imbalance issue. The testing set for our models also showed a small amount of rows for injured players. This analysis was motivated upon seeing precision values for a majority of the models during initial testing. The conclusion was that our testing set contained such a small sample of injured players that our models were never given the opportunity to make a positive prediction. To address this, the probability of putting rows for injured players into the training set was increased. As a result, rows for injured players that were initially in the training set were now in the testing set which increased the number of injured player data from 10 to 22. This was done before training the models as the models must be tested on unseen data.

3.2 Need For Model Engineering Techniques

In addition to using EDA to draw insights on the datasets, it was further used to determine the model engineering techniques that can be applied to improve the performance and training of the models. As seen in Figure 2, the initial features scraped into the dataset showed little to no correlation to the target (see Figure B-1, Appendix B for full correlation matrix). The highest correlated feature with the target had a correlation value of 0.3. This is a problem because little insight can be gained in terms of the impact the features have on our target.

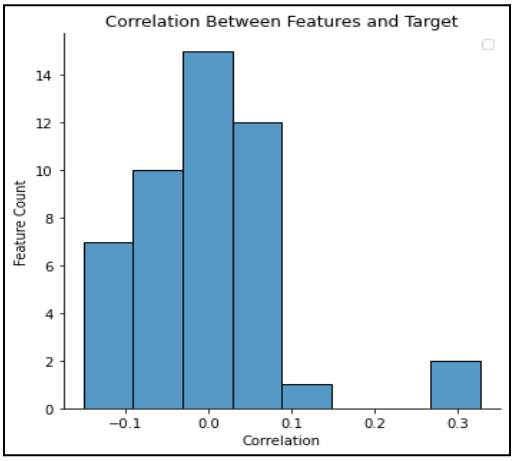

Figure 2: Histogram showing number of features with specific correlation value

This limits the ability to answer the identified problem statement which allows teams to make more informative decisions, as correlation is helpful in showing causal relationships. Another problem with having low correlation of all features is that it provides little insight in determining which features should be kept if feature selection were to be implemented, as it would be best to keep the most informative features. This analysis brought the motive to implement feature engineering in hopes of discovering features that are more correlated with the target.

When implementing models that use gradient descent based algorithms, it is important to take into account the disparity in magnitudes that are shown by the features. One of the classification models in this

project is an ANN (will be discussed in more detail in later sections) which uses a gradient descent algorithm. Essentially, this algorithm finds the optimal point of a cost function which tells us how far off the predictions are from the actual values [5]. The idea is to minimize this function. Having unscaled features increases the amount of time to complete this algorithm which results in slower training. In addition, it can also cause the gradient descent to get stuck at a local optima rather than a global one [6]. Through analyzing the training data it was found that the current features show a large disparity in magnitude (see Figure 3). This brought the motive to implement feature scaling to ensure that all features change in magnitude at the same rate, which results in uniform steps sizes for all features in the algorithm. As a result, the ANN will see improvements in training.

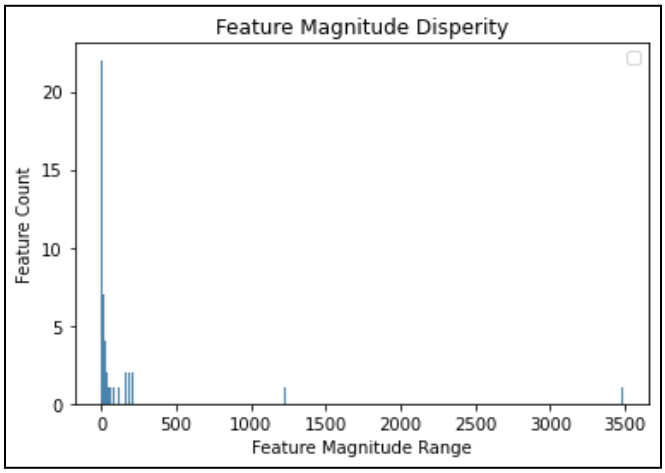

Figure 3. Histogram showing the number of features with a given magnitude range

3.0 Models

Baseline models chosen to initially assess the dataset included logistic regression (LR), a random forest classification model (RF), and an ANN. LR was implemented using lasso (L1) and ridge (L2) regularization in order to improve model accuracy without the need of additional engineering. Regularization allows for the β value's of each feature to be adjusted to combat any present overfitting. Moreover, RF was chosen as it is a powerful classification model due to the ensemble learning (bagging) present. RF automatically accounts for possible flaws in a dataset such as missing values and outliers, and combined with a stable algorithm, helps reduce any overfitting without the need of any additional engineering, making it a good choice for baseline assessment. Additionally, LR and RF are generally simple and fast to implement, with a relatively low amount of parameters. In general, reducing unnecessary complexity when choosing models and their corresponding parameters, allows for models to generalize better on the train set leading to more accurate results when predicting on the test set. Simple models were especially crucial for this investigation due to the relatively small dataset; the use of complex models with highly specified parameters is prone to overfitting when trained on little data. To demonstrate this, an ANN, a complex model with highly specified parameters, was chosen to contrast against LR and RF. Baseline models were largely assessed with default hyperparameters to allow for contrasting against optimized models constructed through model engineering techniques in order to assess the effectiveness of the raw dataset.

3.1 Defining the Threshold

Before implementation of any models, a threshold of 0.4 will be defined to classify the target value. Due to the present class imbalance, it is evident that any model implementation will yield a low precision score compared to recall, indicating that the models are able to predict true negatives well, but will struggle to predict true positives. Therefore, a slightly lowered threshold value will allow for a higher number of true positives with the consequence of a higher number of false negatives. With respect to the problem statement, allowing for soccer players to be incorrectly labeled as at risk of injury is an acceptable outcome as it will only result in increased precaution for the player, likely helping them maintain physical fitness regardless.

3.2 Model Engineering Techniques

Through EDA, it is sufficient to conclude that the raw train-test-validation split of the dataset contains various flaws resulting in poor overall baseline performance. Combined with a relatively small initial dataset of approximately 2000 rows, it is crucial that model engineering techniques are implemented to avoid overfitting and to improve upon the baseline model performance. Before adjusting any feature columns within the dataset, outliers present within the train set were identified and removed using an isolation forest (see Appendix C Figure C-1). The removal of outliers was necessary as these data points can have an increased negative effect on model performance when using a smaller dataset. In addition to the aforementioned SMOTE, three techniques including feature scaling engineering, and selection, were used in conjunction to engineer optimized conditions for each of the three classification models used.

3.2.1 Feature Scaling

To implement feature scaling a specific scaling technique known as normalization was used. This technique rescales the data set such that all values fall in the range of 0 and 1. This technique was used over standardization (another common scaling technique) because a majority of the data does not follow a gaussian distribution, thus, normalization is preferred (see Figure B-2, Appendix B). Using the preprocessing package from Scikit-learn, the normalized data sets for the train, test and validation set were initialized (see Figure C-2 and C-3, Appendix C). After applying the normalized datasets to the neural network, loss scores (summation of training error), accuracy and training time were all improved (see Figure 4).

A.N.N Model	Loss	Accuracy	Train Time (s)
Un-scaled Data	74.77	0.4793	261.9
Scaled Data	1.3878	0.8017	120.3

Figure 4. Scaled vs unscaled neural network

3.2.2 Feature Engineering & Selection

Due to the arbitrary nature of feature engineering an algorithm was implemented that iterated through all features and evaluated the product of them with each other, ranking products based on F-scores which produces a classifications model accuracy. Additionally, domain knowledge of soccer was used by the team to formulate the final three engineered features to be added to the dataset. An increase in correlation was seen in all engineered features (see Appendix C, Figure C-4 and C-5). Finally, feature selection was implemented to address overfitting due to a wide range of features. By removing redundant and lowly correlated features, 15 final features were selected.(see Appendix C, Figure C-6).

3.2.3 Hyperparameter Tuning

With the dataset now optimized, the hyperparameters for LR and RF were tuned using grid search and random search respectively; random search was chosen for RF due to computation time limitations. This step was crucial for the machine learning models as it specialized hyperparameters for the specific use case, implementing cross validation to reduce overfitting and improving prediction accuracy. The final hyperparameters are listed below:

- LR-L1: random_state = 0, C = 10.0, class_weight = 'balanced', solver = "saga"
- LR-L2: random_state = 0, C = 1.0, class_weight = 'balanced', solver = "newton-cg"
- RF: n_estimators = 40, min_samples_split = 6, min_samples_leaf = 5, max_features = 'sqrt', max_depth = 10, criterion = 'entropy', class_weight='balanced'
- ANN: epochs = 250, batch_size = 32, optimizer = ‘adam’, loss = 'binary_crossentropy', activation = relu, 3 hidden layers

4.0 Results

This section will explore initial baseline performance against optimized engineered performance of the chosen models and the results that are derived from this comparison.

4.1 Quantitative Analysis

To quantify the predictive power of our model, we used precision, recall, and score (note that score, in this case, is precision + recall / 2). The reason we chose precision as a scoring metric rather than FPR/TPR is because in our original dataset, there was a large number of negative (0 class) samples. When the number of negative class samples is large, the FPR increases very slowly since the true negatives would be very high. Precision, however, places a larger emphasis on the negative class, since it measures the probability of true positives. Now, one may raise the argument that this issue is irrelevant due to the synthetic oversampling we used, however, it is important to keep in mind that these samples are merely synthesized. SMOTE still uses original data in the negative class, and creates new data points near the existing ones. And so, although the points are unique, they are still very similar to the original ones. And thus, the point still stands that we would rather use metrics that assess the portion of our data that was unique, untouched, and plentiful. Lastly, we had also used ROC AUC to assess the model's ability to differ between a true negative (TN) and a true positive (TP). Below, Table 1 records all the baseline results.

Table 1. Baseline Results

Model Name	**Precision**	**Recall**	**Score**	**ROC AUC**
LR-L2	0.183	0.657	0.420	0.675
LR-L1	0.157	0.486	0.322	0.607
RF	0.000	0.000	0.000	0.500
ANN	0.143	0.029	0.086	0.500

Overall, most baseline models performed poorly; with RF having the worst precision, recall, score, and ROC AUC along with the ANN. Having an ROC AUC of 0.5 indicates that the model’s ability to

distinguish between TP and FP is no better than random guessing. However, after engineering our model's with the aforementioned techniques, we obtain the following results (Table 2):

Table 2. Model Engineered Results

Model Name	Precision	Recall	Score	ROC AUC
LR-L2	0.181	0.851	0.519	0.726
LRL1	0.182	0.571	0.377	0.652
RF	0.184	0.400	0.292	0.608
ANN	0.161	0.286	0.224	0.608

Almost every metric improved, with RF showing improvements of 15-50% across all metrics. We note that LR-L2 performs the best on almost all metrics with a notably high recall of 0.851. One explanation of this is the fact that LR-L2 has the lowest FNR of 23% out of all the models.

4.2 Validation

To assess the fit of our models, we plotted learning curves (Figure 5) for all four engineered models. A loss curve graphs the model's performance with respect to a metric over the experience. We plot both the training and validation set; the training plot gives an indicator of how well the model is "learning" while the validation set gives us an idea of how the model is generalizing.

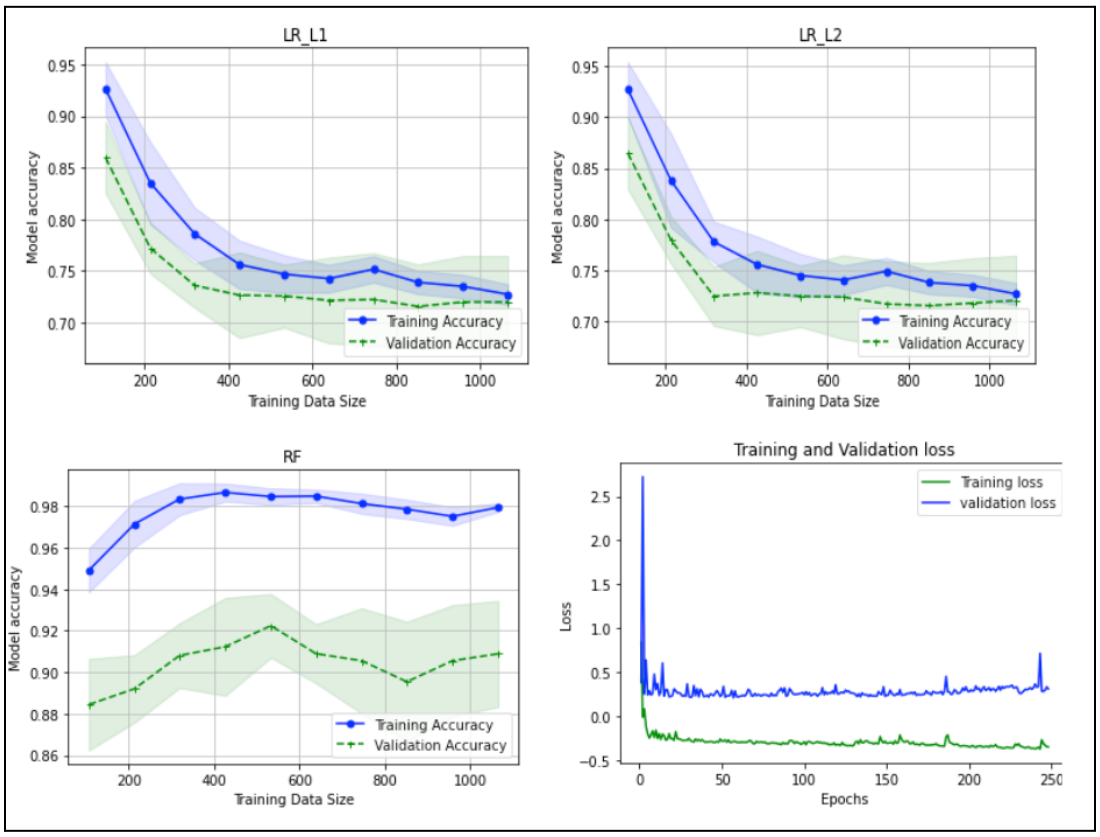

Figure 5. Training Curve of LR-L1, LR-L2, RF, and ANN

For LR-L1, LR-L2, and the ANN we see that the training accuracy begins to converge with experience indicating a relatively good fit. However, we note the small dip near the end of training for LR-L1 and LR-L2 implying the model is capable of further learning and that the training process was halted prematurely. For the RF, we see the model demonstrate clear signs of overfit due to the training accuracy plateauing and staying near 1. Further, the validation accuracy increasing to a point then decreasing is another sign of overfitting. This is because it indicates the accuracy's initial improvement was merely by chance, since it was not able to generalize with more data.

4.3 Discussion of Results

Despite poor performance across the board, our best model, LR-L2, yields a recall of 0.851 and AUC of 0.726. We are satisfied with this result when considering the goal of our investigation. As previously mentioned, we want to provide insight to coaching staff about potential players who may be at risk of injury to help them take the necessary steps and ultimately, reduce the number of injuries suffered across their squad. Incorrectly diagnosing a player as being at risk of injury will only result in increased precaution. In contrast, if we incorrectly classified a player to non-injury when they are prone to injury, it would put them at great risk. Therefore, LR-L2's high recall and adequate ability to distinguish positives from negatives makes it a suitable candidate given our problem statement.

Part of the intention of this model was to help teams make informative decisions when it comes to limiting injuries. However, the obtained and engineered features did not show much correlation at all with the target. Although this limits the ability to show what features contribute to player injuries, the model can be used to provide teams insight into the features that do not need to be monitored when trying to limit player injuries. The model is able to show that the included features are not good indicators of what truly puts a player at risk of injury, thus teams will not have to worry about limiting these features when considering the injury risk of their players.

5.0 Conclusion and Future Steps

The intention of this study was to gain some insight into the ambiguity surrounding soccer injuries and build predictive models capable of identifying whether a player is at risk of injury. EDA uncovered early insights into weak feature correlation and several flaws in the dataset such as class imbalances and disparities in feature magnitude. This trend was showcased in poor baseline performance of LR, RF, and ANN models. The implementation of model engineering techniques resulted in improved performance across the board. The LR-L2 model specifically produced satisfactory results given the context of our study. Despite this, the extremely low precision score still leaves much to be desired. The results obtained showcase the weaknesses associated with the premise of the study. Currently, data collection only relies on Europe's top 5 professional soccer leagues, which severely limits the size of the dataset. Going forward, the investigation should also include data from a range of global soccer leagues. In addition to the small sample size, the dataset also contained many features with very low correlation to the target feature. More relevant features such as biometrics, training regime, diet and detailed medical history may be better suited to accurately predict a player's risk of injury. On the other hand, the models did shed light on factors that are not good indicators of injury. In the end, we were able to address the initial problem statement by obtaining a satisfactory LR-L2 model with a recall of 0.851 and gained insight into factors which do not contribute to injuries.

References

[1] "Football transfers, rumours, market values and news," *Transfermarkt*. [Online]. Available: https://www.transfermarkt.com/. [Accessed: 07-Dec-2022].

[2] "Football statistics and history," *FBref.com*. [Online]. Available: https://fbref.com/en/. [Accessed: 07-Dec-2022].

[3] UEFA.com, "Country coefficients: UEFA coefficients," *UEFA.com*. [Online]. Available: https://www.uefa.com/nationalassociations/uefarankings/country/#/yr/2023. [Accessed: 07-Dec-2022].

[4] Likebupt, "Smote - Azure Machine Learning," *Azure Machine Learning | Microsoft Learn*. [Online]. Available: https://learn.microsoft.com/en-us/azure/machine-learning/component-reference/smote. [Accessed: 07-Dec-2022].

[5] By: IBM Cloud Education, "What is gradient descent?," *IBM*. [Online]. Available: https://www.ibm.com/cloud/learn/gradient-descent#:~:text=Gradient%20descent%20is%20an%20optimization,each%20iteration%20of%20parameter%20updates. [Accessed: 07-Dec-2022].

[6] R. Vashisht, "Machine learning: When to perform a feature scaling?," *atoti*, 10-Oct-2022. [Online]. Available: https://www.atoti.io/articles/when-to-perform-a-feature-scaling/#:~:text=Having%20features%20on%20a%20similar,getting%20stuck%20in%20local%20optima. [Accessed: 07-Dec-2022].

Appendix A

Please find the link to our web scraper here: Link

Please find the download link to our final dataset here: Link

Appendix B: Visualizations and Plots

The following appendix holds any plots and visualizations of data. Plots show a range of detail from results to insights from data analysis.

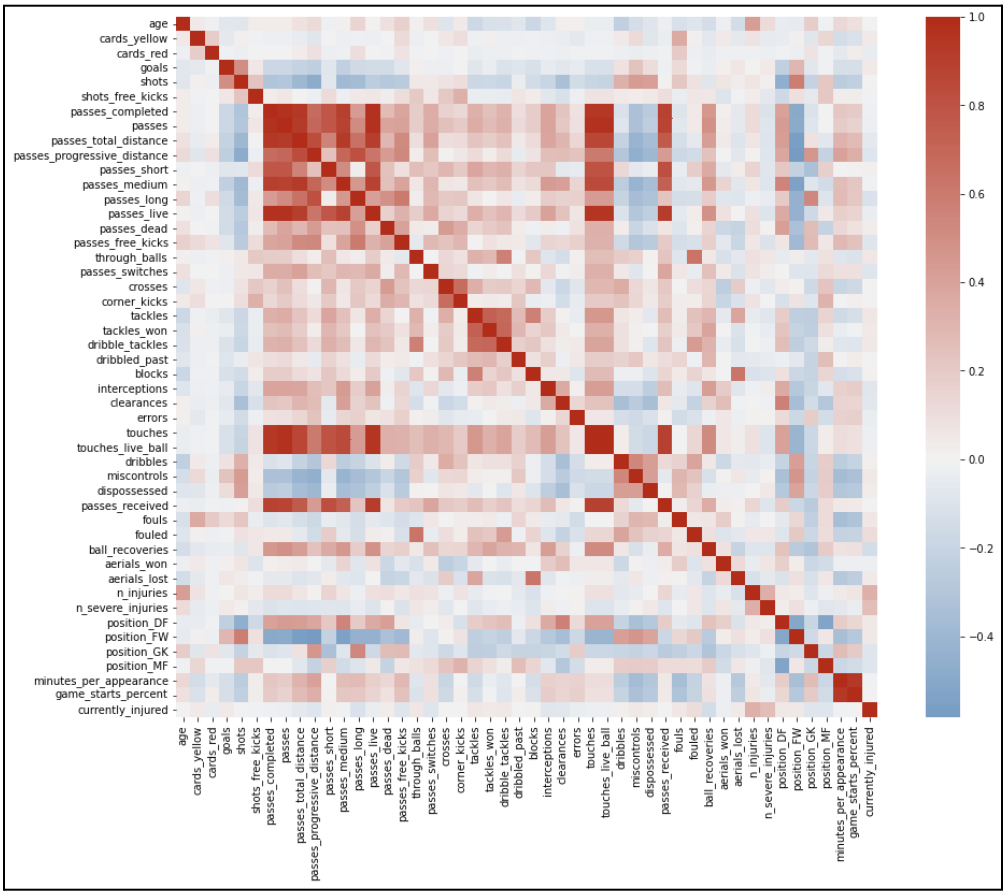

Figure B-1: Heatmat showing correlation of features

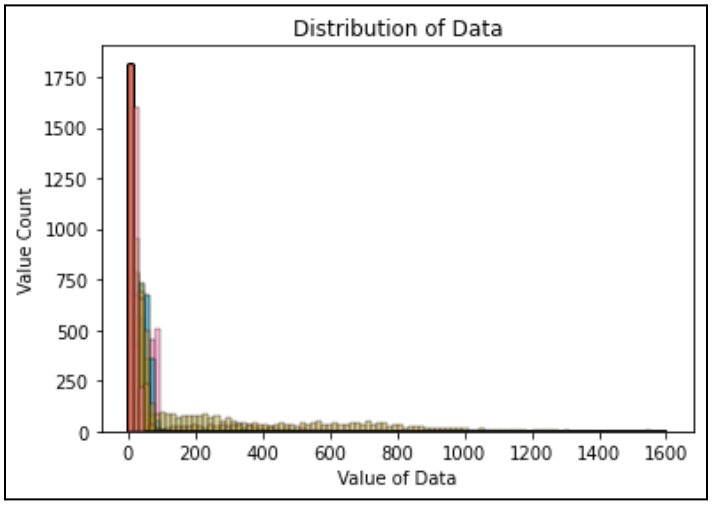

Figure B-2: Histogram showing distribution of dataset

Appendix C: Code and Implementations

The following appendix holds any code or implementations of models, functions and methods. The following code was used to make improvements to the models, or to implement new features/ functions.

```
from sklearn.ensemble import IsolationForest

#create test and train sets
properties = list(df_injury.columns.values)
properties.remove('currently_injured')
X = df_injury[properties]
y = df_injury['currently_injured']


isf = IsolationForest(n_jobs=-1, random_state=1)
isf.fit(X, y)
preds = isf.predict(X)

X['outlier'] = preds
X = X.drop(X[X['outlier'] == -1].index)
X = X.drop('outlier', axis=1)

y = pd.DataFrame(y)
y['outlier'] = preds
y = y.drop(y[y['outlier'] == -1].index)
y = y.drop('outlier', axis=1)
y = y['currently_injured'].squeeze()
```

Figure C-1. Isolation Forest implementation to remove outliers

```
#normalize X train set
normalized_X_train = preprocessing.normalize(X_train)
#normalize X validation set
normalized_X_val = preprocessing.normalize(X_val)
#normalize X test set
normalized_X_test = preprocessing.normalize(X_test)
```

Figure C-2. Code block showing the initialization of normalized data for train, test and validation sets

```
#Train Neural network using unnormalized data
unnormal = nn.fit(X_train, y_train, epochs=(250), batch_size=32, validation_data=(X_val, y_val), verbose=0)

#Train Neural network using normalized data
normal = nn.fit(normalized_X_train, y_train, epochs=(250), batch_size=32, validation_data=(normalized_X_val, y_val), verbose=0)

#Evaluted both neural networks
unnormal_train_score = nn.evaluate(X_test, y_test)
normal_train_score = nn.evaluate(normalized_X_test, y_test)
```

Figure C-3: Code block showing the implementation and evaluation of normalized and unnormalized neural networks

```
# from sklearn.feature_selection import f_classif
# Create a list of F-values for the existing features
feature_F_scores, _ = f_classif(X_train, y_train)

# Iterate through each combination of features
for f1_index, f1 in enumerate(X_train.columns):
  for f2_index, f2 in enumerate(X_train.columns[f1_index + 1:]):
    # Multiply the two features to create a new feature
    new_feature = X_train[[f1]].multiply(X_train[f2], axis=0)
    # Evaluate F-value of new feature
    F_Score_new, p_value_new = f_classif(new_feature, y_train)
    # Evaluate the relative improvement of the new feature
    F_score_improvement = F_Score_new[0] / max(feature_F_scores[[f1_index, f2_index]])
    # Print out features that is sufficiently improved
    if F_score_improvement >= 1.5 and F_Score_new[0] >= 75 and p_value_new < 0.05:
        '''Note that F_score_improvement >= 1.5 and F_Score_new[0] >= 75 is
         relatively arbitrary, and that other values could be used.'''
        print(f'{f1} + {f2} has an F-score of {F_Score_new[0]:.2f}')
        print(f'\tBetter by a factor of {F_score_improvement:.2f} over features in isolation')
        print(f'\tThe result is significant (p = {p_value_new})')
```

Figure C-4. Algorithm used in feature engineering

```
def new_feature_combos(X):
    X_new = X.copy()

    # Correlation of 0.3
    X_new['n_injuries * age'] = X_new.n_injuries * X_new.age

    # Correlation of 0.28
    X_new['n_injuries * touches'] = X_new.n_injuries * X_new.touches

    # Correlation of 0.29
    X_new['n_injuries * miscontrols'] = X_new.n_injuries * X_new.miscontrols

    return X_new

X_train = new_feature_combos(X_train)
X_test = new_feature_combos(X_test)
X_val = new_feature_combos(X_val)
```

Figure C-5. Engineered features and implementation in dataset

```
Index(['shots_free_kicks', 'passes_long', 'crosses', 'corner_kicks',
       'tackles_won', 'dribbled_past', 'fouls', 'fouled', 'n_injuries',
       'n_severe_injuries', 'position_GK', 'position_MF', 'n_injuries * age',
       'n_injuries * touches', 'n_injuries * miscontrols'],
      dtype='object')
```

Figure C-6. List of top 15 features chosen from feature selection