BRAC UNIVERSITY

Inspiring Excellence

CSE422: Artificial Intelligence

Spring 2024

Title: Impact of Player Injuries on Team Performance

Student Information:

Name: MD Adnan Hossain Sadi

ID: 22299314

Section: 03

Name: Sarahat Noor Sakib

ID: 19101454

Section: 03

Submitted to:

MD. Asif Haider & Mahjabin Chowdhury

Department of Computer Science & Engineering (CSE)

Table of contents:

1. Introduction:

This project aims to analyze the performance of football teams after the return of a key player from injury. In competitive football, the absence of critical players due to injuries can significantly impact team dynamics, strategies, and overall performance. When these players return, their reintegration into the team can either improve or disrupt team cohesion. The project investigates how a team's performance changes in the matches immediately following a player's recovery.

Football teams often rely heavily on key players, and their absence due to injury can affect results. While it's generally assumed that a returning player strengthens the team, this may not always be the case. Factors such as fitness levels, match sharpness, team chemistry, and tactical adjustments can all influence outcomes. The challenge lies in quantifying and understanding these effects to determine whether teams experience performance improvement, stagnation, or decline after a player's return.

The motivation for this project stems from the critical role injuries play in football. Coaches, analysts, and fans often debate the impact of returning players, but these discussions are rarely backed by systematic analysis. By providing data-driven insights, this project aims to support teams in decision-making, such as the timing of a player’s return or adjustments to tactics. Additionally, it seeks to deepen our understanding of how individual contributions shape team performance, offering valuable perspectives to players, coaches, and the broader football community.

2. Dataset description:

- Source: Kaggle

  Link-

  https://www.kaggle.com/datasets/amritbiswas007/player-injuries-and-team-performance-dataset/data

- **Dataset Description:**

  There are in total 42 features in our dataset.

```
print(len(data.columns))
✓ [4] < 10 ms
42
```

The target feature of our dataset is “Match1_after_injury_Result” which has only three unique values (lose, win & draw). So, the data of the target feature is categorical and the problem we are trying to solve is a classification problem.

There are 656 data points in the dataset.

```
print(len(data))
✓ [5] < 10 ms
656
```

Our dataset has both quantitative and categorical features.

Below is a breakdown of potential features for this project:

Quantitative Features:

These are numerical values that can be measured or calculated.

Examples include: ‘Age’, ‘FIFA rating’, ‘Match1_before_injury_GD’ etc.

Categorical Features:

These are non-numerical variables that represent distinct categories or groups.

Examples include: ‘Position’, ‘Match1_before_injury_Result’ etc.

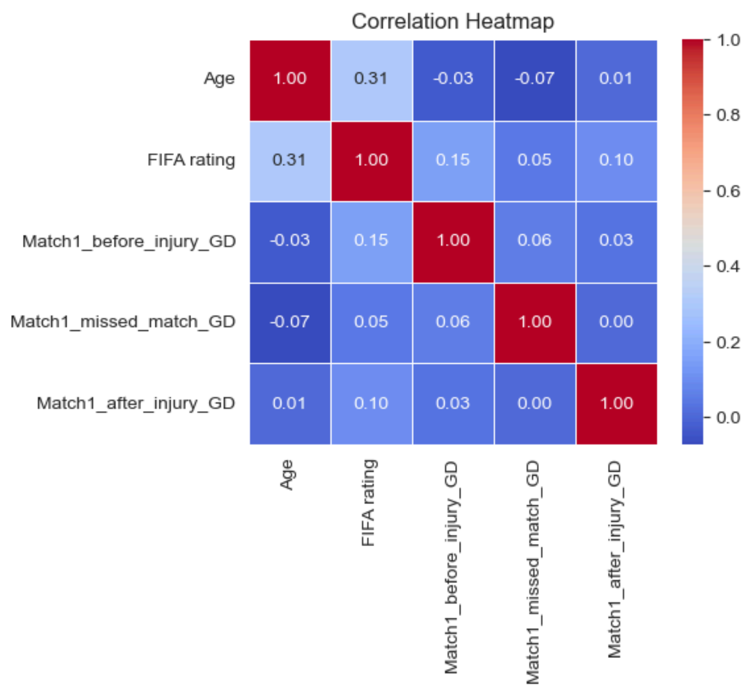
The following heatmap shows the correlation between various features of our dataset. Diagonal elements are all 1, as each variable is perfectly correlated with itself. Age vs. FIFA rating: Correlation of 0.31, indicating a weak positive relationship. FIFA rating vs. Match1_before_injury_GD: Correlation of 0.15, suggesting a weak positive relationship. Match1_missed_match_GD vs. other variables: Mostly close to 0, indicating very weak or no correlation. Match1_after_injury_GD vs. Match1_before_injury_GD: A correlation of 0.03, indicating negligible relationship. We can see that most of the features of our dataset are not strongly correlated. The only obvious correlation we can observe is between ‘Age’ and ‘FIFA rating’ which will not help us that much as none of these are the target feature. This lack of correlation might result in poor accuracy when we train our machine learning model with these dataset.

```python
input_features = ['Age', 'FIFA rating', 'Match1_before_injury_GD', 'Match1_missed_match_GD']
output_features = ['Match1_after_injury_GD']
selected_features = input_features + output_features
correlation_matrix = data[selected_features].corr()
```

✓ [51] < 10 ms

```python
plt.figure(figsize=(5, 4))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm', fmt='.2f', linewidths=0.5)
plt.title('Correlation Heatmap')
plt.show()
```

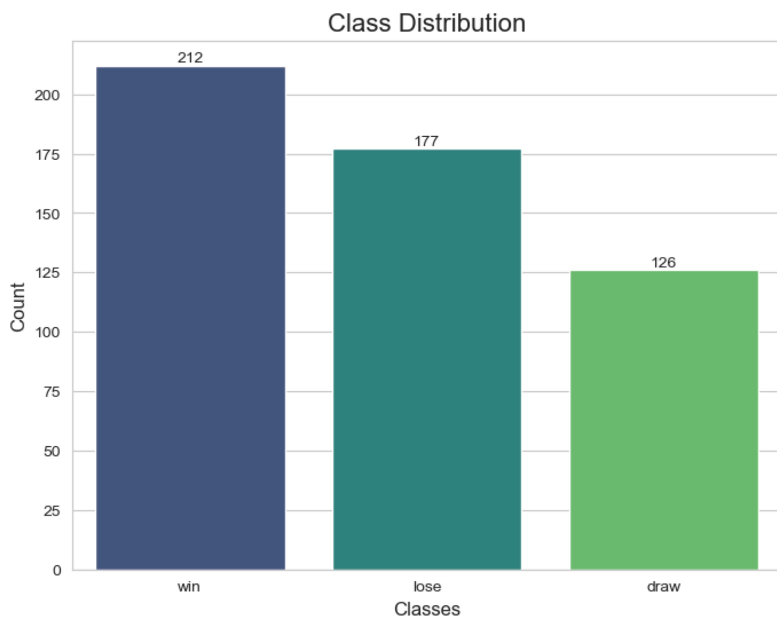
✓ [52] 119ms



- **Imbalance Dataset:** The dataset we used for this project is an imbalance dataset as the target feature does not have equal instances of all the unique classes.

```python
target_column = 'Match1_after_injury_Result'

class_counts = data[target_column].value_counts()

plt.figure(figsize=(8, 6))
sns.barplot(x=class_counts.index, y=class_counts.values, palette='viridis', hue=class_counts.index, legend=False)

plt.xlabel('Classes', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Class Distribution', fontsize=16)

for i, count in enumerate(class_counts.values):
    plt.text(i, count, str(count), ha='center', va='bottom', fontsize=10)

plt.show()
```

✓ [1502] 75ms



3. Data pre-processing:

- **Null values:**

There are null values in our chosen dataset.

```python
data.isnull().sum()
```

[142]

```
Name                                    0
Team Name                               0
Position                                0
Age                                     0
Season                                  0
FIFA rating                             0
Injury                                  0
Date of Injury                          0
Date of return                          0
Match1_before_injury_Result            65
Match1_before_injury_Opposition        65
Match1_before_injury_GD                65
Match1_before_injury_Player_rating     67
Match2_before_injury_Result           101
Match2_before_injury_Opposition       101
Match2_before_injury_GD               101
Match2_before_injury_Player_rating    102
Match3_before_injury_Result           157
Match3_before_injury_Opposition       157
Match3_before_injury_GD               157
```

To remedy this we dropped the columns that have too many null values along with the features that we do not want to work with in our model training.

```python
print("shape of dataframe before dropping:", data.shape)

data = data.drop(['Name', 'Match2_after_injury_Result', 'Match2_after_injury_Opposition', 'Match2_after_injury_GD',
                  'Match2_after_injury_Player_rating', 'Match3_after_injury_Result', 'Match3_after_injury_Opposition',
                  'Match3_after_injury_GD', 'Match3_after_injury_Player_rating', 'Match3_missed_match_Result',
                  'Match3_missed_match_Opposition', 'Match3_missed_match_GD', 'Match1_after_injury_GD',
                  'Match1_after_injury_Player_rating'], axis=1)
print("shape of dataframe after dropping:", data.shape)
```

[143]

```
shape of dataframe before dropping: (656, 42)
shape of dataframe after dropping: (656, 28)
```

We also dropped the rows where the target value which we are trying to predict is null.

```
print("shape of dataframe before dropping:", data.shape)

target_column = 'Match1_after_injury_Result'

data = data.dropna(subset=[target_column])

print("shape of dataframe after dropping:", data.shape)
```

✓ [11] < 10 ms

```
shape of dataframe before dropping: (656, 28)
shape of dataframe after dropping: (515, 28)
```

- **Categorical values:**

  Machine Learning models can only interpret numerical or qualitative data. However, our dataset has many features that are categorical.

```python
data.info()
```

✓ [5] < 10 ms

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 656 entries, 0 to 655
Data columns (total 42 columns):
 #   Column                             Non-Null Count  Dtype
---  ------                             --------------  -----
 0   Name                               656 non-null    object
 1   Team Name                          656 non-null    object
 2   Position                           656 non-null    object
 3   Age                                656 non-null    int64
 4   Season                             656 non-null    object
 5   FIFA rating                        656 non-null    int64
 6   Injury                             656 non-null    object
 7   Date of Injury                     656 non-null    object
 8   Date of return                     656 non-null    object
 9   Match1_before_injury_Result        591 non-null    object
 10  Match1_before_injury_Opposition    591 non-null    object
 11  Match1_before_injury_GD            591 non-null    float64
 12  Match1_before_injury_Player_rating 589 non-null    object
 13  Match2_before_injury_Result        555 non-null    object
 14  Match2_before_injury_Opposition    555 non-null    object
```

For the models to be able to understand these categorical features, we need to encode these data to numeric values.

We used one hot encoding for those features where each classification value is of the same importance.

One Hot Encoding

```python
data = pd.get_dummies(data, columns=['Team Name', 'Position', 'Season', 'Injury', 'Match1_before_injury_Opposition',
                                     'Match2_before_injury_Opposition', 'Match3_before_injury_Opposition',
                                     'Match1_after_injury_Opposition', 'Match1_missed_match_Opposition',
                                     'Match2_missed_match_Opposition'], drop_first=True)
print(data.shape)
data.head()
```

✓ [56] 12ms

```
(515, 311)
```

5 rows × 311 columns

Team Name_Aston Villa	Team Name_Brentford	Team Name_Burnley	Team Name_Everton	Team Name_Man United	Team Name_Newcastle	Team Name_Tottenham	Position_Center Back
False	False	False	False	False	True	False	True
False	False	False	False	False	True	False	True
False	False	False	False	False	True	False	True
False	False	False	False	False	True	False	False
False	False	False	False	False	True	False	False

We used label encoding for the features where the categorical values hold some significance. For example winning is better than a draw or a loss.

Label Encoding

```
data['Match1_before_injury_Result'].unique()
```

✓ [57] < 10 ms

```
array(['draw', 'lose', 'win', nan], dtype=object)
```

```
mappings = {'Match1_before_injury_Result': {'draw': 1, 'lose': 0, 'win': 2},
            'Match2_before_injury_Result': {'draw': 1, 'lose': 0, 'win': 2},
            'Match3_before_injury_Result': {'draw': 1, 'lose': 0, 'win': 2},
            'Match1_after_injury_Result': {'draw': 1, 'lose': 0, 'win': 2},
            'Match1_missed_match_Result': {'draw': 1, 'lose': 0, 'win': 2},
            'Match2_missed_match_Result': {'draw': 1, 'lose': 0, 'win': 2}}

for column, mapping in mappings.items():
    data[column] = data[column].map(mapping)

data.head()
```

✓ [58] 14ms

5 rows × 311 columns

	Age	FIFA rating	Date of Injury	Date of return	Match1_before_injury_Result	Match1_before_injury_GD	Match1_before_injury_Player_rating	Match2_before_injury_Result
0	26	77	Nov 9, 2019	Jan 13, 2020	1.0	0.0	7.4	2.0
2	28	79	Jan 2, 2020	Jan 17, 2020	0.0	-3.0	4.9	0.0
5	30	75	Dec 9, 2019	Jan 10, 2020	1.0	0.0	6	2.0
7	26	77	Sep 22, 2019	Oct 5, 2019	1.0	0.0	6.7	0.0
8	26	77	Dec 15, 2019	Dec 25, 2019	2.0	2.0	6	2.0

- **Date & Time:**

  The features indicating a date in our datasets are stored as strings. For the machine learning model to understand the data we need to convert that to datetime type data and then to a numeric value. For our dataset we felt that the month holds the most important information. So, we just kept the month value as a number.

```
datetime_columns = ['Date of Injury', 'Date of return']

for col in datetime_columns:
    data[col] = pd.to_datetime(data[col], errors='coerce')
    data[col] = data[col].apply(lambda row: row.month)

data.head()
```

✓ [60] 14ms

5 rows × 311 columns

	Age	FIFA rating	Date of Injury	Date of return
0	26	77	11.0	1.0
2	28	79	1.0	1.0
5	30	75	12.0	1.0
7	26	77	9.0	10.0
8	26	77	12.0	12.0

- **Objects to numeric:**

  There are some features left which are in object data type. We converted those to numeric values as well.

```
data['Match1_before_injury_Player_rating'] = pd.to_numeric(data['Match1_before_injury_Player_rating'], errors='coerce')
data['Match2_before_injury_Player_rating'] = pd.to_numeric(data['Match2_before_injury_Player_rating'], errors='coerce')
data['Match3_before_injury_Player_rating'] = pd.to_numeric(data['Match3_before_injury_Player_rating'], errors='coerce')
✓ [59] < 10 ms
```

4. Feature scaling:

We believe that our particular dataset does not require any scaling because there are no such features in our dataset that have overwhelming variance compared to other features. We experimented with MinMaxScaler and Standard Scaler and found that our models accuracy decreases if we use these scalers.

5. Dataset splitting:

We splitted our dataset into a Train set and Test set where the train set contains 70% of the original dataset. Stratified splitting was used to ensure that each unique categorical target data was adequately present in our train and test set.

```python
from sklearn.model_selection import train_test_split

target_column = 'Match1_after_injury_Result'

x = data.drop(target_column, axis=1).fillna(0)
y = data[target_column]

x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.3, random_state=42, stratify=y)
```

✓ [64] < 10 ms

```python
x_train.info()
```

✓ [65] < 10 ms

```
<class 'pandas.core.frame.DataFrame'>
Index: 360 entries, 44 to 617
Columns: 310 entries, Age to Match2_missed_match_Opposition_Wolves
dtypes: bool(293), float64(15), int64(2)
memory usage: 153.6 KB
```

```python
x_test.info()
```

✓ [66] < 10 ms

```
<class 'pandas.core.frame.DataFrame'>
Index: 155 entries, 559 to 538
Columns: 310 entries, Age to Match2_missed_match_Opposition_Wolves
dtypes: bool(293), float64(15), int64(2)
memory usage: 66.1 KB
```

```python
y_train.info()
```

✓ [67] < 10 ms

```
<class 'pandas.core.series.Series'>
Index: 360 entries, 44 to 617
Series name: Match1_after_injury_Result
Non-Null Count  Dtype
--------------  -----
360 non-null    int64
dtypes: int64(1)
memory usage: 5.6 KB
```

```python
y_test.info()
```

✓ [68] < 10 ms

```
<class 'pandas.core.series.Series'>
Index: 155 entries, 559 to 538
Series name: Match1_after_injury_Result
Non-Null Count  Dtype
--------------  -----
155 non-null    int64
dtypes: int64(1)
memory usage: 2.4 KB
```

6. Model training & testing:

- **Decision Tree:**

  We trained a decision tree model using our train set and tested with the test set and found out that this particular model has 42% accuracy for our dataset.

```
from sklearn.tree import DecisionTreeClassifier

model_decision_tree = DecisionTreeClassifier(random_state=42)
model_decision_tree.fit(x_train, y_train)

y_pred_Decision_Tree = model_decision_tree.predict(x_test)

accuracy_Decision_Tree = model_decision_tree.score(x_test, y_test)
accuracy_Decision_Tree = round(accuracy_Decision_Tree, 2)
print(accuracy_Decision_Tree)
```
✓ [69] 15ms

```
0.42
```

- **Random Forest:**

  We trained a random forest model using our train set and tested with the test set and found out that this particular model has 50% accuracy for our dataset.

```
from sklearn.ensemble import RandomForestClassifier

model_random_forest = RandomForestClassifier(random_state=42)
model_random_forest.fit(x_train, y_train)

y_pred_Random_Forest = model_random_forest.predict(x_test)

accuracy_Random_Forest = model_random_forest.score(x_test, y_test)
accuracy_Random_Forest = round(accuracy_Random_Forest, 2)
print(accuracy_Random_Forest)
```
✓ [70] 112ms

```
0.5
```

- **Hist Gradient Boosting Classifier:**

  We trained a hist gradient boosting classifier model using our train set and tested with the test set and found out that this particular model has 52% accuracy for our dataset.

```python
from sklearn.ensemble import HistGradientBoostingClassifier

model_HistGradientBoostingClassifier = HistGradientBoostingClassifier(random_state=42)
model_HistGradientBoostingClassifier.fit(x_train, y_train)

y_pred_HistGradientBoostingClassifier = model_HistGradientBoostingClassifier.predict(x_test)

accuracy_HistGradientBoostingClassifier = model_HistGradientBoostingClassifier.score(x_test, y_test)
accuracy_HistGradientBoostingClassifier = round(accuracy_HistGradientBoostingClassifier, 2)
print(accuracy_HistGradientBoostingClassifier)
```

✓ [71] 1s 399ms

```
0.52
```

- **KNN:**

  We trained a K nearest neighbor model using our train set and tested with the test set and found out that this particular model has 39% accuracy for our dataset.

```python
from sklearn.neighbors import KNeighborsClassifier

modelKNeighborsClassifier = KNeighborsClassifier(n_neighbors=8)
modelKNeighborsClassifier.fit(x_train, y_train)

y_pred_KNeighborsClassifier = modelKNeighborsClassifier.predict(x_test)
accuracy_KNeighborsClassifier = modelKNeighborsClassifier.score(x_test, y_test)
accuracy_KNeighborsClassifier = round(accuracy_KNeighborsClassifier, 2)
print(accuracy_KNeighborsClassifier)
```

✓ [72] 31ms

```
0.39
```

- **Neural Networks:**

  Finally, we implemented a neural network model using tensorflow and it provided an accuracy of 47%.

```python
num_classes = len(np.unique(y))

if num_classes > 2:
    y_train_NN = tf.keras.utils.to_categorical(y_train, num_classes)
    y_test_NN = tf.keras.utils.to_categorical(y_test, num_classes)

model = Sequential([
    Input(shape=(x_train.shape[1],)),
    Dense(64, activation='relu'),
    Dense(32, activation='relu'),
    Dense(num_classes, activation='softmax' if num_classes > 2 else 'sigmoid')
])
```
✓ [1524] 13ms

```python
model.compile(
    optimizer='adam',
    loss='categorical_crossentropy' if num_classes > 2 else 'binary_crossentropy',
    metrics=['accuracy']
)
```
✓ [1525] < 10 ms

```python
history = model.fit(
    x_train,
    y_train_NN,
    epochs=50,
    batch_size=32,
    validation_split=0.3
)
```
✓ [1526] 3s 314ms

```
Epoch 41/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9792 - loss: 0.2494 - val_accuracy: 0.3853 - val_loss: 1.3981
Epoch 42/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9914 - loss: 0.2427 - val_accuracy: 0.3670 - val_loss: 1.4744
Epoch 43/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9517 - loss: 0.2707 - val_accuracy: 0.4220 - val_loss: 1.4440
Epoch 44/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9684 - loss: 0.2318 - val_accuracy: 0.4037 - val_loss: 1.5071
Epoch 45/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9801 - loss: 0.2236 - val_accuracy: 0.3394 - val_loss: 1.4786
Epoch 46/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9844 - loss: 0.1927 - val_accuracy: 0.4220 - val_loss: 1.4996
Epoch 47/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9927 - loss: 0.1848 - val_accuracy: 0.3945 - val_loss: 1.5122
Epoch 48/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 5ms/step - accuracy: 0.9840 - loss: 0.1924 - val_accuracy: 0.3945 - val_loss: 1.5101
Epoch 49/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9904 - loss: 0.1804 - val_accuracy: 0.4037 - val_loss: 1.5080
Epoch 50/50
8/8 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step - accuracy: 0.9916 - loss: 0.1722 - val_accuracy: 0.4312 - val_loss: 1.5838
```

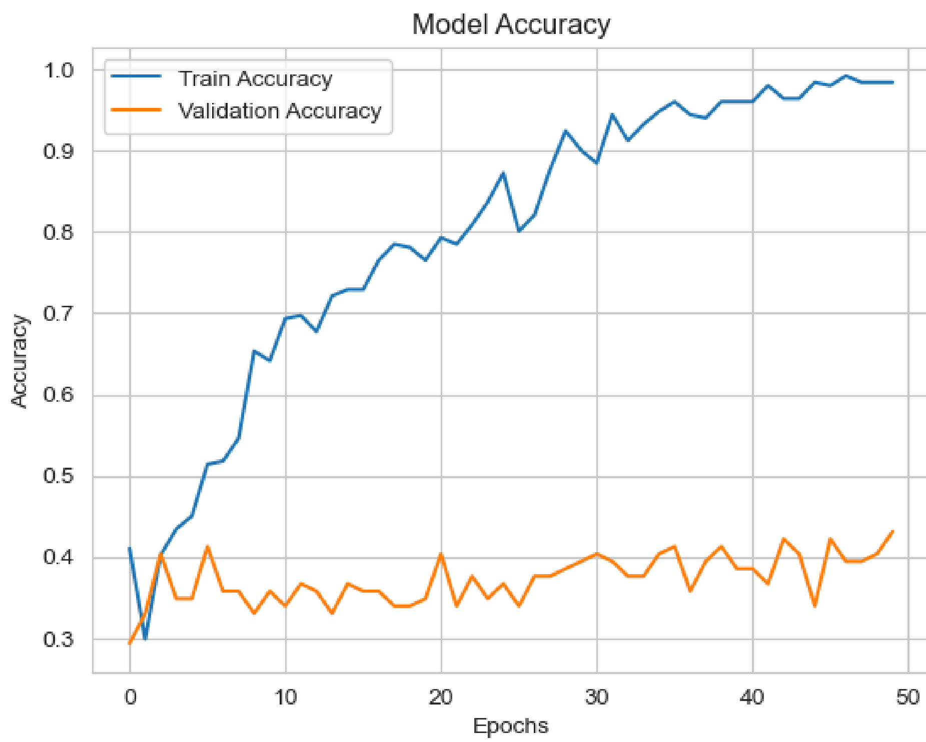
```python
test_loss, test_accuracy_NN = model.evaluate(x_test, y_test_NN)
test_accuracy_NN = round(test_accuracy_NN, 2)
print(f"Test Accuracy: {test_accuracy_NN}")
```
✓ [1527] 50ms

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3ms/step - accuracy: 0.4365 - loss: 1.4133
Test Accuracy: 0.47
```

```python
y_pred_NN = model.predict(x_test)
```
✓ [1528] 69ms

```
5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
```

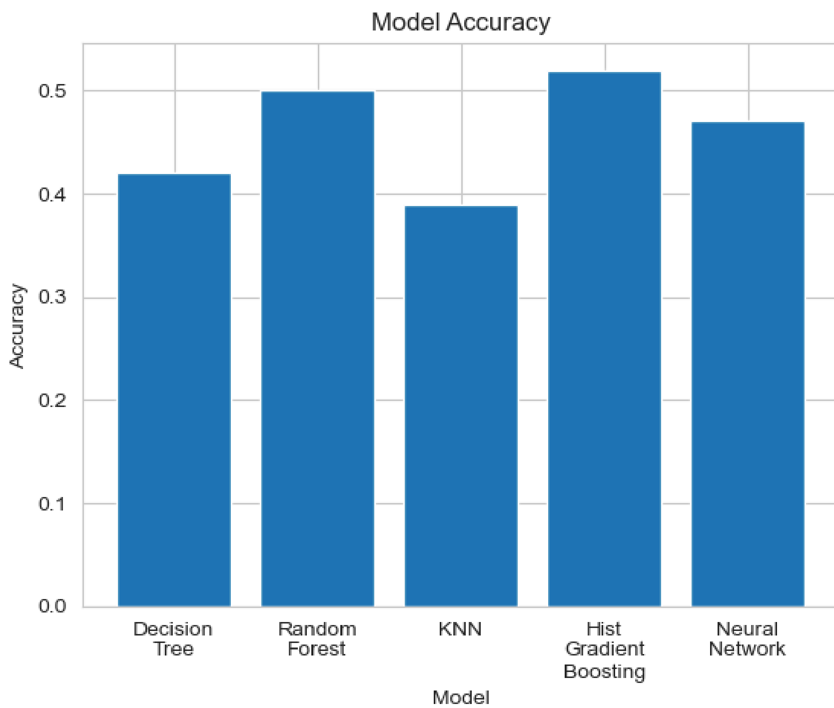
The model loss and accuracy graph for this neural network model was:



7. Model selection/Comparison analysis:

- **Bar chart showcasing prediction accuracy of all models:**

```
models = ['Decision\nTree', 'Random\nForest', 'KNN', 'Hist\nGradient\nBoosting', 'Neural\nNetwork']
accuracies = [accuracy_Decision_Tree, accuracy_Random_Forest, accuracy_KNeighborsClassifier,
              accuracy_HistGradientBoostingClassifier, test_accuracy_NN]

plt.bar(models, accuracies, tick_label=models)
plt.title('Model Accuracy')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.show()
```
✓ [1530] 62ms



- **Precision, recall comparison of each model:**

  (a) Decision Tree:

```
report_decision_tree = classification_report(y_test, y_pred_Decision_Tree)
print(report_decision_tree)
```
✓ [432] < 10 ms

```
              precision    recall  f1-score   support

           0       0.39      0.45      0.42        53
           1       0.35      0.29      0.32        38
           2       0.48      0.47      0.48        64

    accuracy                           0.42       155
   macro avg       0.41      0.40      0.40       155
weighted avg       0.42      0.42      0.42       155
```

(b) Random Forest:

```
report_random_forest = classification_report(y_test, y_pred_Random_Forest)
print(report_random_forest)
```
✓ [433] < 10 ms

```
              precision    recall  f1-score   support

           0       0.49      0.49      0.49        53
           1       0.67      0.16      0.26        38
           2       0.49      0.72      0.59        64

    accuracy                           0.50       155
   macro avg       0.55      0.46      0.44       155
weighted avg       0.54      0.50      0.47       155
```

(c) KNN:

```
report_KNeighborsClassifier = classification_report(y_test, y_pred_KNeighborsClassifier)
print(report_KNeighborsClassifier)
```
✓ [434] < 10 ms

```
              precision    recall  f1-score   support

           0       0.38      0.58      0.46        53
           1       0.35      0.18      0.24        38
           2       0.43      0.36      0.39        64

    accuracy                           0.39       155
   macro avg       0.39      0.38      0.36       155
weighted avg       0.39      0.39      0.38       155
```

(d) Hist Gradient Boosting:

```
report_HistGradientBoostingClassifier = classification_report(y_test, y_pred_HistGradientBoostingClassifier)
print(report_HistGradientBoostingClassifier)
```

✓ [435] < 10 ms

```
              precision    recall  f1-score   support

           0       0.48      0.51      0.50        53
           1       0.48      0.39      0.43        38
           2       0.56      0.59      0.58        64

    accuracy                           0.52       155
   macro avg       0.51      0.50      0.50       155
weighted avg       0.51      0.52      0.51       155
```

(e) Neural Network:

```
y_pred_prob = model.predict(x_test)
if num_classes > 2:
    y_pred_class = np.argmax(y_pred_prob, axis=1)
    y_test_class = np.argmax(y_test_NN, axis=1)
else:
    y_pred_class = (y_pred_prob > 0.5).astype(int).flatten()
    y_test_class = y_test.flatten()

report_NN = classification_report(y_test_class, y_pred_class)
print(report)

```

✓ [1535] 51ms

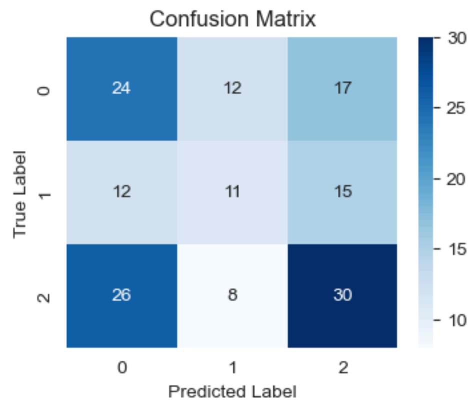
```
1/5 ━━━━━━━━━━━━━━━━━━━━ 0s 9ms/step5/5 ━━━━━━━━━━━━━━━━━━━━ 0s 3m
              precision    recall  f1-score   support

           0       0.42      0.26      0.33        53
           1       0.32      0.26      0.29        38
           2       0.49      0.70      0.58        64

    accuracy                           0.45       155
   macro avg       0.41      0.41      0.40       155
weighted avg       0.43      0.45      0.42       155
```

- **Confusion matrix:**

(a) Decision Tree:

```python
conf_matrix_decision_tree = confusion_matrix(y_test, y_pred_Decision_Tree)

plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_decision_tree, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

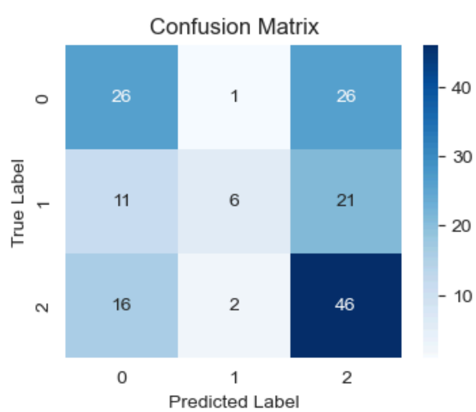
✓ [1536] 75ms



(b) Random Forest:

```python
conf_matrix_random_forest = confusion_matrix(y_test, y_pred_Random_Forest)

plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_random_forest, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

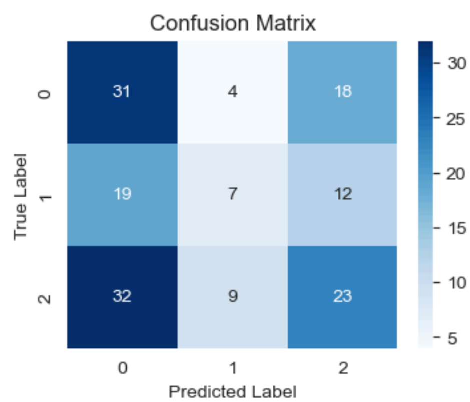
✓ [1537] 71ms



(c) KNN:

```
conf_matrix_KNeighborsClassifier = confusion_matrix(y_test, y_pred_KNeighborsClassifier)

plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_KNeighborsClassifier, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

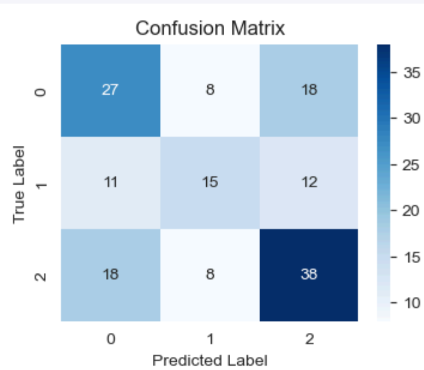
✓ [1538] 75ms



(d) Hist Gradient Boosting:

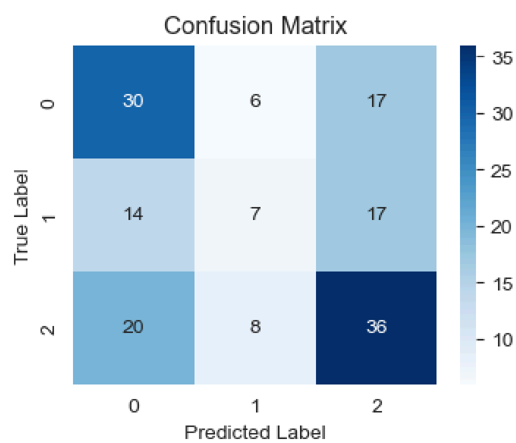
```
conf_matrix_hist_gradient_boosting = confusion_matrix(y_test, y_pred_HistGradientBoostingClassifier)

plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_hist_gradient_boosting, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```

✓ [1539] 75ms



(e) Neural Network:

```python
conf_matrix_NN = confusion_matrix(y_test_class, y_pred_class)

plt.figure(figsize=(4, 3))
sns.heatmap(conf_matrix_NN, annot=True, fmt='d', cmap='Blues', xticklabels=np.unique(y_test),
            yticklabels=np.unique(y_test))
plt.xlabel('Predicted Label')
plt.ylabel('True Label')
plt.title('Confusion Matrix')
plt.show()
```
✓ [1540] 73ms



8. Conclusion:

From the performance metrics of the above models we can conclude that for our particular dataset the Hist Gradient Boosting model was the most successful among the used models. The accuracy of prediction for all the models were not satisfactory. The reason behind this can be the lack of correlation between the features and the target variable which is to be expected as predicting football match results is not an easy task. The result of a football match depends on the performance of each player of a team and the tactics of the coaching staff on the pitch. Predicting a match result with a finite number of attributes with great accuracy is almost an impossible task. This is the thing that makes a football match interesting and enjoyable.