Predicting Injury Risk of NFL Players

Research Question

Can we predict a player's injury risk?

Why this matters?

- **Optimize Roster:** Identify high-risk players before drafting or re-signing.
- **Tailor Practice Loads:** Adjust training intensity to keep athletes healthy.
- **Inform Contract Decisions:** Structure guarantees and incentives around injury likelihood.

Data set

Nfl_data_py

- A python library that includes important information and statistics on NFL data

Injury Data

- Found on Github
- This lists injured players from the 2019-20 NFL season with information about the game and their injury
- https://github.com/sammieerne/NFL-Injury-Analysis/blob/main/Data/game_injury_player_2019_2020_complete_final_update.csv

Checking for Missing Values

```
# Display the shape of the DataFrame
print("Shape of the DataFrame:", df.shape)

# Examine the data types of each column
print("\nData Types of each column:")
print(df.dtypes)

# Identify missing values
missing_values = df.isnull().sum()
missing_percentage = (missing_values / len(df)) * 100
missing_summary = pd.DataFrame({'Missing Values': missing_values, 'Percentage': missing_percentage})
print("\nMissing Values Summary:")
display(missing_summary)

# Display the first 10 rows
print("\nFirst 10 rows of the DataFrame:")
display(df.head(10))
```

Changing ‘contact / non contact’ to 0 and 1

- Changes Contact to 1 and non-Contact to 0
- This allows this to be used for classification and k-fold

```
# Loading the dataset
df = pd.read_excel('/content/football_injury.xlsx')

# Step 1: Cleaning the 'contact / non_contact' column
# Only keep rows that have 'contact' or 'non contact'
df_clean = df[df['Contact/ non-contact'].isin(['contact', 'non contact'])].copy()

# Step 2: Mapping 'contact' -> 1 and 'non contact' -> 0
contact_mapping = {'contact': 1, 'non contact': 0}
df_clean['injury_type'] = df_clean['Contact/ non-contact'].map(contact_mapping)

# Step 3: Dropping the old 'contact / non_contact' column to prevent confusion
df_clean = df_clean.drop(columns=['Contact/ non-contact'])

# Step 4: Print to ensure right results
print(df_clean[['injury_type']].head(10))
```

	injury_type
0	1
1	1
2	1
3	1
5	1
6	1
7	0
8	1
9	1
10	1

Encoding the position of players

- Changes position names into integers for the program to be able to interpret better
- Necessary for machine learning
  - Program can't interpret strings

```
from sklearn.preprocessing import LabelEncoder

# 1. Encoder
le_pos = LabelEncoder()
df_clean['pos_encoded'] = le_pos.fit_transform(df_clean['position'])

# 2. Small table comparing them
print(df_clean[['position', 'pos_encoded']].drop_duplicates().sort_values('pos_encoded'))
```

	position	pos_encoded
0	DB	0
2	DL	1
16	LB	2
8	OL	3
44	QB	4
37	RB	5
922	SPEC	6
29	TE	7
1	WR	8

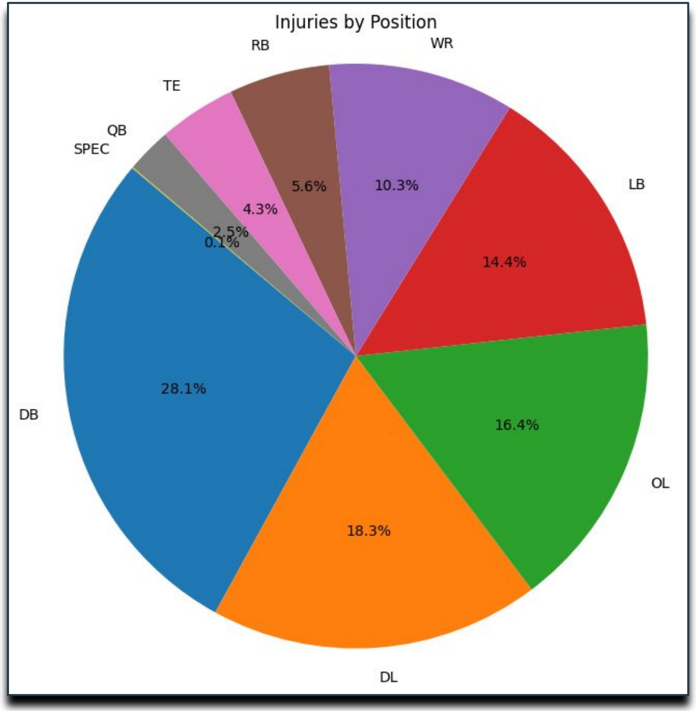
Player Physical & Experience Features

Variables used

- **Position** ⇒ The position the player played when they got injured
- **Height** ⇒ Height (in) of the player
- **Weight** ⇒ Weight (lbs) of the player
- **Contact / non_contact** ⇒ Tells if the player got injured from a contact or non contact play
- **Years_exp** ⇒ Years the player has been in the NFL

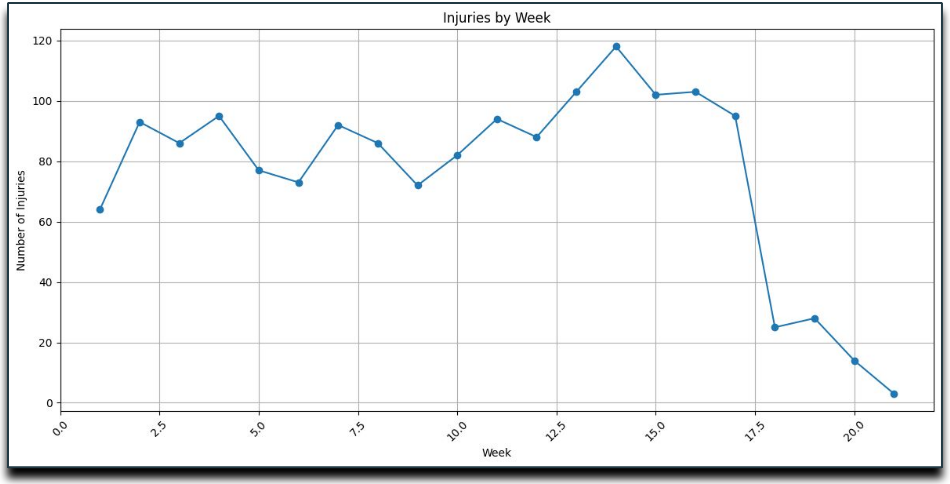
Injuries by Position

- Shows which position gets injured the most
- Defensive backs get injured the most
- Quarterbacks get injured the least
- About a split 50% between Offensive and Defensive players



Injuries by Week

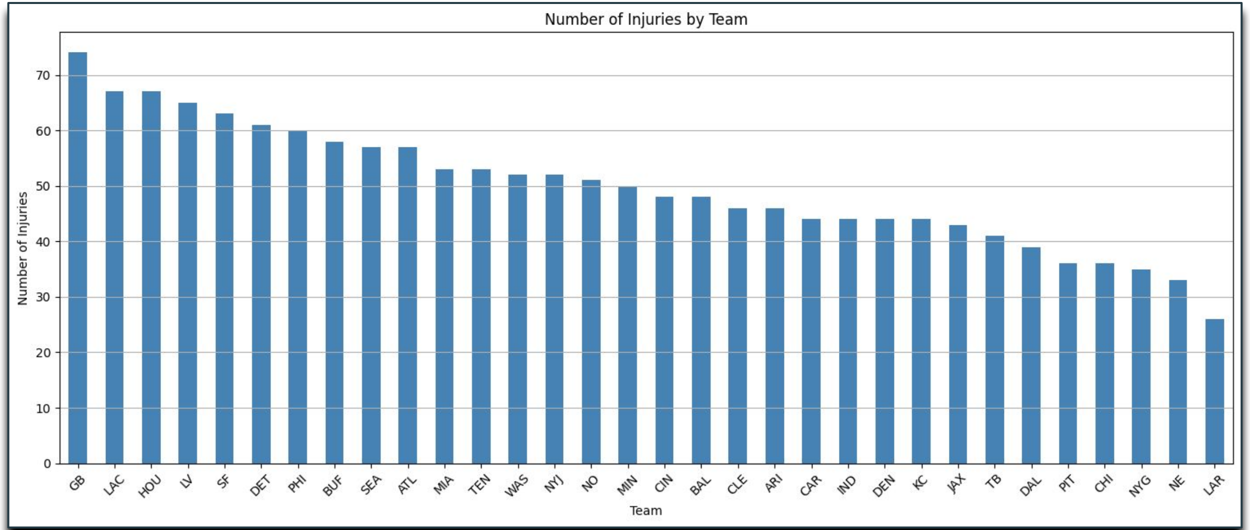
- Injuries are relatively consistent throughout the weeks
- Weeks 13-17 have the highest amount of injuries
  - Week 14 has the most injuries
- After week 17, injuries greatly decrease and gradually decrease afterwards



Numbers of Injuries by Team

- Shows which team had more injuries
- Green Bay Packers had the most injuries
- Los Angeles Rams had the least injuries

* Important to note that 'LAR and LAC' and 'NYG and NYJ' play on the same field



Forest Encoder to Improve Accuracy

- Built a “forest” of decision trees to tell if an injury is “contact” or “non-contact.”
- Turned “position” into numbers and made all inputs use the same scale.
- Tried different numbers of trees, tree depths, and how many samples to split on, and picked the combo that scored highest in cross-validation.
- The model was 86% right overall, but it almost always guessed “contact” and missed almost every “non-contact” case.

```
✅ Best Parameters: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators': 200}

Classification Report:
              precision    recall  f1-score   support

           0       0.33      0.02      0.04        43
           1       0.86      0.99      0.92       265

    accuracy                           0.86       308
   macro avg       0.60      0.51      0.48       308
weighted avg       0.79      0.86      0.80       308
```

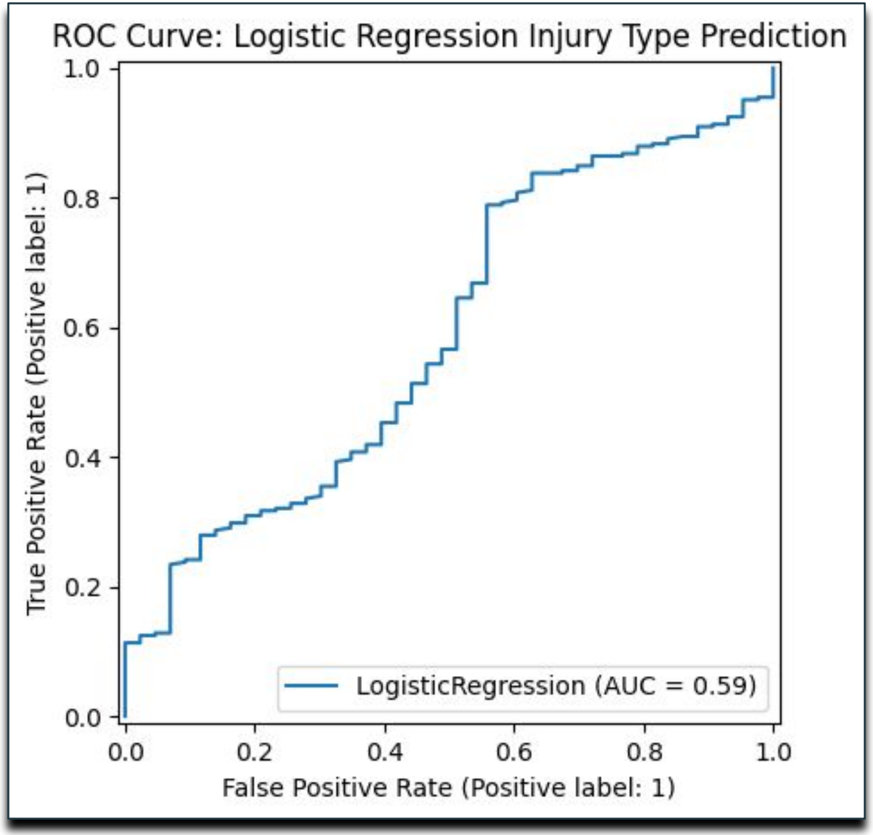
```python
# Loading the data
df = pd.read_excel('/content/football_injury.xlsx')
df = df[df['Contact/ non-contact'].isin(['contact', 'non contact'])].copy()
df['injury_type'] = df['Contact/ non-contact'].map({'contact': 1, 'non contact': 0})
df = df.dropna(subset=['position', 'height', 'weight', 'years_exp'])

# preprocessing
df['pos_encoded'] = LabelEncoder().fit_transform(df['position'])
X = df[['pos_encoded', 'height', 'weight', 'years_exp']]
y = df['injury_type']
X = StandardScaler().fit_transform(X)

# train / test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, stratify=y, random_state=42)

# Grid Search for Random Forest
param_grid = {
    'n_estimators': [100, 200],
    'max_depth': [None, 10, 20],
    'min_samples_split': [2, 5]
}

grid = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5, scoring='accuracy')
grid.fit(X_train, y_train)

print("✅ Best Parameters:", grid.best_params_)

# Evaluate the best model on test set
best_model = grid.best_estimator_
y_pred = best_model.predict(X_test)
```

ROC Curve

ROC Curve helps see how well the model can be split into 2 classes

AUC shows that the model is marginally better than random

AUC = 0.59
– Only slightly above 0.50

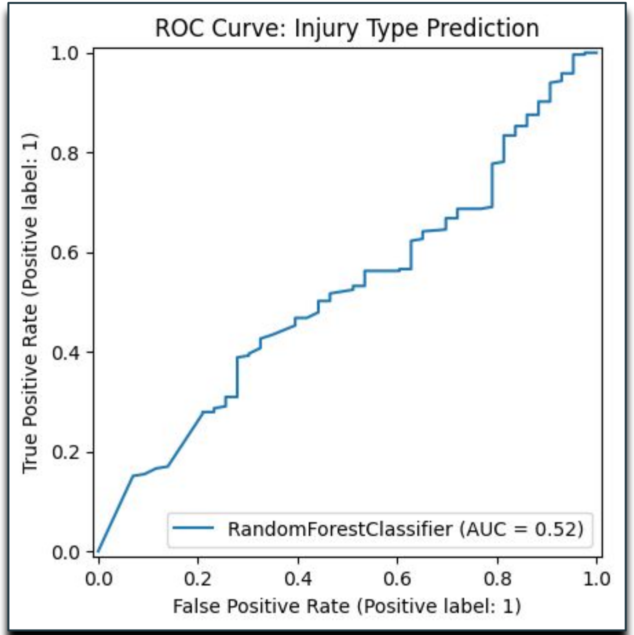


Interpreting ROC Curve

ROC Curve helps see how well the model can be split into 2 classes

X-axis (False Positive Rate)
– % of non-contact injuries mislabeled as contact
Y-axis (True Positive Rate / Recall)
– % of contact injuries correctly identified
Curve near the diagonal
– Model's discrimination ≈ random guessing
AUC = 0.52
– Only slightly above 0.50
Ideal ROC shape
– Sharp bend toward top-left, AUC ↑

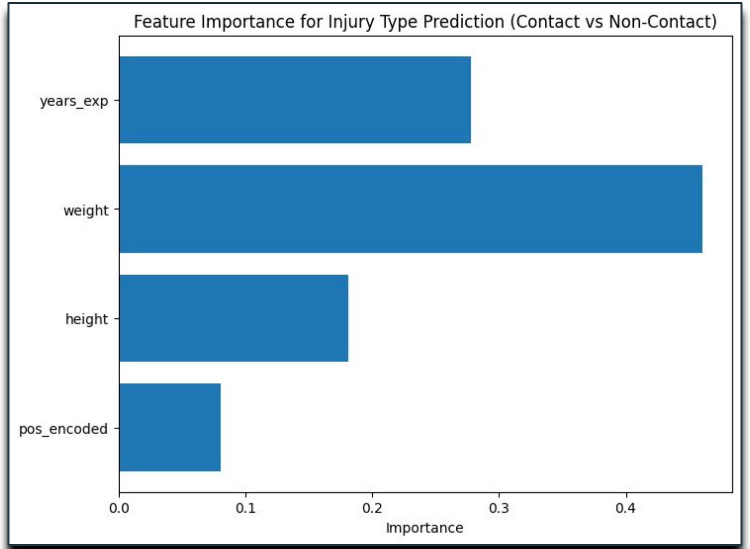
```
# ---------------- Train-Test Split ----------------

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)

# ---------------- Train the Model ----------------

model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# ---------------- Evaluate the Model ----------------

y_pred = model.predict(X_test)
y_prob = model.predict_proba(X_test)[:,1]

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non Contact', 'Contact'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix: Injury Type Prediction')
plt.show()

print("\nROC AUC Score:", roc_auc_score(y_test, y_prob))

# ---------------- Plot ROC Curve ----------------

RocCurveDisplay.from_estimator(model, X_test, y_test)
plt.title('ROC Curve: Injury Type Prediction')
plt.show()
```



Importance of Variable

- Shows which variable affect risk of injury the most
- Weight affects it the most
- Position affect it the least

```python
# ---------------- Feature Importance ----------------

importances = model.feature_importances_
feature_names = X.columns

plt.figure(figsize=(8,6))
plt.barh(feature_names, importances)
plt.xlabel("Importance")
plt.title("Feature Importance for Injury Type Prediction (Contact vs Non-Contact)")
plt.show()
```



K-Fold Accuracy Test

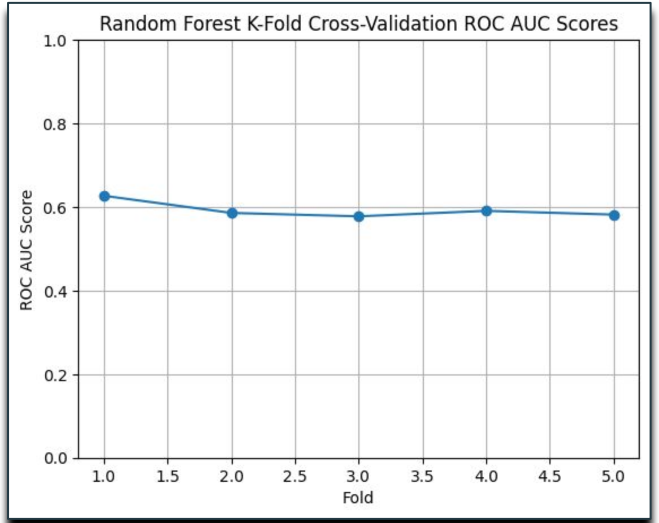
Uses K-fold accuracy test to find the ROC AUC scores

Graph shows that the variance is low

However, as previously seen, it shows a score of 0.59, showing its only marginally better than random guessing.

```
K-Fold Cross-Validation ROC AUC Scores (Random Forest):
[0.62690735 0.58587127 0.57766346 0.59077193 0.5818333 ]

Average ROC AUC Score across folds: 0.5926094621271585
```

```python
from sklearn.model_selection import cross_val_score, KFold

rf_model = RandomForestClassifier(n_estimators=100, random_state=42)

# Define K-Fold
kf = KFold(n_splits=5, shuffle=True, random_state=42)

# Perform cross-validation
cv_scores = cross_val_score(rf_model, X, y, cv=kf, scoring='roc_auc')

# Results
print("\nK-Fold Cross-Validation ROC AUC Scores (Random Forest):")
print(cv_scores)

print("\nAverage ROC AUC Score across folds:", np.mean(cv_scores))

# -------------------- Plot CV Scores --------------------

plt.plot(range(1, 6), cv_scores, marker='o')
plt.title('Random Forest K-Fold Cross-Validation ROC AUC Scores')
plt.xlabel('Fold')
plt.ylabel('ROC AUC Score')
plt.ylim(0, 1)
plt.grid(True)
plt.show()
```



Confusion Matrix

True Negatives (TN): 0

– Non-contact injuries correctly predicted as non-contact

False Positives (FP): 43

– Non-contact injuries misclassified as contact

False Negatives (FN): 0

– Contact injuries missed

True Positives (TP): 265

– Contact injuries correctly predicted as contact

Model Behavior:

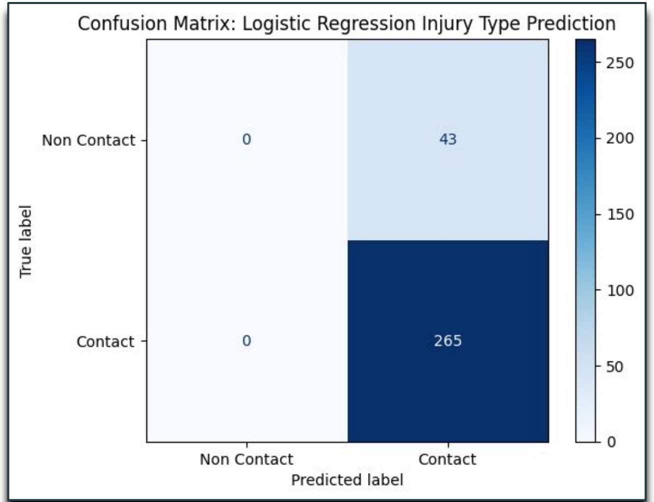
– Predicted every test case as “Contact”

Metrics Impact:

– Recall (Contact) = 265 / (265 + 0) = 100%

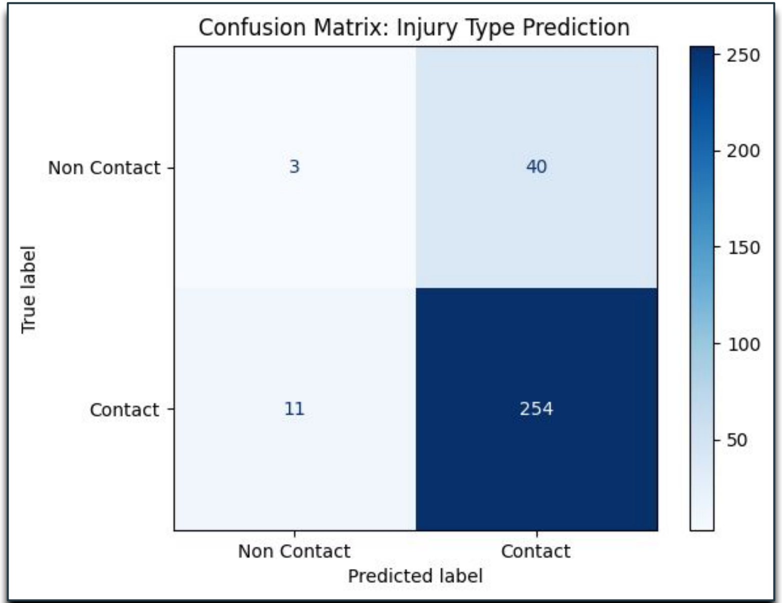
– Specificity (Non-Contact) = 0 / (0 + 43) = 0%

```
log_reg_model = LogisticRegression(max_iter=1000)
log_reg_model.fit(X_train, y_train)

# ---------------- Evaluate the Model ----------------

y_pred = log_reg_model.predict(X_test)
y_prob = log_reg_model.predict_proba(X_test)[:,1]

print("\nClassification Report:")
print(classification_report(y_test, y_pred))

print("\nConfusion Matrix:")
cm = confusion_matrix(y_test, y_pred)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=['Non Contact', 'Contact'])
disp.plot(cmap='Blues')
plt.title('Confusion Matrix: Logistic Regression Injury Type Prediction')
plt.show()

print("\nROC AUC Score:", roc_auc_score(y_test, y_prob))

# ---------------- Plot ROC Curve ----------------

RocCurveDisplay.from_estimator(log_reg_model, X_test, y_test)
plt.title('ROC Curve: Logistic Regression Injury Type Prediction')
plt.show()
```



Confusion Matrix v2

- Model was ran a couple more times and was able to predict the injury type better
    - Accuracy performed better overall, with non contact significantly improving
- The training did predict non-contact mostly wrong, but it was an improvement over the first version



Classification Report:

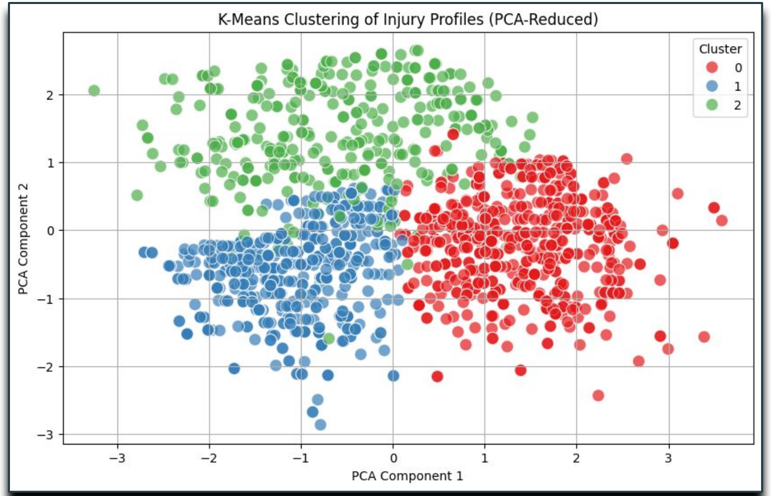
	precision	recall	f1-score	support
0	0.21	0.07	0.11	43
1	0.86	0.96	0.91	265
accuracy			0.83	308
macro avg	0.54	0.51	0.51	308
weighted avg	0.77	0.83	0.80	308

K-Clustering Means

```python
# build X from the cleaned df
df['pos_encoded'] = LabelEncoder().fit_transform(df['position'])
X = df[['pos_encoded', 'height', 'weight', 'years_exp']]

# Standardize
X_scaled = StandardScaler().fit_transform(X)
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_scaled)

# Assign PCA back to the cleaned dataframe
df['pca1'] = X_pca[:, 0]
df['pca2'] = X_pca[:, 1]

kmeans = KMeans(n_clusters=3, random_state=42)
df['cluster'] = kmeans.fit_predict(X_scaled)

# Setting up the plot
plt.figure(figsize=(10, 6))
sns.scatterplot(
    data=df,
    x='pca1',
    y='pca2',
    hue='cluster',
    palette='Set1',
    s=100,
    alpha=0.7
```

Cluster Summarization:

- **Cluster 0:** Heavy & inexperienced players → highest injury rates
- **Cluster 1:** Mid-range height/weight & experience → moderate risk
- **Cluster 2:** Younger, lighter players → lowest injury rates

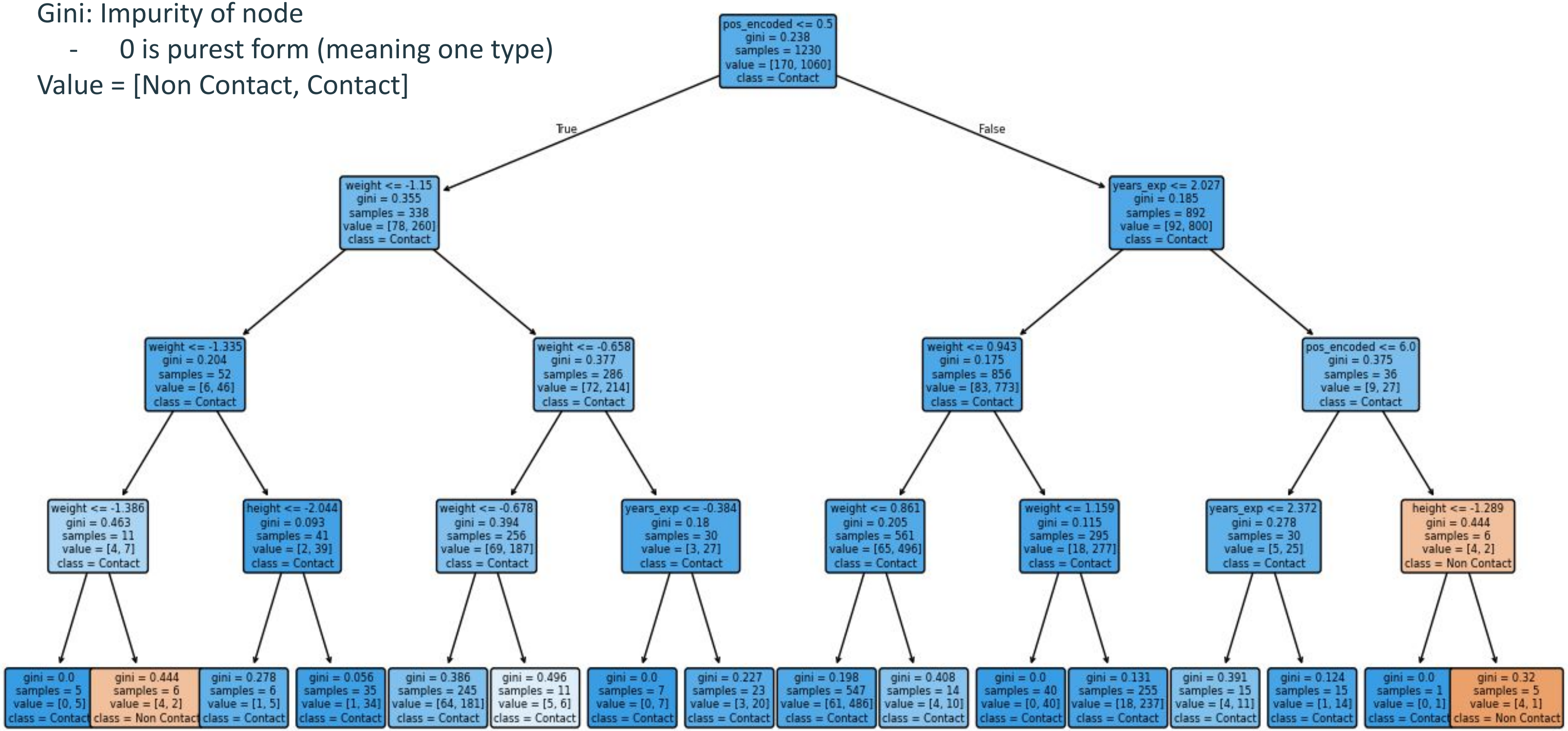


Decision Tree: Contact vs Non-Contact Injury

Gini: Impurity of node

- 0 is purest form (meaning one type)

Value = [Non Contact, Contact]



Implications

- With an AUC of about 0.59, the model is too unreliable to be use
- Clustering revealed 3 player profiles
  - Can be used for customized training, recovery plan
- Still a need for expert oversight
  - People will still need to be reviewed before making any decisions
- Further problems arose when running other tests like SMOTE, Light GDM, GBoost

Future Work

- Validate our model by incorporating more seasons
- Bring in player workload data
    - Combine data
    - Measure speed, acceleration, etc.
    - Snaps per game, targets, etc.
- Improve the model
    - More accurate
- Make another model to show game environment
    - Use field type or team played to determine injury statistics
    - Was attempted, but confusion matrix accuracy was subpar (0.49) - may need a different dataset

Questions?