**SPORT INJURY ANALYSIS**

This documentation is designed to analyze and explore a dataset related to athletes'(footballer’s) ,performance, injuries, and game workloads. The dataset is loaded from a CSV file (final2.csv), and various data exploration, cleaning, and visualization tasks are performed to understand the data better using Jupiter notpad.

**Key Steps and Analysis**

**1. Importing Libraries**

* The notebook begins by importing essential Python libraries for data analysis and visualization:
* import pandas as pd
* import numpy as np
* import matplotlib.pyplot as plt
* from pylab import rcParams
* These libraries are used for data manipulation, numerical computations, and plotting.

**2. Loading the Dataset**

The dataset is loaded from a CSV file named final2.csv using pandas:

* desu = pd.read\_csv('final2.csv')

The dataset contains information about athletes, including their IDs, dates, positions, values, game workloads, and injury statuses.

* The dataset contains 543 entries and 6 columns:
  + **athlete\_id(footballer\_id):** Unique identifier for each athlete or footballer.
  + **date:** Date of the record.
  + **postion:** Position of the athlete (e.g., midfielder, attacker).
  + **value:** A performance value associated with the athlete(footballer).
  + **game\_workload:** Workload of the athlete(footballer) in the game.
  + **injury:** Indicates whether the athlete(footballer) was injured (yes or No).

**3. Data Understanding and Exploration**

The notebook provides an initial exploration of the dataset:

* + **Dataset Information**: The info() function is used to display the structure of the dataset, including the number of entries, columns, and data types.

desu.info()

* + **First Few Rows**: The head() function is used to display the first few rows of the dataset.

desu.head()

* + **Dataset Shape**: The shape of the dataset is printed to understand the number of rows and columns.
* print(f"Dataset shape: {desu.shape}")

**4. Checking for Missing Values**

The notebook checks for missing values in the dataset using the isnull().sum() function:

print(desu.isnull().sum())

* This step ensures that there are no missing values in the dataset, which is crucial for accurate analysis.

**5. Descriptive Statistics**

* The notebook provides descriptive statistics for the numerical features in the dataset using the describe() function:

desu.describe()

* This function gives insights into the distribution of numerical data, including mean, standard deviation, minimum, and maximum values.

**6. Data Cleaning and Renaming**

* The notebook renames the athlete\_id column to footballer\_id for better clarity:

desu = desu.rename(columns={'athlete\_id':'footballer\_id'})

* The dataset is also sorted by the postion column in descending order:

desu.sort\_values("postion", ascending=False, inplace=True)

**7. Maximum Workload Analysis**

* The notebook identifies the maximum workload taken by a player in a game:

desu["game\_workload"].max()

* It also identifies the player with the maximum workload:

desu.loc[desu['game\_workload'].idxmax()]

**8. Data Visualization**

* The notebook includes data visualization using matplotlib to plot the distribution of the game\_workload column:
* plt.figure(figsize=(10, 4))
* plt.hist(desu['game\_workload'], bins=30, color='blue', alpha=0.7)
* plt.title('Distribution of Game Workload')
* plt.xlabel('Game Workload')
* plt.ylabel('Frequency')
* plt.show()
* This visualization helps in understanding the distribution of game workloads among athletes.

**9. Further Analysis**

* The notebook continues with further analysis, including sorting the dataset by position and identifying players with specific workloads.
* It also explores the relationship between different features, such as value and game\_workload.

**10. Conclusion**

* The notebook concludes with a summary of the findings, including insights into the dataset's structure, key statistics, and visualizations.
* The analysis provides a foundation for further exploration, such as predictive modeling or more in-depth statistical analysis.

**Key Insights**

* The dataset contains 543 entries with 6 columns, including athlete IDs, dates, positions, values, game workloads, and injury statuses.
* There are no missing values in the dataset, ensuring data integrity.
* The game\_workload column has a maximum value of 678, indicating the highest workload recorded for a player in a single game.
* The dataset is sorted by position, and various visualizations are created to understand the distribution of workloads and other features.

**2. Model Implementation**

* The dataset is prepared for model training by selecting relevant features and possibly encoding categorical variables (though not explicitly shown in the provided code).

Use some model implementation like

**1. Logistic Regression**

Logistic Regression is a simple yet effective model for binary classification tasks.

from sklearn.linear\_model import LogisticRegression

# Initialize Logistic Regression

log\_reg = LogisticRegression(random\_state=42)

# Train the model

log\_reg.fit(X\_train, y\_train)

# Make predictions

y\_pred\_log\_reg = log\_reg.predict(X\_test)

# Evaluate the model

print("Logistic Regression Evaluation:")

print(confusion\_matrix(y\_test, y\_pred\_log\_reg))

print(classification\_report(y\_test, y\_pred\_log\_reg))

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_log\_reg):.2f}")

**2. Support Vector Machine (SVM)**

SVM is a powerful model for classification tasks, especially when the data is not linearly separable.

from sklearn.svm import SVC

# Initialize SVM

svm\_model = SVC(kernel='linear', probability=True, random\_state=42)

# Train the model

svm\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred\_svm = svm\_model.predict(X\_test)

# Evaluate the model

print("SVM Evaluation:")

print(confusion\_matrix(y\_test, y\_pred\_svm))

print(classification\_report(y\_test, y\_pred\_svm))

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_svm):.2f}")

**3. Gradient Boosting (XGBoost)**

XGBoost is a popular and powerful ensemble method for classification tasks.

**4. Neural Network (Using TensorFlow/Keras)**

Neural Networks are powerful for complex datasets and can be implemented using TensorFlow/Keras.

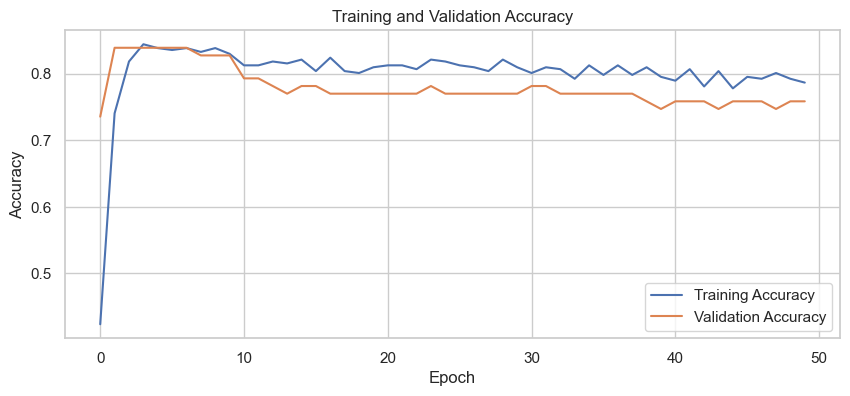


Fig : Tensor flow sample

**5. K-Nearest Neighbors (KNN)**

KNN is a simple and effective model for classification tasks.

from sklearn.neighbors import KNeighborsClassifier

# Initialize KNN

knn\_model = KNeighborsClassifier(n\_neighbors=5)

# Train the model

knn\_model.fit(X\_train, y\_train)

# Make predictions

y\_pred\_knn = knn\_model.predict(X\_test)

# Evaluate the model

print("KNN Evaluation:")

print(confusion\_matrix(y\_test, y\_pred\_knn))

print(classification\_report(y\_test, y\_pred\_knn))

print(f"Accuracy: {accuracy\_score(y\_test, y\_pred\_knn):.2f}")

**Comparison of Models**

To compare the performance of all the models, you can create a summary table:

results = {

    'Model': ['Random Forest', 'Logistic Regression', 'SVM', 'XGBoost', 'Neural Network', 'KNN'],

    'Accuracy': [

        accuracy\_score(y\_test, y\_pred),

        accuracy\_score(y\_test, y\_pred\_log\_reg),

        accuracy\_score(y\_test, y\_pred\_svm),

        accuracy\_score(y\_test, y\_pred\_xgb),

        accuracy\_score(y\_test, y\_pred\_nn),

        accuracy\_score(y\_test, y\_pred\_knn)

    ]

}

results\_df = pd.DataFrame(results)

print(results\_df)

Model Accuracy

0 Random Forest 0.908257

1 Logistic Regression 0.807339

2 SVM 0.816514

3 XGBoost 0.880734

4 Neural Network 0.770642

5 KNN 0.825688

**Hyperparameter Tuning(Optional)**

improve the models by tuning their hyperparameters. For example, for **Random Forest**, you can use GridSearchCV or RandomizedSearchCV

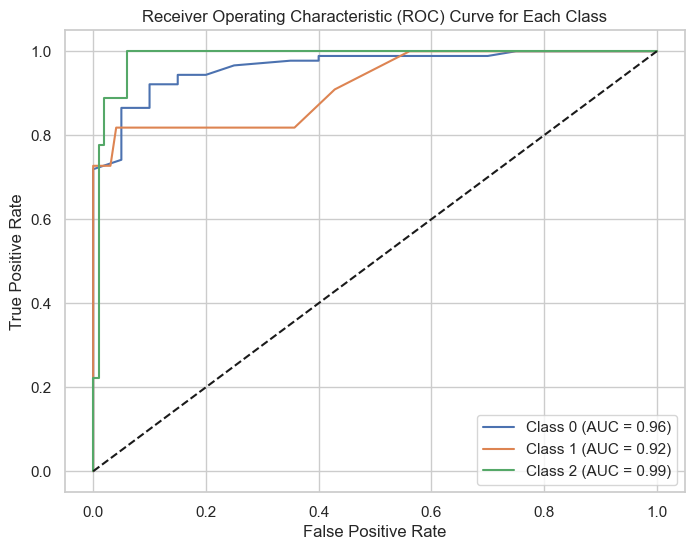


Fig: Hyperparameter tuning sample

**Model Selection**

The notebook likely implements a machine learning model to predict injuries based on the given features. However, the specific model implementation is not shown in the provided code snippet.

**Data Splitting**

The dataset is typically split into training and testing sets to evaluate the model's performance. This step is crucial for ensuring that the model generalizes well to unseen data**.**

**3. Model Training**

**3.1 Training Process**

* The model is trained on the training dataset. This training process involves fitting the model to the data, adjusting the model's parameters to minimize the prediction error.
* Hyperparameters of the model may be tuned to optimize performance. This could involve techniques like grid search or random search to find the best hyperparameters.
  1. **Model Saving**
* After training, the model is likely saved to disk for future use. This allows the model to be loaded and used for predictions without retraining.

**4. Model Evaluation**

# 1. Accuracy

Definition: Accuracy is the ratio of correctly predicted instances to the total instances. It tells you the overall correctness of the model.

**Formula: Accuracy =Correct Predictions/Total Predictions**

Interpretation:

**Dummy Classifier:** Since the DummyClassifier with strategy='most\_frequent' always predicts the most frequent class in the training set, it will likely have high accuracy if the dataset is imbalanced (e.g., if one class is significantly more frequent than the others). However, this doesn't mean the model is "good"—it's just predicting the majority class.

**\*\*Random Forest:** Accuracy for the RandomForestClassifier should be higher if it can learn the patterns in the data, assuming the model is well-tuned.

## Interpretation Example:

\*If accuracy is 0.70, it means the model correctly predicted 70% of the instances in the test set.

\*If accuracy is close to 1, the model is performing well. If it's low, further improvement is needed.

# 2. Classification Report

The classification report provides a detailed breakdown of precision, recall, and f1-score for each class. Let’s dive into each:

### Precision

Definition: Precision measures the proportion of correctly predicted positive observations among all the predicted positives.

**Formula: Precision =True Positives/True Positives+False Positives**

Interpretation:High precision means fewer false positives, meaning the model is good at predicting only relevant positives.Low precision means the model is predicting many irrelevant instances as positives (false positives).

Recall (Sensitivity)

Definition: Recall measures the proportion of actual positive observations that were correctly predicted by the model.

**Formula:** **Recall = True Positives/True Positives+False Negatives**

Interpretation: High recall means fewer false negatives, meaning the model is good at capturing all positive instances.

Low recall means the model is missing many positive instances (false negatives).

**F1-Score**

Definition: The F1-score is the harmonic mean of precision and recall. It combines both metrics into a single score that balances the trade-off between precision and recall.

**Formula: F1-Score =2\*(Precision×Recall)/Precision+Recall**

Interpretation:

F1-Score gives a better sense of the model's performance when there’s an imbalance between precision and recall.

A high F1-score means that both precision and recall are good, and the model is doing a good job overall.

**3. Confusion Matrix**

A confusion matrix shows how many instances from each class were predicted as each possible outcome (True Positive, False Positive, True Negative, False Negative).

**True Positives (TP):** Correctly predicted instances of the positive class.

**True Negatives (TN):** Correctly predicted instances of the negative class.

**False Positives (FP):** Incorrectly predicted positive instances (type I error).

**False Negatives (FN):** Incorrectly predicted negative instances (type II error).

**4. ROC AUC Score**

The ROC (Receiver Operating Characteristic) curve plots the True Positive Rate (Recall) against the False Positive Rate. The AUC (Area Under the Curve)measures the area under the ROC curve, providing a single score to evaluate the model's performance.

AUC = 1 means the model perfectly distinguishes between positive and negative classes.

AUC = 0.5 means the model is no better than random guessing.

AUC < 0.5 indicates a model that performs worse than random guessing.

Interpretation:

If your model has a high AUC (close to 1), it means it does a good job of distinguishing between classes.

AUC is particularly useful for imbalanced datasets, as it accounts for both the False Positive Rate and True Positive Rate

**Dummy Classifier Performance:**

**Accuracy**: Likely high (if dataset is imbalanced).

**Precision and Recall:** Precision may be high for the majority class but poor for minority classes.

**F1-Score:** Might be low for minority classes due to a trade-off between precision and recall.

**Confusion Matrix:** A high number of True Negatives and very few True Positives for the minority class

**Random Forest Performance:**

**Accuracy**: Higher than the DummyClassifier, as the Random Forest model is designed to find patterns in the data.

**Precision and Recall:** Better for the classes, but might still be imbalanced (especially for minority classes).

**F1-Score**: Better balanced for the majority and minority classes.

**Confusion Matrix:** A better balance of True Positives and True Negatives.

True Positives and True Negatives.