✨ Week 10 Student Guide – Machine Learning for Injury Risk Prediction

This guide covers the fundamental steps in applying **Machine Learning (ML)** to predict basketball player injuries, focusing on the concepts of preprocessing, model training, and evaluation in Python using **Scikit-learn (sklearn)**.

💻 Part 1: Machine Learning Fundamentals

1. The Goal: Binary Classification

The objective is **binary classification**, which means predicting one of two outcomes:

- **Class 0 (Negative):** No injury next week.
- **Class 1 (Positive):** Injury next week.

2. The Imbalance Problem (Crucial Takeaway)

The data simulated in the assignment represents a real-world challenge: **Class Imbalance**.

- The generated dataset has an overall injury rate of only (1 out of 1000).
- The executed code shows that the testing set contains **0** injury cases.
- When the positive class (injury) is rare or missing in the test data, the model can achieve **accuracy** simply by predicting "no injury" every time. This indicates a **faulty model** that is highly misleading and cannot predict the target event.

Key Insight: In basketball analytics, perfect metrics in a classification report usually mean the model failed to capture the rare event. Focus on other metrics (like feature importance) to find true insights, even if the accuracy is flawed.

🛠️ Part 2: Preparing the Data

1. Define Features (X) and Target (y)

The independent variables () are the player workload and recovery metrics used to make the prediction, and the dependent variable () is the outcome you are trying to predict.

Feature Group	Metric	Description
Load Metrics	`Minutes_Load`, `HighSpeedRuns`, `JumpLoad`	Training and game demands.
Recovery Metrics	`HeartRate`, `SleepHours`	Baseline fatigue/recovery indicators.
History	`PreviousInjury`	Binary flag for past risk.

```python
X = df[['Minutes_Load', 'HighSpeedRuns', 'JumpLoad', 'HeartRate',
'SleepHours', 'PreviousInjury']]
y = df['InjuryNextWeek']
```

2. Split the Data

The data must be split into a **training set** (to teach the model) and a **testing set** (to evaluate its performance on unseen data). Using `random_state` ensures that your split is reproducible.

```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    X,
    y,
    test_size=0.2, # Use 20% of the data for testing
    random_state=42
)
```

3. Scale Features using StandardScaler

Scaling transforms numerical features to have a mean of 0 and a standard deviation of 1. This prevents features with larger numerical ranges (like `JumpLoad`) from unfairly dominating the learning process over smaller features (like `SleepHours`).

```python
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train) # Fit and transform on the
training data
X_test_scaled = scaler.transform(X_test)      # Only transform the test data
(use training data's scale)
```

⚙️ Part 3: Model Training and Evaluation

1. Logistic Regression

Logistic Regression is a simple, interpretable model used for binary classification. It's often the first model tested in ML projects.

```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report

log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
y_pred_log = log_reg.predict(X_test_scaled)

print(classification_report(y_test, y_pred_log))
```

2. Random Forest Classifier

The Random Forest is an ensemble model (many decision trees) that often provides better performance and is excellent for calculating feature importance.

```python
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
y_pred_rf = rf.predict(X_test)

print(classification_report(y_test, y_pred_rf))
```

3. Confusion Matrix

The confusion matrix visually shows which predictions were correct (true positives/negatives) and incorrect (false positives/negatives) .

```python
from sklearn.metrics import confusion_matrix
import seaborn as sns

cm = confusion_matrix(y_test, y_pred_rf)
sns.heatmap(cm, annot=True, fmt='d', cmap='Reds')
plt.title('Confusion Matrix - Random Forest')
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.show()
```

📊 Part 4: Feature Importance and Summary

1. Feature Importance

The Random Forest model calculates a score for each feature, showing its relative contribution to the final prediction.

- **Key Findings from Executed Code:** In the provided run, `PreviousInjury` and `SleepHours` were the most important features, followed by the workload metrics (`JumpLoad`, `HighSpeedRuns`, `Minutes_Load`).

```python
# Code to visualize the importance scores
feat_importance = pd.Series(rf.feature_importances_, index=X.columns)
plt.figure(figsize=(8,5))
feat_importance.sort_values().plot(kind='barh', color='red')
plt.title('Feature Importance - Injury Risk Prediction (Bulls)')
plt.show()
```

2. 5-Sentence Summary (Assignment Task)

Your summary should integrate the key findings in basketball context, addressing the limitations (class imbalance) and the primary risk factors.

1. **Model Performance & Limitation:** Both the Logistic Regression and Random Forest models showed perfect performance metrics (Accuracy, Precision, Recall) on the test set.
2. **Data Challenge:** This perfect score is misleading due to the extreme class imbalance, as the test set contained zero actual injury cases.
3. **Primary Risk Factors:** Despite the flawed evaluation, the **Feature Importance** plot suggests that prior injury history (`PreviousInjury`) and **recovery metrics** (`SleepHours` and `HeartRate`) are the primary drivers of predicted risk.
4. **Workload Role:** The training load metrics, such as `JumpLoad` and `Minutes_Load`, contributed less to the predictive model in this simulation compared to recovery data.
5. **Conclusion:** The analytics suggest that managing a player's **sleep and heart rate recovery** is the most valuable and immediate intervention for mitigating injury risk, especially for those with a **history of injury**.