

```

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from google.colab import drive
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.preprocessing import LabelEncoder
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import accuracy_score, classification_report

# Setup environment
warnings.filterwarnings("ignore")
drive.mount('/content/drive')

# Load the dataset (Make sure this path is correct for your Drive)
df = pd.read_csv('/content/drive/MyDrive/Project 2/sports_performance_data.csv')

# --- FIX FOR KEYERROR: CREATE THE PREDICTION TARGET ---
# We create 'Is_Fit_For_Next_Match' based on logic:
# If they have no major injury and enough sleep, they are 'Fit' (Yes)
if 'Is_Fit_For_Next_Match' not in df.columns:
    def determine_fitness(row):
        # We use 'Injury_History' and 'Sleep_Hours_per_Night'
        # (Change these names if your CSV columns are spelled differently)
        injury = str(row.get('Injury_History', 'None'))
        sleep = row.get('Sleep_Hours_per_Night', 8)
        return 'Yes' if (injury != 'Major' and sleep > 6) else 'No'

    df['Is_Fit_For_Next_Match'] = df.apply(determine_fitness, axis=1)

# HANDLE MISSING VALUES

# Delete columns with more than 30% missing values
threshold = 30
missing_pct = df.isnull().mean() * 100
df.drop(columns=missing_pct[missing_pct > threshold].index, inplace=True)

# Encode categorical variables (Words to Numbers)
le = LabelEncoder()
for col in df.select_dtypes(include=['object']).columns:
    df[col] = le.fit_transform(df[col].astype(str))

# Impute remaining missing values with Mean (Numbers) or Mode (Categories)
for col in df.columns:
    if df[col].dtype in ['float64', 'int64']:
        df[col] = df[col].fillna(df[col].mean())
    else:
        df[col] = df[col].fillna(df[col].mode()[0])

```

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

APPLYING ML MODELS WITH GRID SEARCH OPTIMIZER

```

X = df.drop('Is_Fit_For_Next_Match', axis=1)
y = df['Is_Fit_For_Next_Match']

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

```

```

# Grid Search to find best settings for the model
param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [None, 10, 20]
}

grid_search = GridSearchCV(RandomForestClassifier(random_state=42), param_grid, cv=5)
grid_search.fit(X_train, y_train)

print(f"Best Parameters: {grid_search.best_params}")
initial_pred = grid_search.predict(X_test)
print(f"Initial Accuracy: {accuracy_score(y_test, initial_pred):.4f}")

```

```

Best Parameters: {'max_depth': 20, 'n_estimators': 100}
Initial Accuracy: 0.8111

```

```

# Method 1: Filter Method (Correlation Analysis)
correlations = df.corr()['Is_Fit_For_Next_Match'].abs().sort_values(ascending=False)
filter_features = correlations[1:11].index.tolist() # Top 10 factors

```

```
# Method 2: Embedded Method (Feature Importance)
rf_selector = RandomForestClassifier().fit(X, y)
importance_df = pd.Series(rf_selector.feature_importances_, index=X.columns)
embedded_features = importance_df.nlargest(10).index.tolist()
```

```
print(f"\nFilter Selected Features: {filter_features}")
print(f"Embedded Selected Features: {embedded_features}")
```

```
# APPLY MODELS TO SELECTED FEATURES & EVALUATE
```

```
X_selected = X[embedded_features]
X_train_s, X_test_s, y_train_s, y_test_s = train_test_split(X_selected, y, test_size=0.2, random_state=42)
```

```
# Final model using only the best features
final_model = RandomForestClassifier(**grid_search.best_params_, random_state=42)
final_model.fit(X_train_s, y_train_s)
final_pred = final_model.predict(X_test_s)
```

```
print("\nFinal Model Evaluation (Using Selected Features):")
print(classification_report(y_test_s, final_pred))
```