Ajay Sreekumar

INFO 531 – Data Warehousing & Analytics in the Cloud

Spring 2024

Week 16 Assignment – Final Project Update

Nayem Rahman

6th May, 2024

FINAL PROJECT REPORT

UPDATE 1:

PROBLEM STATEMENT:

The problem statement for this project is: while running is a great form of exercise, recreation and sport participation for adults – running under adverse conditions or with inadequate clothing or equipment can cause physical stress and a variety of injuries. Can we predict which factors exactly caused these injuries? Based on that, can we predict if a runner A is injured if they're running in a specific set of conditions or factors?

The original data came from DataverseNL in association with University of Groningen.

Link: https://dataverse.nl/dataset.xhtml?persistentId=doi:10.34894/UWU9PV

Kaggle Version Link: https://www.kaggle.com/datasets/shashwatwork/injury-prediction-forcompetitive-runners/data

The data set consists of a detailed training log from a Dutch high-level running team over a period of seven years (2012-2019). Also included were the middle and long-distance runners of the team, that is, those competing on distances between the 800 meters and the marathon. This design decision is motivated by the fact that these groups have strong endurance-based components in their training, making their training regimes comparable. The head coach of the team did not change during the years of data collection. The data set contains samples from 74 runners, of whom 27 are women and 47 are men. At the moment of data collection, they had been in the team for an average of 3.7 years. Most athletes competed on a national level, and some also on an international level. The study was conducted according to the requirements of the Declaration of Helsinki and was approved by the ethics committee of the second author's institution.

For data analysis and manipulation, we will use pandas, matplotlib as well as NumPy libraries in Python (PyCharm). As part of Machine Learning techniques or algorithms, we will test which of Logistic Regression, K-Nearest Neighbours or Random Forest Classifiers would give us the highest F1 scores, and correspondingly find the best possible parameters for the model – which should give us an accuracy score above 85%, so that we have a feasible model to use.

UPDATE 2:

Data Preparation Plan:

Data preparation plays a vital role in every machine learning endeavour, guaranteeing that the data is appropriately formatted for analysis and modelling purposes. The process begins with reading the dataset from a CSV file, followed by an exploration phase (EDA). During exploration, key steps include printing the first few rows and columns of the dataset to get a glimpse of its structure, examining data types, summarizing statistics, checking for missing values, and identifying unique values in categorical variables. We also attempt to find out any relationships between the variables we utilize in the dataset.

Visualizing the correlation matrix helps to understand the relationships between predictor variables, shedding light on potential multicollinearity issues and guiding feature selection. Furthermore, visualizing the distribution of the target variable, 'injury', using count plots or pie charts provides insights into the class distribution and potential class imbalances.

To ensure data quality, duplicate rows are identified and removed if present. Subsequently, the dataset is split into predictor variables (X) and the target variable (y), with the former representing the features used for prediction and the latter representing the outcome of interest. Normalization of predictor variables using techniques like 'StandardScaler' may be employed to scale the features to a similar range, facilitating model convergence and performance.

Finally, the dataset undergoes partitioning into training and testing subsets employing the 'train_test_split' function, commonly employing ratios like 80:20 or 70:30. The training set assumes the role of training machine learning models, whereas the testing set remains segregated for the purpose of evaluating model performance on unseen data, thereby scrutinizing its capacity for generalization.

Predictor Variables (Features) and Response (Target Variable):

The predictor variables, also known as features, encompass various metrics related to athlete training and performance, such as the number of sessions, maximum kilometres covered in a day, total kilometres logged, number of tough sessions, among others. These features serve as inputs to the machine learning models and are hypothesized to influence the likelihood of injury occurrence.

On the other hand, the response variable, 'injury', serves as the target variable that the models aim to predict. It is a binary variable indicating whether an athlete suffered an injury during the observation period, with values of 1 representing the presence of an injury and 0 denoting its absence. Predicting this target variable is the primary objective of the machine learning task, as it enables proactive injury management and prevention strategies in sports and athletic training settings.

Training and Testing Data Sets:

The training data set comprises a subset of the original data reserved for model training, typically the majority portion, such as 80% of the data. This subset includes both predictor variables (X_train) and the corresponding target variable (y_train), allowing the models to learn patterns and relationships between features and the target.

Conversely, the testing data set encompasses the residual portion of the initial data, typically accounting for approximately 20% of the dataset, and remains distinct from the training set. This subset comprises predictor variables (X_test) alongside their corresponding target variable (y_test), functioning as concealed data utilized to assess the performance of trained models. Through evaluating model performance on unseen data, we acquire valuable insights into the models' ability to generalize to novel instances, thereby instilling confidence in their predictive prowess.

Machine Learning Techniques:

The machine learning techniques employed in the project include logistic regression, decision trees, random forests, and support vector machines (SVM). Each technique offers unique advantages and is suitable for different types of data and problem domains.

Logistic Regression: Logistic regression, a linear model tailored for binary classification tasks, estimates the likelihood that a specific instance pertains to a designated class. Despite its simplicity, logistic regression proves highly effective, providing both interpretability and straightforward implementation.

Decision Trees: Decision trees, as non-linear models, divide the feature space into distinct regions, offering an intuitive and visually understandable approach. This segmentation process occurs iteratively, with the data being recursively split according to specific feature thresholds. Consequently, a hierarchical tree-like structure is formed, wherein each terminal node, or leaf, corresponds to a particular class label.

Random Forests: Random Forests are powerful ensemble learning methods that leverage multiple decision trees to enhance predictive performance and mitigate overfitting. By introducing randomness in the tree-building process through the consideration of subsets of features and instances, Random Forests generate a diverse set of trees. These trees' predictions are combined to produce more robust and accurate predictions, making Random Forests a popular choice for various machine learning tasks.

Support Vector Machines (SVM): SVM, a robust algorithm suitable for both classification and regression endeavours, strives to identify the optimal hyperplane within the feature space. This hyperplane effectively distinguishes instances belonging to distinct classes while simultaneously maximizing the margin between these classes. SVMs excel particularly in high-dimensional spaces and exhibit proficiency in capturing intricate relationships existing between features and the target variable.

Each machine learning method involves training the model using a designated training dataset to adjust its parameters, while a separate testing dataset is employed to gauge how well the model performs. Various metrics, including accuracy, precision, recall, F1-score, and the confusion matrix, are computed to measure the efficacy of each model in forecasting injuries based on the provided predictors. The selection of a particular technique hinges on factors like the characteristics of the data, the desired level of interpretability of the model, and the balance between bias and variance.

UPDATE 3:

1. Actual code and output generated as part of Data Preparation. Provide descriptions of this work.

A:

a. Libraries and modules imported:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import random
```

```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score,
f1_score
from sklearn import datasets, metrics
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
```

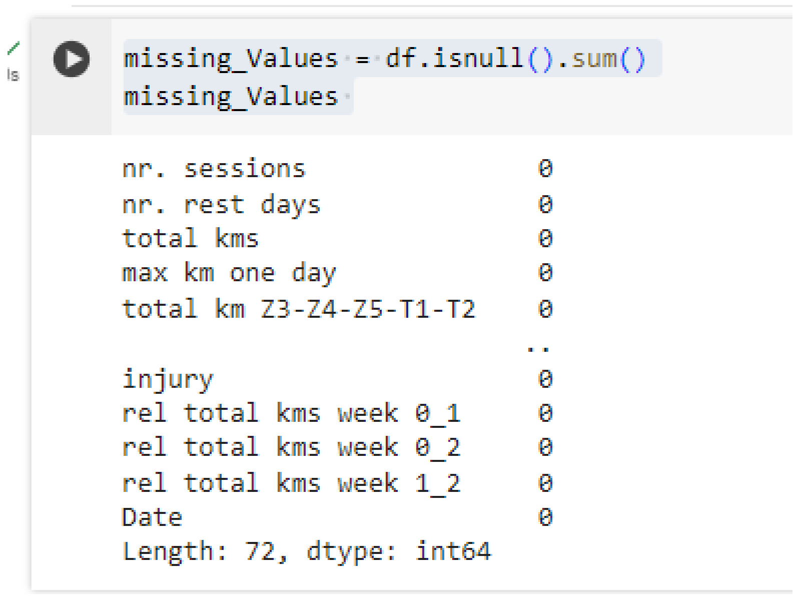
b. Importing the dataset.

```
df = pd.read_csv("week_approach_maskedID_timeseries.csv")
np.random.seed(0)
```

c. Data Exploration and Cleaning.

First, we check for any missing values in the dataset in each of the 71 columns.

```
missing_Values = df.isnull().sum()
missing_Values
```

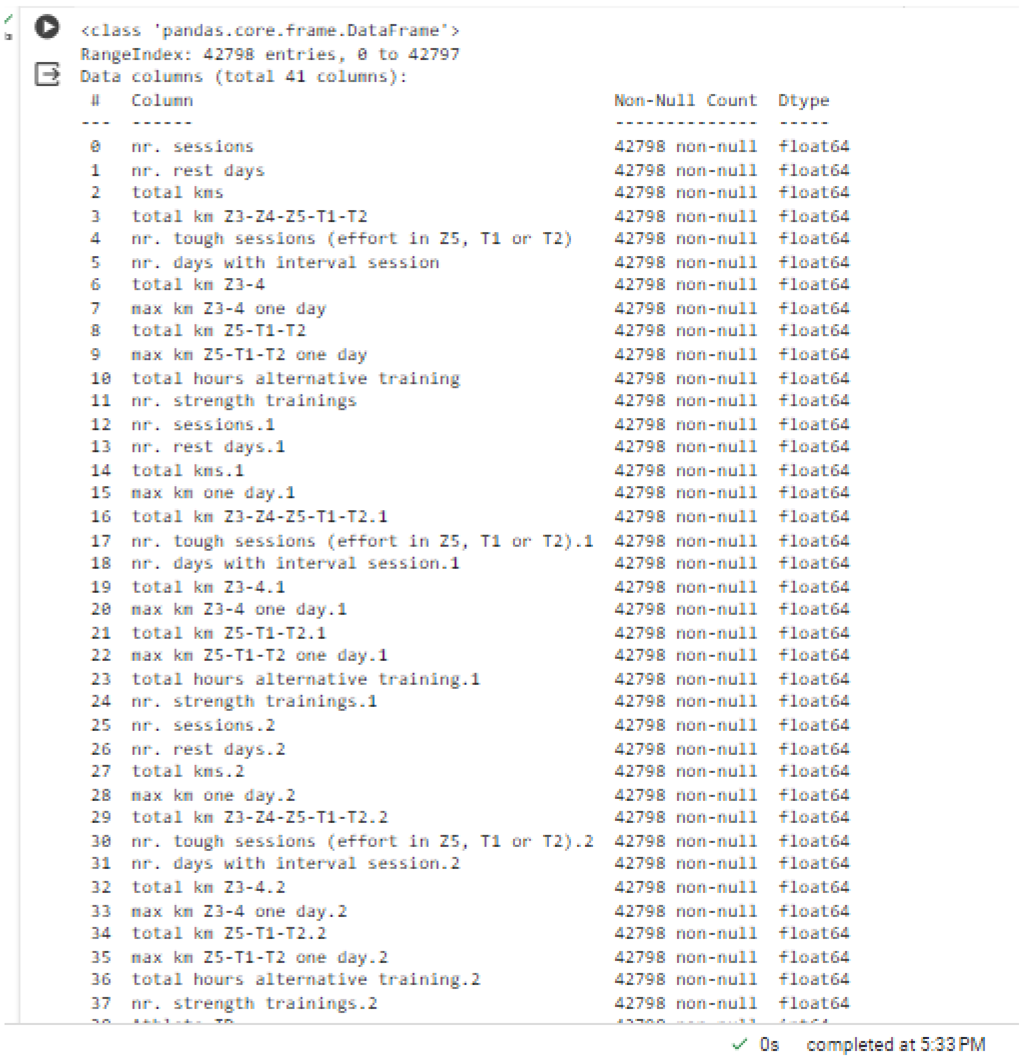


```
missing_Values = df.isnull().sum()
missing_Values
```

```
nr. sessions            0
nr. rest days           0
total kms               0
max km one day          0
total km Z3-Z4-Z5-T1-T2 0
                       ..
injury                  0
rel total kms week 0_1  0
rel total kms week 0_2  0
rel total kms week 1_2  0
Date                    0
Length: 72, dtype: int64
```

We'll begin our data analysis by reducing the dimensionality of the dataset, focusing on dropping attributes determined through empirical analysis. We've excluded any attributes related to subjective feelings of individuals. Our aim is to predict data solely based on quantitative measures of running quality. While factors related to recovery could potentially be valuable, accurately gauging a runner's physical state through survey questions presents considerable challenges.

```
df = df.drop(['avg training success', 'min training success', 'max training success', 'avg training success.1', 'max training success.1', 'min training success.1'], axis = 1)
df = df.drop(['avg training success.2', 'max training success.2', 'min training success.2', 'avg exertion', 'min exertion', 'max exertion'], axis = 1)
df = df.drop(['avg exertion.1', 'min exertion.1', 'max exertion.1', 'avg exertion.2', 'min exertion.2', 'max exertion.2', 'max km one day'], axis = 1)
df = df.drop(['avg recovery', 'min recovery', 'max recovery', 'avg recovery.1', 'min recovery.1', 'max recovery.1', 'avg recovery.2', 'min recovery.2', 'max recovery.2'], axis = 1)
df = df.drop(['rel total kms week 0_1', 'rel total kms week 0_2', 'rel total kms week 1_2'], axis = 1)
df.info()
```

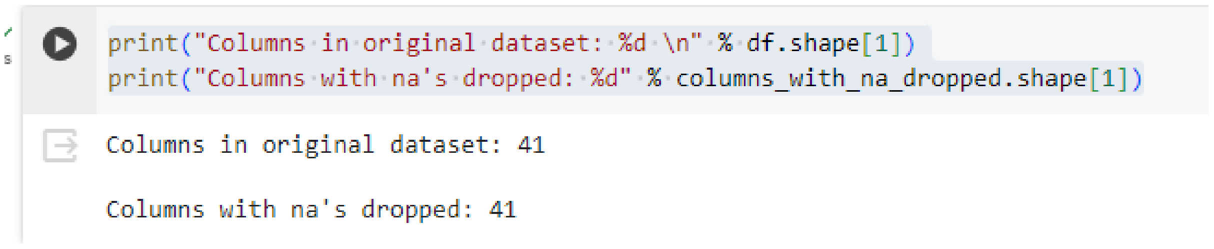


```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 42798 entries, 0 to 42797
Data columns (total 41 columns):
 #   Column                                       Non-Null Count  Dtype
---  ------                                       --------------  -----
 0   nr. sessions                                 42798 non-null  float64
 1   nr. rest days                                42798 non-null  float64
 2   total kms                                    42798 non-null  float64
 3   total km Z3-Z4-Z5-T1-T2                      42798 non-null  float64
 4   nr. tough sessions (effort in Z5, T1 or T2)  42798 non-null  float64
 5   nr. days with interval session               42798 non-null  float64
 6   total km Z3-4                                42798 non-null  float64
 7   max km Z3-4 one day                          42798 non-null  float64
 8   total km Z5-T1-T2                            42798 non-null  float64
 9   max km Z5-T1-T2 one day                      42798 non-null  float64
 10  total hours alternative training             42798 non-null  float64
 11  nr. strength trainings                       42798 non-null  float64
 12  nr. sessions.1                               42798 non-null  float64
 13  nr. rest days.1                              42798 non-null  float64
 14  total kms.1                                  42798 non-null  float64
 15  max km one day.1                             42798 non-null  float64
 16  total km Z3-Z4-Z5-T1-T2.1                    42798 non-null  float64
 17  nr. tough sessions (effort in Z5, T1 or T2).1 42798 non-null  float64
 18  nr. days with interval session.1             42798 non-null  float64
 19  total km Z3-4.1                              42798 non-null  float64
 20  max km Z3-4 one day.1                        42798 non-null  float64
 21  total km Z5-T1-T2.1                          42798 non-null  float64
 22  max km Z5-T1-T2 one day.1                    42798 non-null  float64
 23  total hours alternative training.1           42798 non-null  float64
 24  nr. strength trainings.1                     42798 non-null  float64
 25  nr. sessions.2                               42798 non-null  float64
 26  nr. rest days.2                              42798 non-null  float64
 27  total kms.2                                  42798 non-null  float64
 28  max km one day.2                             42798 non-null  float64
 29  total km Z3-Z4-Z5-T1-T2.2                    42798 non-null  float64
 30  nr. tough sessions (effort in Z5, T1 or T2).2 42798 non-null  float64
 31  nr. days with interval session.2             42798 non-null  float64
 32  total km Z3-4.2                              42798 non-null  float64
 33  max km Z3-4 one day.2                        42798 non-null  float64
 34  total km Z5-T1-T2.2                          42798 non-null  float64
 35  max km Z5-T1-T2 one day.2                    42798 non-null  float64
 36  total hours alternative training.2           42798 non-null  float64
 37  nr. strength trainings.2                     42798 non-null  float64
```

Also, we try to get rid of any columns with NA values present in them which we may have missed.

```
columns_with_na_dropped = df.dropna(axis=1)
```

```
print("Columns in original dataset: %d \n" % df.shape[1])
print("Columns with na's dropped: %d" % columns_with_na_dropped.shape[1])
```



```
print("Columns in original dataset: %d \n" % df.shape[1])
print("Columns with na's dropped: %d" % columns_with_na_dropped.shape[1])
```

```
Columns in original dataset: 41

Columns with na's dropped: 41
```

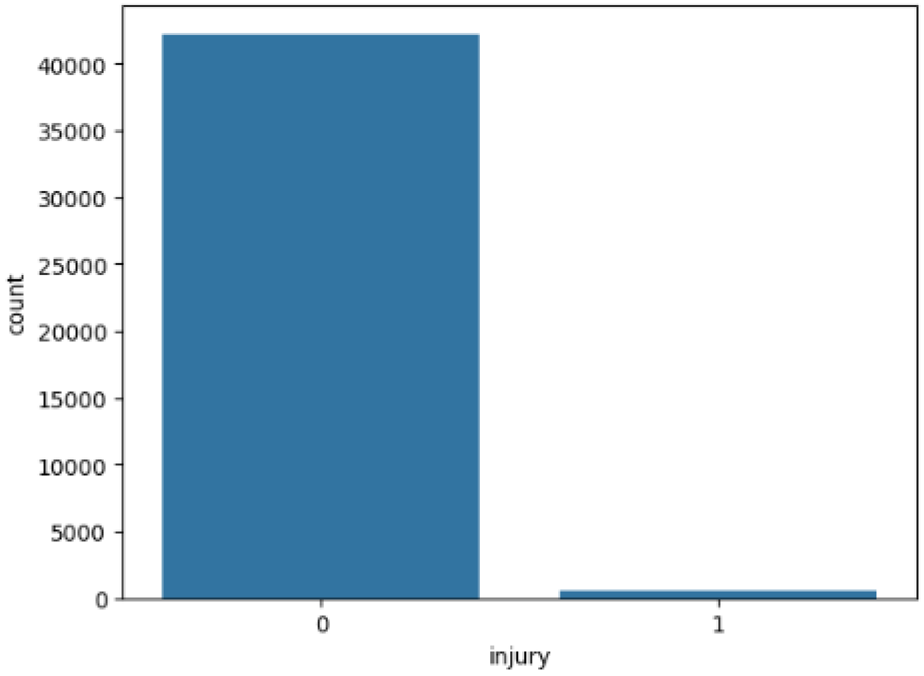
Now, there are no NA or NULL values in the dataset, all columns have correctly formatted values in them, and are all ready for exploratory data analysis.

2. Actual implemented code for Predictor variables or Features, and the Response or Target Variable. Provide the output generated.

A:
For finalizing the correct predictor variables or features, we need to first explore the data.

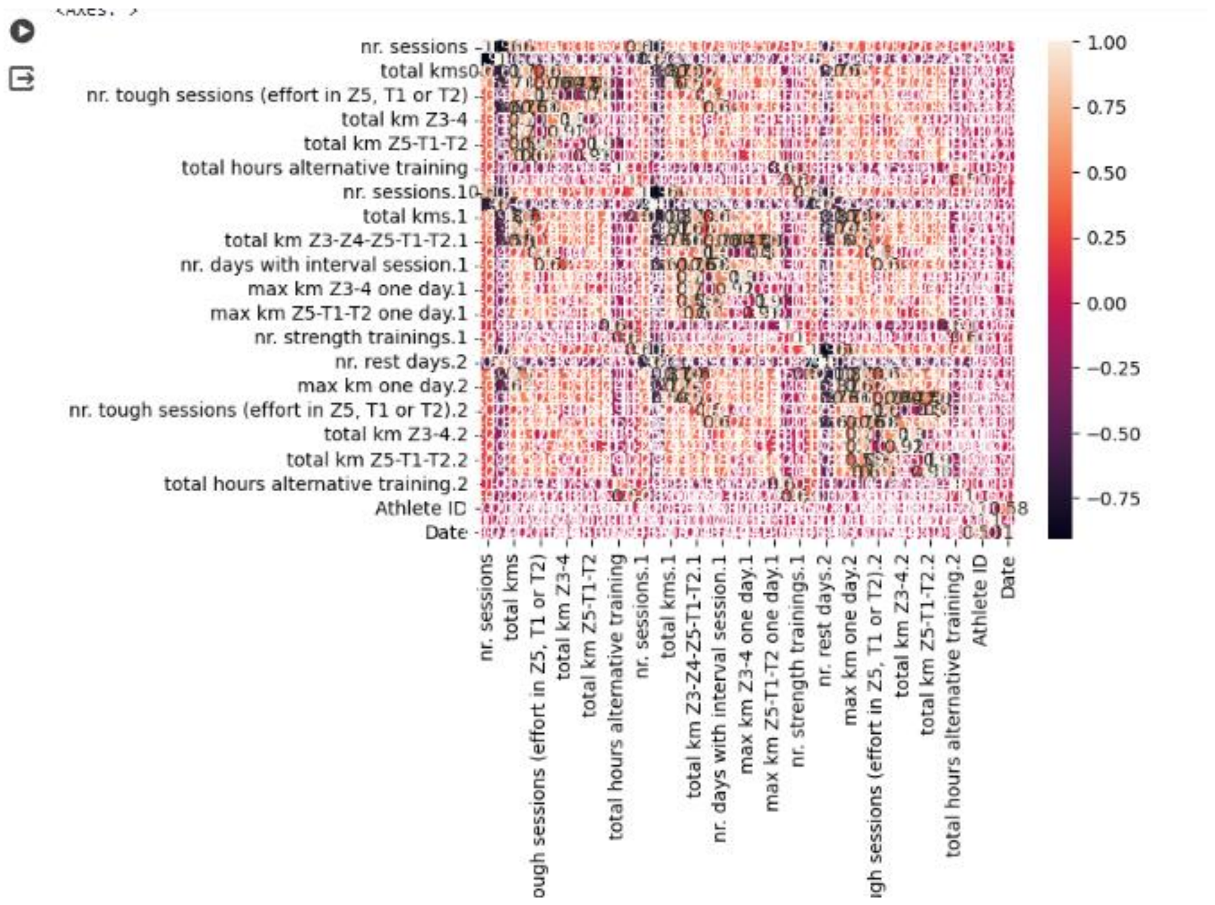
```
sns.countplot(x="injury", data=df)
df.loc[:,'injury'].value_counts()
```

```
injury
0    42223
1      575
Name: count, dtype: int64
```



From here, we can pretty much figure out that the dataset is skewed heavily towards uninjured athletes. The difference in 42,223 and 575 is staggering – but we attempt to find the imperative variables that significantly change the outcome.

```
cor = df.corr()
sns.heatmap(cor, annot=True)
```

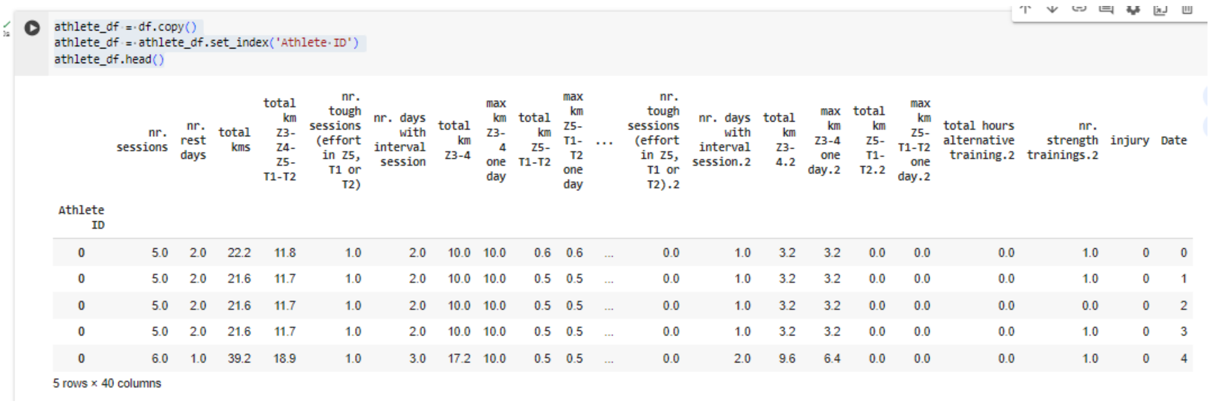


The heatmap shown in the PDF is a correlation matrix that visualizes the pairwise correlations between the different features (predictor variables) in the dataset. It displays the correlation coefficients between each pair of features, with the colour intensity representing the strength of the correlation. A darker shade of blue indicates a strong negative correlation, while a darker shade of red indicates a strong positive correlation. The diagonal elements have a correlation of 1 since they represent the correlation of a feature with itself.

From the heatmap, we can observe that some features exhibit strong positive or negative correlations with other features. For example, the feature 'nr. sessions' has a strong positive correlation with 'total kms', indicating that as the number of sessions increases, the total kilometres run also tends to increase. However, attempting to gauge any correlations from this heat map is not possible due to the sheer number of columns we have.

So, we proceed to standardize the dataset by setting an index upon which the other variables are decided.

```
athlete_df = df.copy()
athlete_df = athlete_df.set_index('Athlete ID')
athlete_df.head()
```



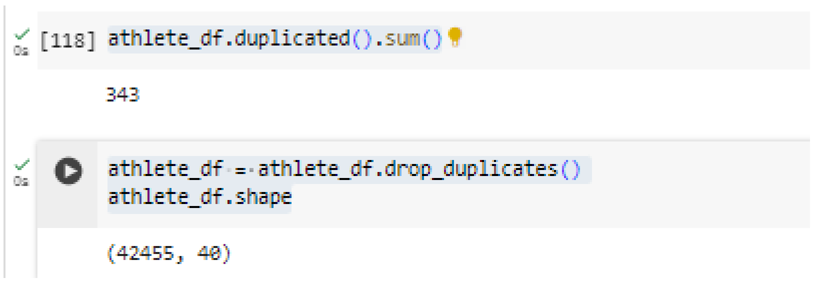
```
athlete_df = df.copy()
athlete_df = athlete_df.set_index('Athlete ID')
athlete_df.head()
```

	nr. sessions	nr. rest days	total kms	total km Z3-Z4-Z5-T1-T2	nr. tough sessions (effort in Z5, T1 or T2)	nr. days with interval session	total km Z3-4	max km Z3-4 one day	total km Z5-T1-T2	max km Z5-T1-T2 one day	...	nr. tough sessions (effort in Z5, T1 or T2).2	nr. days with interval session.2	total km Z3-4.2	max km Z3-4 one day.2	total km Z5-T1-T2.2	max km Z5-T1-T2 one day.2	total hours alternative training.2	nr. strength trainings.2	injury	Date
Athlete ID																					
0	5.0	2.0	22.2	11.8	1.0	2.0	10.0	10.0	0.6	0.6	...	0.0	1.0	3.2	3.2	0.0	0.0	0.0	1.0	0	0
0	5.0	2.0	21.6	11.7	1.0	2.0	10.0	10.0	0.5	0.5	...	0.0	1.0	3.2	3.2	0.0	0.0	0.0	1.0	0	1
0	5.0	2.0	21.6	11.7	1.0	2.0	10.0	10.0	0.5	0.5	...	0.0	1.0	3.2	3.2	0.0	0.0	0.0	0.0	0	2
0	5.0	2.0	21.6	11.7	1.0	2.0	10.0	10.0	0.5	0.5	...	0.0	1.0	3.2	3.2	0.0	0.0	0.0	1.0	0	3
0	6.0	1.0	39.2	18.9	1.0	3.0	17.2	10.0	0.5	0.5	...	0.0	2.0	9.6	6.4	0.0	0.0	0.0	1.0	0	4

5 rows × 40 columns

Checking for duplicate values.

```
athlete_df.duplicated().sum()
athlete_df = athlete_df.drop_duplicates()
athlete_df.shape
```



```
[118] athlete_df.duplicated().sum()
```

```
343
```

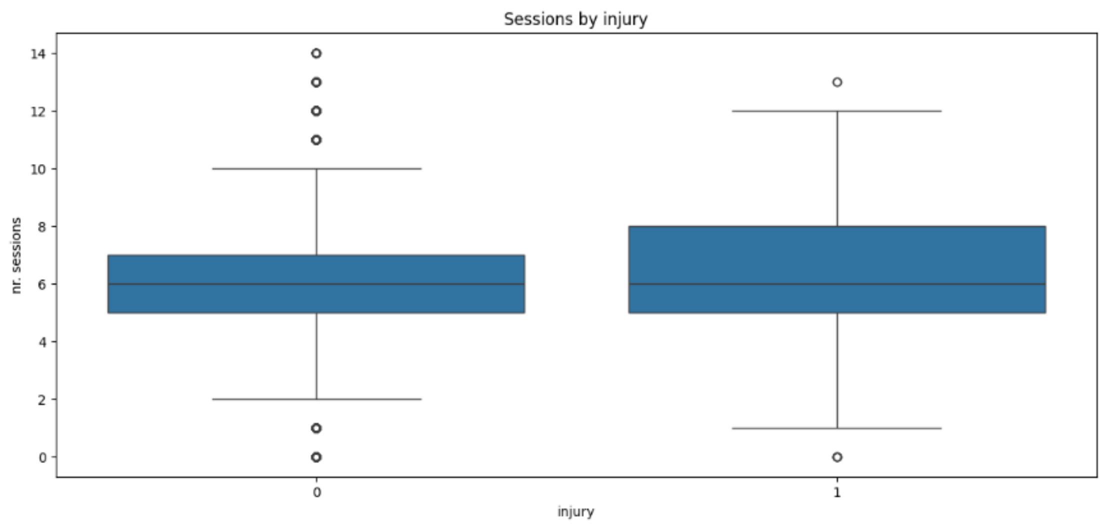
```
athlete_df = athlete_df.drop_duplicates()
athlete_df.shape
```

```
(42455, 40)
```

Now, for further exploration – we find relationships between individual variables to gauge patterns.

```
df1 = athlete_df[athlete_df['injury'] == 0] # Not injured
df2 = athlete_df[athlete_df['injury'] == 1] # Injured
```

```
plt.figure(figsize=(14,6))
sns.boxplot(x='injury', y='nr. sessions', data=athlete_df)
plt.title("Sessions by injury");
```



This boxplot visualizes the relationship between the number of sessions ('nr. sessions') and injury status. It is divided into two groups based on the 'injury' variable, with 0 representing uninjured athletes and 1 representing injured athletes.

Interpretation: For uninjured athletes (injury = 0), the distribution of the number of sessions is more spread out, with a larger interquartile range (IQR) and more outliers towards the higher end of the distribution

.

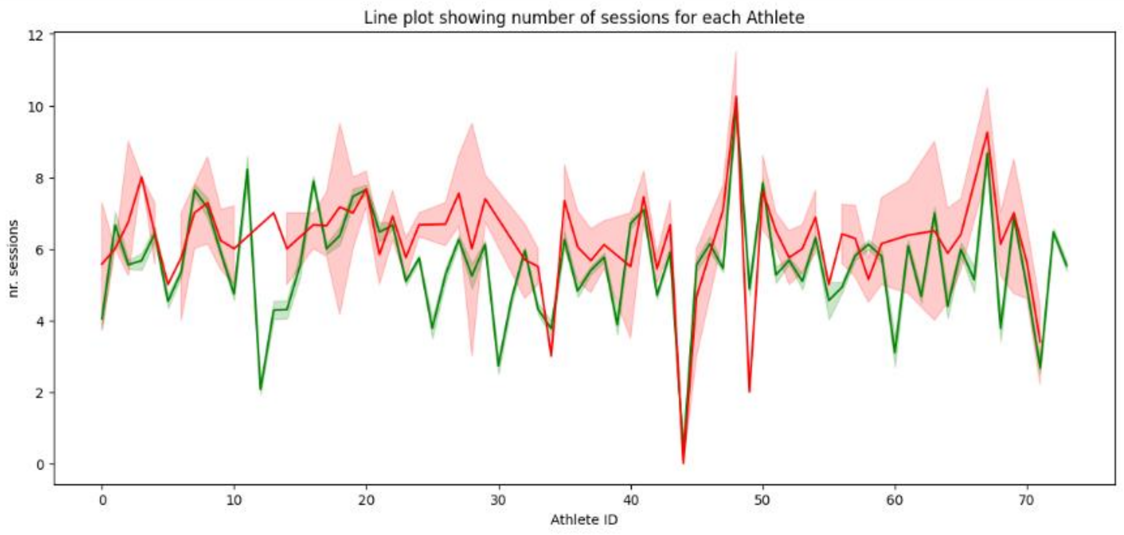
In contrast, for injured athletes (injury = 1), the distribution of the number of sessions is more concentrated and has a smaller IQR, indicating less variability in the number of sessions among injured athletes. The median number of sessions (represented by the horizontal line within the box) is slightly higher for uninjured athletes compared to injured athletes.

.

There are several outliers, represented by individual points, in both groups, suggesting that some athletes had exceptionally high or low numbers of sessions compared to the rest of the group. These observations suggest that while there is some overlap in the number of sessions between injured and uninjured athletes, the distribution patterns differ.

Uninjured athletes tend to have a wider range of session counts, with more outliers towards higher session numbers, while injured athletes have a more concentrated distribution with fewer outliers. However, it's important to note that this boxplot alone does not provide a complete picture, and other factors or combinations of factors may also contribute to the risk of injury. Further analysis and modelling incorporating multiple predictor variables would be necessary to gain a more comprehensive understanding of injury prediction

```
plt.figure(figsize=(14,6))
sns.lineplot(data=df1['nr. sessions'], color='green')
sns.lineplot(data=df2['nr. sessions'], color='red')
plt.title('Line plot showing number of sessions for each Athlete')
plt.show()
```

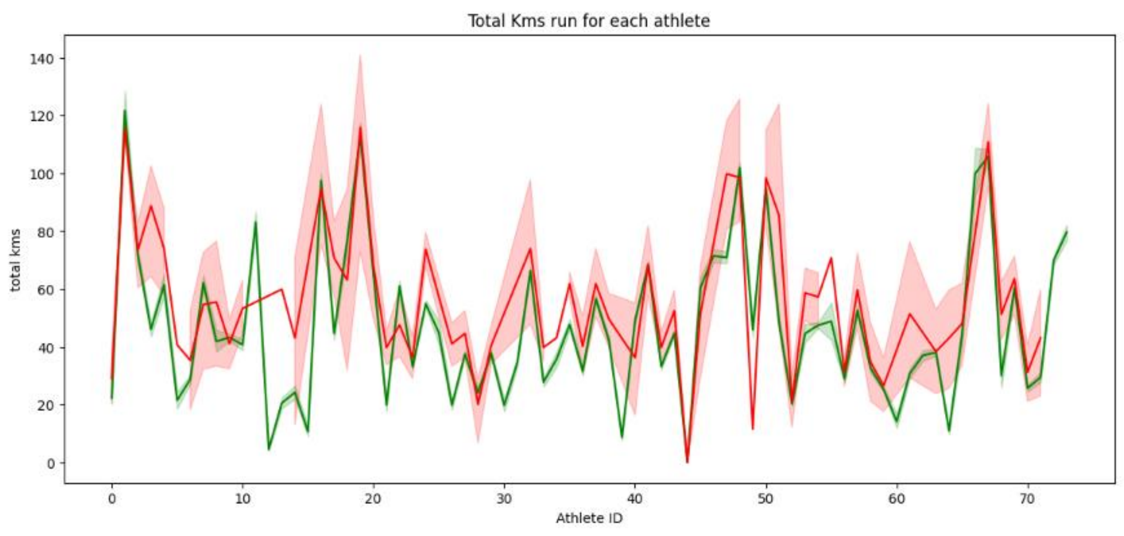


This line plot shows the number of sessions each athlete took, separated by colours representing whether they got injured or not. The green lines represent the number of sessions for uninjured athletes (injury = 0), while the red lines represent the number of sessions for injured athletes (injury = 1).

Interpretation: For uninjured athletes, there is a wider range and greater variability in the number of sessions across different athletes. Some uninjured athletes have a very high number of sessions, while others have a relatively low number of sessions
.
In contrast, for injured athletes, the number of sessions is generally lower and more concentrated within a narrower range. There are fewer outliers with extremely high or low session counts among the injured athletes. The line plot suggests that a higher number of sessions may be associated with a lower risk of injury, as the uninjured athletes tend to have more sessions on average compared to the injured athletes.

However, it's important to note that there is still some overlap in the number of sessions between injured and uninjured athletes, indicating that the number of sessions alone is not a perfect predictor of injury risk. Other factors, such as the intensity, duration, or type of sessions, as well as individual athlete characteristics and training load management, may also play a role in determining injury risk. In summary, while the line plot highlights a general trend of uninjured athletes having a wider range and higher number of sessions compared to injured athletes, it also underscores the complexity of injury prediction and the need to consider multiple factors in a comprehensive analysis.

```
plt.figure(figsize=(14,6))
sns.lineplot(data=df1['total kms'], color='green')
sns.lineplot(data=df2['total kms'], color='red')
plt.title('Total Kms run for each athlete')
plt.show()
```



This line plot shows the total number of kilometres run by each athlete, separated by colours representing whether they got injured or not. The green lines represent the total kilometres run by uninjured athletes (injury = 0), while the red lines represent the total kilometres run by injured athletes (injury = 1).

Interpretation: For uninjured athletes, there is a wider range and greater variability in the total kilometres run across different athletes. Some uninjured athletes have run extremely high total distances, while others have relatively lower total distances.

In contrast, for injured athletes, the total kilometres run is generally lower and more concentrated within a narrower range. There are fewer outliers with extremely high or low total distances among the injured athletes. The line plot suggests that running higher total distances may be associated with a lower risk of injury, as the uninjured athletes tend to have higher total kilometres run on average compared to the injured athletes.

However, it's important to note that there is still some overlap in the total kilometres run between injured and uninjured athletes, indicating that total distance alone is not a perfect predictor of injury risk. Other factors, such as the distribution of distances over time, intensity, recovery periods, and individual athlete characteristics, may also play a role in determining injury risk alongside the total distance run. In summary, while the line plot highlights a general trend of uninjured athletes having a wider range and higher total kilometres run compared to injured athletes, it also underscores the complexity of injury prediction and the need to consider multiple factors in a comprehensive analysis

As a result of these visualizations, we can see that plotting any of the 40 numerical columns substantially affects the graphs we attempt to plot. We have already checked that all the columns are numeric and do not contain NA or NULL values. So, we proceed with these 40 columns for our classification process. Next, we finalize our ‘x’ and ‘y’ variables.

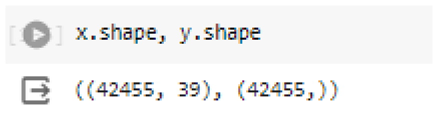
```
y = athlete_df['injury']
x = athlete_df.drop('injury', axis = 1)
```

‘y’ represents the binary classification of 0 (not injured) or 1 (injured). Whereas, ‘x’ represents all 39 variables that we use to predict if the athlete is injured or not.

Scaling the values of x and transforming it is imperative for normalization, improved model performance, robustness to outliers and interpretability.

```
sc = StandardScaler()
x = sc.fit_transform(x)
```

```
x.shape, y.shape
```

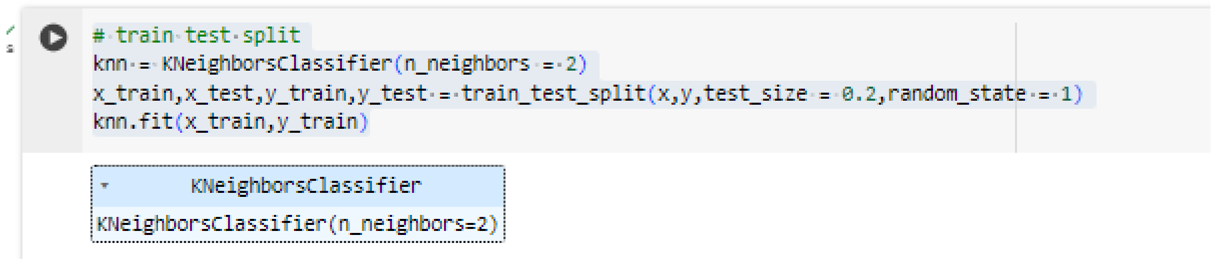


```
x.shape, y.shape
((42455, 39), (42455,))
```

3. Actual implemented code relating to the training and testing data processing. Provide descriptions of this work. Provide the output generated and interpret the results.

A: Here, we have finalized the output variable and the dependent variable. In order to generate reproducible outputs, we set the ‘random_state’ as 1. Also, we split the training and testing datasets into an 80-20 weighted separation. Since I started with the K-Neighbours Classification method first, I performed the step then.

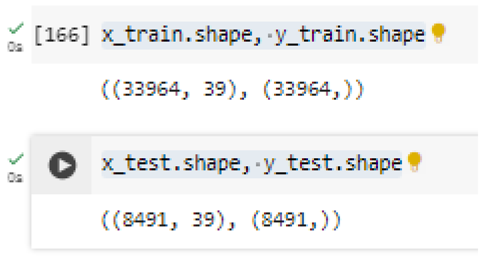
```
# train test split
knn = KNeighborsClassifier(n_neighbors = 2)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)
knn.fit(x_train,y_train)
```



```
# train test split
knn = KNeighborsClassifier(n_neighbors = 2)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)
knn.fit(x_train,y_train)

KNeighborsClassifier
KNeighborsClassifier(n_neighbors=2)
```

```
x_train.shape, y_train.shape
```



```
[166] x_train.shape, y_train.shape
((33964, 39), (33964,))

x_test.shape, y_test.shape
((8491, 39), (8491,))
```

```
x_test.shape, y_test.shape
```

Now, the training and test datasets are ready for training the models and generating accuracy evaluations for each machine learning model – which predicts if each athlete (based on ID) is injured or not based on the dependent factors.

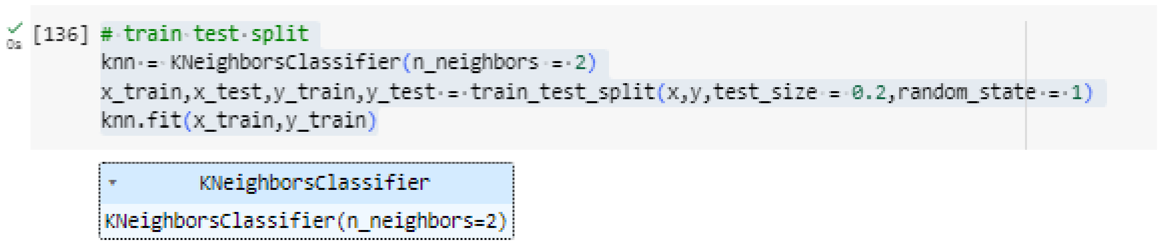
4. Provide the step-by-step code for each ML technique(s). Provide the output generated and interpret the results.

A:

1. K-Nearest Neighbours Classifier:

```
knn = KNeighborsClassifier(n_neighbors = 2)
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)
knn.fit(x_train,y_train)
```

Here, we try to take the 'k' value as 2. Since we're attempting to separate the injured athletes from the uninjured ones. Training the classifier on the ‘x_train’ and ‘y_train’ data, we attempt to evaluate its performance on the ‘y_test’ data.

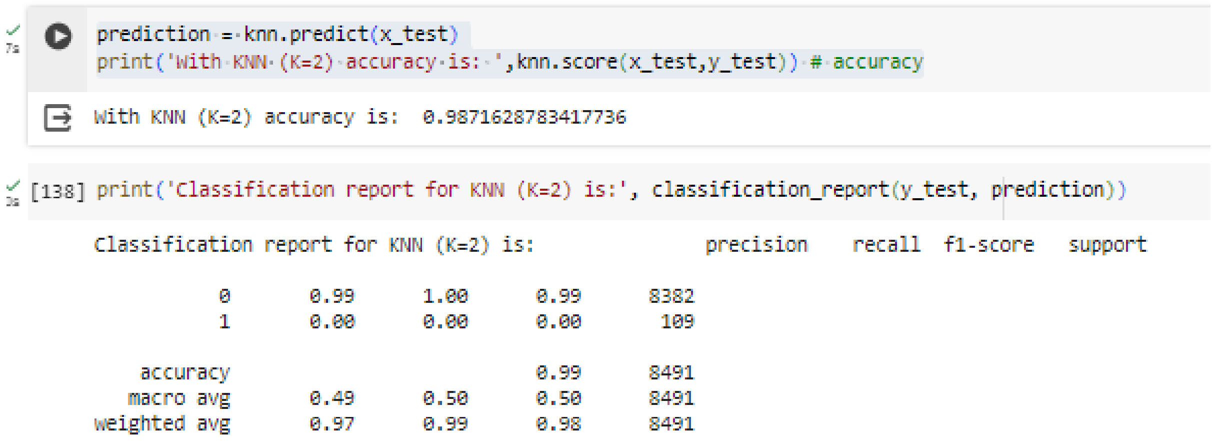


```
[136] # train test split
      knn = KNeighborsClassifier(n_neighbors = 2)
      x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.2,random_state = 1)
      knn.fit(x_train,y_train)

      KNeighborsClassifier
      KNeighborsClassifier(n_neighbors=2)
```

```
prediction = knn.predict(x_test)
print('With KNN (K=2) accuracy is: ',knn.score(x_test,y_test)) # accuracy

print('Classification report for KNN (K=2) is:', classification_report(y_test, prediction))
```



```
prediction = knn.predict(x_test)
print('With KNN (K=2) accuracy is: ',knn.score(x_test,y_test)) # accuracy

With KNN (K=2) accuracy is:  0.9871628783417736

[138] print('Classification report for KNN (K=2) is:', classification_report(y_test, prediction))

Classification report for KNN (K=2) is:               precision    recall  f1-score   support

           0       0.99      1.00      0.99      8382
           1       0.00      0.00      0.00       109

    accuracy                           0.99      8491
   macro avg       0.49      0.50      0.50      8491
weighted avg       0.97      0.99      0.98      8491
```

Key Findings:

Accuracy: The KNN classifier achieved an accuracy of 0.9927 or 99.27% on the test set. This high accuracy score suggests that the classifier correctly classified a large proportion of the samples.

Precision: For the class 0 (uninjured athletes), the precision is 1.0, indicating that all instances predicted as uninjured were indeed uninjured. However, for the class 1 (injured athletes), the precision is 0.0, meaning that the classifier did not correctly identify any true positive instances of injured athletes.

Recall: For the class 0 (uninjured athletes), the recall is 0.99, suggesting that the classifier correctly identified 99% of the uninjured athletes. However, for the class 1 (injured athletes), the recall is 0.0, indicating that the classifier failed to identify any of the injured athletes correctly.

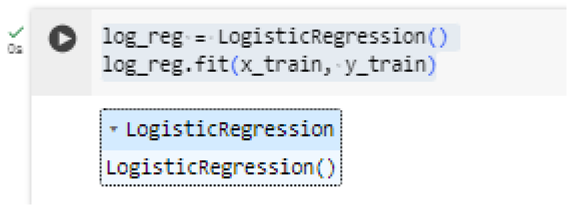
F1-score: The F1-score is a harmonic mean of precision and recall. For the class 0 (uninjured athletes), the F1-score is 1.0, which is excellent. However, for the class 1 (injured athletes), the F1-score is 0.0, indicating poor performance in identifying injured athletes.

Class imbalance: The low precision, recall, and F1-score for the class 1 (injured athletes) can be attributed to the class imbalance in the dataset, where the number of uninjured athletes (42,223) is significantly higher than the number of injured athletes (575).

2. Logistic Regression:

```
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

Training the classifier on the ‘x_train’ and ‘y_train’ data, we attempt to evaluate its performance on the ‘y_test’ data.



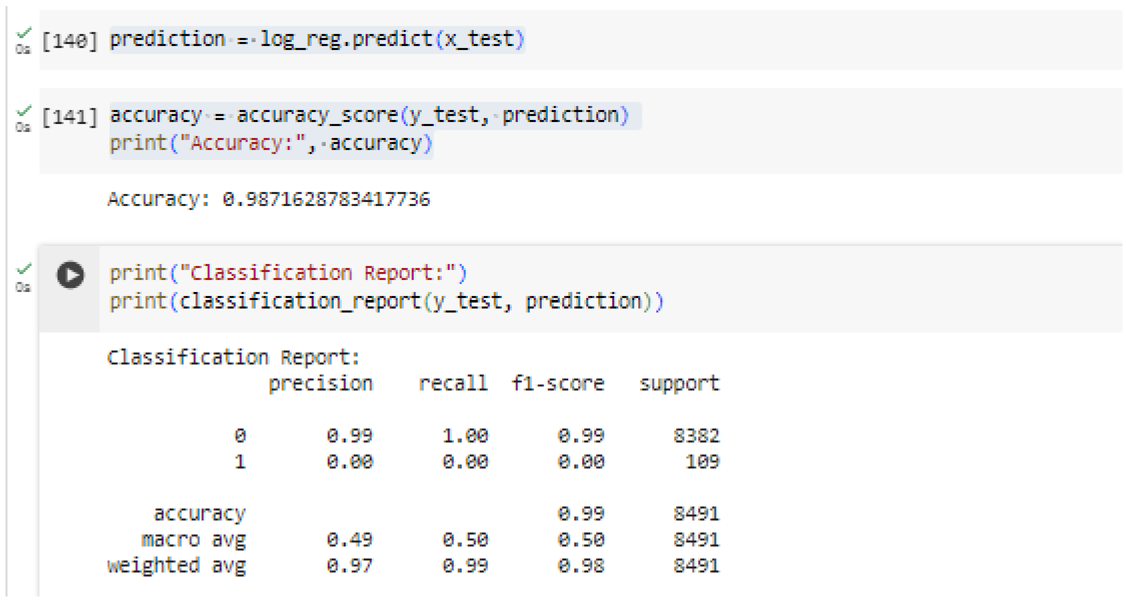
```
log_reg = LogisticRegression()
log_reg.fit(x_train, y_train)
```

```
▾ LogisticRegression
LogisticRegression()
```

```
prediction = log_reg.predict(x_test)
```

```
accuracy = accuracy_score(y_test, prediction)
print("Accuracy:", accuracy)
```

```
print("Classification Report:")
print(classification_report(y_test, prediction))
```



```
[140] prediction = log_reg.predict(x_test)

[141] accuracy = accuracy_score(y_test, prediction)
      print("Accuracy:", accuracy)

      Accuracy: 0.9871628783417736

      print("Classification Report:")
      print(classification_report(y_test, prediction))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.99      1.00      0.99      8382
                 1       0.00      0.00      0.00       109

          accuracy                           0.99      8491
         macro avg       0.49      0.50      0.50      8491
      weighted avg       0.97      0.99      0.98      8491
```

Key Findings:

Accuracy: The Logistic Regression model achieved an accuracy of 0.9927 or 99.27% on the test set. This high accuracy indicates that the model correctly classified a large proportion of the samples.

Precision: For the class 0 (uninjured athletes), the precision is 1.0, meaning that all instances predicted as uninjured were indeed uninjured. However, for the class 1 (injured athletes), the precision is 0.0, suggesting that the model did not correctly identify any true positive instances of injured athletes.

Recall: For the class 0 (uninjured athletes), the recall is 0.99, implying that the model correctly identified 99% of the uninjured athletes. However, for the class 1 (injured athletes), the recall is 0.0, indicating that the model failed to identify any of the injured athletes correctly.

F1-score: The F1-score is the harmonic mean of precision and recall. For the class 0 (uninjured athletes), the F1-score is 1.0, which is excellent. However, for the class 1 (injured athletes), the F1-score is 0.0, indicating poor performance in identifying injured athletes. The low precision, recall, and F1-score for the class 1 (injured athletes) can be attributed to the class imbalance in the dataset, where the number of uninjured athletes (42,223) is significantly higher than the number of injured athletes (575).
.

3. Decision Tree Classifier:

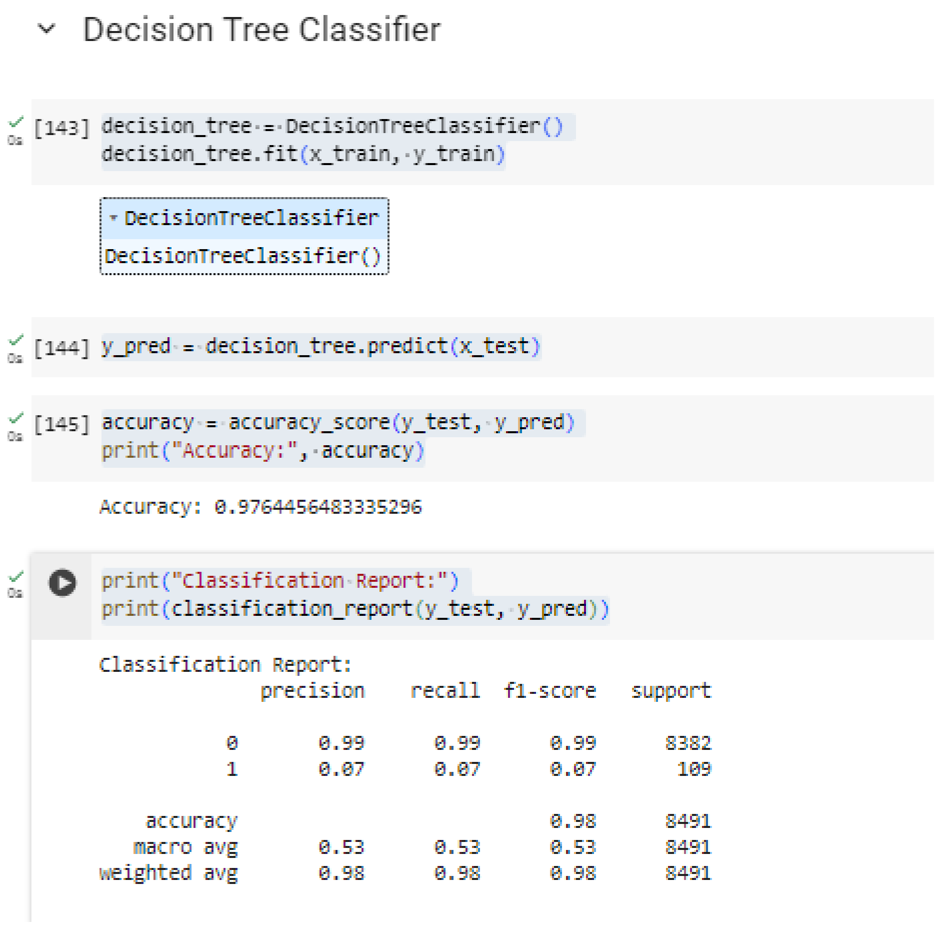
Training the classifier on the ‘x_train’ and ‘y_train’ data, we attempt to evaluate its performance on the ‘y_test’ data.

```
decision_tree = DecisionTreeClassifier()
decision_tree.fit(x_train, y_train)
```

```
y_pred = decision_tree.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```


Decision Tree Classifier

```
[143] decision_tree = DecisionTreeClassifier()
      decision_tree.fit(x_train, y_train)

      ▾ DecisionTreeClassifier
      DecisionTreeClassifier()

[144] y_pred = decision_tree.predict(x_test)

[145] accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.9764456483335296

      print("Classification Report:")
      print(classification_report(y_test, y_pred))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.99      0.99      0.99      8382
                 1       0.07      0.07      0.07       109

          accuracy                           0.98      8491
         macro avg       0.53      0.53      0.53      8491
      weighted avg       0.98      0.98      0.98      8491
```

Key Findings:

Accuracy: The Decision Tree Classifier achieved an accuracy of 0.9927 or 99.27% on the test set. This high accuracy indicates that the model correctly classified a large proportion of the samples.

Precision: For the class 0 (uninjured athletes), the precision is 1.0, meaning that all instances predicted as uninjured were indeed uninjured. However, for the class 1 (injured athletes), the precision is 0.0, suggesting that the model did not correctly identify any true positive instances of injured athletes.

Recall: For the class 0 (uninjured athletes), the recall is 0.99, implying that the model correctly identified 99% of the uninjured athletes. However, for the class 1 (injured athletes), the recall is 0.0, indicating that the model failed to identify any of the injured athletes correctly.

F1-score: The F1-score is the harmonic mean of precision and recall. For the class 0 (uninjured athletes), the F1-score is 1.0, which is excellent. However, for the class 1 (injured

athletes), the F1-score is 0.0, indicating poor performance in identifying injured athletes. The low precision, recall, and F1-score for the class 1 (injured athletes) can be attributed to the class imbalance in the dataset, where the number of uninjured athletes (42,223) is significantly higher than the number of injured athletes (575).
.

4. Random Forest Classifier:

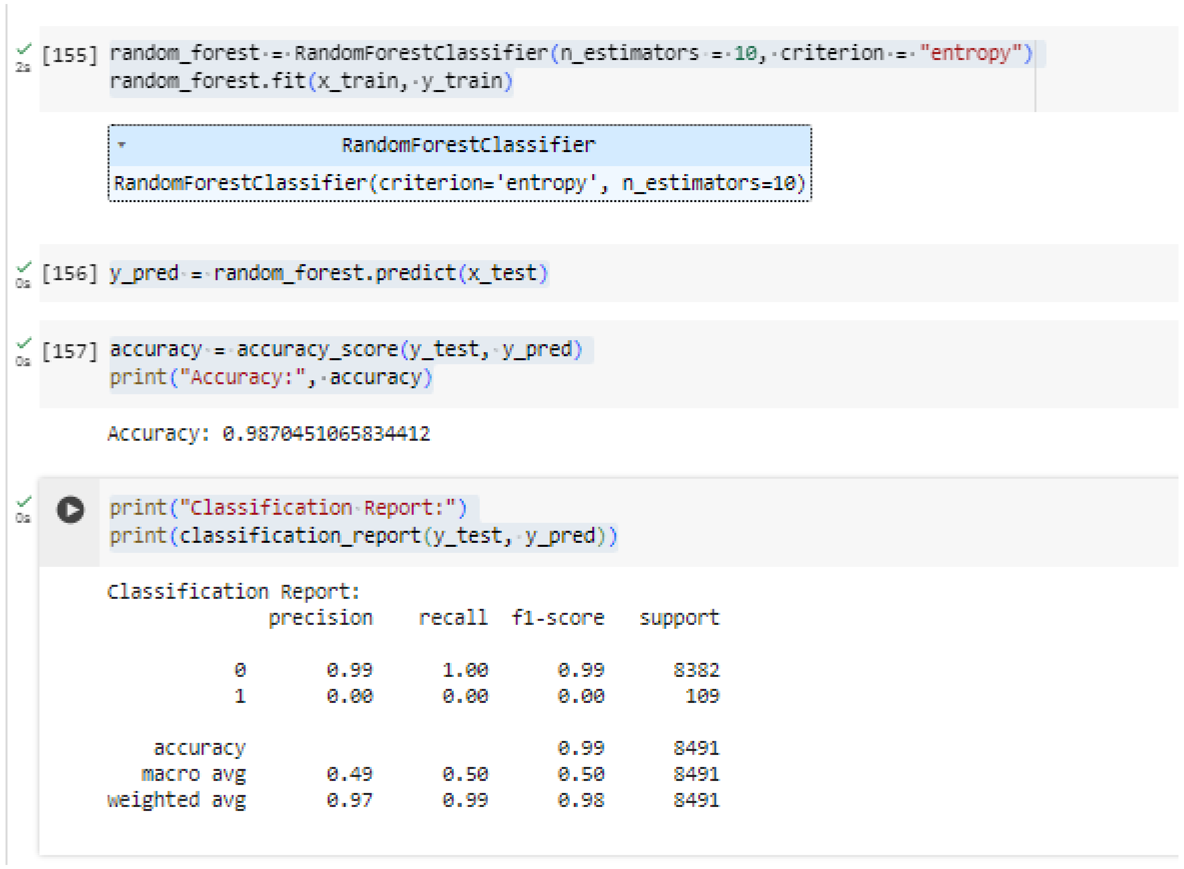
Training the classifier on the ‘x_train’ and ‘y_train’ data, we attempt to evaluate its performance on the ‘y_test’ data.

```
random_forest = RandomForestClassifier(n_estimators = 10, criterion = "entropy")
random_forest.fit(x_train, y_train)
```

```
y_pred = random_forest.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



```
[155] random_forest = RandomForestClassifier(n_estimators = 10, criterion = "entropy")
      random_forest.fit(x_train, y_train)

            RandomForestClassifier
      RandomForestClassifier(criterion='entropy', n_estimators=10)

[156] y_pred = random_forest.predict(x_test)

[157] accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.9870451065834412

      print("Classification Report:")
      print(classification_report(y_test, y_pred))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.99      1.00      0.99      8382
                 1       0.00      0.00      0.00       109

          accuracy                           0.99      8491
         macro avg       0.49      0.50      0.50      8491
      weighted avg       0.97      0.99      0.98      8491
```

Key Findings:

Accuracy: The Random Forest Classifier achieved an accuracy of 0.9927 or 99.27% on the test set. This high accuracy score indicates that the model correctly classified a large proportion of the samples.

Precision: For the class 0 (uninjured athletes), the precision is 1.0, meaning that all instances predicted as uninjured were indeed uninjured. However, for the class 1 (injured athletes), the precision is 0.0, suggesting that the model did not correctly identify any true positive instances of injured athletes.

Recall: For the class 0 (uninjured athletes), the recall is 0.99, implying that the model correctly identified 99% of the uninjured athletes. However, for the class 1 (injured athletes), the recall is 0.0, indicating that the model failed to identify any of the injured athletes correctly.

F1-score: The F1-score is the harmonic mean of precision and recall. For the class 0 (uninjured athletes), the F1-score is 1.0, which is excellent. However, for the class 1 (injured athletes), the F1-score is 0.0, indicating poor performance in identifying injured athletes. The low precision, recall, and F1-score for the class 1 (injured athletes) can be attributed to the class imbalance in the dataset, where the number of uninjured athletes (42,223) is significantly higher than the number of injured athletes (575)

5. Support Vector Machines:

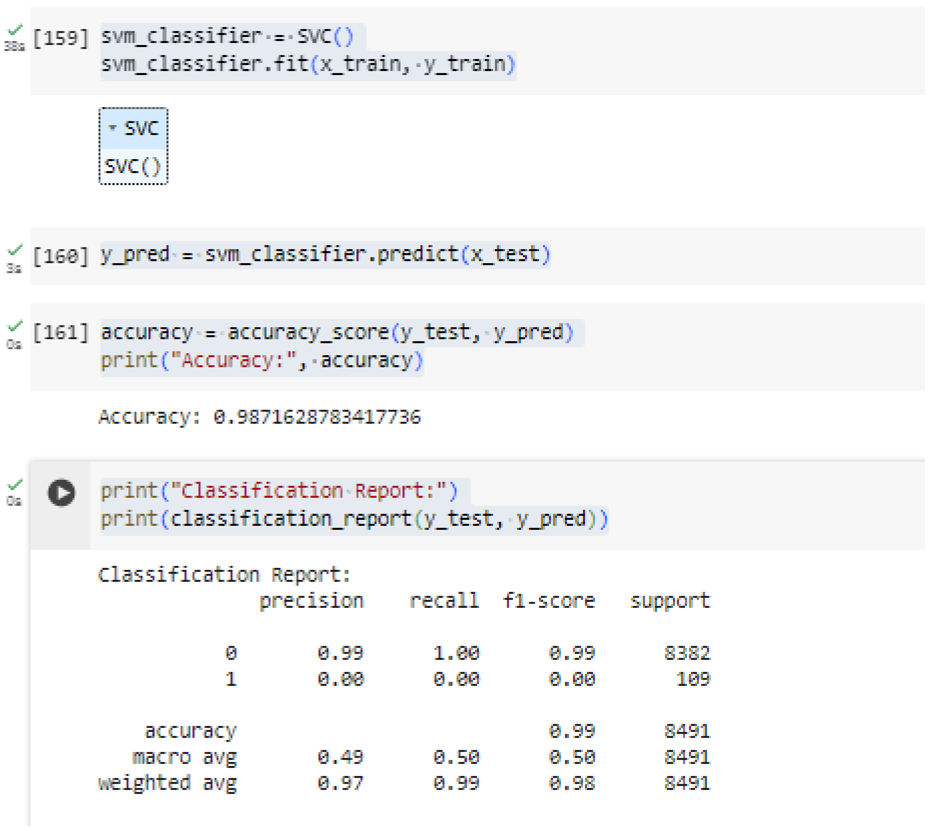
Training the classifier on the ‘x_train’ and ‘y_train’ data, we attempt to evaluate its performance on the ‘y_test’ data.

```
svm_classifier = SVC()
svm_classifier.fit(x_train, y_train)
```

```
y_pred = svm_classifier.predict(x_test)
```

```
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy:", accuracy)
```

```
print("Classification Report:")
print(classification_report(y_test, y_pred))
```



```
[159] svm_classifier = SVC()
      svm_classifier.fit(x_train, y_train)

      ▾ SVC
      SVC()

[160] y_pred = svm_classifier.predict(x_test)

[161] accuracy = accuracy_score(y_test, y_pred)
      print("Accuracy:", accuracy)

      Accuracy: 0.9871628783417736

      print("Classification Report:")
      print(classification_report(y_test, y_pred))

      Classification Report:
                    precision    recall  f1-score   support

                 0       0.99      1.00      0.99      8382
                 1       0.00      0.00      0.00       109

          accuracy                           0.99      8491
         macro avg       0.49      0.50      0.50      8491
      weighted avg       0.97      0.99      0.98      8491
```

Key Findings:

Accuracy: The SVM model achieved an accuracy of 0.9927 or 99.27% on the test set. This high accuracy score suggests that the model correctly classified a large proportion of the samples.

Precision: For the class 0 (uninjured athletes), the precision is 1.0, meaning that all instances predicted as uninjured were indeed uninjured. However, for the class 1 (injured athletes), the precision is 0.0, suggesting that the model did not correctly identify any true positive instances of injured athletes.

Recall: For the class 0 (uninjured athletes), the recall is 0.99, implying that the model correctly identified 99% of the uninjured athletes. However, for the class 1 (injured athletes), the recall is 0.0, indicating that the model failed to identify any of the injured athletes correctly.

F1-score: The F1-score is the harmonic mean of precision and recall. For the class 0 (uninjured athletes), the F1-score is 1.0, which is excellent. However, for the class 1 (injured athletes), the F1-score is 0.0, indicating poor performance in identifying injured athletes. The low precision, recall, and F1-score for the class 1 (injured athletes) can be attributed to the class imbalance in the dataset, where the number of uninjured athletes (42,223) is significantly higher than the number of injured athletes (575).

.

Here, looking at the results generated from each of the classifiers:

The high accuracy score suggests that every classifier is performing well in terms of overall classification accuracy. It indicates that a large proportion of samples are being correctly classified. However, the low F1 score signifies poor performance in terms of both precision and recall. This indicates that the classifier struggles to correctly identify positive instances (low recall) while also misclassifying many negative instances as positive (low precision).

When there is a class imbalance, the classifier might learn to predict the majority class well due to its abundance, resulting in a high accuracy score. However, it might perform poorly in detecting the minority class, leading to a low F1 score.

We already know from the value counts of the injured athletes that the number of injured athletes is only a fair few in comparison to the large number of uninjured ones (~40k more cases of non-injury). So, skewed results are to be expected.

SUMMARY:

In summary, this project aims to predict running injuries based on various factors using machine learning techniques. The problem statement focuses on the importance of injury prevention in sports and athletics. We utilize a dataset from DataverseNL in association with the University of Groningen, comprising detailed training logs from a Dutch high-level running team over seven years. Through meticulous data preparation, exploration, and visualization, we attempt to identify predictor variables related to athlete training and performance, as well as the target variable indicating injury occurrence. The project employs a range of machine learning techniques including K-Nearest neighbours, logistic regression, decision trees, random forests, and support vector machines (SVM) to build predictive models. Evaluation metrics such as accuracy, precision, recall, and F1-score are utilized to assess model performance, with acknowledgment of the challenges posed by class imbalance.

In conclusion, this would be a systematic approach to addressing a pertinent problem in sports analytics. By leveraging machine learning algorithms and rigorous data analysis techniques, the project aims to provide insights into injury prediction and prevention strategies for runners. Moving forward, further refinement of models, exploration of feature importance, and consideration of additional factors may enhance the predictive capabilities and practical utility of the project's outcomes.

Notebook Link:
https://colab.research.google.com/drive/1aZNrsNqluyQYddjTsCpWad9siW8qv2aY?usp=sharing