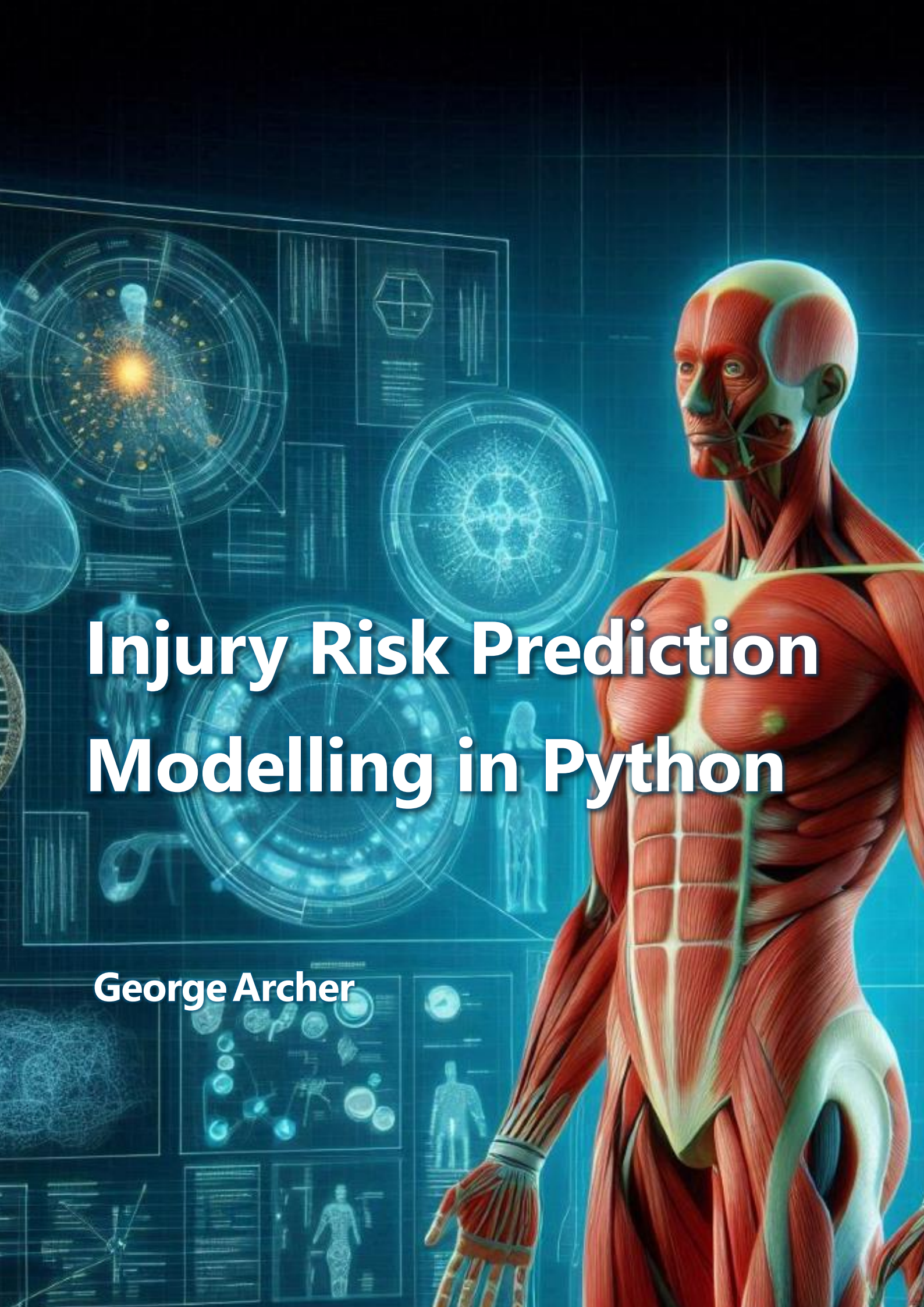


Injury Risk Prediction Modelling in Python

George Archer

MIT License

Contents

Introduction

This file provides a breakdown of the set of classes and functions for the prediction of injury risk of users of Optimi Health app.

Objective

The aim of this set of programs is to construct a process which can take real world data, from Optimi Health, and return classifications of their app users' risk of injury via machine learning algorithms. Recognising the industry-felt shortage of data and general privacy issues, one of the stepping stones for building a predictive model has been a synthetic data generator to allow further data exploration.

Challenge

During the course of this project, the objectives had to be tailored and readjusted with delayed real data receival affecting the scope achievable. As such, artificial data had to be constructed to inform and build the processes required.

Setup instructions

There exist a multitude of python dependencies to run all of the code. As such a requirements file exists. First set the working directory to the location of the scripts (cd "....\Programs". To ensure all the dependencies are installed, in the terminal input paste and run: *pip install -r requirements.txt*

Then run whichever file you want. You can use the sample artificial data file and sample models. Or, run file *create_full_data_record_json* to generate a new random artificial data file.

The python files *model_train / A_sample_on_running_here* provide an example of how a user can use the functionality, training selected or all models on a specified JSON dataset. They also demonstrate how the *injury_predictor* functionality can be used to predict injury for a specified dataset.

Components

To ensure accessibility across devices and potential users of the functionality outlined in this readme, the multi-step process was established in a virtual environment. This means that any packages required for operations of the functions are incumbent installations in the virtual environment.

Table 1.1 provides a brief summary of the key functionality.

Table 1.1 Virtual environment functions

Class/Function	Description
NHANES_data	This class defines functionality to extract real world data from the NHANES database.
create_full_data_record_json	This class defines a set of functions that culminate to create an artificial dataset in place of the Optimi Health dataset.
data_flattening	This function transforms the input data from a JSON file into a more accessible pandas data frame format.
sdv_metadata_generator	Function generates metadata in the format appropriate for the synthetic data vault library, with input data having been passed through the *data_flattening* function.
synth_data_gen_sdv	Function takes the data output from *data_flattening* and respective metadata generated by *sdv_metadata_generator* to then create a synthetic dataset.
distributions	This provides a function which, given data inputs and distribution of a variable, randomly draws a value from desired distribution
EDA_visualisation	This defines a set of functions based on different chart formats, allowing easier consistent format chart creation. Also outputs and saves a set of charts to a designated folder – showing data characteristics.
Custom_snapshot	Takes a dataset with multiple records per user and creates a data snapshot such that all users appear once and the injury incidence meets a certain minimum
ML_models	Runs a set of machine learning models to predict injury
Injury_predict	Functionality for a user to run a trained model and predict injury

NHANES_data:

NHANES_data(construction = 0, file_path_demo = file_demo, file_path_bmx, file_path_d3 = None, file_path_d4 = None):

Overview:

The NHANES_data class is designed to extract data from the NHANES database and merge the different datasets to provide a full user data record.

Why needed:

This functionality was created to aid the development of more realistic artificial data. More specifically, the base functionality enables artificial data to be informed of the covariant relationships between personal statistics such as sex, height, mass and age – rather than random generation allowing for unrealistic height and mass combinations.

The process:

The NHANES 2017-18 data contains interview and examination data for 8704 persons residing in the United States, in order to provide high quality health data (NCHS, 2018b). The database splits datasets into multiple xpt files, meaning that certain characteristics are split among data files.

Class functions merge the two base data files "BMX_J.XPT" and "DEMO_J.XPT" on an ID variable, and keeps only relevant variables, notably age, sex, height and weight to construct personal statistics. (Note: The base data is collected for the period 2017-2018)

The functionality of the class also permits a user some personalisation, allowing them to define and merge other datasets from the NHANES database, and keep certain variables, at their own discretion (via d3 and d4). This is permittable via a binary input variable 'construction', where a value of 0 means the base dataset is extracted and created. Meanwhile a value of 1 allows for user customisation.

Dependencies: NHANES data files (NCHS, 2018a; 2018c; 2018d)

Data_flattening:

Data_flattening(data_file_path)

Overview:

The purpose of this function is to process the raw json data file provided by Optimi Health and to re-design the data structure from nested dictionaries to a pandas data frame format.

Why needed:

This functionality is needed to ensure that the data is in a format accessible to pre-defined python libraries, and to allow easier accessibility of aggregated data for inspection and visualisation. For example, the synthetic data vault library requires data in the format provided by the *data_flattening* function.

Process:

For each user in the raw json data input file, the functionality reallocates each user record and respective data value to among new data frames resulting in a dictionary containing the data frame names and actual data, i.e. df_set = {'primary' : primary_df, 'training_load' : training_load_df,...}

Dependencies: Inputted JSON data file in standard format

distributions:

distribution(low_limit, upper_limit, dist='right')

Overview:

This function is designed to make the random value assignment of data variables more meaningful rather than assuming a more simplistic distribution.

Why needed:

As a result of real data not surfacing this has necessitated the creation of some form of data in order to construct subsequent data processes and retrieve an output of some kind. Hence, given paucity of similar data online, some of the specific variables have to be generated via random assignment between bounds or from a set list. Given the variety of data values, data variables likely possess different distributions, for example the RPE/intensity measure on a 0-10 scale likely requires a left-skew distribution given that the nature of training involves physical exertion and few likely scoring in the 0-2 range.

Process:

The function requires 3 inputs, the bounds [low_limit, upper_limit, and the type of distribution the data variable is believed to be characterised. There are five different options for the distribution:

- 'right' : right-skewed distribution
- 'left": left-skewed distribution
- 'uniform': uniform distribution
- 'symmetric': symmetric distribution
- 'dec_exp': decreasing exponential distribution

The majority of these are constructed via use of the Beta distribution and different specifications of alpha and beta parameters to design a generic shape of the desired distribution. In this way, can assign random values in a slightly more realistic manner. For instance, it is common belief that the share of the population with x number of injuries exponentially decreases as x increases, hence use if 'dec_exp'.

This functionality is instrumental in generating artificial data within the *create_full_data_record_json* class.

Create_full_data_record_json:

Create_full_data_record_json(rand_seed=0, dist=0)

Overview:

This class is designed to bundle together data and functionality which work together to create different instances of artificial data, based off a random seed assignment.

Why needed:

As a result of real data not surfacing this has necessitated the creation of some form of data in order to construct and evaluate subsequent data processes as well as attain an output of some kind. Additionally, due to the nature of certain variables not being mutually exclusive, and distributions not being uniform, data creation has to be thoughtfully crafted to ensure some inference rather than just full random generation.

Process:

The class has multiple functions in order to produce the end result of informed artificial data matching the style of Optimi Health data.

Initiation(rand_seed=0, dist=0):

- Initiates the class calling the default dataset from the NHANES data involving a set of records regarding age, weight, height and gender
- Establishes the random seed and whether using distribution properties or actual data for primary_stats, dist=0 means just attach other generated data to the real NHANES data subset while dist=1 means attain distribution properties from NHANES data and attach.

create_demo_stats(user = None):

- Used if dist = 1 to review distribution of age, sex, weight and height and then assign values to a user based on these distributions and interdependent relations

create_training_load_acwr(user = None, signup_date = None, training_practice = {'Cycling': 2, 'Tennis' : 2, 'Hiking' : 1}, data_format = 'split'):

- Takes a randomly generated list of routine activities and their respective weekly frequency for a single user, in the *training_practice* input, and user sign up date to the app to create a time series record of the user's training routine since use of the app
- Assumes that for each day the user has a set of routine/fixed exercises they perform each week i.e. have football / cricket match play on Saturdays or practice/lessons on Tuesdays every week.
- From this generates a historical record of activity with randomly generated exercise duration and RPE / intensity where the RPE is assumed to follow a left-skewed distribution. RPE varies per week, while session duration is assumed constant week-to-week due to routine nature. This values allow the calculation of the acute-chronic workload ratio (ACWR)

- To add additional randomness a binary determinator has been added to determine if the user is doing the exercises or not. This 'prob_routine' is to account for sickness, other activities and general life events.
- Similarly, there is the potential to incorporate a "spontaneous activity', i.e. friends want to go kayaking this weekend. For now, this has not been included but can be applied in a similar fashion.
- The function returns two datasets, either as individual datasets if 'data_format' is specified as 'split' or as a singular dataset if specified as 'joined'. The 'split' format availability is due to the proposed data structure separating training load and ACWR data.

create_injury_history(user=None, signup_date = None, age = None):

- This randomly generates a user's injury history, specifying a body part from a random uniform distribution and specifying time considerate injury start and end dates.

create_wearable_data(user=None):

- Uses academic study analysis of resting heart rate to provide distributional data (Galarnyk et al., 2020)
- Uses data from NHIS survey to randomly select from for sleep duration (NCHS, 2018a)

create_forms_of_training(user, number_of_methods = 2, sessions_pw=2):

- Randomly selects sample of physical activity methods (based off number_of_methods input, which is determined by the user type) and assigns frequency per week of this routine activity given user type bounds on minimum and maximum number of sessions a week.

create_user_data(num_users=10):

- Collates all data to generate records for the number of user (num_users) specified in the function input.
- Generates user ids and iterates through the users, assigning a user type (routine, specialist or infrequent) which determines some base user characteristics to then iterate through the prior functions mentioned
- Returns a dictionary of the pandas dataframes containing all artificial data

preprocess_data_for_json(data):

- Important to ensure the artificially generated data from *create_user_data()* function is in JSON format with datatypes being transferred to data formats

export_rand_data_as_json(num_participants = 500, filename = 'artificial_injury_data_sample'):

- Utilises the *create_user_data()* function to create a dataset with *num_participants* users and respective records
- It then formats the artificial dataset using the *preprocess_data_for_json()* function and exports the JSON to the respective *filename* in the working directory.

sdv_metadata_generator:

sdv_metadata_generator(flattened_data_dict)

Overview:

This function aims to generate the metadata from the raw data file, following appropriate formatting done by the *data_flattening()* function.

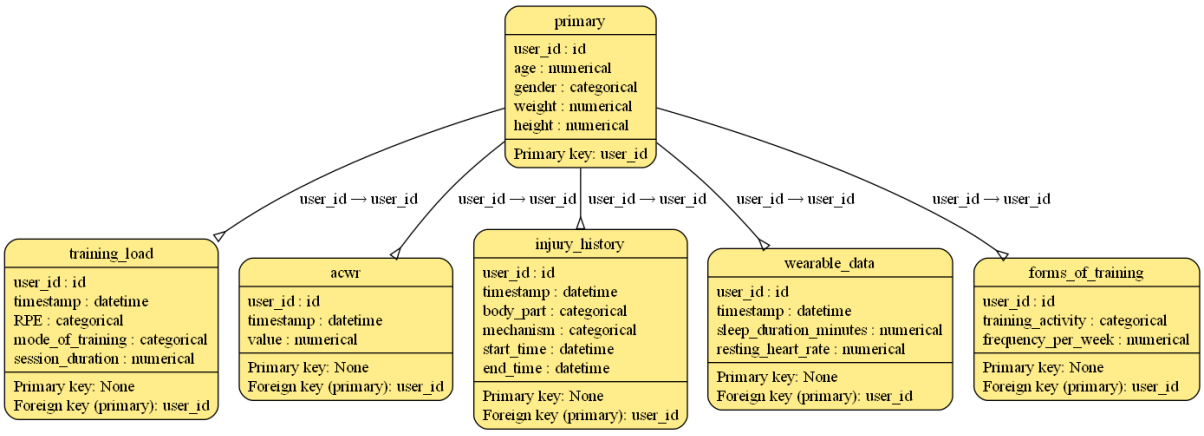
Why needed:

In order to utilise the python library *Synthetic Data Vault (SDV),* and run a synthetic data generator for a multi-table relational dataset, two inputs are needed: the metadata describing each table and the relationships, and secondly the dataset in the format of a dictionary mapping table names to dataframes (Datacebo, 2023a)

This function generates the metadata format, and specifies the library specific data type, known as sdtypes, for the variables.

Process:

The python library provides its own functionality to help with the process, automatically translating the incumbent data types to its own data types. Creating an instance of the MultiTableMetadata class and using the "detect_from_dataframes" function performs this for its users (Datacebo, 2023b). Some manual adjustments such as ensuring ID variables are classified as such are required, and to aid understanding a graphic illustrating relationships is constructed, as below, and saved in the working directory, while a JSON of the metadata is also exported.



synth_data_gen_sdv:

synth_data_gen_sdv(dataset_file_path)

Overview:

This function aims to take a JSON file as an input and return a synthetic dataset, anonymising the data and scaling the dataset allowing more records to be generated.

Why needed:

This functionality is useful in the modelling space as it allows the creation of realistic data. This has potential to allow greater deep learning models and the construction of neural networks. Hence, the synthetic data generator can permit the creation of more complex models with higher predictive power and greater accuracy.

Process:

There are three main steps to the synthetic data generator, however a preliminary step is to process the input file into an appropriate dataframe format via the *data_flattening()* function and generate the respective metadata via the *sdv_metadata_generator()* function.

The base steps are:

1) Creation/initiation of the synthesiser using the metadata
2) Training of the synthesiser using the "real" data
3) And finally, the generation of the synthetic data based on defined parameters

The synthetic data vault library provided an appropriate means for injury prediction due to its ability to accommodate relational databases via its multitable functionality. The library has four synthesisers available but only the Hierarchical Modelling Algorithm (HMA) synthesiser is freely available (Datacebo, 2023c).

EDA_visualisation:

Overview:

This function aims to provide data visualisations of the whole dataset.

Why needed:

This functionality helps an individual understand the dataset, especially the artificial dataset providing visual access to key metrics and distributions of data.

custom_snapshot:

Overview:

Due to significant class imbalances with injury positive instances being a minority snapshot analysis was not plausible due to insufficient instances for ADASYN to generate additional records. Functionality produces a custom snapshot by collating data from other sources.

Why needed:

This functionality is essential to ensure enough positive instances exist in the dataset to train the machine learning models.

ML_models:

ML_models(dataset = input_data, outcome_var = 'injury_imminent', min_instances = 400, min_positive_instances = 0.04, rand_seed_for_snapshot_gen = 0, directory=save_file_path)

Overview:

The script is designed to train and evaluate a variety of machine learning models to predict the injury imminent classification of data instances. The models include a baseline, custom OneR, decision tree, random forest and support vector machine.

The script offers flexibility in model selection and utilises hyperparameter tuning, cost matrix and stratified cross validation to improve the models and provide sufficient evaluation via its custom metrics. To address data paucity and class imbalances, it automatically applies ADASYN and provides synthetic data generation options.

It automatically creates directories to save details on the data analysed, models predicted, including graphics and evaluation metrics with appropriate timestamps.

Why needed:

The script provides the key analysis for the whole project for which all prior scripts detailed have been produced for and predicts the injury risk of individuals.

Process:

The class has a set of functions defined within it:

Function	Description
snapshots_suitable()	Assesses if any naturally-occurring data windows contain sufficient number of positive cases
data_preprocessing(data_snopshot, vars_to_remove, test)	Simply processes the data, separating x and y variables, splitting data into train and test sets, and applying ADASYN to the training set only
abstract_dataset(injury_incidence, rand_seed)	If no naturally-occurring data windows according to snapshots_suitables, abstract_dataset treats time as irrelevant and generates a data window by pulling data for each user from across the dataset – ensuring there is a minimum injury_incidence ratio
eval_metric(y_test, y_pred, y_pred_proba, *, class_names)	Functionality to return the evaluation metrics for predictive models. It returns a host of metrics, namely recall, precision, f1-score, f2-score, AUC-ROC and specification numbers.
Scv(	Conducts split number of fold stratified cross validation to properly assess the model's predictive power.

x_train, y_train, model, splits = 5, rand_state = 7)	
hp_tuning_gs(model, x_train, y_train, param_grid, num_folds_cv = 5, scoring = 'f2')	For inputted model type, with predefined model-specific parameter grid, the function finds the optimal set of hyperparameters to maximise the recall score of the model. It does this by using the experimental HalvingGridSearchCV to minimise computational resources while sustaining likelihood of finding the optimal parameters.
baseline	Uses stratified technique to train a model
oner_custom	Based off the OneR algorithm, single level decision tree
decision_tree	Hyperparameter tuned decision tree
random_forest	Ensemble method, tuned
svm	
desc_stats	Generates some base descriptive statistics
model	Allows the choice of model to be run, or all, trains and creates subsequent graphics and saves all details to created directory
Predict(trained_model, raw_json_file, evaluate=0)	Takes the trained model in form of a joblib file, and using this predicts the injury risk of instances in the raw_json_file

Most of the functionality is pre-defined, to see how to run code and experiment see the example functionality:

```
if __name__ == "__main__":
    root_dir = os.path.dirname(os.path.dirname(os.getcwd()))
    data_wkd = os.path.join(root_dir, 'Artificial data')
    filename = "artificial_injury_data_sample.json"
    filepath = os.path.join(data_wkd,filename)

    input_data = single_data_record(filepath, use_synth_data=0) # increase dataset so at least 1000 records

    test = ML_models(dataset=input_data, outcome_var='injury_imminent')

    test.model('all', 'custom', hypertune = 1, class_weights_var='b')
```

The above code uses the input data from *filepath*, it then processes this data allowing for the generation of synthetic data. Then *ML_models* is initiated and assigned to *test* identifying the outcome variable for prediction as 'injury_imminent.

Then all models are run on the data, using a custom data snapshot, ensuring the models' hyperparameters are tuned and class weights balanced. With the use of grid searching this process of generating the optimal models and their parameters can take over two hours – specifically due to the support vector machine (other models trained faster).

Injury_predictor

Overview:

This function aims to take a JSON file as an input and using a trained model predict the injury risk of each user.

Why needed:

This functionality is useful in providing an easily accessible for the non-creator to use.

Reference List:

1. National Center for Health Statistics (NCHS), 2018a. *National Health Interview Survey (NHIS) 2017 Data Release: Sample Adult file, CSV data* [Online]. National Center for Health Statistics. Available from: https://www.cdc.gov/nchs/nhis/nhis_2017_data_release.htm [Accessed 23 August 2024].

2. National Center for Health Statistics (NCHS), 2018b. *National Health and Nutrition Examination Survey (NHANES) 2017-2018* [Online]. National Center for Health Statistics. Available from: https://wwwn.cdc.gov/nchs/nhanes/continuousnhanes/default.aspx?BeginYear=2017 [Accessed 14 August 2024].

3. National Center for Health Statistics (NCHS), 2020a. *National Health and Nutrition Examination Survey (NHANES) 2017-2018 Data Documentation, Codebook, and Frequencies – Demographic Variables and Sample Weights (DEMO_J)* [Online]. National Center for Health Statistics. Available from: https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/DEMO_J.htm [Accessed 14 August 2024].

4. National Center for Health Statistics (NCHS), 2020b. *National Health and Nutrition Examination Survey (NHANES) 2017-2018 Data Documentation, Codebook, and Frequencies – Body Measures (BMX_J)* [Online]. National Center for Health Statistics. Available from: https://wwwn.cdc.gov/Nchs/Nhanes/2017-2018/BMX_J.htm [Accessed 14 August 2024].

5. National Center for Health Statistics (NCHS), 2018a. *National Health Interview Survey (NHIS) 2017 Data Release: Sample Adult file, CSV data* [Online]. National Center for Health Statistics. Available from: https://www.cdc.gov/nchs/nhis/nhis_2017_data_release.htm [Accessed 23 August 2024].

6. Galarnyk, M., Gouda, P., Quer, G., Steinhubi, S.R. and Topol, E.J., 2020. Inter- and intraindividual variability in daily resting heart rate and its associations with age, sex, sleep, BMI, and time of year: Retrospective, longitudinal cohort study of 92,457 adults. *PLOS One* [Online], 15(2). Available from: https://www.ncbi.nlm.nih.gov/pmc/articles/PMC7001906/ [Accessed 23 August 2024].

7. Datacebo, 2023a. *Synthetic Data Vault: Multi-table-data – Data Preparation* [Online]. Available from: https://docs.sdv.dev/sdv/multi-table-data/data-preparation [Accessed 18 August 2024].

8. Datacebo, 2023a. *Synthetic Data Vault: Multi-table-data – Data Preparation Metadata API* [Online]. Available from: https://docs.sdv.dev/sdv/multi-table-data/data-preparation/multi-table-metadata-api [Accessed 18 August 2024].

9. Datacebo, 2023c. *Synthetic Data Vault: Synthesisers* [Online]. Available from: https://docs.sdv.dev/sdv/multi-table-data/modeling/synthesizers [Accessed 18 August 2024].