 Documentation

**Date:** 09 December 2022 **Version:** 2.0.5 **Author:** pardonaccelerator@gmail.com

Pardon is a Python library developed as a machine learning (ML) and data transformation accelerator, to rapidly prepare data as well as test and deploy supervised ML models, by massively reducing the amount of code required during development.

The library can be used to perform data cleansing and transformation, as well as model and feature selection, data visualisations, model and data version control, and prediction auditing. Data can be quickly prepared for use with ML algorithms, and all data transformations are recorded in a 'script'. This 'script' is then automatically applied to data prior to predictions, which ensures data consistency, transparency, and removes the need for multi-stage data engineering while reducing the chances of error.

The intention of this library is not to replace or automate the work of a data scientist, but to make the core features of data science work more accessible, and quicker to use.

The pardon library functionality includes:

* Data transformation
* Data cleansing
* Feature selection
* Feature engineering
* Model selection
* Class balancing
* Prediction auditing
* Principal Component Analysis
* Data scaling
* Data encoding
* Sentiment analysis
* Hyperparameter tuning
* Model “explainability”
* Model performance metrics
* Model training, testing and validation
* Identifying data outliers
* Testing model output in a REST API
* Data visualisation
* Use unsupervised learning to cluster data

[pardon Process Overview 8](#_Toc117584213)

[Installation 9](#_Toc117584214)

[pip installation 9](#_Toc117584215)

[Adding pardon with a conda dependencies file 10](#_Toc117584216)

[Creating the pardon .whl file 11](#_Toc117584217)

[Creating a virtual environment 12](#_Toc117584218)

[With pip 12](#_Toc117584219)

[With conda 12](#_Toc117584220)

[Installing dependencies using requirements.txt 13](#_Toc117584221)

[Available Models 14](#_Toc117584222)

[Regression Models 14](#_Toc117584223)

[Classification Models 14](#_Toc117584224)

[Clustering Models 14](#_Toc117584225)

[Classes 15](#_Toc117584226)

[pardon.Pardon 15](#_Toc117584227)

[pardon.Pardon Attributes 17](#_Toc117584228)

[test\_size 17](#_Toc117584229)

[target 17](#_Toc117584230)

[scaler\_used 17](#_Toc117584231)

[model\_trained\_date 17](#_Toc117584232)

[model 17](#_Toc117584233)

[model\_type 17](#_Toc117584234)

[best\_hyperparameters 17](#_Toc117584235)

[validation\_training\_time\_mins 17](#_Toc117584236)

[model\_training\_time\_mins 17](#_Toc117584237)

[class\_labels 17](#_Toc117584238)

[failed\_models 17](#_Toc117584239)

[model\_unique\_identifier 17](#_Toc117584240)

[rapid\_ml\_scores 17](#_Toc117584241)

[rapid\_ml\_score\_metric 18](#_Toc117584242)

[best\_validation\_score 18](#_Toc117584243)

[model\_train\_scores 18](#_Toc117584244)

[model\_test\_scores 18](#_Toc117584245)

[model\_eval\_metric 18](#_Toc117584246)

[data 18](#_Toc117584247)

[raw\_data 18](#_Toc117584248)

[train\_data 18](#_Toc117584249)

[test\_data 18](#_Toc117584250)

[columns 18](#_Toc117584251)

[raw\_data\_columns 18](#_Toc117584252)

[class\_distribution 18](#_Toc117584253)

[train\_class\_distribution 18](#_Toc117584254)

[test\_class\_distribution 19](#_Toc117584255)

[model\_fullpath 19](#_Toc117584256)

[input\_data\_fullpath 19](#_Toc117584257)

[prediction\_audit\_fullpaths 19](#_Toc117584258)

[fail\_ons 19](#_Toc117584259)

[config 19](#_Toc117584260)

[pardon.Pardon Methods 20](#_Toc117584261)

[pardon.Pardon.available\_models 20](#_Toc117584262)

[pardon.Pardon.available\_evaluation\_metrics 21](#_Toc117584263)

[pardon.Pardon.available\_chart\_types 22](#_Toc117584264)

[pardon.Pardon.available\_scoring\_metrics 23](#_Toc117584265)

[pardon.Pardon.available\_encoding\_columns 24](#_Toc117584266)

[pardon.Pardon.add\_to\_encoded 25](#_Toc117584267)

[pardon.Pardon.get\_best\_features 26](#_Toc117584268)

[pardon.Pardon.set\_best\_features 27](#_Toc117584269)

[pardon.Pardon.map\_data 28](#_Toc117584270)

[pardon.Pardon.row\_count 29](#_Toc117584271)

[pardon.Pardon.column\_count 30](#_Toc117584272)

[pardon.Pardon.null\_count 31](#_Toc117584273)

[pardon.Pardon.transformation\_count 32](#_Toc117584274)

[pardon.Pardon.number\_of\_classes 33](#_Toc117584275)

[pardon.Pardon.classes 34](#_Toc117584276)

[pardon.Pardon.data\_info 35](#_Toc117584277)

[pardon.Pardon.data\_types 36](#_Toc117584278)

[pardon.Pardon.remove\_unhelpful\_columns 37](#_Toc117584279)

[pardon.Pardon.unhelpful\_columns 38](#_Toc117584280)

[pardon.Pardon.convert\_to\_numeric 39](#_Toc117584281)

[pardon.Pardon.convert\_to\_string 40](#_Toc117584282)

[pardon.Pardon.convert\_to\_date 41](#_Toc117584283)

[pardon.Pardon.create\_class\_labels 42](#_Toc117584284)

[pardon.Pardon.find\_outliers 43](#_Toc117584285)

[pardon.Pardon.remove\_outliers 44](#_Toc117584286)

[pardon.Pardon.remove\_stop\_words 45](#_Toc117584287)

[pardon.Pardon.drop\_nulls 46](#_Toc117584288)

[pardon.Pardon.fill\_nulls 47](#_Toc117584289)

[pardon.Pardon.one\_hot\_encode 48](#_Toc117584290)

[pardon.Pardon.frequency\_encode 49](#_Toc117584291)

[pardon.Pardon.label\_encode 50](#_Toc117584292)

[pardon.Pardon.ordinal\_encode 51](#_Toc117584293)

[pardon.Pardon.columns\_with\_nulls 52](#_Toc117584294)

[pardon.Pardon.numeric\_columns 53](#_Toc117584295)

[pardon.Pardon.string\_columns 54](#_Toc117584296)

[pardon.Pardon.remove\_rows\_containing 55](#_Toc117584297)

[pardon.Pardon.map\_column\_values 56](#_Toc117584298)

[pardon.Pardon.replace\_values 57](#_Toc117584299)

[pardon.Pardon.remove\_columns 58](#_Toc117584300)

[pardon.Pardon.remove\_correlated\_columns 59](#_Toc117584301)

[pardon.Pardon.correlated\_columns 60](#_Toc117584302)

[pardon.Pardon.rename\_columns 61](#_Toc117584303)

[pardon.Pardon.clean\_column\_names 62](#_Toc117584304)

[pardon.Pardon.sort\_by 63](#_Toc117584305)

[pardon.Pardon.correlations 64](#_Toc117584306)

[pardon.Pardon.histogram 65](#_Toc117584307)

[pardon.Pardon.create\_bins 66](#_Toc117584308)

[pardon.Pardon.make\_upper\_case 67](#_Toc117584309)

[pardon.Pardon.drop\_duplicates 68](#_Toc117584310)

[pardon.Pardon.pca 69](#_Toc117584311)

[pardon.Pardon.apply\_func 70](#_Toc117584312)

[pardon.Pardon.add\_func 71](#_Toc117584313)

[pardon.Pardon.ignore\_function\_in\_predictions 73](#_Toc117584314)

[pardon.Pardon.apply\_sentiment 74](#_Toc117584315)

[pardon.Pardon.combine\_columns 75](#_Toc117584316)

[pardon.Pardon.split\_column 76](#_Toc117584317)

[pardon.Pardon.get\_sample\_rows 77](#_Toc117584318)

[pardon.Pardon.scale\_data 78](#_Toc117584319)

[pardon.Pardon.rapid\_ml 79](#_Toc117584320)

[pardon.Pardon.find\_best\_model\_parameters 84](#_Toc117584321)

[pardon.Pardon.train\_model 85](#_Toc117584322)

[pardon.Pardon.explain\_model\_predictions 87](#_Toc117584323)

[pardon.Pardon.model\_summary 88](#_Toc117584324)

[pardon.Pardon.save\_model 89](#_Toc117584325)

[pardon.Pardon.required\_columns 90](#_Toc117584326)

[pardon.Pardon.predict 91](#_Toc117584327)

[pardon.Pardon.model\_script 92](#_Toc117584328)

[pardon.Pardon.model\_parameters 93](#_Toc117584329)

[pardon.Pardon.get\_default\_model\_params 94](#_Toc117584330)

[pardon.Pardon.set\_default\_model\_params 95](#_Toc117584331)

[pardon.Pardon.model\_diagram 96](#_Toc117584332)

[pardon.Pardon.model\_learning\_curve 96](#_Toc117584333)

[pardon.Pardon.get\_feature\_importance 98](#_Toc117584334)

[pardon.Pardon.add\_data 99](#_Toc117584335)

[pardon.Pardon.get\_feature\_contribution 100](#_Toc117584336)

[pardon.Pardon.search\_spaces 101](#_Toc117584337)

[pardon.Pardon.trasnformations 102](#_Toc117584338)

[pardon.Pardon.plot\_data 103](#_Toc117584339)

[pardon.Pardon.output\_data 111](#_Toc117584340)

[pardon.Pardon.create\_clusters 112](#_Toc117584341)

[pardon.Pardon.transform 113](#_Toc117584342)

[pardon.Pardon.add\_fail\_on 114](#_Toc117584343)

[pardon.Pardon.delete\_fail\_on 115](#_Toc117584344)

[pardon.Pardon.apply\_fail\_ons 116](#_Toc117584345)

[pardon.Prediction 117](#_Toc117584346)

[pardon.Prediction Attributes 118](#_Toc117584347)

[target 118](#_Toc117584348)

[model\_identifier 118](#_Toc117584349)

[features 118](#_Toc117584350)

[model\_classes 118](#_Toc117584351)

[class\_labels 118](#_Toc117584352)

[predicted\_datetime 118](#_Toc117584353)

[y 118](#_Toc117584354)

[actual 118](#_Toc117584355)

[is\_correct 118](#_Toc117584356)

[delta 118](#_Toc117584357)

[predicted 118](#_Toc117584358)

[probabilities 118](#_Toc117584359)

[audit 118](#_Toc117584360)

[error 118](#_Toc117584361)

[pardon.Prediction Methods 119](#_Toc117584362)

[pardon.Prediction.api\_output 119](#_Toc117584363)

[pardon.Prediction.as\_dataframe 122](#_Toc117584364)

[pardon.Audit 123](#_Toc117584365)

[pardon.Audit Attributes 124](#_Toc117584366)

[prediction 124](#_Toc117584367)

[fullpath 124](#_Toc117584368)

[PREDICTIONS\_TABLE\_NAME 124](#_Toc117584369)

[pardon.Audit Methods 125](#_Toc117584370)

[pardon.Audit.select\_all\_predictions 125](#_Toc117584371)

[pardon.FailOn 126](#_Toc117584372)

[pardon.FailOn Attributes 128](#_Toc117584373)

[column 128](#_Toc117584374)

[operator 128](#_Toc117584375)

[value 128](#_Toc117584376)

[calculation 128](#_Toc117584377)

[subset 128](#_Toc117584378)

[pardon.FailOn Methods 129](#_Toc117584379)

[pardon.FailOn.apply\_fail\_on 129](#_Toc117584380)

[pardon.FailOnOutput Attributes 130](#_Toc117584381)

[column 130](#_Toc117584382)

[failed 130](#_Toc117584383)

[failed\_value 130](#_Toc117584384)

[pardon.InvalidModel 131](#_Toc117584385)

[pardon.InvalidModel Attributes 132](#_Toc117584386)

[model 132](#_Toc117584387)

[model\_type 132](#_Toc117584388)

[error 132](#_Toc117584389)

[pardon.TestAPI 133](#_Toc117584390)

[Functions 134](#_Toc117584391)

[pardon.load\_model 134](#_Toc117584392)

[pardon.data\_reader 135](#_Toc117584393)

[pardon.remove\_non\_ascii 136](#_Toc117584394)

[pardon.extract\_postcode 137](#_Toc117584395)

[pardon.remove\_characters 138](#_Toc117584396)

[pardon.get\_sentiment 139](#_Toc117584397)

[pardon.flatten\_list 140](#_Toc117584398)

[pardon.create\_intervals 141](#_Toc117584399)

[pardon.to\_json 142](#_Toc117584400)

[pardon.get\_data\_averages 143](#_Toc117584401)

[pardon.save\_pickle 144](#_Toc117584402)

[pardon.load\_pickle 145](#_Toc117584403)

[pardon.assert\_item\_type 146](#_Toc117584404)

[Options 147](#_Toc117584405)

[pardon.pardon\_options 147](#_Toc117584406)

[MIN\_ROW\_REQ 147](#_Toc117584407)

[VALID\_FILE\_FORMATS 147](#_Toc117584408)

[DATETIME\_FORMAT 147](#_Toc117584409)

[AVERAGE\_FAIL\_DEFAULT\_VALUE 147](#_Toc117584410)

[MIN\_SAMPLE\_FOR\_UNDERSAMPLE 147](#_Toc117584411)

[UNDER\_SAMPLING\_MODEL 147](#_Toc117584412)

[OVER\_SAMPLING\_MODEL 147](#_Toc117584413)

[XGBCLASSIFIER\_DEFAULT 147](#_Toc117584414)

[XGBCLASSIFIER\_BINARY\_DEFAULT 147](#_Toc117584415)

[SKLEARN\_CLASSIFIER\_DEFAULT 148](#_Toc117584416)

[SKLEARN\_REGRESSION\_DEFAULT 148](#_Toc117584417)

[REDUCTION\_SCORE\_ON 148](#_Toc117584418)

[BALANCE\_SCALE\_RATIO 148](#_Toc117584419)

[IGNORE\_SINGLE\_CLASS\_ERROR 148](#_Toc117584420)

[pardon.app 150](#_Toc117584421)

[Release History 151](#_Toc117584422)

# Art, design, multimedia, storyboard, writing, script icon - Download on IconfinderFolder, model, save, simulation, simulation and digital twins, technology disruption icon - Download on IconfinderArtificial intelligence, digital knowledge, information technology, knowledge management, machine learning icon - Download on IconfinderData, cleansing, monitoring, detection, process icon - Download on IconfinderModel, print, printing, prototype, 3d printing, technology disruption icon - Download on IconfinderCleaning, data, data cleaning, data cleansing, data validation, database, fileCleaning, data, data cleaning, data cleansing, data validation, database, fileCleaning, data, data cleaning, data cleansing, data validation, database, fileCode, document, extension, extention, file, json, format icon - Download on IconfinderDocument, extension, file, format, page, spreadsheet, xls icon - Download on IconfinderCsv, document, extension, file, format, page icon - Download on IconfinderData, database, storage3d, cube, geometry, image, model, rounded, square icon - Download on IconfinderArt, design, multimedia, storyboard, writing, script icon - Download on Iconfinderpardon Process Overview

Raw data for prediction

Prediction response

Data transformations from the ‘Script’ applied by the model

*Prediction pathway*

Model built

Choose a custom model

*Rapid pathway*

Train and test the model

Selects best performing model



Tries a selection of models

Use automatic data transformations

Perform custom data transformations

*Custom pathway*

Data transformations added to ‘Script’

Create model object with data

# Installation

## pip installation

The easiest way to install and use the pardon library is to [pip](https://pypi.org/project/pip/) install the [whl](https://pypi.org/project/wheel/0.22.0/) file. This will install the library and all the required dependencies. If you first need a virtual environment, you can follow the steps to create one using the [create a virtual environment](#_Creating_a_virtual) information.

To perform the install, do the following:

1. If applicable, activate your virtual environment.

Replace venv\_name with your virtual environment name before using the below commands:

With CMD, navigate to the folder containing your virtual environment and run the following:

.\venv\_name\Scripts\activate

If using anaconda, activate it with the following:

conda activate venv\_name

1. Navigate to the folder containing the pardon-x.x.x.whl file
2. Run the command:

pip install pardon-x.x.x.whl --user

1. To uninstall you can run the following:

pip uninstall pardon

## Adding pardon with a conda dependencies file

You can add the pardon library to a virtual environment on a cloud environment such as [Microsoft Azure](https://azure.microsoft.com/en-gb), by including a reference to your [whl](https://pypi.org/project/wheel/0.22.0/) file in a [conda dependencies](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html) file.

Below shows an example [conda dependencies](https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-pkgs.html) file, including reference to the pardon [whl](https://pypi.org/project/wheel/0.22.0/) file, uploaded to [Azure BLOB storage](https://azure.microsoft.com/en-gb/services/storage/blobs/):

dependencies:

- python=3.9.7

- pip=21.2.3

- pip:

- >-

https://xxxxxx.blob.core.windows.net/azureml-blobstore-xxx-xxx-xxx-xxx-xxxxxxxxxxxx/xxx-xx/pardon-1.0.0-py3-none-any.whl

name: azureml\_xxxxxxxxxxxxxxx

## Creating the pardon .whl file

If you have access to the pardon repo, the .whl file should be present in the pardon upper directory, in a folder called dist. If this doesn’t exist, you can create the folders and .whl file using the setup.py file.

Perform the following steps:

1. Install the wheel library using:

pip install wheel

1. Ensuring you can see the pardon setup.py in the current directory, run the following command:

python setup.py bdist\_wheel

This will create 3 folders: build, dist, and an egg-info folder. The .whl file can be found in the dist folder.

## Creating a virtual environment

If you want to create a virtual environment before installing pardon, this can be done by performing the following steps.

With pip

1. Pip install the virtual environment package using:

pip install virtualenv

1. Create your virtual environment and name it using the following command:

virtualenv virt\_env\_name\_here

1. You can activate your virtual environment using the steps found at the [pip installation](#_pip_installation) section of this document.

With conda

1. Create your virtual environment and name it using the following command:

conda create -n virt\_env\_name\_here

## Installing dependencies using requirements.txt

If you want to install all the necessary dependencies using the pardon requirements.txt file, you can do so using the following:

1. Navigate to the folder containing the requirements.txt file.
2. Run the following command:

pip install -r requirements.txt

# Available Models

## Regression Models

##### [LinearRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html) (default)

##### [Ridge](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Ridge.html#sklearn.linear_model.Ridge)

##### [SGDRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDRegressor.html#sklearn.linear_model.SGDRegressor)

##### [ElasticNet](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNet.html#sklearn.linear_model.ElasticNet)

##### [Lasso](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html#sklearn.linear_model.Lasso)

##### [HuberRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.HuberRegressor.html#sklearn.linear_model.HuberRegressor)

##### [PoissonRegressor](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PoissonRegressor.html#sklearn.linear_model.PoissonRegressor)

## Classification Models

##### [XGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_intro.html) (default)

##### [RandomForestClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

##### [BernoulliNB](https://scikit-learn.org/stable/modules/generated/sklearn.naive_bayes.BernoulliNB.html)

##### [AdaBoostClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html)

##### [LogisticRegression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)

##### [BaggingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html)

##### [HistGradientBoostingClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html)

##### [SGDClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html)

##### [KNeighborsClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html)

##### [PassiveAggressiveClassifier](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.PassiveAggressiveClassifier.html#sklearn.linear_model.PassiveAggressiveClassifier)

## Clustering Models

##### [KMeans (default)](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html?highlight=kmeans#sklearn.cluster.KMeans)

##### [Birch](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.Birch.html#sklearn.cluster.Birch)

##### [MeanShift](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.MeanShift.html" \l "sklearn.cluster.MeanShift)

##### [DBSCAN](https://scikit-learn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html" \l "sklearn.cluster.DBSCAN.set_params)

More models to be added in future versions.

Custom models can be used by passing them to the model argument of the [pardon.Pardon.train\_model](#_pardon.MLData.train_model) method and any parameters can be added using the model\_params argument.

# Classes

## pardon.Pardon

***class pardon.Pardon(data, target=None, columns=[], encoding='latin-1', sep=’,’, error\_bad\_lines=None, test\_size=0.3, remove\_single\_instance\_classes=False, is\_regression=’infer’, stratify=True, shuffle=True, yaml=None)***

Class allowing data transformations, model training, visualisations, and predictions as well as prediction auditing.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data source. Can be a file path or an object.

target :*str, None, default None*

The column name that we are attempting to predict. Note, the column name is case sensitive. If None, no Model methods will be available, but clusters can be created and plotted, data transformations performed, and data visualisations produced. See the [create\_clusters](#_pardon.MLData.create_clusters) and [plot\_data](#_pardon.MLData.plot_data) methods.

columns :*list, default [] (empty list)*

A list of columns that you want to include in your training. If not used, all columns found in the data will be used.

encoding :*str, default ‘latin-1’ {‘latin-1’ , ‘ascii’, ‘utf-8’}*

The encoding used when opening csv files.

sep :*str, default ‘,’*

The delimiter to use when opening csv or txt files.

error\_bad\_lines :*bool, default None*

Raise an error if bad lines with too many or too few delimiters are found. If False, bad lines will be dropped.

test\_size :*float, default 0.3*

The proportion of the data to retain for model testing. 0.3 refers to 30%. Note, if the target is None, the test\_size will be set to 0 and all data will be retained in the [train\_data](#_train_data) attribute.

remove\_single\_instance\_classes :*bool, default False*

Remove the rows of classes that only contain a single instance of that class. Machine learning models require each class to have at least 2 instances for training. This is set to False as the default to ensure users know these rows are being removed. If they are not removed and a single instance is found in the data, the [train\_model](#_pardon.Pardon.train_model) method will fail.

is\_regression :*bool, str, default ‘infer’ {True, False, ‘infer’}*

Determines if the model is a regression task or a classification task. If True the models used will be regression models, if False, classification models. ‘infer’ means this will attempt to be determined automatically by checking if the data is numeric and continuous in the target column.

stratify :*bool, default True*

Use the same proportion of classes encountered in the dataset when splitting data into train and test. For example, if your data contains 5% instances of class A and 95% of class B, these proportions will be maintained in the train and test datasets.

shuffle :*bool, default True*

*[added v1.1.7]*

Shuffle the dataset into a random order during the train/test split. If using time series data, this should be set to False.

yaml :*str, default None*

Reference to a yaml file containing configuration details. This will be saved as a dictionary object in the [config](#_config) attribute.

*Example*

myml = pardon.Pardon(data=’call\_data.csv', target='TreatmentType')

### pardon.Pardon Attributes

#### test\_size

The proportion of data being held for model testing.

#### target

The name of the column containing the classes being predicted.

#### scaler\_used

The name of the type of scaler being used.f

#### model\_trained\_date

The datetime the model finished training.

#### model

The class object of the machine learning model used. These can be seen in the [Available Models](#_Available_Models) section.

#### model\_type

The name of the model being used. Will present as the class name of the model classes listed in the model attribute.

#### best\_hyperparameters

A dictionary object containing the model being tested and the best parameter values found during [hyperparameter tuning](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) with cross validation.

#### validation\_training\_time\_mins

A dictionary object showing the time in minutes the [hyperparameter tuning](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) with cross validation took to complete for each model.

#### model\_training\_time\_mins

The time in minutes the model training took to complete.

#### class\_labels

A dictionary containing the class labels and their corresponding numeric values as used in the model.

#### failed\_models

A list containing the [pardon.InvalidModel](#_pardon.InvalidModel) classes containing the models that failed during the [Pardon.rapid\_ml](#_pardon.MLData.rapid_ml) method model selection.

#### model\_unique\_identifier

A unique identifier for the trained model.

#### rapid\_ml\_scores

A dictionary containing all the models that were trained during the [Pardon.rapid\_ml](#_pardon.MLData.rapid_ml) method and their corresponding performance scores.

#### rapid\_ml\_score\_metric

The metric used to determine the best performing model when using the [Pardon.rapid\_ml](#_pardon.MLData.rapid_ml) method.

#### best\_validation\_score

A dictionary object containing the model being tested and the best score found during [hyperparameter tuning](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) with cross validation.

#### model\_train\_scores

The model performance scores during training.

#### model\_test\_scores

The model performance scores during testing.

#### model\_eval\_metric

[added v1.0.5]  
The model evaluation metric used during training.

#### data

A [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing the data used in the model. This includes training data, test data, and validation data in its current form after any transformations.

#### raw\_data

A [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing the raw data used when instantiating the [pardon.](#_pardon.MLData)Pardon object.

#### train\_data

A [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing the data used in training the model in its current form after any transformations.

#### test\_data

A [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing the data used in testing the model after training in its current form after any transformations.   
Note, if the target was set to None during model instantiation, the test\_data attribute will return an empty [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) and all data held in the train\_data attribute.

#### columns

A list of the data columns used by the model.

#### raw\_data\_columns

A list of the data columns in the raw data used when instantiating the [pardon.](#_pardon.MLData)Pardon object.

#### class\_distribution

A dictionary containing the distribution of each class in the target column in the raw data. Shows both the count and percentage of occurrences of each class.

#### train\_class\_distribution

A dictionary containing the distribution of each class in the target column in the train data. Shows the count of occurrences of each class. Only available after model training.

#### test\_class\_distribution

[added v1.1.7]

A dictionary containing the distribution of each class in the target column in the test data. Shows the count of occurrences of each class. Only available after model training.

#### model\_fullpath

The fullpath to where the model was saved. Only available after the [save\_model](#_pardon.MLData.save_model) method has been completed.

#### input\_data\_fullpath

The fullpath for the input dataset. Only applicable if the [input dataset](#_pardon.MLData) was a file.

#### prediction\_audit\_fullpaths

The fullpath(s) to where any [prediction audits](#_pardon.Audit) have been saved.

#### fail\_ons

A list containing details of all the [FailOns](#_pardon.InvalidModel) added to your model.

#### config

A dictionary containing the details from the yaml file if provided.

### pardon.Pardon Methods

#### pardon.Pardon.available\_models

***pardon.Pardon.available\_models()***

Return a tuple of the available machine learning models. The available models can be seen in the [pardon.Pardon.model](#_model) attribute documentation.

*Example*

available\_models = myml.available\_models()

#### pardon.Pardon.available\_evaluation\_metrics

***pardon.Pardon.available\_evaluation\_metrics()***

Returns a list of the available evaluation metrics used during cross validation in model training. Note, this does not include the evaluation metrics for [XGBClassifier](https://xgboost.readthedocs.io/en/stable/python/python_api.html) models.

*Example*

available\_metrics = myml.available\_evaluation\_metrics()

#### pardon.Pardon.available\_chart\_types

***pardon.Pardon.available\_chart\_types()***

Returns a tuple of the chart types for data visualisation using the [plot\_data](#_pardon.MLData.plot_data) method.

*Example*

available\_charts = myml.available\_chart\_types()

#### pardon.Pardon.available\_scoring\_metrics

***pardon.Pardon.available\_scoring\_metrics()***

Returns a list of the available scoring metrics used to determine the best performing models when using the [rapid\_ml](#_pardon.MLData.rapid_ml) method.

*Example*

available\_scoring\_metrics = myml.available\_scoring\_metrics()

#### pardon.Pardon.available\_encoding\_columns

***pardon.Pardon.available\_encoding\_columns()***

Return a list of columns that are valid for encoding and have not yet been frequency, label, or one hot encoded. This will only show columns with a data type of object.

*Example*

columns\_to\_encode = myml.available\_encoding\_columns()

myml.label\_encode(columns\_to\_encode)

#### pardon.Pardon.add\_to\_encoded

***pardon.Pardon.add\_to\_encoded(columns)***

Add any columns that have been manually created that you want to be ignored by the label encoder. By default, any numeric columns will be ignored by the label encoder.

**Parameters:**

columns :*list*

The names of the columns to be added to the columns to be ignored by the label encoder.

*Example*

columns\_to\_encode = myml.available\_encoding\_columns()

myml.add\_to\_encoded(columns=[‘patient\_call\_type’])

#### pardon.Pardon.get\_best\_features

***pardon.Pardon.get\_best\_features(max\_features=20, min\_contribution\_score=None)*** *[added v1.1.1]*

Returns a list of that are determined to be the [best performing features](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html).

**Parameters:**

max\_features :*int, default 20*

The maximum number of columns to retain. This will mean only the top n best performing columns will be kept.

min\_contribution\_score :*int, default None*

The minimum score returned based on the [ANOVA F-value](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif) for classification tasks and the [F-Statistic](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression) for regression, for the provided sample. If the column does not meet the minimum score, it will not be retained.

*Example*

best\_features = myml.get\_best\_features(max\_features=10, min\_contribution\_score=75)

#### pardon.Pardon.set\_best\_features

***pardon.Pardon.set\_best\_features(max\_features=20, min\_contribution\_score=None)***

Automatically sets the data columns to be used in model training based on [best performing features](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html). This automatically sets the columns using the [select\_best\_features](#_pardon.Pardon.select_best_features) method.

**Parameters:**

max\_features :*int, default 20*

The maximum number of columns to retain. This will mean only the top n best performing columns will be kept.

min\_contribution\_score :*int, default None*

The minimum score returned based on the [ANOVA F-value](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif) for classification tasks and the [F-Statistic](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression) for regression, for the provided sample. If the column does not meet the minimum score, it will not be retained.

*Example*

myml.set\_best\_features(max\_features=10, min\_contribution\_score=75)

#### pardon.Pardon.map\_data

***pardon.Pardon.map\_data(mapping\_dict)***

Map data to a mapping value or file.

**Parameters:**

mapping\_data :*dict*

A dictionary object containing a key of the column name to map, and the value being the data that provides the mapping. The value can be any item accepted by the data parameter as per the [Pardon instantiation](#_pardon.MLData).

Think of this function as equivalent to Excel’s vlookup, where you specify the column name to apply this to, and then the vlookup. An example would be mapping\_data={'PWSymptomGroupDesc': 'symptom\_groups\_prioritised.csv'}. The symptom\_groups\_prioritised.csv will contain each value as found in the PWSymptomGroupDesc column and the corresponding value(s) that should map to. The column(s) containing the corresponding value(s) will be added to the data.

*Example*

myml.map\_data(mapping\_data={'PWSymptomGroupDesc': 'symptom\_groups\_prioritised.csv'})

#### pardon.Pardon.row\_count

***pardon.Pardon.row\_count(dataset=’all’)***

Get a count of rows in the selected dataset.

**Parameters:**

dataset :*str, default ‘all’ {‘all’, ‘train’, ‘test’, ‘raw’}*

The dataset to get the row count for. Valid inputs are all, train, test, raw*. ‘all’ is a combination of train and test data.*

*Example*

raw\_row\_count = myml.row\_count(dataset=’raw’)

#### pardon.Pardon.column\_count

***pardon.Pardon.column\_count()***

Get a count of columns used in the data, equivalent to running *len([Pardon.columns](#_columns)).*

*Example*

column\_count = myml.column\_count()

#### pardon.Pardon.null\_count

***pardon.Pardon.null\_count()***

Returns the count of nulls in each column where at least 1 null is present.

*Example*

nulls = myml.null\_count()

#### pardon.Pardon.transformation\_count

***pardon.Pardon.transformation\_count()***

*Added v1.2.9*

Return a count of the number of transformations applied to the dataset.

*Example*

transform\_count = myml.transformation\_count()

#### pardon.Pardon.number\_of\_classes

***pardon.Pardon.number\_of\_classes()***

Get a count of the number of classes in the [target column](#_target) of the data.

*Example*

class\_count = myml.number\_of\_classes()

#### pardon.Pardon.classes

***pardon.Pardon.classes()***

Returns a list of unique classes in the [target column](#_target) of the data.

*Example*

classes = myml.classes()

#### pardon.Pardon.data\_info

***pardon.Pardon.data\_info()***

Returns a [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing information about the [data](#_data).

*Example*

print(myml.data\_info())

#### pardon.Pardon.data\_types

***pardon.Pardon.data\_types()***

*[Added v1.0.8]*

Returns a [pandas.DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) containing the data types.

*Example*

print(myml.data\_types())

#### pardon.Pardon.remove\_unhelpful\_columns

***pardon.Pardon.remove\_unhelpful\_columns(max\_null\_ratio=0.5)***

Removes columns from the data that have been deemed as unhelpful. An unhelpful column is one categorised as one of the following:

* Every data item in the column is the same value.
* The column data type is a string, and every data item in the column is different.
* The proportion of rows that are null exceeds that specified in the max\_null\_ratio parameter.

**Parameters:**

max\_null\_ratio :*float, default 0.5*

The max proportion of the column that contains null values. 0.5 would mean that if more than 50% of the column values are null, the column will be deemed as unhelpful.

*Example*

myml.remove\_unhelpful\_columns(max\_null\_ratio=0.3)

#### pardon.Pardon.unhelpful\_columns

***pardon.Pardon.unhelpful\_columns(max\_null\_ratio=0.5)***

Returns a list of columns from the data that have been deemed as unhelpful. An unhelpful column is one categorised as one of the following:

* Every data item in the column is the same value.
* The column data type is a string, and every data item in the column is different.
* The proportion of rows that are null exceeds that specified in the max\_null\_ratio parameter.

**Parameters:**

max\_null\_ratio :*float, default 0.5*

The maximum proportion of the column that contains null values. 0.5 would mean that if more than 50% of the column values are null, the column will be deemed as unhelpful.

*Example*

unhelpful = myml.unhelpful\_columns(max\_null\_ratio=0.3)

#### pardon.Pardon.convert\_to\_numeric

***pardon.Pardon.convert\_to\_numeric(columns, fill\_non\_numeric=’median’)***

Converts the columns specified in the [columns parameter](#_columns) to a [numeric datatype](https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html). Specify how to deal with non-numeric or empty rows. If no numeric data is found when determining an average, every row will be set to [Pardon.AVERAGE\_FAIL\_DEFAULT\_VALUE](#_AVERAGE_FAIL_DEFAULT_VALUE) by default.

**Parameters:**

columns :*list*

A list containing the columns to be converted to a [numeric datatype](https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html).

fill\_non\_numeric :*str, int, float, func, default ‘median’ {‘mean’, ‘median’, ‘mode’}*

Fill non-numeric values with the column average, pass a number or a function..

*Example*

myml.convert\_to\_numeric(columns=[‘Age’, ‘Price’], fill\_non\_numeric=’mode’)

#### pardon.Pardon.convert\_to\_string

***pardon.Pardon.convert\_to\_string(columns)***

*[Added v1.0.8]*

Converts the columns specified in the [columns parameter](#_columns) to a string/object datatype.

**Parameters:**

columns :str,*list*

A single column string or a list containing the columns to be converted to a string/object datatype.

*Example*

myml.convert\_to\_string(columns=[‘name, ‘address’])

#### pardon.Pardon.convert\_to\_date

***pardon.Pardon.convert\_to\_datetime(columns, fill\_non\_datetime=None, day\_first=True, year\_first=False, format=None)***

Converts the columns specified in the [columns parameter](#_columns) to a [date datatype](https://pandas.pydata.org/docs/reference/api/pandas.to_datetime.html). Specify how to deal with non-date or empty rows using the fill\_empty parameter.

**Parameters:**

columns :*list*

A list containing the columns to be converted to a [numeric datatype](https://pandas.pydata.org/docs/reference/api/pandas.to_numeric.html).

fill\_non\_datetime:*None, Str, default None {None, ‘mode’}*

Use ‘mode’ to fill empty values with the most common date or pass a date in a string format. Note, if fill\_non\_datetime=None, non-date rows will be dropped.

day\_first :*bool, default True*

If True, parse dates as though the day is first. For example, "10/11/12" is parsed as 2012-11-10.

year\_first :*bool, default False*

If True, parse dates as though the year is first. For example, "10/11/12" is parsed as 2010-11-12. If both day\_first=True and year\_first=True, year\_first will take precedence.

format :*None, str, default None*

The date format to parse the dates as. Follow [the guidance for the correct date format](https://docs.python.org/3/library/datetime.html#strftime-and-strptime-behavior). If None, the format will be inferred automatically.

*Example*

myml.convert\_to\_datetime(columns=[‘ActionDate’, ‘ShiftDate’], fill\_non\_datetime=’2022/12/20’, year\_first=True)

#### pardon.Pardon.create\_class\_labels

***pardon.Pardon.create\_class\_labels(class\_labels)***

Creates string class labels for numeric classes in data. This means you can rename your classes something more relevant while retaining the specific numeric values during model training. Note, this is only applicable for classification models.

**Parameters:**

class\_labels :*dict*

A dictionary containing the current, numeric class names as the dictionary keys, and the new string class label as the value. You must ensure every numeric class item is included and all new string class labels are unique.

*Example*

new\_class\_labels = {0: 'Hear & Treat',

1: 'See & Treat',

2: 'See & Convey'}

myml.create\_class\_labels(class\_labels=new\_class\_labels)

#### pardon.Pardon.find\_outliers

***pardon.Pardon.find\_outliers(columns=[], z\_threshold=3.0, apply\_to\_test=True, return\_as=’values’, ignore\_target=False)***

Returns a dictionary object containing the column name as a key, and then the outlier values as the value.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to check for outliers. An empty list means all numeric columns will be checked for outliers.

z\_threshold :*int, float, default 3.0*

The [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) determining an outlier. A [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) of 3.0 means the value is 3 standard deviations or more away from the column mean.

apply\_to\_test:*bool, default True [added v1.0.8]*

Find outliers for the train and test data.

return\_as :*str, default ‘values’ {‘values’, ‘index’}*

The format to return the outliers. Values means the actual values are returned. This is the recommended format. If not set to values, a [pandas.Series](https://pandas.pydata.org/docs/reference/api/pandas.Series.html) object containing the associated indexes will be returned.

ignore\_target :*bool, default False*

Specify if you want to ignore the target column when finding outliers in the data. This only applies if no columns are supplied to the columns argument. ignore\_target=False means that unless columns are specified, the target column will also be searched and have any outliers removed.

*Example*

outliers = myml.find\_outliers(columns=[‘age’, ‘price’], z\_threshold=2.5)

#### pardon.Pardon.remove\_outliers

***pardon.Pardon.remove\_outliers(columns=[], z\_threshold=3.0, apply\_to\_test=True, ignore\_target=False)***

Removes the outliers from the model’s training data only. Test data remains unchanged.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to remove outliers. An empty list means all numeric columns will be checked for outliers.

z\_threshold :*int, float, default 3.0*

The z score determining an outlier. A [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) of 3.0 means the value is 3 standard deviations or more away from the column mean.

apply\_to\_test:*bool, default True [added v1.0.8]*

Remove outliers from the train and test data.

ignore\_target :*bool, default False*

Specify if you want to ignore the target column when finding outliers in the data. This only applies if no columns are supplied to the columns argument. ignore\_target=False means that unless columns are specified, the target column will also be searched and have any outliers removed.

*Example*

myml.remove\_outliers(columns=[‘age’, ‘price’], z\_threshold=3.0)

#### pardon.Pardon.remove\_stop\_words

***pardon.Pardon.remove\_stop\_words(columns=[])***

Removes “[stop words](https://en.wikipedia.org/wiki/Stop_word)” from text. These are words that are typically filtered prior to processing of natural language, such as: “a”, “the”, “is”, “are”.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to remove stop words from. An empty list means all string columns will be checked for stop words.

*Example*

myml.remove\_stop\_words(columns=’description\_text’)

#### pardon.Pardon.drop\_nulls

***pardon.Pardon.drop\_nulls(columns=[])***

Drops rows containing nulls in the columns specified.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to check for null values. An empty list means all columns will be checked for nulls.

*Example*

myml.drop\_nulls(columns=[‘age’])

#### pardon.Pardon.fill\_nulls

***pardon.Pardon.fill\_nulls(columns=[], fill\_text\_with=’Unknown’, fill\_numeric\_with=’mean’, by\_col\_name={})***

Fill null values in the columns specified with the values specified.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to check for null values. An empty list means all columns will be checked.

fill\_text\_with :*str, default ‘Unknown’*

The value to replace null values with, in text columns.

fill\_numeric\_with :*str, float, int, default ‘mean’*

The value to replace null values with, in numeric columns. This can be a numeric value or mean, median, mode to fill nulls with an average from the column.

by\_col\_name :*dict, default {} (empty dict)*

A dictionary containing the column name as a key and the value to replace nulls with. If this argument is used, all other arguments will be ignored.

*Example*

myml.full\_nulls(columns=[‘person\_age’], fill\_numeric\_with=50)

myml.fill\_nulls(by\_col\_name={‘other\_age’: ‘median’, ‘name’: ‘Unknown’, ‘price’: ‘mean’})

#### pardon.Pardon.one\_hot\_encode

***pardon.Pardon.one\_hot\_encode(columns=[], max\_items\_in\_category=10, remove\_encoded=True)***

[One hot encode](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) the columns specified if they have the max\_items\_in\_catgeory or fewer unique values.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to [one hot encode](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html). An empty list means all columns will be checked.

max\_items\_in\_category :*int, default 10*

The maximum number of unique values to be present in the column before it will be [one hot encoded](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html). For example, if the column contained the values, ‘High’, ‘Medium’, and ‘Low’, that would be 3 unique values. Only applied when no columns are provided.

remove\_encoded :*bool, default True [added in v1.0.3]*

Remove the columns that were encoded. Use False to retain the columns that were one hot encoded (not recommended).

*Example*

myml.one\_hot\_encode(max\_items\_in\_category=15)

#### pardon.Pardon.frequency\_encode

***pardon.Pardon.frequency\_encode(columns=[])***

[Frequency encode](https://contrib.scikit-learn.org/category_encoders/count.html) the columns specified.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to frequency encode. An empty list means all columns will be checked.

*Example*

myml.frequency\_encode(columns=[‘postcode’, ‘latlon’])

#### pardon.Pardon.label\_encode

***pardon.Pardon.label\_encode(columns=[])***

[Label encode](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) the columns specified.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to [label encode](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html). An empty list means all columns will be checked.

*Example*

myml.label\_encode()

#### pardon.Pardon.ordinal\_encode

***pardon.Pardon.ordinal\_encode(columns=[], order\_by=’value’, include\_target=False)***

*[added v1.1.7]*

Ordinal encode the columns specified.

**Parameters:**

columns :*list, default [] (empty list)*

The columns to ordinal encode. An empty list means all columns will be checked.

order\_by :*str, dict, list, function, default 'value', {'value', 'frequency', dict, list, func}*

How to order the values in a column prior to encoding. Specify a dictionary containing the column name as key and how to order that column. Functions will be applied to the entire column as a list to specify the order. Frequency means the most frequently occurring item will have the highest encoded value and so on. Items not seen during training will be given the value 0.

include\_target :*bool, default False*

Include the target when not specifying columns to ordinal encode. Typically, the target should be encoded with the label encoder.

*Example*

myml.ordinal\_encode(columns=[‘name’, ‘postcode’], order\_by=’frequency’)

#### pardon.Pardon.columns\_with\_nulls

***pardon.Pardon.columns\_with\_nulls()***

Return a list of columns that contain null values.

*Example*

null\_vals\_cols = myml.columns\_with\_nulls()

#### pardon.Pardon.numeric\_columns

***pardon.Pardon.numeric\_columns()***

Return a list of columns that have a numeric data type.

*Example*

numeric\_cols = myml.numeric\_columns()

#### pardon.Pardon.string\_columns

***pardon.Pardon.string\_columns(data=None)***

Return a list of columns that have a object data type.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)*, default None*

The data to check for columns with for text/object data types. If no data is specified, the training data will be used.

*Example*

val\_text\_cols = myml.text\_columns(data=myml.validation\_data)

#### pardon.Pardon.remove\_rows\_containing

***pardon.Pardon.remove\_rows\_containing(column\_items)***

Remove rows that contain a specific value in the specified columns.

**Parameters:**

column\_items :*dict*

A dictionary object with the column name as a key and the value to look for as the value. Rows in the column specified containing that value will be dropped. Multiple values can be specified in the form of a list. To remove rows with non-numeric values, pass ‘non-numeric’. You can also pass a function which if returns a value or True, that row will be removed.

*Example*

myml.remove\_rows\_containing(column\_items={‘age’: [-1, 0, 120, ‘non-numeric’], ‘price’: 0})

#### pardon.Pardon.map\_column\_values

***pardon.Pardon.map\_column\_values(mapping\_items)***

Map values in a specified column to a new value. **All values** must be present in the mapping items value else null will be returned. Think of this as equivalent to Excel’s vlookup.

**Parameters:**

mapping\_items :*dict*

A dictionary object with the column name as a key and the value as another dictionary object containing how you want values to be mapped.

*Example*

map\_name\_values = {‘matt’: ‘Matthew’, ‘rich’: ‘Richard’, ‘greg’, ‘Gregg’}

myml.map\_column\_values(mapping\_items={‘name’: map\_name\_values})

#### pardon.Pardon.replace\_values

***pardon.Pardon.replace\_values(replace\_columns)***

Replace values in the column specified with another value.

**Parameters:**

replace\_columns :*dict*

A dictionary object with the column name as a key and the value as a dictionary containing the key as the value to find and the value as the value to replace with. Replacements can be determined by lambdas or static values.

*Example*

number\_replacements = {-1: 0, -99: 0}

text\_replacements = {‘unknown’: ‘Not Specified, ‘N/A’: ‘Not Specified’}

myml.replace\_values(replace\_columns={‘name’: text\_replacements, ‘price’: number\_replacements})

#### pardon.Pardon.remove\_columns

***pardon.Pardon.remove\_columns(columns)***

Remove the columns specified.

**Parameters:**

columns :*list*

A list of the columns to be removed.

*Example*

myml.remove\_columns(columns=[’date’, ‘price\_point’, ‘old\_address’])

#### pardon.Pardon.remove\_correlated\_columns

***pardon.Pardon.remove\_correlated\_columns(max\_correlation=0.9)***

Remove columns with a greater correlation than the max\_correlation argument. Of the 2 correlated columns, the column with the lower [feature\_contribution](#_pardon.MLData.get_feature_contribut) will be removed.

**Parameters:**

max\_correlation :*int, float, default 0.9*

The maximum allowed correlation between 2 columns. Note, 1 is a perfect correlation. This accounts for both positive and negative correlations.

*Example*

myml.remove\_correlated\_columns(max\_correlation=0.8)

#### pardon.Pardon.correlated\_columns

***pardon.Pardon.correlated\_columns(data=None, max\_correlation=0.9)***

Return a list of columns with a greater correlation than the max\_correlation argument. Of the 2 correlated columns, the column with the lower [feature\_contribution](#_pardon.MLData.get_feature_contribut) will be returned.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) *, default None*

The data source used to train a machine learning model. Can be a file path or an object. If None the [train\_data](#_train_data) data set will be used.

max\_correlation :*int, float, default 0.9*

The maximum allowed correlation between 2 columns. Note, 1 is a perfect correlation. This accounts for both positive and negative correlations.

*Example*

correlated\_cols = myml.correlated\_columns(max\_correlation=0.5)

#### pardon.Pardon.rename\_columns

***pardon.Pardon.rename\_columns(column\_names)***

Rename the columns specified.

**Parameters:**

column\_names :*dict*

A dictionary object with the key as the old column name and the value as the new column name.

*Example*

myml.rename\_columns(column\_names={‘date’: ‘Order\_Date’, ‘price’: ‘Price\_in\_pounds’})

#### pardon.Pardon.clean\_column\_names

***pardon.Pardon.clean\_column\_names(remove\_non\_ascii=False)***

Will remove any characters from column names that won’t be accepted by the machine learning models as input, namely characters: “[],<>”.

If duplicates are found after the removal of these characters, they will be renamed with an \_ and the next numeric value in the sequence.

**Parameters:**

remove\_non\_ascii :*bool, default False*

Remove non-ascii characters if found in column names.

*Example*

myml.clean\_column\_names()

#### pardon.Pardon.sort\_by

***pardon.Pardon.sort\_by(columns, desc=False, apply\_to\_test=True)***

*[added v1.2.9]*

Sort data by the columns specified.

**Parameters:**

columns:*list*

A list containing the columns to be sorted by.

desc :*bool, default False*

Sort in a descending order.

apply\_to\_test:*bool, default True*

Apply the sort to your test dataset.

*Example*

myml.sort\_by(columns=[‘price’, ‘customer\_name’], apply\_to\_test=False)

#### pardon.Pardon.correlations

***pardon.Pardon.correlations(columns)***

Produce a correlation plot.

**Parameters:**

columns :*str,**list*

The columns to include in your correlation plot. Leave this to include all columns.

*Example*

myml.correlations()

#### pardon.Pardon.histogram

***pardon.Pardon.histogram(column)***

Produce a histogram plot.

**Parameters:**

column :*str*

The column to produce a histogram for.

*Example*

myml.histogram(column=’customer\_age’)

#### pardon.Pardon.create\_bins

***pardon.Pardon.create\_bins(column, interval, min\_value=’auto’, max\_value=’auto’, new\_column=None, fill\_empty=’mode’)***

Create bins from the column specified. This means a group value for each number, such as 0-5, 5-10, 10-15 and so on. Please note, the bins are inclusive of the max value for each bin, so a value of 5, would be placed into the 0-5 bin.

**Parameters:**

column :*str*

The numeric column to create your bins from.

interval :*int, float*

The interval between bins. For example, 5 would produce bins 5, 10, 15, 20 and so on.

min\_value :*str, int, float, default ‘auto’ {‘auto’ , int, float}*

The minimum value of your group for bins. For example, if you specified 5, intervals would start from 5 and increase by the interval. Using auto will take the minimum value from the column.

max\_value :*str, int, float, default ‘auto’ {‘auto’ , int, float}*

The maximum value of your group for bins. For example, if you specified 100, intervals would increase to 100 from the min\_value by the interval. Using auto will take the maximum value from the column.

new\_column :*str, None, default None*

The name of the column to be created with the bins. Leaving this as None will mean the column in the column argument will be overwritten.

fill\_empty :*str, int, float, default ‘mode’ {‘mean’, ‘median’, ‘mode’, int, float}*

Nulls cannot be binned and so must be filled. Use this argument to determine the value to fill nulls with should they be encountered. This is required even if your column contains no nulls as a defence against future data containing nulls and resulting in an error.

*Example*

myml. create\_bins(column='PatientAge', interval=10, new\_column='age\_10\_year\_bins', fill\_empty=’median’)

#### pardon.Pardon.make\_upper\_case

***pardon.Pardon.make\_upper\_case(columns=[])***

Make the values in the columns specified upper case.

**Parameters:**

columns :*list*

A list of the columns to make the values in upper case. If no columns are supplied, all text columns will be made upper case.

*Example*

myml.make\_upper\_case()

#### pardon.Pardon.drop\_duplicates

***pardon.Pardon.drop\_duplicates(columns=[])***

Drop duplicate rows. If columns are specified, only duplicates occurring across those columns will be considered.

**Parameters:**

columns :*list, default [] (empty list)*

A list of the columns to compare for duplicates. If no columns are supplied, all columns will be included when looking for duplicates.

*Example*

myml.drop\_duplicates()

#### pardon.Pardon.pca

***pardon.Pardon.pca(n\_components=2)***

Use [Principal Component Analysis](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) to reduce the number of features in the dataset. Note, this will only be applied to data just before model training and the underlying data will not be permanently transformed.

**Parameters:**

n\_components :*int, default 2*

An integer of the number of components to reduce the data columns to.

*Example*

myml.pca(n\_components=4)

#### pardon.Pardon.apply\_func

***pardon.Pardon.apply\_func(\*, func, column, new\_column, \*\*kwargs)***

Create a new column or overwrite a column by applying a [lambda expression](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda) or a custom function to an existing column. This function is applied individually to every row in the specified column or all columns.

**Parameters:**

func :[*lambda expression*](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda)*, function*

The [lambda expression](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda) or function to apply to every row in the column argument. If using a function, supply it in an uninstantiated format.

column :*str, None, default None [updated v1.1.3]*

The name of the column to apply the [lambda expression](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda) or custom function. If None, all columns will be passed in the function.

new\_column :*str, None, default None*

The name of new column to be created by applying the [lambda expression](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda) or function to the column. If left as None, the column will be overwritten. If not specifying a column argument, a new\_column must be provided.

func\_name :*str, None, default None [added v1.2.9]*

Give a custom name to your transformation for reference in the Model Script and Model Diagram.

\*\*kwargs :*kwargs*

Any keyword arguments required for your function.

*Example*

myml.apply\_func(column='CallerRelationshipDesc', new\_column='called\_by\_self', func=lambda x: 1 if 'SELF' in str(x).upper() else 0)

#### pardon.Pardon.add\_func

***pardon.Pardon.add\_func(\*, func, apply\_to\_train=True, apply\_to\_test=True, reconcile=True, \*\*kwargs)***

Add a customised function. This will apply your function to the dataset, giving you entire flexibility over the output. The function must take a [pandas DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html) as its first argument, and must return a [pandas DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html). If making significant changes such as removing columns etc, it is strongly recommended to do this last to prevent contention and use the in-built methods available where possible.

**Parameters:**

func :[*lambda expression*](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda)*, function*

The [lambda expression](https://docs.python.org/3.9/tutorial/controlflow.html?highlight=lambda) or function to apply to the dataset.

apply\_to\_train :bool, default True

Apply the function to the training dataset.

apply\_to\_test :bool, default True

Apply the function to the testing dataset.

reconcile :bool, default True *[added v1.0.9]*

Test and reconcile the function. This means it will be run and tested and will take longer but ensures it is valid.

func\_name :*str, None, default None [added v1.2.9]*

Give a custom name to your transformation for reference in the Model Script and Model Diagram.

\*\*kwargs :kwargs

Any keyword arguments required for your function.

*Example*

def my\_new\_func(df, columns, increase\_by=2):

'''a function to increase all values in the columns provided'''

for col in columns:

df[col] = df[col] + increase\_by

return df

myml.add\_func(func=my\_new\_func, apply\_to\_train=True, apply\_to\_test=False, columns=['housing\_median\_age'], increase\_by=5)

#### pardon.Pardon.ignore\_function\_in\_predictions

***pardon.Pardon.ignore\_function\_in\_predictions(ignore\_func, \*\*kwargs)***

*[updated v1.1.7]*

Add the function or name of any functions you want to be ignored when applying future transformations.

**Parameters:**

ignore\_func :*function, str*

The function or name of the function to ignore in future transformations.

kwargs :*keyword arguments*

The keyword arguments for the function.

*Example*

myml.scale\_data(columns=[‘price’, ‘age’])

myml.ignore\_function\_in\_predictions(ignore\_func=myml.scale\_data, columns=[‘price’, ‘age’]))

#### pardon.Pardon.apply\_sentiment

***pardon.Pardon.apply\_func(columns=[], return\_as=’text’)***

Apply [sentiment analysis](https://en.wikipedia.org/wiki/Sentiment_analysis) to the text in the columns specified using the [nltk library](https://www.nltk.org/api/nltk.html). The sentiment analysis will be placed in a new column called <original column name>\_sentiment.

**Parameters:**

columns :*str, list, default [] (empty list)*

The name(s) of the columns to apply the sentiment analysis to.

return\_as:*str, dict, default ‘text’*

Define the output of your sentiment analysis. If ‘text’, each string will have a sentiment analysis determined to be ‘positive’, ‘neutral’, or ‘negative’. You can supply a dictionary object outlining what value to return for each sentiment type. For example, return\_as=dict(‘positive’=3, ‘neutral’=2, ‘negative’=1).

*Example*

myml.apply\_sentiment(columns=[’comments’, ‘description’], return\_as=dict(positive=5, neutral=3, negative=0))

#### pardon.Pardon.combine\_columns

***pardon.Pardon.combine\_columns(column\_1, column\_2, new\_column, sep=’,‘)***

Create a new column by combining the values in column\_1 and column\_2 with a given separator.

**Parameters:**

column\_1 :*str*

The name of the first column to combine values from.

column\_2 :*str*

The name of the second column to combine values from.

new\_column :*str*

The name of the column to be created as a combination of values in column\_1 and column\_2.

sep :*str, default ‘,‘*

The separator used to separate the values in column\_1 and column\_2.

*Example*

myml.combine\_columns(column\_1=’lat’, column\_2=’lon’, new\_column='latlon')

#### pardon.Pardon.split\_column

***pardon.Pardon.split\_column(split\_column, new\_columns, sep=’,‘, drop\_split\_column=False)***

Split the data by a separator into new columns.

**Parameters:**

split\_column :*str*

The name of the column containing the values you wish to split.

new\_columns :*list*

The names of the new columns you wish to create from the split values. Note, there must be at least the number of new columns as there are separators. For example, if the row with the most separators contains 3 separators, this would mean 4 new columns are required to place the split data.

sep :*str, default ‘,‘*

The separator used to split the values in the split column.

drop\_split\_column :*bool, default False*

Drop the split column after the new columns have been created.

*Example*

myml.split\_column(split\_column=’address’, new\_columns=[‘first\_line’, ‘second\_line’, ‘postcode’], sep=’;’, drop\_split\_column=True)

#### pardon.Pardon.get\_sample\_rows

***pardon.Pardon.get\_sample\_rows(n\_rows=1, include\_target=False, as\_json=False)***

Return a random row from the raw input data.

**Parameters:**

n\_rows :*int, default 1*

The number of sample rows to return. If n\_rows is more than the number of available rows, all rows will be returned.

include\_target :*bool, default False*

Include the target column in the returned sample row.

as\_json :*bool, default False [added v1.0.5]*

Return the data in json format.

*Example*

sample\_date = myml.get\_sample\_rows()

Prediction = myml.predict(data=sample\_data)

#### pardon.Pardon.scale\_data

***pardon.Pardon.scale\_data(columns=[], scaler\_type=’standard\_scaler’)***

Scale the data.

**Parameters:**

columns :*str, list, default [] (empty list) [added in v1.0.3]*

The name(s) of the columns to scale. If left empty, all columns (excluding target) will be scaled.

scaler\_type :*str, default ‘standard\_scaler’ {‘standard\_scaler’, ‘min\_max\_scaler’, ‘max\_abs\_scaler’, ‘robust\_scaler’}*

The type of scaler to use. Can use [standard\_scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html), [min\_max\_scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html?highlight=min%20max%20scaler#sklearn.preprocessing.MinMaxScaler), [max\_abs\_scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MaxAbsScaler.html#sklearn.preprocessing.MaxAbsScaler), or [robust\_scaler](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html).

*Example*

myml.scale\_data(scaler\_type=’min\_max\_scaler’)

#### pardon.Pardon.rapid\_ml

***pardon.Pardon.rapid\_ml(model\_fullpath=None, models=[], ignore\_models=[], eval\_metric=’accuracy’, drop\_duplicates=False, max\_features=’all’, use\_ohe=False, label\_encode=True, pca\_n\_components=None, min\_feature\_contribution\_score=None, output\_model\_explanation=False, output\_model\_script=None, make\_text\_upper\_case=False, find\_best\_model\_parameters=False, max\_n\_iterators=None, scoring=’recall\_macro’, scaler\_type=’standard\_scaler’, cleanse\_data=True, train\_model=True, scale\_imbalanced=False, max\_null\_ratio=0.5, max\_correlation=0.9, clear\_data=False, remove\_outliers=False, z\_threshold=3.0, ignore\_target\_outliers =True, n\_clusters=3, cluster\_column\_name=None, cluster\_model=’KMeans’, score\_metric=’accuracy’, cross\_validation=True, \*\*kwargs)***

Automatically clean, encode, and scale data before training and saving a model.

**Parameters:**

model\_fullpath :*str, default None*

The fullpath to where the model will be saved. If not supplied, the model will not be saved.

models :*list, default [] (empty list)*

Specify which type of models to try. Leaving this blank will try all available [models](#_model).

ignore\_models :*list, default [] (empty list)*

Specify which type of [models](#_Available_Models) to ignore.

eval\_metric :*str, default ‘accuracy’*

Specify which evaluation metric is to be used when using cross validation. Check the available metrics using the [available\_evaluation\_metrics](#_pardon.MLData.available_evaluation_) method. The default for a regression model is ‘r2’. The default for classifiers can be changed using the [XGBCLASSIFIER\_DEFAULT](#_XGBCLASSIFIER_DEFAULT), [XGBCLASSIFIER\_BINARY\_DEFAULT](#_XGBCLASSIFIER_BINARY_DEFAULT), [SKLEARN\_CLASSIFIER\_DEFAULT](#_SKLEARN_CLASSIFIER_DEFAULT), and [SKLEARN\_REGRESSION\_DEFAULT](#_SKLEARN_REGRESSION_DEFAULT) attributes. Refer to the [SKLearn documentation](https://scikit-learn.org/stable/modules/model_evaluation.html?) and [XGBoost documention](https://xgboost.readthedocs.io/en/stable/parameter.html) for more information.

drop\_duplicates :*bool, default False*

[Drop duplicate](#_pardon.MLData.drop_duplicates) rows.

max\_features :*int, float, default ‘all’*

The maximum number of columns to keep in the dataset. If an integer is supplied, the top [n performing](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) columns will be retained. If a float is supplied, that proportion of the columns will be retained. For example, max\_features=0.5 will retain the top 50% [best performing features](#_pardon.MLData.set_best_features).

use\_ohe :*bool, default False*

Perform [one hot encoding](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.OneHotEncoder.html) on all viable, string-based columns.

label\_encode :*bool, default True*

Perform [label encoding](https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.LabelEncoder.html) on all viable, string-based columns. If set to False and text data remains, the model will fail during training.

pca\_n\_components :*int, default None*

Perform [Principal Component Analysis](https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html) and reduce the columns to n components.

min\_feature\_contribution\_score :*int, float, default None*

Only retain columns that have a minimum specified [feature contribution](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html) score.

output\_model\_explanation :*bool, str, default False*

Display an output showing [model](#_pardon.MLData.show_model_summary) summary and [model explanations](#_pardon.MLData.explain_model_predict). True will show a summary, passing a .png file fullpath will also save the output to the fullpath specified.

output\_model\_script :*None, str, default None*

Output the [model script](#_pardon.MLData.model_script) to the path specified. If None the model script will not be saved.

make\_text\_upper\_case :*bool, default False*

Make all text in columns upper case as per the [make\_upper\_case](#_pardon.MLData.make_upper_case) method. This is helpful when matching data as matches are case sensitive.

find\_best\_model\_parameters :*bool, default False*

Use [Grid Search Cross Validation](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html) to find the best model hyperparameters. Please note, running this method can take several hours and will consume significant compute so caution advised if running on a cloud platform. Note, if ‘default’ is passed to the ‘models’ argument, the default model will not have the best parameters searched. This is because the same model type will be searched without default parameters already set.

max\_n\_iterators :*int, None, default None*

The maximum number of iterations to run through for each validation when using find\_best\_parameters. A higher number will take longer to complete but will increase the likelihood of the objective converging.

scale\_data :*bool, default True*

Perform data scaling. Unless performed manually, this will only be performed if cleanse\_data is set to True.

scaler\_type :*str, default ‘standard\_scaler’ {‘standard\_scaler’, ‘min\_max\_scaler’}*

[Scale the data](#_pardon.MLData.scale_data).

cleanse\_data :*bool, default True*

Perform data cleansing and format the data so it can be used to train the machine learning models.

drop\_nulls :*bool, default False*

Drop null rows from your input dataset as per the default [drop\_nulls](#_pardon.MLData.drop_nulls) method. If False, nulls will be filled as per the default settings in the [fill\_nulls](#_pardon.MLData.fill_nulls) method.

Note, the [drop\_nulls](#_pardon.MLData.drop_nulls) will not be applied during any predictions and nulls found during prediction will be filled as per the default settings in the [fill\_nulls](#_pardon.MLData.fill_nulls) method.

train\_models :*bool, default True*

Train the models after any data transformations. If set to False, the [train\_model](#_pardon.MLData.train_model) method will not be performed.

scale\_imbalanced :*bool, default False*

If imbalanced class distribution is found, scale using [under](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html) or [over](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html) sampling to balance class distribution before training.

remove\_unhelpful\_columns :*bool, default True*

Remove columns that are deemed unhelpful as per the [remove\_unhelpful\_columns](#_pardon.MLData.remove_unhelpful_colu) method.

max\_null\_ratio *int, float, None, default 0.5*

The max proportion of the column that contains null values. 0.5 would mean that if more than 50% of the column values are null, the column will be deemed as unhelpful and removed. None means this will not be checked.

max\_correlation :*int, float, None, default None*

Remove columns that are correlated by more than the max\_correlation. A correlation of 1 means columns that are 100% correlated are allowed. This calls the [remove\_correlated\_columns](#_pardon.MLData.remove_correlated_col) method. None means this will not be checked.

clear\_data :*bool, default False*

Clear the retained input data from the model before saving to help significantly reduce the model object file size. If this is used many of the model’s methods will no longer be available and the model can no longer be changed or updated.

remove\_outliers :*bool, default False*

Remove outliers from your dataset based on the z\_threshold.

z\_threshold : *int, float, default 3.0*

Rows with a [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) more than that specified will be determined as an outlier and removed using the [remove\_outliers](#_pardon.MLData.remove_outliers) method. A [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) of 3.0 means the value is 3 standard deviations or more away from the column mean. This will only be performed if remove\_outliers=True.

ignore\_target\_outliers : *bool, default True*

Specify if you want to ignore the target column when finding outliers in the data. ignore\_target\_outliers=True means the target column will not be searched and will not have any outliers removed.

n\_clusters : *int, None, default 3*

The number of clusters to create. Each row will be assigned a cluster group. This option triggers the [create\_clusters](#_pardon.MLData.create_clusters) method. None means this will not be performed.

cluster\_column\_name : *str, None, default None*

The name of the column to add the clustered group to. This option triggers the [create\_clusters](#_pardon.MLData.create_clusters) method. None means this will not be performed.

cluster\_model :*str, default ‘KMeans’*

The [clustering model](#_Clustering_Models) to use to create your clusters.

score\_metric :*str, default ‘accuracy’*

Specify which metric is to be used when determining the best performing model in testing. You can see the available scoring metrics using the [available\_scoring\_metrics](#_pardon.MLData.available_scoring_met) method. The default for regression models is ‘r2’. The best performing model will be that with the highest test score unless using scores of errors (such as ‘neg\_mean\_squared\_error’) in which case the lowest score will be considered best. You can view the score metrics that get minimised at the [Pardon.REDUCTION\_SCORE\_ON](#_REDUCTION_SCORE_ON) attribute.

cross\_validation :*bool, default True*

Perform cross validation during model training. This should only ever be switched off if you simply testing an idea and want to reduce training time. It is advised you always perform cross validation.

n\_splits :*int, default 10*

The number of splitting iterations during cross validation.

n\_repeats :*int, default 1*

The number of times to repeat the cross validation.

\*\*kwargs :*keyword arguments [added v1.0.5]*

This allows the user to tweak certain arguments used. Presently the only valid arguments are model\_params, fill\_text\_with, and fill\_numeric\_with.

*Example*

myml.rapid\_ml(model\_fullpath=rapid\_ml.pkl, use\_ohe=True, min\_contribution\_score=100, eval\_metric=’accuracy’, clear\_data=True)

#### pardon.Pardon.find\_best\_model\_parameters

***pardon.Pardon.find\_best\_model\_parameters(model=’default’, max\_n\_iterators=None, eval\_metric=’accuracy’, search\_space=None, n\_splits=10, n\_repeats=3)***

Find and set the best hyperparameters for the specified model using [Grid Search Cross Validation](https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html). It is worth noting this process can take hours to complete.

**Parameters:**

model :*str, class, default ‘default’*

The name of the model type to find the best hyperparameters for. Default means an [XGBoost](https://xgboost.readthedocs.io/en/stable/python/python_intro.html) Classifier will be used for classification problems, and a [Linear Regression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression) model for regression. You can also pass a custom algorithm object to the model parameter, but a search\_space is required when doing so.

max\_n\_iterators :*int, None, default None*

The maximum number of iterations to run through for each validation. A higher number will take longer to complete but will increase the likelihood of the objective converging. None means the model’s default settings will be used.

eval\_metric :*str, default ‘accuracy’*

The evaluation metric determining model performance.

search\_space :*dict, None, default None*

The model parameters to use during the cross validation. If None, the parameters searched will be chosen automatically. You can see the standard search spaces for each model using the [search\_spaces](#_pardon.MLData.search_spaces) method.

n\_splits :*int, default 10*

The number of splitting iterations during cross validation.

n\_repeats :*int, default 3*

The number of times to repeat the cross validation.

*Example*

xgb\_space = dict()

xgb\_space['colsample\_bytree'] = [0.2, 0.5, 0.8]

xgb\_space['max\_depth'] = [4, 5, 8]

xgb\_space['learning\_rate'] = [0.3, 0.5, 0.9]

myml.find\_best\_model\_parameters(model=’XGBClassifier’, search\_space= xgb\_space)

#### pardon.Pardon.train\_model

***pardon.Pardon.train\_model(model=’default’, use\_best\_parameters=True, eval\_metric=’accuracy’, scale\_imbalanced=False, model\_params=None, cross\_validation=True, n\_splits=10, n\_repeats=3)***

Train the machine learning model.

**Parameters:**

model :*str, class, default ‘default’*

The name of the model type to train or a valid classification machine learning (ML) model class. If using a custom ML model, the model must have the fit, predict\_proba (for classification) or predict (for regression), set\_paramas, and score methods. Default means an [XGBoost](https://xgboost.readthedocs.io/en/stable/python/python_intro.html) Classifier will be used for classification problems, and a [Linear Regression](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html?highlight=linear%20regression#sklearn.linear_model.LinearRegression) model for regression.

use\_best\_parameters :*bool, default True*

Use the best hyperparameters for the model. Will only be available if the [pardon.Pardon.find\_best\_model\_parameters](#_pardon.MLData.find_best_model_param) method has been completed.

eval\_metric :*str, default ‘accuracy’*

The evaluation metric to use during cross validation in model training. Use the [available\_evaluation\_metrics](#_pardon.MLData.available_evaluation_) method to see the available options. The default for regression models will be set to ‘r2’. The default for classifiers can be changed using the [XGBCLASSIFIER\_DEFAULT](#_XGBCLASSIFIER_DEFAULT), [XGBCLASSIFIER\_BINARY\_DEFAULT](#_XGBCLASSIFIER_BINARY_DEFAULT), [SKLEARN\_CLASSIFIER\_DEFAULT](#_SKLEARN_CLASSIFIER_DEFAULT), and [SKLEARN\_REGRESSION\_DEFAULT](#_SKLEARN_REGRESSION_DEFAULT) attributes. Refer to the [SKLearn documentation](https://scikit-learn.org/stable/modules/model_evaluation.html?) and [XGBoost documention](https://xgboost.readthedocs.io/en/stable/parameter.html) for more information.

scale\_imbalanced :*bool, default False*

If imbalanced class distribution is found, scale using [under](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html) or [over](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html) sampling to balance class distribution before training.

model\_params :*dict, default None*

A dictionary object containing custom model parameters and values to apply to the model before training.

cross\_validation :*bool, default True*

Perform cross validation during model training. This should only ever be switched off if you simply testing an idea and want to reduce training time. It is advised you always perform cross validation.

n\_splits :*int, default 10*

The number of splitting iterations during cross validation.

n\_repeats :*int, default 3*

The number of times to repeat the cross validation.

*Example*

myml.train\_model(model=’RandomForestClassifier’, scale\_imbalanced=True)

#### pardon.Pardon.explain\_model\_predictions

***pardon.Pardon.explain\_model\_predictions(refresh=False, output\_fullpath=None, max\_features=20)***

Show an explanation of model predictions using [SHAP values](https://shap.readthedocs.io/en/latest/index.html) or [feature importance](https://scikit-learn.org/stable/modules/permutation_importance.html) dependent on the type of model that has been trained.

**Parameters:**

refresh :*bool, default False*

Re-calculate the [SHAP](https://shap.readthedocs.io/en/latest/index.html) values. This happens by default if they have not already been calculated else the ones previously calculated will be used.

output\_fullpath :*str, default None*

The output fullpath to where you want your model explanations to be saved. The output fullpath should end with a reference to the filename as a png file. If None the output will be displayed but not saved.

max\_features :*int, default 20*

Maximum number of features to include in the model output.

*Example*

myml.explain\_model\_predictions(output\_fullpath=’model\_explainer.png’)

#### pardon.Pardon.model\_summary

***pardon.Pardon.model\_summary()***

Display a summary of the model’s performance. This will show a [confusion matrix](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html) and [classification report](https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html?highlight=classification%20report#sklearn.metrics.classification_report).

**Parameters:**

output\_fullpath :*str, default None*

The output fullpath to where you want your model summary to be saved. The output fullpath filename filetype should be a .png file. If None the output will be displayed but not saved.

*Example*

myml.model\_summary()

#### pardon.Pardon.save\_model

***pardon.Pardon.save\_model(model\_fullpath, min\_test\_metric=’accuracy’, min\_test\_score=0, clear\_data=False)***

Save the model to the specified model fullpath.

**Parameters:**

model\_fullpath :*str*

The model fullpath to where you want your model to be saved. The output fullpath should end with a reference to the filename as a [pkl file](https://docs.python.org/3/library/pickle.html).

min\_test\_metric :*str, default ‘accuracy’*

The scoring metric to determine if the model has met the minimum required accuracy.

min\_test\_score :*float, int, default None*

The minimum required model test score for the min\_test\_metric. The model will only be saved if the model has min\_test\_metric score HIGHER than the min\_test\_score. For error metrics as per the [Pardon.REDUCTION\_SCORE\_ON](#_REDUCTION_SCORE_ON) attribute, the model will only be saved if the error was LOWER than min\_test\_score.

clear\_data :*bool, default False*

Clear the retained input data from the model before saving to help significantly reduce the model object file size. If this is used many of the model’s methods will no longer be available and the model can no longer be changed or updated.

*Example*

myml.save\_model(model\_fullpath=’model.pkl’, clear\_data=True, min\_test\_metric=’r2’, min\_test\_score=0.7)

#### pardon.Pardon.required\_columns

***pardon.Pardon.required\_columns()***

Return a list of the columns required for the model to be able to make a prediction from [input data](#_raw_data).

*Example*

cols\_for\_prediction = myml.required\_columns()

#### pardon.Pardon.predict

***pardon.Pardon.predict(data, audit\_fullpath=None, check\_fail\_ons=True)***

Returns a [pardon.Prediction](#_pardon.Prediction) class object containing a prediction from the data.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data source used to train a machine learning model. Can be a file path or an object.

audit\_fullpath :*str, default None*

The fullpath to an output where you want to save a record of all predictions made by the model. This will output a [sqlite database](https://www.sqlite.org/index.html) which will store all predictions. If the file does not exist one will be created, else the one specified will be used. A fullpath with a filename of type db is required.

check\_fail\_ons :*bool, default True*

Check the data for [FailOns](#_pardon.InvalidModel) added to the model. If False, the [Fail Ons](#_pardon.InvalidModel) will not be checked.

*Example*

myml.predict(data=’prediction\_data.csv’, audit\_fullpath=’predictions.db’)

#### pardon.Pardon.model\_script

***pardon.Pardon.model\_script(script\_fullpath, use\_shap\_values=False)***

Produces an output txt file containing the model details and data transformation script.

**Parameters:**

script\_fullpath :*str*

The fullpath to where you want to save the model script.

use\_shap\_values :*bool, default False [added v1.0.5]*

If applicable, use SHAP values when generating feature importance. This will take significantly longer to run.

*Example*

myml.model\_output(script\_fullpath=’model\_script.txt’)

#### pardon.Pardon.model\_parameters

***pardon.Pardon.model\_parameters()***

Return the trained model parameters.

*Example*

model\_params = myml.model\_parameters()

#### pardon.Pardon.get\_default\_model\_params

***pardon.Pardon.get\_model\_params()***

*[added in v1.0.5]*

Returns a dict object containing the default model’s hyperparameters.

*Example*

mod\_params = myml.get\_default\_model\_params()

#### pardon.Pardon.set\_default\_model\_params

***pardon.Pardon.set\_model\_params(\*, reset\_all\_params =False, \*\*kwargs)***

*[added in v1.0.5]*

Used to set the default model’s hyperparameters.

**Parameters:**

reset\_all\_model\_params :*bool, default False*

Set True to reset all model hyperparameters back to the implementation default from the relevant API.

\*\*kwargs :*keyword arguments*

Set the relevant hyperparameters for the default model.

*Example*

myml.set\_default\_model\_params(reset\_all\_params=True, min\_child\_weight=1, reg\_alpha=1.5)

#### pardon.Pardon.model\_diagram

***pardon.Pardon.model\_diagram(diagram\_fullpath=None, fontsize=10)***

Produces a diagram of the model data flow.

**Parameters:**

diagram\_fullpath :*str, default None {None, ‘.svg’, ‘.eps’, ‘.png’, ‘.pdf’ ,‘.jpg’ }*

The fullpath to an output where you want to save the model diagram. The recommended filetype is ‘.svg’. If None the output will be displayed but not saved.

fontsize :*int, default 10*

The font size to use in the diagram.

*Example*

myml.model\_diagram(diagram\_fullpath=’my\_diagram.svg’)

#### pardon.Pardon.model\_learning\_curve

***pardon.Pardon.model\_learning\_curve(n\_splits=5, n\_tests=10, score\_metric='accuracy')*** *[added in v1.0.3]*

Plots the training and cross-validation scores for varying test and training sizes. Note, depending on the type of model, this can take a significant amount of time to complete.

**Parameters:**

n\_splits :*int, default 5*

The number of splitting iterations during cross validation.

n\_tests :*int, default 10*

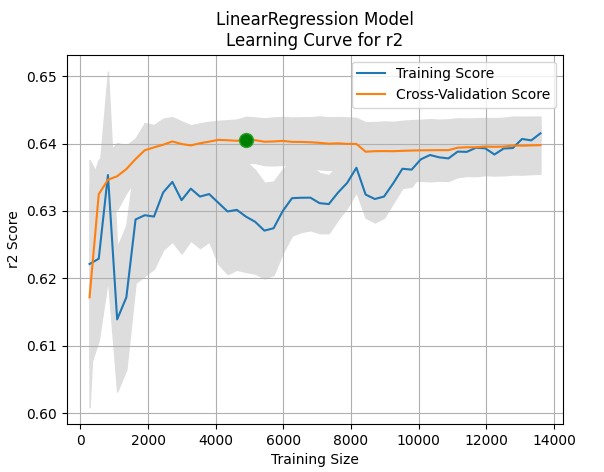
The number of different test sizes to try. The test sizes are then determined by (total rows x 0.8) / n\_tests.  
For example, if your data has 10000 rows, and you set n\_tests = 5. This means (10000 x 0.8) = 8000, 8000 / 5 = 1600, therefore the following test sizes will be used: 1600, 3200, 4800, 6400, 8000.

score\_metric :*str, default ‘accuracy’*

Specify which metric is to be used when performing the cross-validation. You can see the available scoring metrics using the [available\_scoring\_metrics](#_pardon.MLData.available_scoring_met) method. The default for regression models is ‘r2’.

*Example*

myml.model\_learning\_curve(score\_metric=’r2’, n\_tests=50)



#### pardon.Pardon.get\_feature\_importance

***pardon.Pardon.get\_feature\_importance(suppress\_errors=False)***

Produces a dictionary object containing the column and its feature importance score. Note, this is only available after a [model](#_model) has been [trained](#_pardon.MLData.train_model).

**Parameters:**

suppress\_errors :*bool, default False*

If errors are suppressed, any errors will mean an empty dictionary object is returned instead of an error being raised.

*Example*

feat\_importances = myml.get\_feature\_importance(suppress\_errors=True)

print(feat\_importances)

#### pardon.Pardon.add\_data

***pardon.Pardon.add\_data(data, overwrite\_existing=False, train\_model=True)***

*[added v1.0.5]*

Add or overwrite data to an existing model.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The new data to be added to the Pardon class.

overwrite\_existing :*bool, default False*

Overwrite the data that currently exists in your model.

train\_model :*bool, default True*

Train/retrain your current ML model including the new data.

*Example*

myml.add\_data(data=’data\_v2.csv’)

#### pardon.Pardon.get\_feature\_contribution

***pardon.Pardon.get\_feature\_contribution(suppress\_errors=False)***

Produces a dictionary object containing the columns contribution to the prediction of the target based on the [ANOVA F-value score](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_classif.html#sklearn.feature_selection.f_classif) for classification and [F-statistic](https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.f_regression.html#sklearn.feature_selection.f_regression) for regression. Note, data should have been encoded prior to using this method.

**Parameters:**

suppress\_errors :*bool, default False*

If errors are suppressed, any errors will mean an empty dictionary object is returned instead of an error being raised.

*Example*

feat\_contributions = myml.get\_feature\_contribution()

print(feat\_importances)

#### pardon.Pardon.search\_spaces

***pardon.Pardon.search\_spaces()***

Returns a dictionary object containing each of the models and the associated parameters that are searched during the [find\_best\_model\_parameters](#_pardon.MLData.find_best_model_param_1) method.

*Example*

search\_space = myml.search\_space()

print(search\_space)

#### pardon.Pardon.trasnformations

***pardon.Pardon.transformations()***

Return a dict showing the transformations applied by your model and those ignored. Ignored functions are those that were used in the data transformations but will not be applied to data coming in for predictions.

*Example*

transforms = myml.transformations()

print(transforms)

#### pardon.Pardon.plot\_data

***pardon.Pardon.plot\_data(x=None, y=None, c=None, size=None, lat=None, lon=None, n\_predictions=’all’, include\_trendline=True, show\_as=’scatterplot’, data=None, as\_prediction=False, n\_bins=’auto’, columns=[], randomise\_columns=[], randomise\_strategy=’normal\_distribution’, include\_original=True, lineplot\_estimator=’count’, apply\_transformations=True, remove\_outliers=False, z\_threshold=3.0, outlier\_columns=[])***

Plots your data onto a chart. Note, if using your target column, each prediction will be unique as rows are only used once for each prediction. The max n\_predictions is therefore the number of rows in your input dataset.

**Parameters:**

x :*str, None, default None*

The column to plot on the x axis of the scatterplot.

y :*str, None, default None*

The column to plot on the y axis of the scatterplot.

c :*str, None, default None*

The column to use as data colours on the scatterplot, lineplot, or map.

size :*str, None, default None [added v1.0.3]*

The column to use as data marker size on the scatterplot, lineplot, or map.

lat :*str, None, default None*

The column to use as the latitude data for the map. Only used when show\_as = ‘map’.

lon :*str, None, default None*

The column to use as the longitude data for the map. Only used when show\_as = ‘map’.

n\_predictions :*int, str, default ‘all’ {int, ‘all’}*

The number of predictions to plot on your scatterplot. Pass ‘all’ if you want to include the prediction for every row from your input dataset. Will only be relevant when including the target in one of your columns as predictions will not be made otherwise.

include\_trendline :*bool, default True*

Include a trendline on your scatterplot.

show\_as :*str, default ‘scatterplot’ {‘scatterplot’, ‘map’, ‘histogram’, ‘correlation’, ‘timeseries’}*

The type of chart to display the data.

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)*, default None*

The data you want to get a plot. If not specified, data from the original raw input data will be used.

as\_prediction :*bool, default False*

If including the target column on your chart, as\_prediction=True means the predicted values will be used. as\_prediction=False means the raw data from the target column will be used. If you use the target name and ‘[predicted]’ in the x, y, c, or size arguments, the predicted value will be used. For example, if your target is set to “house\_price”, passing “house\_price [predicted]”, the prediction for the target will be displayed.

n\_bins :*int, str, default ‘auto’ {int, ‘auto’}*

The number of bins to split the histogram. Note, if not using a numeric column, n\_bins will be set to the number of unique items.

columns :*list, default [] (empty list)*

Only used when displaying correlations. Specify which columns to include in the correlation heatmap.

randomise\_columns :*list, default [] (empty list)*

Choose columns to be randomised to test the impact of changing values on your model’s predictions.

randomise\_strategy :*str, default ‘normal\_distribution’ {‘normal\_distribution’ , ‘normal\_distribution\_in\_range’, ’random’}*

The strategy determining how random values will be chosen. If normal distribution is selected, for numerical columns, a random value based on a normal distribution will be chosen. If normal distribution in range is selected, the minimum and maximum values allowed in the normal distribution will be the minimum and maximum values found in the column. For random, a random number between the minimum and maximum values will be chosen. For text columns, one item will be chosen from all items in the column, meaning that items that appear more often in the dataset, are more likely to be chosen for a more realistic choice. Note, this is only available for the scatterplot chart type.

include\_original :*bool, default True*

When using randomise\_columns, include\_original=True means a scatterplot will be produced both for your randomised data set, as well as the original data set.

lineplot\_estimator :*str, default ‘mean’ {‘mean’, ‘median’, ‘mode’, ‘sum’, ‘count’, ‘min’, ‘max’, ‘z\_score’} [renamed from timeseries\_estimator to lineplot\_estimator in v1.0.3]*

When using a lineplot chart, specify the type of aggregation to perform on the y axis. For text columns, this will automatically be set to ‘count’. When using z\_score, the max z\_score for each data point will be used.

apply\_transformations :*bool, default True*

Apply the transformations to the data passed to the data parameter. If apply\_transformations=False, none of the transformations from the script will be applied to your data. Use this if you have already applied the transformations manually to the data passed to the data parameter. If data=None and apply\_transformations=False, the model’s transformed training and test data will be used. If apply\_transformations=True and data=None, the raw input data will be used as though it is a new, unseen dataset, and all transformations will be applied. If using randomised, apply\_transformations will be set to True for the dataset regardless as a prediction needs to be made.

remove\_outliers :*bool, default False [renamed to remove\_outliers from ignore\_outliers in v1.0.3]*

Remove any outliers in your dataset. Outliers are determined by the z\_threshold.

z\_threshold :*int, float, default 3.0*

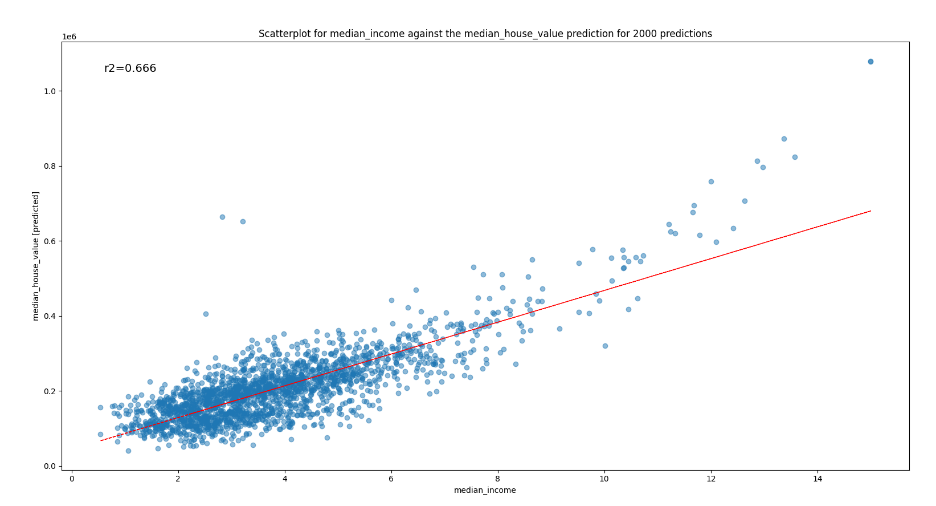
Rows with a [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) more than that specified will be determined as an outlier and removed using the [remove\_outliers](#_pardon.MLData.remove_outliers) method. A [z score](https://docs.scipy.org/doc/scipy/reference/generated/scipy.stats.zscore.html) of 3.0 means the value is 3 standard deviations or more away from the column mean.

outlier\_columns :*list, default [] (empty list)*

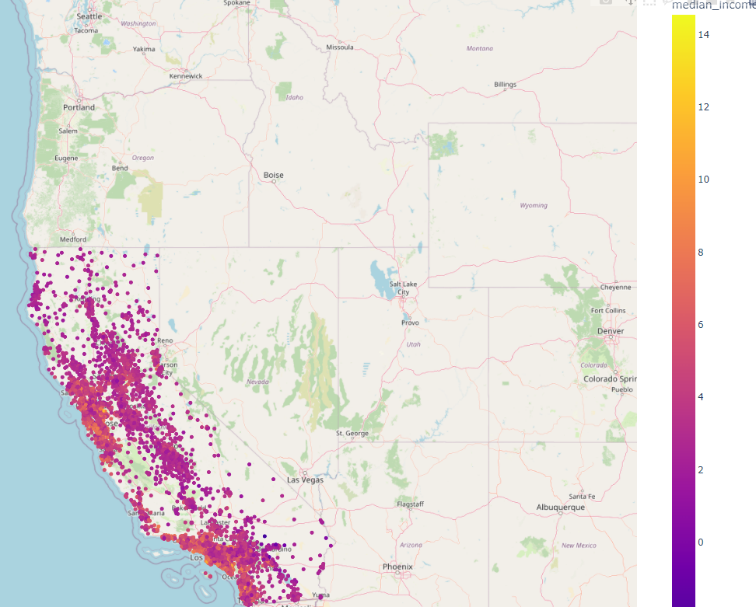
Choose columns to be checked for outliers. If this is empty and ignore\_outliers=True, all columns will be checked for outliers.

*Example*

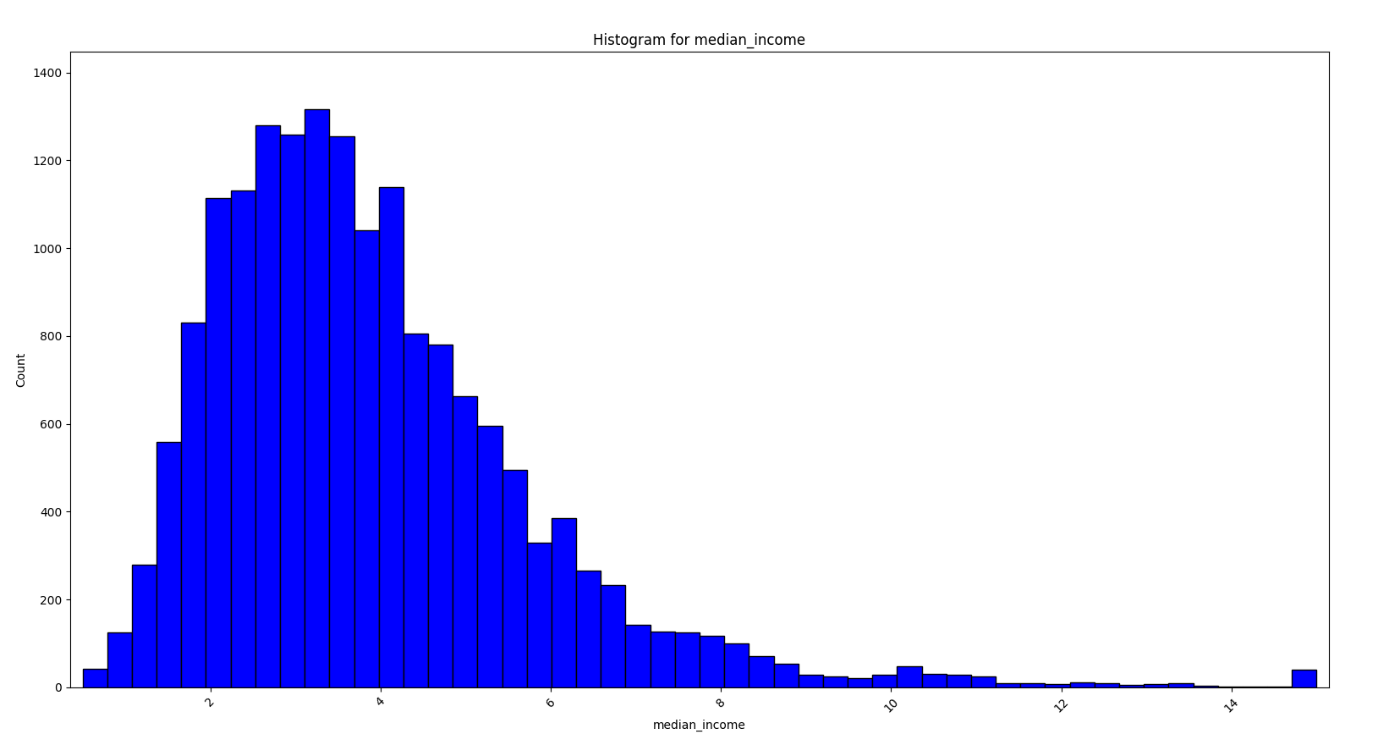
myml.plot\_data(x=’median\_income’, y=’median\_house\_value’, n\_predictions=2000, as\_prediction=True)



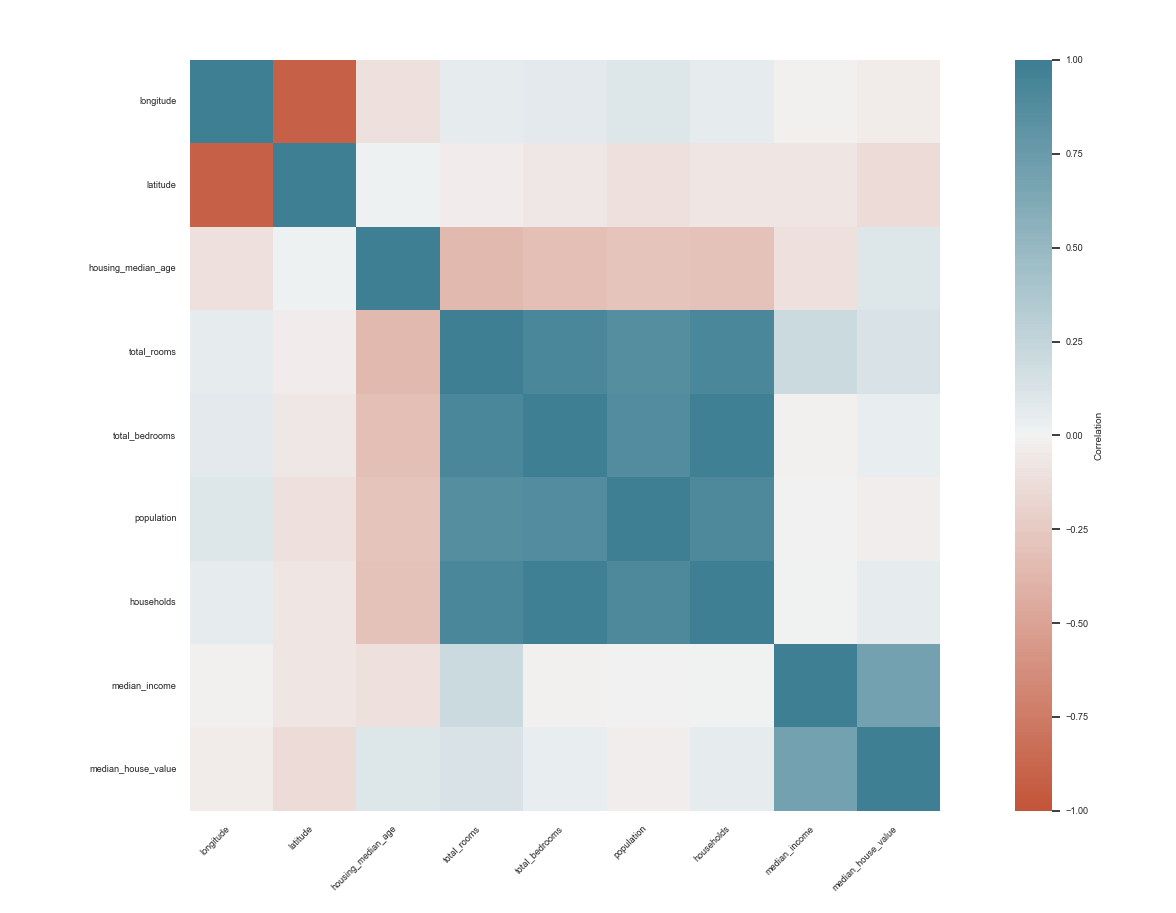
myml.plot\_data(lat=’latitude’, lon=’longitude’, c=’median\_income’, n\_predictions=2000, show\_as=’map’)

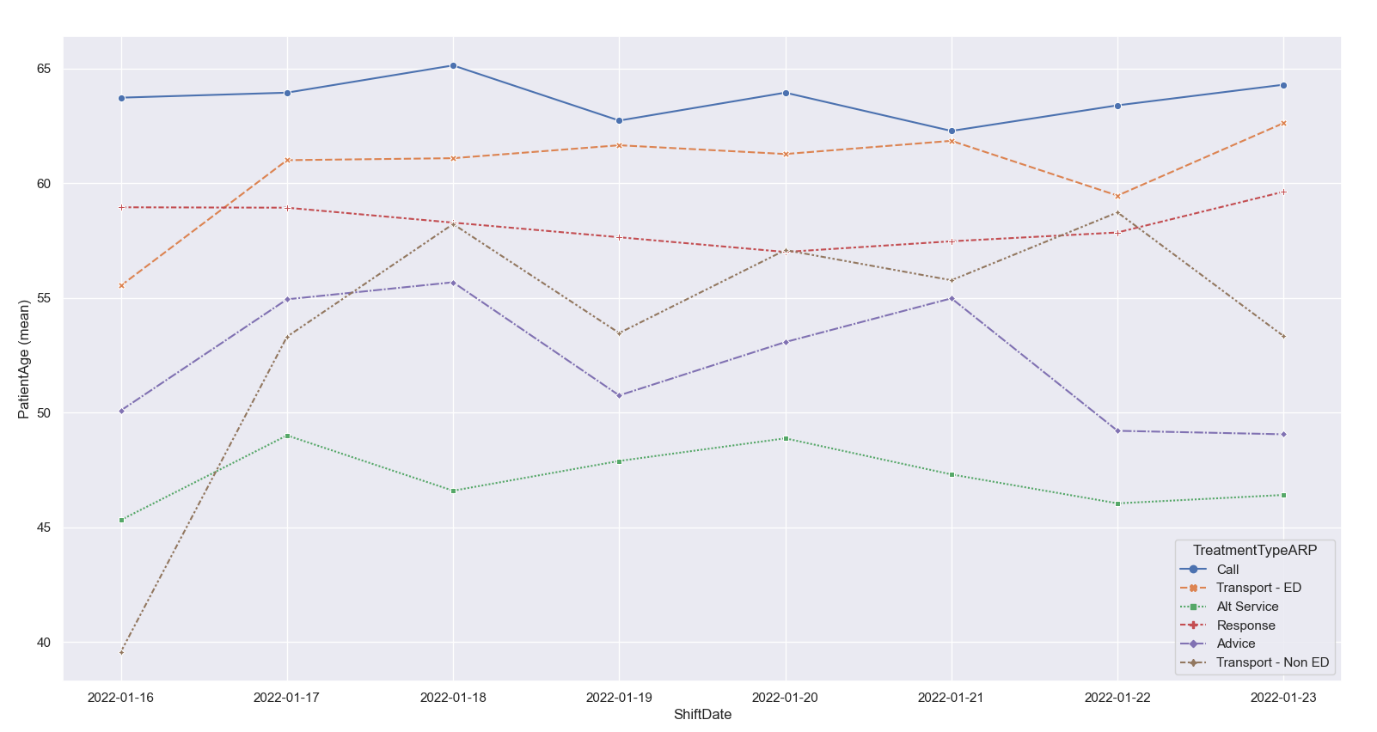
****

myml.plot\_data(x='median\_income', show\_as='histogram', n\_bins=50)



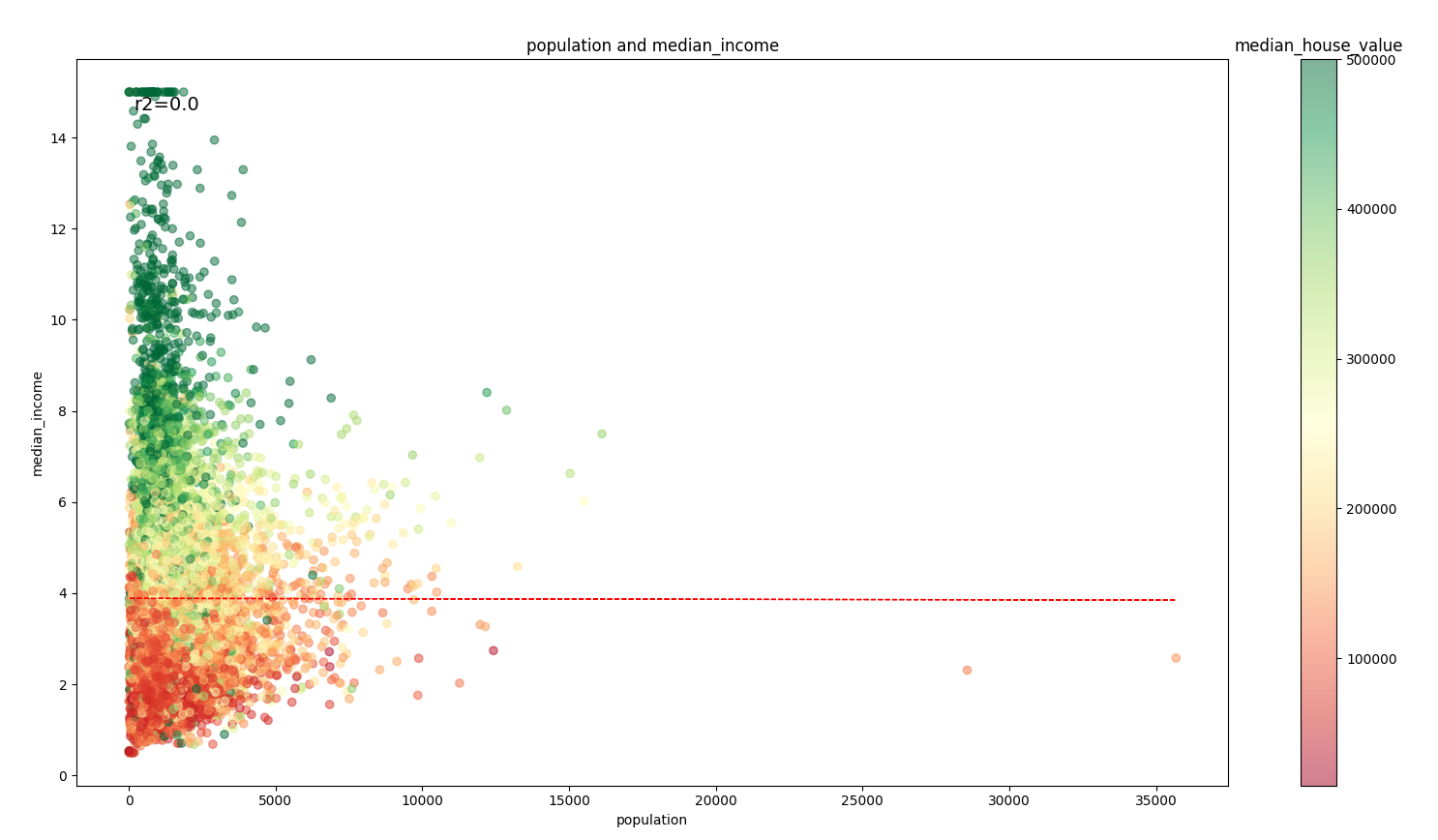
myml.plot\_data(show\_as=’correlation’)

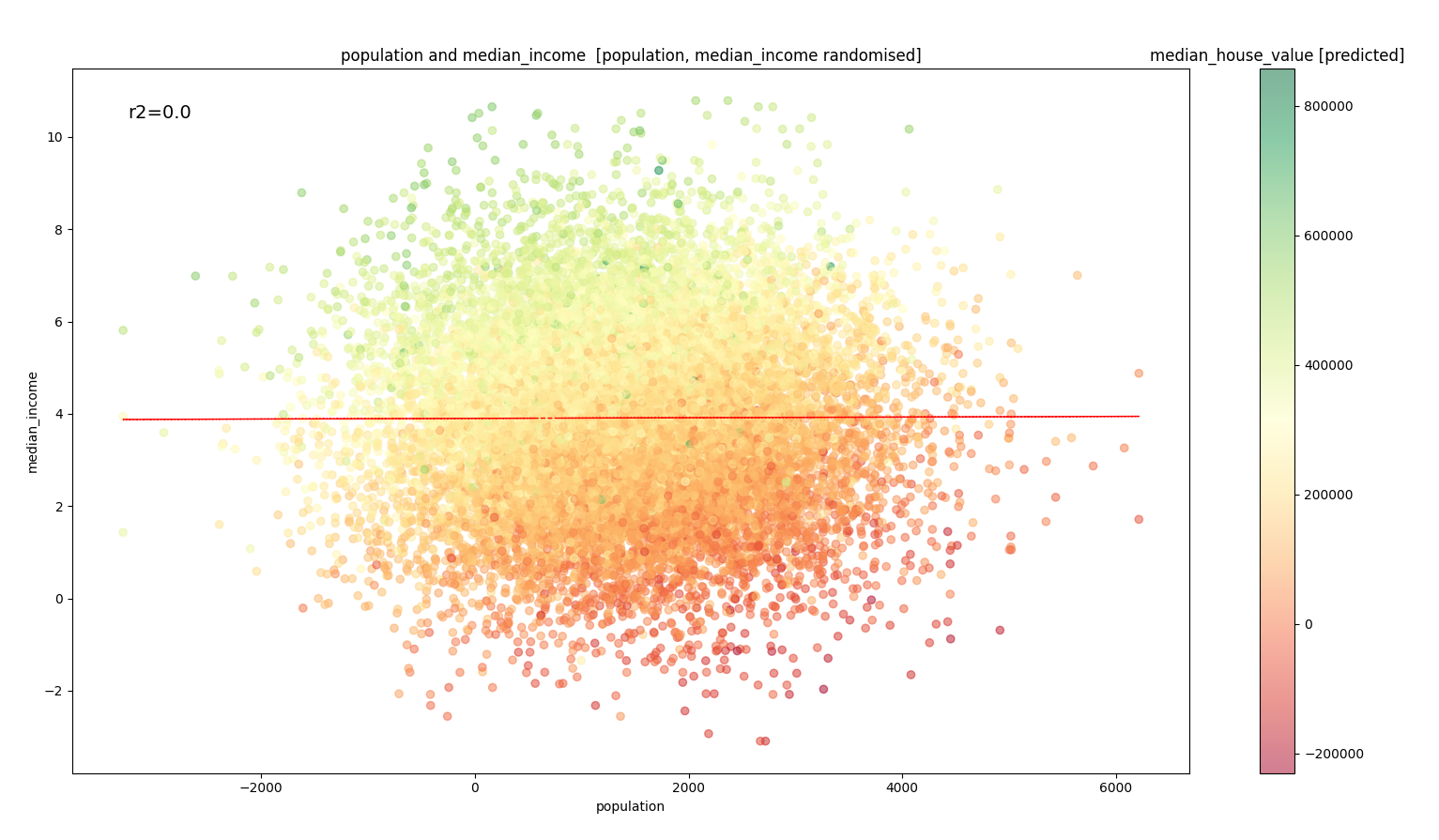




myml.plot\_data(x='ShiftDate', y='PatientAge', c='TreatmentTypeARP', show\_as=’lineplot’, lineplot\_estimator='mean')

myml.plot\_data(x='population', y='median\_income', c='median\_house\_value', show\_as='scatterplot', randomise\_columns=['population', 'median\_income'], include\_original=True, as\_prediction=False)





#### pardon.Pardon.output\_data

***pardon.Pardon.output\_data(data=None, n\_predictions=’all’, output\_fullpath=None, as\_prediction=True, as\_transform=False, apply\_fail\_ons=False)***

Outputs your data to a pandas.DataFrame as well as a new file if requested.

**Parameters:**

output\_fullpath:*str, None, default None {‘.csv’, ‘.txt’, None}*

The output fullpath you want the predictions to be saved to. If None the output will be printed to the terminal. Only .csv or .txt outputs are supported.

n\_predictions :*int, str, default ‘all’ {int, ‘all’}*

The number of predictions to plot. Pass ‘all’ if you want to include the prediction for every row from your input dataset. This is only relevant if as\_prediction=True.

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)*, default None*

The data you want to output. If not specified, data from the original raw input data will be used.

as\_prediction :*bool, default True*

If including the target column on your chart, as\_prediction=True means the predicted values will be used. as\_prediction=False means the raw data from the target column will be used.

as\_transform :*bool, default False*

Output the data including all the data transformations that were applied through the model’s script.

apply\_fail\_ons:*bool, default False [added in v1.0.1]*

Apply any [FailOns](#_pardon.InvalidModel) to your data prior to output. Equivalent to running the [apply\_fail\_ons](#_pardon.Pardon.apply_fail_ons) method.

*Example*

myml.output\_data(output\_fullpath=’predictions.csv’, as\_prediction=False)

#### pardon.Pardon.create\_clusters

***pardon.Pardon.create\_clusters(column\_name, n\_clusters=3, model=’KMeans’)***

Cluster data into n number of groups. Select from the available clustering models.

**Parameters:**

column\_name:*str*

The name of the column you want to add the clustered groups to.

n\_clusters :*int, default 3*

The number of clusters to create for the dataset. Each row will be assigned one of the clustered groups.

model :*str, default ‘KMeans’*

The [clustering model](#_Clustering_Models) used to create your clusters.

*Example*

myml.create\_clusters(column\_name=’clustered\_group\_num’, n\_clusters=5, model=’birch’)

#### pardon.Pardon.transform

***pardon.Pardon.transform(data, include\_ignored\_transformations=False)***

Apply the model script data transformations to a data set.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data to apply the model’s data transformations.

include\_ignored\_transformations :*bool, default False*

*[added v1.2.5]*

Include transformations that are ignored in predictions.

verbose :*bool, default False*

*[added v1.2.6]*

Show which transformations are being applied.

*Example*

output = myml.transform(data=’new\_data.csv’)

print(output)

*See also:* [*model\_script*](#_pardon.MLData.model_script)

#### pardon.Pardon.add\_fail\_on

***pardon.Pardon.add\_fail\_on(fail\_on, strategy=’fail’, warn\_fullpath=None)***

Add a [FailOn](#_pardon.InvalidModel) item to your model. [FailOns](#_pardon.InvalidModel) will be applied to the input dataset before a [prediction](#_pardon.MLData.predict) unless explicitly requested not to.

**Parameters:**

fail\_on :[*FailOn*](#_pardon.InvalidModel) *class*

The [FailOn](#_pardon.InvalidModel) object that will be applied to your data.

strategy:*str, default ‘fail’ {‘fail’, ‘warn’}*

How to react to a FailOn invoking a failure. Fail will raise an error and all processing will stop. Warn means a warning will be printed to the terminal. A line will be written in the warn\_fullpath file if provided.

warn\_fullpath:*str, None, default None*

When using the strategy of warn, where to output the [FailOn](#_pardon.InvalidModel) errors.

*Example*

fail\_on = pardon.FailOn(column='median\_house\_value', operator='nb', value=[4, 6], calculation='mode')

myml.add\_fail\_on(fail\_on=fail\_on, strategy=’warn’, warn\_fullpath=’median\_house\_values\_nb\_5\_6.csv’)

#### pardon.Pardon.delete\_fail\_on

***pardon.Pardon.delete\_fail\_on(fail\_on\_id)***

Delete a [FailOn](#_pardon.InvalidModel) item from your model.

**Parameters:**

fail\_on\_id :*int*

The [FailOn](#_pardon.InvalidModel) object ID that you added using the [pardon.Pardon.add\_fail\_on](#_pardon.MLData.add_fail_on) method. You can check the fail\_on\_ids using the [pardon.Pardon.fail\_ons](#_fail_ons) attribute.

*Example*

myml.delete\_fail\_on(fail\_on\_id=3)

#### pardon.Pardon.apply\_fail\_ons

***pardon.Pardon.apply\_fail\_ons(data)***

Apply the [FailOn](#_pardon.InvalidModel) items to the specified dataset.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data to apply the [FailOn](#_pardon.InvalidModel) to.

*Example*

myml.apply\_fail\_ons(data=’new\_data\_for\_predictions.csv’)

## pardon.Prediction

The class object returned from the [pardon.Pardon.predict](#_pardon.MLData.predict) method.

### pardon.Prediction Attributes

#### target

The predicted target class.

#### model\_identifier

The unique identifier for the model that made the prediction.

#### features

A dictionary object containing key value pairs of the input data passed for prediction.

#### model\_classes

The classes as used in the model training**.**

#### class\_labels

The actual name of the classes as per the input data**.**

#### predicted\_datetime

The datetime the prediction was made**.**

#### y

If training or test data was used, y is the labelled data containing the actual class of the input data.

#### actual

The correct class for the input data. Only available if y is provided.

#### is\_correct

Is the prediction correct. Only available if y is provided.

#### delta

The difference between the prediction and actual. Only used with regression models when y is provided.

#### predicted

The predicted class or value for the input data. If a classification model, this is the class with the highest probability from the prediction.

#### probabilities

The probability of each class in the prediction. Only used in classification models that produce predictions with probabilities.

#### audit

The [pardon.Audit](#_pardon.MLData._Audit_Attributes) class containing information on the predictions audit.

#### error

Details any errors occurred during the prediction. This should return None unless an error was encountered.

### pardon.Prediction Methods

#### pardon.Prediction.api\_output

***pardon.Pardon.Prediction.api\_output(return\_object=’array’, include\_probabilities=True, include\_title=False, as\_json=True)***

Return the output ready to be delivered to an API response.

**Parameters:**

return\_object :*str, default ‘array’ {‘array’, ‘dict’, ‘list’}*

The format of the response. Can be array, dict, or list.

include\_probabilities :*bool, default True*

For classification problems decide if you want to include a probability score for each class. If False, the class with the highest probability will be returned.

include\_title :*bool, default False*

Include a title in the output. This will add the word ‘Prediction’ for classification predictions with no probabilities and regression models, and will add the words ‘Class’, and ‘Probability of Class’, for classification models with probabilities returning as a list.

as\_json :*bool, default True*

Return the output object in json format.

*Example*

prediction = myml.predict(data=’predict\_data.csv’)

response = prediction.api\_output()

print(response)

**Classification model examples:**

prediction.api\_output(include\_probabilities=True, return\_object='list', include\_title=True)

*[[['TreatmentType', 'Probability of Class'], ['Hear & Treat', 0.67], ['See & Convey', 64.21], ['See & Treat', 35.12]]]*

prediction.api\_output(include\_probabilities=True, return\_object='list', include\_title=False)

*[[['Hear & Treat', 0.48], ['See & Convey', 59.04], ['See & Treat', 40.48]]]*

prediction.api\_output(include\_probabilities=True, return\_object='array', include\_title=True)

*[{'TreatmentType': {'See & Treat': '79.78', 'See & Convey': '20.04', 'Hear & Treat': '0.18'}}]*

prediction.api\_output(include\_probabilities=True, return\_object='array', include\_title=False)

*[{'See & Treat': '52.65', 'See & Convey': '47.25', 'Hear & Treat': '0.09'}]*

prediction.api\_output(include\_probabilities=False, return\_object='array', include\_title=True)

*[{'TreatmentType': 'See & Convey'}]*

prediction.api\_output(include\_probabilities=False, return\_object='array', include\_title=False)

*['See & Treat']*

**Regression model examples:**

prediction.api\_output(include\_title=True, return\_object='array')

*[{'median\_house\_value': 234210.51073685993}]*

prediction.api\_output(include\_title=False, return\_object='array')

*[147561.28817500698]pardon.pardon\_predict.Prediction.as\_series*

***pardon.pardon\_predict.Prediction.as\_series()***

Return the predictions as a [pandas Series](https://pandas.pydata.org/docs/reference/api/pandas.Series.html).

*Example*

prediction = myml.predict(data=’predict\_data.csv’)

predictions\_series = prediction.as\_series()

print(predictions\_series)

#### pardon.Prediction.as\_dataframe

***pardon.Prediction.as\_dataframe()***

Return the predictions as a [pandas DataFrame](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html).

*Example*

prediction = myml.predict(data=’predict\_data.csv’)

predictions\_df = prediction.as\_dataframe()

print(predictions\_df)

## pardon.Audit

***class pardon.Audit(prediction=None, fullpath=None)***

Class for the auditing of predictions.

**Parameters:**

prediction :[*pardon.Pardon.Prediction*](#_pardon.MLData._Prediction_Attribute)*, default None*

The [pardon.Pardon.Prediction](#_pardon.MLData._Prediction_Attribute) class object.

fullpath :*str, default None*

The location of the [sqlite database file](https://www.sqlite.org/index.html).

### pardon.Audit Attributes

#### prediction

The [pardon.Pardon.Prediction](#_pardon.MLData._Prediction_Attribute) class object.

#### fullpath

String containing the location of the [sqlite database](https://www.sqlite.org/index.html) file.

#### PREDICTIONS\_TABLE\_NAME

String containing the table name of the predictions table. This is set as ‘Predictions’.

### pardon.Audit Methods

#### pardon.Audit.select\_all\_predictions

***pardon.Audit.select\_all\_predictions(include\_headers=False)***

Return an object containing all the predictions in your Predictions audit table.

**Parameters:**

include\_headers :*bool, default False*

Include column headers as the first item of the list of predictions returned.

*Example*

audit\_db = pardon.Audit(fullpath=‘treatment\_predictions.db’)

all\_predictions = audit\_db.select\_all\_predictions(include\_headers=True)

print(all\_predictions)

## pardon.FailOn

***class pardon.FailOn(column, operator, value, calculation=None, subset=None)***

Class allowing a set of rules to be applied to the input data. If any of these rules match, the input data will invoke a failure. Essentially, add a rule that will cause a failure.

**Parameters:**

column :*str*

The column name you are applying the Fail On rule to.

operator :*str {‘gt’, ‘gte’, ‘e’, ‘ne’,‘lt’, ‘lte’, ‘i’, ‘ni’, ‘b’, ‘nb’}*

The rule logic to be applied. The descriptions are as follows:

* *gt: Greater than*
* *gte: Greater than or equal to*
* *e: Equal to*
* *ne: Not equal to*
* *lt: Less than*
* *lte: Less than or equal to*
* *i: In*
* *ni: Not in*
* *b: Between*
* *nb: Not between*

value :*str, int, float*

The value to compare the logic and calculation to. For example, if you used the operator ‘lt’, and value=5, if the column contains a value less than 5, a failure will be invoked.

calculation :*func, str, None, default None*

The function to apply to the column before making the comparison. The following in-built functions can be passed as strings: ‘mean’, ‘median’, ‘mode’, ‘z\_score’, ‘max’, ‘min’, ‘sum’, ’count’.

Note, if only 1 row is passed for checking, the raw value from the row will be used instead of the calculation, except for ‘z\_score’, where 0 will be used.

subset :*dict, None, default None*

Add any subset of the data you wish to apply the check to. For example, if you only wanted to apply this to customers with a last name of ‘Smith’ in the customer\_last\_name column, you could use subset=dict(customer\_last\_name=’Smith’).

*Example*

fail\_on = pardon.FailOn(column=’CustomerAge’, calculation='mean', operator='gte', value=60, subset=(dict(Customer\_Last\_Name='Smith'))))

### pardon.FailOn Attributes

#### column

The column that the FailOn is applied to.

#### operator

The FailOn operator.

#### value

The FailOn value for comparison.

#### calculation

The FailOn calculation to apply to the column.

#### subset

The subset to apply the FailOn to.

### pardon.FailOn Methods

#### pardon.FailOn.apply\_fail\_on

***pardon.FailOn.apply\_fail\_on(data,*** ***encoding='latin-1', sep=',' )***

Returns a [FailOnOutput](#_pardon.MLData.FailOnOutput_Attribut) object containing the FailOn assessment.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data to apply the FailOn to.

encoding :*str, default ‘latin-1’ {‘latin-1’ , ‘ascii’, ‘utf-8’}*

The encoding used when opening csv files.

sep :*str, default ‘,’*

The delimiter to use when opening csv or txt files.

*Example*

fail\_on = pardon.FailOn(column=’CustomerAge’, calculation='mean', operator='gte', value=60, subset=(dict(Customer\_Last\_Name='Smith'))))

outcome = fail\_on.apply\_fail\_on(data=’new\_customers.csv’)

print(outcome.failed)

### pardon.FailOnOutput Attributes

The class object returned from the [pardon.FailOn.apply\_fail\_on](#_pardon.Audit.apply_fail_on) method.

#### column

The column of the [FailOn](#_pardon.FailOn).

#### failed

A Boolean stating if the [FailOn](#_pardon.InvalidModel) failed or not. True means the [FailOn](#_pardon.InvalidModel) failed, and False means it did not.

#### failed\_value

The value that failed the [FailOn](#_pardon.InvalidModel) failed or not.

## pardon.InvalidModel

***class pardon.InvalidModel(model, error)***

Class containing machine learning model classes that failed during training

**Parameters:**

model :*Class object*

The ML class object that failed during model training.

error :*str*

The error message encountered on fail.

### pardon.InvalidModel Attributes

#### model

The ML class object that failed during model training.

#### model\_type

The type of ML class object that failed during model training.

#### error

The error message encountered on fail.

## pardon.TestAPI

***class pardon.TestAPI(model\_fullpath, return\_object='array', data=None, include\_probabilities=True, include\_title=False)***

Launch an [API on the local host](https://flask.palletsprojects.com/en/2.0.x/) to test the output of your saved Pardon model.

**Parameters:**

model\_fullpath :*str, pardon.Pardon model*

The full path to your [pardon.Pardon](#_pardon.MLData) pkl file or your pardon.Pardon model.

return\_object :*str, default ‘array’ {‘array’, ‘dict’, ‘list’}*

The format you want to display in your [API output](https://flask.palletsprojects.com/en/2.0.x/). Array will display an array object containing a dictionary of the class name and the probability predicted for each class or just the prediction if using a regression model. Can also pass ‘dict’ or ‘list’.

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)*, default None*

The data you want to get a prediction for. If not specified, a random row will be selected from the original raw input data.

include\_probabilities :*bool, default True*

For classification problems decide if you want to include a probability score for each class. If False, the class with the highest probability will be returned.

include\_title :*bool, default False*

Include a title in the output. This will add the target column name for classification. It will also add ‘Probability of Class’, for classification models with probabilities returning as a list.

See the [Pardon.Prediction.api\_output](#_pardon.MLData.Prediction.api_output) method for more information.

*Example:*

pardon.TestAPI(model\_fullpath='C:\\Users\\model.pkl', return\_object=’list’, include\_title=True)

# Functions

## pardon.load\_model

***function pardon.load\_model(model\_fullpath)***

Function to load a previously saved [pardon.Pardon](#_pardon.MLData) class. Returns a [pardon.Pardon](#_pardon.MLData) object.

**Parameters:**

model\_fullpath :*str*

The full path to your [pardon.Pardon](#_pardon.MLData) pkl file.

*Example:*

myml = pardon.load\_model(model\_fullpath='C:\\Users\\model.pkl’)

## pardon.data\_reader

***function pardon.data\_reader(data, encoding=’latin-1’, sep=’,’, error\_bad\_lines=None)***

Function to load a file or non-Pandas DataFrame object into a Pandas DataFrame.

**Parameters:**

data :*csv, txt, xls, xlsx, xlsm, json, parquet, xml, dict, list,* [*numpy.ndarray*](https://numpy.org/doc/stable/reference/generated/numpy.ndarray.html)*,* [*pandas.DataFrame*](https://pandas.pydata.org/docs/reference/api/pandas.DataFrame.html)

The data source. Can be a file path or an object.

encoding :*str, default ‘latin-1’ {‘latin-1’ , ‘ascii’, ‘utf-8’}*

The encoding used when opening csv or xml files.

sep :*str, default ‘,’*

The delimiter to use when opening csv or txt files.

error\_bad\_lines :*bool, default None*

Raise an error if bad lines with too many or too few delimiters are found. If False, bad lines will be dropped.

*Example:*

df = pardon.data\_reader(data=’new\_data.csv’)

## pardon.remove\_non\_ascii

***function pardon.remove\_non\_ascii(string)***

Function to remove non-ascii characters from a string.

**Parameters:**

string :*str*

The string to remove non-ascii characters from.

*Example:*

string = 'this string has bad chars in m©ª«zy¤¥uwÆÇval672'

fixed\_string = pardon.remove\_non\_ascii(string)

## pardon.extract\_postcode

***function pardon.extract\_postcode(string, type=’uk’, return\_empty=False)***

Function to extract a UK postcode or US zip code from a string. If found, a formatted postcode will be returned.

**Parameters:**

string :*str*

The string to extract the postcode from.

type :*str, default ‘uk’ {‘uk’, ‘us’}*

The type of postcode to extract. This can be “uk” for a UK postcode, or “us” for a US zip code.

return\_empty :*bool, default False*

If True, when a match is not found, an empty string will be returned. If False, the original string will be returned.

*Example:*

formatted\_postcode = pardon.extract\_postcode(‘get the postcode from, this string, sw6 6hf’, type=’uk’)

## pardon.remove\_characters

***function pardon.remove\_characters(string, remove\_chars, is\_regex=False)***

Function to remove specified characters from a string, by passing the characters to remove, or a regular expression.

**Parameters:**

string :*str*

The string to remove characters from.

remove\_chars :*str, list*

The character or list of characters, or regex or list of regexes to remove or apply to the string.

is\_regex :*bool, default False*

If you supply regex to the remove\_chars argument, set this flag to true to have the regex applied to the string. If False, a simple find and replace will occur.

*Example:*

test\_string = 'hello\ncan you \tfix this string???'

fixed\_string = pardon.remove\_characters(string=test\_string, remove\_chars=['\n',’\t’, '?')

## pardon.get\_sentiment

***function pardon.get\_sentiment(text, return\_as=’text’)***

Apply sentiment analysis to the text specified using the nltk library.

**Parameters:**

text :*str*

The text to apply the sentiment analysis to.

return\_as :*str, dict, default 'text'*

Define the output of your sentiment analysis. If 'text', each string will have a sentiment analysis determined to be 'positive', 'neutral', or 'negative'. You can supply a dictionary object outlining what value to return for each sentiment type. For example, return\_as=dict('positive'=3, 'neutral'=2, 'negative'=1).

*Example:*

test\_string = ‘Can you tell I am angry with this?!'

sent\_response = {‘positive’: ‘Positive Text’, ‘neutral’: ‘Neutral Text’, ‘negative’: ‘Negative Text’}

sentiment = pardon.get\_sentiment(text=test\_string, return\_as=sent\_response)

## pardon.flatten\_list

***function pardon.flatten\_list(list\_item, ignore\_items= None, deduplicate=False)***

*[added v1.0.8]*

Function to turn a list of lists into a single list.

**Parameters:**

list\_item :*list*

The list containing further lists.

ignore\_items :*list, default None [added v1.1.0]*

A single item or list of items to ignore when adding to your new list.

deduplicate :*bool, default Falses*

Return a deduplicated list.

*Example:*

list\_of\_lists = [[1,2,3],[‘age’,’price’,0,3],[2]]

flat\_list = pardon.flatten\_list(list\_item=list\_of\_lists)

## pardon.create\_intervals

***function pardon.create\_intervals(start, end, interval)***

*[added v1.0.4]*

Function to create a list of numbers for a specified interval.

**Parameters:**

start :*int, float*

The first number to start your interval from.

end :*int, float*

The number to end your interval on.

interval :*int, float*

The size of the interval between each item in the list.

*Example:*

intervals = pardon.create\_intervals(start=1, end=100, interval=5)

## pardon.to\_json

***function pardon.to\_json(object\_to\_json)***

*[added v1.0.5]*

Convert the object to json format.

**Parameters:**

object\_to\_json :*Any*

The object to convert to json. pandas.DataFrame and pandas.Series will be converted using internal methods.

*Example:*

data = myml.data

data\_in\_json = pardon.to\_json(object\_to\_json=data)

## pardon.get\_data\_averages

***function pardon.get\_data\_averages(data)***

*[added v1.0.8]*

Returns a dictionary containing the mean, median, mode, and standard deviation from the data specified.

**Parameters:**

data :*pandas.Series*

The data to get the averages from.

*Example:*

averages = pardon.get\_data\_averages(data=df[‘price’])

## pardon.save\_pickle

***function pardon.save\_pickle(object, output\_fullpath)***

*[added v1.1.1]*

Saves a variable to a pickle object.

**Parameters:**

object :*Any*

The object to be saved.

output\_fullpath :*str*

The output fullpath to where you want your object to be saved. The output fullpath should end with a reference to the filename as a pkl file.

*Example:*

item\_to\_save = dict(testing=True)

pardon.save\_pickle(object= item, output\_fullpath=’item\_saved.pkl')

## pardon.load\_pickle

***function pardon.load\_pickle(object\_fullpath)***

*[added v1.1.1]*

Function to load a pkl file.

**Parameters:**

object\_fullpath :*str*

The fullpath to your pkl file.

*Example:*

loaded\_pkl = pardon.load\_pickle(object\_fullpath=’item\_saved.pkl')

## pardon.assert\_item\_type

***function pardon.*** ***assert\_item\_type(item, item\_types)***

*[added v1.0.5]*

Checks if an item is a particular type and raises a helpful error specifying the valid type if not.

**Parameters:**

item :*Any*

The variable to check for the valid item types.

item\_types :*list*

A list containing the valid item types.

*Example:*

variable\_int = 12

pardon.assert\_item\_type(item=variable\_int, item\_types=[int, floats])

# Options

## pardon.pardon\_options

Several options can be adjusted to suit your needs in the pardon.Pardon class.

#### MIN\_ROW\_REQ

The minimum number of rows required to train a model. Note, this attribute can be changed prior to instantiation if required. Use: *pardon.pardon\_options.MIN\_ROW\_REQ = 10*.

#### VALID\_FILE\_FORMATS

A tuple containing the valid file formats accepted by the [Pardon object data parameter](#_data). Only available after class instantiation. These values should not be changed.

#### DATETIME\_FORMAT

The format to display datetimes. A list of valid date formats can be found [here](https://strftime.org/). Change this by using *pardon.pardon\_options.DATETIME\_FORMAT = '%d %B %Y %H:%M:%S'*

#### AVERAGE\_FAIL\_DEFAULT\_VALUE

The value to use if the function fails when performing an average calculation. An average calculation may fail if the column contains only nulls or other invalid values. To change this, you can use *pardon.pardon\_options.AVERAGE\_FAIL\_DEFAULT\_VALUE = 99*

#### MIN\_SAMPLE\_FOR\_UNDERSAMPLE

The minimum number of rows to be present before under-sampling will be performed when balancing classes. If the number of instances of a class is fewer than the MIN\_SAMPLE\_FOR\_UNDERSAMPLE, over-sampling will be performed, else under-sampling will be performed. This is to prevent rows being lost in smaller datasets. The default is 20,000 rows in the class with the fewest instances. To change this, you can use *pardon.pardon\_options.MIN\_SAMPLE\_FOR\_UNDERSAMPLE = 5000*

#### UNDER\_SAMPLING\_MODEL

The model object to use when performing under-sampling. The default is [random under-sampling](https://imbalanced-learn.org/stable/references/generated/imblearn.under_sampling.RandomUnderSampler.html) To change this, import the necessary libraries and you can use something like *pardon.pardon\_options.UNDER\_SAMPLING\_MODEL = imblearn.under\_sampling.TomekLinks()*

#### OVER\_SAMPLING\_MODEL

The model object to use when performing under-sampling. The default is [random over-sampling](https://imbalanced-learn.org/stable/references/generated/imblearn.over_sampling.RandomOverSampler.html) To change this, import the necessary libraries and you can use something like *pardon.pardon\_options.OVER\_SAMPLING\_MODEL = imblearn.over\_sampling.SMOTE()*

#### XGBCLASSIFIER\_DEFAULT

The default evaluation metric to use for XGBClassifiers. This is set to ‘mlogloss’. You can change this using *pardon.pardon\_options.XGBCLASSIFIER\_DEFAULT = ‘auc’*

#### XGBCLASSIFIER\_BINARY\_DEFAULT

The default evaluation metric to use for XGBClassifiers predicting binary classes. This is set to ‘logloss’. You can change this using *pardon.pardon\_options.XGBCLASSIFIER\_BINARY\_DEFAULT = ‘auc’*

#### SKLEARN\_CLASSIFIER\_DEFAULT

The default evaluation metric to use for SKLearn classification models. This is set to ‘accuracy’. You can change this using *pardon.pardon\_options.SKLEARN\_CLASSIFIER\_DEFAULT = ‘recall\_macro’*

#### SKLEARN\_REGRESSION\_DEFAULT

The default evaluation metric to use for SKLearn classification models. This is set to ‘r2’. You can change this using *pardon.pardon\_options.SKLEARN\_REGRESSION\_DEFAULT = ‘explained\_variance’*

#### REDUCTION\_SCORE\_ON

The metrics that attempt to be minimised when determining best model performance. For these metrics, a lower score indicates a better performing model as opposed to a higher score. This should not be changed.

#### BALANCE\_SCALE\_RATIO

[added v1.0.3]

This is set to 0.8. This ratio determines when classes will be balanced using over or under sampling. The formula for determining if classes should be balanced is calculated as follows:

max\_ratio = (max\_class\_instances / total\_class\_instances) x BALANCE\_SCALE\_RATIO

ratio = number\_of\_class\_instances / total\_class\_instances

if ratio < rax\_ratio for any of the classes, then class balancing will be performed.

Take the following example:

Class\_a = 200 items

Class\_b = 100 items

Class\_c = 50 items

max\_ratio = (200 / 350) x 0.8 = 0.46

ratio (Class\_c) = 50 / 350 = 0.14

**0.14 < 0.46 and so class balancing will be performed.**

Setting the BALANCE\_SCALE\_RATIO = 1 will force class balancing to occur unless every single class has the same number of items. Setting the BALANCE\_SCALE\_RATIO = 0 would mean class balancing would never occur. Therefore, you can tweak this ratio to control the sensitivity when determining class balancing.

You can change this using *pardon.pardon\_options.BALANCE\_SCALE\_RATIO = 0.95*

#### IGNORE\_SINGLE\_CLASS\_ERROR

Set this to True if you want to ignore the error raised when encountering single instances of classes. These rows will need to be removed before model training. Switch this check off by using *pardon.pardon\_options.IGNORE\_SINGLE\_CLASS\_ERROR = True* pardon as an application [beta]

You can use the pardon [rapid\_m](#_pardon.Pardon.rapid_ml)l method and [data visualisations](#_pardon.MLData.plot_data) through a streamlit app for a visual interface.

## pardon.app

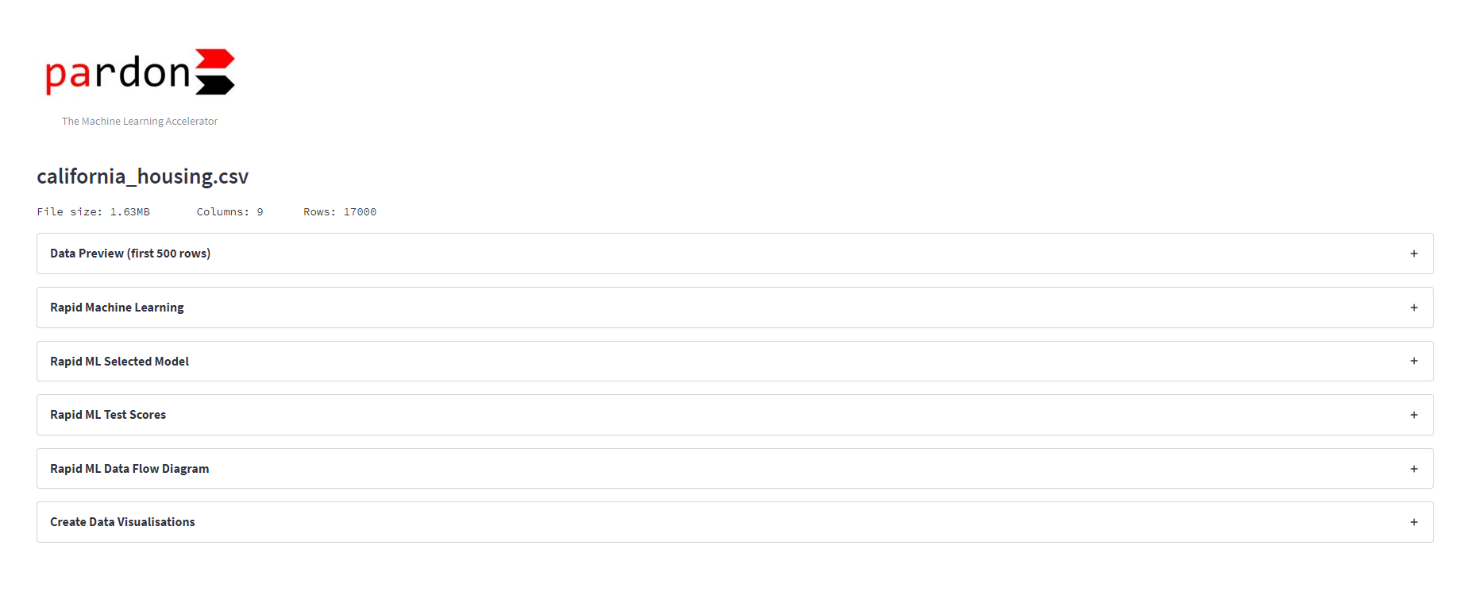
***function pardon.app()***

*[added v1.0.5]*

Launches a streamlit graphical user interface for interacting with pardon visualisations and rapid machine learning methods.

*Example:*

pardon.app()



# Release History

|  |  |  |
| --- | --- | --- |
| Version | Release Date | Notes |
| v1.0.0 | 9 May 2022 | Original version. |
| v1.0.1 | 10 May 2022 | 1. Fixed bug in model diagram that would error if the model was saved but not trained. 2. Removed messages checking for model performance when the model is saved but not trained. 3. Fixed bug that checked if the model was ready for output data even though there was no target. 4. Fixed bug that did not correctly read a dataframe when passing data into the [output\_data](#_pardon.Pardon.output_data) method. 5. Added apply\_fail\_ons argument to [output\_data](#_pardon.Pardon.output_data) method that allows users to apply any [failons](#_pardon.InvalidModel) before saving or outputting data. |
| v1.0.2 | 11 May 2022 | 1. Added ability to load xml files. 2. Flask downgraded to v1.1.4 due to bug causing errors on Azure ML deployment for versions greater than 2. 3. Removed all dependencies not directly required for the project. |
| v1.0.3 | 25 May 2022 | 1. Renamed ignore\_outliers argument to remove\_outliers in the [plot\_data](#_pardon.Pardon.plot_data) method. This was to ensure consistency of naming across methods. 2. Updated eval\_metrics to remove the neg\_ prefix to sklearn metrics. This is because the scoring metric and evaluation metric differ. Check the [sklearn documentation](https://scikit-learn.org/stable/modules/model_evaluation.html?) for more information. 3. Updated [one\_hot\_encode](#_pardon.Pardon.one_hot_encode) method to allow numeric columns to be one hot encoded and added the remove\_encoding argument to give users the option to retain the columns that were encoded. 4. Added columns argument to the [scale\_data](#_pardon.Pardon.scale_data) method, as well as max\_abs\_scaler to the available scaler types. 5. Added the [Pardon.BALANCE\_SCALE\_RATIO](#_BALANCE_SCALE_RATIO) for more control when determining the balance classes. 6. Updated class distribution method to ensure correct class names are returned even after they have been encoded. 7. Made the [select\_best\_features](#_pardon.Pardon.select_best_features) function available to users. 8. Added the available\_chart\_types method to show which chart types are available in plot data. 9. Updated the name of the argument timeseries\_estimator and chart type from timeseries to lineplot\_estimator and lineplot respectively. 10. Added the [model\_learning\_curve](#_pardon.Pardon.model_learning_curve) method. 11. Added the size attribute to the [plot\_data](#_pardon.MLData.plot_data) method. |
| v1.0.4 | 8 June 2022 | 1. Added [create\_intervals](#_pardon.create_intervals) and [create\_interval\_labels](#_pardon.create_interval_labels) functions. 2. Updated all doc strings with documentation. |
| v1.0.5 | 26 July 2022 | 1. Added [to\_json](#_pardon.to_json) function. 2. Added as\_json parameter to [get\_sample\_rows](#_pardon.Pardon.get_sample_rows) method. 3. Added DBSCAN clustering model. 4. Added z\_score as a valid lineplot\_estimator in the [plot\_data](#_pardon.MLData.plot_data) method. 5. Updated [model\_script](#_pardon.MLData.model_script) method to include the test scores of the models that were tested but not selected. 6. Added the [get\_default\_model\_params](#_pardon.Pardon.get_default_model_par) method. 7. Added the [set\_default\_model\_params](#_pardon.Pardon.set_default_model_par) method. 8. Added the [assert\_item\_type](#_pardon.assert_item_type) function. 9. Updated available scoring and evaluation metrics. 10. Updated model script to include additional details of model and data. 11. Added [add\_data](#_pardon.Pardon.add_data) method so users can add more data to an existing model. 12. Added use\_shap\_values argument to the [model\_script](#_pardon.Pardon.model_script) method to help increase speed of script building when using an XGBoost model. 13. Added the [pardon app launcher](#_pardon.app_build.app). |
| v1.0.6 | 28 July 2022 | 1. Updated XGBClassifier, to only use cross validation when there are 2 or more classes – this is to prevent an error. 2. Updated streamlit to use tabs in front end rather than dropdown menu. 3. Fixed bug in [find\_outliers](#_pardon.Pardon.find_outliers) functions. 4. Removed data argument from [find\_outliers](#_pardon.Pardon.find_outliers) function. |
| v1.0.7 | 29 July 2022 | 1. Fixed bug in the [create\_bins](#_pardon.Pardon.create_bins) function. |
| v1.0.8 | 02 August 2022 | 1. Added [flatten\_list](#_pardon.flatten_list) function. 2. Renamed convert\_to\_date function to [convert\_to\_datetime](#_pardon.Pardon.convert_to_date) to better reflect the type. 3. Renamed the full\_empty arguments to be fill\_non\_numeric for convert\_to\_numeric function and to fill\_non\_datetime for convert\_to\_datetime function. 4. Added ability to pass a function to the [convert\_to\_numeric](#_pardon.Pardon.convert_to_numeric) function. 5. Removed ability to drop non-datetime rows in convert\_to\_datetime function as this could causes errors during predictions. 6. Added the [convert\_to\_string](#_pardon.Pardon.convert_to_string) function. 7. Added the apply\_to\_test argument to the [find\_outliers](#_pardon.Pardon.find_outliers) and [remove\_outliers](#_pardon.Pardon.remove_outliers) functions. |
| v1.0.9 | 03 August 2022 | 1. Updated [label\_encode](#_pardon.Pardon.label_encode) function to ensure all object columns will have all items converted to strings prior to label encoding. 2. Updated [rapid\_ml](#_pardon.MLData.rapid_ml) function so that the max\_n\_iterators argument is used in model training through the model\_params argument. 3. Added the reconcile argument to the [add\_func](#_pardon.Pardon.add_func) function. |
| v1.1.0 | 03 August 2022 | 1. Updated the [flatten\_list](#_pardon.flatten_list) function to include the ignore\_items argument. 2. Updated [replace\_values](#_pardon.Pardon.replace_values) to allow for lambdas. |
| v1.1.1 | 04 August 2022 | 1. Updated [replace\_values](#_pardon.Pardon.replace_values) to allow for lambdas for both replace and replace\_with. 2. Updated the [scale\_data](#_pardon.MLData.scale_data) function to only validate the columns that are being scaled. 3. Updated the [find\_best\_model\_parameters](#_pardon.Pardon.find_best_model_param) function to remove bug when using cross\_validation on xgbclassifier with binary classes. 4. Renamed select\_best\_features to [get\_best\_features](#_pardon.Pardon.get_best_features) and added error check to ensure max\_features could not exceed max\_columns available. 5. Added the [IGNORE\_SINGLE\_CLASS\_ERROR](#_IGNORE_SINGLE_CLASS_ERROR) option to allow users to ignore the error raised if their data contains single instances of a class in the target columns. 6. Updated [flatten\_list](#_pardon.flatten_list) function to work with all array types. 7. Added apply\_to\_items argument to [flatten\_list](#_pardon.flatten_list) function. 8. Added the [load\_pickle](#_pardon.load_pickle) and [save\_pickle](#_pardon.save_pickle) functions. |
| v1.1.2 | 04 August 2022 | 1. Updated [flatten\_list](#_pardon.flatten_list) function so that it removes a big with recursion. |
| v1.1.3 | 05 August 2022 | 1. Updated the [one\_hot\_encode](#_pardon.Pardon.one_hot_encode) function so that it only applies the max\_items\_in\_category check when no columns have been provided. 2. Added the [ignore\_func\_in\_transformations](#_pardon.Pardon.ignore_func_in_transf) function. 3. Added equality check for Pardon classes. 4. Added validator to prevent users from overwriting the target column after the model has been created. 5. Added the [null\_count](#_pardon.Pardon.null_count) functions. 6. Added the [correlations](#_pardon.Pardon.correlations) function. 7. Added the [histogram](#_pardon.Pardon.histogram) function. 8. Added ability to add datasets from another model using + operator. 9. Updated [apply\_func](#_pardon.Pardon.apply_func) function to allow for row based functions. |
| v1.1.4 | 5 August 2022 | 1. Added the [config](#_config) attribute and yaml argument to load any yaml files with your [Pardon](#_pardon.Pardon) class. |
| v1.1.5 | 8 August 2022 | 1. Updated data\_reader so that it takes a copy of a dataframe rather than returns the dataframe so that it does not change the original. 2. Removed bug in validation that prevented the removal of columns of data in the transform method. |
| v1.1.6 | 8 August 2022 | 1. Updated the fit transform functions to work with pardon [transform](#_pardon.Pardon.transform) function. |
| V1.1.7 | 5 September 2022 | 1. Fixed bug in [remove\_outliers](#_pardon.MLData.remove_outliers) method. 2. Changed order in [rapid\_ml](#_pardon.MLData.rapid_ml) method so that [fill\_nulls](#_pardon.Pardon.fill_nulls) runs before [remove\_outliers](#_pardon.MLData.remove_outliers) to prevent errors. 3. Renamed text\_columns function to [string\_columns](#_pardon.Pardon.string_columns). 4. Removed potential bugs in transform functions. 5. Fixed bug in [app\_build](#_pardon.app_build.app) Model Tester. 6. Updated the [save\_model](#_pardon.Pardon.save_model) function to work with non-trained models. 7. Updated the columns argument on the Pardon class so that the columns removed are retained in the [remove\_columns](#_pardon.Pardon.remove_columns) function so if they exist in future datasets, they will be removed. 8. Updated [GUI](#_pardon.app_build.app) so users can export their transformed data. 9. Added [ordinal\_encode](#_pardon.Pardon.ordinal_encode) function. 10. Updated the [rapid\_ml](#_pardon.MLData.rapid_ml) function to include options to ordinal encode data. 11. Removed stratify argument from the [rapid\_ml](#_pardon.MLData.rapid_ml) function. 12. Added the [test\_class\_distribution](#_test_class_distribution) attribute. 13. Updated the [TestAPI](#_pardon.TestAPI) class so a [pardon.Pardon](#_pardon.MLData) model can be passed directly. 14. Added the shuffle argument to the [Pardon](#_pardon.MLData) class instantiation. 15. Updated the [ignore\_function\_in\_predictions](#_pardon.Pardon.ignore_function_in_pr) function so users can specify exactly which function to ignore rather than just functions with the same name. 16. Updated the [add\_data](#_pardon.Pardon.add_data) function so that new data can be added to models that had their data cleared during saving. 17. Updated scoring metrics to include roc\_auc. 18. Added feature\_contribution to the [model\_script](#_pardon.MLData.model_script) output. 19. Fixed bug that meant non-data functions were running in certain situations. 20. Added include trendline check box and updated bugs in Data Visualisation section of [app\_build](#_pardon.app_build.app). 21. Added log\_loss score and eval metric. 22. Added the [transformations](#_pardon.Pardon.trasnformations) function showing what transformations are applied to the data. |
| V1.1.8 | 13 September 2022 | 1. Fixed bug in [model\_script](#_pardon.MLData.model_script) that would error if the data had been cleared on saving. |
| V1.1.9 | 13 September 2022 | 1. Fixed bug in [model\_script](#_pardon.MLData.model_script) that caused an error if not using a trained model. |
| V1.2.0 | 14 September 2022 | 1. Added pardon version to [model\_script](#_pardon.MLData.model_script) output as well as the \_\_version\_\_ attribute to the package. |
| V1.2.1 | 14 September 2022 | 1. Fixed bug in version. |
| V1.2.2 | 14 September 2022 | 1. Added additional column validation check when creating Pardon class. |
| V1.2.3 | 14 September 2022 | 1. Fixed bug in [remove\_columns](#_pardon.Pardon.remove_columns) argument that would remove all columns if no list was provided. |
| V1.2.4 | 15 September 2022 | 1. Updated [remove\_unhelpful\_columns](#_pardon.Pardon.remove_unhelpful_colu) to only run if there are 2 or more rows. 2. Updated the [fill\_nulls](#_pardon.MLData.fill_nulls) method so that if single rows are null, the average will automatically be changed to the default average value as no average can be found. |
| V1.2.5 | 16 September 2022 | 1. Added the include\_ignored\_transformations argument to the [transform](#_pardon.Pardon.transform) method to allow users to include or ignore the different transformations. |
| V1.2.6 | 16 September 2022 | 1. Added verbose argument to the [transform](#_pardon.Pardon.transform) method. |
| V1.2.7 | 20 September 2022 | 1. Updated [flatten\_list](#_pardon.flatten_list) function to create lists into an array with dtype objects to avoid deprecation warning from numpy. Also set deduplicate to be False by default. 2. Dependency updates. |
| V1.2.8 | 20 September 2022 | 1. Updated bug when getting train data columns. |
| V1.2.9 | 25 October 2022 | 1. Added func\_name argument to [apply\_func](#_pardon.Pardon.apply_func) and [add\_func](#_pardon.Pardon.add_func). 2. Updated [app\_build.app](#_pardon.app_build.app) to include more data transformations. 3. Fixed bug in [model\_learning\_curve](#_pardon.Pardon.model_learning_curve). 4. Updated assert\_item\_type function to correctly obtain invalid item type name. 5. Added [transformation\_count](#_pardon.Pardon.transformation_count) method. 6. Added [sort\_by](#_pardon.Pardon.sort_by) method. |
| V2.0.3 | 31 October 2022 | 1. Pypi release. |
| V2.0.4 | 09 December 2022 | 1. Pypi release. |
| V2.0.5 | 09 December 2022 | 1. Pypi release. |