Crimes Dummy Project

Release v.0.1

Jose Angel Velasco

Apr 20, 2021

Chapter 1

Indices and tables

- genindex
- modindex
- search

Chapter 2

Data Analysis

`dummy_project_data_analysis.main()`

Main program of Crimes Exploratory Data Analysis

This function execute a descriptive analysis of the crimes data set and provides insight about and it can support de decision process related to surveillance schedule

Chapter 3

Predictive Modelling

`dummy_project_predictive_modeling.`**`main`**( )

Main program of Crimes Predictive Modelling

This function execute a predictive modelling process in which the crimes data set is grouped in time-space basis, and a flag indicating violent crime or not is added. Then a machine learning model is trained to classify the crimes registers into violent /non-violent crimes

Chapter 4

Inference

`dummy_project_inference.`**`main`**`( )`

Main program of Crimes Inference

This function execute a inference procedure to decide whether a crime is violent or not based on the machine learning model trained for classification

Chapter 5

Auxiliar Functions

`dummy_project_utils.`**`calculate_classification_metrics`** (*y_hat*, *y_test*, *logger*)

Calculate Classification Metrics

Calculate deveral classifcation metrics: confusion matrix, precision, recall and AUC ROC

param y_hat actual labels obteined

param y_test target labels

param logger logger to record errors

`dummy_project_utils.`**`check_nan`** (*dataframe*, *logger*)

Check NaNs
Check if the input dataframe contains NaN values and fill with 0

Parameters
- **`dataframe`** – Data Frame if input data
- **`logger`** – logger to record exception

Returns output dataframe with 0 values in NaN values

`dummy_project_utils.`**`enconde_input_features`** (*x*, *logger*)

Encode Input Features Encode the input features using label encoder

Parameters
- **`x`** – INput features dataframe
- **`logger`** – logger:

Return x Data frame with encoded input features (_ENCODED)

`dummy_project_utils.`**`get_frequencies`** (*df*, *column*, *logger*)

Get frequencies
Obtain some distribution of frequency to known how is distributed the incidences

Parameters
- **`df`** – Data frame
- **`column`** – Column to obtain the distribution of frequency of occurrence
- **`logger`** – logger to record exception

Returns dataframe with the crimes frequencies

`dummy_project_utils.`**`load_data`** (*path*, *logger*)

Load input data
Load the crimes input data

Parameters
- **`path`** (*`str`*) – path of the csv file to upload
- **`logger`** – logger to record exception

Returns dataframe with the data

`dummy_project_utils.`**`plot_barplot`** (*df*, *var_x*, *var_y*, *path*, *logger*)

Plot Barplot

Plot a barplot to compare the incidence of crimes according to characteristics

Parameters
- **`df`** – Dataframe to plot
- **`var_x`** – x-axis variable
- **`var_y`** – y-axis variable
- **`path`** – path where it is saved plot in png format
- **`logger`** – logger to record exception

`dummy_project_utils.`**`plot_scatterplot`** (*df*, *var_x*, *var_y*, *scale*, *path*, *logger*)

Plot scatter

plot with the coordinates of the crimes

Parameters
- **`df`** – Dataframe to plot
- **`var_x`** – x-axis variable
- **`var_y`** – y-axis variable
- **`scale`** – scale of the point
- **`path`** – path where it is saved plot in png format

`dummy_project_utils.`**`save_model`** (*model*, *model_name*, *logger*, *models_path*)

** Save Model**

Save the predictive mdel in .sav file in models path indicated in config.ini file

Parameters
- **`model`** – Model
- **`model_name`** – model name (str)
- **`logger`** – logger
- **`models_path`** – path to the models folder (str)

`dummy_project_utils.`**`set_up_logger`** (*path*)

Set up logger

Configure the logger to record all the envents in the execution of the code

Parameters
- **`path`** (*`str`*) – path where to store logs example: 'logslog_file_name'
- **`logger`** – logger to record exception

Returns logger logger

Chapter 6

Tests

class `dummy_project_test.`**`TestFunction`** (*methodName='runTest'*)

Test Case

This clase defines several unitary tests to be produced

`test_load_data` ( )

** Test Load Data**

Test that the load data is a data frame

Parameters **`self`** –

`test_load_data_empty` ( )

** Test no-empty data frame**

Test that the load data is a data frame and is not empty

Parameters **`self`** –

Python Module Index

Index