Function List Documentation

Example summary area.

# Purpose

This document outlines a set of custom Python functions designed to facilitate various aspects of data preprocessing, modelling, and analysis. Each function is carefully developed to solve specific problems, improve workflow efficiency, and provide reusable code components that can be integrated into different projects. The document serves as a comprehensive guide to understanding, implementing, and utilizing these functions in practical applications.

### Sample format for li

***List item title.*** List contents.

### Table of Contents

* [API\_Unlimited](#api_unlimited)
* [FindMissingVal](#findmissingval)
* [HandleNullValues](#handlenullvalues)
* [Combine\_Dataset](#combine_dataset)
* [Scale\_data](#scale_data)
* [PCA\_Reduction](#pca_reduction)
* [Encode\_data](#encode_data)
* [Feature\_Engineering](#feature_engineering)
* [ExtractCoords](#extractcoords)
* [Min\_Distance](#min_distance)
* [Plot\_Correlation\_Heatmaps](#plot_correlation_heatmaps)
* [Optimal\_K\_Clusters](#optimal_k_clusters)
* [Find\_Optimal\_Clusters](#find_optimal_clusters)
* [Dimensionality\_Reduction](#dimensionality_reduction)
* [Plot\_Estimator](#plot_estimator)
* [Linear\_Regression](#linear_regression)
* [Polynomial\_Regression](#polynomial_regression)
* [LogisticRegression Class](#logisticregression_class)
* [LinearSVM Class](#linearsvm_class)

## Function Overview

Below is a list of the functions described in this document, along with a brief explanation of their primary purpose:

|  |  |
| --- | --- |
| **Function Name** | **Description** |
| API\_Unlimited | Fetches unlimited records from a specified API. |
| FindMissingVal | Identifies missing values in a dataset. |
| HandleNullValues | Handles missing data by removing or filling them. |
| Combine\_Dataset | Combines multiple datasets using a specified join mode. |
| Scale\_data | Scales data using various scaling methods like 'minmax', 'zscore'. |
| PCA\_Reduction | Performs Principal Component Analysis (PCA). |
| Encode\_data | Encodes categorical columns using various encoding techniques. |
| Feature\_Engineering | Applies transformations and generates new features in a dataset. |
| ExtractCoords | Extracts coordinates as tuples (latitude, longitude) from a DataFrame. |
| Min\_Distance | Calculates the minimum geodesic distance between points. |
| Plot\_Correlation\_Heatmaps | Plots Pearson and Spearman correlation heatmaps for a dataset. |
| Optimal\_K\_Clusters | Determines the optimal number of clusters based on silhouette scores. |
| Find\_Optimal\_Clusters | Finds the optimal number of clusters using the elbow method. |
| Dimensionality\_Reduction | Reduces data dimensionality using PCA or t-SNE. |
| Plot\_Estimator | Plots decision boundaries of an estimator along with data points. |
| Linear\_Regression | Fits a linear regression model to the data. |
| Polynomial\_Regression | Fits a polynomial regression model and plots the fitted curve. |
| LogisticRegression Class | Implements a logistic regression model using gradient descent. |
| LinearSVM Class | Implements a linear Support Vector Machine using gradient descent. |

**API\_Unlimited**

**Description:**

The API\_Unlimited function is designed to retrieve unlimited records from the City of Melbourne open data API. It fetches the dataset in CSV format and loads it into a pandas DataFrame. The function allows users to pass in the dataset name and API key (optional for now) to download the dataset.

**Parameters:**

* datasetname (string): The unique dataset identifier from the City of Melbourne open data portal.
* apikey (string, optional): The API key for authentication if required (commented out in current implementation).

A screen shot of a computer program

Description automatically generated

**Returns:**

* pandas.DataFrame: The function returns a DataFrame containing the dataset. If the API call fails, it prints an error message with the corresponding status code.

**Edge Cases:**

* If the API call fails, the function prints the error status code and returns None.
* The function assumes that the CSV is delimited by semicolons (;), so it may need adjustment if the dataset uses a different delimiter.
* A screen shot of a computer code

  Description automatically generated

**Notes:**

* You need the correct dataset identifier to download data from the API.
* This implementation fetches all records from the dataset using limit = -1. Adjust this value if you want to limit the number of records.

2.

**FindMissingVal**

**Description:**

The FindMissingVal function identifies and returns the number of missing (NaN) values for each feature in a pandas DataFrame. It returns a list of dictionaries, where each dictionary contains the feature name and the count of missing values.

**Parameters:**

* df (pandas.DataFrame): The DataFrame for which you want to check for missing values.

**Returns:**

* list: A list of dictionaries, where each dictionary contains:
  + Feature: The column name (string).
  + Number of Missing Values: The count of missing values (int) in that column.

A computer error message

Description automatically generated

**Edge Cases:**

* The function does not differentiate between columns that have 0 missing values and columns that have NaN values; all columns will be included in the result.
* The function assumes that missing values are represented as NaN in the DataFrame.

**Notes:**

* The function uses the isnull() method to identify missing values in the DataFrame.

3.

**handle\_null\_values**

**Description:**

The handle\_null\_values function processes missing (NaN) values in a specified dataset. It allows you to either remove rows containing null values or fill them with the mean, median, or mode of the column.

**Parameters:**

* dataset (pandas.DataFrame): The dataset (DataFrame) where missing values need to be handled.
* columns (list): A list of column names where missing values will be processed.
* action (string): The action to perform on the missing values. Options are:
  + 'remove': Removes rows with missing values in the specified columns.
  + 'mean': Fills missing values in the specified columns with the mean of the column.
  + 'median': Fills missing values in the specified columns with the median of the column.
  + 'mode': Fills missing values in the specified columns with the mode of the column.

**Returns:**

* pandas.DataFrame: The modified dataset with missing values either removed or filled based on the specified action.

A screen shot of a computer code

Description automatically generated

4.

**Combine\_Dataset**

**Description:**

The Combine\_Dataset function merges multiple pandas DataFrames based on common columns using a specified join mode. This function checks for common columns between datasets and combines them using the chosen merge strategy (default is outer join).

**Parameters:**

* datasets (list of pandas.DataFrame): A list of DataFrames to be combined.
* mode (string, optional): The type of join operation to use. Options are:
  + 'inner': Returns rows that have matching values in both datasets.
  + 'outer': Returns all rows, combining where possible and filling with NaN where no match is found (default).
  + 'left': Returns all rows from the left dataset and matching rows from the right.
  + 'right': Returns all rows from the right dataset and matching rows from the left.

**Returns:**

* pandas.DataFrame: The combined DataFrame based on the common columns and the specified join mode.

A screenshot of a computer

Description automatically generated

**Edge Cases:**

* If no datasets are provided, a ValueError is raised.
* If there are no common columns between the datasets, a ValueError is raised, alerting the user to provide datasets with common columns.

5.

**Scale\_data**

**Description:**

The Scale\_data function scales the specified columns in a pandas DataFrame using various scaling methods. It creates a scaled version of the DataFrame without modifying the original one.

**Parameters:**

* dataframe (pandas.DataFrame): The input DataFrame to scale.
* columns (list): The list of columns to be scaled.
* method (string, optional): The scaling method to use. The available options are:
  + 'minmax': Scales the data to a range between 0 and 1 (default).
  + 'zscore': Standardizes the data by subtracting the mean and dividing by the standard deviation (Z-score normalization).
  + 'powertransformer': Applies a power transformation to make the data more Gaussian-like.
  + 'absscalar': Scales the data based on the absolute maximum value.
  + 'robustscalar': Scales the data by removing the median and scaling according to the interquartile range (IQR).
  + 'normalizer': Scales individual samples to unit norm.
  + 'quantile': Transforms the data to follow a uniform or normal distribution.

**Returns:**

* pandas.DataFrame: A new DataFrame with the specified columns scaled using the selected method.

A screen shot of a computer

Description automatically generated

**Edge Cases:**

* If any column specified in columns is not found in the DataFrame, a ValueError is raised with the missing column names.
* If an invalid scaling method is provided, a ValueError is raised, prompting the user to enter one of the supported methods.

**Notes:**

* The function uses sklearn’s scalers for different scaling techniques.

6.

**PCA\_Reduction**

**Description:**

The PCA\_Reduction function performs Principal Component Analysis (PCA) on a numerical dataset to reduce its dimensionality. It checks for non-numerical values and NaN values before applying PCA. The resulting components are returned in a new DataFrame, with the number of components specified by the user.

**Parameters:**

* data (pandas.DataFrame): The input DataFrame containing numerical features.
* components\_num (int): The number of principal components to retain.

**Returns:**

* pandas.DataFrame: A new DataFrame containing the principal components (PC1, PC2, …).
* None: Returned in case of an error, such as non-numerical or missing values in the data.

A screen shot of a computer code

Description automatically generated

**Notes:**

* The function only works with numerical data. It is important to handle categorical variables or missing values before passing the data to this function.
* The number of components (components\_num) should be less than or equal to the number of features in the dataset.

7.

Memory updated

Here’s the documentation for your encode\_data function:

**encode\_data**

**Description:**

The encode\_data function encodes categorical columns in a dataset using different encoding methods, such as label encoding, one-hot encoding, and ordinal encoding. It allows users to specify which columns to encode and the encoding technique to apply.

**Parameters:**

* dataset (pandas.DataFrame): The input DataFrame containing the data to be encoded.
* encoding\_type (string, optional): The type of encoding to use. Options are:
  + 'label': Uses label encoding to convert each unique category value into a numerical label.
  + 'one-hot': Uses one-hot encoding to create binary columns for each category.
  + 'ordinal': Uses ordinal encoding to assign ordered numerical values to each category. The default value is 'label'.
* columns (list, optional): A list of columns to encode. If None, all columns in the dataset will be encoded (default: None).
* target (string, optional): The target column to encode (currently not used, can be expanded for target-specific encoding).

**Returns:**

* pandas.DataFrame: A new DataFrame with the specified columns encoded based on the selected encoding type.

A screen shot of a computer program

Description automatically generated

**Notes:**

* Label encoding is useful for ordinal categorical data, where the categories have a meaningful order.
* One-hot encoding is useful when each category is independent and should be represented as a separate feature.
* Ordinal encoding is useful when categorical variables have an inherent order.

8.

**Description:**

The feature\_engineering function applies various transformations to both numerical and categorical features of a DataFrame. It includes steps for imputation, scaling, and encoding, as well as the generation of new features such as squared and cubed terms for numerical features.

**Parameters:**

* df (pandas.DataFrame): The input DataFrame to be processed.
* numerical\_features (list): A list of column names representing numerical features in the dataset.
* categorical\_features (list): A list of column names representing categorical features in the dataset.

**Returns:**

* pd.DataFrame: A new DataFrame with transformed features, including the original processed features and newly generated ones (squared and cubed terms for numerical features).

A computer screen shot of text

Description automatically generated

A computer screen shot of text

Description automatically generated

**Notes:**

* The generated squared and cubed terms for numerical features can help capture non-linear relationships in models.
* One-hot encoding is applied to categorical features, ensuring that any unseen categories are ignored during transformation.

**9.**

**ExtractCoords**

**Description:**

The ExtractCoords function extracts geographic coordinates (latitude, longitude) from a DataFrame. It can handle both separate latitude and longitude columns, or a single combined column with coordinates. The function returns a list of tuples containing latitude and longitude as floating-point numbers.

**Parameters:**

* df (pandas.DataFrame): The DataFrame containing the coordinate data.
* lat\_col (string, optional): The column name for latitude. If not provided, the function will try to auto-detect it based on common naming conventions (e.g., columns containing 'lat').
* long\_col (string, optional): The column name for longitude. If not provided, the function will try to auto-detect it based on common naming conventions (e.g., columns containing 'lon' or 'lng').
* combined\_col (string, optional): The column name that contains both latitude and longitude in a combined format (e.g., "123.0, 123.5").

**Returns:**

* list: A list of tuples representing the extracted coordinates in the format (latitude, longitude). If the extraction fails, an empty list is returned.

A screen shot of a computer program

Description automatically generated

10.

Memory updated

Here’s the documentation for your min\_distance function:

**min\_distance**

**Description:**

The min\_distance function calculates the minimum geodesic distance from a given point to a list of other points. It uses the geodesic distance in meters, making it suitable for calculating distances between geographic coordinates (latitude, longitude).

**Parameters:**

* point (tuple): A tuple representing the coordinates (latitude, longitude) of the point from which the distance is calculated.
* list\_of\_points (list of tuples): A list of tuples, each containing the coordinates (latitude, longitude) of comparison points.

**Returns:**

* float: The minimum geodesic distance (in meters) from the given point to the closest point in the list.

**Notes:**

* The function uses the geodesic distance from the geopy.distance module, which calculates the shortest path between two points on the Earth’s surface.

A computer screen with many colorful text

Description automatically generated

**Notes:**

* The function uses the geodesic distance from the geopy.distance module, which calculates the shortest path between two points on the Earth’s surface.
* This function is suitable for geographic applications, such as finding the closest point of interest or calculating distances between two sets of coordinates.

**plot\_correlation\_heatmaps**

**Description:**

The plot\_correlation\_heatmaps function generates two heatmaps to visualize Pearson and Spearman correlations between variables in a dataset. It helps analyze linear and rank-based relationships between variables, respectively.

**Parameters:**

* data (2D numpy array or pandas.DataFrame): The input dataset for correlation analysis.
* labels (list of strings): A list of labels or column names corresponding to the variables in the dataset.
* order (list of integers, optional): A list of indices specifying the order of the variables in the heatmap for aesthetic purposes. If not provided, the default order will be used.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer program

Description automatically generated

**Example Output:**

The function creates a figure with two subplots:

1. **Pearson Correlation Heatmap**: Visualizes linear correlations between variables.
2. **Spearman Correlation Heatmap**: Visualizes rank-based correlations between variables.

**Notes:**

* The Pearson correlation coefficient measures the strength of the linear relationship between variables.
* The Spearman correlation coefficient measures the strength of the rank-based relationship (non-linear, monotonic relationship).
* The order parameter allows reordering the variables to improve the readability or aesthetics of the heatmaps.

**corrheatmap**

**Description:**

The corrheatmap function is a helper function used to plot a heatmap of correlation values. It visualizes a matrix of correlation coefficients with annotated values and color scaling.

**Parameters:**

* R (2D numpy array): The correlation matrix to visualize.
* labels (list of strings): The labels for the rows and columns of the heatmap.

**Example Usage:**

This function is called within plot\_correlation\_heatmaps to display the heatmaps.

**Example Output:**

The function displays a heatmap with:

* Color scaling from red (negative correlation) to blue (positive correlation).
* Annotated correlation coefficients inside each cell.
* Labels for both axes based on the provided labels.

**optimal\_k\_clusters using silhouette score**

**Description:**

The optimal\_k\_clusters function determines the optimal number of clusters for K-means clustering by evaluating silhouette scores across a range of cluster numbers. It plots the silhouette scores for different values of k and returns the number of clusters with the highest silhouette score.

**Parameters:**

* data (numpy array or pandas.DataFrame): The dataset on which K-means clustering will be performed.
* k\_range (range or list of integers): A range of values for the number of clusters (k) to test.

**Returns:**

* optimal\_k (int): The optimal number of clusters, which corresponds to the highest silhouette score.

A screenshot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

A graph with blue lines and numbers

Description automatically generated

**Plot:**

* **X-axis**: Number of clusters (k).
* **Y-axis**: Silhouette Score.
* **Marker**: Indicates the silhouette score for each value of k.

**Notes:**

* **Silhouette Score**: A metric used to measure how similar an object is to its own cluster compared to other clusters. A higher score indicates better-defined clusters

**find\_optimal\_clusters using elbow method**

**Description:**

The find\_optimal\_clusters function determines the optimal number of clusters for K-means clustering using the elbow method. It visualizes the elbow plot to help identify the point at which increasing the number of clusters yields diminishing returns in terms of reduced distortion (sum of squared distances from points to their assigned cluster center).

**Parameters:**

* data (numpy array or pandas.DataFrame): The dataset on which K-means clustering will be performed. It is recommended that the data be preprocessed, such as being PCA-transformed.
* k\_range (tuple of integers, optional): A range specifying the minimum and maximum number of clusters (k) to test. Default is (2, 12).

**Returns:**

* The function generates and displays an elbow plot, showing the distortion for each value of k in the specified range. The optimal number of clusters is identified by the "elbow point," where the distortion starts to decrease more slowly.

A screenshot of a computer program

Description automatically generated

A graph with a line going up

Description automatically generated

**dimensionality\_reduction**

**Description:**

The dimensionality\_reduction function applies dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-Distributed Stochastic Neighbor Embedding (t-SNE) to reduce the number of dimensions in a dataset. This is particularly useful for visualizing high-dimensional data or improving the performance of machine learning models.

**Parameters:**

* X (numpy array or pandas.DataFrame): The input data to be reduced, typically a 2D array where each row is a data point and each column is a feature.
* method (string, optional): The method to use for dimensionality reduction. Options are:
  + 'pca': Principal Component Analysis (PCA).
  + 'tsne': t-SNE for nonlinear dimensionality reduction. Default is 'pca'.
* n\_components (int, optional): The number of dimensions to reduce the data to. Default is 2.
* kwargs (optional): Additional arguments to be passed to the selected dimensionality reduction method (PCA or t-SNE).

**Returns:**

* X\_reduced (numpy array): The reduced dataset, where each data point has been projected into a lower-dimensional space of n\_components dimensions.

A screenshot of a computer program

Description automatically generated

**encode\_data**

**Description:**

The encode\_data function encodes categorical features in a DataFrame using different encoding techniques. It supports label encoding, one-hot encoding, ordinal encoding, and target encoding based on the specified encoding type.

**Parameters:**

* X (pandas.DataFrame): The input DataFrame containing features to be encoded.
* encoding\_type (string, optional): The encoding method to use. Options include:
  + 'label': Converts each unique category into an integer.
  + 'one-hot': Creates binary columns for each category.
  + 'ordinal': Assigns ordered numerical values to categories.
  + 'target': Encodes categories based on the mean value of the target variable for each category. Default is 'label'.
* columns (list of strings, optional): The list of column names to be encoded. If None, all columns will be encoded.
* target (pandas.Series, optional): The target variable used for target encoding. Required only when encoding\_type is set to 'target'.

**Returns:**

* X\_encoded (pandas.DataFrame): A DataFrame with the specified columns encoded.

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Notes:**

* **Label Encoding**: Each category in a column is assigned an integer based on alphabetical order.
* **One-Hot Encoding**: Creates new columns for each category, and assigns binary values (1 or 0).
* **Ordinal Encoding**: Assigns ordered numerical values based on category order.
* **Target Encoding**: Encodes categories based on the mean target value for each category.
  + Requires a target variable for mapping.

**plot\_estimator**

**Description:**

The plot\_estimator function visualizes the decision boundaries of a trained estimator (e.g., classifier) on a 2D feature space. It generates a mesh grid to plot the decision regions and overlays the data points to show how well the estimator separates the classes.

**Parameters:**

* estimator (object): The model to plot, which must follow the scikit-learn API and have .fit() and .predict() methods.
* X (numpy array): The feature data (2D array) used to train the estimator, where each row is a data point and each column is a feature.
* y (numpy array): The target labels corresponding to the feature data.
* mesh\_step (float, optional): The step size for the mesh grid. A smaller value provides a finer grid but increases computational time. Default is 0.1.
* cmap\_light (ListedColormap, optional): Colormap for the background decision regions. Default is ListedColormap(['#FFAAAA', '#AAFFAA', '#AAAAFF']).
* cmap\_bold (ListedColormap, optional): Colormap for the data points. Default is ListedColormap(['#FF0000', '#00FF00', '#0000FF']).

**Returns:**

* None: The function directly plots the decision boundary and the data points.

A screen shot of a computer

Description automatically generated

A screen shot of a computer program

Description automatically generated

A colorful map with dots

Description automatically generated

**Example Output:**

A plot displaying:

* **Decision Boundaries**: The background regions are colored based on the predicted class of the model in those regions.
* **Data Points**: The original data points are plotted with a distinct color for each class and a black edge for visual clarity.

**Notes:**

* The function assumes that the dataset X has exactly two features, as the decision boundary is plotted in a 2D plane.
* The colormaps can be customized by passing cmap\_light and cmap\_bold if specific color schemes are required.
* For models like KNeighborsClassifier, LogisticRegression, DecisionTreeClassifier, etc., the function will plot the decision regions based on how well the model fits the data

**linear\_regression using a function**

**Description:**

The linear\_regression function performs simple linear regression on a set of data. It calculates the best-fit line by determining the slope and y-intercept that minimize the sum of squared residuals between the predicted and actual values.

**Parameters:**

* x (array-like): The independent variable (input feature).
* y (array-like): The dependent variable (target output).

**Returns:**

* beta (float): The slope of the best-fit line.
* alpha (float): The y-intercept of the best-fit line.

**Example Usage:**

A screenshot of a computer program

Description automatically generated

A computer screen with text on it

Description automatically generated

**polynomial\_regression**

**Description:**

The polynomial\_regression function fits a polynomial regression model to the provided data. It calculates the best-fit polynomial of a specified degree and plots the polynomial curve along with the original data points.

**Parameters:**

* x (array-like): The independent variable (input feature).
* y (array-like): The dependent variable (target output).
* degree (int): The degree of the polynomial to fit to the data.

**Returns:**

* p (numpy.poly1d): A polynomial object that represents the fitted regression model.

A screenshot of a computer program

Description automatically generated

**LogisticRegression Class**

**Description:**

This LogisticRegression class implements logistic regression using gradient descent for binary classification tasks. It includes methods for fitting the model to data and predicting class labels based on the learned parameters (weights and bias). The model uses the sigmoid activation function to map the linear combination of features to a probability.

**Parameters:**

* learning\_rate (float, optional): The learning rate for gradient descent. Default is 0.01.
* num\_iterations (int, optional): The number of iterations for gradient descent optimization. Default is 1000.

**Methods:**

**sigmoid(z)**

* **Description**: Computes the sigmoid of z, mapping real numbers to the range (0, 1).
* **Parameters**:
  + z (array-like): Input values.
* **Returns**: The sigmoid transformation of z.

**fit(X, y)**

* **Description**: Fits the logistic regression model to the training data using gradient descent.
* **Parameters**:
  + X (numpy array): The training data, with shape (n\_samples, n\_features).
  + y (numpy array): The target labels, with shape (n\_samples,).
* **Operation**:
  + Initializes the weights and bias to zero.
  + Performs gradient descent for the specified number of iterations, updating weights and bias to minimize the loss.
* **Returns**: None (updates internal weights and bias).

**predict(X)**

* **Description**: Predicts the class labels for the input data using the learned weights and bias.
* **Parameters**:
  + X (numpy array): The data for which predictions are made, with shape (n\_samples, n\_features).
* **Returns**: numpy array of predicted class labels (0 or 1).

A screen shot of a computer program

Description automatically generated

A screen shot of a computer code

Description automatically generated

**Notes:**

* **Sigmoid Function**: The sigmoid function is used to map the linear combination of the input features to a probability between 0 and 1.
* **Gradient Descent**: The model uses gradient descent to minimize the logistic loss function by adjusting the weights and bias.
* **Prediction Threshold**: The class labels are predicted based on a threshold of 0.5. If the predicted probability is greater than 0.5, the class label is 1; otherwise, it is 0.

**LinearSVM Class**

**Description:**

The LinearSVM class implements a linear Support Vector Machine (SVM) for binary classification using gradient descent optimization. It uses a hinge loss function and incorporates L2 regularization to penalize large weights. The model adjusts weights and bias through gradient updates over multiple iterations.

**Parameters:**

* learning\_rate (float, optional): The learning rate for gradient descent. Default is 0.001.
* lambda\_param (float, optional): The regularization parameter (L2 penalty). Default is 0.01.
* n\_iters (int, optional): The number of iterations for gradient descent. Default is 1000.

**Methods:**

**fit(X, y)**

* **Description**: Fits the SVM model to the training data using gradient descent. The method minimizes the hinge loss function while regularizing the model using the L2 norm of the weights.
* **Parameters**:
  + X (numpy array): Training data, shape (n\_samples, n\_features).
  + y (numpy array): Target labels, shape (n\_samples,), where the labels are expected to be binary (0 or 1).
* **Operation**:
  + Converts the target labels y into values of -1 and 1 (SVM requires this format).
  + Initializes the weights and bias to zero.
  + Performs gradient updates based on whether the sample satisfies the hinge loss condition.
* **Returns**: None (updates internal weights w and bias b).

**predict(X)**

* **Description**: Predicts the class labels for input data using the learned SVM model.
* **Parameters**:
  + X (numpy array): Input data, shape (n\_samples, n\_features).
* **Returns**:
  + numpy array: Predicted class labels (either -1 or 1), based on the sign of the linear output.

A screen shot of a computer code

Description automatically generated

**Notes:**

* **Hinge Loss**: The hinge loss function penalizes points that lie inside the margin or on the wrong side of the decision boundary.
* **Regularization**: The regularization term (lambda\_param) helps to prevent overfitting by penalizing large weight values.
* **Binary Classification**: This implementation of SVM is designed for binary classification. It converts target labels of 0 to -1 as required by SVM.

**Limitations:**

* The LinearSVM class is designed for binary classification and will not work out-of-the-box for multiclass problems.
* The gradient descent approach may require careful tuning of the learning rate and number of iterations for convergence.

## Author

Aremu Akintomiwa James

Master of Data science