Contents

[MACHINE LEARNING ALGORITHMS 2](#_Toc204521964)

[MACHINE LEARNING ALGORITHMS CLASSIFICATION 2](#_Toc204521965)

[ANOMALY DETECTION ALGORITHM 2](#_Toc204521966)

[CLUSTERING ALGORITHM 2](#_Toc204521967)

[DATA PRE-PROCESSING USING PYTHON 3](#_Toc204521968)

[DATA PROCESSING USING PYTHON 7](#_Toc204521969)

[REGRESSION 8](#_Toc204521970)

[SIMPLE LINEAR REGRESSION 8](#_Toc204521971)

[MULTIPLE LINEAR REGRESSION 8](#_Toc204521972)

[POLYNOMIAL REGRESSION 9](#_Toc204521973)

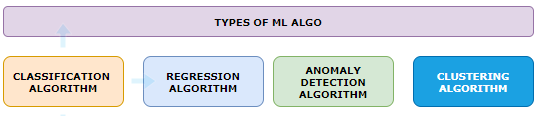
[SUPPORT VECTOR REGRESSION 9](#_Toc204521974)

[DECISION TREE REGRESSION 9](#_Toc204521975)

[RANDOM FOREST REGRESSION 9](#_Toc204521976)

# MACHINE LEARNING ALGORITHMS

# MACHINE LEARNING ALGORITHMS CLASSIFICATION



## ANOMALY DETECTION ALGORITHM

* This is used to detect if data deviates from the norm
* Example: used to detect fraudulent credit card purchase

## CLUSTERING ALGORITHM

A computer screen with text and a diagram

AI-generated content may be incorrect.

# DATA PRE-PROCESSING USING PYTHON

|  |  |
| --- | --- |
|  | * In this example we will perform data preprocessing step on this following data * It’s a user profile data – of an ecommerce website with a flag which says whether user has made purchase or not! * As a data processing step – we create two entities –  1. The first is **the matrix of features**, which contains separately these three columns (country, age, salary.) 2. And second is the dependent variable vector, which is last column(“Purchased”), because that's the column we want to predict.   *Note : This is exactly what we must do in this first data pre-processing phase.* |

|  |  |
| --- | --- |
| Step 1: Importing the libraries | import numpy as np import matplotlib.pyplot as plt import pandas as pd |
| Step 2: Importing the dataset | dataset = pd.read\_csv('Data.csv') X = dataset.iloc[:, :-1].values y = dataset.iloc[:, -1].values  OUTPUT  [['France' 44.0 72000.0]  ['Spain' 27.0 48000.0]  ['Germany' 30.0 54000.0]  ['Spain' 38.0 61000.0]  ['Germany' 40.0 nan]  ['France' 35.0 58000.0]  ['Spain' nan 52000.0]  ['France' 48.0 79000.0]  ['Germany' 50.0 83000.0]  ['France' 37.0 67000.0]]  ['No' 'Yes' 'No' 'No' 'Yes' 'Yes' 'No' 'Yes' 'No' 'Yes'] |
| Step 3: Missing Data   * For missing data, we can make use of Python library Scikit-learn. It is an open-source machine learning library built on top of **NumPy**, **SciPy**, and **matplotlib**. * It provides simple and efficient tools for data mining and data analysis. * For example - For missing salary - We will replace the missing salary with average salary in the column   **from sklearn.impute import SimpleImputer**  **imputer= SimpleImputer(missing\_values=np.nan, strategy='mean') imputer.fit(X[:, 1:3]) # Assuming columns 1 and 2 have missing values X[:, 1:3] = imputer.transform(X[:, 1:3]) # Transform the data to fill missing values**   * SimpleImputer from scikit-learn to fill missing values (NaN) in specific columns of the feature matrix X with the mean of each column. * It fits the imputer on columns 1 and 2 (indexing starts at 0), replaces missing values with the computed mean | |

Step 4: Encoding the categorial data

* Encoding categorical data is **crucial in machine learning** because most ML algorithms require **numerical input** to perform mathematical computations.

Common Encoding Techniques

|  |  |  |
| --- | --- | --- |
| Technique | Description | Best For |
| Label Encoding | Assigns A Unique Number To Each Category | Ordinal Data (E.G., "Low", "High") |
| One-Hot Encoding | Creates Binary Columns For Each Category | Nominal Data (E.G., "Red", "Blue") |
| Ordinal Encoding | Encodes Categories With Meaningful Order | Ordered Categories |
| Target Encoding | Replaces Categories With The Mean Of The Target Variable For Each Category | High-Cardinality Categorical Data |

Example

Suppose we have a column Fuel Type with values: ["Petrol", "Diesel", "Electric"]

* Label Encoding: Petrol → 0, Diesel → 1, Electric → 2 (May imply an order that doesn’t exist)
* One-Hot Encoding:

| Petrol | Diesel | Electric |
| --- | --- | --- |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 1 |

Why Encode Categorical Data?

* ML Models Work with Numbers
  1. Algorithms like linear regression, decision trees, and neural networks **cannot interpret text** or labels directly. They need **numerical representations** to process the data.
* Preserves Information
  1. Encoding transforms categories into numbers **without losing the meaning** of the data. For example, converting "Red", "Blue", "Green" into numerical form allows the model to still distinguish between them.
* Improves Model Performance
  1. Proper encoding helps the model **understand relationships** between variables, which can lead to **better predictions** and **faster training**.
* Avoids Bias from Arbitrary Numbers
  1. Some encoding methods (like **One-Hot Encoding**) prevent the model from assuming an **ordinal relationship** where none exists.
  2. For example, assigning "Low", "Medium", "High" as 1, 2, 3 implies a ranking, which may or may not be appropriate.

CODE

Taking the example further will apply “Hot Encoding” of “County” column and “label encoding” to the “Purchased” column.

One Hot Encoding OF Country Column

|  |
| --- |
| from sklearn.preprocessing import OneHotEncoder  ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),[0])],remainder='passthrough') X = np.array(ct.fit\_transform(X)) # Apply one-hot encoding to the first column |

LABELLED Encoding OF Country Column

|  |
| --- |
| from sklearn.preprocessing import LabelEncoder  le = LabelEncoder() y = le.fit\_transform(y) # Apply label encoding to the dependent variable |

EXAMPLE – CODING EXERCISE

Dataset

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| PassengerId | Survived | Pclass | Name | Sex | Age | SibSp | Parch | Ticket | Fare | Cabin | Embarked |
| 2 | 1 | 1 | Cumings, Mrs. John Bradley (Florence Briggs Thayer) | female | 38 | 1 | 0 | PC 17599 | 71.2833 | C85 | C |
| 4 | 1 | 1 | Futrelle, Mrs. Jacques Heath (Lily May Peel) | female | 35 | 1 | 0 | 113803 | 53.1 | C123 | S |
| 7 | 0 | 1 | McCarthy, Mr. Timothy J | male | 54 | 0 | 0 | 17463 | 51.8625 | E46 | S |
| 11 | 1 | 3 | Sandstrom, Miss. Marguerite Rut | female | 4 | 1 | 1 | PP 9549 | 16.7 | G6 | S |
| 12 | 1 | 1 | Bonnell, Miss. Elizabeth | female | 58 | 0 | 0 | 113783 | 26.55 | C103 | S |
| 22 | 1 | 2 | Beesley, Mr. Lawrence | male | 34 | 0 | 0 | 248698 | 13 | D56 | S |

Coding Exercise 3: Encoding Categorical Data for Machine Learning

**1**: Import required libraries - Pandas, Numpy, and required classes for this task - ColumnTransformer, OneHotEncoder, LabelEncoder.

**2**: Start by loading the Titanic dataset into a pandas data frame. This can be done using the pd.read\_csv function. The dataset's name is 'titanic.csv'.

**3**: Identify the categorical features in your dataset that need to be encoded. You can store these feature names in a list for easy access later.

**4**: To apply OneHotEncoding to these categorical features, create an instance of the ColumnTransformer class. Make sure to pass the OneHotEncoder() as an argument along with the list of categorical features.

**5**: Use the fit\_transform method on the instance of ColumnTransformer to apply the OneHotEncoding.

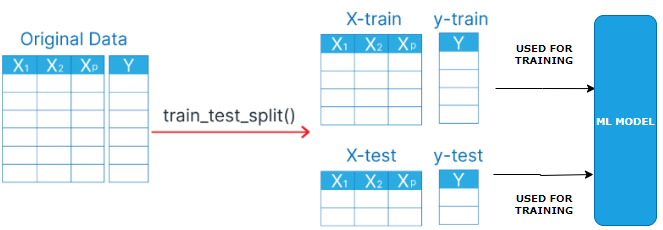
**6**: The output of the fit\_transform method should be converted into a NumPy array for further use.

**7**: The 'Survived' column in your dataset is the dependent variable. This is a binary categorical variable that should be encoded using LabelEncoder.

**8.**  Print the updated matrix of features and the dependent variable vector

|  |
| --- |
| # Importing the necessary libraries import pandas as pd import numpy as np from sklearn.compose import ColumnTransformer from sklearn.preprocessing import OneHotEncoder, LabelEncoder  # Load the dataset dataset = pd.read\_csv("titanic.csv")  # Identify the categorical data categorical\_features = ['Sex', 'Embarked', 'Pclass']  # Implement an instance of the ColumnTransformer class  ct = ColumnTransformer(transformers=[('encoder', OneHotEncoder(),categorical\_features)],remainder='passthrough')  # Apply the fit\_transform method on the instance of ColumnTransformer ct\_fit = ct.fit\_transform(dataset) # Apply one-hot encoding to the first column  # Convert the output into a NumPy array X = np.array(ct.fit\_transform(dataset))  # Use LabelEncoder to encode binary categorical data le = LabelEncoder()  Y = le.fit\_transform(dataset["Survived"]) # Apply label encoding to the dependent variable  # Print the updated matrix of features and the dependent variable vector print(X) print(Y) |

Step 5: Training Versus Test Data



|  |
| --- |
| from sklearn.model\_selection import train\_test\_split  x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1) |

The train\_test\_split method arrays or matrices into random train and test subsets.

Parameters:

* X: Features (input data).
* y: Labels (target data).
* test\_size: Fraction or number of samples to use as the test set (e.g., 0.2 means 20% for testing).
* train\_size: Fraction or number of samples for training (optional, usually inferred).
* random\_state: Seed for random number generator (ensures reproducibility).
* shuffle: Whether to shuffle data before splitting (default is True).
* stratify: Ensures the split preserves the proportion of labels (optional, usually for classification).
* Returns: Splits the data into x\_train, x\_test, y\_train, and y\_test.

EXAMPLE

1. Import necessary Python libraries: pandas, train\_test\_split from sklearn.model\_selection, and StandardScaler from sklearn.preprocessing.
2. Load the Iris dataset using Pandas read.csv. Dataset name is iris.csv.
3. Use train\_test\_split to split the dataset into an 80-20 training-test set.
4. Apply random\_state with 42 value in train\_test\_split function for reproducible results.
5. Print X\_train, X\_test, Y\_train, and Y\_test to understand the dataset split.
6. Use StandardScaler to apply feature scaling on the training and test sets.
7. Print scaled training and test sets to verify feature scaling.

Step 6: Feature Scaling

# DATA PROCESSING USING PYTHON

# REGRESSION

* A **regression model** is a type of statistical or machine learning model used to understand the relationship between a **dependent variable** (what you're trying to predict) and one or more **independent variables** (the inputs or predictors).
* **In simple terms:** A regression model helps answer questions like:
  + "How does the price of a house depend on its size, location, and number of bedrooms?"
  + "How does advertising spending affect sales?"

Types Of Regression Models

1. Linear Regression

* Assumes a straight-line relationship between variables.
* Example: y = a + bx
* Multiple Linear Regression:
* Like linear regression, but with multiple input variables.
* Example: y = a + b*1x*1 + b*2x*2 + … + b*nx*n
* Polynomial Regression:
* Models curved relationships by including powers of the input variables.
* Example: y = a + bx + cx2
* Logistic Regression:
* Used when the output is categorical (e.g., yes/no, 0/1).
* Despite the name, it's used for classification, not regression.
* Ridge, Lasso, And Elastic Net Regression:
* Variants of linear regression that include regularization to prevent overfitting.

What It’s Used For

* Predicting future values (e.g., stock prices, sales).
* Understanding relationships between variables.
* Making data-driven decisions in business, science, and engineering.

SIMPLE LINEAR REGRESSION

A diagram of equations

AI-generated content may be incorrect.

## MULTIPLE LINEAR REGRESSION

## POLYNOMIAL REGRESSION

## SUPPORT VECTOR REGRESSION

## DECISION TREE REGRESSION

## RANDOM FOREST REGRESSION