Contents

[MACHINE LEARNING 2](#_Toc199099849)

[MACHINE LEARNING PROCESS 2](#_Toc199099850)

[TRAINING SET AND TEST SET 2](#_Toc199099851)

[FEATURE SCALING 4](#_Toc199099852)

[NORMALIZATION 5](#_Toc199099853)

[STANDARDIZATION 5](#_Toc199099854)

[EXAMPLE - NORMALIZATION 6](#_Toc199099855)

[DATA PRE-PROCESSING USING PYTHON 7](#_Toc199099856)

[DATA PROCESSING USING PYTHON 11](#_Toc199099857)

[REGRESSION 11](#_Toc199099858)

[SIMPLE LINEAR REGRESSION 12](#_Toc199099859)

[MULTIPLE LINEAR REGRESSION 12](#_Toc199099860)

[POLYNOMIAL REGRESSION 12](#_Toc199099861)

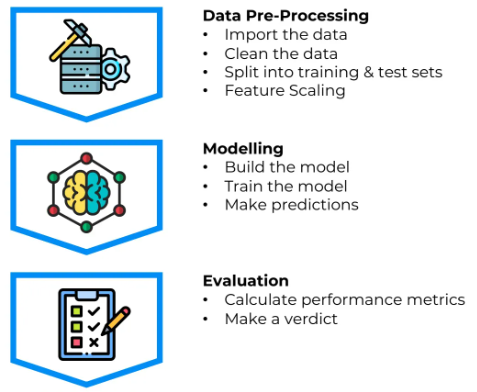
[SUPPORT VECTOR REGRESSION 12](#_Toc199099862)

[DECISION TREE REGRESSION 12](#_Toc199099863)

[RANDOM FOREST REGRESSION 12](#_Toc199099864)

# MACHINE LEARNING

## MACHINE LEARNING PROCESS

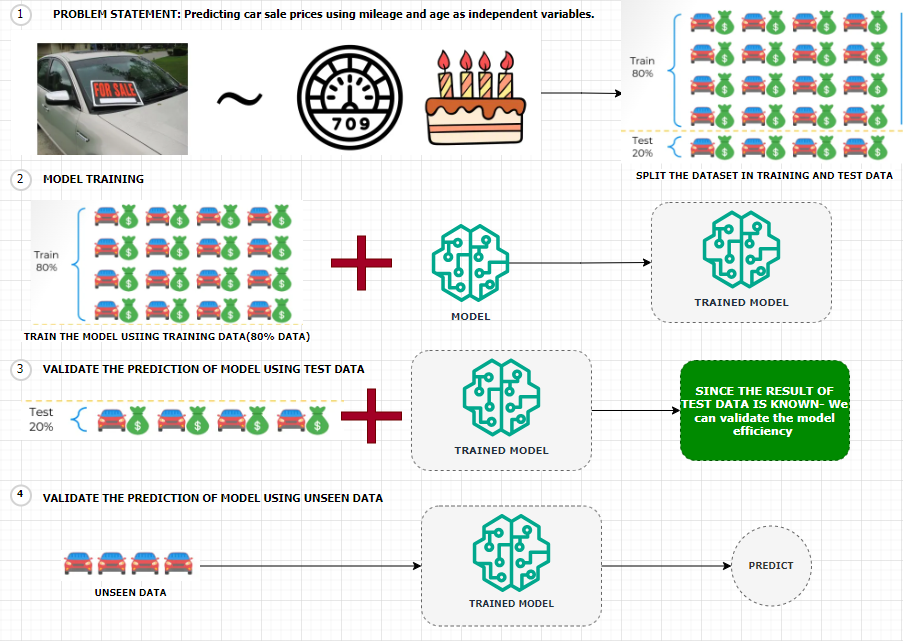


## TRAINING SET AND TEST SET

A diagram of a car with a price tag

AI-generated content may be incorrect.

Importance Of Splitting Data Set into a training set and a test set.



Predicting Car Sale Prices Using Linear Regression

Let’s consider a scenario where we aim to **predict the sale prices of cars** using a **linear regression model**. In this case:

* **Dependent Variable**: Sale price of the car
* **Independent Variables**:
  + Mileage of the car
  + Age of the car

**Dataset Overview**

We are provided with a dataset containing information on **20 cars**. To build and evaluate our model effectively, we split the dataset into two parts:

* **Training Set (80%)**:
  + Contains data for **16 cars**
  + Used to train the linear regression model
* **Test Set (20%)**:
  + Contains data for **4 cars**
  + Set aside before training to evaluate the model’s performance

Model Training and Evaluation

1. **Model Building**:  
   We train the linear regression model using only the training set. This means the model learns the relationship between mileage, age, and sale price from these 16 cars.
2. **Prediction on Test Set**:  
   After training, we apply the model to the 4 cars in the test set. These cars were **not part of the training process**, so the model has **no prior knowledge** of them.
3. **Comparison with Actual Prices**:  
   Since we already know the **actual sale prices** of the test set cars, we can now compare:
   * **Predicted Prices** (from the model)
   * **Actual Prices** (from the dataset)
4. **Model Evaluation**:  
   This comparison allows us to assess how well our model performs on unseen data. Metrics such as **Mean Absolute Error (MAE)** or **Root Mean Squared Error (RMSE)** can be used to quantify the model’s accuracy.

## FEATURE SCALING

Feature scaling **transforms the values of features to be on a similar scale**, typically to improve model performance and training stability.

We can understand feature scaling using the example below. Note: ***Feature scaling is always applied at column level***

A table with numbers and lines

AI-generated content may be incorrect.

There are 2 main techniques of feature scaling

1. **NORMALIZATION**
2. **STANDARDIZATION**

### NORMALIZATION

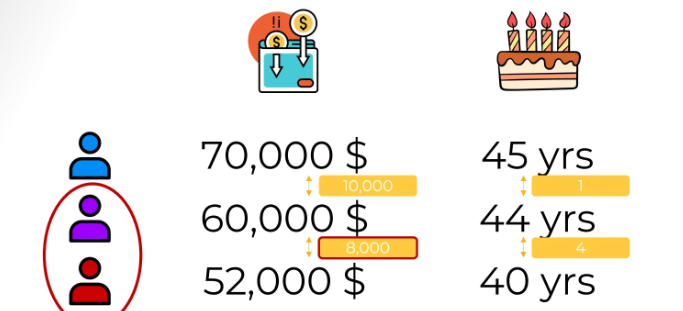
|  |  |
| --- | --- |
|  | * The normalization value lies between the closed interval of [0;1] |

|  |  |  |
| --- | --- | --- |
| **X1 (Price)** | **X-XMIN** | **Normalized Value(X1)** |
| $179.43 | $0.00 | 0.00 |
| $641.87 | $179.43 | 0.39 |
| $556.30 | $376.87 | 0.814959779 |
| $578.47 | $116.03 | 0.250908226 |
| $591.12 | $411.69 | 0.890256033 |
|  |  |  |
| X1-MAX |  | $641.87 |
| X1-MIN |  | $179.43 |
| X1MAX - X1MIN |  | **$462.44** |

### STANDARDIZATION

|  |  |  |
| --- | --- | --- |
| A math equation with numbers and symbols  AI-generated content may be incorrect. | µ | Average |
|  | Standard Deviation |
| * The value is lies in closes interval of [-3,3] * If data has some outliers – it will exist outside this range | |

### EXAMPLE - NORMALIZATION



1. Let's imagine we have a data set where we have two columns, annual income of a person and their age of

a blue, purple and red person.

1. **We must identify whether the purple person is like a “red” person or “blue” person . This is the task of clustering data. For that we need to do normalization of data as the units of the data is not uniform**

A close-up of a number

AI-generated content may be incorrect.

1. After normalizing, our values will look like above. Hence with the normalized data – From salary column perspective. The purple person is almost right in the middle between the red and the blue people(0.44), whereas in the age column, the purple person is closest to the blue person.

|  |
| --- |
| Scikit-learn (also written as scikit-learn or sklearn) is a powerful and widely used open-source machine learning library for the Python. It provides simple and efficient tools for:  🔍 Key Features   * Classification: Identifying which category an object belongs to (e.g., spam detection). * Regression: Predicting a continuous-valued attribute (e.g., house prices). * Clustering: Grouping similar data points (e.g., customer segmentation). * Dimensionality Reduction: Reducing the number of features (e.g., PCA). * Model Selection: Comparing, validating, and choosing parameters and models. * Preprocessing: Feature extraction, normalization, and transformation.   🧰 Built On  Scikit-learn is built on top of:   * NumPy: For numerical operations. * SciPy: For scientific computing. * Matplotlib: For plotting (indirectly used). * joblib: For model persistence and parallel processing. |

# 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.

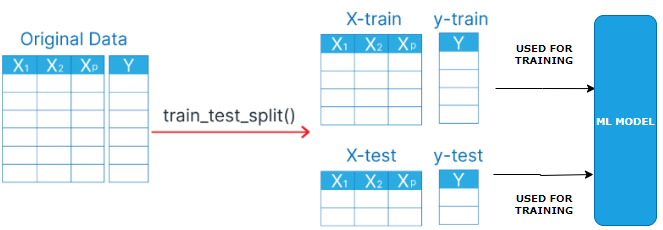
**6**: The output of the fit\_transform method should be converted into a NumPy array for further use.

**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



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