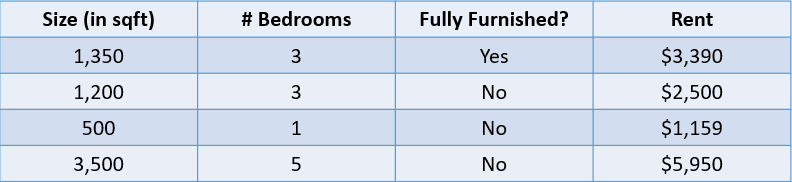
All machine learning models can be classified as supervised or unsupervised. The biggest difference between the two is that a supervised algorithm requires labeled input and output training data, while an unsupervised model can process raw, unlabeled datasets.

Supervised machine learning models can then be further classified into regression and classification algorithms.

**Machine Learning Regression Models**

Regression algorithms are used to predict a continuous outcome (y) using independent variables (x).

For example, look at the table below:



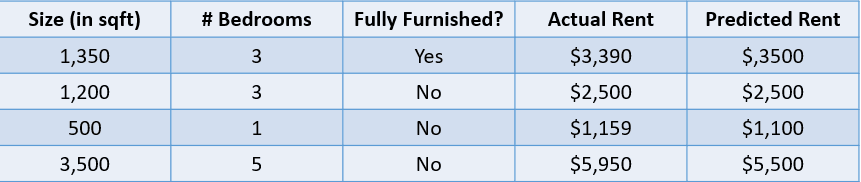
In this case, we would like to predict the rent of a house based on its size, the number of bedrooms, and whether it is fully furnished. The dependent variable, “Rent”, is numeric, which makes this a regression problem.

**Regression Metrics**

A common misconception by data science beginners is that a regression model can be evaluated using a metric like accuracy. Accuracy is a metric used to assess the performance of classification models, as will be explained later in this article.

Regression models, on the other hand, are evaluated using metrics such as MAE (Mean Absolute Error), MSE (Mean Squared Error), and RMSE (Root Mean Squared Error).

Let’s add a predicted value to the house price problem above and evaluate these predictions using a few regression metrics:



**1. Mean Absolute Error:**

The mean absolute error calculates the sum of the difference between all true and predicted values, and divides this by the total number of observations. Here is the formula to calculate MAE:

MAE Formula

**2. Mean Squared Error:**

The formula to calculate a model’s mean squared error is similar to that of its mean absolute error:

MSE Formula

Note that while the mean absolute error calculates the average absolute distance between the actual and predicted value, the mean squared error finds the averaged squared distance between actual and predicted values.

**3. Root Mean Squared Error:**

The RMSE of an estimator is calculated by finding the square root of its mean squared error. One advantage of calculating a dataset’s RMSE over its MSE is that the error is returned in the same unit of the variable we are predicting.

**Simple Linear Regression**

**Linear Regression is a fundamental algorithm in machine learning and statistics used for modeling the relationship between a dependent variable (target) and one or more independent variables (features). When there is one independent variable, it is called Simple Linear Regression; when there are multiple independent variables, it is called Multiple Linear Regression.**

Linear regression is a linear approach to modeling the relationship between a dependent and one or more independent variables. This algorithm involves finding a line that best fits the data at hand.

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Sample Data

data = pd.DataFrame({

'X1': [1, 2, 3, 4, 5],

'X2': [2, 4, 6, 8, 10],

'Y': [1.1, 2.0, 2.9, 4.1, 5.0]

})

# Features and Target

X = data[['X1', 'X2']]

y = data['Y']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Model Initialization

model = LinearRegression()

# Train the Model

model.fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluation

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Coefficients: {model.coef\_}")

print(f"Intercept: {model.intercept\_}")

print(f"MSE: {mse}")

print(f"R^2: {r2}")

**Concept**

Linear Regression assumes a linear relationship between the dependent variable yyy and the independent variables xxx. The goal is to find the best-fitting line (or hyperplane in higher dimensions) that minimizes the error between the predicted values and the actual values.

The equation of the model is:

y=β0+β1x1+β2x2+…+βnxn+ϵy = \beta\_0 + \beta\_1 x\_1 + \beta\_2 x\_2 + \ldots + \beta\_n x\_n + \epsilony=β0​+β1​x1​+β2​x2​+…+βn​xn​+ϵ

Where:

* yyy: Dependent variable (output/target).
* x1,x2,…,xnx\_1, x\_2, \ldots, x\_nx1​,x2​,…,xn​: Independent variables (features).
* β0\beta\_0β0​: Intercept.
* β1,β2,…,βn\beta\_1, \beta\_2, \ldots, \beta\_nβ1​,β2​,…,βn​: Coefficients (slopes for each feature).
* ϵ\epsilonϵ: Error term (residuals).

**Steps in Linear Regression**

1. **Formulate the Problem**: Identify the dependent and independent variables.
2. **Fit the Model**:
   * Estimate the coefficients (β\betaβ) by minimizing the cost function: Cost Function: J(β)=12m∑i=1m(yi−y^i)2\text{Cost Function: } J(\beta) = \frac{1}{2m} \sum\_{i=1}^m (y\_i - \hat{y}\_i)^2Cost Function: J(β)=2m1​i=1∑m​(yi​−y^​i​)2 Here, y^i=β0+∑j=1nβjxij\hat{y}\_i = \beta\_0 + \sum\_{j=1}^n \beta\_j x\_{ij}y^​i​=β0​+∑j=1n​βj​xij​.
3. **Solve Using Optimization**:
   * In simple cases, coefficients can be derived using the Normal Equation: β=(XTX)−1XTy\beta = (X^T X)^{-1} X^T yβ=(XTX)−1XTy
   * For large datasets, optimization techniques like **Gradient Descent** are used.
4. **Evaluate the Model**:
   * Use metrics such as:
     + **Mean Squared Error (MSE)**: Measures average squared difference between actual and predicted values.
     + **R-squared (R2R^2R2)**: Indicates the proportion of variance explained by the model.
5. **Interpret Results**:
   * Analyze the coefficients to understand the relationship between variables.

**Assumptions of Linear Regression**

1. **Linearity**: The relationship between independent and dependent variables is linear.
2. **Independence**: Observations are independent.
3. **Homoscedasticity**: Constant variance of residuals.
4. **Normality**: Residuals are normally distributed.
5. **No Multicollinearity**: Independent variables should not be highly correlated.

**Advantages**

* Simple and easy to interpret.
* Works well for linearly separable data.
* Computationally efficient.

**Limitations**

* Sensitive to outliers.
* Assumes linear relationships.
* Performance degrades with multicollinearity or non-linearity.

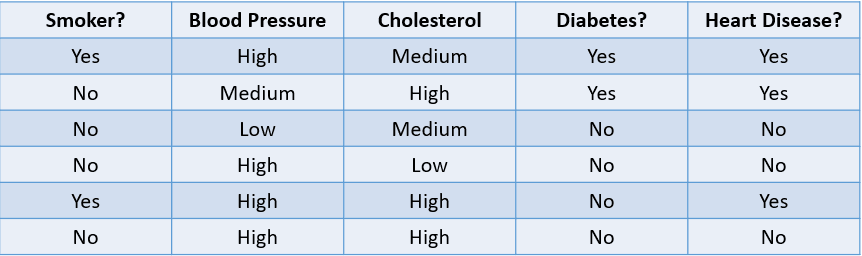
For datasets where the relationship between variables is not linear, techniques like Polynomial Regression or other non-linear models should be used.

---------------------------------------------------------------------------------------------------------

**Machine Learning Classification Models**

We use Classification algorithms to predict a discrete outcome (y) using independent variables (x). The dependent variable, in this case, is always a class or category.

For example, predicting whether a patient is likely to develop heart disease based on their risk factors is a classification problem:

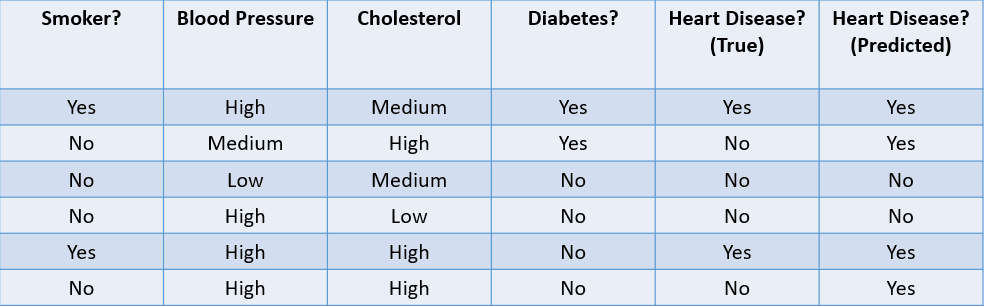


The table above showcases a classification problem with four independent variables and one dependent variable, heart disease. Since there are only two possible outcomes (Yes and No), this is called a binary classification problem.

**Classification Metrics**

There are many ways to evaluate a classification model. While accuracy is the most used metric, it is not always the most reliable.

Let’s look at some common methods used to evaluate a classification algorithm based on the dataset below:



**1. Accuracy**: Accuracy can be defined as the fraction of correct predictions made by the machine learning model.

The formula to calculate accuracy is:

Accuracy Formula

In this case, the accuracy is 46, or 0.67.

**2. Precision**: Precision is a metric used to calculate the quality of positive predictions made by the model. It is defined as:

Precision Formula

The above model has a precision of 24, or 0.5.

**3. Recall**: Recall is used to calculate the quality of negative predictions made by the model. It is defined as:

Recall Formula

The above model has a recall of 2/2 or 1.

We often use a metric called the F1-Score to find the harmonic mean of a classifier’s precision and recall. Simply put, the F1-Score combines precision and recall into a single metric by computing their average.

**Logistic Regression**

Logistic Regression is a statistical method used for binary classification problems, where the target variable has two possible outcomes (e.g., 0 or 1, True or False). Despite its name, logistic regression is a classification algorithm, not a regression algorithm. It predicts the probability of a binary outcome using a sigmoid function.

Concept

The logistic regression model estimates the probability of the dependent variable yy being 1 given the independent variables XX. The equation is modeled as:

P(y=1∣X)=11+e−(β0+β1x1+β2x2+…+βnxn)P(y=1|X) = \frac{1}{1 + e^{-(\beta\_0 + \beta\_1 x\_1 + \beta\_2 x\_2 + \ldots + \beta\_n x\_n)}}

Where:

* P(y=1∣X)P(y=1|X): Probability of the target being 1.
* β0\beta\_0: Intercept.
* β1,β2,…,βn\beta\_1, \beta\_2, \ldots, \beta\_n: Coefficients for each feature.
* ee: Exponential function.

The model predicts the probability, and the final classification is determined using a threshold (e.g., 0.5). If P(y=1∣X)≥0.5P(y=1|X) \geq 0.5, predict y=1y=1; otherwise, predict y=0y=0.

Steps in Logistic Regression

1. Formulate the Problem:
   * Identify the target variable (binary) and the independent variables (features).
2. Fit the Model:
   * Logistic Regression uses the Maximum Likelihood Estimation (MLE) to estimate coefficients (β\beta).
   * The cost function to optimize is the Log-Loss (Cross-Entropy Loss): J(β)=−1m∑i=1m[yilog⁡(y^i)+(1−yi)log⁡(1−y^i)]J(\beta) = -\frac{1}{m} \sum\_{i=1}^m \left[ y\_i \log(\hat{y}\_i) + (1 - y\_i) \log(1 - \hat{y}\_i) \right] Where y^i=P(y=1∣Xi)\hat{y}\_i = P(y=1|X\_i) is the predicted probability.
3. Optimize:
   * Gradient Descent or advanced optimization algorithms (e.g., L-BFGS) are used to minimize the cost function.
4. Evaluate the Model:
   * Metrics for classification include:
     + Accuracy: Correct PredictionsTotal Predictions\frac{\text{Correct Predictions}}{\text{Total Predictions}}
     + Precision: True PositivesTrue Positives + False Positives\frac{\text{True Positives}}{\text{True Positives + False Positives}}
     + Recall: True PositivesTrue Positives + False Negatives\frac{\text{True Positives}}{\text{True Positives + False Negatives}}
     + F1-Score: Harmonic mean of precision and recall.
     + AUC-ROC Curve: Measures the trade-off between true positive rate and false positive rate.

Assumptions of Logistic Regression

1. Binary Outcome: The target variable should be binary (can be extended to multi-class problems using softmax or one-vs-rest strategies).
2. Independent Observations: Observations should be independent of each other.
3. Linearity of Log-Odds: The independent variables should have a linear relationship with the log-odds.
4. No Multicollinearity: Independent variables should not be highly correlated.
5. Large Sample Size: Helps improve the reliability of maximum likelihood estimates.

Implementation in Python

Here’s an example of Logistic Regression using the scikit-learn library:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Sample Data

data = pd.DataFrame({

'Feature1': [1, 2, 3, 4, 5],

'Feature2': [2, 4, 6, 8, 10],

'Target': [0, 0, 1, 1, 1]

})

# Features and Target

X = data[['Feature1', 'Feature2']]

y = data['Target']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Model Initialization

model = LogisticRegression()

# Train the Model

model.fit(X\_train, y\_train)

# Predictions

y\_pred = model.predict(X\_test)

# Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

Advantages

1. Simple and easy to implement.
2. Efficient for binary classification problems.
3. Outputs probabilities, useful for decision-making.

Limitations

1. Assumes linearity of log-odds, which may not hold for all datasets.
2. Sensitive to outliers.
3. May struggle with complex relationships between features and the target variable.

For more complex problems, consider other classifiers like Decision Trees, Random Forests, or Neural Networks.

Logistic regression is a simple classification model that predicts the probability of an event taking place.

unlike the linear regression algorithm, logistic regression is modeled with an S-shaped curve. This is known as the logistic function and has the following formula:

Logistic Function Formula

While the linear function does not have an upper and lower bound, the logistic function ranges between 0 and 1. The model predicts a probability that ranges from 0 to 1, which determines the class that the data point belongs to.

In this spam email example, if the text contains little to no suspicious keywords, then the probability of it being spam will be low and close to 0. On the other hand, an email with many suspicious keywords will have a high probability of being spam, close to 1.

For binary classification problems like the above, the default threshold of a logistic regression model is 0.5, which means that data points with a higher probability than 0.5 will automatically be assigned a label of 1. This threshold value can be manually changed depending on your use case to achieve better results.

Now, recall that in linear regression, we found the line of best fit by minimizing the sum of squared error between the predicted and true values. In logistic regression, however, the coefficients are estimated using a technique called maximum likelihood estimation instead of least squares.

---------------------------------------------------------------------------------------------------------

**K-Nearest Neighbors**

KNN is a classification algorithm that classifies a data point based on what group the data points nearest to it belong to.

The K-Nearest Neighbors algorithm works like this:

* **Step 1**: The model first stores all the training data.
* **Step 2:** Then, it calculates the distance from the new data point to all points in the dataset.
* **Step 3**: The model sorts these data points based on their distance to the new data point.
* **Step 4**: The new data point is assigned to the class of its nearest neighbors depending on the value of “k.”

The **K-Nearest Neighbors (KNN)** algorithm is a simple and versatile supervised learning algorithm used for both classification and regression tasks. It works by finding the KK nearest data points (neighbors) to a given query point and making predictions based on the majority class (for classification) or the average value (for regression) of these neighbors.

**How KNN Works**

1. **Choose KK**: Decide the number of neighbors (KK) to consider.
2. **Calculate Distance**: Compute the distance between the query point and all points in the dataset using a distance metric like:
   * **Euclidean Distance**: d=∑i=1n(xi−yi)2d = \sqrt{\sum\_{i=1}^n (x\_i - y\_i)^2}
   * **Manhattan Distance**: d=∑i=1n∣xi−yi∣d = \sum\_{i=1}^n |x\_i - y\_i|
   * Other metrics: Minkowski, Hamming, Cosine, etc.
3. **Identify Neighbors**: Select the KK data points closest to the query point.
4. **Predict the Output**:
   * **For Classification**: Use majority voting among the neighbors.
   * **For Regression**: Take the average (or weighted average) of the neighbors' values.

**Steps in KNN Algorithm**

1. **Prepare the Data**:
   * Ensure the dataset is properly scaled because KNN is sensitive to the scale of features (e.g., use MinMaxScaler or StandardScaler).
2. **Choose KK**:
   * KK is usually an odd number for binary classification to avoid ties.
   * Use techniques like cross-validation to determine the optimal KK.
3. **Compute Distances**:
   * For each test point, calculate the distance to all training points.
4. **Make Predictions**:
   * Sort the distances, pick the top KK nearest neighbors, and predict based on their values.
5. **Evaluate the Model**:
   * Use metrics like accuracy, precision, recall, and F1-score for classification.
   * Use Mean Squared Error (MSE) or Mean Absolute Error (MAE) for regression.

**Implementation in Python**

Here’s an example of KNN for classification using the scikit-learn library:

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.neighbors import KNeighborsClassifier

from sklearn.metrics import accuracy\_score, confusion\_matrix, classification\_report

# Sample Dataset

data = pd.DataFrame({

'Feature1': [1, 2, 3, 6, 7, 8],

'Feature2': [2, 3, 4, 5, 6, 7],

'Target': [0, 0, 0, 1, 1, 1]

})

# Features and Target

X = data[['Feature1', 'Feature2']]

y = data['Target']

# Train-Test Split

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Initialize KNN with K=3

knn = KNeighborsClassifier(n\_neighbors=3)

# Train the Model

knn.fit(X\_train, y\_train)

# Predictions

y\_pred = knn.predict(X\_test)

# Evaluation

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

**Advantages**

1. Simple to understand and implement.
2. No assumptions about data distribution.
3. Effective for small datasets and problems with well-separated classes.

**Limitations**

1. **Computational Cost**: Can be slow for large datasets as it requires computing distances for all points.
2. **Storage**: Needs to store the entire training dataset.
3. **Curse of Dimensionality**: Performance degrades with high-dimensional data.
4. **Sensitive to Noise**: Outliers can affect predictions.

**Tips for Using KNN**

1. **Scale Features**: Always scale features to avoid dominance of variables with larger scales.
2. **Optimize KK**: Use cross-validation to find the best KK. Too small a KK can lead to overfitting, while too large a KK can lead to underfitting.
3. **Dimensionality Reduction**: Use techniques like PCA to reduce dimensions if the dataset is high-dimensional.

KNN is a simple and effective algorithm for many problems, but its efficiency can be improved using techniques like KD-trees or Ball-trees for faster neighbor searches.

---------------------------------------------------------------------------------------------------------

**Machine Learning Tree-Based Models**

Tree-based models are supervised machine learning algorithms that construct a tree-like structure to make predictions. They can be used for both classification and regression problems.

**Decision Trees**

A decision tree is the simplest tree-based machine learning algorithm. This model allows us to continuously split the dataset based on specific parameters until a final decision is made.

The decision tree will choose a variable to split on first based on a metric called entropy. It will stop splitting when a “pure split” is obtained, i.e., when all the data points belong to a single class.

There are many ways to build a decision tree. The tree needs to find a feature to split on first, second, third, etc. This structure is created based on a metric called information gain. The best possible decision tree is one with the highest information gain.

**Random Forests**

The [**random forest**](https://www.datacamp.com/tutorial/random-forests-classifier-python) model is a tree-based algorithm that helps us mitigate some of the problems that arise when using decision trees, one of which is overfitting. Random forests are created by combining the predictions made by multiple decision tree models and returning a single output.

It does this in two steps:

* **Step 1**: First, the rows and variables of the dataset are randomly sampled with replacement. Multiple decision trees are then created and trained on each data sample.
* **Step 2**: Next, the predictions made by all these decision trees are combined to come up with a single output. For instance, if 3 separate decision trees were trained and 2 of them predicted “Yes” while 1 predicted “No,” then the final outcome of the random forest algorithm would be “Yes.”