Loading Dataset

In machine learning, the dataset is the foundation upon which the model is built. The model uses the data to learn patterns, relationships, and generalizations that can then be used to make predictions or classifications on new, unseen data. The dataset is usually split into two parts: a training set and a test set. The training set is used to train the model and optimize its parameters, while the test set is used to evaluate the performance of the model.

Loading the dataset involves reading the data from its source, whether it be a file, a database, or an API, and then parsing it into a format that can be used by the model. This can involve converting the data into a numerical format, handling missing values, and normalizing or standardizing the data.

Once the dataset is loaded and preprocessed, it can be used to train the model. By training the model on a diverse and representative dataset, the model is able to generalize and make accurate predictions on new data.

In summary, loading and preprocessing the dataset is a crucial step in the machine learning process as it ensures that the data is in the correct format and ready to be used by the model, it helps the model to learn from the data and make predictions or classifications and it also helps to improve the performance of the model by cleaning and transforming the data.

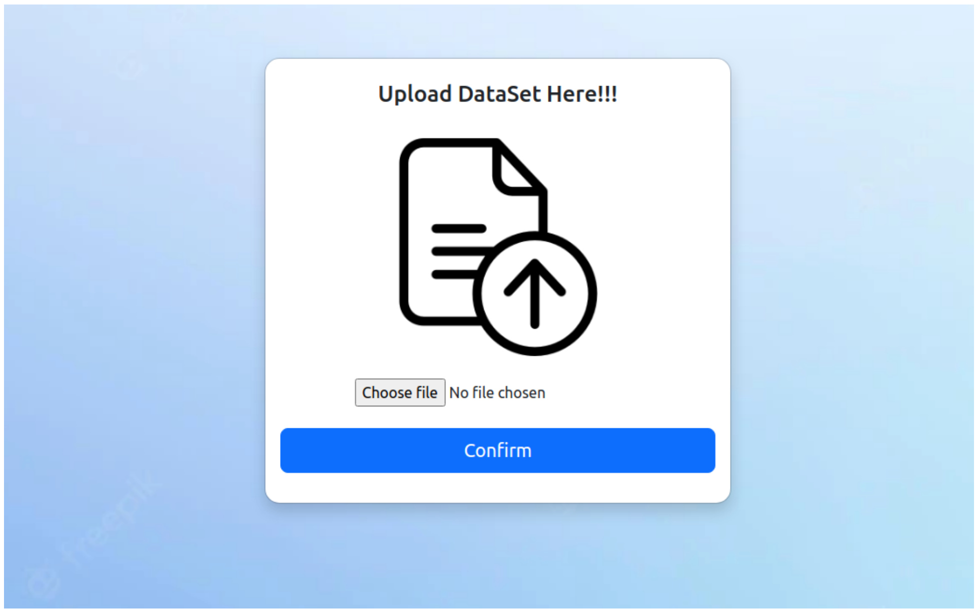


Figure 1.2: Load Dataset

Removing Unwanted Columns

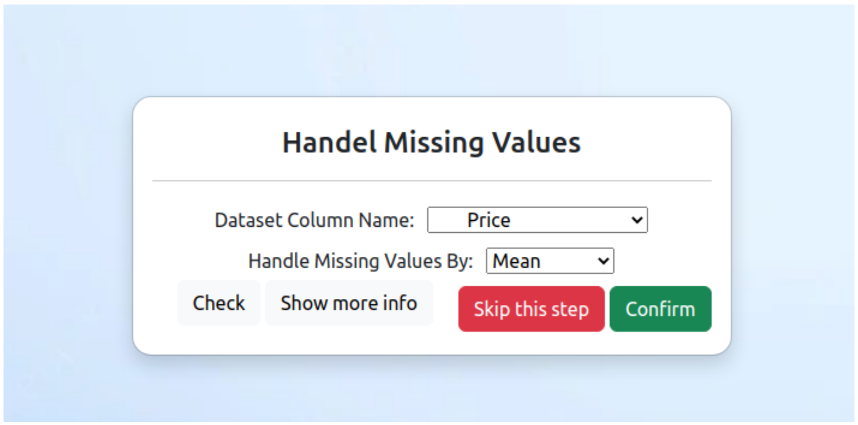
Removing unwanted columns in machine learning is done to improve the performance of the model and to prevent overfitting.

One of the main reasons for removing unwanted columns is that they may contain irrelevant or redundant information that does not contribute to the prediction or classification task. For example, if a dataset contains demographic information such as age and gender, but the task at hand is to predict whether a customer will default on a loan, the age and gender columns may not be relevant and can be removed. Keeping irrelevant columns in the dataset can lead to increased complexity and computational cost, and can also negatively impact the model's performance.

Another reason for removing unwanted columns is that they may be highly correlated with other columns, which can lead to multicollinearity. Multicollinearity occurs when two or more predictor variables in a model are highly correlated, which can cause the model to have unstable coefficients and make it difficult to interpret the results. Removing highly correlated columns can help to eliminate multicollinearity and improve the model's performance.

Removing unwanted columns can also help to prevent overfitting. Overfitting occurs when a model is trained on a dataset with too many features and it starts to fit the noise in the data, rather than the underlying patterns. Removing unwanted columns can help to reduce the number of features and make the model simpler, which can help to prevent overfitting.

In summary, removing unwanted columns in machine learning is done to improve the performance of the model by eliminating irrelevant or redundant information, eliminating multicollinearity, and preventing overfitting. By removing unwanted columns, the model becomes simpler and more interpretable, and is less likely to fit the noise in the data.


Handel Missing Values

Dataset Column Name: Price

Handle Missing Values By: Mean

Figure 1.3: Remove Unwanted Columns

Handle Missing Values

Handling missing values in a dataset is an important step in the data preprocessing process. Missing values can occur for various reasons, such as a malfunctioning data collection device, missing data due to survey nonresponse, or data that is not collected for certain observations. These missing values can have a significant impact on the analysis and the conclusions that can be drawn from the data. Therefore, it is essential to handle missing values properly to ensure that the analysis is accurate and unbiased

There are several techniques that can be used to handle missing values, including:

1.Mean: Replacing missing values with the mean of the column. This is a good option if the column is approximately normally distributed. The mean is calculated by summing up all the values in a column and then dividing by the number of observations. This method is easy to implement and computationally efficient. However, it can be sensitive to outliers and may not be suitable if the data is skewed.

When the data is continuous and the distribution is normal or close to normal, it may be appropriate to impute missing values with the mean of the non-missing observations.

2.Mode: Replacing missing values with the mode of the column. This is a good option if the column is categorical. The mode is the most frequently occurring value in a column. This method is easy to implement and suitable for categorical data. However, it does not work well for continuous variables as it can give unrealistic results.

When the data is categorical, it may be appropriate to impute missing values with the mode (most common value) of the non-missing observations.

3.Median: Replacing missing values with the median of the column. This is a good option if the column is not normally distributed. The median is the middle value of a column when the values are sorted in ascending order. This method is robust to outliers and can handle skewed data. However, it requires more computational resources than mean or mode.

When the data is continuous and the distribution is skewed, it may be appropriate to impute missing values with the median of the non-missing observations.

4.Delete row: Removing the entire row if it contains a missing value. This is a good option if the missing values are not too many. This method is simple to implement, but it can lead to a loss of information if a large number of observations are removed.

It is not a recommended method but it can be used if the dataset is quite large or if the number of missing values is small.

The pandas.isnull().sum() function in the Python library pandas is used to check for and count the number of missing values in a DataFrame or a Series. This function returns the number of missing values in the DataFrame or Series.

Ultimately, the best approach will depend on the specific characteristics of your dataset and the requirements of your analysis. It is also a good idea to try multiple imputation methods and compare the results.

Once you have the count of missing values, you can use it to make decisions about how to handle the missing values. For example, if the number of missing values is low, you may choose to drop the rows with missing values. If the number of missing values is high, you may choose to fill the missing values with the mean or median of the column.

It's important to note that when handling missing values, it's important to consider the specific characteristics of the dataset and the impact of the missing data on the analysis. It's also important to consider that if the missing data is not handled properly, it can lead to biased or incorrect results. Also, one can use more advanced techniques like multiple imputation or Iterative imputation as well to handle missing values.

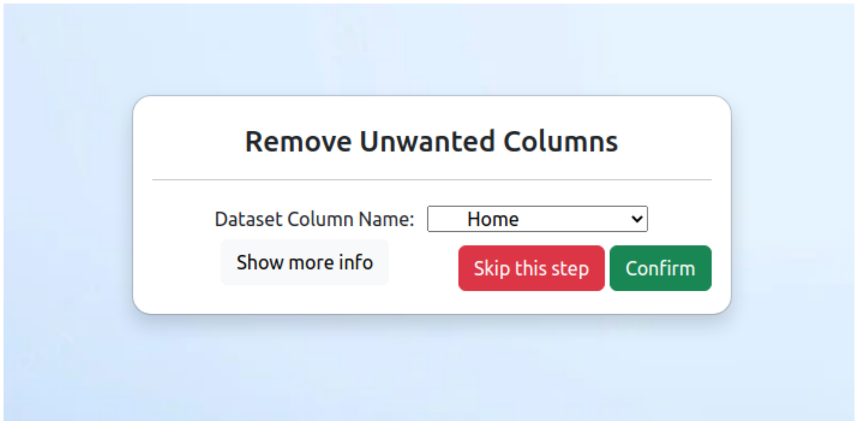


Figure 1.4: Handle Missing Values

Encoding

In machine learning, it is common to use numerical input data, as most models require numerical values as input features. This is because many models are based on mathematical equations that are designed to work with numerical data. For example, linear regression, decision trees and neural networks all require numerical input data.

Categorical variables, which are non-numeric, need to be converted into numerical values in order to use them as input features in these models. This conversion process is known as encoding, and the resulting encoded values can then be used as input features in the model. This conversion process is necessary because the model is not able to understand the categorical variables in their original form.

Additionally, encoding categorical variables can also help improve the performance of the model. For example, one-hot encoding can create new binary columns for each category, which can help increase the amount of information that the model can learn from. And label encoding can help to reduce the dimensionality of the data set as well as provide an ordinal relationship between the categories.

In summary, converting categorical data into numerical values is necessary in machine learning because most models require numerical input data and encoding can help improve the performance of the model.

Label encoding is a process of converting categorical variables, which are non-numeric, into numerical values. This is typically done in order to use these variables as input features in machine learning models, as most models require numerical input data.

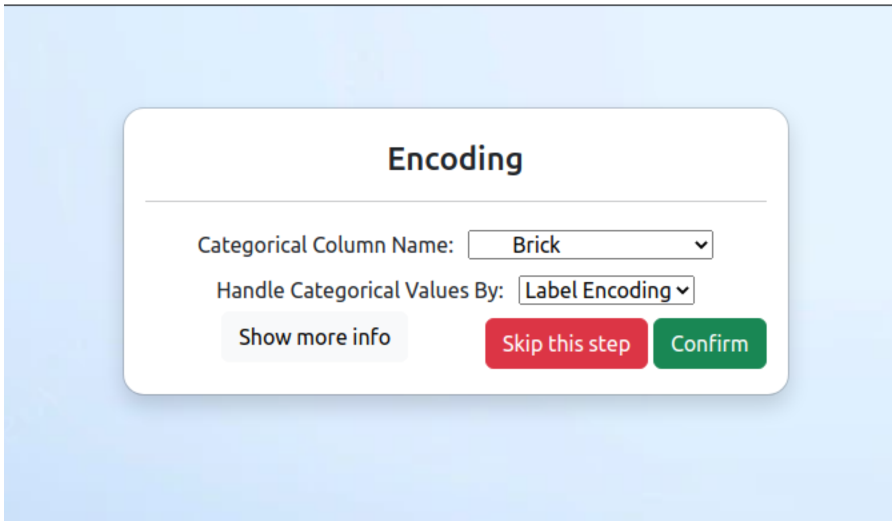
The process works by assigning a unique integer value to each category in the column. For example, if a column has three categories - "red", "green", "blue" - label encoding would convert them to 0, 1, 2 respectively. This is done using the LabelEncoder class in Python's scikit-learn library. The class has a fit and transform method, where the fit method is used to learn the mapping of the categories to integers and the transform method is used to apply the mapping to the data.

This method is useful when the categorical data has a natural ordinal relationship, such as "high", "medium", "low" or "large", "medium", "small". An ordinal relationship implies that the values have a natural ordering and that the distance between the values can be meaningfully compared. For example, in the case of sizes, large is greater than medium and medium is greater than small, and so on. However, it can lead to problems if there is no ordinal relationship, such as "red", "green", "blue" as the encoded values may be misinterpreted as having a ordinal relationship.

One way to avoid this problem is to use one-hot encoding, which creates a binary column for each category. The process works by creating a new binary column for each category and marking it as 1 if the category is present in the original column and 0 otherwise. This is done using the OneHotEncoder class in scikit-learn. The class has a fit and transform method, similar to the LabelEncoder class.

It is important to note that label encoding should only be applied to the categorical variables in your dataset, and not the target variable if you are performing supervised learning. This is because the target variable is usually a categorical variable with no ordinal relationship, and encoding it would introduce an artificial ordinal relationship.

In summary, label encoding is a simple and effective method of converting categorical variables to numerical values, but it should only be used when there is an ordinal relationship between the categories. In other cases, one-hot encoding should be used to avoid introducing an artificial ordinal relationship. It's also important to keep in mind that this method should only be applied to the input features and not the target variable of a supervised learning model.


Encoding

Categorical Column Name: Brick

Handle Categorical Values By: Label Encoding

Figure 1.5: Encoding

Detect and Remove Outliers

Outliers in machine learning are data points that are significantly different from the other data points in the dataset. These data points can have a negative impact on the performance of a machine learning model if they are not handled properly.

Outliers can occur for a variety of reasons, such as data entry errors, measurement errors, or natural variations in the data. They can also occur when there are extreme values in the data that are not representative of the majority of the data.

In simple terms, outliers are data points that are far away from the other data points in the dataset, they can be seen as an anomaly or rare event that can skew the dataset.

There are several techniques to handle outliers in machine learning, such as:

- Removing them from the dataset]
- Transforming the data to make it more robust to outliers]
- Using algorithms that are specifically designed to handle outliers, such as robust regression or robust principal component analysis

It's important to be careful when handling outliers as removing them may not always be the best solution as it can be important information for certain problems like anomaly detection. However, for some problems, it's necessary to handle outliers as they can skew the results of the model.

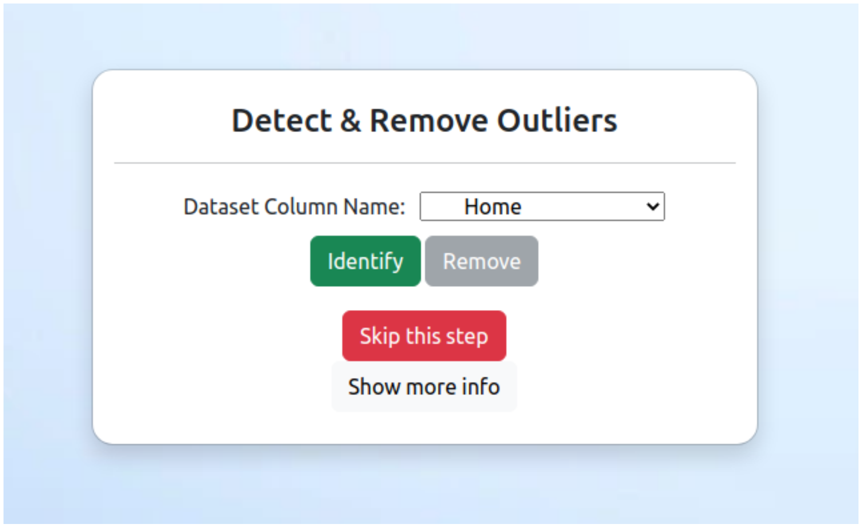


Figure 1.6: Identifying Removing Outliers

A box plot, also known as a whisker plot, is a way to display the distribution of a dataset. It displays a summary of the set of data containing the minimum, first quartile, median, third quartile, and maximum.

- The box in a box plot represents the interquartile range (IQR), which is the difference between the third quartile (Q3) and the first quartile (Q1). The box itself contains the middle 50
- The line in the box represents the median (the middle value) of the data.
- The whiskers extend from the box and display the minimum and maximum values of the data, excluding any outliers.
- Outliers are the data points that are significantly different from the other data points in the dataset. In a box plot, outliers are identified as the data points that fall outside of the whiskers, which are the lines that extend from the box to the minimum and maximum values.

Outliers are calculated as follows:

- Interquartile Range (IQR) = Q3 - Q1
- Lower bound = Q1 - 1.5*IQR
- Upper bound = Q3 + 1.5*IQR
- Any data points that fall outside of the lower and upper bounds are considered outliers.

In simple terms, an outlier is a point that falls more than 1.5*IQR below the first quartile or above the third quartile.

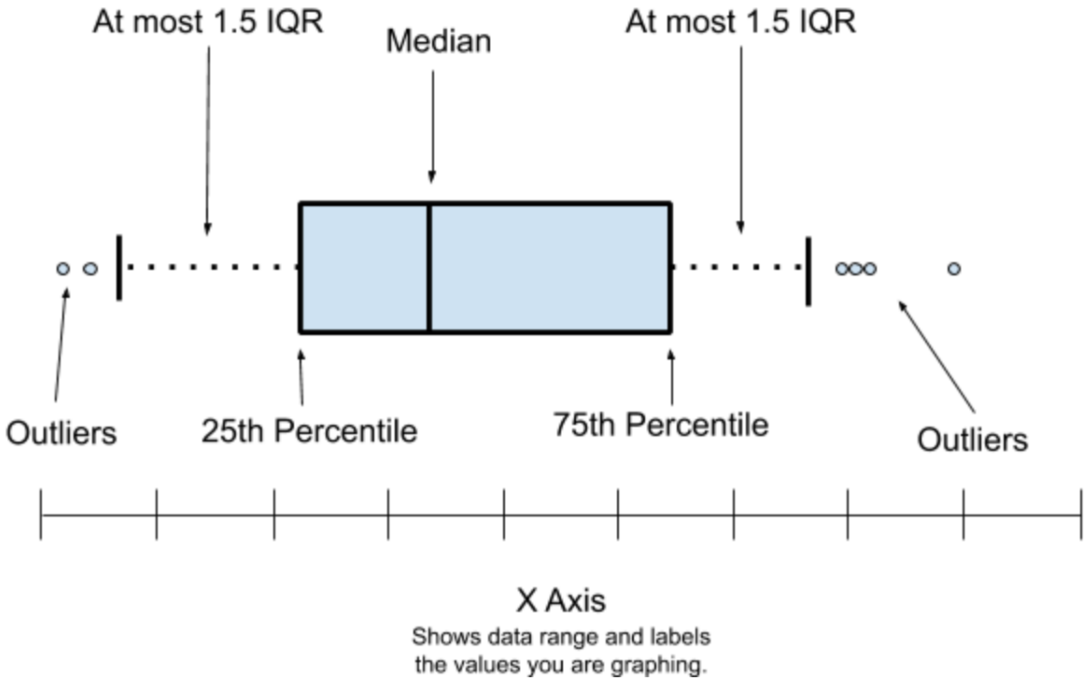


Figure 1.7: Identifying Removing Outliers

Normalization

Normalization is a technique used in machine learning to scale the values of input features to a common range. The goal of normalization is to ensure that all input features are on a similar scale and have similar importance in the model. Without normalization, certain input features with a larger scale could dominate the model, while other features with a smaller scale would have little impact.

There are several techniques for normalization, but the most common ones are min-max normalization and standardization.

Min-max normalization, also known as feature scaling, scales the values of a feature to a specific range, typically between 0 and 1. This is done by subtracting the minimum value of the feature from each value, and then dividing by the range of the feature. The formula for min-max normalization is: **(value - min) / (max - min)**

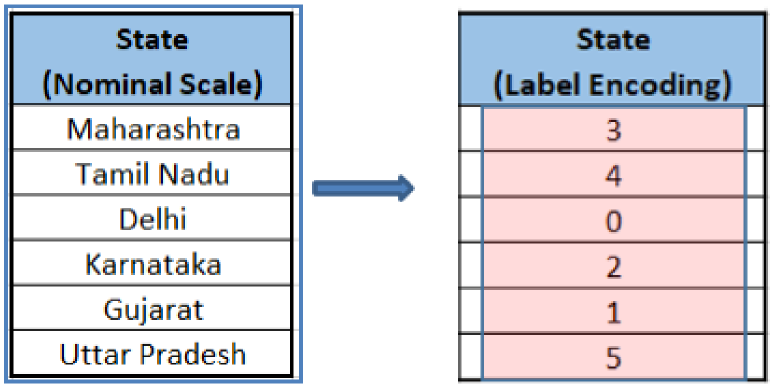
This method is very simple to implement and preserves the original shape of the distribution of the data. However, it can be affected by outliers, as they can cause the data to be compressed into a small range.

Standardization, also known as z-score normalization, scales the values of a feature to have a mean of 0 and a standard deviation of 1. This is done by subtracting the mean of the feature from each value, and then dividing by the standard deviation. The formula for standardization is:**(value - mean) / standard deviation** This method is less affected by outliers, as it uses the mean and standard deviation of the data. However, it can change the shape of the distribution of the data, as it tends to pull the data towards the center.

Another method is the use of logarithms, it can be useful when the data has a large range and a skewed distribution. By taking the logarithm of the data, we can reduce the range and make the distribution more normal. It's important to keep in mind that logarithms of negative or zero values are not defined, so this method is only applicable to positive numbers.

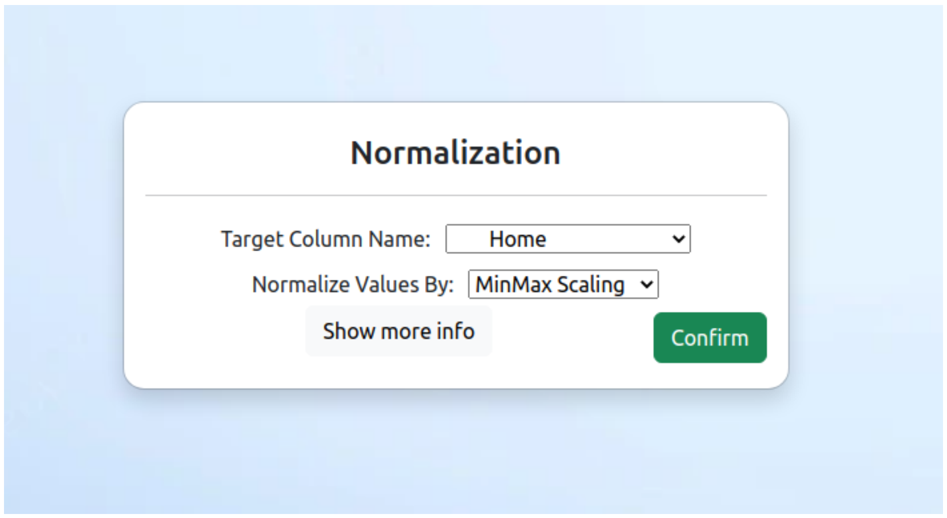
It is important to note that normalization should only be applied to input features, and not the target variable. Additionally, it is generally a good practice to normalize the data before training a model, rather than normalizing the predictions after the model is trained.

In summary, normalization is a technique used in machine learning to scale the values of input features to a common range, to ensure that all input features are on a similar scale and have similar importance in the model. Min-max normalization, standardization and logarithmic scaling are the most commonly used techniques for normalization. However, it's important to note that normalization should only be applied to input features, and not the target variable, and it's generally a good practice to normalize the data before training a model, rather than normalizing the predictions after the model is trained.



State (Nominal Scale)	State (Label Encoding)
Maharashtra	3
Tamil Nadu	4
Delhi	0
Karnataka	2
Gujarat	1
Uttar Pradesh	5

Figure 1.8: Label Encoding



Normalization

Target Column Name: Home

Normalize Values By: MinMax Scaling

Figure 1.9: Normalization

Train Test Split

Train-test split is a method used in machine learning to evaluate the performance of a model. The idea is to split the data into two sets: a training set and a testing set. The model is trained on the training set and then its performance is evaluated on the testing set. This allows us to assess the model's ability to generalize to new data, rather than just memorizing the training data.

The process of train-test split can be done in different ways. One way is to randomly split the data into two sets, for example, 80% for training and 20% for testing. This method is called simple random sampling. It is important to set a random seed to ensure that the split is reproducible.

Another way to split the data is k-fold cross validation. In this method, the data is divided into k subsets, and the model is trained k times, each time using a different subset as the test set. The performance of the model is then averaged over the k runs. This method is more robust than simple random sampling, as it provides a better estimate of the model's performance.

It is important to note that the choice of the test set size and the number of folds can have an impact on the results. A larger test set size will provide a better estimate of the model's performance, but it will also reduce the amount of data available for training. Similarly, increasing the number of folds will provide a better estimate of the model's performance, but it will also increase the computational cost.

It is also important to be careful when choosing the data to split. If the data is not representative of the population, the model will not generalize well. To avoid this, it is recommended to use stratified sampling, which ensures that the proportion of different classes is the same in the training and testing sets.

In summary, train-test split is an important technique used in machine learning to evaluate the performance of a model. There are different ways to split the data, and the choice of the method and the parameters depends on the characteristics of the data and the research questions. It's important to pay attention to the data representation and the size of the sets to get accurate results and avoid overfitting.

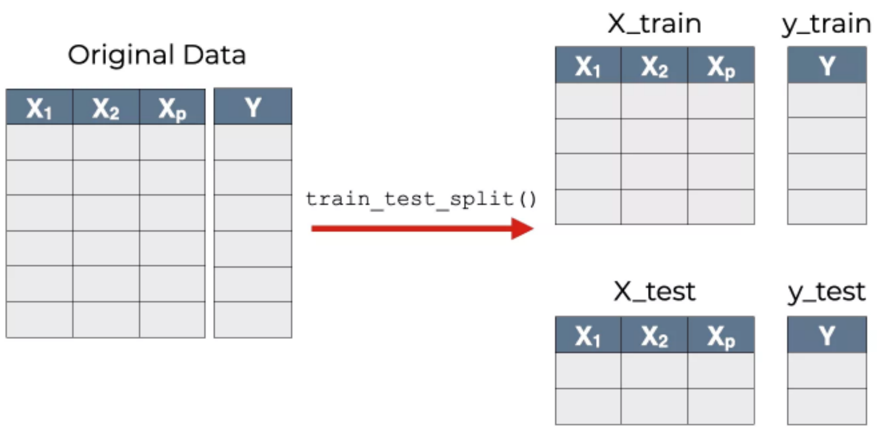


Figure 1.10: Train Test Split

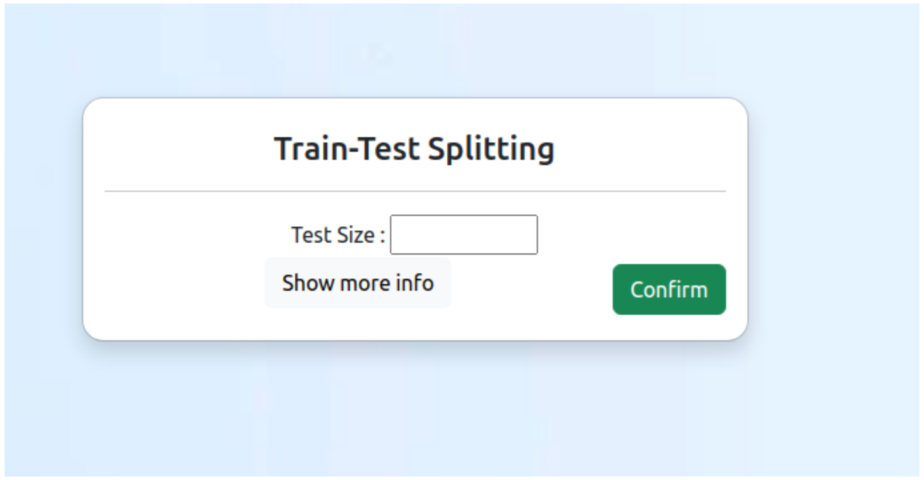


Figure 1.11: Train Test Split

Model Building

A machine learning model is defined as a mathematical representation of the output of the training process. Machine Learning models can be understood as a program that has been trained to find patterns within new data and make predictions. These models are represented as a mathematical function that takes requests in the form of input data, makes predictions on input data, and then provides an output in response. First, these models are trained over a set of data, and then they are provided an algorithm to reason over data, extract the pattern from feed data and learn from those data. Once these models get trained, they can be used to predict the unseen dataset.

Choosing a model and training it are two important steps in machine learning (ML). Here is a more detailed explanation of these steps:

Choosing a model: When choosing a model, there are several factors to consider, including:

- The type of problem you are trying to solve (e.g. classification, regression, clustering, etc.)
- The type of data you have (e.g. continuous, categorical, text, images, etc.)
- The complexity of the problem (e.g. number of features, number of classes, etc.)
- The desired outcome (e.g. high accuracy, interpretability, etc.)

Based on these factors, you will be able to choose a model that is appropriate for your problem. For example, if you are trying to solve a binary classification problem, you may choose a logistic regression or a decision tree.

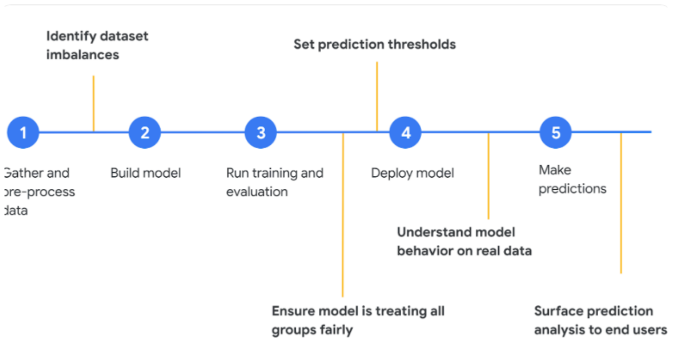

Identify dataset imbalances
Set prediction thresholds
1
2
3
4
5
Gather and pre-process data
Build model
Run training and evaluation
Deploy model
Make predictions
Understand model behavior on real data
Ensure model is treating all groups fairly
Surface prediction analysis to end users

Figure 1.12: Model Building

- **Linear Regression**

Linear regression is a statistical method that is used to model the relationship between a dependent variable (also known as the response variable or target) and one or more independent variables (also known as explanatory variables or predictors). The goal of linear regression is to find the best-fitting line (or hyperplane in multiple dimensions) that describes the relationship between the variables.

The mathematical equation for a simple linear regression model is: y = b0 + b1*x

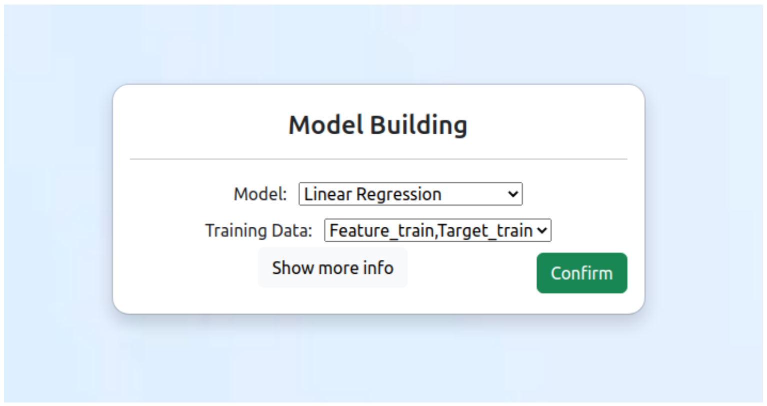
where y is the dependent variable, x is the independent variable, b0 is the y-intercept, and b1 is the slope of the line. In multiple linear regression, the equation is extended to include multiple independent variables:

y = b0 + b1x1 + b2x2 + ... + bn*xn

where x1, x2, ..., xn are the independent variables and b1, b2, ..., bn are the coefficients. The goal is still to find the values of the coefficients that minimize the difference between the predicted values and the actual values.

In both simple and multiple linear regression, the coefficients are typically found using a method called least squares, which minimizes the sum of the squared differences between the predicted values and the actual values. Once the coefficients are found, the model can be used to make predictions about the dependent variable given new values of the independent variables. Linear regression can be used for both simple and multiple regression.

It's important to note that linear regression assumes that the relationship between the dependent variable and the independent variable is linear and that the errors are normally distributed and homoscedastic. Additionally, Linear Regression assumes that there is no multicollinearity among the independent variables.



Model: Linear Regression
Training Data: Feature_train,Target_train

Figure 1.13: Model Building Using Linear Regression

- **Logistic Regression**

Logistic regression is a statistical method that is used to model the relationship between a binary dependent variable and one or more independent variables. The binary dependent variable is also known as the response or outcome variable, and can take on only two values, such as 1 or 0, Yes or No, or True or False. The independent variables, also known as predictor or explanatory variables, are the variables that are used to explain or predict the outcome.

The logistic regression model is a type of generalized linear model (GLM) that uses the logistic function to model the probability of the outcome occurring. The logistic function, also known as the sigmoid function, is an S-shaped curve that maps any input value to a value between 0 and 1. The logistic function has the following form:

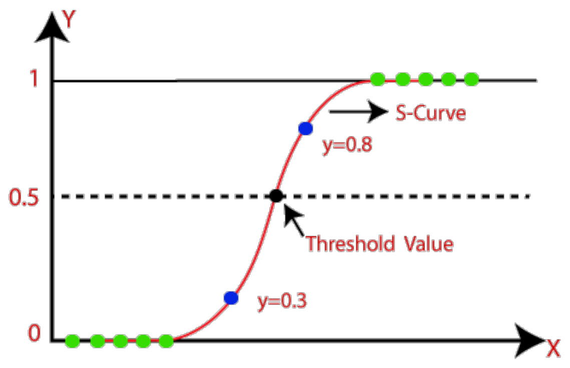
$p(x) = 1 / (1 + e^{-}(B0 + B1x1 + B2x2 + \ldots + Bn * xn))$

where p(x) is the probability of the outcome occurring given the independent variables x1, x2, ..., xn, B0, B1, B2, ..., Bn are the coefficients of the independent variables, and e is the natural logarithm base.

The coefficients in the logistic equation are estimated from the data using maximum likelihood estimation. Maximum likelihood estimation is a method that finds the values of the coefficients that maximize the likelihood of the observed data. The resulting equation can then be used to make predictions about the probability of the outcome occurring for new data.

Once the model is built, it can be evaluated using various metrics such as accuracy, precision, recall, and the confusion matrix. The model can also be fine-tuned using techniques such as regularization or feature selection to improve its performance.

Logistic regression is a powerful tool for building predictive models and making decisions in various fields such as finance, healthcare, marketing, and social sciences, as it can handle both linear and non-linear relationships and it can be easily extended to handle multiple categorical outcome variables (multinomial logistic regression) or multiple binary outcomes (ordinal logistic regression)



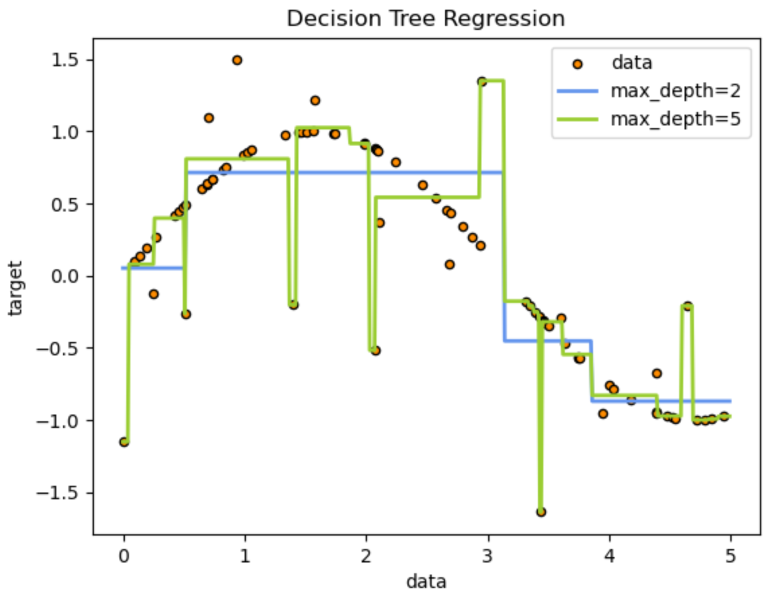
- **Decision Tree Regressor**

A decision tree regressor is a type of supervised machine learning algorithm that is used for predicting a continuous outcome variable based on several input features. The decision tree regressor builds a model in the form of a tree structure, where each internal node represents a feature or attribute, each branch represents a decision or rule based on the value of that feature, and each leaf node represents the predicted value of the target variable.

The goal of decision tree regressor is to approximate the mapping function (f) from input variables to output variables, by partitioning the input space into smaller regions and then make a prediction for each sub-region. This can be done by recursive binary splitting (Dichotomization) where it starts with the entire dataset and then repeatedly splits the dataset on one of the features that gives the best reduction in variance.

The algorithm builds the tree by recursively partitioning the data into subsets based on the values of the input features. The final tree structure is built such that the leaf nodes contain the predicted value of the target variable for the input data present in that region of the input space.

A decision tree regressor algorithm is a powerful tool for both classification and regression problems, it's easy to understand, interpret and visualize. However, it can be prone to overfitting, and it might not be the best option when the dataset has very high dimensionality.



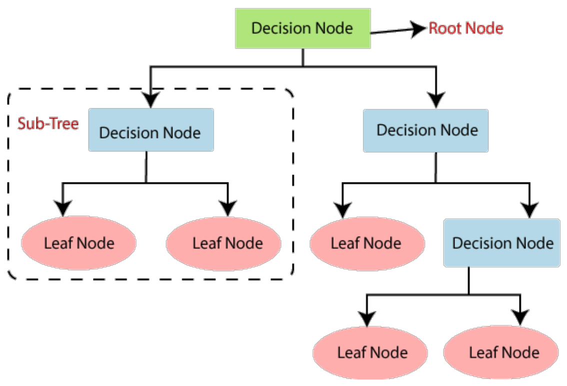
- **Decision Tree Classifier**

A decision tree classifier is a type of machine learning algorithm that can be used for both classification and regression tasks. It is called a ”decision tree” because it creates a tree-like model of decisions and their possible consequences. The tree is constructed by recursively splitting the data into subsets based on the values of the input features. The goal of this process is to create subsets of data that are as homogeneous as possible with respect to the target variable.

At each internal node of the tree, the algorithm selects the feature that provides the most information gain, and splits the data into subsets based on the values of that feature. The information gain is a measure of how much the impurity of the data decreases after the split. The process is repeated recursively on each subset of data until the data at a particular node is pure, or the maximum depth of the tree is reached. The final result is a tree that represents a set of decision rules that can be used to predict the target variable for new instances.

One of the main advantages of decision tree classifiers is that they are easy to understand and interpret. The tree structure provides a clear representation of the decision rules, and the feature importances can be used to identify the most important features for the classification task. Additionally, decision tree classifiers can handle both categorical and numerical features, and they can handle missing data and outliers.

However, decision tree classifiers also have some drawbacks. One of the main problems is that they can easily overfit the data, especially when the tree is deep and has a large number of leaves. This can lead to poor generalization performance on new data. To avoid overfitting, various techniques can be used such as pruning, limiting the maximum depth of the tree, or using ensemble methods like Random Forest.Another drawback is that decision tree classifiers can be sensitive to small changes in the data, and this can lead to instability in the tree structure. This is known as the problem of high variance. To address this problem, various methods can be used such as bagging, boosting, or randomization.



- **Random Forest Regressor**

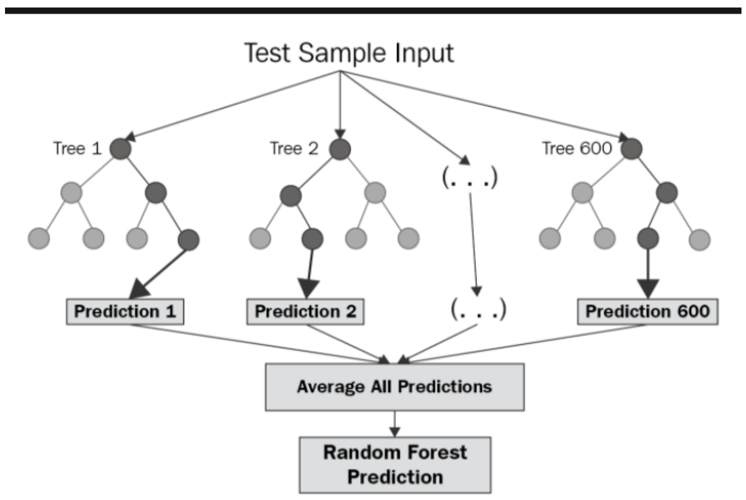
A random forest classifier is an ensemble machine learning algorithm that is built on top of decision trees. The main idea behind an ensemble method is to combine the predictions of multiple models to improve the overall performance. In the case of a random forest classifier, multiple decision trees are trained on different subsets of the data, and the final prediction is made by averaging the predictions of all the trees.

The process of creating a random forest classifier starts by selecting a random subset of the data, known as the bootstrap sample, to train each decision tree. The bootstrap sample is created by randomly selecting data points with replacement. This ensures that each decision tree is trained on a slightly different subset of the data, and this helps to reduce the correlation between the trees and improve the overall performance.

Another key aspect of random forest classifiers is the random selection of features at each split. Instead of using all the features to find the best split, a random subset of the features is used. This helps to decorrelate the trees and further reduce overfitting.

The final prediction of a random forest classifier is made by averaging the predictions of all the decision trees. In the case of a classification task, the majority vote of all the trees is used to make the final prediction. In the case of a regression task, the average of all the predictions is used.

One of the main advantages of random forest classifiers is their high accuracy and good generalization performance. They are able to handle high dimensional data and are robust to outliers and noise. They also have the ability to handle categorical and numerical features as well as missing data. Additionally, they provide feature importance, which helps in identifying the most important features for the classification task.



- **Random Forest Classifier**

Random Forest classifier is an ensemble machine learning algorithm that combines multiple decision trees to improve the overall performance. The algorithm is based on the idea of creating multiple decision trees, also known as "forest", and then averaging their results to make a final prediction. This approach helps to reduce the variance and bias of the model, and improves the generalization performance on new data.

The process of creating a random forest classifier starts by selecting a random subset of the data, known as the bootstrap sample, to train each decision tree. The bootstrap sample is created by randomly selecting data points with replacement. This ensures that each decision tree is trained on a slightly different subset of the data, and this helps to reduce the correlation between the trees and improve the overall performance. This process is known as Bagging.

Another key aspect of random forest classifiers is the random selection of features at each split. Instead of using all the features to find the best split, a random subset of the features is used. This helps to decorrelate the trees and further reduce overfitting. This process is known as Random Subspace.

The final prediction of a random forest classifier is made by averaging the predictions of all the decision trees. In the case of a classification task, the majority vote of all the trees is used to make the final prediction. In the case of a regression task, the average of all the predictions is used.

Random Forest algorithm can be used for both classification and regression problems. In classification, the majority vote of all the trees is used to make the final prediction. In regression, the average of all the predictions is used. Random Forest algorithm can be used to find the most important features in a dataset.

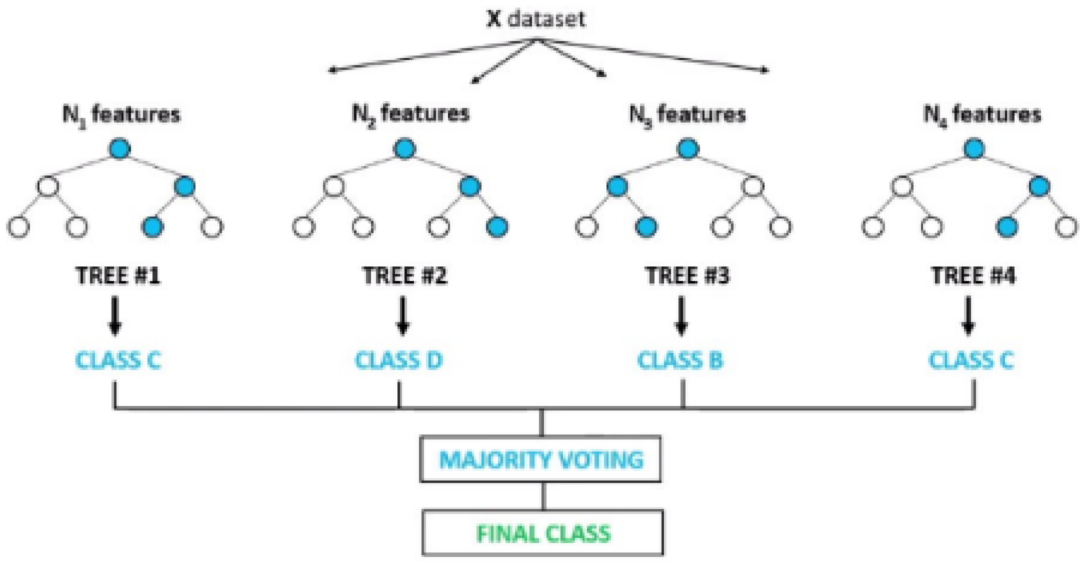
One of the main advantages of random forest classifiers is their ability to handle high dimensional data and are robust to outliers and noise. They also have the ability to handle categorical and numerical features as well as missing data. Additionally, they provide feature importance, which helps in identifying the most important features for the classification task.

Random Forest algorithm also has a built-in feature to handle missing data. When the data has missing values, the algorithm will take care of them by averaging the result of the tree that was generated by considering the missing value and the tree that was generated by ignoring it.

Another important advantage of Random Forest is that it is not prone to overfitting. Decision tree is prone to overfitting, but random forest reduces overfitting by creating multiple decision trees and then averaging their results.

However, random forest classifiers also have some drawbacks. They can be computationally expensive, especially when the number of trees is large, or the data is very high dimensional.

Additionally, the model can be hard to interpret, as the final decision is based on the average of multiple decision trees, which can make it difficult to understand the underlying logic behind the predictions.



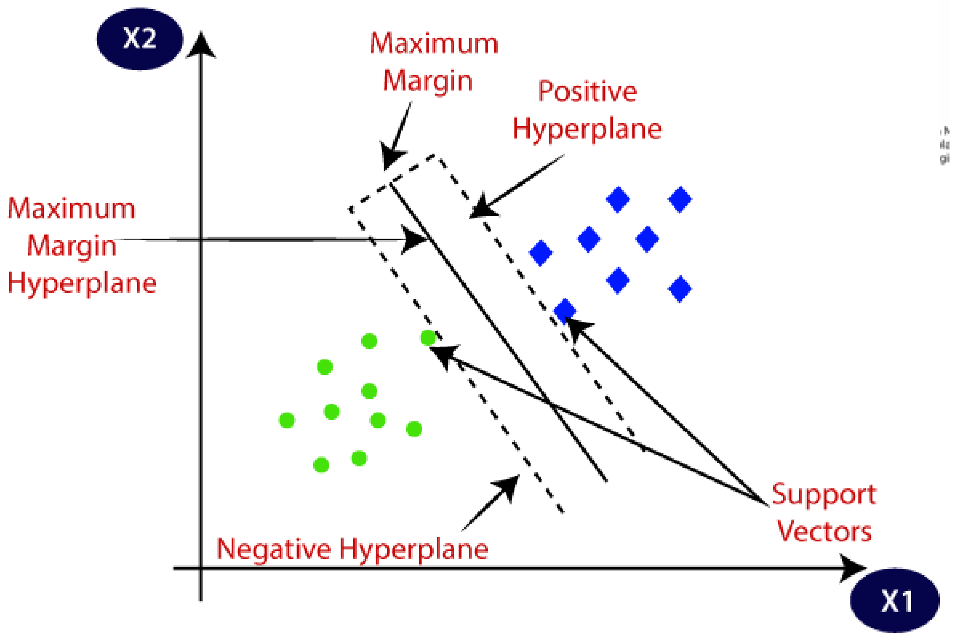
Support Vector Machine

Support Vector Machines (SVMs) are a type of supervised learning algorithm that can be used for classification and regression tasks. The main idea behind SVMs is to find a hyperplane (a line or a plane in a high-dimensional space) that best separates the data into different classes. The hyperplane is chosen in such a way that it maximizes the margin, which is the distance between the hyperplane and the closest data points from each class, also known as support vectors.

SVMs are particularly useful when the data is not linearly separable, meaning that a straight line or a plane cannot be used to separate the classes. In such cases, SVMs use a technique called the kernel trick, which maps the data into a higher-dimensional space where it becomes linearly separable. Commonly used kernel functions include the polynomial kernel and the radial basis function (RBF) kernel.

SVMs have several advantages, including high accuracy, the ability to handle high-dimensional data, and the ability to work well with small datasets. However, they can be sensitive to the choice of kernel function and the values of the parameters, and they can also be computationally expensive when working with large datasets.

Overall, Support Vector Machines (SVMs) is a powerful and versatile algorithm that can be used for a wide range of classification and regression tasks. It can be useful when the data is not linearly separable, and it is a great alternative to other algorithm like Logistic Regression, Decision Trees, Random Forest etc.



Model Evaluation

Confusion matrix, precision, recall, and accuracy are all commonly used metrics for evaluating the performance of a machine learning model.

A confusion matrix is a table that is used to define the performance of a classification algorithm. It compares the predicted values with the true values and shows the number of correct and incorrect predictions. The entries in the matrix are typically represented as counts or percentages. The diagonal elements represent the number of correct predictions, while the off-diagonal elements represent the number of incorrect predictions. Confusion matrices are useful for understanding where the model is making errors and can help to identify specific areas for improvement.

Model evaluation is a crucial step in the machine learning process. It helps to determine the effectiveness of a model in making predictions. One common way to evaluate a classification model is through the use of a confusion matrix, which illustrates the number of true positive, true negative, false positive, and false negative predictions made by the model.

Precision is the number of true positive predictions divided by the number of true positive and false positive predictions. It measures the ability of the model to not label a negative sample as positive. High precision means that the model is not producing many false positives. It is particularly useful when the cost of false positives is high.

Precision = (True Positives) / (True Positives + False Positives)

Recall is the number of true positive predictions divided by the number of true positive and false negative predictions. It measures the ability of the model to find all the positive samples. High recall means that the model is not producing many false negatives. It is particularly useful when the cost of false negatives is high.

Recall = (True Positives) / (True Positives + False Negatives)

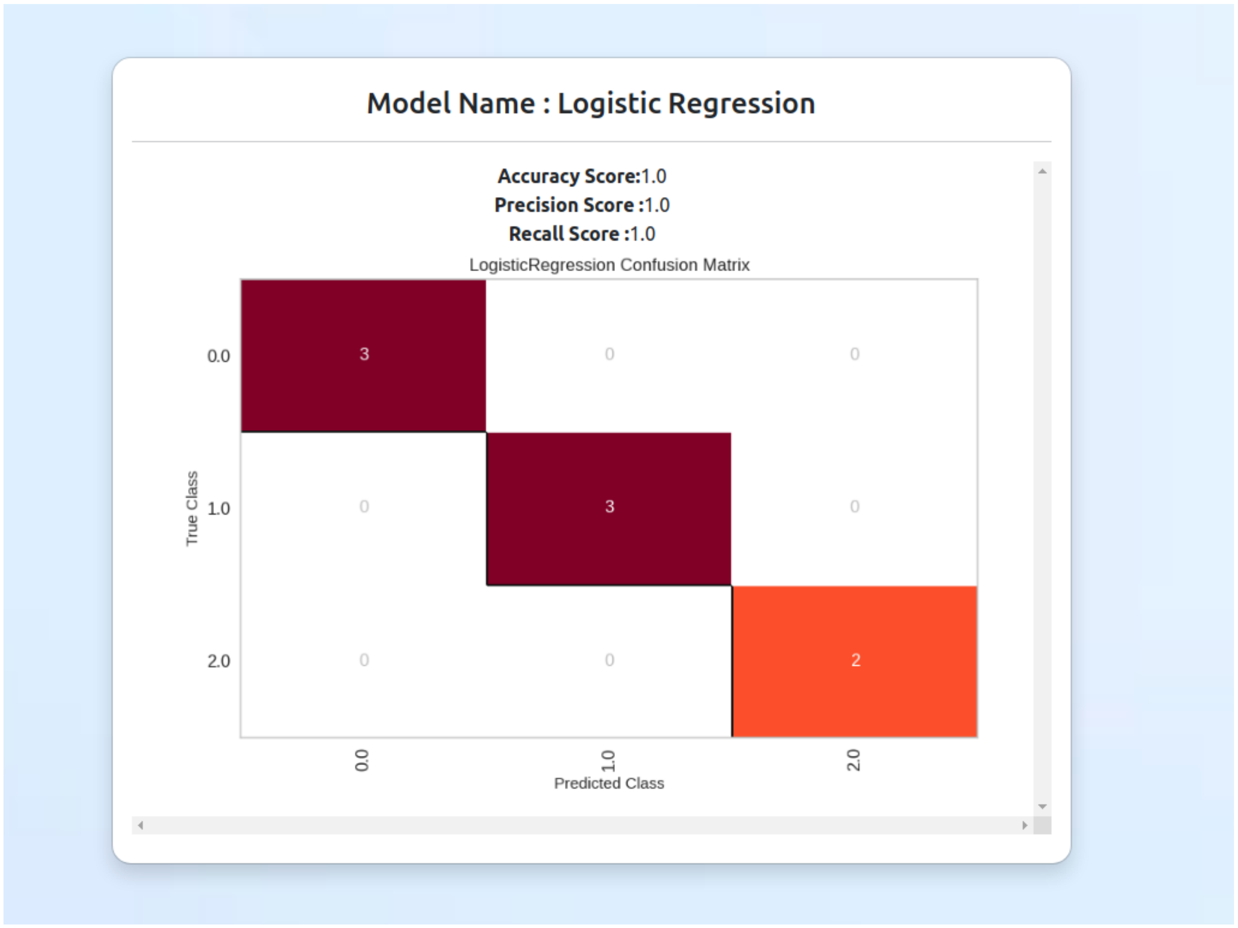
Accuracy is the number of correct predictions divided by the total number of predictions. It is a simple measure of how well the model is able to predict the class labels. It is commonly used as an overall measure of model performance but can be misleading in cases where the class distribution is imbalanced.

Accuracy = (True Positives + True Negatives) / (Total Predictions)

In addition to these three metrics, other evaluation metrics such as F1-score, AUC-ROC and Log-loss can also be used depending on the requirements of the problem and the nature of the data.

When evaluating the performance of a model, it is important to keep in mind the specific context and goals of the project. In some cases, a high recall may be more important than a high precision, while in other cases, a high precision may be more important than a high recall. It's also important to consider the trade-offs between these different metrics. A model with high precision may have low recall, and vice versa.

In conclusion, Confusion matrix, precision, recall, and accuracy are all important measures of a model's performance, and the choice of which metric to use may depend on the specific application and context. It's important to evaluate a model using multiple metrics and understand the trade-offs between them in order to make an informed decision about a model's performance.



Model Name : Logistic Regression

Accuracy Score:1.0
Precision Score :1.0
Recall Score :1.0

Figure 1.14: Model Evaluation

Model Prediction

Model prediction is the process of using a trained machine learning model to make predictions on new, unseen data. The model takes input data and applies the learned patterns and relationships from the training data to make predictions about the output variable.

For example, in a binary classification problem, the model would predict one of two possible outcomes, such as "positive" or "negative". In a regression problem, the model would predict a numerical value, such as a price or a quantity.

The process of making predictions with a machine learning model typically involves the following steps:

Inputting the new data into the model Running the data through the model's trained parameters Generating a prediction based on the model's learned patterns and relationships It is important to note that the accuracy of a model's predictions is highly dependent on the quality and relevance of the training data. A model that has been trained on a large and diverse dataset is likely to make more accurate predictions than a model that has been trained on a small or biased dataset.

Additionally, in real-world scenarios, the model is built with certain assumptions and it may not perform well on unseen data that doesn't fit those assumptions. That's why it's important to evaluate the model's performance on unseen data and update the model if necessary.

In summary, model prediction is the process of using a trained machine learning model to make predictions on new, unseen data by inputting it into the model, running it through the model's trained parameters, and generating a prediction based on the model's learned patterns and relationships. The accuracy of predictions is highly dependent on the quality and relevance of the training data.

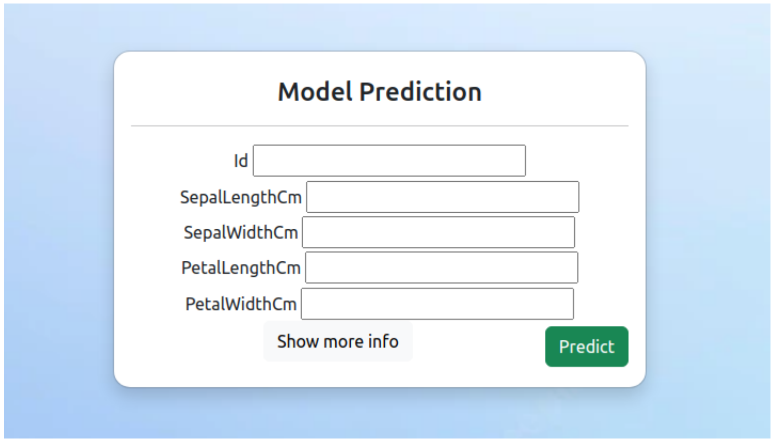


Figure 1.15: Model Evaluation