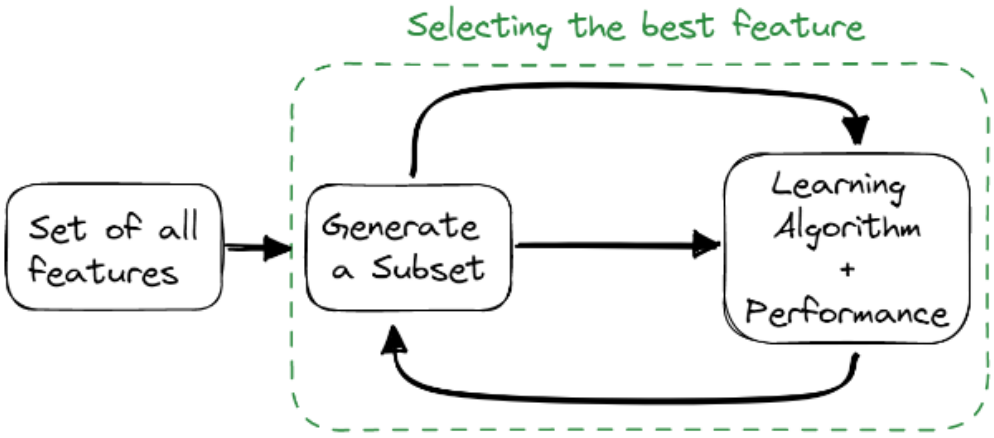
Feature selection

it is the process of picking the most relevant and informative features from a dataset to improve model performance and interpretability. It involves methods like correlation, mutual information, Recursive Feature Elimination, and LASSO. The aim is to keep important features while discarding irrelevant ones, resulting in better and more efficient machine learning models.

what is Embedded Methods?

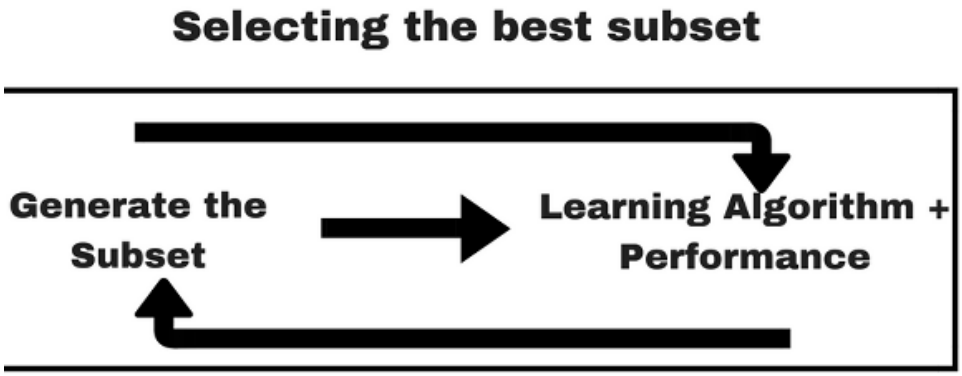
mbedded methods are feature selection techniques which perform feature selection as part of the model construction process. They are called embedded methods because feature selection is embedded within the construction of the machine learning model. These methods aim to solve the limitations of filter and wrapper methods by including the interactions of the features while also being more computationally efficient.



1. **LASSO (Least Absolute Shrinkage and Selection Operator)**: LASSO is a linear regression method that adds a penalty term proportional to the absolute values of the coefficients to the ordinary least squares (OLS) cost function. This penalty encourages many coefficients to become exactly zero, effectively performing feature selection. The LASSO regularization term is given by α * ∑|β|, where α is the regularization strength and β represents the model's coefficients.

2. **Ridge Regression (L2 Regularization)**: While not strictly a feature selection method, ridge regression is an embedded method that can help mitigate the impact of irrelevant or collinear features. It adds an L2 regularization term proportional to the squared magnitudes of the coefficients to the OLS cost function. The regularization term is given by α * ∑β^2, where α is the regularization strength and β represents the model's coefficients. Ridge regression tends to shrink less important features' coefficients towards zero, effectively reducing their impact on the model.

Both LASSO and ridge regression control model complexity through regularization, with LASSO being more aggressive in setting coefficients to zero, leading to explicit feature selection. The choice between LASSO and ridge regression depends on the problem and the amount of feature sparsity desired.



Advantages of embedded methods in linear models:

1. Feature selection and model training are performed simultaneously, which can lead to more accurate models.
2. LASSO, in particular, can yield a sparse model, which is desirable when dealing with high-dimensional datasets with many irrelevant features.
3. These methods handle multicollinearity effectively, making the models more stable and interpretable.

Limitations:

1. Embedded methods may not be suitable for datasets with a small number of samples or a high signal-to-noise ratio, as they may suffer from overfitting.
2. The effectiveness of embedded methods depends on the appropriate choice of regularization strength (α). Tuning this hyperparameter is crucial to achieve the best performance.

Overall, embedded methods like LASSO and ridge regression offer valuable feature selection capabilities and regularization for linear models, leading to more interpretable and efficient models with reduced risk of overfitting.

Feature Selection using LASSO

In [1]:

```
from sklearn import datasets
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes

df.head()
```

Out[1]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [2]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc[:,-1],test_size=0.2,random_state=1
```

In [3]:

```
X_train.shape
```

Out[3]:

```
(614, 8)
```

In [4]:

```
X_train.head()
```

Out[4]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
663	9	145	80	46	130	37.9	0.637	40
712	10	129	62	36	0	41.2	0.441	38
161	7	102	74	40	105	37.2	0.204	45
509	8	120	78	0	0	25.0	0.409	64
305	2	120	76	37	105	39.7	0.215	29

In [5]:

```
from sklearn.preprocessing import StandardScaler

cols = X_train.columns

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

X_train_scaled = pd.DataFrame(X_train_scaled,columns=cols)
X_test_scaled = pd.DataFrame(X_test_scaled,columns=cols)
```

In [6]:

```
X_train_scaled.head()
```

Out[6]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
0	1.516591	0.750524	0.564756	1.652015	0.437496	0.795825	0.529526	0.567932
1	1.812018	0.244752	-0.347904	1.020973	-0.678474	1.228654	-0.069689	0.398450
2	0.925736	-0.608739	0.260536	1.273390	0.222886	0.704013	-0.794249	0.991638
3	1.221164	-0.039745	0.463350	-1.250779	-0.678474	-0.896139	-0.167519	2.601722
4	-0.551400	-0.039745	0.361943	1.084077	0.222886	1.031914	-0.760619	-0.364222

In [8]:

```
from sklearn.linear_model import Lasso

lasso = Lasso(alpha=0.01)

lasso.fit(X_train_scaled, y_train)
```

Out[8]:

```
Lasso
Lasso(alpha=0.01)
```

In [9]:

```
lasso.coef_
```

Out[9]:

```
array([ 6.26290474e-02,  1.85968700e-01, -3.50258983e-02, -0.00000000e+00,
       -1.62558777e-04,  8.51781764e-02,  2.83810683e-02,  1.84611338e-02])
```

In [10]:

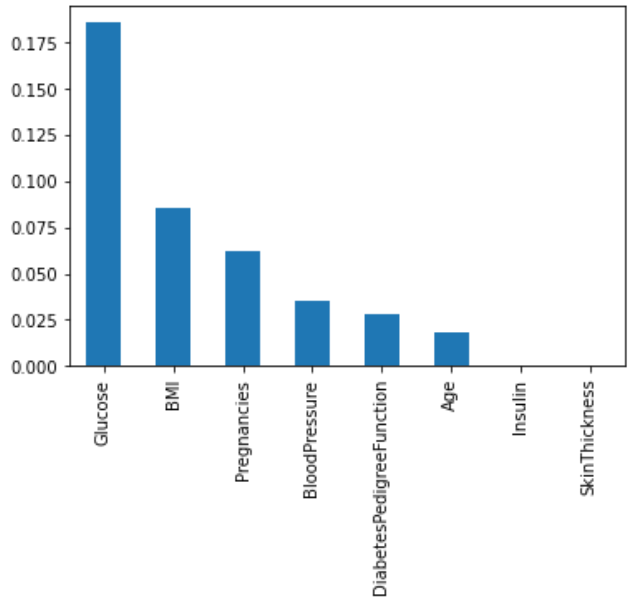
```
import matplotlib.pyplot as plt
import numpy as np

x = pd.Series(np.abs(lasso.coef_),index=cols)

x.sort_values(ascending=False).plot(kind='bar')
```

Out[10]:

```
<AxesSubplot:>
```



Tree Based Models

In the context of feature selection, embedded techniques using Decision Trees are methods that leverage decision tree-based algorithms to perform feature selection within the model training process. These methods automatically select important features as part of the tree-building process, making them embedded within the model itself. Two popular approaches are:

1. Random Forest Feature Importance: Random Forest is an ensemble learning algorithm based on decision trees. During the construction of each tree in the forest, the algorithm evaluates different features to find the best splits. By aggregating the results across all trees, the algorithm computes the feature importance scores. Features that consistently contribute to better splits are considered more important, while features with low importance may be less relevant for the target variable.
2. Gradient Boosting Feature Importance: Gradient Boosting is another ensemble learning algorithm that constructs decision trees sequentially, where each tree aims to correct the mistakes of the previous one. The algorithm computes feature importance based on how much each feature contributes to reducing the overall loss during the boosting process.

Advantages of using Decision Tree-based embedded feature selection techniques:

1. Interactions between features are captured: Decision trees can naturally capture interactions between features, which makes them effective in identifying important feature combinations that may be missed by other methods.
2. No need for separate feature selection step: These techniques do not require a pre-processing step to select features, simplifying the overall modeling pipeline.
3. Robustness to different data types: Decision trees can handle both numerical and categorical features without requiring extensive data transformations.

Limitations:

1. May not work well with high-dimensional data: Decision tree-based methods can be computationally expensive and less effective when dealing with datasets with a large number of features.
2. Model-specific importance: The feature importance scores may vary depending on the algorithm used (e.g., Random Forest vs. Gradient Boosting), which could affect the selection of important features.
3. Prone to overfitting: Decision trees are susceptible to overfitting, and thus the feature importance scores can be biased when applied to small datasets.

To maximize the effectiveness of these embedded techniques, it is essential to consider hyperparameter tuning and cross-validation during the model training process. Additionally, one can use domain knowledge to interpret the importance scores and further refine the feature selection process if needed.

In [11]:

```
from sklearn import datasets
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes

df.head()
```

Out[11]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age	Outcome
0	6	148	72	35	0	33.6	0.627	50	1
1	1	85	66	29	0	26.6	0.351	31	0
2	8	183	64	0	0	23.3	0.672	32	1
3	1	89	66	23	94	28.1	0.167	21	0
4	0	137	40	35	168	43.1	2.288	33	1

In [12]:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc[:,-1],test_size=0.2,random_state=1
```

In [13]:

```
X_train.head()
```

Out[13]:

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunction	Age
663	9	145	80	46	130	37.9	0.637	40
712	10	129	62	36	0	41.2	0.441	38
161	7	102	74	40	105	37.2	0.204	45
509	8	120	78	0	0	25.0	0.409	64
305	2	120	76	37	105	39.7	0.215	29

In [14]:

```
from pandas.core.common import random_state
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier

dt = DecisionTreeClassifier()
rf = RandomForestClassifier()

#dt.fit(X_train,y_train)
rf.fit(X_train,y_train)
```

Out[14]:

RandomForestClassifier

RandomForestClassifier()

In [15]:

```
# This function accesses the 'feature_importances_' attribute of the 'rf' object
rf.feature_importances_
```

Out[15]:

```
array([0.08905321, 0.24159796, 0.08823792, 0.0760338 , 0.07973243,
       0.16332996, 0.12567651, 0.13633822])
```

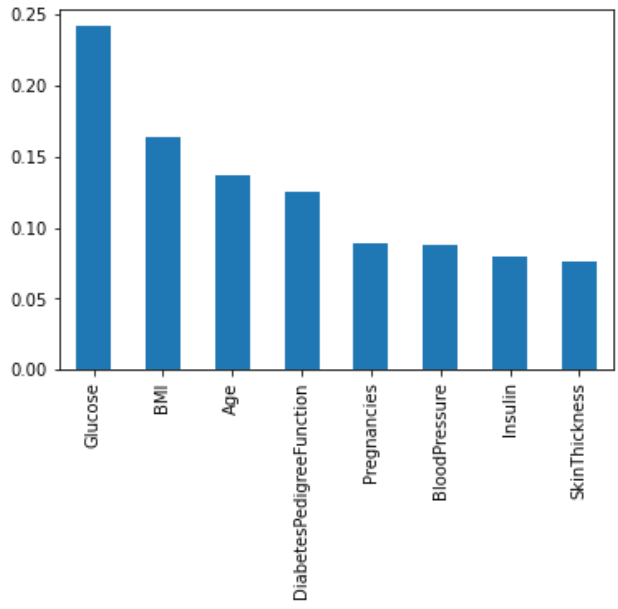
In [16]:

```
x = pd.Series(np.abs(rf.feature_importances_),index=cols)

x.sort_values(ascending=False).plot(kind='bar')
```

Out[16]:

<AxesSubplot:>



SelectFromModel

SelectFromModel is a feature selection technique in scikit-learn, a popular machine learning library in Python, which is considered an embedded method. It works by selecting features based on their importance scores computed during the training of an underlying model. This method is commonly used with tree-based models like Random Forest and Gradient Boosting, but it can also be applied with other models that provide feature importance scores.

The basic idea behind `SelectFromModel` is to set a threshold for feature importance, and only retain the features that have importance scores greater than or equal to that threshold. By doing so, it automatically performs feature selection as part of the model training process.

Here's how SelectFromModel works:

1. Train a machine learning model (e.g., Random Forest, Gradient Boosting, etc.) on the given dataset.
2. Retrieve the feature importance scores from the trained model.
3. Set a threshold for feature importance. Features with importance scores greater than or equal to this threshold will be selected.
4. Retain only the selected features for further analysis or model training.

Using `SelectFromModel` can be helpful in reducing the number of features and improving model efficiency, especially when dealing with high-dimensional datasets. It can also help to identify the most relevant features for the target variable.

In [17]:

```
from sklearn import datasets
import pandas as pd

df = pd.read_csv('https://raw.githubusercontent.com/npradaschnor/Pima-Indians-Diabetes-Dataset/master/diabetes

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(df.iloc[:,0:-1],df.iloc[:,-1],test_size=0.2,random_state=1
```

In [18]:

```
from sklearn.tree import DecisionTreeClassifier

model = DecisionTreeClassifier()
```

In [23]:

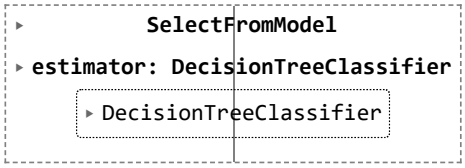
```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(model, threshold='mean')
```

In [24]:

```
sfm.fit(X_train, y_train)
```

Out[24]:



In [25]:

```
sfm.get_support(indices=True)
```

Out[25]:

```
array([1, 5, 7], dtype=int64)
```

In [26]:

```
X_train_trans = sfm.transform(X_train)

X_train_trans = pd.DataFrame(X_train_trans,columns=sfm.feature_names_in_[sfm.get_support(indices=True)])

X_train_trans
```

Out[26]:

	Glucose	BMI	Age
0	145.0	37.9	40.0
1	129.0	41.2	38.0
2	102.0	37.2	45.0
3	120.0	25.0	64.0
4	120.0	39.7	29.0
...	...	...	...
609	157.0	39.4	30.0
610	187.0	33.9	34.0
611	126.0	43.4	42.0
612	171.0	43.6	26.0
613	102.0	32.9	46.0

614 rows × 3 columns

Advantages and Disadvantage

Advantages:

- **Performance**: They are generally more accurate than filter methods since they take the interactions between features into account.
- **Efficiency**: They are more computationally efficient than wrapper methods since they fit the model only once.
- **Less Prone to Overfitting**: They introduce some form of regularization, which helps to avoid overfitting. For example, Lasso and Ridge regression add a penalty to the loss function, shrinking some coefficients to zero

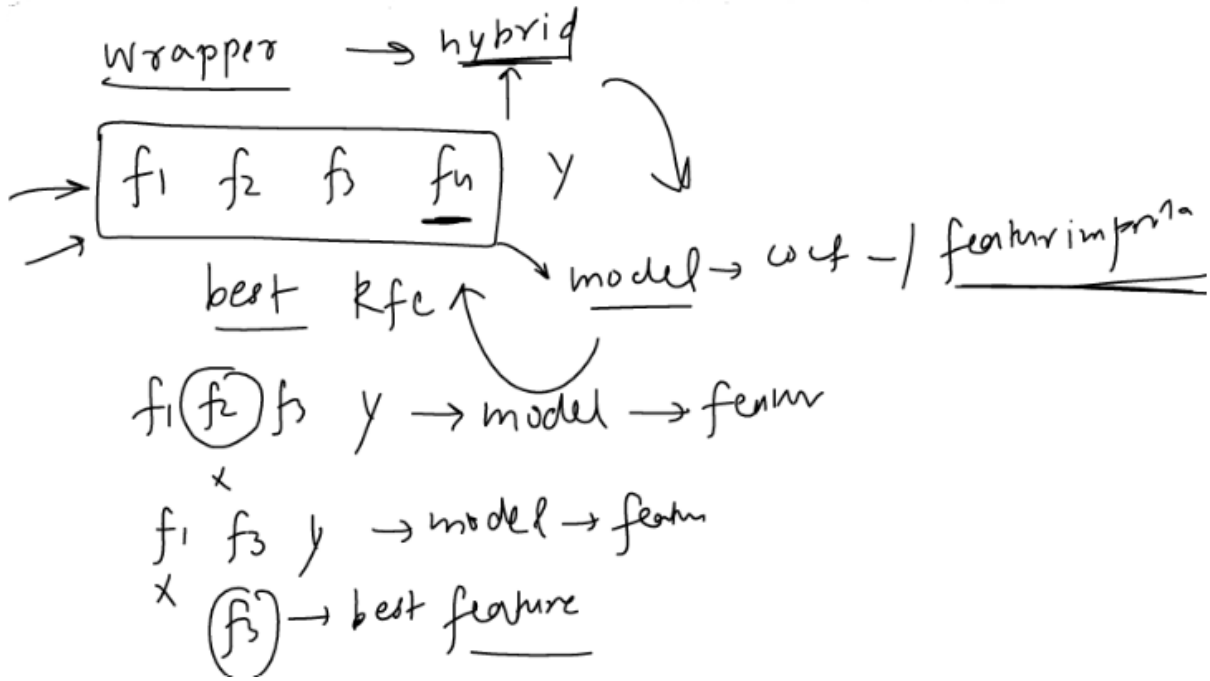
Disadvantages:

- **Model Specific**: Since they are tied to a specific machine learning model, the selected features are not necessarily optimal for other models.
- **Complexity**: They can be more complex and harder to interpret than filter methods. For example, understanding why Lasso shrinks some coefficients to zero and not others can be non-trivial.
- **Tuning Required**: They often have hyperparameters that need to be tuned, like the regularization strength in Lasso and Ridge regression.
- **Stability**: Depending on the model and the data, small changes in the data can result in different sets of selected features. This is especially true for models that can fit complex decision boundaries, like decision trees

Wrapper Methods

Recursive Feature Elimination

Recursive Feature Elimination (RFE) is a feature selection technique used to identify the most relevant features in a dataset. It works by recursively fitting a model and removing the least important features until the desired number of features is reached or a predefined stopping criterion is met.



Here's how Recursive Feature Elimination works:

1. Choose a machine learning model: Select a machine learning model that can rank the importance of features, such as a linear model, support vector machine, or decision tree-based model.
2. Train the model: Fit the chosen model on the entire dataset, using all available features.
3. Rank the features: Use the feature importances (coefficients, feature weights, or other importance metrics) provided by the model to rank the features in terms of their importance for the task at hand.
4. Eliminate the least important feature: Remove the feature with the lowest importance ranking from the dataset.
5. Re-train the model: Fit the model again, but this time using the reduced feature set (without the least important feature).
6. Repeat the process: Iterate the elimination process, ranking features again, eliminating the least important one, and retraining the model, until the desired number of features is reached or a stopping criterion (e.g., a specific number of features or a predefined threshold for performance) is met.

The number of features to be retained after the RFE process can be set based on domain knowledge or determined through cross-validation.

RFE is an effective feature selection method as it takes into account the interactions between features and captures the relative importance of each feature based on the model's performance. It helps to create simpler and more interpretable models, reduces the risk of overfitting, and can lead to improved generalization to new data.

step by step in detailed

In [28]:

```
import pandas as pd
import numpy as np
```

In [29]:

```
df = pd.read_csv('https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a9b0a5d22642a06d3d5b9bc
df.head()
```

Out[29]:

	sepal_length	sepal_width	petal_length	petal_width	species
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa

In [30]:

```
X = df.iloc[:,0:-1]
y = df.iloc[:,-1]
```

In [31]:

```
X.head()
```

Out[31]:

	sepal_length	sepal_width	petal_length	petal_width
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [32]:

```
from sklearn.ensemble import RandomForestClassifier

rf = RandomForestClassifier()

rf.fit(X,y)
```

Out[32]:

```
RandomForestClassifier
RandomForestClassifier()
```

In [33]:

```
rf.feature_importances_
```

Out[33]:

```
array([0.08356431, 0.02454347, 0.46383039, 0.42806183])
```

In [34]:

```
# Remove the 'sepal_width' column from the DataFrame X and modify it in place.
X.drop(columns='sepal_width',inplace=True)

X.head()
```

Out[34]:

	sepal_length	petal_length	petal_width
0	5.1	1.4	0.2
1	4.9	1.4	0.2
2	4.7	1.3	0.2
3	4.6	1.5	0.2
4	5.0	1.4	0.2

In [35]:

```
rf = RandomForestClassifier()
rf.fit(X,y)

rf.feature_importances_
```

Out[35]:

```
array([0.18600689, 0.42553452, 0.38845859])
```

In [36]:

```
# DataFrame after dropping the 'sepal_length' column.
X.drop(columns='sepal_length',inplace=True)
X.head()
```

Out[36]:

	petal_length	petal_width
0	1.4	0.2
1	1.4	0.2
2	1.3	0.2
3	1.5	0.2
4	1.4	0.2

In [37]:

```
rf = RandomForestClassifier()
rf.fit(X,y)

rf.feature_importances_
```

Out[37]:

```
array([0.52458236, 0.47541764])
```

In [38]:

```
X.drop(columns='petal_length',inplace=True)
X.head()
```

Out[38]:

	petal_width
0	0.2
1	0.2
2	0.2
3	0.2
4	0.2

Explanation

the process of Recursive Feature Elimination (RFE) using a RandomForestClassifier as the machine learning model. RFE is a feature selection technique that iteratively removes the least important features from the dataset based on the feature importances provided by the model.

Here's a step-by-step explanation of the code:

- The code starts by importing the necessary libraries, namely pandas and numpy. The dataset 'iris.csv' is read into a pandas DataFrame called 'df'.
- The features (input variables) are extracted from the DataFrame 'df' and stored in the variable 'X', while the target variable 'petal_width' is stored in the variable 'y'.
- A RandomForestClassifier model is created and trained using all the available features in 'X'.
- The feature importances obtained from the trained model are printed to the console using the 'rf.feature_importances_' command. This shows how important each feature is for the model's performance.
- Next, the code removes the 'sepal_width' column from the DataFrame 'X' using the 'X.drop(columns='sepal_width', inplace=True)' command. The 'inplace=True' argument ensures that the DataFrame 'X' is modified directly.
- The DataFrame 'X' is printed to the console to show the result after removing the 'sepal_width' column.
- The RandomForestClassifier model is retrained using the modified 'X' and the target variable 'y'.
- The feature importances of the retrained model are printed to the console again.

Similarly, the code repeats the process by removing the 'sepal_length' column from the D

Sklearn RFE

In [39]:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.feature_selection import RFE
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Load iris dataset
url = "https://gist.githubusercontent.com/curran/a08a1080b88344b0c8a7/raw/0e7a9b0a5d22642a06d3d5b9bcbad9890c8ee
df = pd.read_csv(url)

# Separate features and target variable
X = df.drop("species", axis=1)
y = df["species"]

# Initialize RandomForestClassifier
model = RandomForestClassifier()

# Initialize RFE
rfe = RFE(estimator=model, n_features_to_select=1)

# Fit RFE
rfe.fit(X, y)

# Print the ranking
ranking = rfe.ranking_
print("Feature ranking:")

for i, feature in enumerate(X.columns):
    print(f"{feature}: {ranking[i]}")
```

```
Feature ranking:
sepal_length: 3
sepal_width: 4
petal_length: 2
petal_width: 1
```

Filter Methods

Mutual Information

Mutual Information (MI) is a statistical measure commonly used in feature selection to quantify the dependency or mutual information between two random variables. In the context of feature selection, it measures how much information one feature contains about the target variable.

$$MI = \sum_{x \in X} \sum_{y \in Y} p(x,y) \log \left[\frac{p(x,y)}{p(x)\,p(y)} \right]$$

where

$p(x,y)$ → Joint prob of X and Y

$p(x)$ → marginal prob of X

$p(y)$ → marginal prob of Y

The concept of Mutual Information is derived from information theory, and it can be used to evaluate the relevance of individual features with respect to the target variable in both classification and regression tasks. Higher MI values indicate a stronger relationship between the feature and the target variable, suggesting that the feature contains more relevant information for predicting the target.

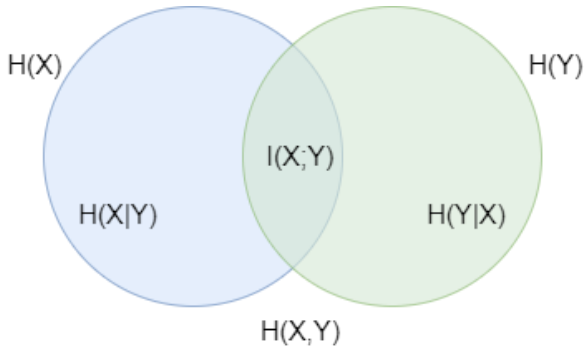
The Mutual Information between two discrete random variables X and Y is defined as:

MI(X, Y) = ∑∑ P(x, y) * log(P(x, y) / (P(x) * P(y)))

where P(x, y) is the joint probability distribution of X and Y, and P(x) and P(y) are the marginal probability distributions of X and Y, respectively.

In scikit-learn, you can use the `mutual_info_classif` function for classification tasks and `mutual_info_regression` for regression tasks to compute the mutual information scores for each feature.

Note: Mutual Information may not be as effective for high-dimensional datasets or datasets with strong linear relationships between features and the target variable. In such cases, other feature selection methods like LASSO, Recursive Feature Elimination, or tree-based feature importance may be more appropriate.



In [40]:

```
import pandas as pd

data = {
    'A': ['a1', 'a2', 'a1', 'a1', 'a2', 'a1', 'a2', 'a2'],
    'B': ['b1', 'b2', 'b2', 'b1', 'b1', 'b2', 'b2', 'b1']
}

df = pd.DataFrame(data)
```

In [41]:

```
marginal_prob = pd.crosstab(df['A'], df['B'], margins=True, normalize='all')
marginal_prob
```

Out[41]:

B	b1	b2	All
A			
a1	0.25	0.25	0.5
a2	0.25	0.25	0.5
All	0.50	0.50	1.0

In [42]:

```
from sklearn.feature_selection import mutual_info_classif
from sklearn.datasets import load_iris
import pandas as pd

# Load iris dataset
iris = load_iris()
X = iris['data']
y = iris['target']

# Compute mutual information
mi = mutual_info_classif(X, y)

# Print mutual information
for i, mi_value in enumerate(mi):
    print(f"Feature {i}: Mutual Information = {mi_value}")
```

```
Feature 0: Mutual Information = 0.4804093357325343
Feature 1: Mutual Information = 0.25035635773022347
Feature 2: Mutual Information = 0.9911552053675978
Feature 3: Mutual Information = 1.0036820279610252
```

In [43]:

```
from sklearn.feature_selection import SelectKBest, mutual_info_classif
from sklearn.datasets import load_iris

# Load iris dataset
iris = load_iris()
X = iris['data']
y = iris['target']

# Create SelectKBest feature selector
selector = SelectKBest(mutual_info_classif, k=2)

# Fit and transform
X_new = selector.fit_transform(X, y)

# Get columns to keep and create new dataframe with those only
cols = selector.get_support(indices=True)

print(iris.feature_names)
print(cols)
```

```
['sepal length (cm)', 'sepal width (cm)', 'petal length (cm)', 'petal width (cm)']
[2 3]
```

Overview of feature selection Techniques

Filter Methods:

- **Variance Threshold**: Removes all features whose variance doesn't meet a certain threshold. Use this when you have many features and you want to remove those that are constants or near constants.
- **Correlation Coefficient**: Finds the correlation between each pair of features. Highly correlated features can be removed since they contain similar information. Use this when you suspect that some features are highly correlated.
- **Chi-Square Test**: This statistical test is used to determine if there's a significant association between two variables. It's commonly used for categorical variables. Use this when you have categorical features and you want to find their dependency with the target variable.
- **Mutual Information**: Measures the dependency between two variables. It's a more general form of the correlation coefficient and can capture non-linear dependencies. Use this when you want to measure both linear and non-linear dependencies between features and the target variable.
- **ANOVA (Analysis of Variance)**: ANOVA is a statistical test that stands for "Analysis of Variance". ANOVA tests the impact of one or more factors by comparing the means of different samples. Use this when you have one or more categorical independent variables and a continuous dependent variable

Wrapper Methods:

- **Recursive Feature Elimination (RFE)**: Recursively removes features, builds a model using the remaining attributes, and calculates model accuracy. It uses model accuracy to identify which attributes contribute the most. Use this when you want to leverage the model to identify the best features
- **Sequential Feature Selection (SFS)**: Adds or removes one feature at the time based on the classifier performance until a feature subset of the desired size k is reached. Use this when computational cost is not an issue and you want to find the optimal feature subset.
- **Exhaustive Feature Selection**: This is a brute-force evaluation of each feature subset. This method, as the name suggests, tries out all possible combinations of variables and returns the best subset. Use this when the number of features is small, as it can be computationally expensive.

Embedded Methods:

- **Lasso Regression**: Lasso (Least Absolute Shrinkage and Selection Operator) is a regression analysis method that performs both variable selection and regularization. Use this when you want to create a simple and interpretable model.
- **Ridge Regression**: Ridge regression is a method used to analyze multiple regression data that suffer from multicollinearity. Unlike Lasso, it doesn't lead to feature selection but rather minimizes the complexity of the model.
- **Elastic Net**: This method is a combination of Lasso and Ridge. It incorporates penalties from both methods and is particularly useful when there are multiple correlated features.
- **Random Forest Importance**: Random forests provide a straightforward method for feature selection, namely mean decrease impurity (MDI). Use this when you want to leverage the power of random forests for feature selection

In [ ]: