# DATA SCIENCE-ASSIGNMENT

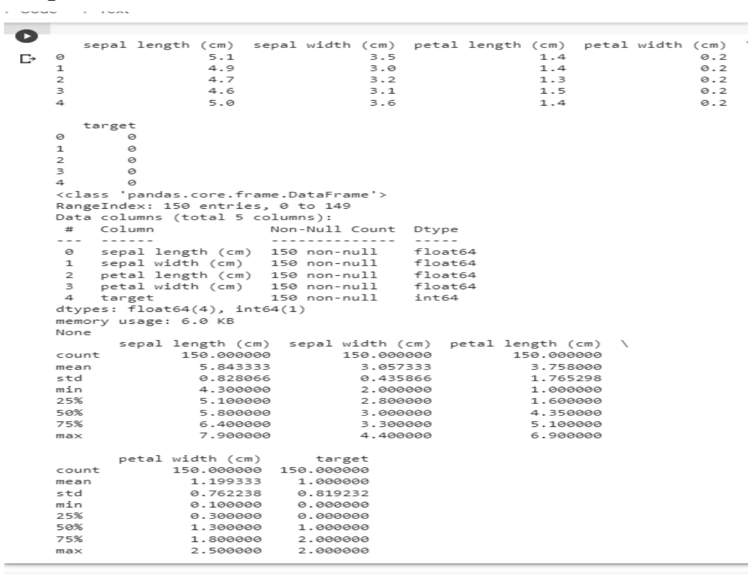
1. Perform Explanatory Data Analysis (EDA) on Iris Dataset.

(EDA Google collab link: https://colab.research.google.com/drive/1dg_aP-z9SvODd4Vkc738om4S7V4vTUDC?usp=share_link )

## DATA ANALYSIS ON IRIS DATA SET

```
#loading dataset from sklearn

from sklearn.datasets import load_iris

import pandas as pd

iris = load_iris()

df = pd.DataFrame(data=iris.data, columns=iris.feature_names)

df['target'] = iris.target


#basic information about the data

print(df.head())

print(df.info())

print(df.describe())
```

Output:



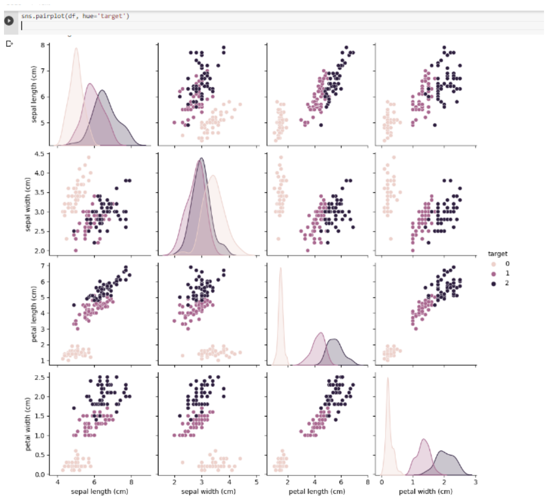
```
   sepal length (cm)  sepal width (cm)  petal length (cm)  petal width (cm)  \
0                5.1               3.5                1.4               0.2
1                4.9               3.0                1.4               0.2
2                4.7               3.2                1.3               0.2
3                4.6               3.1                1.5               0.2
4                5.0               3.6                1.4               0.2

   target
0       0
1       0
2       0
3       0
4       0
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150 entries, 0 to 149
Data columns (total 5 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   sepal length (cm)  150 non-null    float64
 1   sepal width (cm)   150 non-null    float64
 2   petal length (cm)  150 non-null    float64
 3   petal width (cm)   150 non-null    float64
 4   target             150 non-null    int64
dtypes: float64(4), int64(1)
memory usage: 6.0 KB
None
       sepal length (cm)  sepal width (cm)  petal length (cm)  \
count         150.000000        150.000000         150.000000
mean            5.843333          3.057333           3.758000
std             0.828066          0.435866           1.765298
min             4.300000          2.000000           1.000000
25%             5.100000          2.800000           1.600000
50%             5.800000          3.000000           4.350000
75%             6.400000          3.300000           5.100000
max             7.900000          4.400000           6.900000

       petal width (cm)      target
count        150.000000  150.000000
mean           1.199333    1.000000
std            0.762238    0.819232
min            0.100000    0.000000
25%            0.300000    0.000000
50%            1.300000    1.000000
75%            1.800000    2.000000
max            2.500000    2.000000
```

#visualising the data

import seaborn as sns

sns.pairplot(df, hue='target')

Output:



```
sns.pairplot(df, hue='target')
```

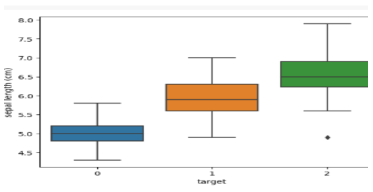
#differences between different types of iris

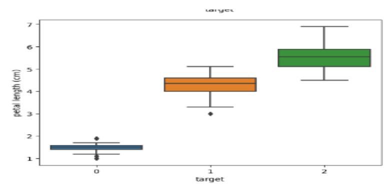
import matplotlib.pyplot as plt

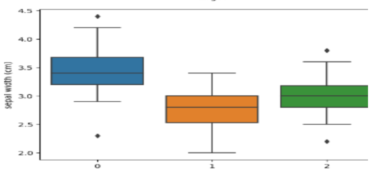
import seaborn as sns

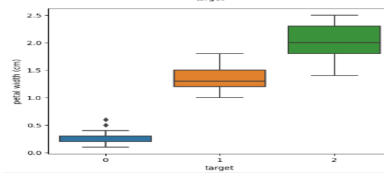
```
# Create a boxplot for each feature, broken down by target class

sns.boxplot(x='target', y='sepal length (cm)', data=df)

plt.show()

sns.boxplot(x='target', y='sepal width (cm)', data=df)

plt.show()

sns.boxplot(x='target', y='petal length (cm)', data=df)

plt.show()

sns.boxplot(x='target', y='petal width (cm)', data=df)

plt.show()

# Create a scatterplot matrix of all features, colored by target class

sns.pairplot(df, hue='target')

plt.show()
```

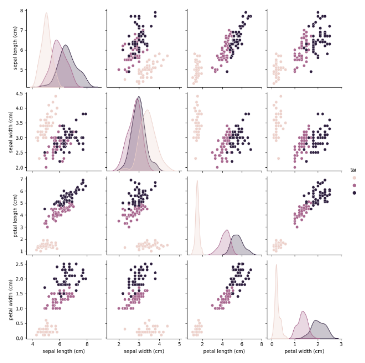
Output:











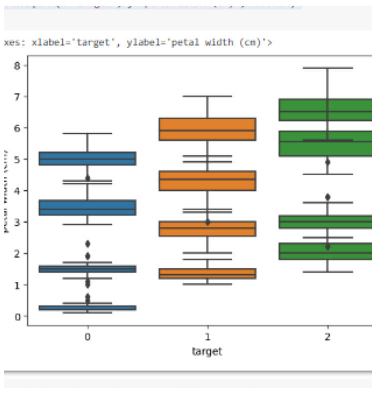
```
#visualising the distribution of each feature using boxplot

sns.boxplot(x='target', y='sepal length (cm)', data=df)
```

sns.boxplot(x='target', y='sepal width (cm)', data=df)

sns.boxplot(x='target', y='petal length (cm)', data=df)

sns.boxplot(x='target', y='petal width (cm)', data=df)

Output:



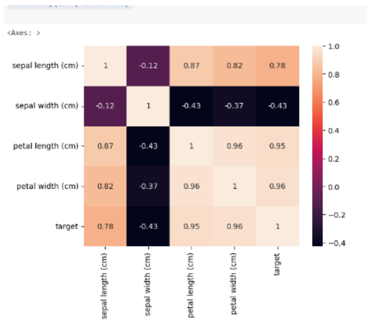
```
xes: xlabel='target', ylabel='petal width (cm)'>
```

#calculating the correlation coefficients of the features

corr = df.corr()

sns.heatmap(corr, annot=True)

Output:

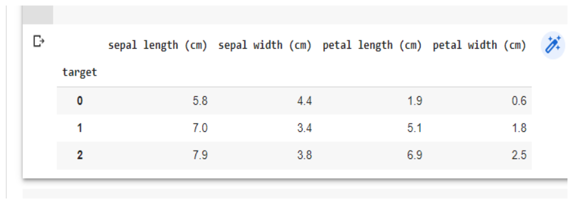


```
<Axes: >
```

## STATISTICAL SUMMARY ANALYSIS

```
# Calculate the mean of each feature for each target class

df.groupby('target').mean()

# Calculate the median of each feature for each target class

df.groupby('target').median()

# Calculate the standard deviation of each feature for each target class

df.groupby('target').std()

# Calculate the minimum value of each feature for each target class

df.groupby('target').min()

# Calculate the maximum value of each feature for each target class

df.groupby('target').max()
```

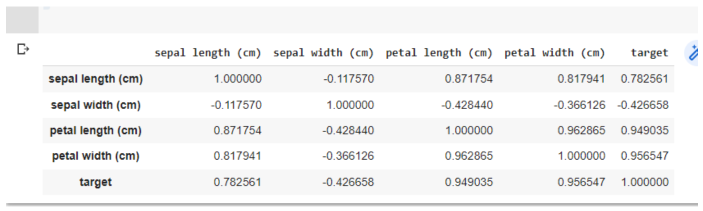
Output:



| target | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) |
|---|---|---|---|---|
| 0 | 5.8 | 4.4 | 1.9 | 0.6 |
| 1 | 7.0 | 3.4 | 5.1 | 1.8 |
| 2 | 7.9 | 3.8 | 6.9 | 2.5 |

```
df.corr()
```

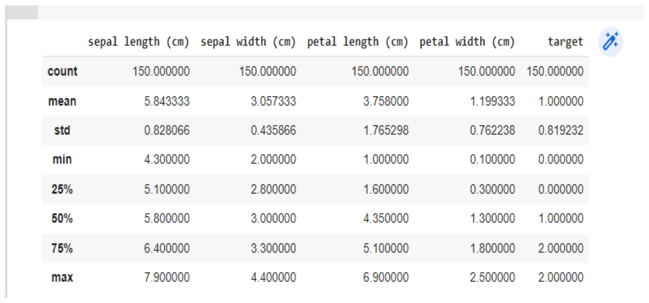
Output:



| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **sepal length (cm)** | 1.000000 | -0.117570 | 0.871754 | 0.817941 | 0.782561 |
| **sepal width (cm)** | -0.117570 | 1.000000 | -0.428440 | -0.366126 | -0.426658 |
| **petal length (cm)** | 0.871754 | -0.428440 | 1.000000 | 0.962865 | 0.949035 |
| **petal width (cm)** | 0.817941 | -0.366126 | 0.962865 | 1.000000 | 0.956547 |
| **target** | 0.782561 | -0.426658 | 0.949035 | 0.956547 | 1.000000 |

df.describe()

Output:



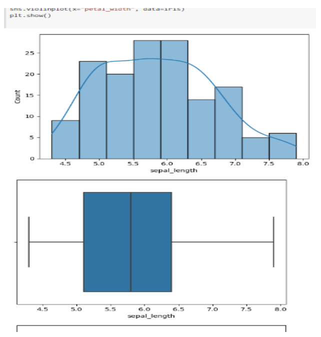
| | sepal length (cm) | sepal width (cm) | petal length (cm) | petal width (cm) | target |
|---|---|---|---|---|---|
| **count** | 150.000000 | 150.000000 | 150.000000 | 150.000000 | 150.000000 |
| **mean** | 5.843333 | 3.057333 | 3.758000 | 1.199333 | 1.000000 |
| **std** | 0.828066 | 0.435866 | 1.765298 | 0.762238 | 0.819232 |
| **min** | 4.300000 | 2.000000 | 1.000000 | 0.100000 | 0.000000 |
| **25%** | 5.100000 | 2.800000 | 1.600000 | 0.300000 | 0.000000 |
| **50%** | 5.800000 | 3.000000 | 4.350000 | 1.300000 | 1.000000 |
| **75%** | 6.400000 | 3.300000 | 5.100000 | 1.800000 | 2.000000 |
| **max** | 7.900000 | 4.400000 | 6.900000 | 2.500000 | 2.000000 |

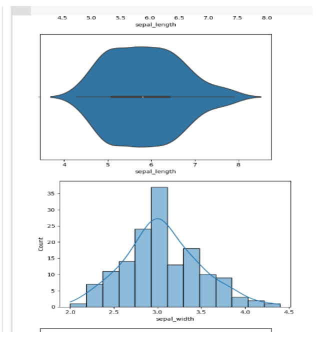
## UNIVARIENT ANALYSIS

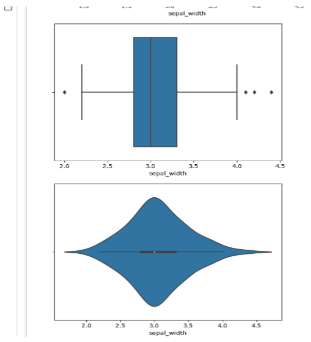
```
import seaborn as sns
iris = sns.load_dataset("iris")
# Univariate analysis of sepal length
sns.histplot(data=iris, x="sepal_length", kde=True)
plt.show()
sns.boxplot(x="sepal_length", data=iris)
plt.show()
sns.violinplot(x="sepal_length", data=iris)
plt.show()
```

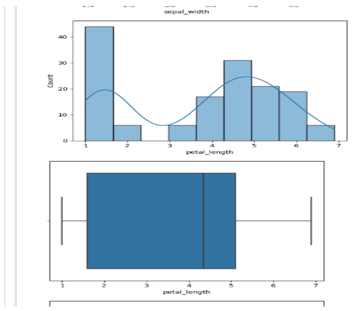
```
# Univariate analysis of sepal width
sns.histplot(data=iris, x="sepal_width", kde=True)
plt.show()
sns.boxplot(x="sepal_width", data=iris)
plt.show()
sns.violinplot(x="sepal_width", data=iris)
plt.show()
# Univariate analysis of petal length
sns.histplot(data=iris, x="petal_length", kde=True)
plt.show()
sns.boxplot(x="petal_length", data=iris)
plt.show()
sns.violinplot(x="petal_length", data=iris)
plt.show()
# Univariate analysis of petal width
sns.histplot(data=iris, x="petal_width", kde=True)
plt.show()
sns.boxplot(x="petal_width", data=iris)
plt.show()
sns.violinplot(x="petal_width", data=iris)
plt.show()
```

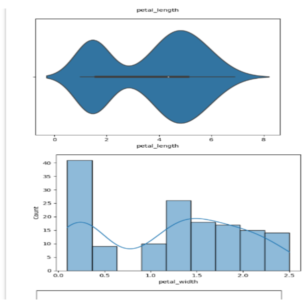
Output:

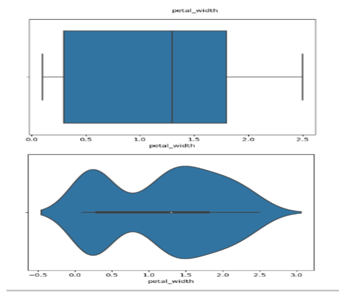

```
sns.violinplot(x='petal_width', data=iris)
plt.show()
```











## BIVARIENT ANALYSIS

import seaborn as sns

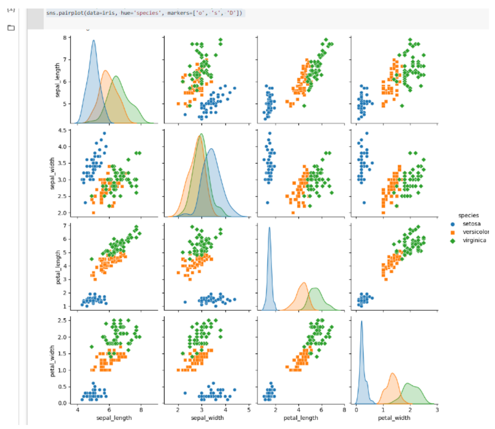
# Load the Iris dataset

iris = sns.load_dataset('iris')

# Create scatterplots of pairs of features, colored by species

sns.pairplot(data=iris, hue='species', markers=['o', 's', 'D'])

Output:



```
sns.pairplot(data=iris, hue='species', markers=['o', 's', 'D'])
```

## MULTIVARIENT ANALYSIS

import seaborn as sns

# Load the Iris dataset

iris = sns.load_dataset('iris')

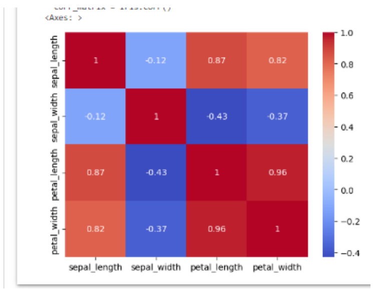
# Calculate the correlation matrix

corr_matrix = iris.corr()

# Create a heatmap of the correlation matrix

sns.heatmap(corr_matrix, annot=True, cmap='coolwarm')

Output:



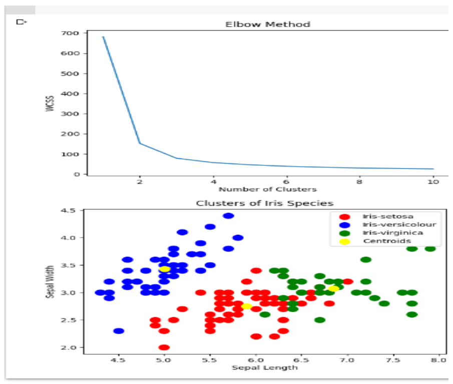
```
<Axes: >
```

## CLUSTER ANALYSIS

```
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
# Load the Iris dataset
iris = sns.load_dataset('iris')
# Create an array of the feature values
X = iris.iloc[:, :-1].values
# Find the optimal number of clusters using the elbow method
wcss = []
for i in range(1, 11):
    kmeans = KMeans(n_clusters=i, init='k-means++', max_iter=300, n_init=10, random_state=0)
    kmeans.fit(X)
    wcss.append(kmeans.inertia_)
plt.plot(range(1, 11), wcss)
plt.title('Elbow Method')
plt.xlabel('Number of Clusters')
plt.ylabel('WCSS')
plt.show()
# Apply the k-means algorithm with the optimal number of clusters
kmeans = KMeans(n_clusters=3, init='k-means++', max_iter=300, n_init=10, random_state=0)
y_kmeans = kmeans.fit_predict(X)
# Visualize the clusters
plt.scatter(X[y_kmeans == 0, 0], X[y_kmeans == 0, 1], s = 100, c = 'red', label = 'Iris-setosa')
plt.scatter(X[y_kmeans == 1, 0], X[y_kmeans == 1, 1], s = 100, c = 'blue', label =
'Iris-versicolour')
```

```
plt.scatter(X[y_kmeans == 2, 0], X[y_kmeans == 2, 1], s = 100, c = 'green', label =
'Iris-virginica')

plt.scatter(kmeans.cluster_centers_[:, 0], kmeans.cluster_centers_[:, 1], s = 100, c = 'yellow',
label = 'Centroids')

plt.title('Clusters of Iris Species')

plt.xlabel('Sepal Length')

plt.ylabel('Sepal Width')

plt.legend()

plt.show()
```

Output:



## Dimensionality reduction using PCA

```
from sklearn.decomposition import PCA

import seaborn as sns

# Load the Iris dataset

iris = sns.load_dataset('iris')

# Separate features and target variable

X = iris.iloc[:, :-1].values

y = iris.iloc[:, -1].values
```

# Standardize the data

from sklearn.preprocessing import StandardScaler

sc = StandardScaler()

X_std = sc.fit_transform(X)

# Apply PCA with 2 components

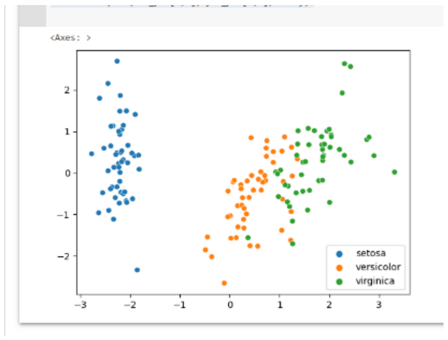
pca = PCA(n_components=2)

X_pca = pca.fit_transform(X_std)

# Plot the PCA results

sns.scatterplot(x=X_pca[:,0], y=X_pca[:,1], hue=y)

Output:



```
<Axes: >
```

## PREDICTIONS

import numpy as np

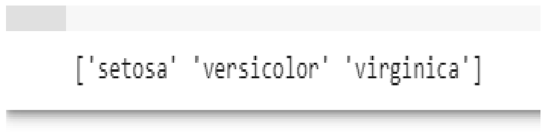
from sklearn.neighbors import KNeighborsClassifier

# Load the Iris dataset

iris = sns.load_dataset('iris')

```
# Separate features and target variable
X = iris.iloc[:, :-1].values
y = iris.iloc[:, -1].values
# Train a K-nearest neighbors classifier
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X, y)
# Create new data to make predictions on
new_data = np.array([[5.1, 3.5, 1.4, 0.2], [6.2, 2.9, 4.3, 1.3], [7.7, 3.0, 6.1, 2.3]])
# Make predictions on the new data
predictions = knn.predict(new_data)
# Print the predicted species for each data point
print(predictions)
```

Output:

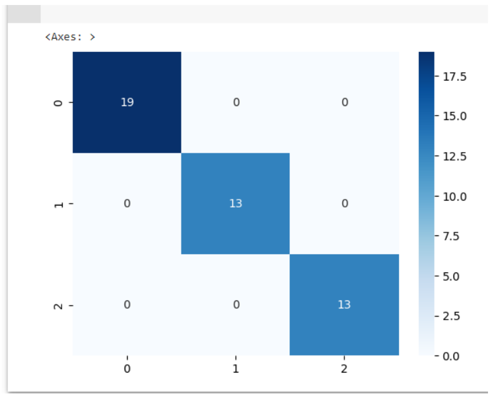


```
['setosa' 'versicolor' 'virginica']
```

```
#representing in graphs
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
# Load the Iris dataset
iris = sns.load_dataset('iris')
# Separate features and target variable
X = iris.iloc[:, :-1].values
```

```
y = iris.iloc[:, -1].values

# Split the data into training and test sets

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# Train a K-nearest neighbors classifier

knn = KNeighborsClassifier(n_neighbors=3)

knn.fit(X_train, y_train)

# Make predictions on the test set

y_pred = knn.predict(X_test)

# Compute the accuracy score and confusion matrix

acc = accuracy_score(y_test, y_pred)

cm = confusion_matrix(y_test, y_pred)

# Plot the confusion matrix

sns.heatmap(cm, annot=True, cmap='Blues')
```

Output:



```
<Axes: >
```

(EDA Google collab link: https://colab.research.google.com/drive/1dg_aP-z9SvODd4Vkc738om4S7V4vTUDC?usp=share_link)

## 2. What is Decision Tree? Draw decisison tree by taking the example of Play Tennis.

A decision tree is a type of model, similar to a flowchart, that illustrates options and their potential results, such as chance event outcomes, resource costs, and utility. It is one of the often employed tools in data mining and machine learning for the resolution of classification and prediction issues.A supervised learning approach used in machine learning for both classification and regression issues is a decision tree. It entails building a tree-like model of choices and potential outcomes. The interior nodes of the tree stand for choices, and the leaf nodes represent results. Here is an example of a decision tree for the tennis problem:

```
              Outlook
                /  \

         Sunny     Not Sunny

           |           |

        Humidity    Play Tennis

         /    \

      High     Normal

       |         |

   Don't play   Play

     Tennis     Tennis
```

The initial choice in this decision tree is based on the viewpoint. The humidity will determine whether or not tennis will be played if the weather is sunny. If the humidity is high, it is decided not to play; nevertheless, if it is normal, it is decided to play. The choice is always to play tennis if the forecast is cloudy.

## 3. In K-means or KNN, we use Euclidean distance to calculate the distance between nearest neighbours. Why not manhattan distance?

Different distance metrics, such as Euclidean distance and Manhattan distance, can be used in K-means or K-NN algorithms to determine the separation between the closest neighbours. Since Euclidean distance is the most understandable and well-known distance metric and performs well in many situations, it is frequently utilised.

- When the underlying data distribution is smooth and continuous, Manhattan distance can be less precise than Euclidean distance. In some circumstances, Manhattan distance may overestimate the true distance and the straight-line distance represented by Euclidean distance may be a better representation of the actual distance between two places.
- The diagonal distance between two sites is not taken into account by the Manhattan distance. When two locations are diagonal to one another, this can cause the distance between them to be overestimated. Euclidean distance, on the other hand, takes the diagonal distance into account, making it a more precise distance measurement when points are not aligned with the axes.
- When imagining the distance between two points, Manhattan distance is less understandable than Euclidean distance. The straight-line distance between two points, which is simple to see, is the equivalent of the Euclidean distance. Manhattan distance, on the other hand, can be more challenging to visualise because it represents the sum of the differences in each dimension.
- For some forms of data, the Manhattan distance might not be appropriate. Euclidean distance might be a more appropriate way to estimate distance in particular circumstances, for instance, where the data may have a naturally geometric meaning.

## 4. How to test and know whether or not we have overfitting problem?

A prevalent issue in machine learning is overfitting, which occurs when a model performs well on training data but badly on test data. We can use the following techniques to determine if we have an overfitting issue:

- Train-Test Split: Split the data into a training set and a test set to create a train-test split. The model should be tested after being trained on the training set. The model is probably overfitting if it performs well on the training set but poorly on the test set.
- Cross-validation: Use cross-validation to assess the effectiveness of the model. K-fold cross-validation involves splitting the data into k pieces and training the model k times, using a different fold as the test set and the remaining data as the training set. The model's performance can be evaluated using the average performance over the k folds. The model is probably overfitting if it performs well on the training set but poorly on the validation set.
- Learning curves: Plot the model's performance on the training set and the test set as a function of the volume of training data to create learning curves. It is probably overfitting if the test error is high but the training error is low.
- Regularisation: To lessen the complexity of the model and avoid overfitting, use regularisation techniques like L1 or L2 regularisation. Regularisation changes the loss function by adding a penalty term, which deters the model from trying to fit the noise in the training set.
- Consider using simpler models: If the model is still overfitting after regularisation, take into account employing a simpler model, like a linear model, a decision tree with a shallower branch, or a simpler neural network architecture.

## 5. How is KNN different from K-means Clustering?

Machine learning methods for data analysis and pattern recognition include KNN (K-Nearest Neighbours) and K-means clustering, however their approaches are fundamentally different.

### KNN:

For classification and regression applications, KNN is a supervised learning method. By using a distance metric to identify the k data points closest to a new data point, it assigns a class or value depending on the majority or average of those k neighbours. Since KNN uses an instance-based learning algorithm, it memorises the training set and bases its predictions on how closely the fresh data resembles the training set.

### K-means clustering:

On the other hand, K-means clustering is an unsupervised learning algorithm used to cluster or group together similar data points. It operates by dividing the data into k clusters, where k is the user-specified number of clusters. Each data point is iteratively assigned to the closest cluster centre by the algorithm, which then recalculates the centre of each cluster using the mean of the data points in that cluster. The algorithm iterates until there is no longer a significant change in the cluster centres.

## 6. Can you explain the difference between a Test set and a Validation set?

The function they play in the machine learning workflow is the key distinction between the validation set and the test set. While the test set is used to estimate the model's final performance, the validation set is used to adjust the model and optimise its hyperparameters.

### Validation Set:

The validation set is specifically utilised during training to assess the model's performance on data that wasn't included for training. As a result, we can modify the model's hyperparameters and architecture to enhance its performance. A subset of the training data, the validation set is often lower in size than the training set.

### Test Set:

On the other hand, the test set is used to assess the model's ultimate performance after it has been trained and optimised. This is crucial in determining how well the model will extrapolate to fresh, untested data. The size of the test set is often similar to that of the training set. It is an entirely different set of data that was not utilised during training or validation.

## 7. How can you avoid overfitting in KNN?

- Using a higher value of k makes the model less sensitive to noise and less likely to overfit the training data, which is one method for preventing overfitting in KNN. The model may become less accurate and less able to detect subtle patterns in the data, though, if k is given a large value.
- Utilising feature selection or dimensionality reduction techniques to lessen the amount of features or dimensions in the data is another method to prevent overfitting in KNN. This can make it simpler for the model to recognise the underlying patterns by removing noise and unimportant data from the input.
- Techniques for regularisation can also be used to prevent overfitting in KNN. The process of regularisation involves adding a penalty term to the algorithm's distance metric. This penalty term pushes the model to select simpler solutions that are more adaptable to new data.
- Cross-validation should be used to assess the model's performance and identify overfitting. In cross-validation, the data are split into training and validation sets, and the performance of the model is assessed on the validation set. The model may be overfitting the training data if it performs well on the training set but badly on the validation set. To lessen overfitting in this situation, the model should be retrained using a higher value of k or other methods.
- several other methods like data augmentation, distance weighted KNN, Ensembling, Local outlier factor can also be used.

## 8. What is precision?

Precision is a statistic used in machine learning to assess how well a binary classification model is performing. Out of all the positive predictions made by the model, it calculates the percentage of real positive forecasts. The ratio of true positives (TP) to the total of true positives and false positives (FP) is used to calculate precision.

Precision is equal to TP/(TP + FP).

In other words, precision provides the answer to the question: How many of the cases that were predicted as positive truly are positive? When the cost of false positives is large, like in medical diagnosis or fraud detection, precision is a relevant indicator. It is crucial for the model's predictions to be highly precise in these cases because a false positive forecast could have negative effects.

## 9. Explain how a ROC Curve works?

A binary classification model's performance is represented graphically by the ROC (Receiver Operating Characteristic) curve, which demonstrates how well the model can distinguish between the positive and negative examples in the dataset. Plotting the true positive rate (TPR) vs the false positive rate (FPR) at various threshold values results in the curve.

Let's define the TPR and FPR first to better understand how the ROC curve functions:

- True Positive Rate (TPR): The percentage of positive occurrences that the model correctly classifies. It also goes by the names sensitivity and recall. TP is the number of true positives, and FN is the number of false negatives, and TPR is computed as TP / (TP + FN).

- False Positive Rate (FPR): The percentage of negative occurrences that the model wrongly interprets as positive. The formula for calculating false positive rate (FPR) is FP / (FP + TN), where FP stands for false positives and TN for true negatives.

The model produces predictions on the test set in a binary classification problem that can either be positive or negative. To calculate the number of true positives, false positives, true negatives, and false negatives, the predictions are then contrasted with the actual labels. Plotting TPR against FPR at various threshold values yields the ROC curve. The probability threshold at which a sample is categorised as positive is determined by the threshold value, which is a value between 0 and 1.

## 10. What is accuracy?

A binary classification model's performance is typically evaluated using accuracy in machine learning. It calculates the percentage of all predictions that the model made that were accurate. To put it another way, accuracy indicates how frequently the model correctly predicts the class of a given instance.

A model's accuracy is determined using the following formula:

Accuracy is equal to (TP+TN)/(TP+TN+FP+FN)

where TP (True Positives) is the total number of instances that were correctly predicted to be positive, TN (True Negatives) is the total number of instances that were correctly predicted to be negative, FP (False Positives) is the total number of instances that were incorrectly predicted to be positive, and FN (False Negatives) is the total number of instances that were incorrectly predicted to be negative.

## 11. What is F1 Score?

A typical evaluation metric in machine learning is the F1 score, which combines precision and recall to give a single assessment of a model's effectiveness. It is especially helpful in binary classification issues with imbalanced classes, where one class could contain a lot less occurrences than the other.

The harmonic mean of recall and precision is used to generate the F1 score.

F1 score is calculated as follows: 2 * (precision * recall) / (precision + recall), where precision is the percentage of true positives (TP) among all positive predictions (TP + FP) and recall represents the percentage of TP among all true positives (TP + FN).

## 12. What is Recall?

Recall is a machine learning evaluation metric that quantifies a model's capacity to accurately identify positive examples out of all real positive cases in a dataset. Other names for it include sensitivity and true positive rate.

Recall is determined by dividing the number of true positive (TP) instances by the total number of real positive (TP + FN) cases. In other words, it assesses the model's capacity to avoid false negatives, which are occasions in which the model mistakenly classifies positive data as negative.

This formula can be used to describe recall:

Recall is TP / (TP + FN).

## 13. What is a Confusion Matrix and why do we need it?

A table called a confusion matrix is frequently used to assess how well a machine learning model is working. It lists how many predictions a model made correctly and incorrectly for each class in the dataset. The confusion matrix includes two categories—positive and negative—for each of its two dimensions—actual and predicted. The true positive (TP), false positive (FP), true negative (TN), and false negative (FN) counts are represented by the four cells of the matrix. Accuracy, precision, recall, and F1 score are just a few of the evaluation metrics for the model that can be computed using the confusion matrix. These metrics offer information about the model's performance for each class and can be used to assess whether the model is overfitting or not or a poor fit to the data.

### Example:

Take the binary classification problem, for instance, where we are attempting to determine whether or not an email is spam. This could be the confusion matrix for this issue:

| | Predicted spam | Predicted not spam |
|---|---|---|
| Actual spam | 95 | 5 |
| Actual not spam | 10 | 890 |

95 spam emails were accurately identified as spam in this example (true positives), while 5 non-spam emails were falsely identified as spam (false positives). It accurately identified 890 spam-free emails as such (true negatives) while misidentifying 10 spam-free emails (false negatives)

## 14. What do you mean by AUC Curve?

The phrase "Area Under the Curve" refers to the region beneath the Receiver Operating Characteristic (ROC) curve. The model's overall performance over all potential discrimination thresholds is measured by the AUC. It has a value between 0 and 1, with a higher number denoting better performance. An AUC of 0.5 means the model is performing no better than guessing at random, and an AUC of 1.0 means the model is functioning perfectly.

The AUC is a helpful indicator for evaluating the effectiveness of various models and choosing the most appropriate threshold for prediction. As it considers the trade-off between TPR and FPR for various threshold values, it is helpful in imbalanced classification issues when one class is significantly more prevalent than another.

## 15. What is Precision-Recall Trade-Off?

In binary classification, the precision-recall trade-off is a concept that describes the interrelationship between precision and recall and how modifications to one statistic may have an impact on the other. Recall is the percentage of true positives among all actual positives, whereas precision is the percentage of true positives among all positive forecasts. Generally speaking, a rise in precision frequently causes a fall in recall, and vice versa. This is so because recall indicates how thorough the positive predictions are, while precision measures how accurate they are.

## 16. What are Decision trees?

A supervised learning technique known as a decision tree is used in machine learning to perform classification and regression problems. They are built on a hierarchical framework that depicts a series of choices and potential outcomes. Each node in a decision tree represents a choice made in response to a feature or attribute, and each branch denotes the result of that choice. The conclusion or forecast is represented by the tree's leaves. In order to maximise information gain at each split, the decision tree algorithm iteratively divides the data into subgroups depending on the most crucial properties.

## 17. Explain the structure of a Decision tree

A hierarchical set of nodes that reflect decisions or attributes and their potential consequences forms the foundation of decision trees' structure. The final nodes of the tree are known as leaf nodes, while the top node is known as the root node. A flowchart or tree-like diagram can be used to describe the structure of a decision tree, with each node and branch represented by a box or a line. The algorithm repeatedly produces judgements and divides the data into smaller subsets based on the importance of each feature, and the structure is controlled by the features or attributes of the dataset. The algorithm's objective is to build a tree that can correctly categorise or predict new data using the training set of information.

## 18. What are some advantages of using Decision trees?

Using decision trees as a machine learning method has various benefits:

1. Simple to Understand: Even for non-technical consumers, decision trees are simple to perceive and comprehend. A series of defensible logical conclusions are represented by a tree structure that is simple to understand.

2. Handle Both Numerical and Categorical Data: Without the requirement for pre-processing or data normalisation, decision trees can handle both numerical and categorical data.

3. Non-Parametric: Because decision trees are non-parametric, they do not assume anything about how the data are distributed. They become more adaptable as a result, making them suited for a greater variety of data kinds and distributions.

4. Quick and Scalable: Even for big datasets, decision trees are quick and scalable. They can handle massive volumes of data with ease and can be trained quickly.

5. Feature Selection: Decision trees can be used for feature selection and feature engineering since they automatically choose the most significant features.

6. Robust to missing values: Decision trees can manage missing values by assigning them to the most prevalent class in the dataset, which makes them robust to missing values.

7. Ensemble Methods: By combining decision trees with other decision trees, ensemble methods like random forests can increase the model's accuracy and robustness.

## 19. How is a Random Forest related to Decision trees?

Decision trees are an extension of the ensemble learning technique known as Random Forest. In a random forest, many alternative decision trees are built using various subsets of the data and randomly selected characteristics. A final forecast is then created by combining the output of these decision trees. The shortcomings of decision trees, which might overfit the training data and have large variance, are addressed with random forests. A random forest can lower variation and produce more reliable and accurate forecasts by merging the outputs of many decision trees.

## 20. How are the different nodes of the Decision trees represented?

Decision trees' various nodes are illustrated as follows:

1. Root Node: The root node is the topmost node in a decision tree. It is a representation of the complete dataset under consideration.

2. Internal Nodes: Based on input attributes, internal nodes indicate decisions. Each internal node evaluates the worth of a particular feature and chooses a course of action depending on the test results.

3. Leaf Nodes: Leaf nodes show the decision tree's results. A single class or value is represented by each leaf node and is associated with a particular set of input attributes.

4. Branches: The decision-making routes from one node to another are represented by branches. The results of the choice test that was run at each node are labelled on them.

## 21. What type of node is considered pure?

In a decision tree, a leaf node that only has instances of one class of objects is said to be pure. This indicates that no additional splits are needed in that branch of the tree because every instance in that leaf node is a member of the same class. In other words, a pure node reflects a decision border between several classes and has the highest degree of homogeneity inside it. A decision tree algorithm aims to produce a tree with the fewest number of pure leaf nodes and the least amount of depth and complexity.

## 22. How would you deal with an overfitted Decision tree?

Several solutions can be used if a decision tree is overfitting the training set:

1. Pruning: Pruning is one of the most used methods for preventing overfitting in decision trees. Pruning entails deleting tree components that don't significantly improve the model's ability to classify objects correctly. Techniques like reduced error pruning and cost complexity pruning can be used for this.

2. Early Stopping: The early termination of the tree-building process is another method to avoid overfitting. This can be done by imposing a minimum requirement for the quantity of training samples in each leaf node or by restricting the maximum depth of the tree.

3. Cross-Validation: Cross-validation can be used to assess a decision tree model's performance on unidentified data. In order to do this, the data must be divided into training and validation sets, the model must be built using the training data, and it must then be evaluated using the validation data. To provide a more precise assessment of the model's performance, this procedure can be done numerous times with various data splits.

4. Feature Selection: Using too many features in the decision tree model can also lead to overfitting. Techniques for feature selection can be used to determine which features are most crucial for the model's classification accuracy and to omit the less crucial ones.

5. Ensemble approaches: Decision tree models can be made more accurate and resilient by employing ensemble approaches like random forests or boosting. These techniques integrate several decision trees to create a more robust model that resists overfitting.

## 23. What are some disadvantages of using Decision trees and how would you solve them?

Using decision trees as a machine learning technique has some drawbacks, including:

Decision trees are prone to overfitting, which leads to subpar performance on new data. Techniques like trimming, early stopping, cross-validation, and feature selection, which were covered in the preceding response, can be used to address this.

- Instability: Decision trees can become unstable because they are sensitive to even little changes in the data. Boosting or random forests ensemble approaches, which can lower the variance in the model and increase its stability, can be used to address this.
- Bias: Features having a lot of levels or values may be favoured by decision trees. By identifying the most crucial qualities and excluding the less significant ones, this problem can be solved utilising feature selection approaches.
- Interpretability: Decision trees are simple to grasp and interpret, but when working with massive datasets or high-dimensional feature spaces, they can become complex and challenging to explain. This can be overcome by explaining the decision-making process with simpler models or visualisations.
- Missing data handling: Decision trees are unable to deal with missing data, which might result in biassed or inaccurate predictions. Other machine learning algorithms that can handle missing data can be used to solve this problem. These techniques include imputation, which fills in missing values.

## 24. What is Gini Index and how is it used in Decision trees?

The splitting criteria for the nodes in decision trees are decided using the Gini index, a measure of impurity or inequality. The frequency with which a randomly selected element would be erroneously classified if it were randomly categorised in accordance with the distribution of classes in the node is measured statistically.

The Gini index has a range of 0 to 1, with 0 denoting perfect classification (all the elements fall into the same class), and 1 denoting total impurity (the elements are evenly dispersed among all classes).

The characteristic that maximises the separation between the classes at each node is chosen in decision trees using the Gini index. The splitting criterion is chosen to be the feature with the lowest Gini index, or the feature that generates the most homogeneous subgroups. Recursively performing this process continues until either a stopping requirement is reached or all leaf nodes are pure (i.e., contain only one class).

## 25. How would you define the stopping criteria for the decision trees?

To avoid overfitting the model to the training set of data, decision trees provide halting criteria. Overfitting happens when the tree is too complicated and captures noise and peculiarities in the training data instead of generalising to new, unexplored data. Stopping criteria try to strike a balance between the tree's intricacy and its capacity to fit the data.

Here are a few typical decision-tree stopping criteria:

1. Maximum depth: Set a maximum number of layers for the depth of the tree. This assists in keeping the tree from growing too complicated.

2. Minimum number of samples per leaf node: When the number of samples in a node drops below a predetermined level, the splitting process is terminated. This stops the tree from growing minuscule, disruptive leaf nodes.

3. Minimum impurity decrease: Stop splitting a node if the impurity measure (such as the Gini index or entropy) does not fall below a predetermined level. This stops the tree from splitting up ineffectively to increase the purity of the nodes.

4. Maximum number of leaf nodes: Limit the number of leaf nodes in the tree to a maximum number. By doing so, the tree is kept from overcomplicating and overfitting the data.

## 26. What is Entropy?

Entropy is a measure of impurity or homogeneity in a node that is used in decision trees. In a binary classification issue, a node is deemed impure if it contains a combination of samples from several classes while being regarded pure if all of the samples are from the same class. Entropy is a metric that measures a node's impurity based on the percentage of samples from each class.

Its formula is as follows: $H(S) = -sum_i=1c p_i log_2 p_i$,

where $S$ denotes the set of samples in the node, $c$ denotes the number of classes, and $p_i$ denotes the percentage of samples in class $i$. When the samples are evenly distributed throughout the classes, the entropy is at its highest point, and at its lowest point (i.e., 0) when all the samples belong to the same class.

## 27. How do we measure the Information?

Entropy can be used to calculate the degree of information in a set of data in the context of decision trees and information theory. Entropy is a metric for how random or impure a set of data is. In other words, it assesses the amount of surprise or uncertainty surrounding a random event's outcome.

The definition of entropy is:

$$ H(S) = -sum_i=1n log_2 p_i p_i $$

where $p_i$ is the percentage of the data that belongs to the $i$-th class and $S$ is a set of data with $n$ different classes.

## 28. What is the difference between Post-pruning and Pre-pruning?

### Pre-pruning:

Setting a stopping condition before constructing a decision tree is referred to as pre-pruning. Accordingly, during the building process, the tree will be pruned depending on many criteria, such as the smallest number of samples needed to split a node, the maximum depth of the tree, or the minimal increase in impurity or information gain.

### Post-pruning:

On the other hand, post-pruning describes the process of initially growing a complete decision tree before thinning it back by eliminating part of the branches. Cross-validation or a different validation set are frequently used to accomplish this. The goal is to eliminate any branches that do not enhance the model's generalisation performance or that do not improve accuracy on the validation set.

## 29. Compare Linear regression and Decision trees.

### Linear Regression:

The assumption behind linear regression is that the relationship between the input and output variables is linear. Finding the best-fit line or plane that minimises the sum of squared errors between the predicted values and the actual values is the aim of linear regression. Since linear regression is a parametric method, it is dependent on a number of data assumptions, including normality, linearity, and homoscedasticity.

### Decision Trees:

Contrarily, decision trees are non-parametric models that can manage relationships between the input and output variables that are not linear. To maximise the homogeneity of the output variable within each subset, decision trees iteratively divided the data into subsets depending on the values of the input variables. Mixed data types can be handled by decision trees.

## 30. What is the relationship between Information Gain and Information Gain Ratio?

Decision trees employ information gain and information gain ratio to assess the quality of a split or a feature.

### Information Gain:

Information gain is the difference between a parent node's impurity and the weighted sum of its offspring nodes' impurities. It assesses the amount of knowledge gathered about the target variable by dividing the data according to a specific attribute.

### Information Gain Ratio:

Information gain ratio, on the other hand, considers both the information gain and the intrinsic information of a feature. It adjusts the information gain measure to account for any bias towards features that have a lot of different possible values. The ratio of the information gain to the inherent information of the feature is used to determine the information gain ratio.

## 31. Compare Decision Trees and K-Nearest Neighbours.

### Decision Trees:

Based on the values of the characteristics, decision trees operate by repeatedly dividing the input space into smaller and smaller sections. To produce homogeneous subgroups of samples, each split is chosen to maximise the information gain or another splitting criterion. Decision trees can represent non-linear and interaction effects between features and are simple to understand and visualise. However, if the tree is too deep and the training set contains a lot of noisy or sparse data, they may be prone to overfitting.

### K-Nearest Neighbours:

On the other hand, K-NN is a lazy learning method that maintains all training data and bases predictions on the query instance's k-nearest neighbours. The class of the question instance is determined by a majority vote of the k-nearest neighbours, and the distance between instances is often calculated using the Euclidean distance. K-NN is straightforward and does not make any assumptions regarding the distribution of the data at its core. It can capture complex decision limits and be successful for high-dimensional, noisy data. The choice of k, however, can have an impact on the model's performance and it can be sensitive to unimportant factors.

## 32. While building Decision Tree how do you choose which attribute to split at each node?

The attribute to divide at each node of a decision tree is determined by the following standards:

1. Information Gain: The information gained about the target variable by splitting on a certain property is measured by information gain. It is described as the distinction between the parent node's entropy and the weighted average of the child nodes' entropies. At each node, the property that produces the greatest information gain is selected for splitting.

2. Gini Index: Like entropy, the Gini Index is a measure of impurity. It calculates the likelihood of misclassifying a randomly selected dataset element. Attribute splitting at each node occurs when using the lowest Gini Index-producing attribute.

3.Gain Ratio: The Information Gain criteria favours qualities that have a lot of different values. By leveraging the intrinsic information of the attribute to normalise the Information Gain, Gain Ratio eliminates this bias. At each node, the characteristic that produces the highest Gain Ratio is selected for splitting.

## 33. How would you compare different algorithms to build Decision Trees?

We can use a variety of evaluation criteria, such as accuracy, precision, recall, F1 score, and AUC-ROC, to compare various decision tree-building techniques.Multiple decision tree models can be trained using various methods, such as ID3, C4.5, CART, and Random Forest, and their performance can then be assessed using these metrics on a validation set. To assess how well each model performs across various validation sets, we may also use methods like cross-validation.

## 34. How do you Gradient Boost decision trees?

A common method for creating numerous decision trees repeatedly is called Gradient Boosted Decision Trees (GBDT). Each new tree in the GBDT method aims to fix the flaws of the previous tree.

The steps for gradient-boosted decision trees are as follows:

1. Use the training data to create the first decision tree.

2. Predict the target variable for each instance in the training data using this decision tree.

3. Subtract the anticipated values from the actual values to calculate the residuals.

4. Use the residuals as the target variable while training the following decision tree.

5. Combine the new decision tree's predictions with those made by the earlier trees.

6. Keep going through steps 3 through 5 until the desired number of trees is reached or the residuals can no longer be further decreased.

7. In order to create predictions about new data, employ the ensemble of decision trees.

The gradient boosting approach measures the error in the predictions using a loss function, such as mean squared error or cross-entropy. The method then modifies the decision trees' parameters to reduce this mistake.

## 35. What are the differences between Decision Trees and Neural Networks?

Two common methods in machine learning are decision trees and neural networks. Both are used for classification and regression problems, although they differ fundamentally in the following ways:

1. Construction: Decision trees are made up of nodes and branches that resemble a tree, with each node denoting a decision and each branch denoting the result of that decision. On the other hand, neural networks have an interconnected network of nodes, each of which stands in for a neuron.

2. Training: To build decision trees, data is recursively divided into smaller subsets according to the attributes that best distinguish the data. On the other hand, neural networks are trained using the backpropagation algorithm, which modifies the weights of the connections between neurons in order to reduce the loss function.

3. Interpretability: Because they help us visualise the decision-making process, decision trees are very interpretable. On the other hand, neural networks are frequently referred to as "black boxes" since it is challenging to comprehend the logic that underlies their predictions.

4. The amount of data needed: Small to medium-sized datasets with a reasonable number of characteristics are best for decision trees. Larger datasets and more features can be handled by neural networks, but they need a lot of training data to avoid overfitting.

5. Complexity: Decision trees tend to have less complex models than neural networks, which, depending on the situation, can be both a benefit and a drawback.