Predicting Red Wine Quality

```
#Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import matplotlib.style as style
import plotly.express as px
```

```
df = pd.read_csv('winequality-red.csv')
print(df.head())
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

This dataset contains some chemical properties and quality ratings of red wines.

- fixed acidity: Most acids involved with wine or fixed or nonvolatile
- volatile acidity: The amount of acetic acid in wine, which at too high of levels can lead to an unpleasant
- citric acid: Found in small quantities, citric acid can add 'freshness' and flavor to wines
- residual sugar: The amount of sugar remaining after fermentation stops
- chlorides: The amount of salt in the wine
- free sulfur dioxide: It prevents microbial growth and the oxidation of wine
- total sulfur dioxide: SO2 becomes evident in the nose and taste of wine
- density: The density of water is close to that of water depending on the percent alcohol and sugar content
- pH: How acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic)
- sulphates: A wine additive which can contribute to sulfur dioxide gas (S02) levels, which acts as an antimicrobial and antioxidant
- alcohol: The percent alcohol content of the wine
- quality: Output variable (based on sensory data, score between 0 and 10)

```
[5] print(df.shape)
```

The dataset consists of 1599 rows and 12 columns.

```
(1599, 12)
```

```
print(df.describe())
```

	fixed acidity	volatile acidity	citric acid	residual sugar	\
count	1599.000000	1599.000000	1599.000000	1599.000000	
mean	8.319637	0.527821	0.270976	2.538806	
std	1.741096	0.179060	0.194801	1.409928	
min	4.600000	0.120000	0.000000	0.900000	
25%	7.100000	0.390000	0.090000	1.900000	
50%	7.900000	0.520000	0.260000	2.200000	
75%	9.200000	0.640000	0.420000	2.600000	
max	15.900000	1.580000	1.000000	15.500000	

	chlorides	free sulfur dioxide	total sulfur dioxide	density	\
count	1599.000000	1599.000000	1599.000000	1599.000000	
mean	0.087467	15.874922	46.467792	0.996747	
std	0.047065	10.460157	32.895324	0.001887	
min	0.012000	1.000000	6.000000	0.990070	
25%	0.070000	7.000000	22.000000	0.995600	
50%	0.079000	14.000000	38.000000	0.996750	
75%	0.090000	21.000000	62.000000	0.997835	
max	0.611000	72.000000	289.000000	1.003690	

	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000
mean	3.311113	0.658149	10.422983	5.636023
std	0.154386	0.169507	1.065668	0.807569
min	2.740000	0.330000	8.400000	3.000000
25%	3.210000	0.550000	9.500000	5.000000
50%	3.310000	0.620000	10.200000	6.000000
75%	3.400000	0.730000	11.100000	6.000000
max	4.010000	2.000000	14.900000	8.000000

- For each column, 'count'(highlighted with green lines) shows the number of samples for all instances. We can see that each column has a value of 1599, indicating that there are no missing data.
- 'mean' (highlighted with red line), represents the average value for each column. For example, the mean value of 'fixed acidity' is 8.319637and the mean value of 'volatile acidity' is 0.527821.
- 'std' indicates the standard deviation for each column. Standard deviation measures the spread of values in a column. A higher standard deviation value indicates that the data shows more variability.
- min and max display the minimum and maximum values for each column. For instance, in the 'residual sugar' column (highlighted with blue line), the minimum value is 0.9 and the maximum value is 15.5.

```
[7]  print(df.info())
```

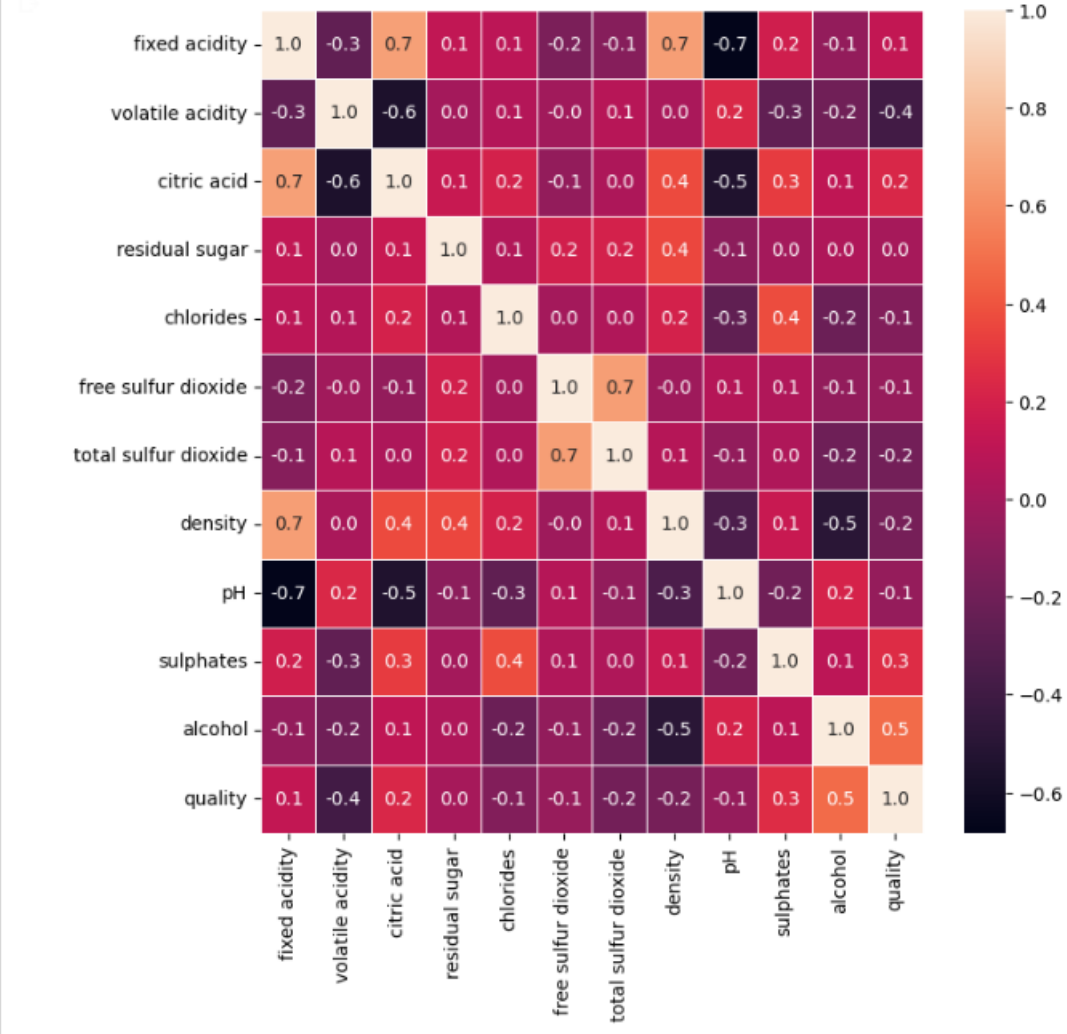
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 150.0 KB
None
```

While all columns except for the 'quality' column contain decimal numbers (float64), the 'quality' column contains integer values (int64).

What is the correlation matrix ? Why it is important ?

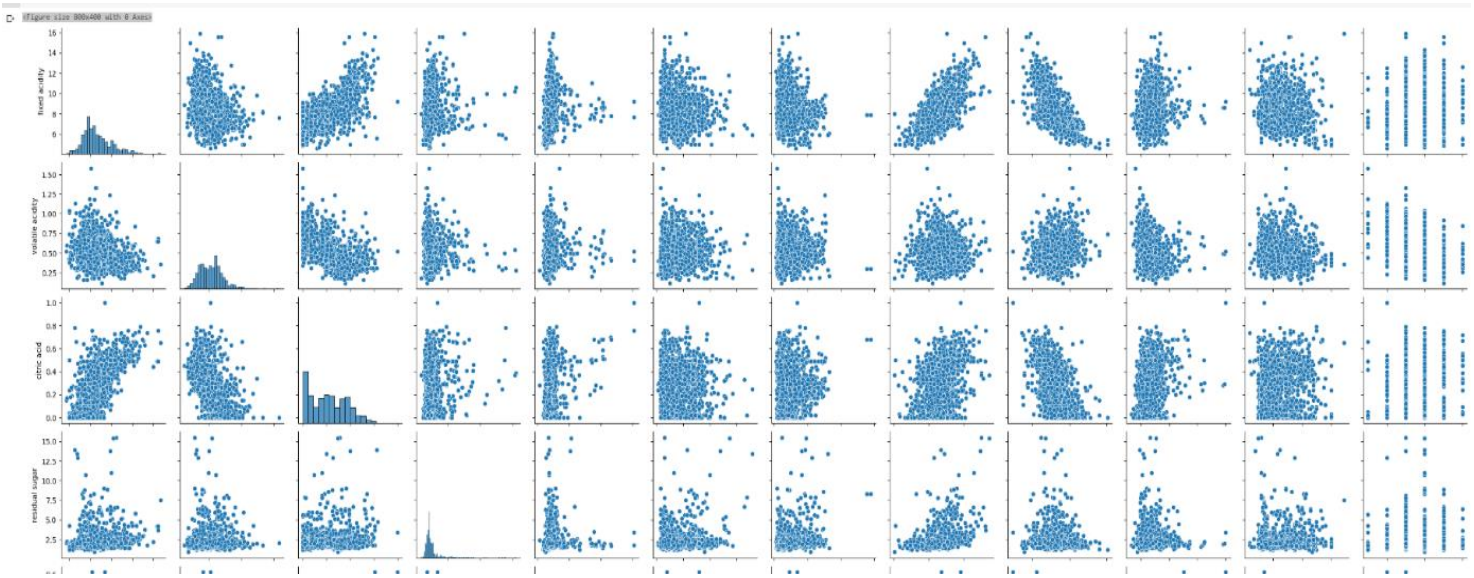
The correlation matrix measures the relationship between each column and the other columns. The correlation values between columns range from -1 to 1. A positive correlation value indicates a direct proportional relationship between two columns, while a negative correlation value indicates an inverse proportional relationship. A value of 0 indicates no relationship between the two columns.

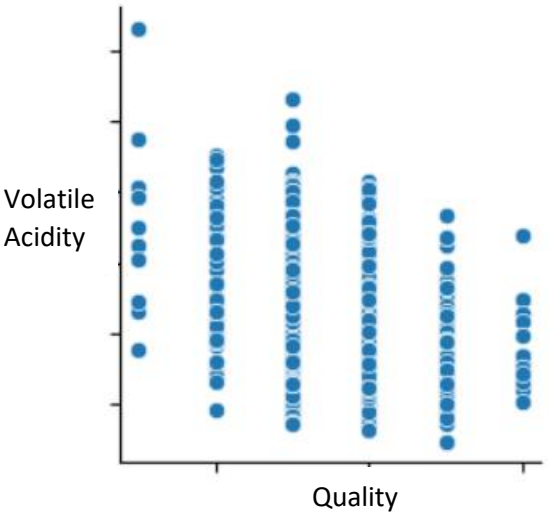
```
corr_matrix = df.corr()
fig, ax = plt.subplots(figsize = (8, 8))
sns.heatmap(corr_matrix, annot = True, fmt = '0.1f', linewidths = .5, ax = ax)
plt.show()
```



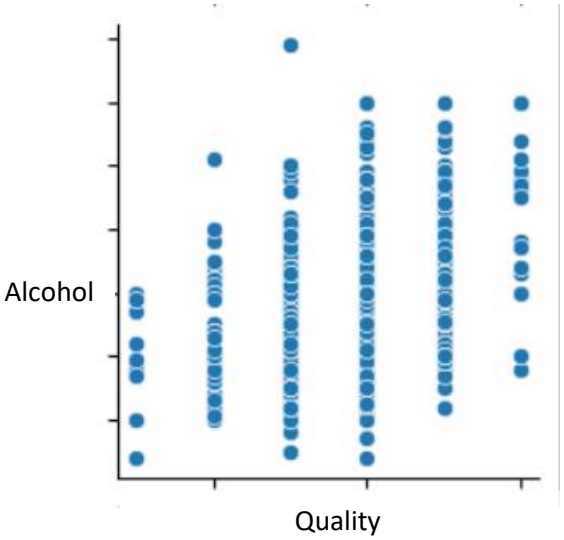
```
plt.figure(figsize = (12, 6))
sns.pairplot(df)
plt.show()
```

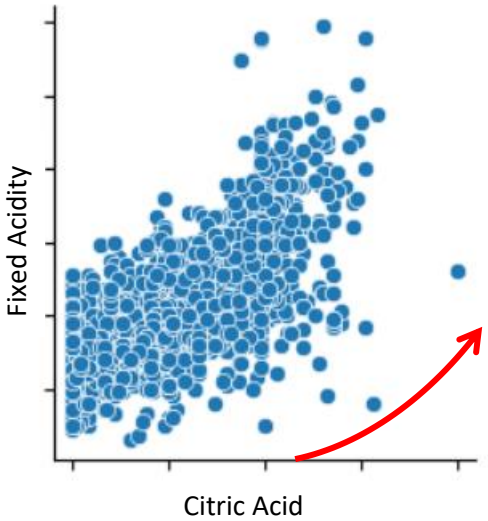
Below is a correlation plot where we can better understand the pairwise relationships between columns.



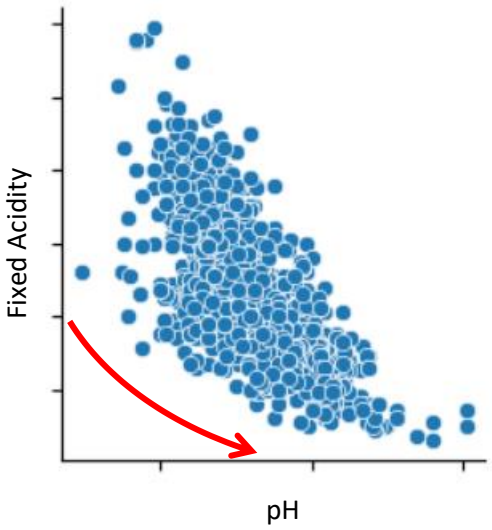


Focusing on the correlation of the "quality" column with other columns, the highest positive correlation value is observed with the "alcohol" column (0.476166), while the highest negative correlation value is observed with the "volatile acidity" column (-0.390558).





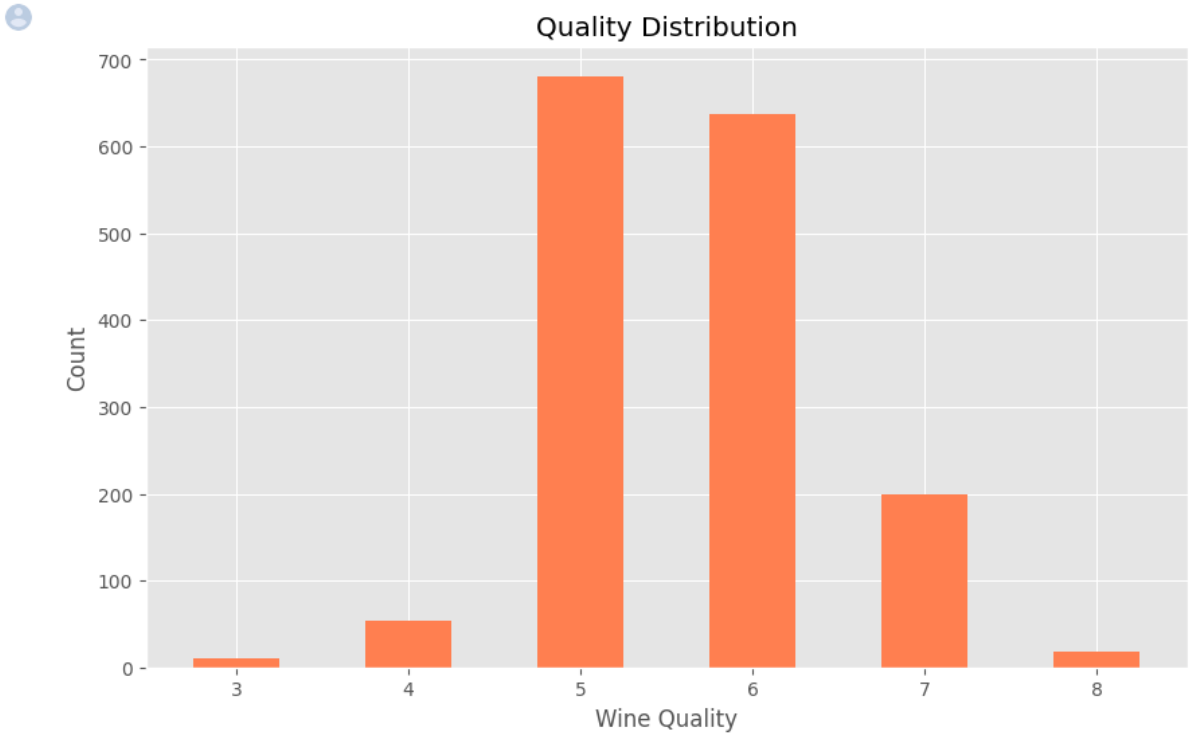
It is possible to observe a positive correlation (0.7) between the "fixed acidity" column and "citric acid," as well as a negative correlation (-0.7) between "fixed acidity" and "pH".



```
style.use('ggplot')

plt.figure(figsize=(10, 6))
quality_counts = df['quality'].value_counts().sort_index()
plt.bar(quality_counts.index, quality_counts.values, color='coral', width=0.5)
plt.xlabel('Wine Quality')
plt.ylabel('Count')
plt.title('Quality Distribution')

plt.show()
```



What is Imbalanced Classification Problem?

The imbalanced classification problem refers to a classification problem where there is an imbalance in the number of observations between classes due to numerical disparity. In this case, there is an imbalance between classes because one class contains more observations compared to the other class. Imbalanced classification problems can adversely affect classification models. For example, it can be difficult to correctly identify the minority class as the model may focus on the majority class. This can lead to false positive or false negative outcomes.

Different approaches can be used to address imbalanced classification problems, such as:

- **Undersampling or Oversampling**: Balancing the class distribution by reducing or increasing the number of observations in the minority class.
- **Class Weighting**: Addressing the imbalance by assigning higher weights to the minority class in the classification algorithm.
- **Synthetic Sampling**: Generating synthetic examples for the minority class to balance the class distribution.
- **Hierarchical Classification**: Dividing the class classification into sub-classes and then classifying the sub-classes to address the imbalance.

In this scenario, we will use the synthetic oversampling method called SMOTE (Synthetic Minority Over-sampling Technique). But what is SMOTE?

SMOTE (Synthetic Minority Over-sampling Technique) is a synthetic oversampling method used in the context of imbalanced classification problems. It aims to address the class imbalance by generating synthetic examples of the minority class. By creating synthetic samples, SMOTE increases the representation of the minority class in the dataset, thus improving the balance between classes.

Data Preprocessing

```
[9]  X = df.drop('quality', axis = 1)
     y = df['quality']
```

```
[10] y.value_counts()
```

```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

BEFORE SMOTE

```
[11] from imblearn.over_sampling import SMOTE

     smote = SMOTE()
     X, y = smote.fit_resample(X, y)
```

```
y.value_counts()
```

```
5    681
6    681
7    681
4    681
8    681
3    681
Name: quality, dtype: int64
```

AFTER SMOTE

Feature Scaling

```
[13] from sklearn.preprocessing import StandardScaler
     scaler = StandardScaler()
     X_scaled = scaler.fit_transform(X)
     X_scaled
```

```
array([[-0.59264956,  0.47411732, -1.35580918, ...,  1.20516195,
        -0.62410772, -1.13058303],
       [-0.34993289,  1.20215717, -1.35580918, ..., -0.82092168,
         0.13878675, -0.78434701],
       [-0.34993289,  0.71679727, -1.15835382, ..., -0.42877646,
        -0.05193687, -0.78434701],
       ...,
       [-0.03438045, -0.4448529 ,  0.62666741, ..., -0.91242968,
         0.74913133,  1.73276117],
       [-0.28443013, -0.17678032,  0.32779351, ..., -0.82048872,
         0.39182147,  2.06865968],
       [-0.61827491, -0.88398807,  0.13552941, ..., -0.57329206,
         0.34648009,  0.58231973]])
```

Model Training

```
# Classify function
from sklearn.model_selection import cross_val_score, train_test_split
def classify(model, X_scaled, y):
  x_train, x_test, y_train, y_test = train_test_split(X_scaled, y, test_size = 0.25, random_state = 7)
  #train the model
  model.fit(x_train, y_train)
  print('Accuracy: ', model.score(x_test, y_test) * 100)

  # cross_validation
  score = cross_val_score(model, X_scaled, y, cv = 5)
  print('Cross validation score: ', np.mean(score) * 100)
```

```
[98] # LogisticRegression
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
classify(model, X_scaled, y)
```

```
Accuracy:  60.861056751467714
Cross validation score:  56.38773256562114
```

```
[99] # DecisionTreeClassifier
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier()
classify(model, X_scaled, y)
```

```
Accuracy:  80.33268101761253
Cross validation score:  74.4007984366443
```

```
[100] # RandomForestClassifier
from sklearn.ensemble import RandomForestClassifier
model = RandomForestClassifier()
classify(model, X_scaled, y)
```

```
Accuracy:  87.08414872798434
Cross validation score:  80.71425365027396
```

```
# ExtraTreesClassifier
from sklearn.ensemble import ExtraTreesClassifier
model = ExtraTreesClassifier()
classify(model, X_scaled, y)
```

```
Accuracy:  88.06262230919765
Cross validation score:  82.13396258600103
```

```
[102] # LGBMClassifier
import lightgbm
model = lightgbm.LGBMClassifier()
classify(model, X_scaled, y)
```

```
Accuracy:  86.98630136986301
Cross validation score:  81.4736363282688
```

Model	Accuracy	Cross Validation Score
Logistic Regression	60.86	56.39
Decision Tree Classifier	80.33	74.40
Random Forest Classifier	87.08	80.71
Extra Trees Classifier	88.06	82.13
LGBM Classifier	86.99	81.47

As a result, the Extra Tree Classifier model provided us with the best outcome.

The Extra Tree Classifier is a machine learning algorithm that belongs to the family of ensemble methods and is based on decision trees. It is an extension of the Random Forest algorithm and shares many similarities with it.

The main idea behind the Extra Tree Classifier is to create an ensemble of decision trees and make predictions based on the majority vote of the individual trees. However, unlike Random Forests, which select the best split among a subset of features, the Extra Tree Classifier selects the splits randomly. This randomness leads to a higher level of diversity among the trees, which can potentially improve the model's generalization ability and reduce overfitting.

The name "Extra Trees" comes from the fact that the algorithm builds an "extra" number of trees compared to Random Forests. Each tree is grown using a random subset of the training data and a random subset of features. The final prediction is obtained by averaging the predictions of all the trees in the ensemble.

The Extra Tree Classifier is known for its fast training speed and good performance on a wide range of classification tasks. It can handle both numerical and categorical features and is less prone to overfitting compared to other algorithms. Additionally, it can handle imbalanced datasets reasonably well.

Overall, the Extra Tree Classifier is a powerful and versatile algorithm that can be used for various classification problems, making it a popular choice in machine learning.