Decision TreeClassifier on Wine-Quality-Dataset

Steps

1. Data

Data Profiling

Basic Operations

Data Cleaning

Statistical Analysis (Analysis of features)

2. EDA

Univariate Analysis

Multivariate Analysis(VIF)

3. Pre-processing

duplicate values handling

Null value handling

4. Model Building

i.Decision Tree

ii. GridSearchCV

5.Evaluation of the model

i. accuracy score

In [49]:

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

import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline

import scipy.stats as stats

import warnings
warnings.filterwarnings('ignore')
```

In [ ]:

```python
# Data Profiling
```

In [2]:

```python
df = pd.read_csv(r"https://raw.githubusercontent.com/shrikant-temburwar/Wine-Qual
```

In [3]:

```python
df
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
...	...	...	...	...	...	...	...	...	...	...	
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

1599 rows × 12 columns

In [ ]:

In [4]:
```
df.head()
```

Out[4]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4

In [43]:
```
df.tail()
```

Out[43]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
1593	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

In [44]:
```
#datatype & describe...
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1359 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1359 non-null   float64
 1   volatile acidity      1359 non-null   float64
 2   citric acid           1359 non-null   float64
 3   residual sugar        1359 non-null   float64
 4   chlorides             1359 non-null   float64
 5   free sulfur dioxide   1359 non-null   float64
 6   total sulfur dioxide  1359 non-null   float64
 7   density               1359 non-null   float64
 8   pH                    1359 non-null   float64
 9   sulphates             1359 non-null   float64
 10  alcohol               1359 non-null   float64
 11  quality               1359 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 138.0 KB
```

In [5]:
```
df['quality'].unique()
```
Out[5]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [6]:
```
len(df['quality'].unique())
```
Out[6]:
```
6
```

In [7]:
```
df['quality'].value_counts()
```
Out[7]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [45]:
```
#shape the df
df.shape
```
Out[45]:
```
(1359, 12)
```

In [46]:
```
#Columns of the df
df.columns
```
Out[46]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

EDA & Feature Engineering.....

In [47]:
```
numeric_features = [feature for feature in df.columns if df[feature].dtype != 'O'
categorical_features = [feature for feature in df.columns if df[feature].dtype ==
# print these feature...
print('We have {} numerical features : {}'.format(len(numeric_features), numeric_
print('\nWe have {} categorical features : {}'.format(len(categorical_features),
```

```
We have 12 numerical features : ['fixed acidity', 'volatile acidity', 'citric a
cid', 'residual sugar', 'chlorides', 'free sulfur dioxide', 'total sulfur dioxi
de', 'density', 'pH', 'sulphates', 'alcohol', 'quality']

We have 0 categorical features : []
```

Statistical Analysis.....

In [51]:
```
df.describe().T
```

Out[51]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1359.0	8.310596	1.736990	4.60000	7.1000	7.9000	9.20000	15.90000
volatile acidity	1359.0	0.529478	0.183031	0.12000	0.3900	0.5200	0.64000	1.58000
citric acid	1359.0	0.272333	0.195537	0.00000	0.0900	0.2600	0.43000	1.00000
residual sugar	1359.0	2.523400	1.352314	0.90000	1.9000	2.2000	2.60000	15.50000
chlorides	1359.0	0.088124	0.049377	0.01200	0.0700	0.0790	0.09100	0.61100
free sulfur dioxide	1359.0	15.893304	10.447270	1.00000	7.0000	14.0000	21.00000	72.00000
total sulfur dioxide	1359.0	46.825975	33.408946	6.00000	22.0000	38.0000	63.00000	289.00000
density	1359.0	0.996709	0.001869	0.99007	0.9956	0.9967	0.99782	1.00369
pH	1359.0	3.309787	0.155036	2.74000	3.2100	3.3100	3.40000	4.01000
sulphates	1359.0	0.658705	0.170667	0.33000	0.5500	0.6200	0.73000	2.00000
alcohol	1359.0	10.432315	1.082065	8.40000	9.5000	10.2000	11.10000	14.90000
quality	1359.0	5.623252	0.823578	3.00000	5.0000	6.0000	6.00000	8.00000

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
240
```

In [10]:
```
df = df.drop_duplicates()
```

In [11]:
```
df.drop_duplicates().sum()
```

Out[11]:
```
fixed acidity           11294.100000
volatile acidity          719.560000
citric acid               370.100000
residual sugar           3429.300000
chlorides                 119.760000
free sulfur dioxide     21599.000000
total sulfur dioxide    63636.500000
density                  1354.527460
pH                       4498.000000
sulphates                 895.180000
alcohol                 14177.516667
quality                  7642.000000
dtype: float64
```

In [12]:
```
#split the data
```

In [13]:
```
df.drop_duplicates()
```

Out[13]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9
...	...	...	...	...	...	...	...	...	...	...	
1593	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

1359 rows × 12 columns

In [14]:
```
df.duplicated().sum()
```

Out[14]:
```
0
```

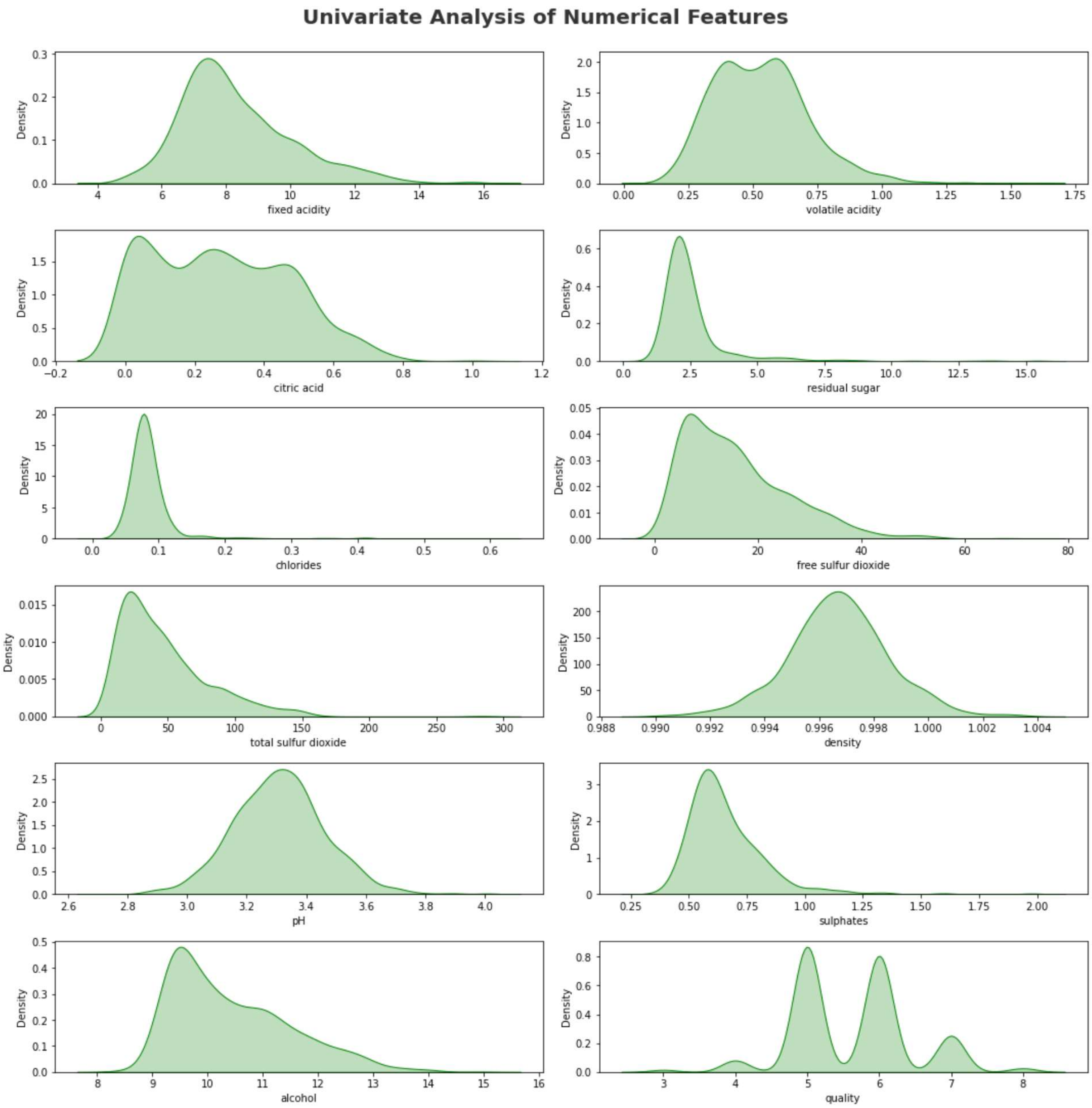
In [15]:
```
df.isnull().sum()
```

Out[15]:
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

Univariate Analysis....

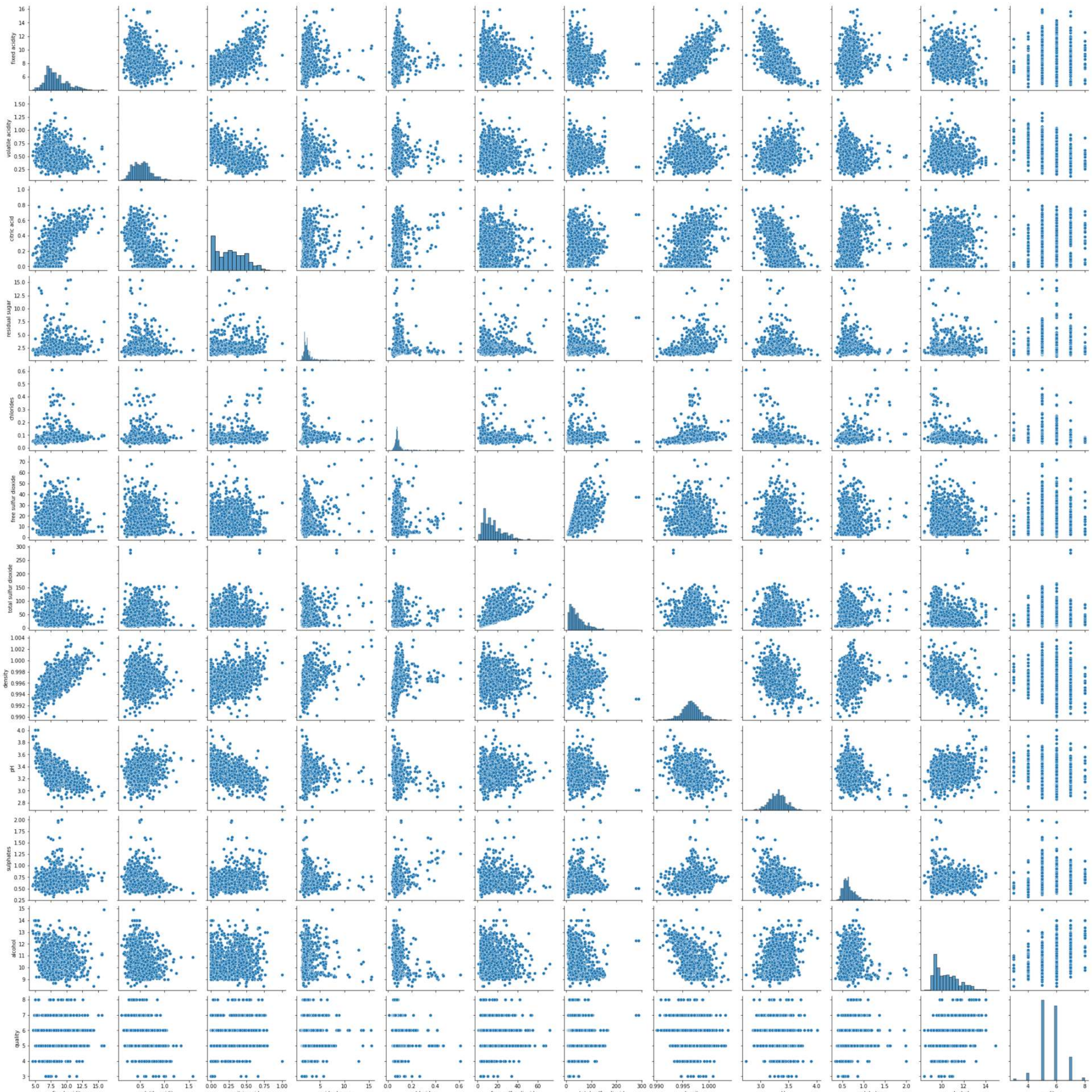
In [50]:
```
plt.figure(figsize=(15, 15))
plt.suptitle('Univariate Analysis of Numerical Features', fontsize=20, fontweight

for i in range(0, len(numeric_features)):
    plt.subplot(6,2, i+1)
    sns.kdeplot(x=df[numeric_features[i]],shade=True, color='g')
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



In [54]:
```
sns.pairplot(df)
```

Out[54]:
```
<seaborn.axisgrid.PairGrid at 0x20b0dfd95e0>
```

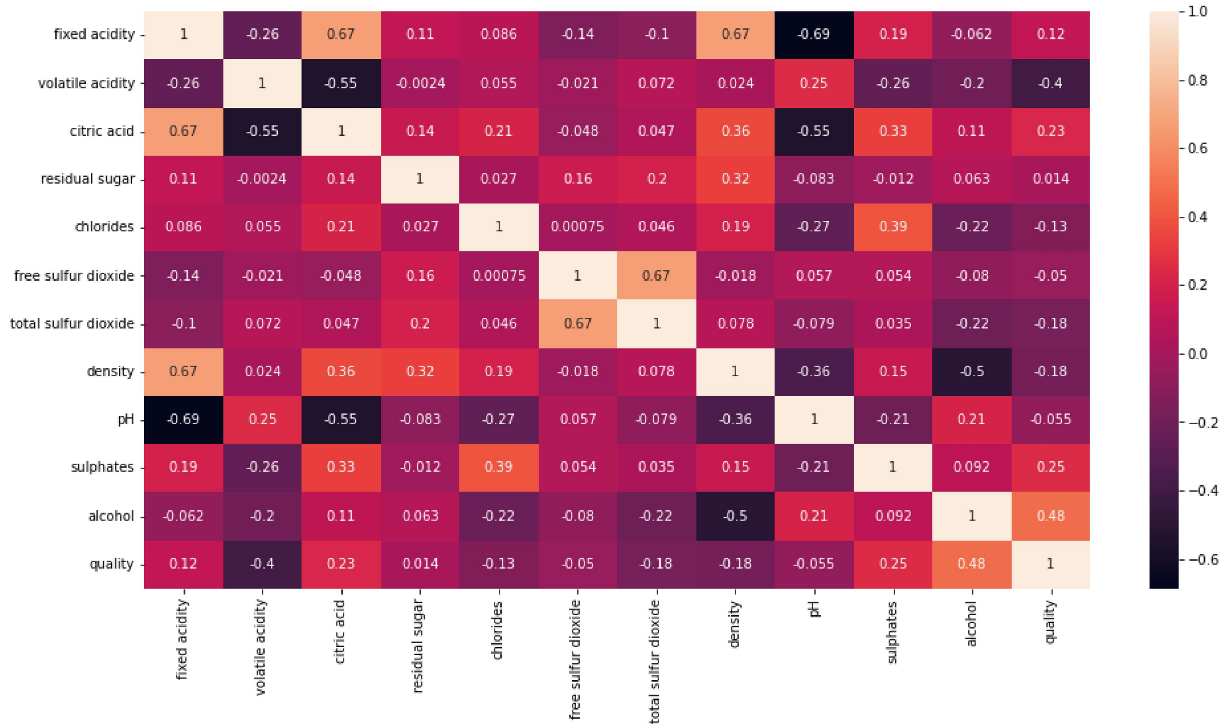


In [ ]:

multivariate analysis....

In [55]:
```
plt.figure(figsize=(16,8))
sns.heatmap(df[numeric_features].corr() ,annot=True)
```

Out[55]: <AxesSubplot:>



checking for multicollinearity....

In [56]:
```
from statsmodels.stats.outliers_influence import variance_inflation_factor
```

In [57]:
```
vif_data = pd.DataFrame()
vif_data['vif'] = [variance_inflation_factor(df[numeric_features].values,i) for i
vif_data['features'] = df[numeric_features].columns
vif_data
```

Out[57]:

	vif	features
0	75.023033	fixed acidity
1	17.387181	volatile acidity
2	9.195827	citric acid
3	4.915782	residual sugar
4	6.440176	chlorides
5	6.442192	free sulfur dioxide
6	6.601411	total sulfur dioxide
7	1547.276977	density
8	1102.707051	pH
9	22.810607	sulphates
10	146.378710	alcohol
11	74.885884	quality

split the data....

In [16]:
```
X = df.drop('quality',axis=1)
```

In [17]:
```
y = df['quality']
```

In [23]:
```
from sklearn.model_selection import train_test_split,GridSearchCV
```

In [24]:
```
X_train,X_test,y_train,y_test=train_test_split(X,y,test_size=0.33,random_state=1(
```

Decision Tree Classification Algorithm

Decision Tree is a Supervised learning technique that can be used for both classification and Regression problems, but mostly

it is preferred for solving Classification problems. It is a tree-structured classifier, where internal nodes represent the

features of a dataset, branches represent the decision rules and each leaf node represents the outcome.

Why use Decision Trees?

There are various algorithms in Machine learning, so choosing the best algorithm for the given dataset and problem is the main

point to remember while creating a machine learning model.

Decision Tree Terminologies

.Root Node:

Root node is from where the decision tree starts. It represents the entire dataset, which further gets divided into

two or more homogeneous sets.

.Leaf Node:

Leaf nodes are the final output node, and the tree cannot be segregated further after getting a leaf node.

.Splitting:

Splitting is the process of dividing the decision node/root node into sub-nodes according to the given conditions.

.Branch/Sub Tree:

A tree formed by splitting the tree.

.Pruning:

Pruning is the process of removing the unwanted branches from the tree.

.Parent/Child node:

The root node of the tree is called the parent node, and other nodes are called the child nodes.

Decision Tree classifier....

In [ ]:

In [25]:
```
from sklearn.tree import DecisionTreeClassifier
model=DecisionTreeClassifier()
```

In [26]:
```
model.fit(X_train,y_train)
```
Out[26]:
```
DecisionTreeClassifier()
```

In [27]:
```
model.score(X_train,y_train)
```
Out[27]:
```
1.0
```

In [28]:
```
y_predict=model.predict(X_test)
```

In [29]:
```
from sklearn.metrics import accuracy_score
```

In [31]:
```
accuracy_score(y_test,y_predict)
```
Out[31]:
```
0.48329621380846327
```

In [ ]:
```
# observation -> train data accuracy is more than test data acccuracy , the model
```

Implementing GridSearchCV....

What is Grid Search?

Grid search is a technique for tuning hyperparameter that may facilitate build a model and evaluate a model for every

combination of algorithms parameters per grid. We might use 10 fold cross-validation to search the best value for that tuning

hyperparameter. Parameters like in decision criterion, max_depth, min_sample_split, etc. These values are called

hyperparameters. To get the simplest set of hyperparameters we will use the Grid Search method. In the Grid Search, all the

mixtures of hyperparameters combinations will pass through one by one into the model and check the score on each model. It

gives us the set of hyperparameters which gives the best score. Scikit-learn package as a means of automatically iterating over

these hyperparameters using cross-validation. This method is called Grid Search.

In [34]:
```
grid_param = {
    'criterion': ['gini', 'entropy'],
    'max_depth' : range(2,32,1),
    'min_samples_leaf' : range(1,10,1),
    'min_samples_split': range(2,10,1),
    'splitter' : ['best', 'random'],
}
```

In [35]:
```
from sklearn.model_selection import GridSearchCV
grid_searh=GridSearchCV(estimator=model,param_grid=grid_param,cv=5)
```

In [36]:
```
grid_searh.fit(X_train,y_train)
```

Out[36]:
```
GridSearchCV(cv=5, estimator=DecisionTreeClassifier(),
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(2, 32),
                         'min_samples_leaf': range(1, 10),
                         'min_samples_split': range(2, 10),
                         'splitter': ['best', 'random']})
```

In [37]:
```
grid_searh.best_params_
```

Out[37]:
```
{'criterion': 'gini',
 'max_depth': 9,
 'min_samples_leaf': 7,
 'min_samples_split': 3,
 'splitter': 'random'}
```

In [38]:
```
l_with_best_params=DecisionTreeClassifier(criterion= 'entropy',max_depth= 28,min_s
```

In [39]:
```
model_with_best_params.fit(X_train,y_train)
```

Out[39]:
```
DecisionTreeClassifier(criterion='entropy', max_depth=28, min_samples_leaf=9,
                       splitter='random')
```

visualize model

In [42]:
```
from sklearn import tree
import matplotlib.pyplot as plt
fig=plt.figure(figsize=(25,15))
tree.plot_tree(model_with_best_params,filled=True,fontsize=10)
```

```
samples = 26\nvalue = [0, 0, 7, 16, 3, 0]'),

 Text(0.5675675675675675, 0.10714285714285714, 'entropy = 1.096\nsamples = 11
\nvalue = [0, 0, 2, 8, 1, 0]'),
 Text(0.5945945945945946, 0.10714285714285714, 'entropy = 1.4\nsamples = 15\n
value = [0, 0, 5, 8, 2, 0]'),
 Text(0.6081081081081081, 0.32142857142857145, 'X[10] <= 9.94\nentropy = 1.56
8\nsamples = 18\nvalue = [1, 0, 5, 10, 2, 0]'),
 Text(0.5945945945945946, 0.25, 'entropy = 1.658\nsamples = 9\nvalue = [1, 0,
2, 5, 1, 0]'),
 Text(0.6216216216216216, 0.25, 'entropy = 1.352\nsamples = 9\nvalue = [0, 0,
3, 5, 1, 0]'),
 Text(0.6891891891891891, 0.39285714285714285, 'X[2] <= 0.107\nentropy = 1.47
1\nsamples = 89\nvalue = [0, 2, 25, 51, 11, 0]'),
 Text(0.6621621621621622, 0.32142857142857145, 'X[2] <= 0.062\nentropy = 1.46
1\nsamples = 24\nvalue = [0, 1, 3, 15, 5, 0]'),
 Text(0.6486486486486487, 0.25, 'entropy = 1.242\nsamples = 15\nvalue = [0,
0, 2, 10, 3, 0]'),
 Text(0.6756756756756757, 0.25, 'entropy = 1.658\nsamples = 9\nvalue = [0, 1,
1, 5, 2, 0]'),
```

In [40]:
```
y_prediction2=model_with_best_params.predict(X_test)
```

In [41]:
```
accuracy_score(y_test,y_prediction2)
```

Out[41]:
```
0.5278396436525612
```

In [ ]:
```
# observation -> here the accuracy is increase after aplying hyperparameter tunir
```

▪

In [ ]:
```

```