IMPORTING THE LIBRARIES

In [20]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

READ THE DATASET

In [21]:
```
df=pd.read_csv('winequalityN.csv')
```

DATA ANALYSIS

In [22]:
```
df.head()
```

Out[22]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.27	0.36	20.7	0.045	45.0	170.0	1.0010	3.00	0.45	8.8	6
1	white	6.3	0.30	0.34	1.6	0.049	14.0	132.0	0.9940	3.30	0.49	9.5	6
2	white	8.1	0.28	0.40	6.9	0.050	30.0	97.0	0.9951	3.26	0.44	10.1	6
3	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6
4	white	7.2	0.23	0.32	8.5	0.058	47.0	186.0	0.9956	3.19	0.40	9.9	6

In [23]:
```
df.tail()
```

Out[23]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	NaN	11.2	6
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

In [24]:
```
df.columns
```

Out[24]:
```
Index(['type', 'fixed acidity', 'volatile acidity', 'citric acid',
       'residual sugar', 'chlorides', 'free sulfur dioxide',
       'total sulfur dioxide', 'density', 'pH', 'sulphates', 'alcohol',
       'quality'],
      dtype='object')
```

In [25]:
```
df.shape
```

Out[25]: (6497, 13)

In [26]:
```
df.describe()
```

Out[26]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	6487.000000	6489.000000	6494.000000	6495.000000	6495.000000	6497.000000	6497.000000	6497.000000	6488.000000	6493.000000	6497.000000	6497.000000
mean	7.216579	0.339691	0.318722	5.444326	0.056042	30.525319	115.744574	0.994697	3.218395	0.531215	10.491801	5.818378
std	1.296750	0.164649	0.145265	4.758125	0.035036	17.749400	56.521855	0.002999	0.160748	0.148814	1.192712	0.873255
min	3.800000	0.080000	0.000000	0.600000	0.009000	1.000000	6.000000	0.987110	2.720000	0.220000	8.000000	3.000000
25%	6.400000	0.230000	0.250000	1.800000	0.038000	17.000000	77.000000	0.992340	3.110000	0.430000	9.500000	5.000000
50%	7.000000	0.290000	0.310000	3.000000	0.047000	29.000000	118.000000	0.994890	3.210000	0.510000	10.300000	6.000000
75%	7.700000	0.400000	0.390000	8.100000	0.065000	41.000000	156.000000	0.996990	3.320000	0.600000	11.300000	6.000000
max	15.900000	1.580000	1.660000	65.800000	0.611000	289.000000	440.000000	1.038980	4.010000	2.000000	14.900000	9.000000

In [27]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   type                  6497 non-null   object 
 1   fixed acidity         6487 non-null   float64
 2   volatile acidity      6489 non-null   float64
 3   citric acid           6494 non-null   float64
 4   residual sugar        6495 non-null   float64
 5   chlorides             6495 non-null   float64
 6   free sulfur dioxide   6497 non-null   float64
 7   total sulfur dioxide  6497 non-null   float64
 8   density               6497 non-null   float64
 9   pH                    6488 non-null   float64
 10  sulphates             6493 non-null   float64
 11  alcohol               6497 non-null   float64
 12  quality               6497 non-null   int64  
dtypes: float64(11), int64(1), object(1)
memory usage: 660.0+ KB
```

In [28]:
```
df.isnull().sum()
```

Out[28]:
```
type                     0
fixed acidity           10
volatile acidity         8
citric acid              3
residual sugar           2
chlorides                2
free sulfur dioxide      0
total sulfur dioxide     0
density                  0
pH                       9
sulphates                4
alcohol                  0
quality                  0
dtype: int64
```

REPLACING THE NULL VALUES

In [29]:
```
df.fillna(method='ffill',inplace=True)
```

In [30]:
```
df.isnull().sum()
```

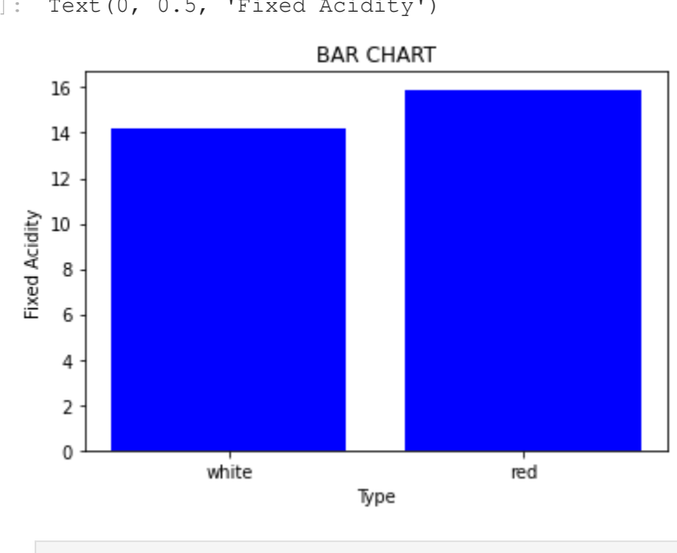
Out[30]:
```
type                    0
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

DATA VISUALIZATION

MATPLOTLIB

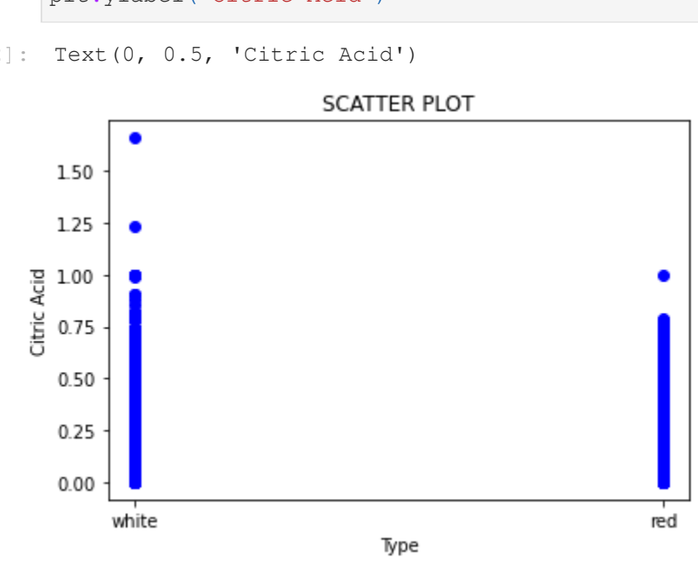
In [55]:
```
plt.bar(df['type'],df['fixed acidity'],color='b')
plt.title("BAR CHART")
plt.xlabel('Type')
plt.ylabel("Fixed Acidity")
```

Out[55]: Text(0, 0.5, 'Fixed Acidity')



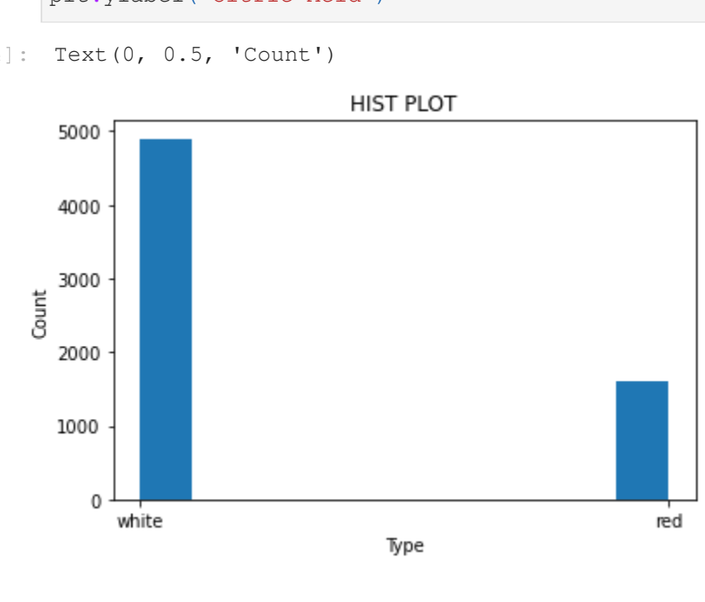
In [52]:
```
plt.scatter(df['type'],df['citric acid'],color='b')
plt.title("SCATTER PLOT")
plt.xlabel('Type')
plt.ylabel("Citric Acid")
```

Out[52]: Text(0, 0.5, 'Citric Acid')



In [54]:
```
plt.hist(df['type'])
plt.title("HIST PLOT")
plt.xlabel('Type')
plt.ylabel("Citric Acid")
```

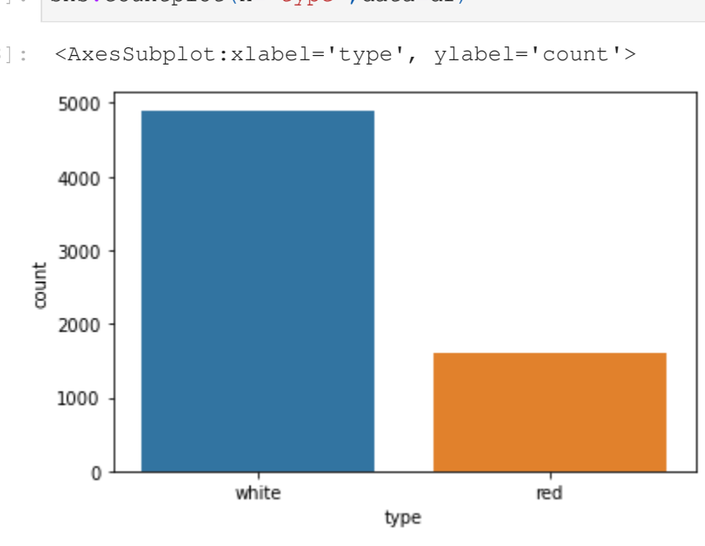
Out[54]: Text(0, 0.5, 'Count')



SEABORN

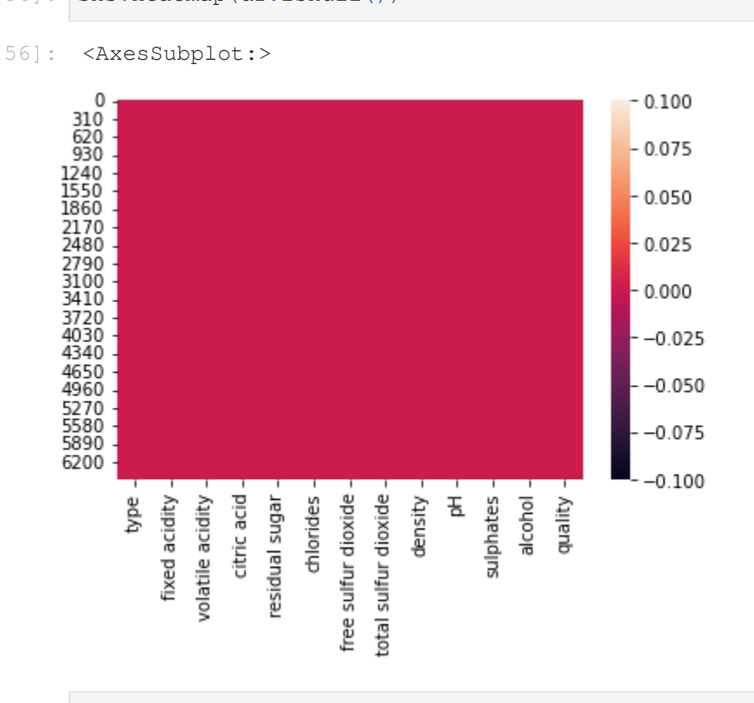
In [48]:
```
sns.countplot(x="type",data=df)
```

Out[48]: <AxesSubplot:xlabel='type', ylabel='count'>



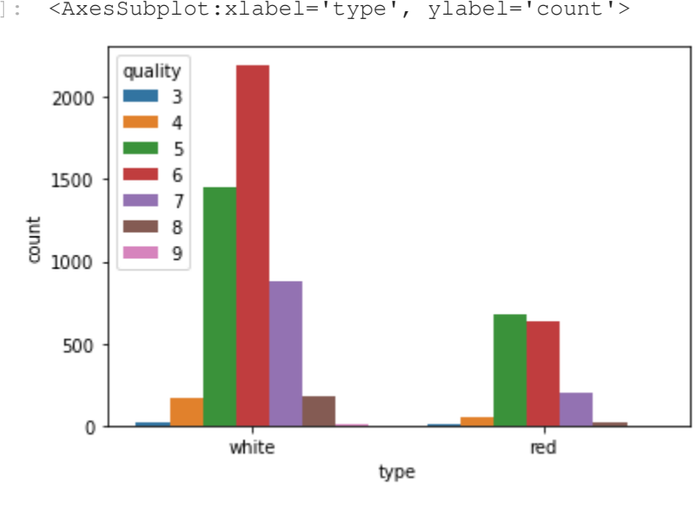
In [56]:
```
sns.heatmap(df.isnull())
```

Out[56]: <AxesSubplot:>



In [57]:
```
sns.countplot(x="type",hue="quality",data=df)
```

Out[57]: <AxesSubplot:xlabel='type', ylabel='count'>



MACHINE LEARNING

CLASSIFICATION ALGORITHM

CONVERTING STRING DATA INTO NUMERIC VALUES

In [58]:
```
df
```

Out[58]:

	type	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	white	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	white	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	white	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	white	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	red	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6493	red	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.58	11.2	6
6494	red	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	red	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	red	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6497 rows × 13 columns

In [61]:
```
a=pd.get_dummies(df['type'],drop_first=True)
```

In [62]:
```
a
```

Out[62]:

	white
0	1
1	1
2	1
3	1
4	1
...	...
6492	0
6493	0
6494	0
6495	0
6496	0

6497 rows × 1 columns

In [63]:
```
df.drop(columns='type',inplace=True)
```

In [64]:
```
df
```

Out[64]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6
...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
6493	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.58	11.2	6
6494	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
6495	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
6496	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

6497 rows × 12 columns

In [70]:
```
df1=pd.concat([df,a],axis=1)
```

In [71]:
```
df1
```

Out[71]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality	white
0	7.0	0.270	0.36	20.7	0.045	45.0	170.0	1.00100	3.00	0.45	8.8	6	1
1	6.3	0.300	0.34	1.6	0.049	14.0	132.0	0.99400	3.30	0.49	9.5	6	1
2	8.1	0.280	0.40	6.9	0.050	30.0	97.0	0.99510	3.26	0.44	10.1	6	1
3	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6	1
4	7.2	0.230	0.32	8.5	0.058	47.0	186.0	0.99560	3.19	0.40	9.9	6	1
...	...	...	...	...	...	...	...	...	...	...	...	...	...
6492	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5	0
6493	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.58	11.2	6	0
6494	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6	0
6495	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5	0
6496	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6	0

6497 rows × 13 columns

FEATURE SCALING

In [72]:
```
from sklearn.preprocessing import StandardScaler
```

In [73]:
```
b=StandardScaler()
```

In [74]:
```
b.fit(df1)
```

Out[74]: StandardScaler()

SPLIT THE DATA

In [81]:
```
x=df.values
x
```

Out[81]:
```
array([[ 7.   ,  0.27 ,  0.36 , ...,  0.45 ,  8.8  ,  6.   ],
       [ 6.3  ,  0.3  ,  0.34 , ...,  0.49 ,  9.5  ,  6.   ],
       [ 8.1  ,  0.28 ,  0.4  , ...,  0.44 , 10.1  ,  6.   ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  0.75 , 11.   ,  6.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  0.71 , 10.2  ,  5.   ],
       [ 6.   ,  0.31 ,  0.47 , ...,  0.66 , 11.   ,  6.   ]])
```

In [82]:
```
y=a.values
y
```

Out[82]:
```
array([[1],
       [1],
       [1],
       ...,
       [0],
       [0],
       [0]], dtype=uint8)
```

TRAIN AND TEST

In [83]:
```
from sklearn.model_selection import train_test_split
```

In [84]:
```
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.3,random_state=1)
```

LOGISTIC REGRESSION

In [86]:
```
from sklearn.linear_model import LogisticRegression
reg=LogisticRegression()
```

In [87]:
```
reg.fit(x_train,y_train)
```

```
C:\Users\Sharathsai\anaconda3\lib\site-packages\sklearn\utils\validation.py:993: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples, ), for example using rav
el().
  y = column_or_1d(y, warn=True)
C:\Users\Sharathsai\anaconda3\lib\site-packages\sklearn\linear_model\_logistic.py:814: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
```

Out[87]: LogisticRegression()

In [88]:
```
y_pred=reg.predict(x_test)
y_pred
```

Out[88]: array([1, 0, 1, ..., 0, 1, 1], dtype=uint8)

In [89]:
```
y_test
```

Out[89]:
```
array([[1],
       [0],
       [1],
       ...,
       [0],
       [1],
       [1]], dtype=uint8)
```

In [90]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [91]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[91]:
```
array([[ 452,   24],
       [  12, 1462]], dtype=int64)
```

In [94]:
```
acc=accuracy_score(y_test,y_pred)
acc*100
```

Out[94]: 98.15384615384616

DECISION TREE

In [104]:
```
from sklearn.tree import *
reg=DecisionTreeClassifier()
```

In [105]:
```
reg.fit(x_train,y_train)
```

Out[105]: DecisionTreeClassifier()

In [106]:
```
y_pred=reg.predict(x_test)
y_pred
```

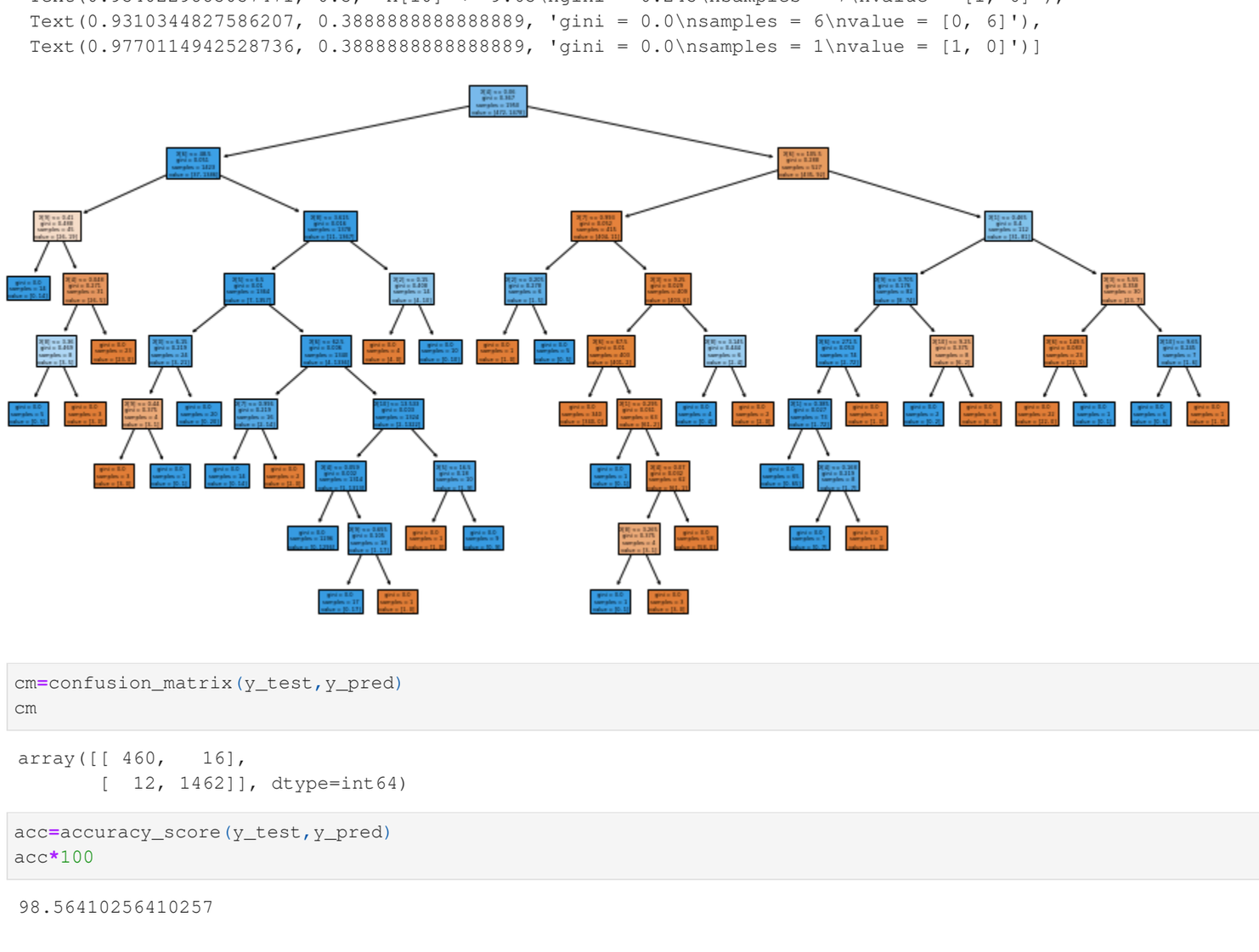
Out[106]: array([1, 0, 1, ..., 0, 1, 1], dtype=uint8)

In [107]:
```
y_test
```

Out[107]:
```
array([[1],
       [0],
       [1],
       ...,
       [0],
       [1],
       [1]], dtype=uint8)
```

In [108]:
```
d=reg.fit(x_test,y_pred)
plt.figure(figsize=(15,7))
plot_tree(d,filled=True)
```

Out[108]:
```
[Text(0.40301724137931033, 0.9444444444444444, 'X[4] <= 0.06\ngini = 0.367\nsamples = 1950\nvalue = [472, 1478]'),
 Text(0.15660919540229884, 0.8333333333333334, 'X[6] <= 48.5\ngini = 0.051\nsamples = 1423\nvalue = [37, 1386]'),
 Text(0.04597701149425287, 0.7222222222222222, 'X[9] <= 0.41\ngini = 0.488\nsamples = 45\nvalue = [26, 19]'),
 Text(0.022988505747126436, 0.6111111111111112, 'gini = 0.0\nsamples = 14\nvalue = [0, 14]'),
 Text(0.06896551724137931, 0.6111111111111112, 'X[4] <= 0.048\ngini = 0.271\nsamples = 31\nvalue = [26, 5]'),
 Text(0.04597701149425287, 0.5, 'X[8] <= 3.36\ngini = 0.469\nsamples = 8\nvalue = [3, 5]'),
 Text(0.022988505747126436, 0.3888888888888889, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.06896551724137931, 0.3888888888888889, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.09195402298850575, 0.5, 'gini = 0.0\nsamples = 23\nvalue = [23, 0]'),
 Text(0.2672413793103448, 0.7222222222222222, 'X[8] <= 3.615\ngini = 0.016\nsamples = 1378\nvalue = [11, 1367]'),
 Text(0.20114942528735633, 0.6111111111111112, 'X[5] <= 6.5\ngini = 0.01\nsamples = 1364\nvalue = [7, 1357]'),
 Text(0.13793103448275862, 0.5, 'X[0] <= 6.15\ngini = 0.219\nsamples = 24\nvalue = [3, 21]'),
 Text(0.11494252873563218, 0.3888888888888889, 'X[9] <= 0.44\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.09195402298850575, 0.2777777777777778, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.13793103448275862, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.16091954022988506, 0.3888888888888889, 'gini = 0.0\nsamples = 20\nvalue = [0, 20]'),
 Text(0.26436781609195403, 0.5, 'X[6] <= 62.5\ngini = 0.006\nsamples = 1340\nvalue = [4, 1336]'),
 Text(0.20689655172413793, 0.3888888888888889, 'X[7] <= 0.996\ngini = 0.219\nsamples = 16\nvalue = [2, 14]'),
 Text(0.1839080459770115, 0.2777777777777778, 'gini = 0.0\nsamples = 14\nvalue = [0, 14]'),
 Text(0.22988505747126436, 0.2777777777777778, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.3218390804597701, 0.3888888888888889, 'X[10] <= 13.533\ngini = 0.003\nsamples = 1324\nvalue = [2, 1322]'),
 Text(0.27586206896551724, 0.2777777777777778, 'X[4] <= 0.059\ngini = 0.002\nsamples = 1314\nvalue = [1, 1313]'),
 Text(0.25287356321839083, 0.16666666666666666, 'gini = 0.0\nsamples = 1296\nvalue = [0, 1296]'),
 Text(0.2988505747126437, 0.16666666666666666, 'X[9] <= 0.655\ngini = 0.105\nsamples = 18\nvalue = [1, 17]'),
 Text(0.27586206896551724, 0.05555555555555555, 'gini = 0.0\nsamples = 17\nvalue = [0, 17]'),
 Text(0.3218390804597701, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.367816091954023, 0.2777777777777778, 'X[5] <= 16.5\ngini = 0.18\nsamples = 10\nvalue = [1, 9]'),
 Text(0.3448275862068966, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.39080459770114945, 0.16666666666666666, 'gini = 0.0\nsamples = 9\nvalue = [0, 9]'),
 Text(0.3333333333333333, 0.6111111111111112, 'X[2] <= 0.15\ngini = 0.408\nsamples = 14\nvalue = [4, 10]'),
 Text(0.3103448275862069, 0.5, 'gini = 0.0\nsamples = 4\nvalue = [4, 0]'),
 Text(0.3563218390804598, 0.5, 'gini = 0.0\nsamples = 10\nvalue = [0, 10]'),
 Text(0.6494252873563219, 0.8333333333333334, 'X[6] <= 105.5\ngini = 0.288\nsamples = 527\nvalue = [435, 92]'),
 Text(0.4827586206896552, 0.7222222222222222, 'X[7] <= 0.993\ngini = 0.052\nsamples = 415\nvalue = [404, 11]'),
 Text(0.42528735632183906, 0.6111111111111112, 'X[2] <= 0.205\ngini = 0.278\nsamples = 6\nvalue = [1, 5]'),
 Text(0.40229885057471265, 0.5, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.4482758620689655, 0.5, 'gini = 0.0\nsamples = 5\nvalue = [0, 5]'),
 Text(0.5402298850574713, 0.6111111111111112, 'X[3] <= 9.25\ngini = 0.029\nsamples = 409\nvalue = [403, 6]'),
 Text(0.4942528735632184, 0.5, 'X[6] <= 67.5\ngini = 0.01\nsamples = 403\nvalue = [401, 2]'),
 Text(0.47126436781609193, 0.3888888888888889, 'gini = 0.0\nsamples = 340\nvalue = [340, 0]'),
 Text(0.5172413793103449, 0.3888888888888889, 'X[1] <= 0.295\ngini = 0.061\nsamples = 63\nvalue = [61, 2]'),
 Text(0.4942528735632184, 0.2777777777777778, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5402298850574713, 0.2777777777777778, 'X[4] <= 0.07\ngini = 0.032\nsamples = 62\nvalue = [61, 1]'),
 Text(0.5172413793103449, 0.16666666666666666, 'X[8] <= 3.265\ngini = 0.375\nsamples = 4\nvalue = [3, 1]'),
 Text(0.4942528735632184, 0.05555555555555555, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.5402298850574713, 0.05555555555555555, 'gini = 0.0\nsamples = 3\nvalue = [3, 0]'),
 Text(0.5632183908045977, 0.16666666666666666, 'gini = 0.0\nsamples = 58\nvalue = [58, 0]'),
 Text(0.5862068965517241, 0.5, 'X[8] <= 3.145\ngini = 0.444\nsamples = 6\nvalue = [2, 4]'),
 Text(0.5632183908045977, 0.3888888888888889, 'gini = 0.0\nsamples = 4\nvalue = [0, 4]'),
 Text(0.6091954022988506, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue = [2, 0]'),
 Text(0.8160919540229885, 0.7222222222222222, 'X[1] <= 0.465\ngini = 0.4\nsamples = 112\nvalue = [31, 81]'),
 Text(0.7241379310344828, 0.6111111111111112, 'X[9] <= 0.705\ngini = 0.176\nsamples = 82\nvalue = [8, 74]'),
 Text(0.6781609195402298, 0.5, 'X[6] <= 271.5\ngini = 0.053\nsamples = 74\nvalue = [2, 72]'),
 Text(0.6551724137931034, 0.3888888888888889, 'X[1] <= 0.385\ngini = 0.027\nsamples = 73\nvalue = [1, 72]'),
 Text(0.632183908045977, 0.2777777777777778, 'gini = 0.0\nsamples = 65\nvalue = [0, 65]'),
 Text(0.6781609195402298, 0.2777777777777778, 'X[4] <= 0.168\ngini = 0.219\nsamples = 8\nvalue = [1, 7]'),
 Text(0.6551724137931034, 0.16666666666666666, 'gini = 0.0\nsamples = 7\nvalue = [0, 7]'),
 Text(0.7011494252873564, 0.16666666666666666, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7011494252873564, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]'),
 Text(0.7701149425287356, 0.5, 'X[10] <= 9.25\ngini = 0.375\nsamples = 8\nvalue = [6, 2]'),
 Text(0.7471264367816092, 0.3888888888888889, 'gini = 0.0\nsamples = 2\nvalue = [0, 2]'),
 Text(0.7931034482758621, 0.3888888888888889, 'gini = 0.0\nsamples = 6\nvalue = [6, 0]'),
 Text(0.9080459770114943, 0.6111111111111112, 'X[3] <= 5.55\ngini = 0.358\nsamples = 30\nvalue = [23, 7]'),
 Text(0.8620689655172413, 0.5, 'X[6] <= 149.5\ngini = 0.083\nsamples = 23\nvalue = [22, 1]'),
 Text(0.8390804597701149, 0.3888888888888889, 'gini = 0.0\nsamples = 22\nvalue = [22, 0]'),
 Text(0.8850574712643678, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [0, 1]'),
 Text(0.9540229885057471, 0.5, 'X[10] <= 9.65\ngini = 0.245\nsamples = 7\nvalue = [1, 6]'),
 Text(0.9310344827586207, 0.3888888888888889, 'gini = 0.0\nsamples = 6\nvalue = [0, 6]'),
 Text(0.9770114942528736, 0.3888888888888889, 'gini = 0.0\nsamples = 1\nvalue = [1, 0]')]
```



In [109]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[109]:
```
array([[ 460,   16],
       [  12, 1462]], dtype=int64)
```

In [110]:
```
acc=accuracy_score(y_test,y_pred)
acc*100
```

Out[110]: 98.56410256410257

RANDOM FOREST

In [111]:
```
from sklearn.ensemble import RandomForestClassifier
```

In [120]:
```
reg=RandomForestClassifier(n_estimators=30)
```

In [121]:
```
reg.fit(x_train,y_train)
```

```
C:\Users\Sharathsai\AppData\Local\Temp\ipykernel_11864\310678519.py:1: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example using ravel().
  reg.fit(x_train,y_train)
```

Out[121]: RandomForestClassifier(n_estimators=30)

In [122]:
```
y_pred=reg.predict(x_test)
y_pred
```

Out[122]: array([1, 0, 1, ..., 0, 1, 1], dtype=uint8)

In [123]:
```
y_test
```

Out[123]:
```
array([[1],
       [0],
       [1],
       ...,
       [0],
       [1],
       [1]], dtype=uint8)
```

In [124]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[124]:
```
array([[ 470,    6],
       [   2, 1472]], dtype=int64)
```

In [125]:
```
acc=accuracy_score(y_test,y_pred)
acc*100
```

Out[125]: 99.58974358974359

K Nearest Neighbors

In [126]:
```
from sklearn.neighbors import KNeighborsClassifier
```

In [141]:
```
reg=KNeighborsClassifier(n_neighbors=5)
```

In [142]:
```
reg.fit(x_train,y_train)
```

```
C:\Users\Sharathsai\anaconda3\lib\site-packages\sklearn\neighbors\_classification.py:198: DataConversionWarning: A column-vector y was passed when a 1d array was expected. Please change the shape of y to (n_samples,), for example u
sing ravel().
  return self._fit(X, y)
```

Out[142]: KNeighborsClassifier()

In [143]:
```
y_pred=reg.predict(x_test)
y_pred
```

Out[143]: array([1, 0, 1, ..., 0, 1, 0], dtype=uint8)

In [144]:
```
y_test
```

Out[144]:
```
array([[1],
       [0],
       [1],
       ...,
       [0],
       [1],
       [1]], dtype=uint8)
```

In [145]:
```
from sklearn.metrics import confusion_matrix,accuracy_score
```

In [146]:
```
cm=confusion_matrix(y_test,y_pred)
cm
```

Out[146]:
```
array([[ 395,   81],
       [  46, 1428]], dtype=int64)
```

In [147]:
```
acc=accuracy_score(y_test,y_pred)
acc*100
```

Out[147]: 93.48717948717949

ACCURACY RESULT

LOGISTIC REGRESSION-98.15%

DECISON TREE-99.56%

RANDOM FOREST-99.59%

K Nearest Neighbors-93.49%