```python
# importing libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

import warnings
warnings.filterwarnings('ignore')

# importing dataset
df = pd.read_csv('winequality-red.csv')

df
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
1               7.8             0.880         0.00             2.6
0.098
2               7.8             0.760         0.04             2.3
0.092
3              11.2             0.280         0.56             1.9
0.075
4               7.4             0.700         0.00             1.9
0.076
...             ...               ...          ...             ...
...
1594            6.2             0.600         0.08             2.0
0.090
1595            5.9             0.550         0.10             2.2
0.062
1596            6.3             0.510         0.13             2.3
0.076
1597            5.9             0.645         0.12             2.0
0.075
1598            6.0             0.310         0.47             3.6
0.067

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
```

```
...                    ...                    ...      ...   ...
...
1594                 32.0                  44.0  0.99490  3.45
0.58
1595                 39.0                  51.0  0.99512  3.52
0.76
1596                 29.0                  40.0  0.99574  3.42
0.75
1597                 32.0                  44.0  0.99547  3.57
0.71
1598                 18.0                  42.0  0.99549  3.39
0.66

      alcohol  quality
0         9.4        5
1         9.8        5
2         9.8        5
3         9.8        6
4         9.4        5
...       ...      ...
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6

[1599 rows x 12 columns]
```

```
# checking whether the data is balanced or not
df.quality.value_counts()
```

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

```
df.shape
```

```
(1599, 12)
```

```
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
```

```
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
```

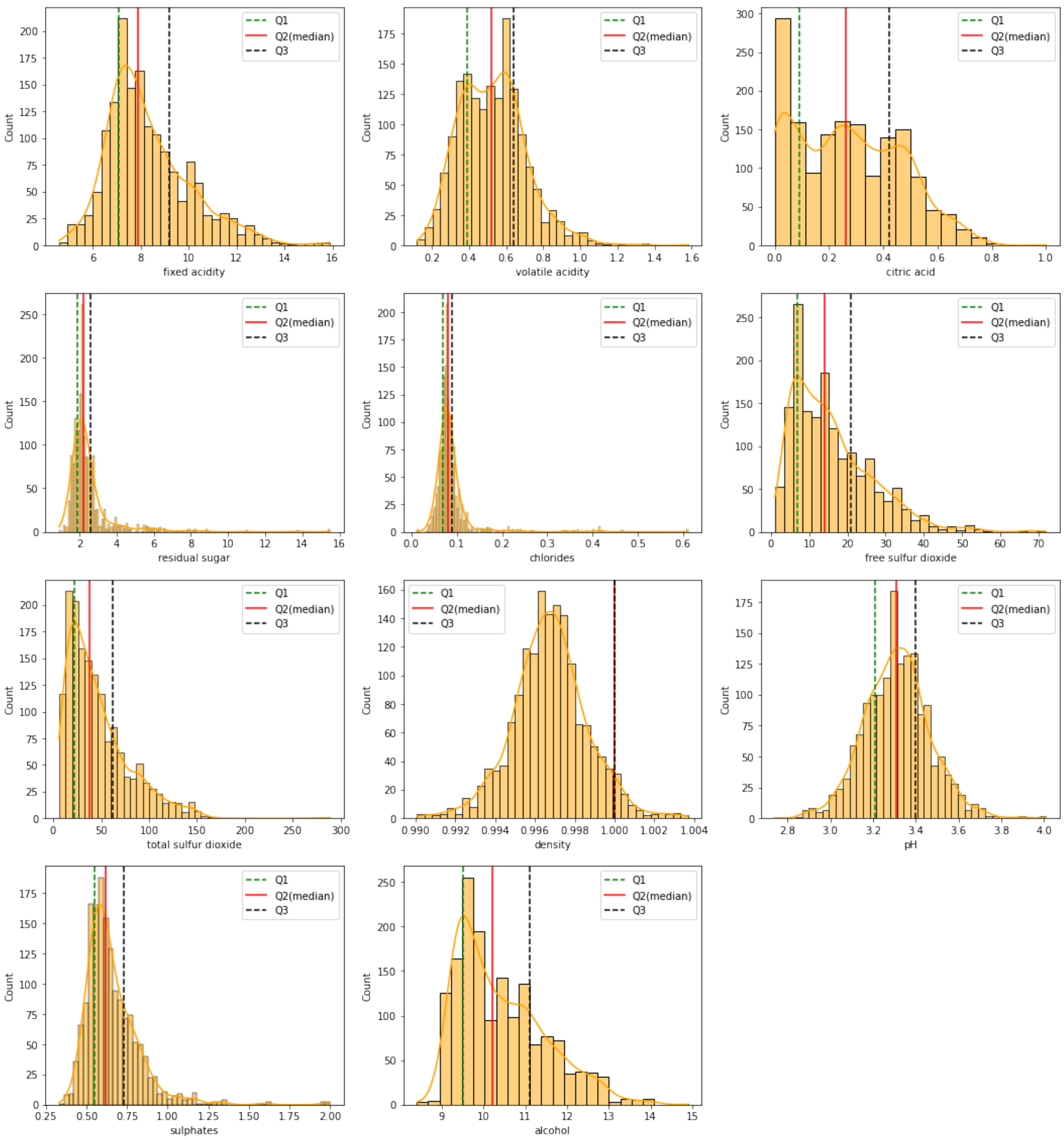
```
25%       3.210000     0.550000     9.500000     5.000000
50%       3.310000     0.620000    10.200000     6.000000
75%       3.400000     0.730000    11.100000     6.000000
max       4.010000     2.000000    14.900000     8.000000
```

```
# checking if there are any null values present in the data
df.isnull().sum()
```

```
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
```

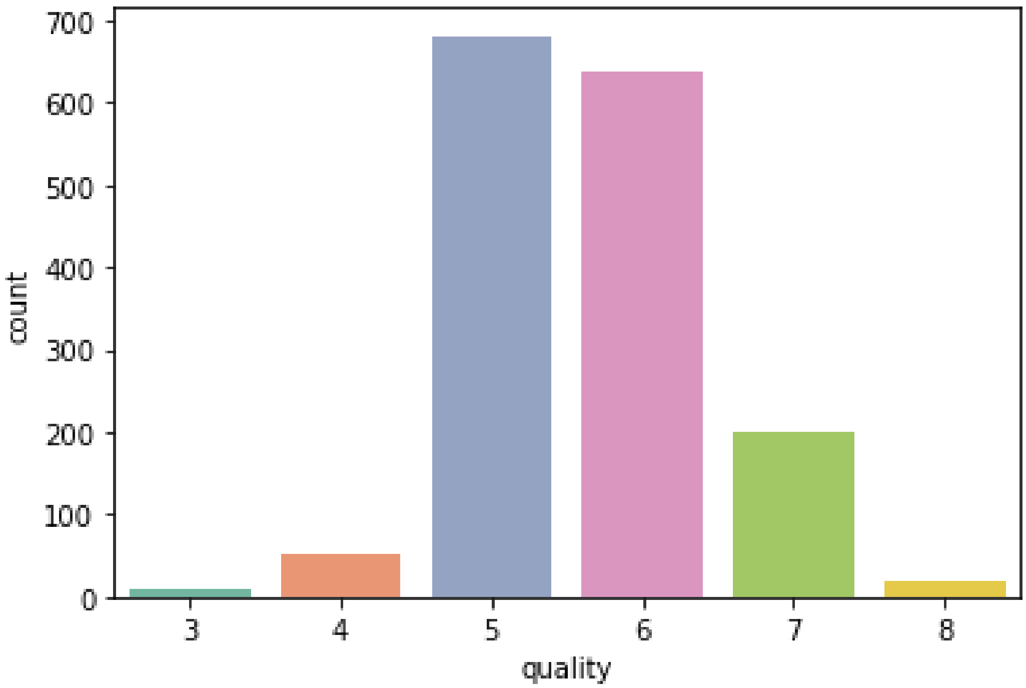
Exploratory Data Analysis

```
# let's check distribution of the data
features_ = df.columns[:-1]

def get_percentile(feature,q_range):
    dist = df[feature].describe()[str(q_range) + '%']
    return round(dist,2)
def render_counterplot():
    fig = plt.figure(figsize=(18,20))
    for column, feature in enumerate(features_):
        fig.add_subplot(4,3, column +1)
        q1 = get_percentile(feature,25)
        q2 = get_percentile(feature,50)
        q3 = get_percentile(feature,75)
        sns.histplot(data = df, x = feature, kde=True, color='orange')
        plt.axvline(q1, linestyle ='--', color='green',label='Q1')
        plt.axvline(q2,color='red',label='Q2(median)')
        plt.axvline(q3,linestyle='--',color='black',label='Q3')
        plt.legend()
render_counterplot()
```

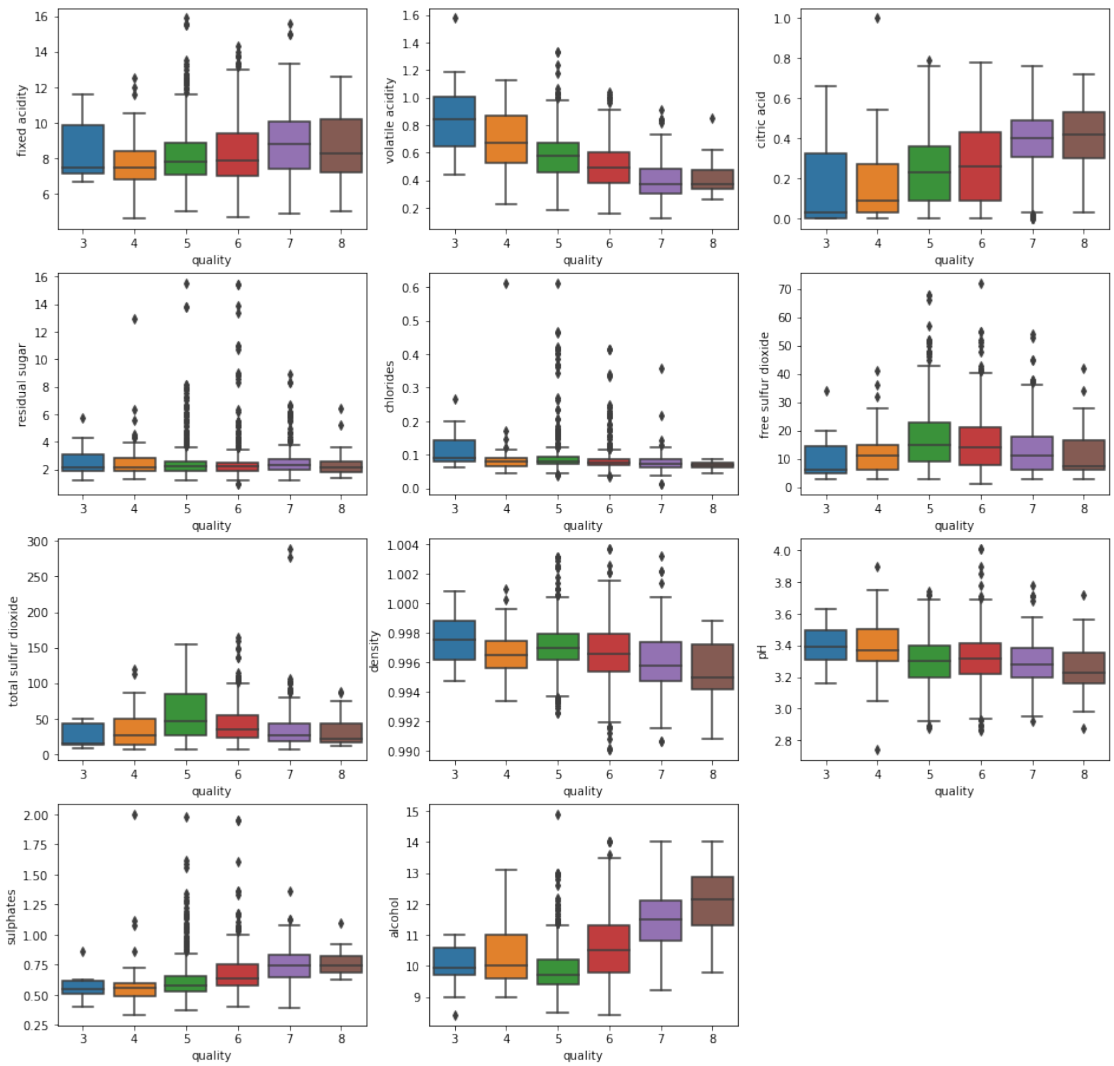



```python
# We can clearly look at countplot and can tell that data is imbalanced
sns.countplot(x='quality',data=df,palette='Set2')
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25a18f37910>
```




```
# Outlier check using a box plot
features_ = df.columns[:-1]
fig = plt.figure(figsize=(16,20))
for column, feature in enumerate(features_):
    fig.add_subplot(5,3, column+1)
    sns.boxplot(data=df, x='quality',y =feature)
```




```
# Split features and target
x = df.drop('quality',axis= 1)
y = df['quality']

x

      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0               7.4             0.700         0.00             1.9
0.076
1               7.8             0.880         0.00             2.6
0.098
2               7.8             0.760         0.04             2.3
0.092
3              11.2             0.280         0.56             1.9
0.075
4               7.4             0.700         0.00             1.9
0.076
...             ...               ...          ...             ...
```

```
...
1594            6.2              0.600         0.08              2.0
0.090
1595            5.9              0.550         0.10              2.2
0.062
1596            6.3              0.510         0.13              2.3
0.076
1597            5.9              0.645         0.12              2.0
0.075
1598            6.0              0.310         0.47              3.6
0.067

      free sulfur dioxide  total sulfur dioxide  density    pH
sulphates  \
0                    11.0                  34.0  0.99780  3.51
0.56
1                    25.0                  67.0  0.99680  3.20
0.68
2                    15.0                  54.0  0.99700  3.26
0.65
3                    17.0                  60.0  0.99800  3.16
0.58
4                    11.0                  34.0  0.99780  3.51
0.56
...                   ...                   ...      ...   ...
...
1594                 32.0                  44.0  0.99490  3.45
0.58
1595                 39.0                  51.0  0.99512  3.52
0.76
1596                 29.0                  40.0  0.99574  3.42
0.75
1597                 32.0                  44.0  0.99547  3.57
0.71
1598                 18.0                  42.0  0.99549  3.39
0.66

      alcohol
0         9.4
1         9.8
2         9.8
3         9.8
4         9.4
...       ...
1594     10.5
1595     11.2
1596     11.0
1597     10.2
1598     11.0
```

```
[1599 rows x 11 columns]
```

```
x.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 11 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
dtypes: float64(11)
memory usage: 137.5 KB
```

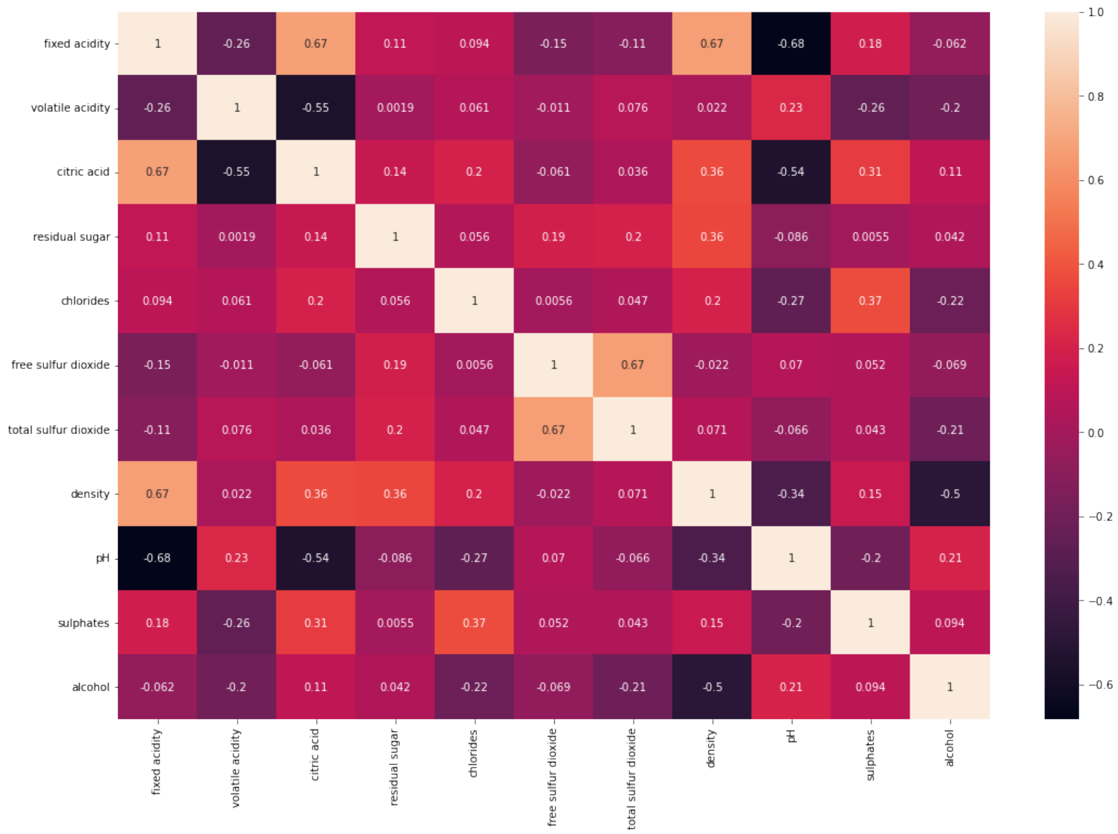
```
# Importing StandardScaler to perform scaling
from sklearn.preprocessing import StandardScaler

# Scaling down the data
scaler = StandardScaler()
x[x.columns] = scaler.fit_transform(x[x.columns])

x
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar
chlorides  \
0         -0.528360          0.961877    -1.391472       -0.453218  -
0.243707
1         -0.298547          1.967442    -1.391472        0.043416   
0.223875
2         -0.298547          1.297065    -1.186070       -0.169427   
0.096353
3          1.654856         -1.384443     1.484154       -0.453218  -
0.264960
4         -0.528360          0.961877    -1.391472       -0.453218  -
0.243707
...             ...               ...          ...             ...
...
1594      -1.217796          0.403229    -0.980669       -0.382271   
0.053845
1595      -1.390155          0.123905    -0.877968       -0.240375  -
0.541259
```

```
1596      -1.160343         -0.099554    -0.723916       -0.169427  -
0.243707
1597      -1.390155          0.654620    -0.775267       -0.382271  -
0.264960
1598      -1.332702         -1.216849     1.021999        0.752894  -
0.434990

      free sulfur dioxide  total sulfur dioxide   density        pH  \
0               -0.466193             -0.379133  0.558274  1.288643
1                0.872638              0.624363  0.028261 -0.719933
2               -0.083669              0.229047  0.134264 -0.331177
3                0.107592              0.411500  0.664277 -0.979104
4               -0.466193             -0.379133  0.558274  1.288643
...                   ...                   ...       ...       ...
1594             1.542054             -0.075043 -0.978765  0.899886
1595             2.211469              0.137820 -0.862162  1.353436
1596             1.255161             -0.196679 -0.533554  0.705508
1597             1.542054             -0.075043 -0.676657  1.677400
1598             0.203223             -0.135861 -0.666057  0.511130

      sulphates   alcohol
0     -0.579207 -0.960246
1      0.128950 -0.584777
2     -0.048089 -0.584777
3     -0.461180 -0.584777
4     -0.579207 -0.960246
...         ...       ...
1594  -0.461180  0.072294
1595   0.601055  0.729364
1596   0.542042  0.541630
1597   0.305990 -0.209308
1598   0.010924  0.541630

[1599 rows x 11 columns]
```

```
# Plotting heatmap to understand the correlation among the data
plt.figure(figsize=(18,12))
sns.heatmap(x.corr(), annot=True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x25a18e88e50>
```



Handling Imbalanced Dataset

```
# importing SMOTE from imblearn.over_sampling
from imblearn.over_sampling import SMOTE

# Balancing the data using SMOTE through oversampling
smote = SMOTE()
x_sm, y_sm = smote.fit_resample(x,y)

# We can see now the data is balanced
y_sm.value_counts()

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

# splitting the data into training and testing data
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test =
train_test_split(x_sm,y_sm,test_size=0.2, random_state=1)

# importing required libraries for model building
import tensorflow
```

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.layers import Dropout
```

```python
x_train
```

```
      fixed acidity  volatile acidity  citric acid  residual sugar  
chlorides  \
2816      -0.493870         -0.750377     0.128517        0.731666  -
0.077948
2229      -0.516772          3.868725    -1.237565       -0.311323   
0.084507
3808       1.076554         -1.357236     1.367263       -0.315980  -
0.171282
546       -0.470907          0.123905    -0.159061       -0.382271  -
0.201199
1406      -0.068735         -1.607903     0.354443        1.817111  -
0.541259
...             ...               ...          ...             ...   
...
3839       1.950527         -1.198530     1.977409       -0.298537  -
0.367053
1096      -0.987984          1.101539    -0.929318        2.100902   
0.627696
3980       0.740875         -1.227771     1.459367        0.162481  -
0.192889
235       -0.643266          0.570823    -1.391472       -0.453218   
0.202621
1061       0.448342         -0.714066     1.176051       -0.524166  -
0.349975

      free sulfur dioxide  total sulfur dioxide   density        pH  \
2816            -1.169080             -1.168244 -0.934234  0.472234   
2229            -0.180105             -0.410224 -0.325409  0.485903   
3808             1.520592              0.349847 -0.791488 -1.840997   
546             -0.561823             -0.561586  0.823281  0.899886   
1406            -0.753085             -0.744040  0.346269 -0.590348   
...                   ...                   ...       ...       ...   
3839            -0.991384             -0.610950  0.800658 -2.421489   
1096            -0.657454             -0.896085 -0.104243  0.251958   
3980            -0.959733             -0.900978 -0.115522 -1.043897   
235             -0.179300             -0.257497  0.001760  0.381544   
1061            -0.848716             -0.926494 -1.127169 -0.655141   

      sulphates   alcohol  
2816  -0.502492  1.114231  
2229  -0.992795 -0.227450  
3808  -0.133500  1.085057  
546    0.719081 -0.866379  
1406   1.663290  0.447763  
```

```
...          ...       ...
3839   1.226051 -0.553997
1096  -0.992298  0.353895
3980   1.288895  1.092978
235   -0.461180 -1.335715
1061   0.187963  1.949639

[3268 rows x 11 columns]
```

```
# creating a sequence for an ANN model
model = Sequential()
model.add(tensorflow.keras.layers.Input(shape=11,))
model.add(tensorflow.keras.layers.Dense(32,activation='relu'))
model.add(tensorflow.keras.layers.Dense(64,activation='relu'))
# model.add(tensorflow.keras.layers.Dropout(0.3))
model.add(tensorflow.keras.layers.Dense(128,activation='relu'))
model.add(tensorflow.keras.layers.Dense(6,activation='softmax'))
```

```
# LabelEncoding because categorical_crossentropy take data in one-hot
encoded format
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
y_train = le.fit_transform(y_train)
y_test = le.fit_transform(y_test)
```

```
# Converted into 0-5 but still not one-hot encoded
y_test
```

```
array([0, 4, 0, 1, 4, 4, 0, 2, 1, 4, 3, 2, 4, 2, 4, 1, 0, 3, 2, 4, 2,
5,
       5, 5, 5, 0, 3, 3, 0, 5, 5, 0, 3, 0, 4, 1, 2, 5, 4, 3, 4, 0, 5,
3,
       1, 2, 4, 4, 4, 4, 2, 4, 1, 4, 1, 0, 5, 0, 1, 5, 0, 2, 0, 5, 0,
2,
       3, 3, 1, 3, 1, 1, 2, 0, 0, 0, 1, 3, 2, 2, 0, 2, 5, 0, 2, 2, 3,
5,
       2, 5, 3, 1, 4, 5, 1, 3, 2, 3, 3, 1, 2, 4, 0, 5, 4, 0, 1, 0, 0,
4,
       3, 0, 1, 3, 1, 0, 1, 0, 4, 2, 2, 4, 1, 0, 4, 3, 3, 4, 5, 4, 4,
3,
       0, 0, 5, 0, 3, 2, 1, 5, 2, 0, 3, 0, 3, 4, 3, 3, 0, 1, 1, 2, 2,
1,
       4, 4, 0, 5, 1, 0, 3, 0, 3, 5, 1, 5, 5, 3, 2, 0, 0, 5, 3, 4, 4,
5,
       4, 4, 1, 3, 1, 0, 5, 2, 3, 2, 4, 4, 2, 4, 0, 2, 0, 1, 2, 1, 1,
3,
       4, 3, 3, 1, 2, 4, 0, 1, 5, 1, 4, 0, 0, 0, 5, 1, 0, 0, 3, 3, 1,
1,
       1, 5, 3, 1, 1, 1, 3, 0, 3, 1, 4, 3, 2, 3, 1, 1, 3, 5, 3, 4, 0,
4,
       5, 5, 2, 4, 2, 0, 5, 5, 5, 5, 1, 5, 4, 1, 2, 1, 4, 0, 1, 5, 2,
```

3,
1, 1, 3, 1, 5, 3, 4, 4, 5, 3, 0, 5, 1, 2, 3, 2, 3, 1, 3, 0, 0,
2,
4, 3, 1, 1, 3, 2, 1, 3, 2, 5, 5, 2, 2, 1, 5, 1, 2, 4, 5, 2, 2,
4,
4, 0, 5, 4, 1, 4, 4, 3, 5, 3, 3, 2, 3, 4, 3, 0, 5, 3, 5, 3, 3,
0,
0, 1, 1, 0, 1, 3, 1, 3, 1, 5, 4, 2, 2, 2, 4, 1, 3, 1, 5, 4, 4,
2,
5, 3, 1, 2, 5, 0, 4, 1, 5, 1, 2, 2, 3, 2, 2, 1, 1, 1, 3, 5, 1,
4,
3, 1, 5, 0, 1, 2, 5, 0, 4, 2, 0, 5, 4, 0, 1, 2, 2, 1, 2, 2, 3,
5,
1, 0, 3, 2, 4, 1, 3, 0, 0, 4, 5, 5, 4, 5, 2, 3, 0, 3, 2, 1, 2,
0,
0, 4, 3, 2, 5, 2, 2, 3, 2, 5, 3, 5, 5, 5, 4, 5, 3, 3, 1, 5, 4,
1,
1, 1, 4, 2, 5, 1, 3, 1, 4, 5, 2, 2, 4, 2, 3, 1, 4, 2, 1, 1, 0,
0,
3, 5, 4, 2, 4, 4, 5, 4, 2, 2, 0, 1, 3, 0, 4, 0, 5, 4, 1, 2, 3,
4,
1, 3, 3, 1, 1, 1, 5, 0, 4, 4, 2, 0, 0, 2, 1, 0, 2, 1, 3, 2, 3,
5,
4, 0, 2, 4, 0, 5, 5, 3, 1, 4, 1, 1, 1, 3, 2, 2, 1, 2, 5, 1, 5,
5,
5, 1, 3, 0, 5, 3, 5, 1, 4, 0, 5, 3, 4, 1, 1, 3, 0, 2, 1, 4, 0,
3,
0, 5, 0, 0, 5, 2, 1, 5, 1, 5, 2, 1, 0, 1, 2, 2, 0, 4, 1, 3, 0,
1,
4, 0, 4, 2, 5, 5, 2, 5, 2, 5, 5, 3, 4, 4, 2, 2, 0, 3, 1, 2, 5,
5,
2, 3, 0, 4, 4, 0, 1, 3, 5, 4, 0, 4, 3, 5, 3, 2, 1, 5, 1, 5, 1,
5,
0, 1, 4, 3, 5, 0, 2, 4, 1, 5, 0, 0, 3, 3, 2, 1, 5, 2, 3, 5, 2,
2,
1, 0, 3, 4, 2, 2, 4, 0, 1, 3, 3, 4, 1, 5, 1, 3, 0, 2, 2, 3, 0,
0,
3, 3, 0, 1, 0, 2, 5, 5, 3, 3, 3, 5, 1, 1, 2, 2, 3, 2, 5, 2, 5,
2,
5, 3, 1, 3, 0, 5, 3, 2, 2, 3, 3, 1, 0, 1, 5, 5, 5, 1, 1, 0, 2,
5,
3, 3, 1, 2, 2, 1, 1, 1, 3, 4, 5, 1, 5, 1, 3, 1, 0, 4, 3, 5, 5,
5,
3, 2, 4, 4, 3, 4, 1, 0, 5, 4, 2, 2, 5, 2, 5, 2, 1, 2, 1, 2, 5,
0,
4, 4, 2, 5, 0, 2, 4, 1, 2, 0, 1, 3, 1, 2, 1, 0, 3, 3, 3, 4, 5,
5,
4, 0, 3, 1, 2, 1, 1, 1, 3, 1, 5, 3, 4, 1, 0, 4, 4, 0, 2, 0, 5,
0,
1, 0, 4, 1, 5, 5, 3, 4, 0, 1, 5, 0, 1, 0, 0, 0, 5, 4, 5, 1, 3,

```
3,
       1, 5, 3, 0], dtype=int64)
```

```
# Flattening the array to feed the data to input layers
y_train = pd.DataFrame(y_train.reshape(len(y_train),1))
y_test = pd.DataFrame(y_test.reshape(len(y_test),1))
```

```
y_train
```

```
      0
0     1
1     0
2     5
3     3
4     3
...  ..
3263  5
3264  3
3265  5
3266  3
3267  5

[3268 rows x 1 columns]
```

```
# Now converting the flattened array into one-hot encoded format
y_train = tensorflow.keras.utils.to_categorical(y_train,6)
y_test = tensorflow.keras.utils.to_categorical(y_test,6)
```

```
# Building the model
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics
=['accuracy'])
```

```
# Training the model
model.fit(x_train,y_train,batch_size=32,epochs=100)
```

```
Epoch 1/100
103/103 [==============================] - 1s 2ms/step - loss: 1.3252
- accuracy: 0.4517
Epoch 2/100
103/103 [==============================] - 0s 2ms/step - loss: 0.9626
- accuracy: 0.6248
Epoch 3/100
103/103 [==============================] - 0s 2ms/step - loss: 0.8438
- accuracy: 0.6827
Epoch 4/100
103/103 [==============================] - 0s 1ms/step - loss: 0.7625
- accuracy: 0.7087
Epoch 5/100
103/103 [==============================] - 0s 1ms/step - loss: 0.7011
- accuracy: 0.7323
Epoch 6/100
103/103 [==============================] - 0s 2ms/step - loss: 0.6508
```

```
- accuracy: 0.7537
Epoch 7/100
103/103 [==============================] - 0s 2ms/step - loss: 0.6105
- accuracy: 0.7641
Epoch 8/100
103/103 [==============================] - 0s 2ms/step - loss: 0.5916
- accuracy: 0.7708
Epoch 9/100
103/103 [==============================] - 0s 2ms/step - loss: 0.5575
- accuracy: 0.7861
Epoch 10/100
103/103 [==============================] - 0s 2ms/step - loss: 0.5298
- accuracy: 0.7953
Epoch 11/100
103/103 [==============================] - 0s 2ms/step - loss: 0.5055
- accuracy: 0.8011
Epoch 12/100
103/103 [==============================] - 0s 1ms/step - loss: 0.4918
- accuracy: 0.8103
Epoch 13/100
103/103 [==============================] - 0s 2ms/step - loss: 0.4704
- accuracy: 0.8192
Epoch 14/100
103/103 [==============================] - 0s 1ms/step - loss: 0.4594
- accuracy: 0.8219
Epoch 15/100
103/103 [==============================] - 0s 1ms/step - loss: 0.4371
- accuracy: 0.8384
Epoch 16/100
103/103 [==============================] - 0s 2ms/step - loss: 0.4199
- accuracy: 0.8348
Epoch 17/100
103/103 [==============================] - 0s 2ms/step - loss: 0.4090
- accuracy: 0.8421
Epoch 18/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3942
- accuracy: 0.8513
Epoch 19/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3878
- accuracy: 0.8482
Epoch 20/100
103/103 [==============================] - 0s 1ms/step - loss: 0.3799
- accuracy: 0.8565
Epoch 21/100
103/103 [==============================] - 0s 1ms/step - loss: 0.3647
- accuracy: 0.8565
Epoch 22/100
103/103 [==============================] - 0s 1ms/step - loss: 0.3529
- accuracy: 0.8647
Epoch 23/100
```

```
103/103 [==============================] - 0s 1ms/step - loss: 0.3417
- accuracy: 0.8635
Epoch 24/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3394
- accuracy: 0.8715
Epoch 25/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3374
- accuracy: 0.8632
Epoch 26/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3169
- accuracy: 0.8748
Epoch 27/100
103/103 [==============================] - 0s 2ms/step - loss: 0.3171
- accuracy: 0.8782
Epoch 28/100
103/103 [==============================] - 0s 1ms/step - loss: 0.3034
- accuracy: 0.8779
Epoch 29/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2968
- accuracy: 0.8807
Epoch 30/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2850
- accuracy: 0.8932
Epoch 31/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2813
- accuracy: 0.8892
Epoch 32/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2810
- accuracy: 0.8905
Epoch 33/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2741
- accuracy: 0.8966
Epoch 34/100
103/103 [==============================] - 0s 2ms/step - loss: 0.2652
- accuracy: 0.8957
Epoch 35/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2638
- accuracy: 0.8975
Epoch 36/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2571
- accuracy: 0.9018
Epoch 37/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2458
- accuracy: 0.9054
Epoch 38/100
103/103 [==============================] - 0s 2ms/step - loss: 0.2335
- accuracy: 0.9155
Epoch 39/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2348
- accuracy: 0.9091
```

```
Epoch 40/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2311
- accuracy: 0.9146
Epoch 41/100
103/103 [==============================] - 0s 2ms/step - loss: 0.2249
- accuracy: 0.9159
Epoch 42/100
103/103 [==============================] - 0s 2ms/step - loss: 0.2278
- accuracy: 0.9152
Epoch 43/100
103/103 [==============================] - 0s 2ms/step - loss: 0.2124
- accuracy: 0.9204
Epoch 44/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2122
- accuracy: 0.9238
Epoch 45/100
103/103 [==============================] - 0s 1ms/step - loss: 0.2042
- accuracy: 0.9241
Epoch 46/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1984
- accuracy: 0.9250
Epoch 47/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1982
- accuracy: 0.9250
Epoch 48/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1855
- accuracy: 0.9339
Epoch 49/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1899
- accuracy: 0.9296
Epoch 50/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1841
- accuracy: 0.9339
Epoch 51/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1794
- accuracy: 0.9342
Epoch 52/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1701
- accuracy: 0.9406
Epoch 53/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1701
- accuracy: 0.9397
Epoch 54/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1743
- accuracy: 0.9382
Epoch 55/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1612
- accuracy: 0.9406
Epoch 56/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1612
```

```
- accuracy: 0.9458
Epoch 57/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1606
- accuracy: 0.9437
Epoch 58/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1685
- accuracy: 0.9428
Epoch 59/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1511
- accuracy: 0.9486
Epoch 60/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1459
- accuracy: 0.9535
Epoch 61/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1454
- accuracy: 0.9498
Epoch 62/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1438
- accuracy: 0.9532
Epoch 63/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1376
- accuracy: 0.9498
Epoch 64/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1356
- accuracy: 0.9504
Epoch 65/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1246
- accuracy: 0.9587
Epoch 66/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1298
- accuracy: 0.9559
Epoch 67/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1337
- accuracy: 0.9550
Epoch 68/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1267
- accuracy: 0.9587
Epoch 69/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1362
- accuracy: 0.9538
Epoch 70/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1188
- accuracy: 0.9621
Epoch 71/100
103/103 [==============================] - 0s 1ms/step - loss: 0.1113
- accuracy: 0.9636
Epoch 72/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1105
- accuracy: 0.9642
Epoch 73/100
```

```
103/103 [==============================] - 0s 1ms/step - loss: 0.1219
- accuracy: 0.9596
Epoch 74/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1012
- accuracy: 0.9666
Epoch 75/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0999
- accuracy: 0.9697
Epoch 76/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1060
- accuracy: 0.9642
Epoch 77/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0958
- accuracy: 0.9712
Epoch 78/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0966
- accuracy: 0.9673
Epoch 79/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0966
- accuracy: 0.9706
Epoch 80/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1025
- accuracy: 0.9709
Epoch 81/100

103/103 [==============================] - 0s 1ms/step - loss: 0.1107
- accuracy: 0.9624
Epoch 82/100
103/103 [==============================] - 0s 2ms/step - loss: 0.1219
- accuracy: 0.9584
Epoch 83/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0859
- accuracy: 0.9746
Epoch 84/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0803
- accuracy: 0.9749
Epoch 85/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0958
- accuracy: 0.9673
Epoch 86/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0969
- accuracy: 0.9673
Epoch 87/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0755
- accuracy: 0.9764
Epoch 88/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0693
- accuracy: 0.9807
Epoch 89/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0975
```

```
- accuracy: 0.9691
Epoch 90/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0766
- accuracy: 0.9755
Epoch 91/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0798
- accuracy: 0.9758
Epoch 92/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0720
- accuracy: 0.9780
Epoch 93/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0679
- accuracy: 0.9807
Epoch 94/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0711
- accuracy: 0.9780
Epoch 95/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0656
- accuracy: 0.9807
Epoch 96/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0775
- accuracy: 0.9734
Epoch 97/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0616
- accuracy: 0.9847
Epoch 98/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0590
- accuracy: 0.9835
Epoch 99/100
103/103 [==============================] - 0s 1ms/step - loss: 0.0722
- accuracy: 0.9783
Epoch 100/100
103/103 [==============================] - 0s 2ms/step - loss: 0.0577
- accuracy: 0.9838
```

```
<keras.callbacks.History at 0x25a25a67220>
```

```
y_train
```

```
array([[0., 1., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 0., 1.],
       ...,
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1.]], dtype=float32)
```

```
# Predicting by rounding the value returned by softmax because metrics
accepts values less than 1
y_pred = model.predict(x_test).round()
```

```
26/26 [==============================] - 0s 1ms/step

y_pred

array([[1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0.]], dtype=float32)

# Finding out maximum value to get the original data
y_pred = np.argmax(y_pred,axis=1)
y_pred

array([0, 4, 0, 1, 2, 4, 0, 2, 1, 0, 3, 2, 4, 2, 4, 1, 0, 3, 3, 4, 2,
5,
       5, 5, 5, 0, 3, 3, 0, 5, 5, 0, 3, 0, 4, 1, 2, 5, 4, 3, 3, 0, 5,
4,
       1, 3, 4, 4, 4, 4, 2, 4, 1, 4, 1, 0, 5, 0, 2, 5, 0, 2, 0, 5, 0,
3,
       3, 3, 1, 3, 1, 1, 2, 0, 0, 0, 1, 2, 2, 2, 0, 2, 5, 0, 2, 2, 3,
5,
       2, 5, 2, 1, 4, 5, 1, 3, 2, 2, 3, 4, 2, 4, 0, 5, 4, 0, 1, 0, 0,
4,
       0, 0, 1, 2, 2, 0, 1, 0, 4, 2, 2, 4, 1, 0, 4, 2, 3, 4, 5, 4, 3,
2,
       0, 0, 5, 0, 4, 3, 1, 5, 2, 0, 3, 0, 3, 4, 2, 2, 0, 1, 1, 3, 2,
1,
       4, 4, 0, 5, 1, 0, 2, 0, 2, 5, 1, 5, 5, 4, 2, 0, 0, 5, 3, 4, 4,
5,
       4, 4, 1, 3, 1, 0, 5, 2, 4, 2, 4, 4, 2, 4, 0, 2, 0, 1, 2, 2, 1,
3,
       4, 2, 0, 1, 2, 4, 0, 1, 5, 1, 4, 0, 0, 0, 5, 1, 0, 0, 2, 3, 1,
1,
       1, 5, 3, 1, 1, 1, 3, 0, 3, 1, 4, 3, 2, 2, 1, 1, 3, 5, 2, 2, 0,
3,
       5, 5, 2, 4, 2, 0, 5, 5, 5, 5, 1, 5, 4, 1, 4, 1, 4, 0, 1, 5, 2,
3,
       1, 2, 3, 1, 5, 3, 4, 4, 5, 3, 0, 5, 1, 1, 3, 2, 3, 1, 2, 0, 0,
3,
       4, 3, 1, 1, 3, 3, 1, 3, 2, 5, 5, 2, 2, 1, 5, 1, 2, 4, 5, 3, 3,
3,
       4, 0, 5, 4, 1, 4, 4, 3, 5, 3, 1, 3, 3, 4, 2, 0, 5, 3, 5, 3, 3,
0,
       0, 1, 1, 0, 1, 3, 1, 3, 1, 5, 4, 2, 3, 2, 4, 1, 2, 1, 5, 2, 4,
2,
       5, 3, 1, 2, 5, 0, 4, 1, 5, 1, 0, 2, 1, 2, 2, 1, 1, 1, 3, 5, 1,
4,
       3, 1, 5, 0, 0, 2, 5, 0, 4, 2, 0, 5, 4, 0, 1, 1, 2, 1, 2, 1, 5,
```

```
5,
       1, 0, 3, 3, 4, 1, 4, 0, 0, 4, 5, 5, 4, 5, 2, 2, 0, 2, 2, 1, 2,
0,
       0, 4, 5, 2, 5, 2, 2, 3, 3, 5, 0, 5, 5, 5, 4, 5, 3, 3, 1, 5, 4,
1,
       1, 1, 4, 2, 5, 1, 3, 3, 4, 5, 2, 2, 0, 2, 3, 1, 4, 2, 1, 1, 0,
0,
       3, 5, 2, 2, 4, 4, 5, 4, 2, 2, 0, 1, 0, 0, 4, 0, 5, 4, 3, 2, 2,
4,
       1, 3, 0, 3, 1, 1, 5, 0, 4, 4, 2, 0, 0, 1, 1, 0, 2, 1, 3, 2, 3,
5,
       4, 0, 2, 3, 0, 5, 5, 2, 1, 4, 1, 1, 1, 2, 2, 2, 1, 2, 5, 1, 5,
5,
       5, 1, 3, 0, 5, 3, 5, 1, 4, 0, 5, 2, 4, 1, 1, 3, 0, 2, 1, 4, 0,
3,
       0, 5, 0, 0, 5, 3, 1, 5, 1, 5, 2, 1, 0, 1, 2, 3, 0, 4, 1, 3, 0,
1,
       4, 0, 4, 1, 5, 5, 3, 5, 2, 5, 5, 3, 4, 4, 4, 2, 0, 3, 1, 2, 5,
5,
       2, 3, 0, 2, 4, 0, 1, 3, 5, 0, 0, 4, 3, 5, 4, 2, 1, 5, 1, 5, 3,
5,
       0, 1, 4, 3, 5, 0, 0, 4, 1, 5, 0, 0, 3, 1, 0, 1, 5, 3, 2, 5, 3,
3,
       1, 0, 3, 4, 2, 2, 4, 0, 1, 2, 2, 4, 1, 5, 1, 4, 0, 3, 2, 2, 0,
0,
       3, 3, 0, 1, 0, 2, 5, 5, 3, 3, 3, 5, 1, 1, 3, 2, 3, 2, 5, 2, 5,
3,
       5, 3, 1, 3, 0, 5, 2, 2, 2, 3, 3, 1, 0, 1, 5, 5, 5, 1, 1, 0, 2,
5,
       2, 3, 1, 1, 2, 1, 1, 1, 2, 4, 5, 1, 5, 1, 3, 1, 0, 4, 3, 5, 5,
5,
       3, 2, 4, 4, 3, 4, 1, 0, 5, 4, 2, 2, 5, 3, 5, 2, 1, 3, 1, 2, 5,
0,
       3, 2, 2, 5, 0, 2, 3, 1, 1, 0, 1, 1, 1, 2, 1, 0, 4, 2, 3, 4, 5,
5,
       4, 0, 0, 1, 3, 1, 1, 1, 3, 1, 5, 3, 2, 1, 0, 4, 3, 0, 1, 0, 5,
0,
       1, 0, 4, 1, 5, 5, 3, 4, 0, 1, 5, 0, 1, 0, 0, 0, 5, 4, 5, 1, 3,
3,
       1, 5, 5, 0], dtype=int64)
```

```
y_test
```

```
array([[1., 0., 0., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0.],
       [1., 0., 0., 0., 0., 0.],
       ...,
       [0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 1., 0., 0.],
       [1., 0., 0., 0., 0., 0.]], dtype=float32)
```

```python
# Finding out maximum value to get the original data
y_test = np.argmax(y_test,axis=1)
y_test
```

```
array([0, 4, 0, 1, 4, 4, 0, 2, 1, 4, 3, 2, 4, 2, 4, 1, 0, 3, 2, 4, 2,
5,
       5, 5, 5, 0, 3, 3, 0, 5, 5, 0, 3, 0, 4, 1, 2, 5, 4, 3, 4, 0, 5,
3,
       1, 2, 4, 4, 4, 4, 2, 4, 1, 4, 1, 0, 5, 0, 1, 5, 0, 2, 0, 5, 0,
2,
       3, 3, 1, 3, 1, 1, 2, 0, 0, 0, 1, 3, 2, 2, 0, 2, 5, 0, 2, 2, 3,
5,
       2, 5, 3, 1, 4, 5, 1, 3, 2, 3, 3, 1, 2, 4, 0, 5, 4, 0, 1, 0, 0,
4,
       3, 0, 1, 3, 1, 0, 1, 0, 4, 2, 2, 4, 1, 0, 4, 3, 3, 4, 5, 4, 4,
3,
       0, 0, 5, 0, 3, 2, 1, 5, 2, 0, 3, 0, 3, 4, 3, 3, 0, 1, 1, 2, 2,
1,
       4, 4, 0, 5, 1, 0, 3, 0, 3, 5, 1, 5, 5, 3, 2, 0, 0, 5, 3, 4, 4,
5,
       4, 4, 1, 3, 1, 0, 5, 2, 3, 2, 4, 4, 2, 4, 0, 2, 0, 1, 2, 1, 1,
3,
       4, 3, 3, 1, 2, 4, 0, 1, 5, 1, 4, 0, 0, 0, 5, 1, 0, 0, 3, 3, 1,
1,
       1, 5, 3, 1, 1, 1, 3, 0, 3, 1, 4, 3, 2, 3, 1, 1, 3, 5, 3, 4, 0,
4,
       5, 5, 2, 4, 2, 0, 5, 5, 5, 5, 1, 5, 4, 1, 2, 1, 4, 0, 1, 5, 2,
3,
       1, 1, 3, 1, 5, 3, 4, 4, 5, 3, 0, 5, 1, 2, 3, 2, 3, 1, 3, 0, 0,
2,
       4, 3, 1, 1, 3, 2, 1, 3, 2, 5, 5, 2, 2, 1, 5, 1, 2, 4, 5, 2, 2,
4,
       4, 0, 5, 4, 1, 4, 4, 3, 5, 3, 3, 2, 3, 4, 3, 0, 5, 3, 5, 3, 3,
0,
       0, 1, 1, 0, 1, 3, 1, 3, 1, 5, 4, 2, 2, 2, 4, 1, 3, 1, 5, 4, 4,
2,
       5, 3, 1, 2, 5, 0, 4, 1, 5, 1, 2, 2, 3, 2, 2, 1, 1, 1, 3, 5, 1,
4,
       3, 1, 5, 0, 1, 2, 5, 0, 4, 2, 0, 5, 4, 0, 1, 2, 2, 1, 2, 2, 3,
5,
       1, 0, 3, 2, 4, 1, 3, 0, 0, 4, 5, 5, 4, 5, 2, 3, 0, 3, 2, 1, 2,
0,
       0, 4, 3, 2, 5, 2, 2, 3, 2, 5, 3, 5, 5, 5, 4, 5, 3, 3, 1, 5, 4,
1,
       1, 1, 4, 2, 5, 1, 3, 1, 4, 5, 2, 2, 4, 2, 3, 1, 4, 2, 1, 1, 0,
0,
       3, 5, 4, 2, 4, 4, 5, 4, 2, 2, 0, 1, 3, 0, 4, 0, 5, 4, 1, 2, 3,
4,
       1, 3, 3, 1, 1, 1, 5, 0, 4, 4, 2, 0, 0, 2, 1, 0, 2, 1, 3, 2, 3,
5,
```

```
       4, 0, 2, 4, 0, 5, 5, 3, 1, 4, 1, 1, 1, 3, 2, 2, 1, 2, 5, 1, 5,
5,
       5, 1, 3, 0, 5, 3, 5, 1, 4, 0, 5, 3, 4, 1, 1, 3, 0, 2, 1, 4, 0,
3,
       0, 5, 0, 0, 5, 2, 1, 5, 1, 5, 2, 1, 0, 1, 2, 2, 0, 4, 1, 3, 0,
1,
       4, 0, 4, 2, 5, 5, 2, 5, 2, 5, 5, 3, 4, 4, 2, 2, 0, 3, 1, 2, 5,
5,
       2, 3, 0, 4, 4, 0, 1, 3, 5, 4, 0, 4, 3, 5, 3, 2, 1, 5, 1, 5, 1,
5,
       0, 1, 4, 3, 5, 0, 2, 4, 1, 5, 0, 0, 3, 3, 2, 1, 5, 2, 3, 5, 2,
2,
       1, 0, 3, 4, 2, 2, 4, 0, 1, 3, 3, 4, 1, 5, 1, 3, 0, 2, 2, 3, 0,
0,
       3, 3, 0, 1, 0, 2, 5, 5, 3, 3, 3, 5, 1, 1, 2, 2, 3, 2, 5, 2, 5,
2,
       5, 3, 1, 3, 0, 5, 3, 2, 2, 3, 3, 1, 0, 1, 5, 5, 5, 1, 1, 0, 2,
5,
       3, 3, 1, 2, 2, 1, 1, 1, 3, 4, 5, 1, 5, 1, 3, 1, 0, 4, 3, 5, 5,
5,
       3, 2, 4, 4, 3, 4, 1, 0, 5, 4, 2, 2, 5, 2, 5, 2, 1, 2, 1, 2, 5,
0,
       4, 4, 2, 5, 0, 2, 4, 1, 2, 0, 1, 3, 1, 2, 1, 0, 3, 3, 3, 4, 5,
5,
       4, 0, 3, 1, 2, 1, 1, 1, 3, 1, 5, 3, 4, 1, 0, 4, 4, 0, 2, 0, 5,
0,
       1, 0, 4, 1, 5, 5, 3, 4, 0, 1, 5, 0, 1, 0, 0, 0, 5, 4, 5, 1, 3,
3,
       1, 5, 3, 0], dtype=int64)
```

```python
# Accuracy Score
from sklearn.metrics import accuracy_score
accuracy_score(y_test,y_pred)*100
```

```
85.57457212713936
```

```python
# Classification Report
from sklearn.metrics import classification_report
print(classification_report(y_pred,y_test))
```

```
              precision    recall  f1-score   support

           0       1.00      0.91      0.95       141
           1       0.94      0.93      0.93       162
           2       0.72      0.70      0.71       138
           3       0.63      0.70      0.66       125
           4       0.85      0.90      0.87       112
           5       1.00      0.98      0.99       140

    accuracy                           0.86       818
   macro avg       0.86      0.85      0.85       818
```

```
weighted avg       0.86      0.86      0.86       818
```