Assignment-4

Name: K Naga Sai Krishna

Project Title:

Grapes to Greatness: Machine Learning in Wine Quality Prediction

Description:

Predicting wine quality using machine learning is a common and valuable application in the field of data science and analytics. Wine quality prediction involves building a model that can assess and predict the quality of a wine based on various input features, such as chemical composition, sensory characteristics, and environmental factors.

The two datasets are related to red and white variants of the Portuguese "Vinho Verde" wine. For more details, consult the reference [Cortez et al., 2009]. Due to privacy and logistic issues, only physicochemical (inputs) and sensory (the output) variables are available (e.g. there is no data about grape types, wine brand, wine selling price, etc.).

These datasets can be viewed as classification or regression tasks. The classes are ordered and not balanced (e.g. there are much more normal wines than excellent or poor ones).

Dataset: link **Task**:

- Load the Dataset
- Data preprocessing including visualization
- Machine Learning Model building
- Evaluate the model
- Test with random observation

Importing libraries and dataset

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
[ ] from google.colab import files
    uploaded = files.upload()
```

Browse... No files selected. Upload widget is only available when the cel

```
Saving winequality-red.csv to winequality-red.csv
```

```
[ ] import io
    df = pd.read_csv(io.BytesIO(uploaded['winequality-red.csv']))
```

```
df.head()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

```
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.shape
```

```
(1599, 12)
```

Data Preprocessing and visualization

```
[ ] df.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.636023
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.807569
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

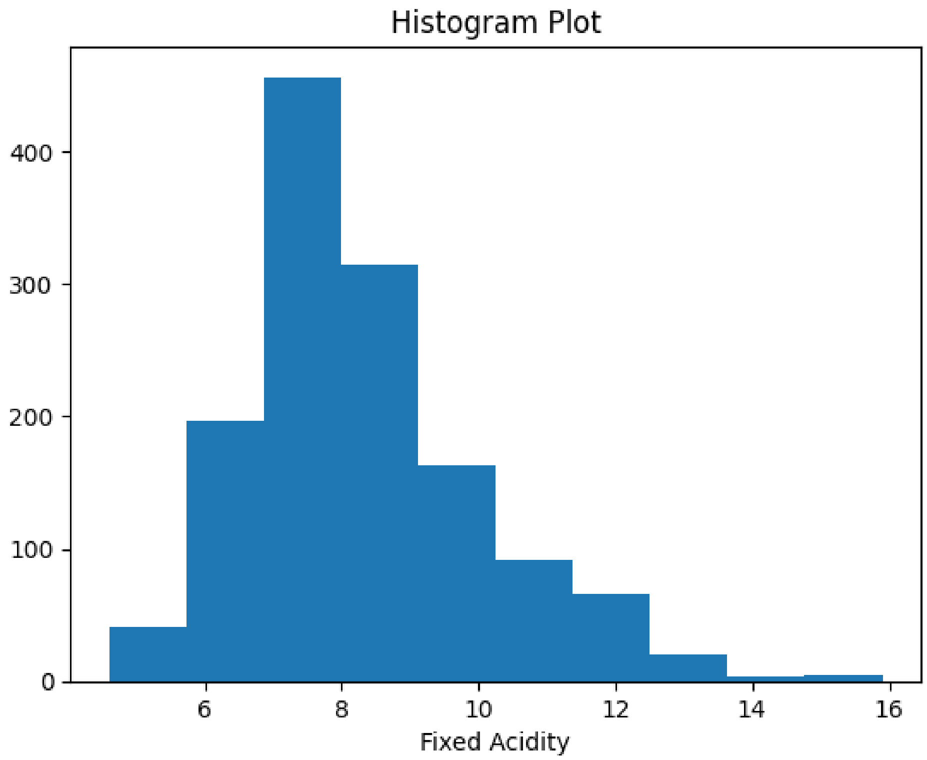
```
[ ] df.duplicated().sum()
```

```
240
```

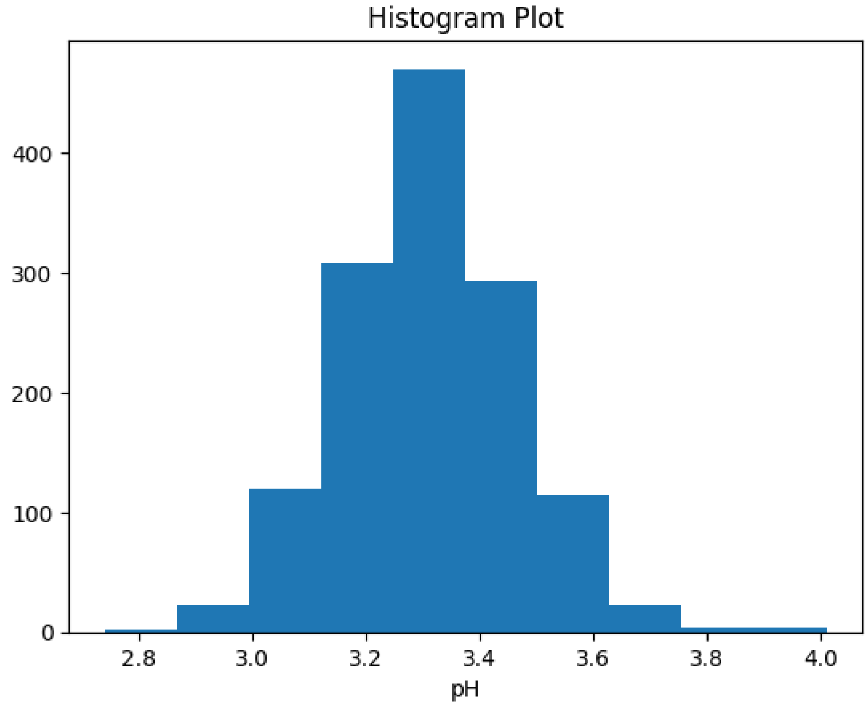
```
df.drop_duplicates(subset = None, keep='first', inplace=True, ignore_index=False)
df.shape
```

```
(1359, 12)
```

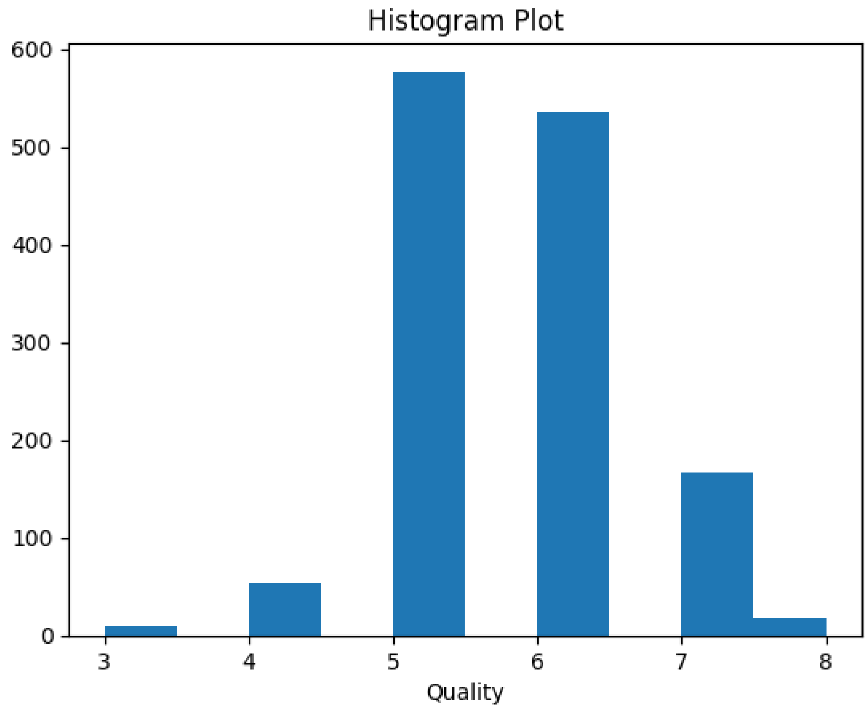
```
[ ] plt.hist(df['fixed acidity'])
    plt.title("Histogram Plot")
    plt.xlabel("Fixed Acidity")
    plt.show()
```



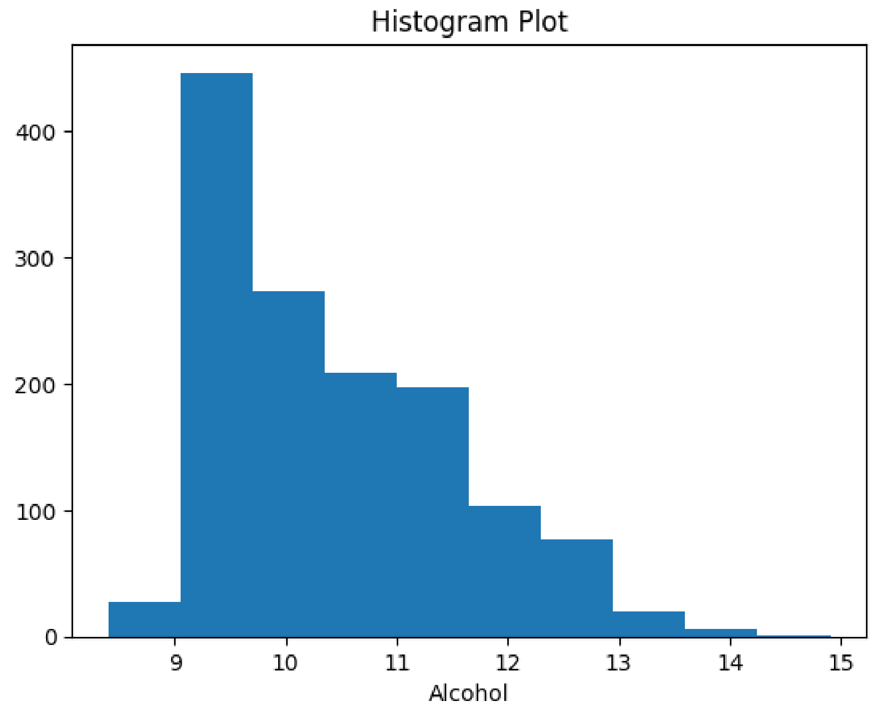
```
plt.hist(df['pH'])
plt.title("Histogram Plot")
plt.xlabel("pH")
plt.show()
```



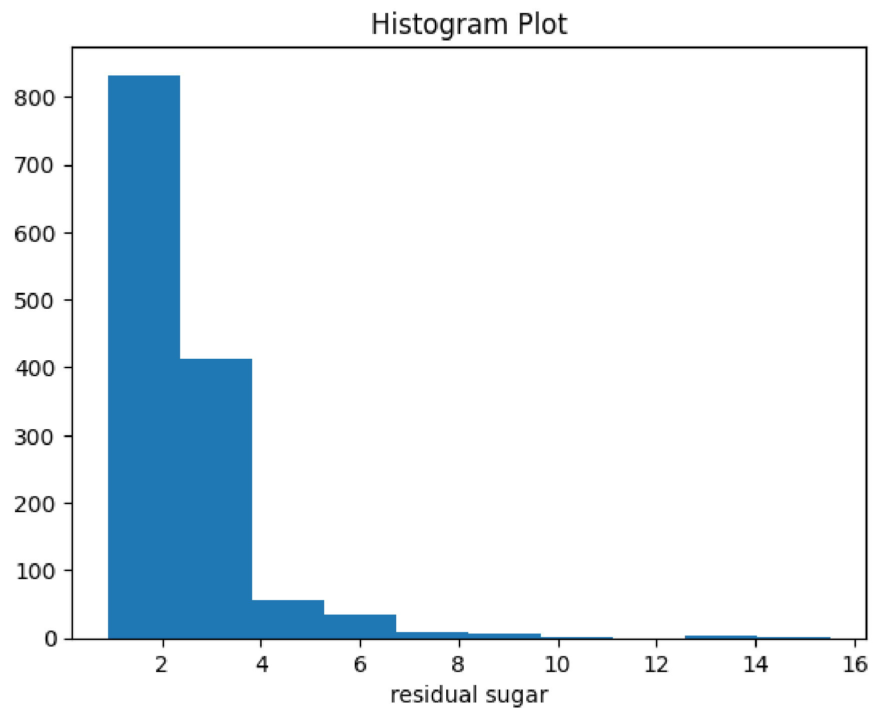

```
plt.hist(df['quality'])
plt.title("Histogram Plot")
plt.xlabel("Quality")
plt.show()
```



```
plt.hist(df['alcohol'])
plt.title("Histogram Plot")
plt.xlabel("Alcohol")
plt.show()
```

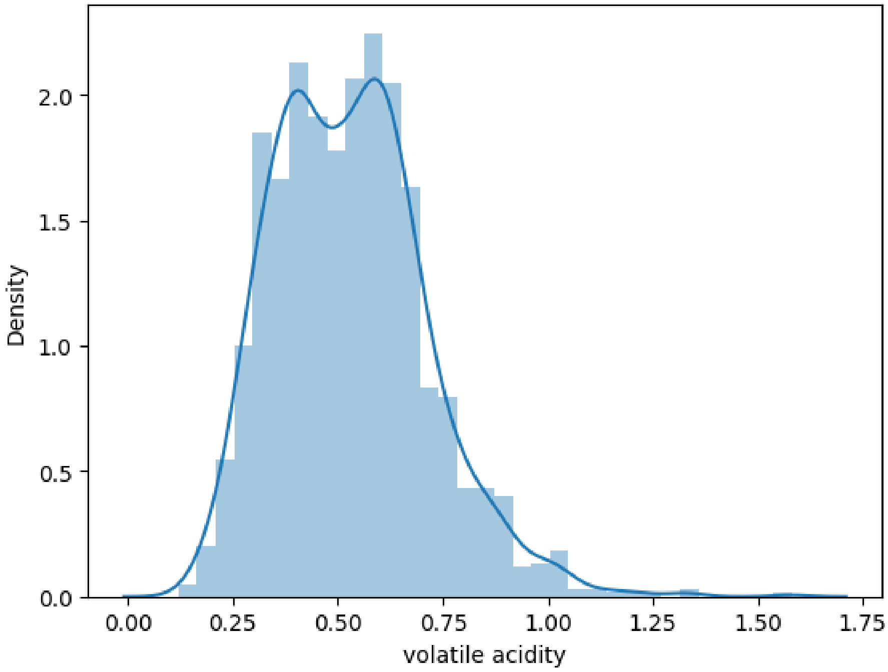



```
plt.hist(df['residual sugar'])
plt.title("Histogram Plot")
plt.xlabel("residual sugar")
plt.show()
```



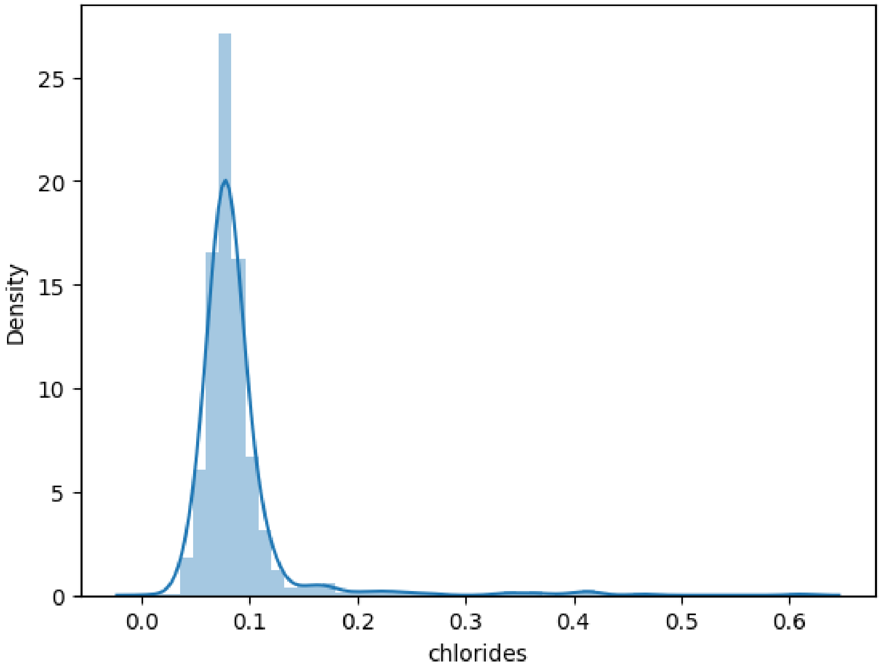
```
[16] sns.distplot(df['volatile acidity'])
plt.show()
```

```
sns.distplot(df['volatile acidity'])
```

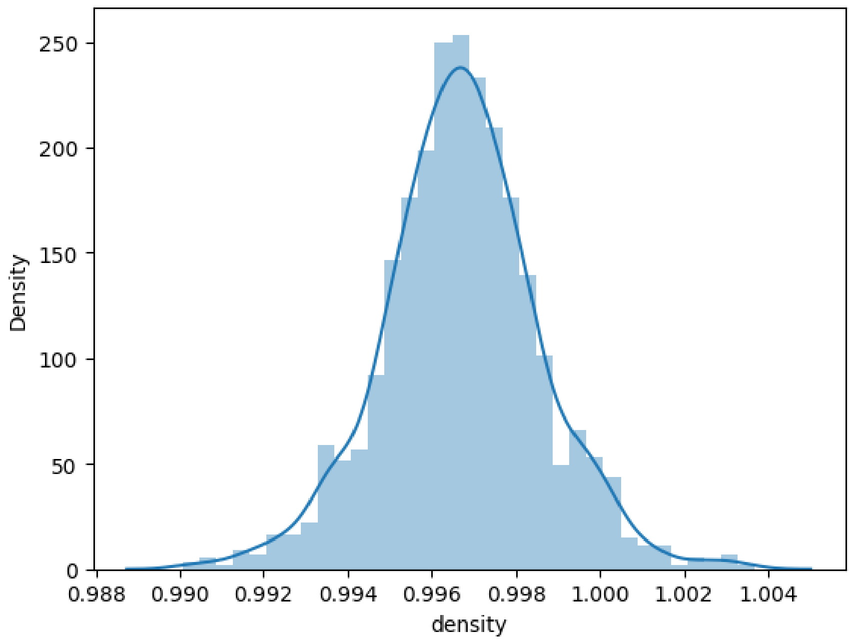



```
[17] sns.distplot(df['chlorides'])
```

```
sns.distplot(df['chlorides'])
<Axes: xlabel='chlorides', ylabel='Density'>
```



```
[18] sns.distplot(df['density'])
```



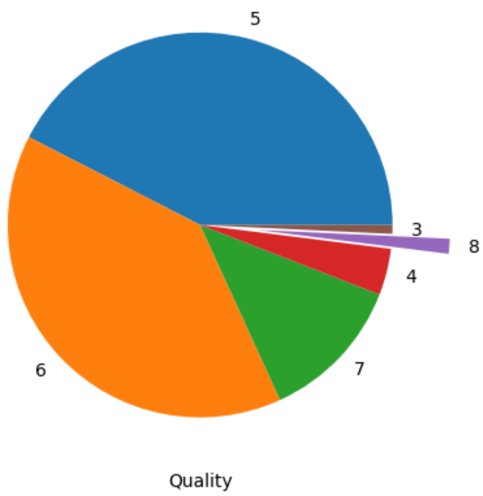

```
[20] df['quality'].unique()
```

```
array([5, 6, 7, 4, 8, 3])
```

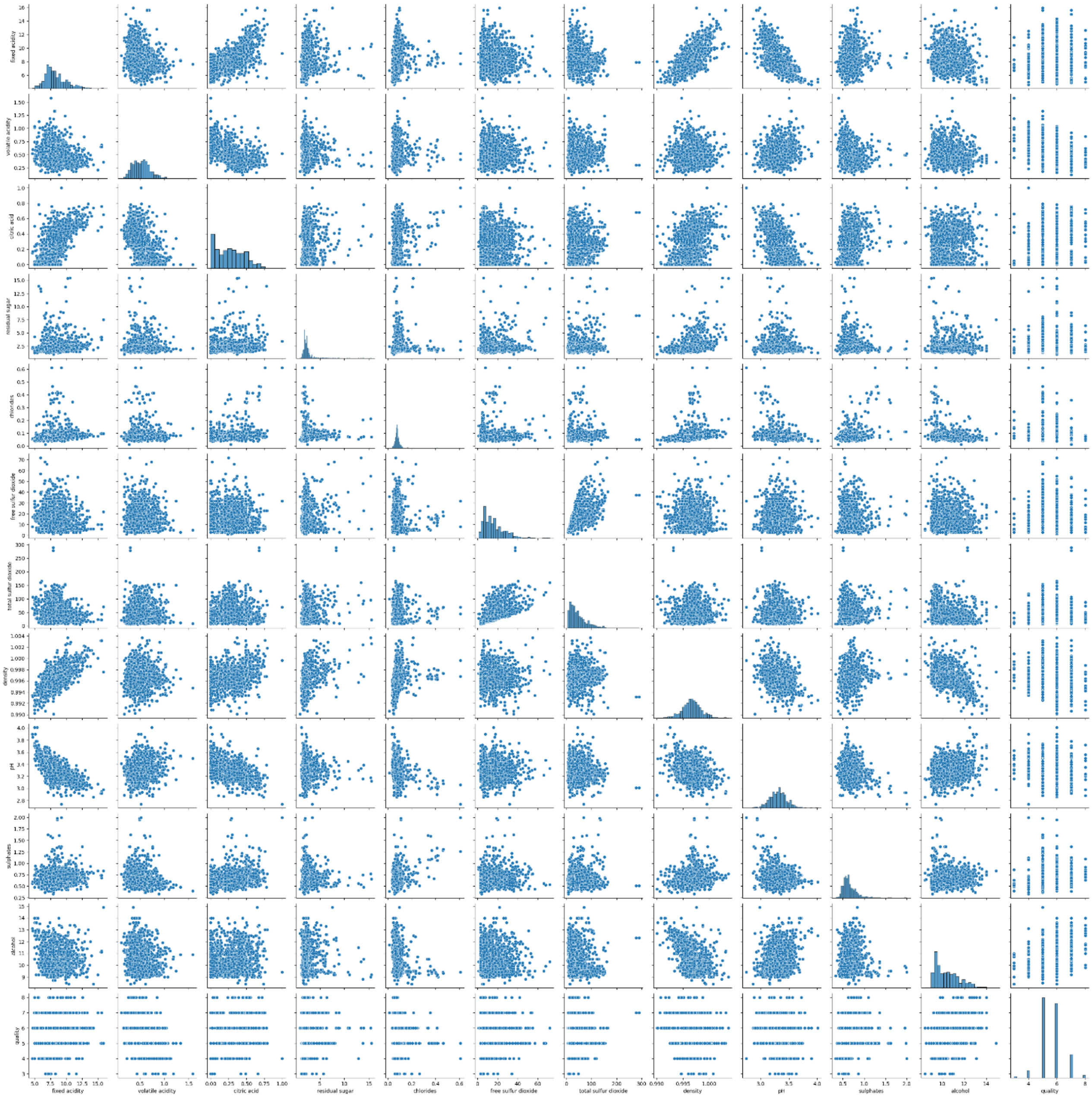
```
[21] df['quality'].value_counts()
```

```
5    577
6    535
7    167
4     53
8     17
3     10
Name: quality, dtype: int64
```

```
[22] labels =[5, 6, 7, 4, 8, 3]
     plt.pie(df['quality'].value_counts(), [0,0,0,0,0.3,0],labels=labels)
     plt.xlabel("Quality")
     plt.show()
```



```
plt.figure(figsize = (6,3))
sns.pairplot(df)
plt.show()
```

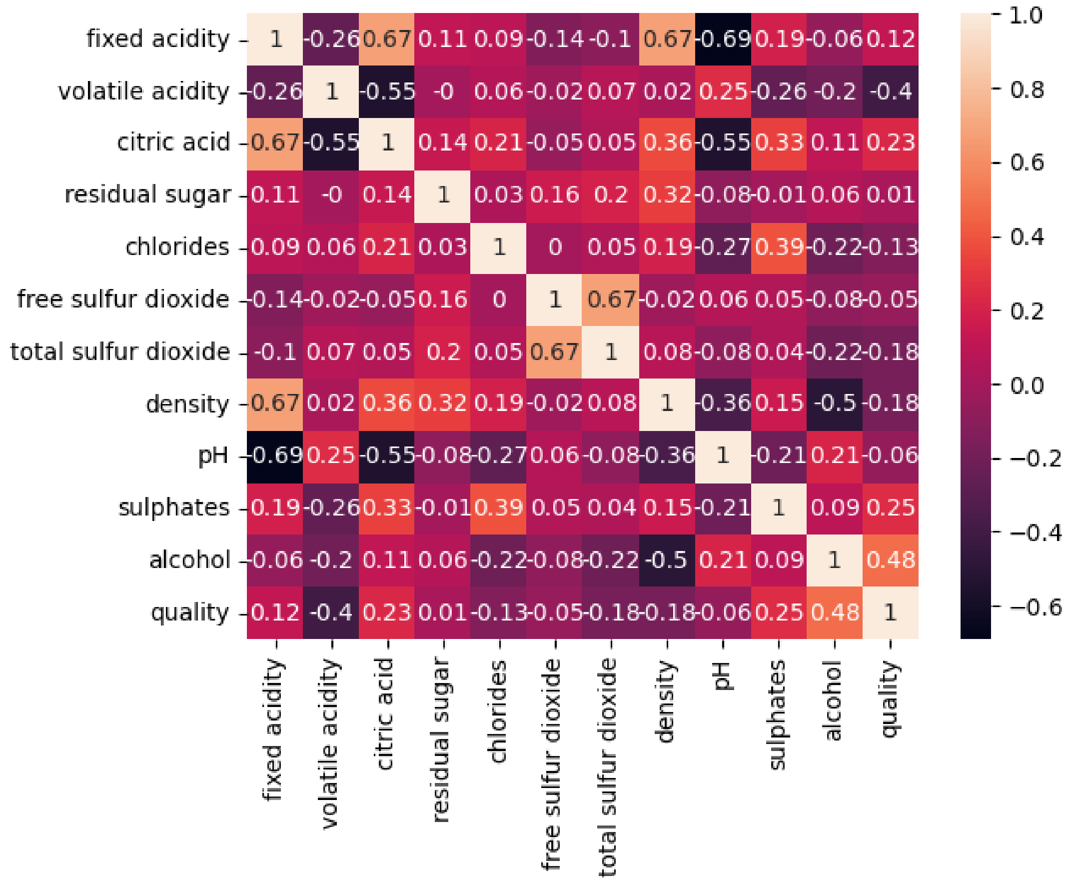


```
[23] df.corr()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.000000	-0.255124	0.667437	0.111025	0.085886	-0.140580	-0.103777	0.670195	-0.686685	0.190269	-0.061596	0.119024
volatile acidity	-0.255124	1.000000	-0.551248	-0.002449	0.055154	-0.020945	0.071701	0.023943	0.247111	-0.256948	-0.197812	-0.395214
citric acid	0.667437	-0.551248	1.000000	0.143892	0.210195	-0.048004	0.047358	0.357962	-0.550310	0.326062	0.105108	0.228057
residual sugar	0.111025	-0.002449	0.143892	1.000000	0.026656	0.160527	0.201038	0.324522	-0.083143	-0.011837	0.063281	0.013640
chlorides	0.085886	0.055154	0.210195	0.026656	1.000000	0.000749	0.045773	0.193592	-0.270893	0.394557	-0.223824	-0.130988
free sulfur dioxide	-0.140580	-0.020945	-0.048004	0.160527	0.000749	1.000000	0.667246	-0.018071	0.056631	0.054126	-0.080125	-0.050463
total sulfur dioxide	-0.103777	0.071701	0.047358	0.201038	0.045773	0.667246	1.000000	0.078141	-0.079257	0.035291	-0.217829	-0.177855
density	0.670195	0.023943	0.357962	0.324522	0.193592	-0.018071	0.078141	1.000000	-0.355617	0.146036	-0.504995	-0.184252
pH	-0.686685	0.247111	-0.550310	-0.083143	-0.270893	0.056631	-0.079257	-0.355617	1.000000	-0.214134	0.213418	-0.055245
sulphates	0.190269	-0.256948	0.326062	-0.011837	0.394557	0.054126	0.035291	0.146036	-0.214134	1.000000	0.091621	0.248835
alcohol	-0.061596	-0.197812	0.105108	0.063281	-0.223824	-0.080125	-0.217829	-0.504995	0.213418	0.091621	1.000000	0.480343
quality	0.119024	-0.395214	0.228057	0.013640	-0.130988	-0.050463	-0.177855	-0.184252	-0.055245	0.248835	0.480343	1.000000

```
sns.heatmap(round(df.corr(), 2), annot = True)
```

<Axes: >



Machine Learning Model building

LinearRegression

```
[25] from sklearn.linear_model import LinearRegression
     from sklearn.model_selection import train_test_split
```

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

```
[27] X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = .30, random_state = 100)
     model = LinearRegression()
     model.fit(X_train, y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[28] model.coef_
```

```
array([ 8.82657098e-04, -1.04445981e+00, -2.24821629e-02, -8.78724391e-03,
       -1.87747130e+00,  3.50743758e-03, -2.98858684e-03,  1.20926558e+01,
       -4.50913815e-01,  7.53163363e-01,  3.09192788e-01])
```

```
[29] model.intercept_
```

```
-7.814500198162955
```

```
[30] cdf = pd.DataFrame(model.coef_, X.columns, columns = ['coef'])
     cdf
```

	coef
fixed acidity	0.000883
volatile acidity	-1.044460
citric acid	-0.022482
residual sugar	-0.008787
chlorides	-1.877471
free sulfur dioxide	0.003507
total sulfur dioxide	-0.002989
density	12.092656
pH	-0.450914
sulphates	0.753163
alcohol	0.309193

```
[52] predictions = model.predict(X_test)
     predictions
```

```
5.41825747, 5.6106855 , 5.11524635, 5.51916807, 5.52356774,
5.66452959, 5.61424993, 6.21443215, 5.65830698, 5.80740641,
5.34236374, 6.18970555, 5.02576985, 5.0500667 , 6.2908901 ,
6.14334669, 5.68404463, 5.39955916, 5.69689581, 4.95143588,
5.67052065, 4.96402255, 6.33589581, 5.43159913, 5.87539915,
5.30146795, 5.29120578, 6.00312799, 6.64424644, 5.75186911,
5.98551231, 6.95667413, 5.89577837, 5.81403918, 5.08258789,
4.78698734, 6.01223488, 5.75252897, 6.04351791, 6.40175768,
6.29049117, 5.44461587, 6.48204029, 6.37509443, 5.61785473,
5.52864608, 5.37761807, 6.24933435, 5.23673935, 5.20973899,
5.14085583, 6.1998214 , 5.6623879 , 5.42188294, 5.73810237,
4.88055742, 6.42163682, 6.02573495, 5.70443952, 5.76093995,
5.06608663, 5.01035374, 5.56266691, 5.5943268 , 5.12513864,
5.23875985, 5.82929584, 5.55615838, 5.87565273, 5.2575833 ,
5.82123201, 5.95006561, 5.61486496, 5.66095371, 6.33049879,
5.3131268 , 5.01902972, 6.49351491, 5.32526058, 5.26998314,
5.57821655, 5.19615037, 5.33226504, 4.98108167, 5.85241247,
5.15407274, 5.99438   , 6.26890797, 5.73852351, 5.97798849,
6.20760882, 5.75605287, 5.31624921, 5.39935703, 5.03487146,
5.17137477, 5.84749725, 5.06010971, 5.58426893, 5.00593568,
5.22950265, 5.30325659, 4.92629585, 6.96472412, 5.39150501,
5.27174237, 5.60441355, 5.48199229, 5.1902976 , 4.92223503,
5.34753128, 5.92448947, 5.19977833, 5.42294069, 4.92333742,
4.87075579, 5.47579929, 6.04304586, 5.35399815, 5.67827697,
5.63910553, 6.3296033 , 6.03847041, 5.78384811, 4.93626843,
5.43325454, 5.41691961, 5.87409152, 5.15516169, 5.83453202,
5.40310369, 6.62625285, 5.66489416, 6.41408019, 5.49746938,
5.43396993, 4.94204116, 5.51972088, 5.29347191, 5.57632383,
5.49515598, 5.19135683, 5.75044961, 5.03420408, 5.78727203,
5.35953471, 5.38951279, 5.51727761, 5.34949077, 5.92692111,
5.81366382, 5.19977205, 5.19195729, 5.13153779, 5.12363638,
6.13495759, 6.16201238, 5.9315704 , 5.08364193, 5.49740149,
6.22522466, 5.16396859, 5.80398088, 6.1564361 , 5.35190791,
6.65967594, 5.62228272, 5.26768446, 5.14300129, 4.9249084 ,
5.1614825 , 5.3079856 , 5.84682518, 5.59837282, 5.70096898,
5.45216772, 6.00943676, 5.6639586 , 6.55688064, 5.24032559,
5.67955412, 6.26808934, 5.50830121, 5.53483336, 4.89665549,
5.6113892 , 4.81809429, 6.4955782 , 5.56178373, 6.05156124,
5.1727363 , 5.52508481, 5.00469373, 5.96476922, 5.22735759,
5.35114422, 6.18651317, 6.18716445, 6.67197923, 5.813316  ,
6.43242895, 5.86953901, 5.64179997, 5.21005535, 5.4512213 ,
6.60193625, 5.67304161, 5.56963   , 5.07060802, 6.06618944,
5.83571746, 6.80757096, 6.75619871, 5.31813911, 4.80789969,
5.6433768 , 6.48912425, 5.42848262, 5.50788123, 4.99650438,
5.04258776, 5.34831683, 4.86555744, 6.28856571, 5.35356334,
5.56519499, 6.75484006, 5.17736078, 5.61818433, 5.09446985,
5.464777  , 5.28369571, 6.18381851, 5.88112447, 4.96968273,
6.11399226, 5.96155976, 5.19392579, 5.76430913, 5.06347491,
5.39203891, 6.11902071, 5.69315541, 5.65111863, 5.06069734,
6.25231293, 6.57815258, 5.18285502, 5.86284869, 4.84060679,
6.15382871, 5.16280762, 5.20244015, 5.96951545, 5.35749529,
6.17991386, 5.20786956, 5.06391784, 6.4224535 , 6.47558471,
5.96169027, 5.14072652, 5.52312853, 5.52097944, 6.3751867 ,
6.32318345, 5.78087318, 6.13723861, 5.03260913, 5.70832145,
4.99869104, 5.35141248, 5.63018128, 5.79254373, 6.01745181,
5.63437811, 5.5263634 , 5.43956262, 6.20201743, 5.46911145,
5.84803411, 5.13103556, 5.1138624 , 4.85453986, 5.26280005,
6.26985797, 6.77667977, 5.0832558 ])
```

Evaluating

```
from sklearn import metrics
from sklearn.metrics import accuracy_score
```

```
[33] MAE = metrics.mean_absolute_error(y_test, predictions)
     MAE
```

```
0.5249898285704159
```

```
[34] MSE = metrics.mean_squared_error(y_test, predictions)
     MSE
```

```
0.473853947526031
```

```
[35] RMSE = np.sqrt(MSE)
     RMSE
```

```
0.6883705016384933
```

```
[36] r2= metrics.r2_score(y_test, predictions)
     r2
```

```
0.3541180613543832
```

Predicting Random Observations

```
[37] print(model.predict([[6.8,0.6,0.23,5.5,0.041,6,13,0.99532,3.62,0.43,14.3]]))
```

```
[6.56561315]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X doe
  warnings.warn(
```

```
[38] print(model.predict([[7.8,0.4,0.53,5.5,0.071,6,16,0.99732,3.62,0.63,12.3]]))
```

```
[6.25978563]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X doe
  warnings.warn(
```

```
[39] print(model.predict([[9.9,0.4,0.56,6.2,0.1,6.0,19.0,0.999,3.4,0.82,11.3]]))
```

```
[6.1448262]
```

Decision Tree Classifier

```
from sklearn.tree import DecisionTreeClassifier
dt= DecisionTreeClassifier()
dt.fit(X_train,y_train)
predictiondt= dt.predict(X_test)
predictiondt
```

```
array([6, 6, 6, 5, 5, 5, 7, 6, 6, 5, 6, 5, 5, 5, 6, 6, 6, 6, 4, 7, 5, 4,
       7, 6, 5, 5, 5, 5, 5, 5, 5, 6, 6, 5, 7, 7, 7, 5, 5, 6, 5, 5, 5, 5,
       5, 6, 5, 6, 5, 5, 7, 5, 6, 5, 8, 6, 5, 5, 7, 5, 6, 5, 7, 6, 6, 6,
       6, 6, 5, 5, 6, 5, 6, 5, 7, 6, 5, 5, 6, 5, 6, 5, 6, 5, 6, 6, 5, 5,
       5, 5, 6, 6, 6, 5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 6, 5, 7, 7, 6, 7, 5,
       4, 6, 6, 5, 6, 3, 7, 6, 5, 4, 6, 5, 4, 6, 5, 5, 6, 5, 6, 6, 5, 6,
       5, 3, 7, 6, 6, 6, 6, 5, 6, 5, 7, 6, 6, 5, 5, 6, 6, 6, 7, 7, 7, 5,
       5, 4, 5, 6, 6, 6, 7, 6, 7, 6, 6, 5, 6, 7, 5, 6, 5, 7, 5, 6, 5, 6,
       7, 7, 6, 6, 5, 5, 5, 6, 5, 7, 6, 6, 5, 5, 6, 6, 5, 6, 6, 6, 6, 6,
       5, 5, 6, 5, 5, 5, 5, 5, 5, 6, 5, 5, 6, 6, 5, 6, 5, 5, 6, 5, 5, 5,
       3, 6, 5, 7, 5, 6, 6, 6, 5, 5, 5, 5, 5, 3, 3, 5, 6, 6, 6, 6, 6, 6,
       6, 6, 5, 5, 6, 5, 5, 6, 5, 7, 5, 7, 6, 6, 5, 6, 5, 5, 6, 5, 6, 5,
       6, 5, 5, 7, 5, 6, 7, 5, 5, 5, 5, 7, 5, 6, 5, 6, 6, 6, 5, 5, 7, 6,
       5, 5, 6, 5, 5, 6, 6, 6, 5, 6, 6, 5, 6, 6, 6, 7, 5, 5, 5, 6, 5, 6,
       5, 7, 5, 6, 4, 6, 6, 5, 6, 6, 6, 6, 7, 6, 5, 3, 6, 4, 5, 5, 5, 6,
       6, 8, 6, 6, 4, 6, 6, 6, 5, 5, 5, 6, 5, 7, 5, 5, 6, 5, 6, 5, 6, 4,
       7, 5, 5, 7, 6, 5, 6, 5, 5, 7, 6, 7, 5, 7, 7, 5, 5, 5, 5, 6, 5, 6,
       7, 6, 5, 5, 6, 8, 6, 5, 5, 6, 7, 7, 6, 7, 5, 6, 5, 5, 7, 4, 6, 4,
       6, 5, 7, 6, 6, 3, 4, 6, 6, 5, 7, 4])
```

```
[41] accuracy_score(y_test, predictiondt)
```

```
0.5294117647058824
```

Predicting for random values

```
[45] print(dt.predict([[6.8,0.6,0.23,5.5,0.041,6,13,0.99532,3.62,0.43,14.3]]))
```

```
[5]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X
  warnings.warn(
```

```
[47] print(dt.predict([[7.8,0.4,0.53,5.5,0.071,6,16,0.99732,3.62,0.63,12.3]]))
```

```
[7]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X
  warnings.warn(
```

```
print(dt.predict([[9.9,0.4,0.56,6.2,0.1,6.0,19.0,0.999,3.4,0.82,11.3]]))
```

```
[7]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X
  warnings.warn(
```

Random Forest Classifier

```python
from sklearn.ensemble import RandomForestClassifier
rf= RandomForestClassifier()
rf.fit(X_train,y_train)
predictionrf= rf.predict(X_test)
predictionrf
```

```
array([6, 7, 6, 7, 5, 5, 6, 6, 7, 6, 6, 6, 5, 6, 5, 6, 5, 6, 6, 7, 5, 5,
       7, 5, 5, 6, 5, 5, 5, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 5, 5, 5, 5,
       5, 6, 7, 5, 5, 5, 7, 5, 6, 5, 7, 6, 5, 5, 7, 5, 6, 5, 6, 6, 6, 6,
       6, 6, 5, 5, 5, 5, 6, 5, 6, 7, 6, 5, 6, 6, 6, 6, 7, 5, 6, 6, 5, 5,
       5, 5, 6, 5, 6, 6, 5, 6, 5, 5, 6, 5, 5, 6, 6, 6, 5, 7, 7, 6, 7, 6,
       6, 6, 6, 5, 6, 5, 7, 6, 5, 5, 6, 5, 5, 6, 5, 5, 6, 7, 5, 6, 5, 6,
       5, 5, 7, 6, 6, 5, 6, 5, 6, 5, 6, 6, 6, 6, 5, 6, 6, 6, 7, 6, 6, 6,
       5, 5, 6, 6, 6, 6, 6, 6, 7, 6, 6, 5, 6, 7, 5, 5, 5, 7, 5, 5, 6, 6,
       6, 5, 6, 6, 5, 5, 5, 5, 5, 5, 6, 6, 5, 5, 6, 6, 6, 6, 6, 5, 5, 6,
       5, 5, 6, 5, 6, 5, 6, 5, 6, 6, 5, 6, 6, 6, 5, 6, 5, 5, 6, 5, 6, 5,
       5, 5, 5, 7, 5, 5, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6,
       6, 6, 6, 5, 5, 6, 5, 6, 5, 7, 6, 6, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5,
       6, 5, 6, 5, 5, 6, 6, 5, 5, 5, 5, 6, 6, 7, 5, 6, 5, 6, 6, 6, 6, 6,
       5, 5, 5, 5, 5, 6, 6, 6, 5, 5, 6, 6, 6, 5, 6, 6, 5, 6, 5, 6, 5, 6,
       5, 6, 5, 6, 5, 6, 6, 6, 6, 6, 7, 6, 6, 6, 5, 5, 5, 6, 5, 5, 6, 6,
       6, 6, 7, 6, 5, 5, 7, 5, 6, 5, 5, 6, 5, 7, 6, 6, 7, 5, 6, 5, 6, 5,
       6, 5, 5, 6, 6, 5, 6, 5, 6, 6, 6, 5, 5, 6, 7, 5, 6, 5, 6, 6, 5, 6,
       5, 6, 5, 5, 7, 7, 6, 5, 5, 6, 7, 6, 6, 6, 5, 6, 5, 5, 6, 6, 6, 5,
       5, 5, 6, 5, 6, 5, 5, 6, 5, 6, 8, 5])
```

```python
[43] accuracy_score(y_test, predictionrf)
```

```
0.5882352941176471
```

Predicting for random values

```python
[44] print(rf.predict([[6.8,0.6,0.23,5.5,0.041,6,13,0.99532,3.62,0.43,14.3]]))
```

```
[5]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
  warnings.warn(
```

```python
[48] print(rf.predict([[9.9,0.4,0.56,6.2,0.1,6.0,19.0,0.999,3.4,0.82,11.3]]))
```

```
[6]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
  warnings.warn(
```

```python
[46] print(rf.predict([[7.8,0.4,0.53,5.5,0.071,6,16,0.99732,3.62,0.63,12.3]]))
```

```
[6]
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning:
  warnings.warn(
```