```
pip install ucimlrepo
```

```
Collecting ucimlrepo
  Downloading ucimlrepo-0.0.6-py3-none-any.whl (8.0 kB)
Installing collected packages: ucimlrepo
Successfully installed ucimlrepo-0.0.6
```

```
from ucimlrepo import fetch_ucirepo

# fetch dataset
wine = fetch_ucirepo(id=109)

# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets

# metadata
print(wine.metadata)

# variable information
print(wine.variables)
```

```
{'uci_id': 109, 'name': 'Wine', 'repository_url': 'https://archive.ics.uci.edu/dataset/109/wine', 'data_url': 'https://archive.ics.uci.edu/static/public/109/data.csv', 'abstract': 'U
                          name     role         type demographic  \
0                        class   Target  Categorical        None
1                      Alcohol  Feature   Continuous        None
2                    Malicacid  Feature   Continuous        None
3                          Ash  Feature   Continuous        None
4              Alcalinity_of_ash  Feature   Continuous        None
5                    Magnesium  Feature      Integer        None
6                Total_phenols  Feature   Continuous        None
7                   Flavanoids  Feature   Continuous        None
8           Nonflavanoid_phenols  Feature   Continuous        None
9                Proanthocyanins  Feature   Continuous        None
10               Color_intensity  Feature   Continuous        None
11                           Hue  Feature   Continuous        None
12  0D280_0D315_of_diluted_wines  Feature   Continuous        None
13                       Proline  Feature      Integer        None

   description units missing_values
0         None  None             no
1         None  None             no
2         None  None             no
3         None  None             no
4         None  None             no
5         None  None             no
6         None  None             no
7         None  None             no
8         None  None             no
9         None  None             no
10        None  None             no
11        None  None             no
12        None  None             no
13        None  None             no
```

```
# This Python 3 environment comes with many helpful analytics libraries installed
# It is defined by the kaggle/python docker image: https://github.com/kaggle/docker-python

# For example, here's several helpful packages to Load in

import numpy as np # Linear algebra
import pandas as pd # data processing, CSV file I/0 (e.g. pd.read_csv)
import matplotlib.pyplot as plt # data visualization
import seaborn as sns # statistical data visualization
%matplotlib inline

# Input data files are available in the " .. /input/" directory.
# For example, running this (by clicking run or pressing Shift+Enter) will list all files under the input directory

import os
for dirname, _, filenames in os.walk('/kaggle/input'):
 for filename in filenames:
  print(os.path.join(dirname, filename))

# Any results you write to the current directory are saved as output.


import warnings

warnings.filterwarnings('ignore')


from ucimlrepo import fetch_ucirepo

# fetch dataset
wine = fetch_ucirepo(id=109)

# data (as pandas dataframes)
X = wine.data.features
y = wine.data.targets

# metadata
print(wine.metadata)

# variable information
print(wine.variables)


x1 = pd.DataFrame(X)
y1 = pd.DataFrame(y)
df = pd.concat([x1, y1], axis= 1)
df
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocy
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	
...	...	...	...	...	...	...	...	...	
173	13.71	5.65	2.45	20.5	95	1.68	0.61	0.52	
174	13.40	3.91	2.48	23.0	102	1.80	0.75	0.43	
175	13.27	4.28	2.26	20.0	120	1.59	0.69	0.43	
176	13.17	2.59	2.37	20.0	120	1.65	0.68	0.53	
177	14.13	4.10	2.74	24.5	96	2.05	0.76	0.56	

178 rows × 14 columns

Next steps: View recommended plots

```
df.shape
```

```
(178, 14)
```

```
df.head()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols	Proanthocyan
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	

Next steps: View recommended plots

```
col_names = df.columns
```

```
col_names
```

```
Index(['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium',
       'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols',
       'Proanthocyanins', 'Color_intensity', 'Hue',
       '0D280_0D315_of_diluted_wines', 'Proline', 'class'],
      dtype='object')
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178 entries, 0 to 177
Data columns (total 14 columns):
 #   Column                        Non-Null Count  Dtype  
---  ------                        --------------  -----  
 0   Alcohol                       178 non-null    float64
 1   Malicacid                     178 non-null    float64
 2   Ash                           178 non-null    float64
 3   Alcalinity_of_ash             178 non-null    float64
 4   Magnesium                     178 non-null    int64  
 5   Total_phenols                 178 non-null    float64
 6   Flavanoids                    178 non-null    float64
 7   Nonflavanoid_phenols          178 non-null    float64
 8   Proanthocyanins               178 non-null    float64
 9   Color_intensity               178 non-null    float64
 10  Hue                           178 non-null    float64
 11  0D280_0D315_of_diluted_wines  178 non-null    float64
 12  Proline                       178 non-null    int64  
 13  class                         178 non-null    int64  
dtypes: float64(11), int64(3)
memory usage: 19.6 KB
```

```
numerical = [var for var in df.columns if df[var].dtype != 'O']

print('There are {} numerical variables\n'.format(len(numerical)))

print('The numerical variables are :', numerical)
```

```
There are 14 numerical variables

The numerical variables are : ['Alcohol', 'Malicacid', 'Ash', 'Alcalinity_of_ash', 'Magnesium', 'Total_phenols', 'Flavanoids', 'Nonflavanoid_phenols', 'Proanthocyanins', 'Color_intens
```

```
print(round(df[numerical].describe()),2)
```

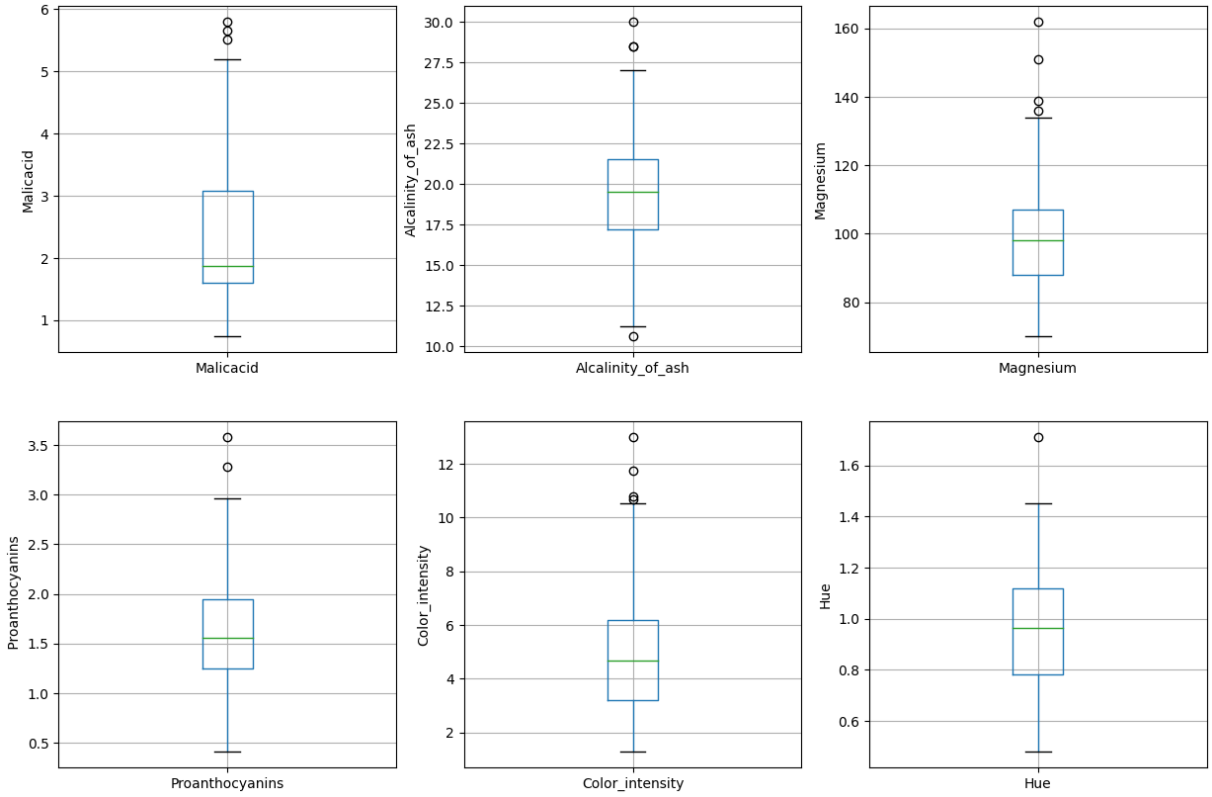
```
       Alcohol  Malicacid    Ash  Alcalinity_of_ash  Magnesium  Total_phenols  \
count    178.0      178.0  178.0              178.0      178.0          178.0   
mean      13.0        2.0    2.0               19.0      100.0            2.0   
std        1.0        1.0    0.0                3.0       14.0            1.0   
min       11.0        1.0    1.0               11.0       70.0            1.0   
25%       12.0        2.0    2.0               17.0       88.0            2.0   
50%       13.0        2.0    2.0               20.0       98.0            2.0   
75%       14.0        3.0    3.0               22.0      107.0            3.0   
max       15.0        6.0    3.0               30.0      162.0            4.0   

       Flavanoids  Nonflavanoid_phenols  Proanthocyanins  Color_intensity  \
count       178.0                 178.0            178.0            178.0   
mean          2.0                   0.0              2.0              5.0   
std           1.0                   0.0              1.0              2.0   
min           0.0                   0.0              0.0              1.0   
25%           1.0                   0.0              1.0              3.0   
50%           2.0                   0.0              2.0              5.0   
75%           3.0                   0.0              2.0              6.0   
max           5.0                   1.0              4.0             13.0   

         Hue  0D280_0D315_of_diluted_wines  Proline  class  
count  178.0                         178.0    178.0  178.0  
mean     1.0                           3.0    747.0    2.0  
std      0.0                           1.0    315.0    1.0  
min      0.0                           1.0    278.0    1.0  
25%      1.0                           2.0    500.0    1.0  
```

```
       50%      1.0                          3.0    674.0    2.0
       75%      1.0                          3.0    985.0    3.0
       max      2.0                          4.0   1680.0    3.0   2
```

```
Index([1, 13, 14, 50, 59, 73, 121, 122, 127, 157], dtype='int64')
```

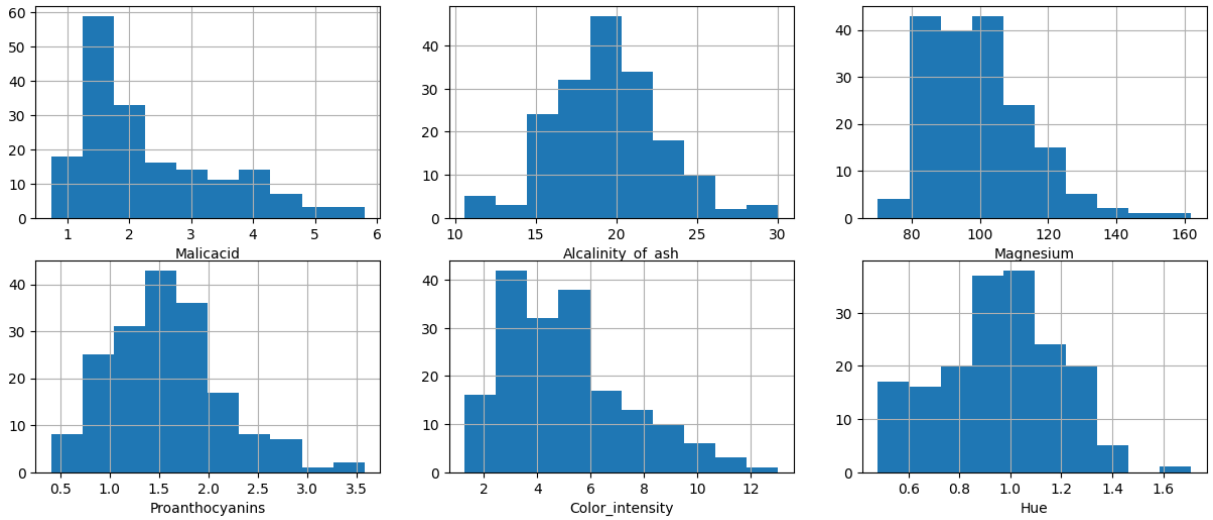
```
plt.figure(figsize=(15,10))

plt.subplot(2, 3, 1)
fig = df.boxplot(column='Malicacid')
fig.set_title('')
fig.set_ylabel('Malicacid')

plt.subplot(2, 3, 2)
fig = df.boxplot(column='Alcalinity_of_ash')
fig.set_title('')
fig.set_ylabel('Alcalinity_of_ash')

plt.subplot(2, 3, 3)
fig = df.boxplot(column='Magnesium')
fig.set_title('')
fig.set_ylabel('Magnesium')

plt.subplot(2, 3, 4)
fig = df.boxplot(column='Proanthocyanins')
fig.set_title('')
fig.set_ylabel(' Proanthocyanins')

plt.subplot(2, 3, 5)
fig = df.boxplot(column='Color_intensity')
fig.set_title('')
fig.set_ylabel('Color_intensity')

plt.subplot(2, 3, 6)
fig = df.boxplot(column='Hue')
fig.set_title('')
fig.set_ylabel('Hue')
```

```
Text(0, 0.5, 'Hue')
```


Malicacid
Malicacid
Alcalinity_of_ash
Alcalinity_of_ash
Magnesium
Magnesium
Proanthocyanins
Proanthocyanins
Color_intensity
Color_intensity
Hue
Hue

```
plt.figure(figsize=(15,6))

plt.subplot(2, 3, 1)
fig = df.Malicacid.hist(bins=10)
fig.set_xlabel('Malicacid')
fig.set_ylabel('')

plt.subplot(2, 3, 2)
fig = df.Alcalinity_of_ash.hist(bins=10)
fig.set_xlabel('Alcalinity_of_ash')
fig.set_ylabel('')

plt.subplot(2, 3, 3)
fig = df.Magnesium.hist(bins=10)
fig.set_xlabel('Magnesium')
fig.set_ylabel('')

plt.subplot(2, 3, 4)
fig = df.Proanthocyanins.hist(bins=10)
fig.set_xlabel('Proanthocyanins')
fig.set_ylabel('')

plt.subplot(2, 3, 5)
fig = df.Color_intensity.hist(bins=10)
fig.set_xlabel('Color_intensity')
fig.set_ylabel('')

plt.subplot(2, 3, 6)
fig = df.Hue.hist(bins=10)
fig.set_xlabel('Hue')
fig.set_ylabel('')
```

```
Text(0, 0.5, '')
```




```
# since all 4 are skewed, next step would be interquartile range to find the outliers
IQR = df['Malicacid'].quantile(0.75) - df['Malicacid'].quantile(0.25)
Lower_fence = df['Malicacid'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Malicacid'].quantile(0.75) + (IQR * 1.5)
print(f"Malicacid outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Malicacid outliers are values < -0.6174999999999997  or > 5.3025
```

```
IQR = df['Alcalinity_of_ash'].quantile(0.75) - df['Alcalinity_of_ash'].quantile(0.25)
Lower_fence = df['Alcalinity_of_ash'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Alcalinity_of_ash'].quantile(0.75) + (IQR * 1.5)
print(f"Alcalinity_of_ash outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Alcalinity_of_ash outliers are values < 10.749999999999998  or > 27.950000000000003
```

```
IQR = df['Magnesium'].quantile(0.75) - df['Magnesium'].quantile(0.25)
Lower_fence = df['Magnesium'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Magnesium'].quantile(0.75) + (IQR * 1.5)
print(f"Magnesium outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Magnesium outliers are values < 59.5  or > 135.5
```

```
IQR = df['Magnesium'].quantile(0.75) - df['Magnesium'].quantile(0.25)
Lower_fence = df['Magnesium'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Magnesium'].quantile(0.75) + (IQR * 1.5)
print(f"Magnesium outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Magnesium outliers are values < 59.5  or > 135.5
```

```
IQR = df['Proanthocyanins'].quantile(0.75) - df['Proanthocyanins'].quantile(0.25)
Lower_fence = df['Proanthocyanins'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Proanthocyanins'].quantile(0.75) + (IQR * 1.5)
print(f"Proanthocyanins outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Proanthocyanins outliers are values < 0.20000000000000018  or > 3.0
```

```
IQR = df['Color_intensity'].quantile(0.75) - df['Color_intensity'].quantile(0.25)
Lower_fence = df['Color_intensity'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Color_intensity'].quantile(0.75) + (IQR * 1.5)
print(f"Color_intensity outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Color_intensity outliers are values < -1.2500000000000009  or > 10.670000000000002
```

```
IQR = df['Hue'].quantile(0.75) - df['Hue'].quantile(0.25)
Lower_fence = df['Hue'].quantile(0.25) - (IQR * 1.5)
Upper_fence = df['Hue'].quantile(0.75) + (IQR * 1.5)
print(f"Hue outliers are values < {Lower_fence}  or > {Upper_fence}")
```

```
Hue outliers are values < 0.2762499999999998  or > 1.6262500000000002
```

```
X = df.drop(['class'], axis=1)

y = df['class']
```

```
# split X and y into training and testing sets

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 0)
```

```
# check the shape of X_train and X_test

X_train.shape, X_test.shape
```

```
((142, 13), (36, 13))
```

```
# check data types in X_train

X_train.dtypes
```

```
Alcohol                         float64
Malicacid                       float64
Ash                             float64
Alcalinity_of_ash               float64
Magnesium                         int64
Total_phenols                   float64
Flavanoids                      float64
```

```
Nonflavanoid_phenols            float64
Proanthocyanins                 float64
Color_intensity                 float64
Hue                             float64
0D280_0D315_of_diluted_wines    float64
Proline                           int64
dtype: object
```

```
# display categorical variables

categorical = [col for col in X_train.columns if X_train[col].dtypes == 'O']

categorical
```

```
[]
```

```
# display numerical variables

numerical = [col for col in X_train.columns if X_train[col].dtypes != '0']

numerical
```

```
['Alcohol',
 'Malicacid',
 'Ash',
 'Alcalinity_of_ash',
 'Magnesium',
 'Total_phenols',
 'Flavanoids',
 'Nonflavanoid_phenols',
 'Proanthocyanins',
 'Color_intensity',
 'Hue',
 '0D280_0D315_of_diluted_wines',
 'Proline']
```

```
X_train.isnull().sum()
```

```
Alcohol                         0
Malicacid                       0
Ash                             0
Alcalinity_of_ash               0
Magnesium                       0
Total_phenols                   0
Flavanoids                      0
Nonflavanoid_phenols            0
Proanthocyanins                 0
Color_intensity                 0
Hue                             0
0D280_0D315_of_diluted_wines    0
Proline                         0
dtype: int64
```

```
X_test.isnull().any()
```

```
Alcohol                         False
Malicacid                       False
Ash                             False
Alcalinity_of_ash               False
```

```
Magnesium                     False
Total_phenols                 False
Flavanoids                    False
Nonflavanoid_phenols          False
Proanthocyanins               False
Color_intensity               False
Hue                           False
0D280_0D315_of_diluted_wines  False
Proline                       False
dtype: bool
```

```
def max_value(df3, variable, top):
    return np.where(df3[variable]>top, top, df3[variable])

for df3 in [X_train, X_test]:
    df3['Malicacid'] = max_value(df3, 'Malicacid', 5.30)
    df3['Alcalinity_of_ash'] = max_value(df3, 'Alcalinity_of_ash', 27.95)
    df3['Magnesium'] = max_value(df3, 'Magnesium', 135.5)
    df3['Proanthocyanins'] = max_value(df3, 'Proanthocyanins', 3)
    df3['Color_intensity'] = max_value(df3, 'Color_intensity', 10.67)
    df3['Hue'] = max_value(df3, 'Hue', 1.63)
```

```
X_train['Malicacid'].max(), X_test['Malicacid'].max()
```

```
(5.3, 5.3)
```

```
X_train['Alcalinity_of_ash'].max(), X_test['Alcalinity_of_ash'].max()
```

```
(27.95, 27.95)
```

```
X_train['Magnesium'].max(), X_test['Magnesium'].max()
```

```
(135.5, 132.0)
```

```
X_train['Proanthocyanins'].max(), X_test['Proanthocyanins'].max()
```

```
(3.0, 2.45)
```

```
X_train['Color_intensity'].max(), X_test['Color_intensity'].max()
```

```
(10.67, 10.67)
```

```
X_train['Hue'].max(), X_test['Hue'].max()
```

```
(1.63, 1.38)
```

```
X_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000
mean	12.984859	2.368662	2.366901	19.536620	99.739437	2.258662	1.949155	0.363521
std	0.807175	1.104345	0.269684	3.392529	13.154391	0.611691	0.975921	0.127709
min	11.030000	0.740000	1.360000	10.600000	70.000000	1.100000	0.470000	0.130000
25%	12.347500	1.602500	2.222500	17.250000	89.000000	1.705000	1.037500	0.270000
50%	13.040000	1.895000	2.360000	19.500000	98.000000	2.210000	2.035000	0.340000
75%	13.637500	3.222500	2.560000	21.500000	106.750000	2.735000	2.760000	0.450000
max	14.750000	5.300000	3.220000	27.950000	135.500000	3.880000	3.740000	0.660000

```
cols = X_train.columns
```

```
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

```
X_train = pd.DataFrame(X_train, columns=[cols])
```

```
X_test = pd.DataFrame(X_test, columns=[cols])
```

```
X_train.describe()
```

	Alcohol	Malicacid	Ash	Alcalinity_of_ash	Magnesium	Total_phenols	Flavanoids	Nonflavanoid_phenols
count	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000	142.000000
mean	0.525500	0.357163	0.541345	0.515079	0.454037	0.416785	0.452341	0.440606
std	0.216983	0.242181	0.144991	0.195535	0.200830	0.220033	0.298447	0.240960
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.354167	0.189145	0.463710	0.383285	0.290076	0.217626	0.173547	0.264151
50%	0.540323	0.253289	0.537634	0.512968	0.427481	0.399281	0.478593	0.396226
75%	0.700941	0.544408	0.645161	0.628242	0.561069	0.588129	0.700306	0.603774
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

```
# Model training
# train a logistic regression model on the training set
from sklearn.linear_model import LogisticRegression

# instantiate the model
logreg = LogisticRegression(solver='liblinear', random_state=0)

#fit the model
logreg.fit(X_train, y_train)
```

```
LogisticRegression(random_state=0, solver='liblinear')
```

```
# Predicting results
y_pred_test = logreg.predict(X_test)
y_pred_test
```

```
array([1, 3, 2, 1, 2, 2, 1, 3, 2, 2, 3, 3, 1, 2, 3, 2, 1, 1, 3, 1, 2, 1,
       1, 2, 2, 2, 2, 2, 2, 3, 1, 1, 2, 1, 1, 1])
```

```
# predict proba: predicts possibilities for the target variable
logreg.predict_proba(X_test)[:,0]
```

```
array([0.84608661, 0.08048314, 0.33554459, 0.80152316, 0.22039744,
       0.24522358, 0.87468447, 0.03848718, 0.15948773, 0.05898538,
       0.14507607, 0.0358635 , 0.93262766, 0.47681744, 0.07875799,
       0.12976804, 0.79007758, 0.95638587, 0.08820506, 0.84199176,
       0.47786966, 0.67550962, 0.47265773, 0.23584062, 0.08901126,
       0.15152147, 0.20646317, 0.04988917, 0.07608428, 0.07232845,
       0.83715721, 0.86643051, 0.0791357 , 0.81872064, 0.87889396,
       0.67623076])
```

```
logreg.predict_proba(X_test)[:,1]
```

```
array([0.10773965, 0.04382495, 0.65481533, 0.1525236 , 0.66114073,
       0.74433276, 0.07438344, 0.12116596, 0.78373262, 0.80051396,
       0.1289791 , 0.07885248, 0.03352518, 0.51535383, 0.06169256,
       0.85700534, 0.14896474, 0.01820935, 0.40806803, 0.14198268,
       0.51407897, 0.25405006, 0.43617131, 0.73190936, 0.59297914,
       0.78249892, 0.7421782 , 0.86155799, 0.73512681, 0.0449725 ,
       0.12680024, 0.09818553, 0.64410061, 0.05781283, 0.08848967,
       0.30025453])
```

```
logreg.predict_proba(X_test)[:,2]
```

```
array([0.04617374, 0.87569191, 0.00964009, 0.04595323, 0.11846182,
       0.01044366, 0.05093209, 0.84034686, 0.05677965, 0.14050066,
       0.72594482, 0.88528401, 0.03384715, 0.00782873, 0.85954945,
       0.01322662, 0.06095768, 0.02540478, 0.50372691, 0.01602556,
       0.00805136, 0.07044032, 0.09117095, 0.03225002, 0.31800959,
       0.06597962, 0.05135863, 0.08855284, 0.18878891, 0.88269905,
       0.03604255, 0.03538396, 0.27676368, 0.12346652, 0.03261637,
       0.02351471])
```

```
# Check accuracy score
from sklearn.metrics import accuracy_score

print('Model accuracy score: {0:0.4f}'.format(accuracy_score(y_test, y_pred_test)))
```

```
Model accuracy score: 0.9722
```

```
train_score = logreg.score(X_train, y_train)
test_score = logreg.score(X_test, y_test)
print(f'Train set score: {train_score}')
print(f'Test set score: {test_score}')
```

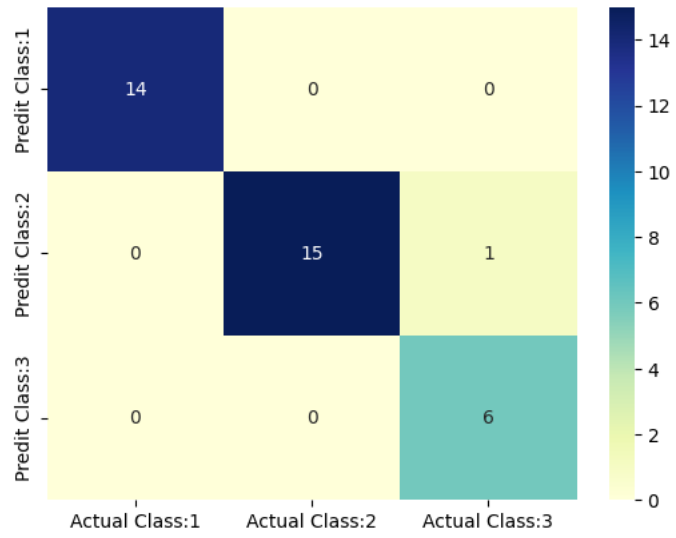
```
Train set score: 0.9788732394366197
Test set score: 0.9722222222222222
```

```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix

cm = confusion_matrix(y_test, y_pred_test)

print('Confusion matrix\n\n', cm)

print('\nTrue Positives(TP) = ', cm[0,0])

print('\nTrue Negatives(TN) =', cm[1,1])

print('\nFalse Positives(FP) =', cm[0,1])

print('\nFalse Negatives(FN) =', cm[1,0])
```

```
Confusion matrix

 [[14  0  0]
 [ 0 15  1]
 [ 0  0  6]]

True Positives(TP) =  14

True Negatives(TN) = 15

False Positives(FP) = 0

False Negatives(FN) = 0
```

```
cm_matrix = pd.DataFrame(data=cm, columns=['Actual Class:1', 'Actual Class:2', 'Actual Class:3'],
                         index=['Predit Class:1', 'Predit Class:2', 'Predit Class:3'])
sns.heatmap(cm_matrix, annot=True, fmt='d', cmap='YlGnBu')
```

```
<Axes: >
```




```
# classification metrices
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred_test))
```

```
                  precision    recall  f1-score   support

               1       1.00      1.00      1.00        14
               2       1.00      0.94      0.97        16
               3       0.86      1.00      0.92         6

        accuracy                           0.97        36
       macro avg       0.95      0.98      0.96        36
    weighted avg       0.98      0.97      0.97        36
```

```
y_pred_prob = logreg.predict_proba(X_test)[0:10]
y_pred_prob
```

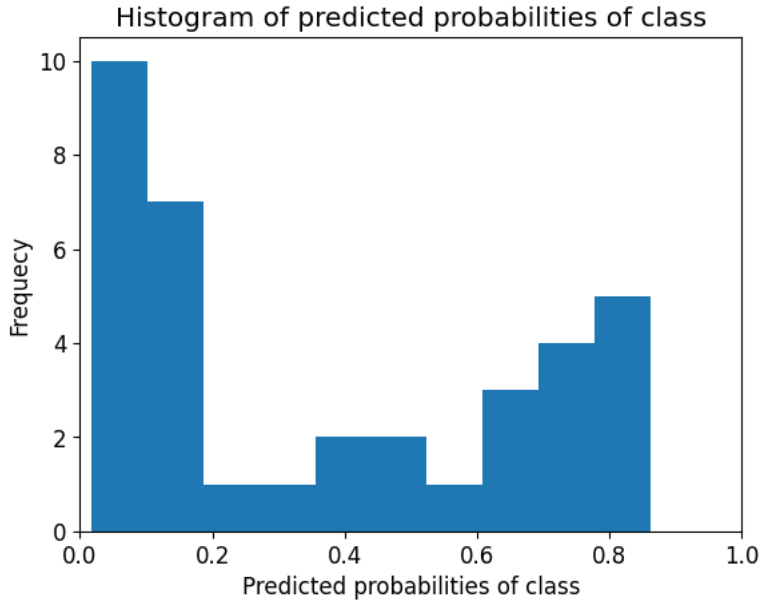
```
array([[0.84608661, 0.10773965, 0.04617374],
       [0.08048314, 0.04382495, 0.87569191],
       [0.33554459, 0.65481533, 0.00964009],
       [0.80152316, 0.1525236 , 0.04595323],
       [0.22039744, 0.66114073, 0.11846182],
       [0.24522358, 0.74433276, 0.01044366],
       [0.87468447, 0.07438344, 0.05093209],
       [0.03848718, 0.12116596, 0.84034686],
       [0.15948773, 0.78373262, 0.05677965],
       [0.05898538, 0.80051396, 0.14050066]])
```

```
# print the first 10 predicted probabilities
logreg.predict_proba(X_test)[0:10, 1]
```

```
array([0.10773965, 0.04382495, 0.65481533, 0.1525236 , 0.66114073,
       0.74433276, 0.07438344, 0.12116596, 0.78373262, 0.80051396])
```

```
# store the predicted probabilities
y_pred1 = logreg.predict_proba(X_test)[:,1]


# plot histogram of predicted probabilities
# adjust the font size
plt.rcParams['font.size'] = 12

# plot histogram with 10 bins
plt.hist(y_pred1, bins=10)

# set the title of predicted probabilities
plt.title('Histogram of predicted probabilities of class')

# set the x-axis limit
plt.xlim(0,1)

plt.xlabel('Predicted probabilities of class')
plt.ylabel('Frequecy')
```

```
Text(0, 0.5, 'Frequecy')
```



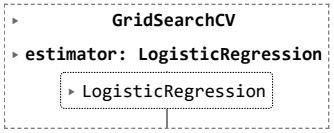

```
# k-Fold Cross Validation
from sklearn.model_selection import cross_val_score

scores = cross_val_score(logreg, X_train, y_train, cv = 5, scoring='accuracy')
print('Cross-validation scores:{}'.format(scores))
```

```
Cross-validation scores:[0.93103448 0.96551724 0.96428571 1.         0.96428571]
```

```
# compute Average cross-validation score
score_mean = scores.mean()
print(f'Average cross-validation score: {score_mean}')
```

```
Average cross-validation score: 0.9650246305418719
```

```
# Hyperparameter Optimization using GridSearchCV
from sklearn.model_selection import GridSearchCV

parameters = [{'penalty': ['l1', 'l2']},
              {'C':[1, 10, 100, 100]}]

grid_search = GridSearchCV(estimator = logreg,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)

grid_search.fit(X_train, y_train)
```




```
# examine the best model
# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))

# print parameters that give the best results
print('Parameters that give the best results :', '\n\n', (grid_search.best_params_))

# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :', '\n\n', (grid_search.best_estimator_))
```

```
GridSearch CV best score : 0.9650


Parameters that give the best results :

 {'C': 1}


Estimator that was chosen by the search :

 LogisticRegression(C=1, random_state=0, solver='liblinear')
```

```
# calculate Gridsearch CV score on test set
print('Gridsearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)))
```

```
Gridsearch CV score on test set: 0.9722
```