COMP4220: Machine Learning, Spring 2022, Assignment 3

Please submit one pdf file for all questions.

```
#importing the libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from google.colab import files

url =
'https://raw.githubusercontent.com/nnguyen09/Machine-Learning-/master/
assignment3/wine.csv'
df = pd.read_csv(url)
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]
```

variables (based on physicochemical tests):

Tips

An interesting thing to do is to set an arbitrary cutoff for your dependent variable (wine quality): 7 or higher getting classified as '1' and the remainder as '0'.

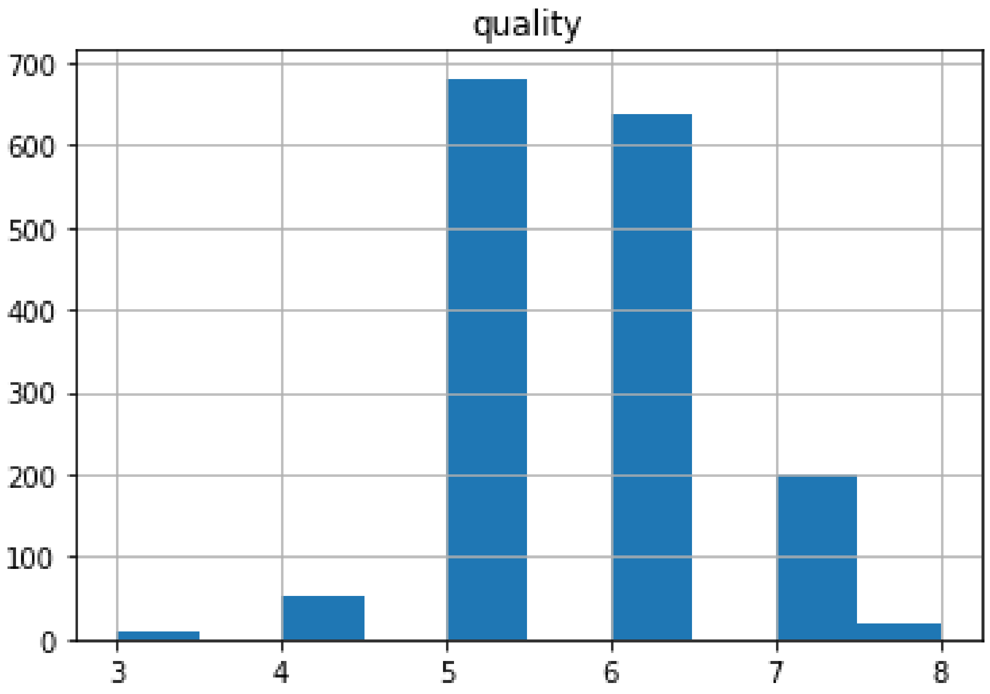
This allows you to convert this problem into a classification problem.

1. Since we want to classify the wine base on the quality so we want to look at the distribution of the wine quality

Make a histogram plot for the quality column to see the distribution of the wine quality

```
x='quality'
df.hist(column=x)
```

```
array([[<matplotlib.axes._subplots.AxesSubplot object at
0x7efddb9040d0>]],
      dtype=object)
```



2. Show the number of null values using sum() method. If there are null values then remove them from the dataset

```
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
```

3. Since we want to categorize the dependent variable (wine quality)

Change the quality column to 1 if the quality >= 7, and 0 if the quality is < 7

Show the dataset after making this change

Hint: the quality column should only have 0s and 1s after the change

```
df['quality'] = [1 if x >= 7 else 0 for x in df ['quality']]
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        0
1         9.8        0
2         9.8        0
3         9.8        0
4         9.4        0
...       ...      ...
1594     10.5        0
1595     11.2        0
1596     11.0        0
1597     10.2        0
1598     11.0        0

[1599 rows x 12 columns]
```

4. Create y as the quality column and X as everything but the quality column

```
# X=good quality, Y=bad quality
X = df.drop(['quality'], axis = 'columns')
y = df['quality']
df['quality'].value_counts()
```

```
0    1382
1     217
Name: quality, dtype: int64
```

5. Split the dataset into the training and test set using "train_test_split".

Split the training and test set into 70-30 ratio

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=.30, random_state=42)
```

6. Apply Feature Scaling method for X_train and X_test with "StandardScaler" from "sklearn.preprocessing"

Hint: use StandardScaler.fit_transform for "X_train" and use StandardScaler.transform for "X_test"

```
from sklearn.preprocessing import StandardScaler

scalar = StandardScaler()

X_train = scalar.fit_transform(X_train)
print(X_train)

X_test = scalar.transform(X_test)
```

```
[[ 1.69536131e-01 -1.72107140e+00  4.59303345e-01 ...  1.01180685e+00
   1.22661179e+00  5.50057013e-01]
 [ 2.44606730e+00 -4.01957443e-01  1.84105501e+00 ... -2.10687612e+00
   1.22661179e+00 -2.05174641e-01]
 [-6.47680186e-01  3.77472102e-02 -1.28054303e-03 ...  4.92026353e-01
   2.97270776e-01  5.50057013e-01]
 ...
 [-6.47680186e-01  4.77451864e-01 -1.07597628e+00 ...  1.27169710e+00
  -6.90154049e-01 -8.66002338e-01]
 [-2.39072027e-01 -1.83099757e+00  4.08127357e-01 ...  3.72184202e-02
   8.20025095e-01  1.39969262e+00]
 [-1.46489650e+00 -1.33632983e+00 -5.24565306e-02 ...  4.92026353e-01
  -6.90154049e-01  2.91015593e+00]]
```

7. Train the logistic regression model on the training set using (solver='lbfgs', random_state = 42, max_iter = 1000)

```
from sklearn.linear_model import LogisticRegression
log_reg = LogisticRegression(solver='lbfgs', random_state = 42,
max_iter = 1000)
log_reg.fit(X_train, y_train)
```

```
LogisticRegression(max_iter=1000, random_state=42)
```

8.Predict the results of x_test

```
y_test_pred = log_reg.predict(X_test)
print(y_test_pred)
```

```
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 1 0
0 0
 0 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
```

```
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 0
 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0
0 0
 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0
 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 1 0 0
0 0
 0 0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 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 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1
 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0
1 0
 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 0 0
0]
```

9.Make the confusion matrix and show the result

```
from sklearn.metrics import confusion_matrix

#Generate the confusion matrix
cf_matrix = confusion_matrix(y_test, y_test_pred)
print(cf_matrix)
```

```
[[395  18]
 [ 47  20]]
```

10. find the precision_score, recall_score, and f1_score and print them

```
from sklearn.metrics import precision_score, recall_score, f1_score

precision_score(y_test,y_test_pred)
```

```
0.5263157894736842
```

```
recall_score(y_test,y_test_pred)
```

```
0.29850746268656714
```

```
f1_score(y_test,y_test_pred)
```

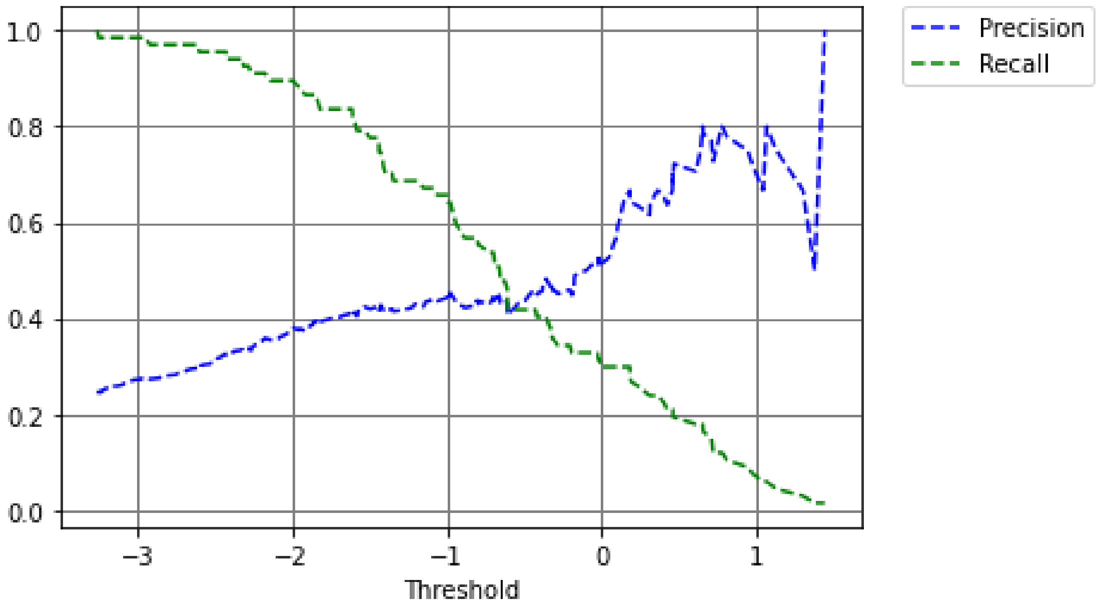
```
0.38095238095238093
```

11. Use the precision_recall_curve() function to compute precision and recall for all possible thresholds

```
from sklearn.metrics import precision_recall_curve
y_score = log_reg.decision_function(X_test)
precisions, recalls, thresholds = precision_recall_curve(y_test,
y_score)
```

12. Use Matplotlib to plot precision and recall as functions of the threshold value

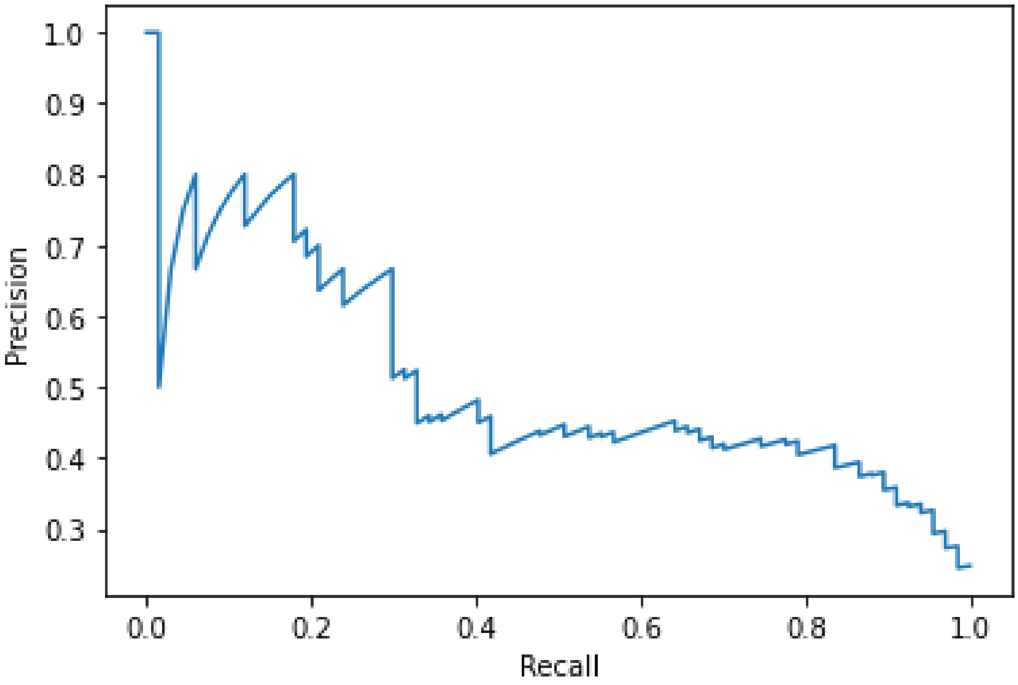
```
import matplotlib.pyplot as plt
def plot_precision_recall_vs_thresholds(precisions, recalls,
thresholds):
    plt.plot(thresholds, precisions[:-1], "b--", label="Precision")
    plt.plot(thresholds, recalls[:-1], "g--", label="Recall")
    plt.xlabel("Threshold")
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left',
borderaxespad=0.)
    plt.grid(b=True, which="both", axis="both", color='gray',
linestyle='-', linewidth=1)

plot_precision_recall_vs_thresholds(precisions, recalls, thresholds)
plt.show()
```



13. Plot the precision vs recall plot

```
fig, ax = plt.subplots()
ax.plot(recalls, precisions)
ax.set_ylabel('Precision')
ax.set_xlabel('Recall')
plt.show
```

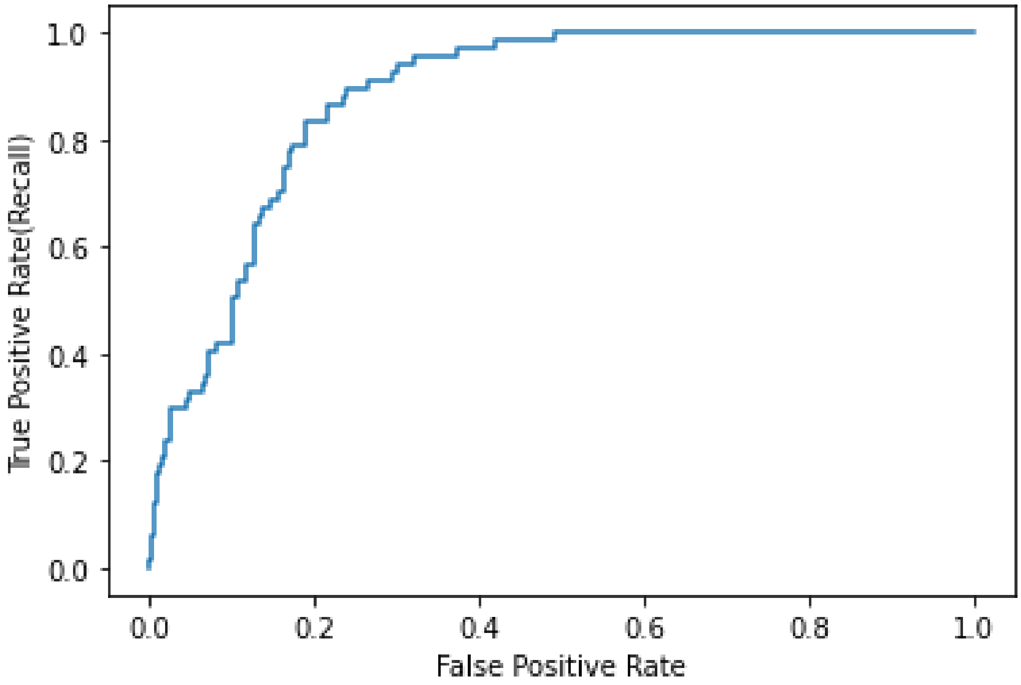
```
<function matplotlib.pyplot.show>
```



14. Plot the ROC Curve

```
from sklearn.metrics import roc_curve
pr, tpr, thresh = roc_curve(y_test, y_score)
fig, ax = plt.subplots()
ax.plot(pr, tpr)
ax.set_ylabel('True Positive Rate(Recall)')
ax.set_xlabel('False Positive Rate')
plt.show
```

```
<function matplotlib.pyplot.show>
```



15. Find the area under the ROC Curve

```
from sklearn.metrics import roc_auc_score
print (roc_auc_score(y_test, y_score))
```

```
0.8816450435473963
```