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
import sklearn.metrics as metrics
```

```
data = pd.read_csv("wine.csv")
data
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56
...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66

1599 rows × 12 columns

variables (based on physicochemical tests):

1. fixed acidity
2. volatile acidity
3. citric acid
4. residual sugar
5. chlorides

6. free sulfur dioxide
7. total sulfur dioxide
8. density
9. pH
10. sulphates
11. alcohol
12. quality (score between 0 and 10)

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

```
data.hist('quality',bins=25,figsize=(10,10))
# display histogram
plt.show()
```

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

```
data.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

```
data['quality'] = [1 if x >= 7 else 0 for x in data['quality']]
data
```

```
data['quality'].value_counts()
```

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

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

```
data.columns
```

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

```
data.index
```

```
RangeIndex(start=0, stop=1599, step=1)
```

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

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=0.3)
```

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

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
model = LogisticRegression(solver='lbfgs', random_state = 42, max_iter = 1000)
model.fit(X_train,y_train)
```

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

8.Predict the results of x_test

```
# predicts for positive outcomes (217)
y_pred_proba = model.predict_proba(X_test)[:,1]
y_pred_proba
```

```
            0.07892671, 0.04908873, 0.09817907, 0.013937  , 0.03510062,
            0.00430654, 0.01710288, 0.00974855, 0.01692177, 0.05159108,
            0.0813706 , 0.29687942, 0.01780104, 0.01272994, 0.5853521 ,
            0.09196407, 0.01879852, 0.00899429, 0.04148619, 0.09929425,
            0.01376242, 0.00731176, 0.04423716, 0.01230927, 0.01729944,
            0.28156655, 0.33662097, 0.06186459, 0.17362241, 0.13752229,
            0.07071015, 0.00690633, 0.01674352, 0.35790641, 0.01940677,
            0.5853521 , 0.04582706, 0.0262078 , 0.01271848, 0.15924794,
            0.53523074, 0.42428267, 0.54269506, 0.46085965, 0.38512386,
            0.08786908, 0.01089613, 0.15158026, 0.0855403 , 0.01829301,
            0.01659513, 0.02710096, 0.61907859, 0.04886267, 0.38087334,
            0.48445008, 0.39480443, 0.02031364, 0.09109343, 0.00998956,
            0.01480843, 0.11164663, 0.26680605, 0.00613664, 0.01086391,
            0.03158378, 0.02600405, 0.00758559, 0.03201015, 0.03929083,
            0.07794825, 0.58139302, 0.14026812, 0.06758283, 0.07203149,
            0.21401424, 0.00789193, 0.00650227, 0.03158378, 0.02092885,
            0.01012938, 0.04880079, 0.00573878, 0.04905121, 0.04492326,
            0.33039046, 0.1058772 , 0.32353275, 0.1872686 , 0.02079002,
            0.16896232, 0.15948484, 0.10193401, 0.01879073, 0.01351362,
            0.12129211, 0.28551073, 0.10339664, 0.01707809, 0.02552215,
             0.13866903, 0.00988957, 0.11260554, 0.01575313, 0.00928492,
             0.13148387, 0.28552878, 0.05922287, 0.07335933, 0.00599182,
             0.20446349, 0.07729755, 0.02461178, 0.00531663, 0.15466877,
             0.25919911, 0.53290377, 0.00796328, 0.13419491, 0.17167332,
             0.21293671, 0.24586467, 0.00492095, 0.00868027, 0.0511825 ,
             0.00602579, 0.01753709, 0.19877117, 0.58923069, 0.55783516,
             0.10235367, 0.04263696, 0.00222551, 0.03999841, 0.03480236,
             0.00737786, 0.01303564, 0.07653556, 0.02217   , 0.03122231,
             0.12674711, 0.07284613, 0.07168458, 0.17826297, 0.12192999,
             0.03292827, 0.69199685, 0.03054208, 0.06698294, 0.01725945,
             0.03910062, 0.02453481, 0.01201647, 0.08515306, 0.02654142,
             0.04148619, 0.01749013, 0.01999423, 0.02023251, 0.07511064,
             0.35507621, 0.09277066, 0.00671647, 0.37548757, 0.03617876,
             0.05633298, 0.05827273, 0.12064114, 0.01050624, 0.29850672,
             0.01268142, 0.01283795, 0.01594151, 0.0153197 , 0.2458678 ,
             0.04563866, 0.23432494, 0.00583547, 0.00593611, 0.75701668,
             0.00754137, 0.01602336, 0.00535125, 0.58923069, 0.03812344,
             0.0287668 , 0.20705711, 0.01249443, 0.32581441, 0.48565828,
             0.09146622, 0.02581082, 0.01602336, 0.05033314, 0.0023843 ,
             0.03525152, 0.05173   , 0.10156913, 0.10678326, 0.02173893,
             0.06747727, 0.04255414, 0.73948364, 0.51132838, 0.03284445,
             0.51606558, 0.01390867, 0.03114844, 0.01263723, 0.15033269,

             0.02152692, 0.2788616 , 0.00413103, 0.01902191, 0.00619046,
             0.00793077, 0.02985571, 0.02288476, 0.02049913, 0.40550653,
             0.01804627, 0.02440662, 0.03913495, 0.02336221, 0.56647462,
             0.07178716, 0.01594151, 0.16689822, 0.00676315, 0.61066342,
             0.02678397, 0.18876383, 0.02552032, 0.16075834, 0.37603785,
             0.00687201, 0.00169144, 0.00988957, 0.03177861, 0.11332208,
             0.03049786, 0.04805446, 0.03319618, 0.07889214, 0.09903398,
             0.1824793 , 0.1722001 , 0.01119773, 0.38143141, 0.01686866,
             0.00864324, 0.14729668, 0.01010078, 0.00703798, 0.02731319,
             0.03556735, 0.01647288, 0.02581082, 0.39959885, 0.56910311,
             0.7345827 , 0.18809664, 0.09882069, 0.68378809, 0.42722622,
             0.01986788, 0.11260554, 0.02941604, 0.08448427, 0.01148314,
             0.02152837, 0.27571579, 0.14057558, 0.53639275, 0.25002627,
             0.01410642, 0.02346209, 0.24431986, 0.42480786, 0.01779871,
             0.00698797, 0.46985917, 0.03910062, 0.00625101, 0.57720415,
             0.04336139, 0.02501697, 0.02856993, 0.36379256, 0.01806206,
```

```
# prediction for all data to make the confusion matrix
y_pred = model.predict(X_test)
y_pred
```

```
      array([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, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
             0, 0, 1, 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, 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, 1, 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, 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, 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,
             1, 0, 0, 0, 0, 1, 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, 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, 1, 0, 0, 0, 0, 0, 0,
             0, 0, 0, 0, 0, 1, 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, 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, 1, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1, 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, 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, 1, 0, 0, 1, 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, 1, 0, 0, 0, 0])
```

9.Make the confusion matrix and show the result

```
from sklearn.metrics import confusion_matrix
cm = metrics.confusion_matrix(y_test, y_pred)
cm
```

```
array([[404,  10],
       [ 44,  22]])
```

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

```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
print('precision: %.3f' % precision_score(y_test, y_pred))
print('Recall: %.3f' % recall_score(y_test, y_pred))
print('f1 score: %.3f' % f1_score(y_test, y_pred))
```

```
precision: 0.688
Recall: 0.333
f1 score: 0.449
```

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

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

```
#11 and 12 are done here
from sklearn.metrics import precision_recall_curve
# calculate pr-curve
precision, recall, thresholds = precision_recall_curve(y_test, y_pred_proba)
# plot the roc curve for the model
thresh = len(y_test[y_test==1]) / len(y_test)
```

13. Plot the precision vs recall plot

```
plt.plot([0,1], [thresh,thresh], linestyle='--', label='threshold')
plt.plot(recall, precision, marker='.')
# axis labels
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
# show the plot
plt.show()
```

14. Plot the ROC Curve

15. Find the area under the ROC Curve

```
# I did 14 and 15 in the same step
from sklearn.metrics import roc_auc_score
from sklearn.metrics import roc_curve
logit_roc_auc = roc_auc_score(y_test, y_pred)
precision, recall, thresholds = roc_curve(y_test, y_pred_proba)
plt.figure()
plt.plot(precision, recall, label='Logistic Regression (area = %0.2f)' % logit_roc_auc)
plt.plot([0, 1], [0, 1],'r--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('True positive rate')
plt.ylabel('False positive rate')
plt.title('ROC curve')
plt.legend(loc="lower right")
plt.show()
```