assignment-4

September 21, 2023

1 Grapes to Greatness: Machine Learning in Wine Quality

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
df=pd.read_csv('/content/winequality-red.csv')
df
```

[1]:

	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

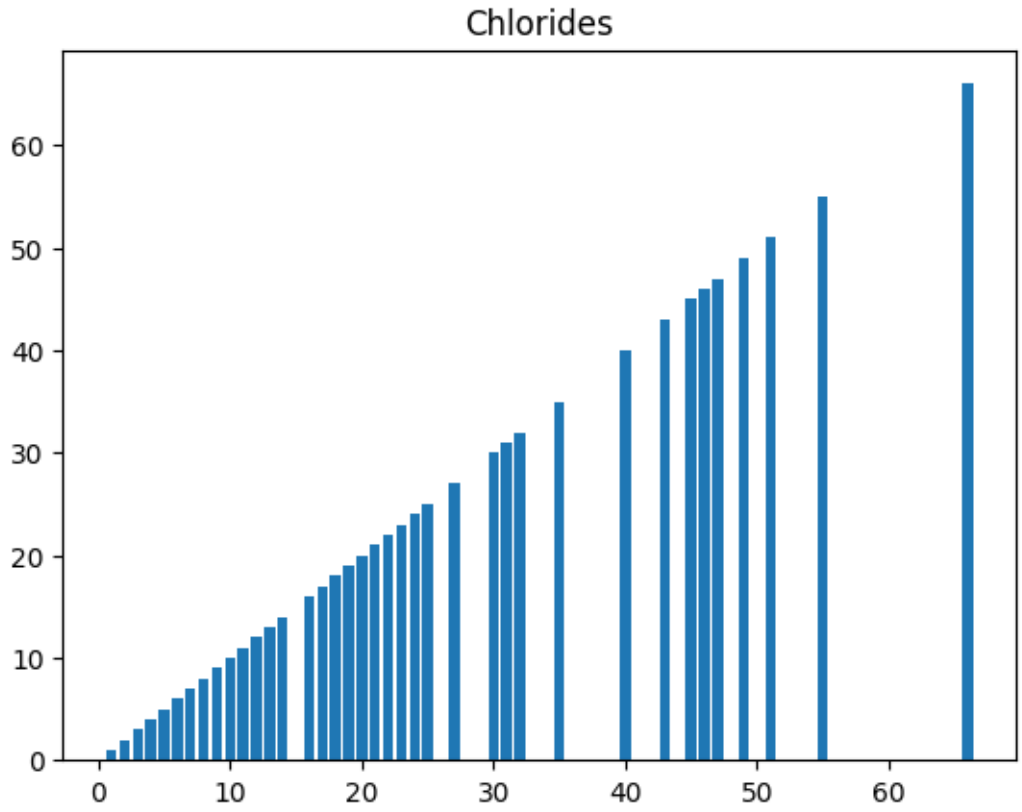
```
1          9.8        5
2          9.8        5
3          9.8        6
4          9.4        5
…          …        …
1594      10.5        5
1595      11.2        6
1596      11.0        6
1597      10.2        5
1598      11.0        6

[1599 rows x 12 columns]
```

[11]:
```
df.describe()
```

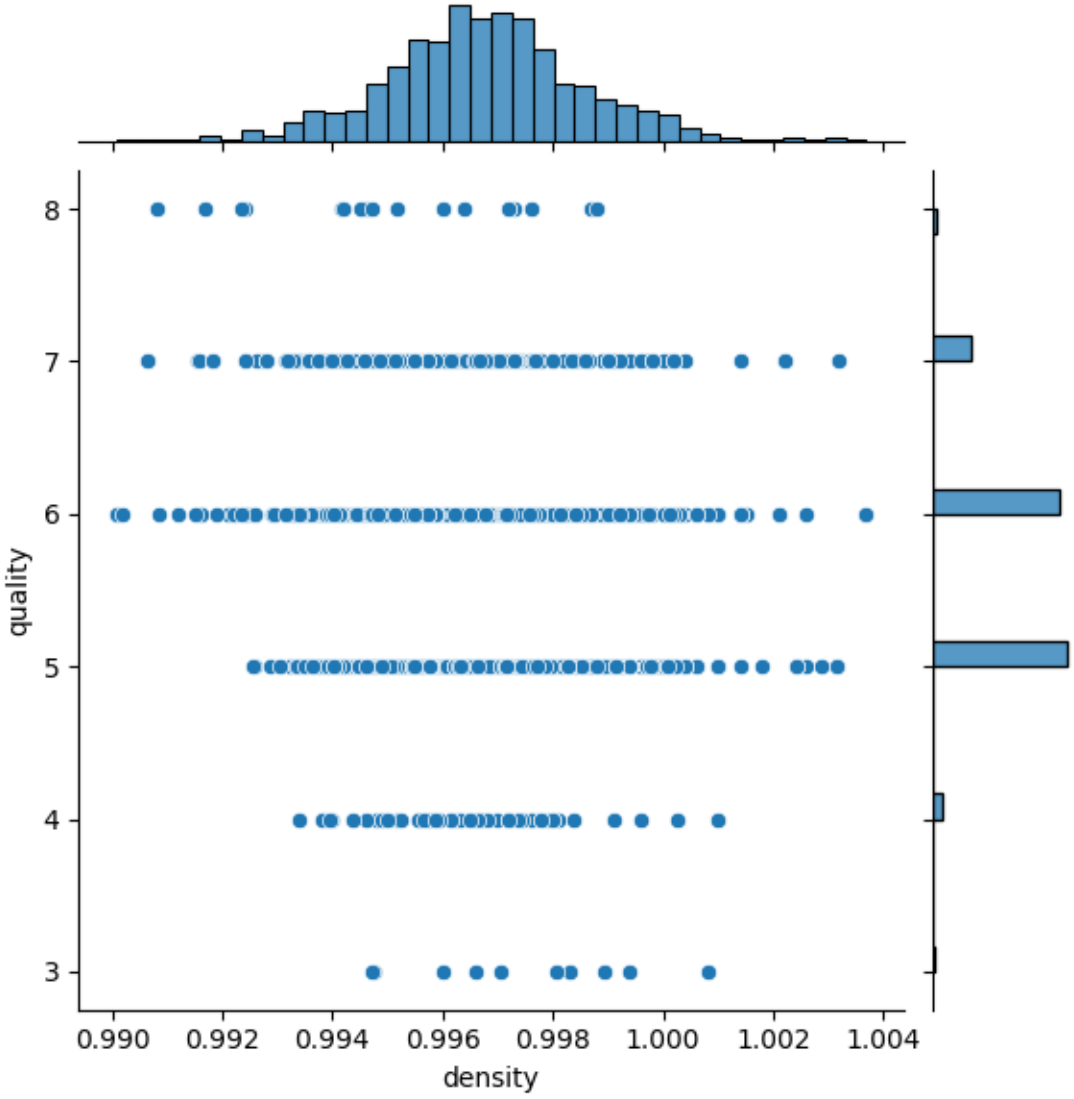
[11]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1599.000000       1599.000000  1599.000000     1599.000000
mean        8.319637          0.527821     0.270976        2.538806
std         1.741096          0.179060     0.194801        1.409928
min         4.600000          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
```

[35]:
```
plt.bar(df.chlorides.value_counts(),df.chlorides.value_counts())
plt.title("Chlorides")
```

```
plt.show()
```

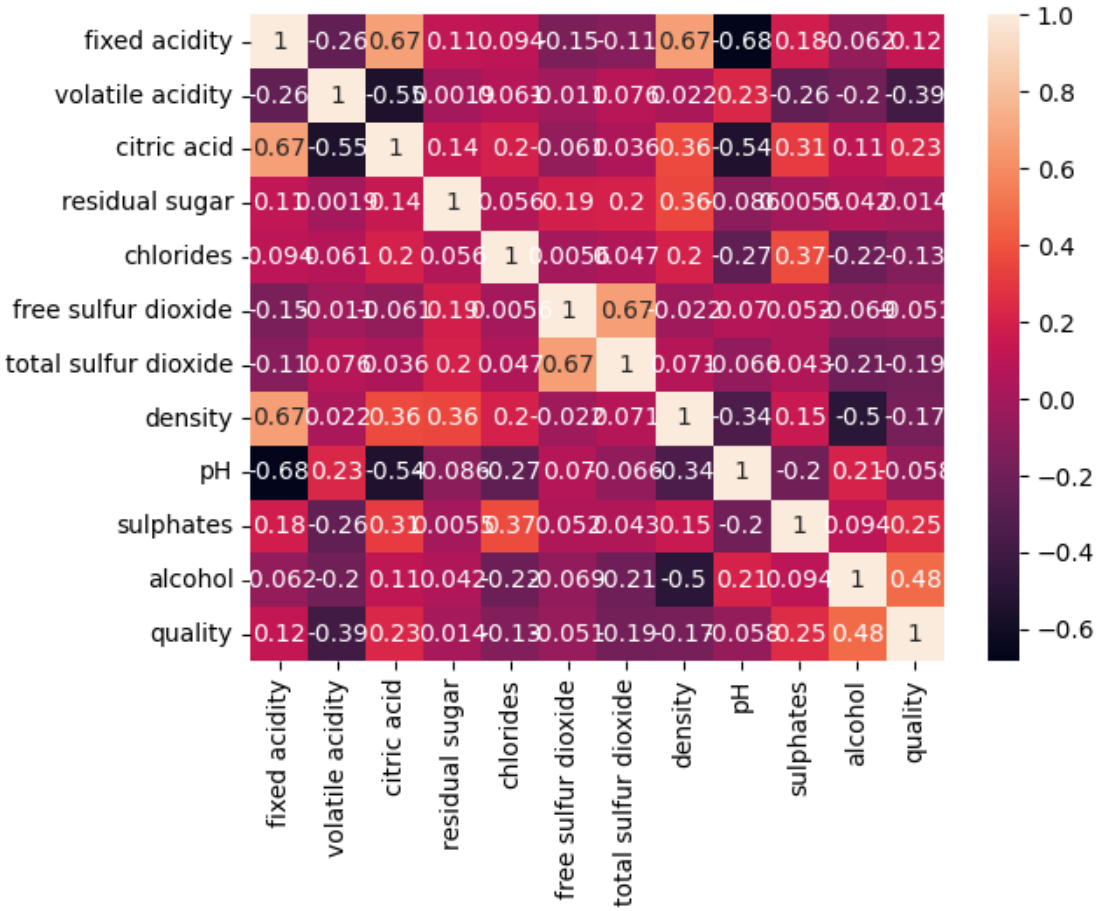


[7]:
```
sns.jointplot(x="density",y="quality",data=df)
plt.show()
```



[10]:
```
sns.heatmap(df.corr(),annot=True)
```

[10]: <Axes: >



[14]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
X=df.drop("quality",axis=1)
y=df["quality"]
X.head()
```

[14]:

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

```
4                11.0                  34.0   0.9978  3.51       0.56

   alcohol
0      9.4
1      9.8
2      9.8
3      9.8
4      9.4
```

[15]:
```
sc=StandardScaler()
X_scaled=sc.fit_transform(X)
X_train,X_test,y_train,y_test=train_test_split(X_scaled,y,test_size=0.
 ↪2,random_state=42)
```

[15]:
```
array([[ 0.21852997,  0.90601191,  0.20039205, …,  1.09426457,
         0.48302886,  1.10483337],
       [-1.27524919, -1.77549685,  0.66254621, …, -0.39596939,
        -0.40216729,  1.38643512],
       [ 1.48249695, -0.76993107,  1.02199944, …, -0.07200549,
         0.54204194, -0.58477711],
       …,
       [-0.6432657 ,  0.51495855, -1.08336951, …,  1.28864292,
        -0.69723268, -0.86637886],
       [-0.24109439, -1.83136161,  0.4057939 , …,  0.05758008,
         0.83710732,  1.38643512],
       [-1.44760832, -1.32857872, -0.05636026, …,  0.51112954,
        -0.69723268,  2.8883111 ]])
```

[19]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
le=LogisticRegression()
model=le.fit(X_train,y_train)
y_pred = model.predict(X_test)
cm = confusion_matrix(y_test, y_pred)
cm
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/linear_model/_logistic.py:458:
ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-
regression
  n_iter_i = _check_optimize_result(
```

[19]:
```
array([[ 0,  0,  1,  0,  0,  0],
       [ 0,  1,  7,  2,  0,  0],
       [ 0,  0, 98, 32,  0,  0],
       [ 0,  0, 46, 76, 10,  0],
       [ 0,  0,  3, 30,  9,  0],
       [ 0,  0,  0,  1,  4,  0]])
```

[21]:
```
from sklearn.metrics import accuracy_score,␣
↪confusion_matrix,classification_report,roc_auc_score,roc_curve
accuracy_score(y_test,y_pred)
```

[21]:
```
0.575
```

[22]:
```
pd.crosstab(y_test,y_pred)
```

[22]:

col_0	4	5	6	7
quality				
3	0	1	0	0
4	1	7	2	0
5	0	98	32	0
6	0	46	76	10
7	0	3	30	9
8	0	0	1	4

[25]:
```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         1
           4       1.00      0.10      0.18        10
           5       0.63      0.75      0.69       130
           6       0.54      0.58      0.56       132
           7       0.39      0.21      0.28        42
           8       0.00      0.00      0.00         5

    accuracy                           0.57       320
   macro avg       0.43      0.27      0.28       320
weighted avg       0.56      0.57      0.55       320
```

```
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
```

```
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1344:
UndefinedMetricWarning: Precision and F-score are ill-defined and being set to
0.0 in labels with no predicted samples. Use `zero_division` parameter to
control this behavior.
  _warn_prf(average, modifier, msg_start, len(result))
```

[27]:
```
probability  = model.predict_proba(X_test)[:,1]
probability
```

[27]:
```
array([0.02483477, 0.02249377, 0.01574946, 0.01134614, 0.03202448,
       0.01491135, 0.00750669, 0.11853224, 0.02170618, 0.04960947,
       0.01348905, 0.13794541, 0.02885185, 0.02524655, 0.02042773,
       0.00977959, 0.03329229, 0.01903721, 0.00330975, 0.03185402,
       0.1797074 , 0.03375518, 0.06405695, 0.01701967, 0.07199606,
       0.03828524, 0.00477106, 0.08112635, 0.04058424, 0.01038622,
       0.03132   , 0.04077697, 0.01130467, 0.04878004, 0.02851514,
       0.02944521, 0.01913737, 0.06835138, 0.03249786, 0.02057124,
       0.02567618, 0.02042431, 0.006988  , 0.01425204, 0.01374079,
       0.03947643, 0.00374969, 0.01676378, 0.08833596, 0.07250037,
       0.01094354, 0.02734811, 0.09025171, 0.01177792, 0.04743786,
       0.01557644, 0.02994908, 0.09684315, 0.02995632, 0.06248571,
       0.01361311, 0.02148332, 0.03028016, 0.04898658, 0.00497826,
       0.01510166, 0.01183327, 0.04781076, 0.00504487, 0.01450206,
       0.00975868, 0.08957871, 0.0110487 , 0.03338539, 0.02254037,
       0.04057309, 0.00302335, 0.00481278, 0.02595905, 0.00514559,
       0.06328622, 0.00571601, 0.02323339, 0.03167214, 0.02807292,
       0.00498152, 0.01519544, 0.071952  , 0.00589653, 0.10965556,
       0.00579155, 0.04130017, 0.09166548, 0.0216257 , 0.00971574,
       0.01430115, 0.02018015, 0.02031068, 0.15743163, 0.02627518,
       0.1286683 , 0.03879046, 0.01325419, 0.10345292, 0.04510503,
       0.01106421, 0.01385973, 0.00778097, 0.03758517, 0.03038102,
       0.00631245, 0.00592356, 0.00255582, 0.0085207 , 0.02894913,
       0.01094736, 0.06563413, 0.01373076, 0.05589608, 0.01637058,
       0.01182063, 0.0395007 , 0.01413268, 0.02768522, 0.0678964 ,
       0.04112568, 0.00826422, 0.02335786, 0.03401157, 0.02896179,
       0.0110487 , 0.11028868, 0.03168882, 0.00694686, 0.07250037,
       0.06947725, 0.03621314, 0.00512094, 0.01887089, 0.0232298 ,
       0.01022772, 0.00711342, 0.00791129, 0.05265072, 0.01430848,
       0.03762969, 0.05080835, 0.13812178, 0.00655973, 0.01417962,
       0.01731361, 0.02659349, 0.03403913, 0.01742294, 0.0110487 ,
       0.00525754, 0.05574201, 0.01837955, 0.02863305, 0.02180492,
       0.00855826, 0.01296316, 0.00719941, 0.02473188, 0.02877727,
       0.01831045, 0.11902339, 0.03350339, 0.00606012, 0.01003063,
       0.01231073, 0.07909799, 0.01248025, 0.00938227, 0.00922779,
       0.00984898, 0.00563226, 0.01627412, 0.01257727, 0.26411334,
```

```
0.09242519, 0.00722218, 0.00895868, 0.0108839 , 0.09369835,
0.00256229, 0.14726728, 0.013673  , 0.00316726, 0.00938227,
0.05751076, 0.00648566, 0.01672627, 0.00407532, 0.01105994,
0.04465489, 0.13261131, 0.01265878, 0.02343221, 0.01106421,
0.04312645, 0.00825491, 0.02558483, 0.04303992, 0.00167707,
0.06365584, 0.02879044, 0.02108765, 0.02017862, 0.00209676,
0.04510175, 0.00560467, 0.01235811, 0.00154654, 0.01053313,
0.03851546, 0.07504775, 0.01853088, 0.04535837, 0.03899463,
0.0425178 , 0.00846753, 0.0104536 , 0.00717051, 0.23341551,
0.04884486, 0.04044068, 0.00566036, 0.00527932, 0.10618048,
0.00553059, 0.03591392, 0.01170368, 0.03609803, 0.00592857,
0.01106421, 0.06540582, 0.02807292, 0.06035659, 0.0120259 ,
0.04567327, 0.04198665, 0.06234175, 0.0061261 , 0.03219853,
0.02733513, 0.02068696, 0.04479066, 0.00227449, 0.00794547,
0.00523833, 0.0414681 , 0.04079487, 0.02742506, 0.35729187,
0.02987603, 0.02290243, 0.03998963, 0.01302718, 0.00323973,
0.01024649, 0.08956821, 0.01627412, 0.00917579, 0.04400462,
0.00749356, 0.03743595, 0.01220615, 0.00907912, 0.00546852,
0.0184154 , 0.00756899, 0.01566495, 0.01252444, 0.02138715,
0.04502192, 0.01317294, 0.00726284, 0.00533177, 0.05413836,
0.00522663, 0.12843439, 0.0386677 , 0.02689816, 0.01919488,
0.02910286, 0.03470307, 0.00711342, 0.03192086, 0.21076717,
0.0268413 , 0.01016235, 0.00992186, 0.01227231, 0.01608297,
0.00649988, 0.04338943, 0.00832624, 0.0088752 , 0.00587206,
0.00899913, 0.02178773, 0.00842123, 0.01201218, 0.04519739,
0.05589983, 0.01363781, 0.0050594 , 0.03334108, 0.00328764,
0.03285411, 0.03042666, 0.02367536, 0.04470886, 0.04878004,
0.0542435 , 0.02056979, 0.04453912, 0.00576608, 0.05107761])
```

[34]:
```
model.predict([[7.4, 0.700, 0.00, 1.9, 0.076, 11.0, 34.0, 0.99780, 3.51, 0.56,␣
↪9.4]])
```

[34]:
```
array([5])
```