DSC 680 Project 3 Red Wine Python

May 10, 2020

[16]:
```python
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
# from sklearn import preprocessing
# from sklearn.ensemble import RandomForestRegressor
# from sklearn.pipeline import make_pipeline
# from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.linear_model import LinearRegression
from sklearn import metrics
import seaborn as sns
import matplotlib.pyplot as plt
```

[2]:
```python
# Read the data
red = pd.read_csv('winequality-red.csv', sep = ';')
red.head()
```

[2]:
```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.4              0.70         0.00             1.9      0.076
1            7.8              0.88         0.00             2.6      0.098
2            7.8              0.76         0.04             2.3      0.092
3           11.2              0.28         0.56             1.9      0.075
4            7.4              0.70         0.00             1.9      0.076

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 11.0                  34.0   0.9978  3.51       0.56
1                 25.0                  67.0   0.9968  3.20       0.68
2                 15.0                  54.0   0.9970  3.26       0.65
3                 17.0                  60.0   0.9980  3.16       0.58
4                 11.0                  34.0   0.9978  3.51       0.56

   alcohol  quality
0      9.4        5
1      9.8        5
2      9.8        5
3      9.8        6
4      9.4        5
```

[3]:
```
print(red.shape)
```

```
(1599, 12)
```

[4]:
```
red.describe()
```

[4]:
```
       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
```

[5]:
```
# Search for missing values
print(red.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
```

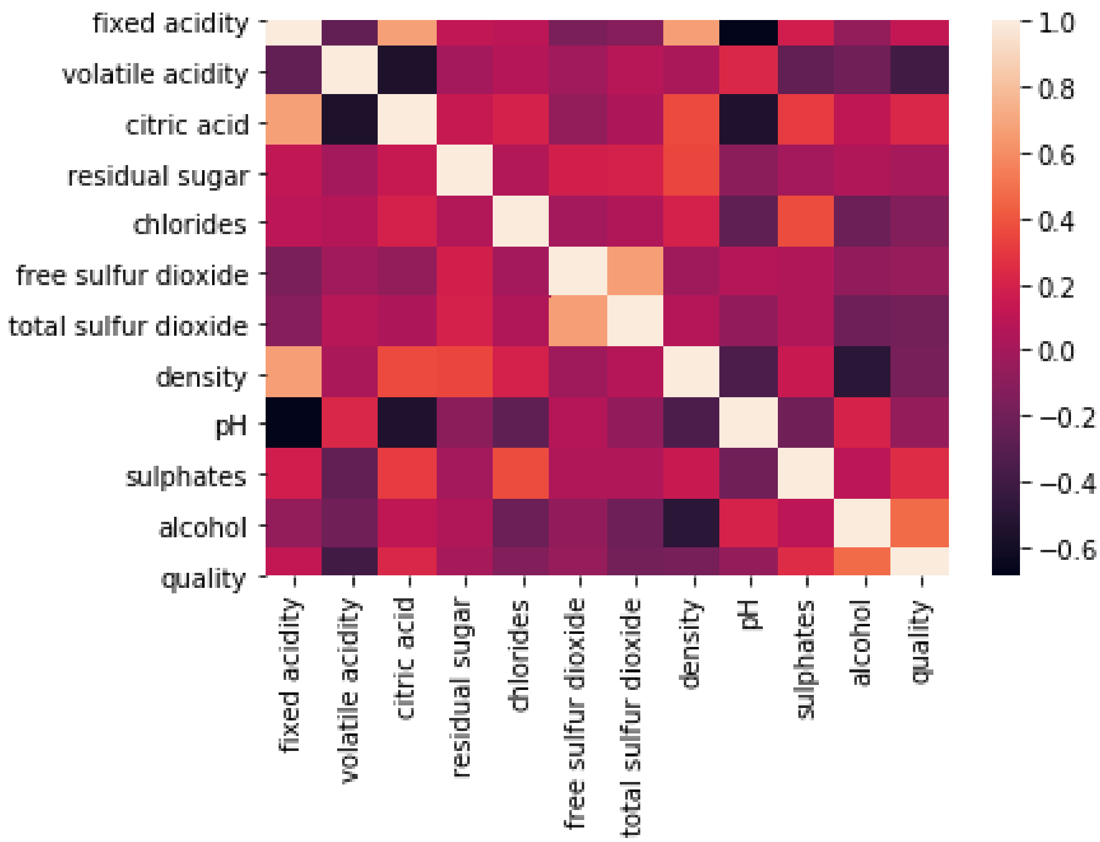
[6]:
```
# Get column names
col_red_names = red.columns
print(col_red_names)
# Get column data types
print(red.dtypes)
```

```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
fixed acidity           float64
volatile acidity        float64
citric acid             float64
residual sugar          float64
chlorides               float64
free sulfur dioxide     float64
total sulfur dioxide    float64
density                 float64
pH                      float64
sulphates               float64
alcohol                 float64
quality                   int64
dtype: object
```

[7]:
```
correlations = red.corr()['quality'].drop('quality')
print(correlations)
```

```
fixed acidity           0.124052
volatile acidity       -0.390558
citric acid             0.226373
residual sugar          0.013732
chlorides              -0.128907
free sulfur dioxide    -0.050656
total sulfur dioxide   -0.185100
density                -0.174919
pH                     -0.057731
sulphates               0.251397
alcohol                 0.476166
Name: quality, dtype: float64
```

[8]:
```
sns.heatmap(red.corr())
plt.show()
```



[11]:
```
# Separate target from training features

y = red.quality
x = red.drop('quality', axis=1)
```

[12]:
```
# Split data into train and test sets
x_train,x_test,y_train,y_test=train_test_split(x,y,random_state=3)
```

[13]:
```
# fitting linear regression to training data
regressor = LinearRegression()
regressor.fit(x_train,y_train)

# this gives the coefficients of the 10 features selected above.
print(regressor.coef_)
```

```
[ 2.32736478e-02 -9.91535569e-01 -1.41267594e-01  8.11925585e-03
 -1.59192407e+00  5.50005690e-03 -3.54198549e-03 -6.06916616e+00
 -4.06325022e-01  8.23603060e-01  2.94180891e-01]
```

[14]:
```python
# To predict the quality of wine with this model

train_pred = regressor.predict(x_train)
print(train_pred)
test_pred = regressor.predict(x_test)
print(test_pred)
```

```
[5.33390209 5.33458216 5.94987004 … 6.39109929 6.20184044 5.27719203]
[5.09908272 5.65580865 5.90927233 6.13810421 5.00495043 5.44066916
 5.05213654 6.15418124 5.52055599 5.77519663 5.61796132 5.23498287
 5.23127869 5.31466808 6.46439345 5.04000017 5.85280918 5.19300859
 6.0919118  6.34255254 6.41600994 5.52588684 5.80534686 4.93255733
 5.16159004 5.48207651 5.13834113 6.59480979 5.89478275 5.73709
 6.09133736 6.29529369 4.91616391 5.88376873 5.10515437 5.96400538
 6.80732578 5.03724291 5.25485064 5.88376873 5.17431803 4.84899008
 6.4903037  5.40465942 5.30375415 5.83513199 5.70825368 5.23988973
 5.24870634 5.46267267 5.08516492 5.61701512 6.01804854 6.32751521
 5.4628648  5.36127481 5.10151339 4.92009423 5.2240759  5.08722001
 4.79258875 5.43567381 5.25054561 5.6798788  5.85050157 6.52603804
 5.37941315 5.71598525 5.16966353 5.98159839 5.63912543 5.6004759
 5.74068429 5.22739422 5.98184324 5.51332746 5.40647057 5.68342011
 5.64578506 5.73709    6.23278066 5.29710528 4.66398697 6.0425789
 5.53767287 5.17796008 5.21203744 5.95953904 5.51273214 5.64429718
 5.70470381 5.64311292 5.72484629 5.31747436 5.37088603 5.40115158
 4.81676854 5.44991141 5.47380406 6.536759   6.14384914 5.63963684
 6.0764213  6.18184545 5.7333969  4.93007305 4.7323061  5.04093649
 5.45014754 5.78173092 6.44429962 5.47723449 6.46843631 5.94017696
 5.43087432 5.2047468  5.3484345  5.20416986 6.19702214 5.62516963
 5.83959227 5.19657408 5.17215817 5.26956927 5.74449907 5.64311292
 6.14688619 5.89603605 5.49258006 5.40446964 5.25143958 5.40731851
 5.70426783 5.67629516 6.58239965 5.88935979 6.38222076 5.73042293
 5.37034033 5.14004751 5.58608223 6.59675055 5.24400122 5.25667208
 5.54847605 5.16833418 5.76174468 6.10180871 6.93344996 4.99041658
 5.02045525 4.6936361  5.83279744 5.0202852  5.22739422 5.70787157
 5.629781   5.33023534 5.22441656 5.84507489 5.6222602  5.78080916
 5.52275613 6.07368903 5.63220544 5.49065808 5.96678121 4.82131562
 5.25184801 5.67298643 5.7315135  6.61949155 5.0208738  5.90323477
 5.85139093 5.20850319 5.68647856 5.5107533  5.40465942 6.37989991
 6.72172128 4.98279688 5.87260366 5.75498799 5.74800561 5.61884195
 5.711974   5.42385235 6.05973318 5.58256745 5.8850805  6.52201233
 5.00475912 5.39601068 5.1816475  5.16966353 5.4628648  5.75133323
 5.69516241 4.92576461 5.12771702 4.98053813 6.18910436 5.66320044
 5.44465486 5.56328176 4.99497389 5.83781624 5.31649795 5.48545825
 5.69404608 5.6982245  5.70724638 5.82340331 5.79368731 6.02336481
 6.20271572 5.27399606 5.04145317 5.21266809 5.38322644 4.97897504
 6.20624438 5.44065494 5.94166869 5.2181219  6.61288079 5.08900422
 5.29347654 5.03590649 6.17278277 5.77436736 4.8608174  5.72198789
```

```
 5.29496071 5.35533595 5.17179335 6.29984401 5.59779315 4.9609217
 6.10214544 6.03200835 6.17804629 5.42115437 6.76301676 6.21226379
 6.0861762  5.22659704 5.44997449 5.54506957 5.35533595 5.2412872
 5.74150467 5.25027186 6.12552958 5.42771411 5.83939322 4.81992631
 6.06232111 5.08298843 6.43084951 6.06506094 5.70218957 5.70426783
 4.90078952 5.99868147 5.28886652 5.70122979 5.42595998 5.11848917
 6.48395933 5.30489074 5.96848836 5.67858152 6.4906849  6.19855569
 5.09593756 5.46687509 5.30216736 5.23795665 6.38222076 5.37816899
 5.40613284 5.99854543 4.99943884 5.88539114 5.3527035  5.18049126
 5.48464387 5.91747035 4.82427368 6.84444638 5.17024164 4.93584868
 5.71503791 5.67672479 5.29391086 6.28837157 6.88282676 6.58112403
 5.94280234 6.33341072 5.90237267 5.56893141 6.00359575 5.51815475
 5.54851918 5.7333969  5.31204514 5.1501443  6.2167232  5.30100053
 6.21938621 6.10351244 5.87211737 5.4294743  4.84344446 6.0468106
 5.24475561 5.11875658 6.4639068  5.49851347 6.74724534 5.10901438
 5.1652877  5.30672745 5.71398551 5.10794561 5.53117148 5.31620746
 5.16515101 4.97232263 5.45290614 5.38377742 5.3484098  5.17608733
 6.11753324 5.61279425 5.73151521 6.34758037 5.90725513 5.5107533
 5.69333789 5.14471327 5.72896485 6.40189738 6.15898319 5.46033547
 6.05278052 5.73135561 6.23901028 5.23795665 5.54774909 5.04649062
 4.9961328  5.07773137 5.80157819 5.39684518 6.01142215 5.14597383
 6.51552207 5.3931712  5.49159322 5.64862532 5.52478877 5.26990054
 6.4906849  5.82378383 5.03571259 5.24297506 5.49335    5.28186183
 5.50973691 5.04135168 5.29294649 5.45465132 5.36525182 6.12969147
 4.99593587 5.30216736 6.07959665 5.2047468  5.13404621 4.65965123
 6.15655748 6.15713628 6.50577148 5.79958773 5.72198789 6.39773211
 6.13733354 5.89478275 6.05255784 6.03220643 5.37428389 5.40499404
 5.6190012  5.40207228 5.75989893 5.25849574]
```

[17]:
```python
# calculating rmse
train_rmse = mean_squared_error(train_pred, y_train) ** 0.5
print(train_rmse)
test_rmse = mean_squared_error(test_pred, y_test) ** 0.5
print(test_rmse)
```

```
0.6524682504422629
0.6269476348621655
```

[18]:
```python
# rounding off the predicted values for test set
predicted_data = np.round_(test_pred)
print(predicted_data)
print('Mean Absolute Error:', metrics.mean_absolute_error(y_test, test_pred))
print('Mean Squared Error:', metrics.mean_squared_error(y_test, test_pred))
print('Root Mean Squared Error:', np.sqrt(metrics.mean_squared_error(y_test,␣
↪test_pred)))
```

```
[5. 6. 6. 6. 5. 5. 5. 6. 6. 6. 6. 5. 5. 5. 6. 5. 6. 5. 6. 6. 6. 6. 6. 5.
 5. 5. 5. 7. 6. 6. 6. 6. 5. 6. 5. 6. 7. 5. 5. 6. 5. 5. 6. 5. 5. 6. 6. 5.
```

```
 5. 5. 5. 6. 6. 6. 5. 5. 5. 5. 5. 5. 5. 5. 5. 6. 6. 7. 5. 6. 5. 6. 6. 6.
 6. 5. 6. 6. 5. 6. 6. 6. 6. 5. 5. 6. 6. 5. 5. 6. 6. 6. 6. 6. 6. 5. 5. 5.
 5. 5. 5. 7. 6. 6. 6. 6. 6. 5. 5. 5. 5. 6. 6. 5. 6. 6. 5. 5. 5. 5. 6. 6.
 6. 5. 5. 5. 6. 6. 6. 6. 5. 5. 5. 5. 6. 6. 7. 6. 6. 6. 5. 5. 6. 7. 5. 5.
 6. 5. 6. 6. 7. 5. 5. 5. 6. 5. 5. 6. 6. 5. 5. 6. 6. 6. 6. 6. 6. 5. 6. 5.
 5. 6. 6. 7. 5. 6. 6. 5. 6. 6. 5. 6. 7. 5. 6. 6. 6. 6. 6. 5. 6. 6. 6. 7.
 5. 5. 5. 5. 5. 6. 6. 5. 5. 5. 6. 6. 5. 6. 5. 6. 5. 5. 6. 6. 6. 6. 6. 6.
 6. 5. 5. 5. 5. 5. 6. 5. 6. 5. 7. 5. 5. 5. 6. 6. 5. 6. 5. 5. 5. 6. 6. 5.
 6. 6. 6. 5. 7. 6. 6. 5. 5. 6. 5. 5. 6. 5. 6. 5. 6. 5. 6. 5. 6. 6. 6. 6.
 5. 6. 5. 6. 5. 5. 6. 5. 6. 6. 6. 6. 5. 5. 5. 5. 6. 5. 5. 6. 5. 6. 5. 5.
 5. 6. 5. 7. 5. 5. 6. 6. 5. 6. 7. 7. 6. 6. 6. 6. 6. 6. 6. 6. 5. 5. 6. 5.
 6. 6. 6. 5. 5. 6. 5. 5. 6. 5. 7. 5. 5. 5. 6. 5. 6. 5. 5. 5. 5. 5. 5. 5.
 6. 6. 6. 6. 6. 6. 6. 5. 6. 6. 6. 5. 6. 6. 6. 5. 6. 5. 5. 5. 6. 5. 6. 5.
 7. 5. 5. 6. 6. 5. 6. 6. 5. 5. 5. 5. 6. 5. 5. 5. 5. 6. 5. 5. 6. 5. 5. 5.
 6. 6. 7. 6. 6. 6. 6. 6. 6. 6. 5. 5. 6. 5. 6. 5.]
Mean Absolute Error: 0.4836851598205273
Mean Squared Error: 0.3930633368592633
Root Mean Squared Error: 0.6269476348621655
```

[19]:
```python
# displaying coefficients of each feature
coeffecients = pd.DataFrame(regressor.coef_,x.columns)
coeffecients.columns = ['Coeffecient']
print(coeffecients)
```

```
                      Coeffecient
fixed acidity            0.023274
volatile acidity        -0.991536
citric acid             -0.141268
residual sugar           0.008119
chlorides               -1.591924
free sulfur dioxide      0.005500
total sulfur dioxide    -0.003542
density                 -6.069166
pH                      -0.406325
sulphates                0.823603
alcohol                  0.294181
```

[ ]: