wine-quality-analysis-using-regression

September 26, 2023

[1]:
```
# 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

import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)

# Input data files are available in the read-only "../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))

# You can write up to 20GB to the current directory (/kaggle/working/) that gets␣
 ↪preserved as output when you create a version using "Save & Run All"
# You can also write temporary files to /kaggle/temp/, but they won't be saved␣
 ↪outside of the current session
```

[2]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

[3]:
```
df = pd.read_csv('WineQT.csv')
df
```

[3]:
```
      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
...             ...               ...          ...             ...        ...
1138            6.3             0.510         0.13             2.3      0.076
1139            6.8             0.620         0.08             1.9      0.068
```

```
1140           6.2           0.600      0.08           2.0      0.090
1141           5.9           0.550      0.10           2.2      0.062
1142           5.9           0.645      0.12           2.0      0.075

      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
...                   ...                   ...      ...   ...        ...
1138                 29.0                  40.0  0.99574  3.42       0.75
1139                 28.0                  38.0  0.99651  3.42       0.82
1140                 32.0                  44.0  0.99490  3.45       0.58
1141                 39.0                  51.0  0.99512  3.52       0.76
1142                 32.0                  44.0  0.99547  3.57       0.71

      alcohol  quality    Id
0         9.4        5     0
1         9.8        5     1
2         9.8        5     2
3         9.8        6     3
4         9.4        5     4
...       ...      ...   ...
1138     11.0        6  1592
1139      9.5        6  1593
1140     10.5        5  1594
1141     11.2        6  1595
1142     10.2        5  1597

[1143 rows x 13 columns]
```

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

[4]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    1143.000000       1143.000000  1143.000000     1143.000000
mean        8.311111          0.531339     0.268364        2.532152
std         1.747595          0.179633     0.196686        1.355917
min         4.600000          0.120000     0.000000        0.900000
25%         7.100000          0.392500     0.090000        1.900000
50%         7.900000          0.520000     0.250000        2.200000
75%         9.100000          0.640000     0.420000        2.600000
max        15.900000          1.580000     1.000000       15.500000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  1143.000000          1143.000000           1143.000000  1143.000000
mean      0.086933            15.615486             45.914698     0.996730
```

```
std        0.047267            10.250486            32.782130     0.001925
min        0.012000             1.000000             6.000000     0.990070
25%        0.070000             7.000000            21.000000     0.995570
50%        0.079000            13.000000            37.000000     0.996680
75%        0.090000            21.000000            61.000000     0.997845
max        0.611000            68.000000           289.000000     1.003690

                pH    sulphates      alcohol      quality           Id
count  1143.000000  1143.000000  1143.000000  1143.000000  1143.000000
mean      3.311015     0.657708    10.442111     5.657043   804.969379
std       0.156664     0.170399     1.082196     0.805824   463.997116
min       2.740000     0.330000     8.400000     3.000000     0.000000
25%       3.205000     0.550000     9.500000     5.000000   411.000000
50%       3.310000     0.620000    10.200000     6.000000   794.000000
75%       3.400000     0.730000    11.100000     6.000000  1209.500000
max       4.010000     2.000000    14.900000     8.000000  1597.000000
```

[5]:
```python
df.info()
```

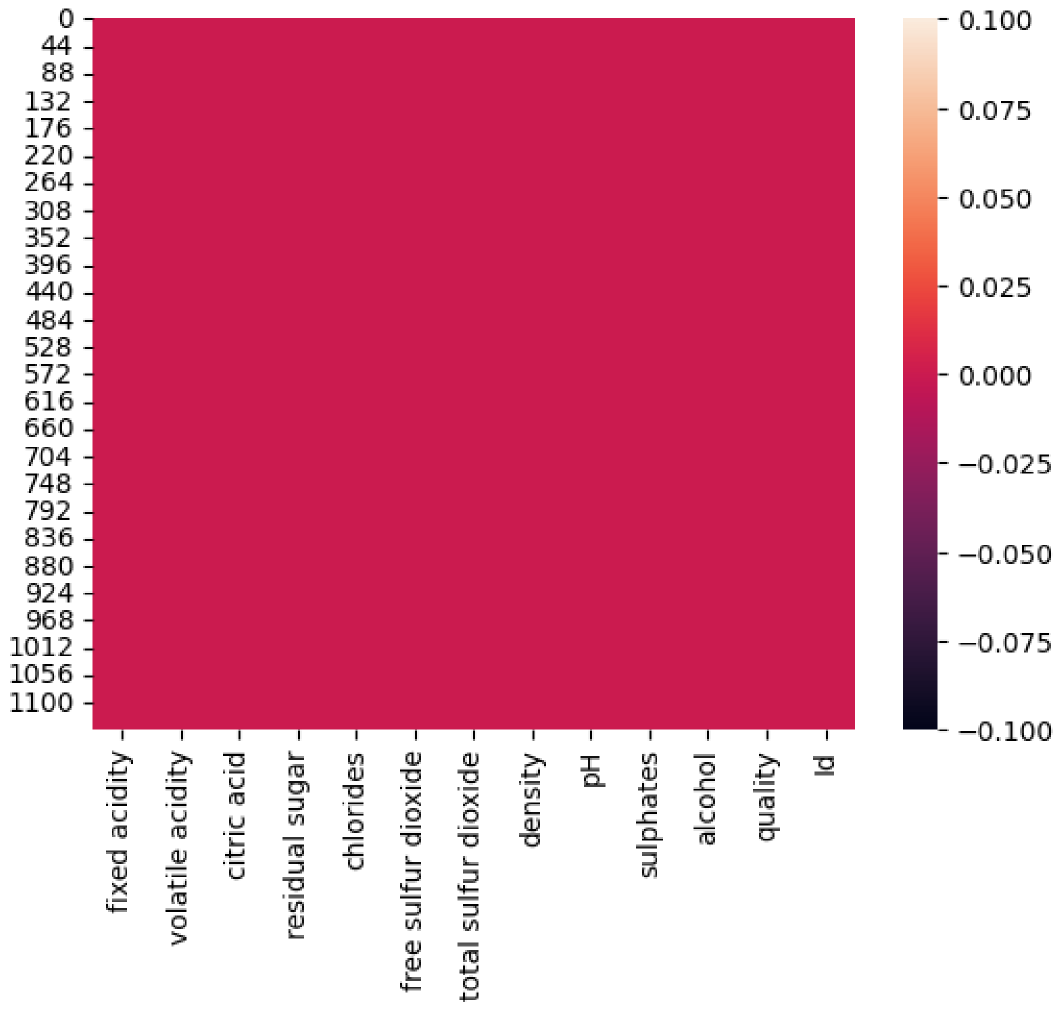
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1143 entries, 0 to 1142
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1143 non-null   float64
 1   volatile acidity      1143 non-null   float64
 2   citric acid           1143 non-null   float64
 3   residual sugar        1143 non-null   float64
 4   chlorides             1143 non-null   float64
 5   free sulfur dioxide   1143 non-null   float64
 6   total sulfur dioxide  1143 non-null   float64
 7   density               1143 non-null   float64
 8   pH                    1143 non-null   float64
 9   sulphates             1143 non-null   float64
 10  alcohol               1143 non-null   float64
 11  quality               1143 non-null   int64
 12  Id                    1143 non-null   int64
dtypes: float64(11), int64(2)
memory usage: 116.2 KB
```

0.0.1 no null values

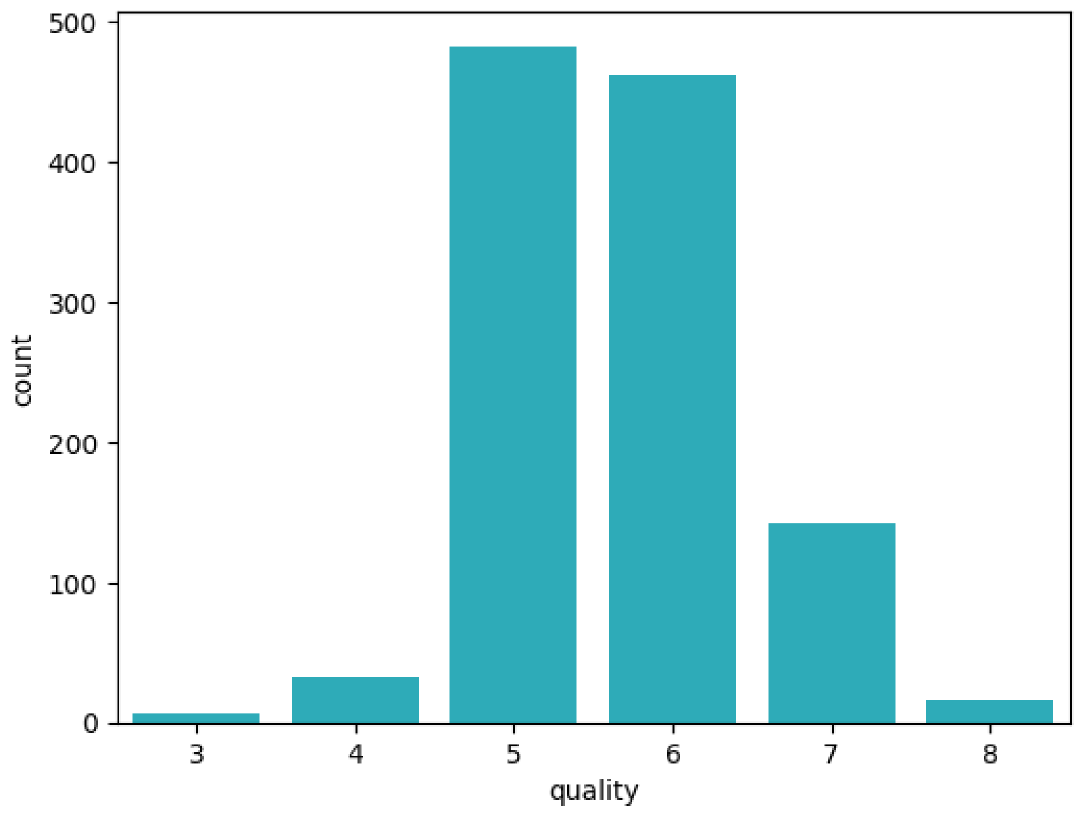
0.0.2 visulize for null values

[6]:
```python
import seaborn as sns
sns.heatmap(df.isnull())
```

[6]:
```
<AxesSubplot: >
```



[7]:
```
# vusulize count of quality
sns.countplot(data = df, x = df['quality'], color= sns.color_palette()[9])
plt.show()
```



[8]:
```
#input data
x = df.iloc[:,:11]
x.head()
```

[8]:
```
   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
0      9.4
1      9.8
```

```
2      9.8
3      9.8
4      9.4
```

[9]:
```
#output data
y = df.iloc[:,11:12]
y.head()
```

[9]:
```
   quality
0        5
1        5
2        5
3        6
4        5
```

0.1 split data into train and test

[10]:
```
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= .
 ↪25,random_state=12)
```

[11]:
```
x_train.head()
```

[11]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
1038            7.9             0.200         0.35             1.7      0.054
638             7.4             0.635         0.10             2.4      0.080
765             8.5             0.340         0.40             4.7      0.055
81              9.4             0.400         0.31             2.2      0.090
489             9.2             0.920         0.24             2.6      0.087

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
1038                  7.0                  15.0  0.99458  3.32       0.80
638                  16.0                  33.0  0.99736  3.58       0.69
765                   3.0                   9.0  0.99738  3.38       0.66
81                   13.0                  62.0  0.99660  3.07       0.63
489                  12.0                  93.0  0.99980  3.48       0.54

      alcohol
1038     11.9
638      10.8
765      11.6
81       10.5
489       9.8
```

0.2 scale the data

[12]:
```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
x_train = sc.fit_transform(x_train)
x_test = sc.fit_transform(x_test)
```

[13]:
```
x_train
```

[13]:
```
array([[-0.23190535, -1.84245871,  0.45326153, ...,  0.02751629,
         0.83401434,  1.42166275],
       [-0.52374511,  0.55407797, -0.84588607, ...,  1.70270196,
         0.18723471,  0.37290974],
       [ 0.11830237, -1.07115955,  0.71309105, ...,  0.4140976 ,
         0.01084027,  1.1356392 ],
       ...,
       [-0.34864125,  0.85708836, -0.22229522, ..., -0.03691393,
        -1.0475264 , -0.86652564],
       [-0.99068874,  1.10500594, -1.2616133 , ...,  0.99396956,
        -0.75353566, -0.48516091],
       [ 0.29340624, -1.62208752,  1.44061371, ...,  0.47852782,
        -0.69473751, -1.15254919]])
```

[14]:
```
x_test
```

[14]:
```
array([[ 0.83963515,  0.04571776,  1.37174921, ..., -0.66865691,
         0.6058255 ,  0.90524944],
       [ 0.08005559, -1.97474479,  1.08285357, ...,  0.20713709,
         0.6058255 ,  1.07632962],
       [-0.19122283, -1.57065228,  0.69765939, ...,  0.51992066,
        -0.32948404,  1.16186971],
       ...,
       [-0.84229102,  0.96935778, -1.37275933, ...,  1.02037438,
         0.37199811, -0.29231182],
       [ 0.08005559, -0.18519225, -0.84311733, ..., -0.79377034,
        -0.73868197, -0.97663254],
       [-0.24547851, -1.10883227,  0.60136085, ..., -0.04308976,
         0.31354127,  0.3920089 ]])
```

1 LinearRegression

[15]:
```
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(x_train,y_train)
```

[15]:
```
LinearRegression()
```

[16]:
```
#Predicting the test set result
y_pred = lr.predict(x_test)
```

[17]:
```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
# Calculate evaluation metrics
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, y_pred)

print("Mean Absolute Error:", mae)
print("Mean Squared Error:", mse)
print("Root Mean Squared Error:", rmse)
print("R-squared:", r2)
```

```
Mean Absolute Error: 0.5143692288772529
Mean Squared Error: 0.4466318515999171
Root Mean Squared Error: 0.6683052084189657
R-squared: 0.28434612651882896
```

2 Support Vector Regressor

[18]:
```
from sklearn.svm import SVR
svr = SVR(kernel='rbf')
svr.fit(x_train,y_train)
```

```
C:\Users\shivs\anaconda3\Lib\site-packages\sklearn\utils\validation.py:1184:
DataConversionWarning: A column-vector y was passed when a 1d array was
expected. Please change the shape of y to (n_samples, ), for example using
ravel().
  y = column_or_1d(y, warn=True)
```

[18]:
```
SVR()
```

[19]:
```
y_pred_svr = svr.predict(x_test)
y_pred_svr
```

[19]:
```
array([6.07837003, 6.21826791, 6.06904514, 5.51922936, 5.49701865,
       5.0464841 , 5.85929132, 4.98278073, 5.1664285 , 6.58155529,
       5.46525041, 5.1854067 , 5.01439475, 4.98823248, 5.09041047,
       4.9491099 , 5.89835232, 5.84271973, 5.15769836, 6.30728981,
       5.25079339, 5.11742525, 4.96345776, 5.55225868, 5.57779887,
       5.71717254, 5.71702483, 5.02673959, 5.62610822, 4.92411501,
       5.57094214, 5.63706285, 5.18218594, 5.18288003, 5.64892325,
       5.91944208, 5.09200407, 5.26596778, 5.5188043 , 5.66189109,
       5.60674896, 5.66189109, 4.95561156, 5.7344404 , 5.8977396 ,
       4.91987445, 5.93325773, 5.8172807 , 5.45154511, 6.25583049,
       5.76743799, 6.21284083, 5.34438289, 5.70556665, 6.56072001,
       6.57440205, 5.90523183, 5.54476259, 6.31490625, 4.95384797,
       5.56230762, 6.60520854, 5.97264289, 5.65388327, 5.92529893,
       5.16445224, 4.92008195, 6.10151423, 4.98698548, 5.90192542,
       5.12279306, 5.85675985, 5.57175204, 5.8597259 , 5.91216414,
       4.95126465, 6.24820242, 4.88394445, 5.85000763, 5.67003837,
       5.20885606, 5.8919132 , 5.73204691, 6.10255564, 5.64319119,
       4.73133567, 5.3838868 , 5.39135041, 5.46405134, 6.22181804,
       5.31143196, 5.83319738, 5.12380685, 5.09155609, 5.91865445,
       4.92861213, 6.08852023, 6.23200664, 5.58996514, 6.458952  ,
       5.73038932, 5.32125964, 5.74097941, 5.70556665, 4.95384797,
       5.33495576, 5.70172189, 5.37002236, 6.68142775, 6.7618333 ,
       5.30103648, 5.13533283, 6.37027081, 5.9229749 , 5.33970578,
       5.3266891 , 5.66510044, 6.20613738, 6.68880678, 6.479259  ,
       6.11621348, 5.2303019 , 7.12766649, 4.85362213, 5.65693876,
       5.42227329, 6.11363992, 6.22387461, 5.24589517, 5.51960768,
       5.82847846, 5.62704069, 5.73131164, 5.70417154, 5.18720504,
       5.18218594, 5.72884243, 5.16873236, 5.38439838, 6.9409716 ,
       5.21592012, 6.97578144, 5.46438735, 6.03245621, 4.969765  ,
       6.92325775, 6.21707597, 5.62374367, 5.62610822, 4.68327915,
       5.46833266, 5.0461287 , 5.01505039, 5.25786767, 5.29579624,
       5.27912451, 5.37498789, 4.86426885, 4.9093795 , 5.07673976,
       6.56235532, 4.96168843, 6.64474326, 5.52727373, 5.67674338,
       6.83946077, 6.41378527, 5.85830233, 5.25718578, 5.95515385,
       5.02324894, 5.75584941, 5.50844586, 5.49749899, 4.98934389,
       6.08694428, 6.02759942, 5.16358006, 5.32478633, 4.99155161,
       5.82453786, 5.36393021, 6.01345298, 5.21445665, 5.34511376,
       5.27430361, 6.5723734 , 5.12060014, 5.46020247, 6.13694716,
       5.43988205, 5.07688471, 5.99386761, 5.075961  , 6.08327897,
       6.10380093, 5.17646139, 6.16585042, 5.11728571, 5.08219642,
       5.2643713 , 5.13979551, 5.37518733, 5.21137363, 5.87138566,
       5.60206626, 5.01108803, 5.49026155, 6.40673387, 5.52558597,
       5.50424672, 5.10389012, 5.91709832, 5.84783973, 6.40804598,
       5.04673667, 5.22347667, 5.17272174, 5.64563315, 6.2992712 ,
       5.19577953, 6.84031999, 5.02871918, 6.15889257, 5.84261355,
       5.71978398, 5.04857979, 5.98444988, 6.08571055, 5.59283342,
       6.86239995, 5.09639639, 5.65334324, 5.30666866, 5.17947265,
       6.46929686, 5.32537935, 4.88347585, 5.91433563, 5.92286152,
       4.79344064, 4.83731738, 5.24264032, 5.53695642, 6.01565189,
       4.95773032, 6.2762183 , 5.43988205, 5.56615368, 5.4402394 ,
       5.54925322, 5.05877033, 6.42155046, 5.91579167, 6.5802308 ,
       5.1796325 , 6.41881052, 6.08795147, 6.82741404, 5.37239931,
       5.01757471, 5.49854768, 4.91436746, 6.65446805, 4.91056632,
       5.10785202, 6.46357597, 5.79205622, 6.35300457, 6.10443234,
       5.27957812, 5.65766101, 5.78030912, 6.1301421 , 6.40673387,
       6.6287756 , 6.35449889, 5.43556546, 5.74692251, 5.91709832,
       5.81100503, 6.2555767 , 6.14029902, 5.47493124, 5.30542261,
       6.15354495])
```

[20]:
```
# Calculate evaluation metrics
mae_svr = mean_absolute_error(y_test, y_pred_svr)
mse_svr = mean_squared_error(y_test, y_pred_svr)
rmse_svr = np.sqrt(mse_svr)
r2_svr = r2_score(y_test, y_pred_svr)

print("Mean Absolute Error:", mae_svr)
print("Mean Squared Error:", mse_svr)
print("Root Mean Squared Error:", rmse_svr)
print("R-squared:", r2_svr)
```

```
Mean Absolute Error: 0.47655708362025173
Mean Squared Error: 0.41097827040612117
Root Mean Squared Error: 0.6410758694617363
R-squared: 0.3414751095804127
```

3 Decision Tree Regressor

[21]:
```
from sklearn.tree import DecisionTreeRegressor
dtr = DecisionTreeRegressor(random_state=12)
dtr.fit(x_train, y_train)
```

[21]:
```
DecisionTreeRegressor(random_state=12)
```

[22]:
```
y_pred_dtr = dtr.predict(x_test)
y_pred_dtr
```

[22]:
```
array([6., 5., 6., 5., 5., 5., 6., 5., 6., 5., 5., 4., 5., 4., 5., 5., 6.,
       8., 5., 7., 5., 5., 5., 5., 6., 5., 6., 5., 5., 5., 6., 7., 5., 5.,
       6., 5., 6., 6., 5., 5., 6., 5., 5., 5., 6., 6., 5., 7., 6., 6., 5.,
       6., 6., 6., 7., 7., 6., 5., 6., 5., 5., 7., 4., 6., 6., 5., 5., 6.,
       4., 6., 5., 5., 6., 6., 6., 5., 8., 5., 7., 5., 5., 6., 5., 5., 6.,
       5., 5., 5., 5., 6., 5., 6., 5., 5., 4., 5., 6., 6., 6., 6., 5., 5.,
       7., 6., 5., 6., 7., 5., 7., 7., 5., 5., 6., 6., 5., 6., 6., 6., 7.,
       6., 7., 5., 6., 6., 6., 4., 7., 6., 5., 6., 6., 6., 6., 5., 5., 5.,
       5., 5., 5., 7., 6., 7., 5., 6., 5., 8., 6., 5., 5., 5., 6., 5., 5.,
       6., 5., 5., 5., 5., 5., 5., 8., 5., 6., 6., 6., 7., 6., 5., 5., 6.,
       6., 6., 5., 5., 5., 6., 6., 4., 5., 5., 6., 5., 6., 5., 5., 5., 6.,
       5., 6., 6., 5., 5., 6., 5., 6., 7., 5., 6., 5., 5., 5., 4., 6., 5.,
       4., 5., 5., 5., 6., 5., 6., 6., 5., 5., 7., 5., 6., 5., 5., 7., 5.,
       6., 5., 5., 6., 6., 5., 6., 6., 7., 7., 5., 6., 5., 5., 5., 4., 3.,
       7., 6., 5., 5., 5., 6., 6., 5., 7., 5., 6., 6., 6., 5., 7., 6., 7.,
       6., 5., 5., 7., 6., 5., 6., 5., 6., 5., 6., 6., 7., 7., 7., 5., 7.,
       6., 6., 6., 8., 6., 6., 6., 5., 7., 6., 6., 6., 6., 7.])
```

[23]:
```
# Calculate evaluation metrics
mae_dtr = mean_absolute_error(y_test, y_pred_dtr)
mse_dtr = mean_squared_error(y_test, y_pred_dtr)
rmse_dtr = np.sqrt(mse_dtr)
r2_dtr = r2_score(y_test, y_pred_dtr)

print("Mean Absolute Error:", mae_dtr)
print("Mean Squared Error:", mse_dtr)
print("Root Mean Squared Error:", rmse_dtr)
print("R-squared:", r2_dtr)
```

```
Mean Absolute Error: 0.5559440559440559
Mean Squared Error: 0.7657342657342657
Root Mean Squared Error: 0.8750624353349112
R-squared: -0.22696285848613074
```

In summary, lower values for MAE, MSE, and RMSE are desirable, as they indicate better predictive performance. A higher R^2 value indicates a better fit of the model to the data, with values closer to 1 being better. Algorithm 2 seems to perform better than Algorithm 1 and Algorithm 3 based on these metrics.

4 Perform k-fold cross-validation

1. Linear Regression

[24]:
```
from sklearn.model_selection import cross_val_score
lr_k = LinearRegression()
k = 5
mse_scores = -cross_val_score(lr_k, x, y, cv=k, scoring='neg_mean_squared_error')
```

we use - here because of we use scoring='neg_mean_squared_error' this

[25]:
```
mse_scores
```

[25]:
```
array([0.44290968, 0.46117682, 0.40050413, 0.40189841, 0.4247529 ])
```

2. Support Vector Regressor

[26]:
```
from sklearn.model_selection import GridSearchCV, KFold

# Define the parameter grid
param_grid = {'C': [0.01, 0.1, 1, 10], 'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.
 ↪01, 0.05]}
```

[27]:
```
# Create an SVR model
svr_k = SVR()

# Initialize KFold with k splits
k = 5
```

```
kf = KFold(n_splits=k, shuffle=True, random_state=12)
```

[28]:
```
# Perform grid search with cross-validation
grid_search = GridSearchCV(svr_k, param_grid, cv=kf,␣
↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(x, y.values.ravel())
```

[28]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=12, shuffle=True),
             estimator=SVR(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10],
                         'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05]},
             scoring='neg_mean_squared_error')
```

- GridSearchCV is a technique used for hyperparameter tuning in machine learning. It's employed to find the best combination of hyperparameters for a machine learning model, leading to improved model performance.
- cv=kf: This parameter specifies the cross-validation strategy to use during the grid search
- n_jobs=-1: This parameter specifies the number of CPU cores to use for parallel computation. Setting it to -1 means to use all available CPU cores, which can significantly speed up the grid search if you have a multi-core processor.

[29]:
```
# Get the best hyperparameters
grid_search.best_params_
```

[29]:
```
{'C': 10, 'gamma': 0.005}
```

[30]:
```
# Get the best estimator (SVR model with best hyperparameters)
best_model = grid_search.best_estimator_
```

[31]:
```
# Calculate the average Mean Squared Error across all folds using the best model
mse_scores = -cross_val_score(best_model, x, y.values.ravel(), cv=k,␣
↪scoring='neg_mean_squared_error')
```

[32]:
```
mse_scores
```

[32]:
```
array([0.46114719, 0.54387765, 0.3905256 , 0.51144096, 0.49882219])
```

- now get best score 0.390526. This is the lowest MSE value, and therefore, it represents the best performance on the third fold of cross-validation.

5 Improvement

[33]:
```
# train test split
x_train, x_test, y_train, y_test = train_test_split(x, y, test_size= .
↪20,random_state=12)
```

[34]:
```
# Define the parameter grid
param_grid_1 = {'C': [0.01, 0.1, 1, 10], 'gamma': [0.0001, 0.0005, 0.001, 0.005,␣
↪0.01, 0.05]}
```

[35]:
```
# Create an SVR model
svr_k = SVR()

# Initialize KFold with k splits
k = 5
kf = KFold(n_splits=k, shuffle=True, random_state=12)
```

[36]:
```
# Perform grid search with cross-validation
grid_search = GridSearchCV(svr_k, param_grid_1, cv=kf,␣
↪scoring='neg_mean_squared_error', n_jobs=-1)
grid_search.fit(x_train, y_train.values.ravel())
```

[36]:
```
GridSearchCV(cv=KFold(n_splits=5, random_state=12, shuffle=True),
             estimator=SVR(), n_jobs=-1,
             param_grid={'C': [0.01, 0.1, 1, 10],
                         'gamma': [0.0001, 0.0005, 0.001, 0.005, 0.01, 0.05]},
             scoring='neg_mean_squared_error')
```

[37]:
```
# Get the best estimator (SVR model with best hyperparameters)
best_model = grid_search.best_estimator_
```

[38]:
```
# Calculate the average Mean Squared Error across all folds using the best model
mse_scores = -cross_val_score(best_model, x_train, y_train.values.ravel(),␣
↪cv=kf, scoring='neg_mean_squared_error')
mse_scores.mean()
```

[38]:
```
0.46535398740587636
```

- Average Mean Squared Error (Cross-Validation): 0.46535398740587597

[39]:
```
# Calculate Mean Squared Error on the test set using the best model
y_pred = best_model.predict(x_test)
mse_test = mean_squared_error(y_test, y_pred)
mse_test
```

[39]:
```
0.49675628780242
```

- Mean Squared Error on Test Set: 0.49675628780242076