wine_quality_red_and_white

May 11, 2023

[1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

[2]:
```
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
red_wine = pd.read_csv("winequality-red.csv")
white_wine = pd.read_csv("winequality-white.csv")
```

[4]:
```
red_wine.shape
```

[4]:
```
(1599, 12)
```

[5]:
```
white_wine.shape
```

[5]:
```
(4898, 12)
```

[6]:
```
wine = pd.concat([red_wine, white_wine], ignore_index=True)
```

[7]:
```
wine_df = pd.concat([red_wine, white_wine], ignore_index=True)
```

[8]:
```
wine.shape
```

[8]:
```
(6497, 12)
```

[9]:
```
wine.head()
```

[9]:
```
   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
```

[10]:
```
wine.tail()
```

[10]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
6492            6.2              0.21         0.29             1.6      0.039
6493            6.6              0.32         0.36             8.0      0.047
6494            6.5              0.24         0.19             1.2      0.041
6495            5.5              0.29         0.30             1.1      0.022
6496            6.0              0.21         0.38             0.8      0.020

      free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
6492                 24.0                  92.0  0.99114  3.27       0.50
6493                 57.0                 168.0  0.99490  3.15       0.46
6494                 30.0                 111.0  0.99254  2.99       0.46
6495                 20.0                 110.0  0.98869  3.34       0.38
6496                 22.0                  98.0  0.98941  3.26       0.32

      alcohol  quality
6492     11.2        6
6493      9.6        5
6494      9.4        6
6495     12.8        7
6496     11.8        6
```

[11]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6497 entries, 0 to 6496
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         6497 non-null   float64
 1   volatile acidity      6497 non-null   float64
 2   citric acid           6497 non-null   float64
 3   residual sugar        6497 non-null   float64
 4   chlorides             6497 non-null   float64
 5   free sulfur dioxide   6497 non-null   float64
```

```
 6   total sulfur dioxide  6497 non-null   float64
 7   density               6497 non-null   float64
 8   pH                    6497 non-null   float64
 9   sulphates             6497 non-null   float64
 10  alcohol               6497 non-null   float64
 11  quality               6497 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 609.2 KB
```

[12]:
```
wine.isnull().sum()
```

[12]:
```
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
```

[13]:
```
wine.describe()
```

[13]:
```
       fixed acidity  volatile acidity  citric acid  residual sugar  \
count    6497.000000       6497.000000  6497.000000     6497.000000
mean        7.215307          0.339666     0.318633        5.443235
std         1.296434          0.164636     0.145318        4.757804
min         3.800000          0.080000     0.000000        0.600000
25%         6.400000          0.230000     0.250000        1.800000
50%         7.000000          0.290000     0.310000        3.000000
75%         7.700000          0.400000     0.390000        8.100000
max        15.900000          1.580000     1.660000       65.800000

         chlorides  free sulfur dioxide  total sulfur dioxide      density  \
count  6497.000000          6497.000000           6497.000000  6497.000000
mean      0.056034            30.525319            115.744574     0.994697
std       0.035034            17.749400             56.521855     0.002999
min       0.009000             1.000000              6.000000     0.987110
25%       0.038000            17.000000             77.000000     0.992340
50%       0.047000            29.000000            118.000000     0.994890
75%       0.065000            41.000000            156.000000     0.996990
max       0.611000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
```

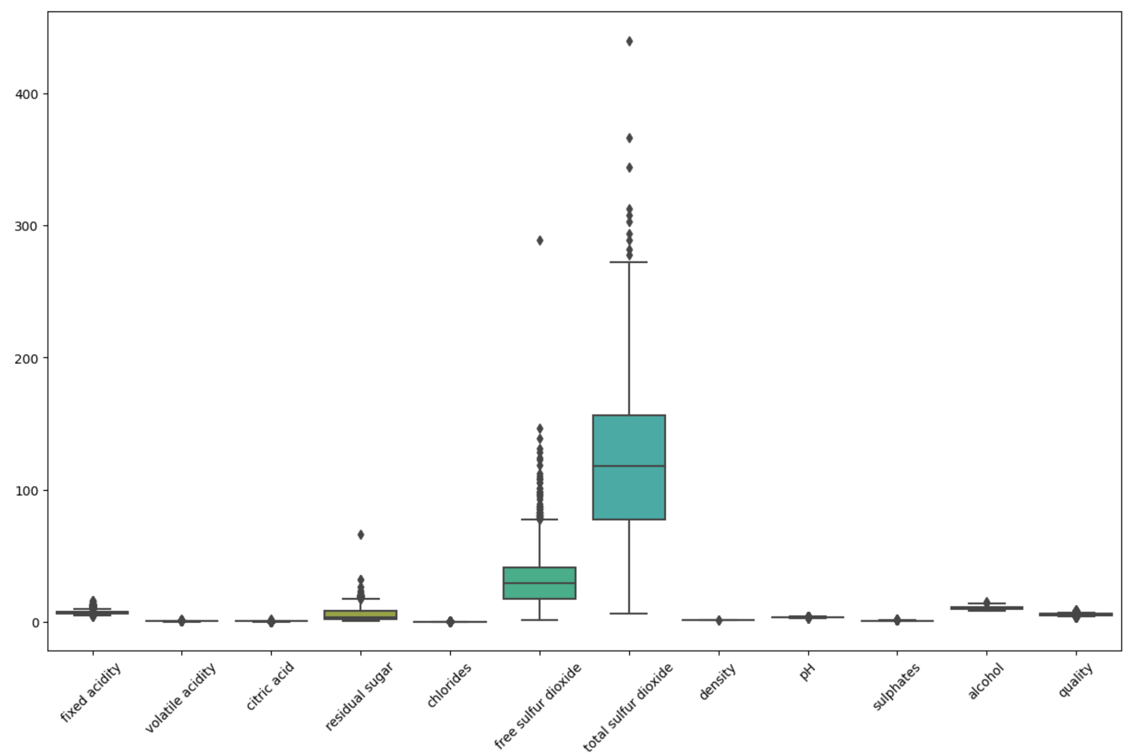
```
count  6497.000000  6497.000000  6497.000000  6497.000000
mean      3.218501     0.531268    10.491801     5.818378
std       0.160787     0.148806     1.192712     0.873255
min       2.720000     0.220000     8.000000     3.000000
25%       3.110000     0.430000     9.500000     5.000000
50%       3.210000     0.510000    10.300000     6.000000
75%       3.320000     0.600000    11.300000     6.000000
max       4.010000     2.000000    14.900000     9.000000
```

0.1 Outlier detection and Visualization

[14]:
```python
from scipy import stats
import seaborn as sns
```

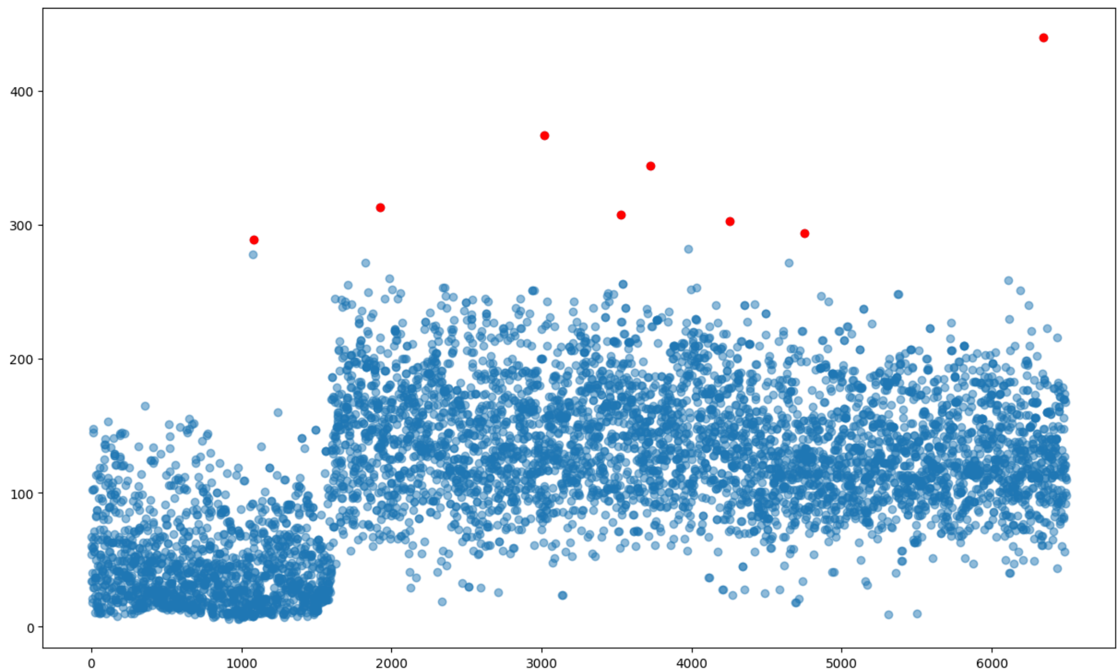
[15]:
```python
plt.subplots(figsize=(15, 9))
plt.xticks(rotation=45)
sns.boxplot(data=wine)
```

[15]: `<Axes: >`

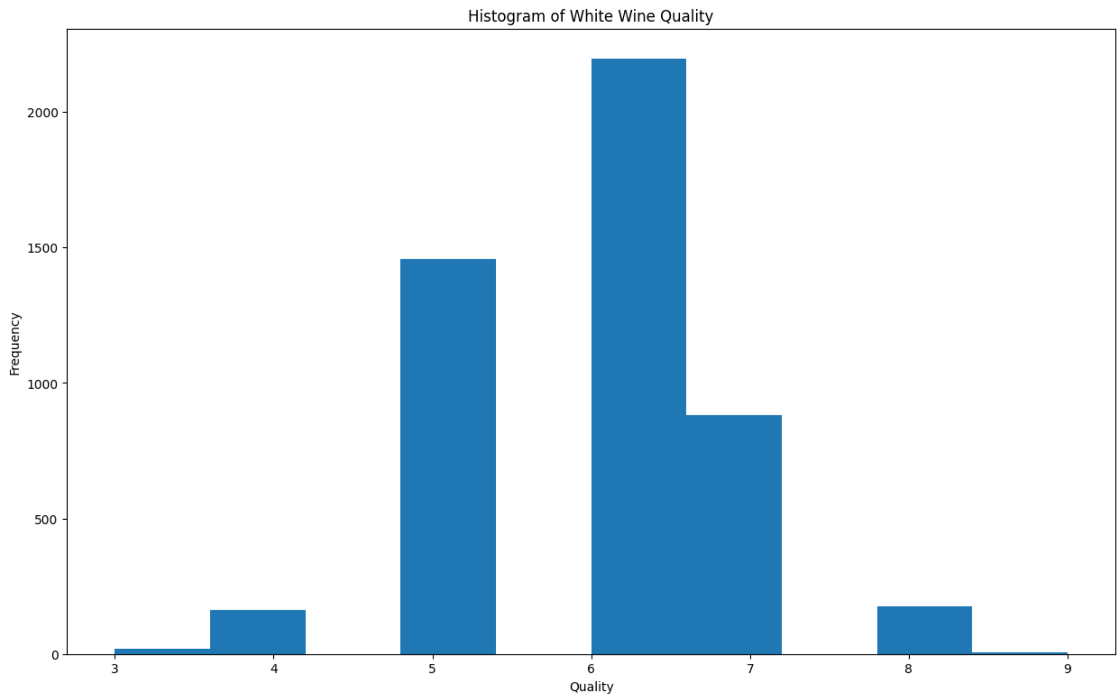


[16]:
```python
# calculate z-scores and identify outliers
z_scores = stats.zscore(wine['total sulfur dioxide'])
threshold = 3
```

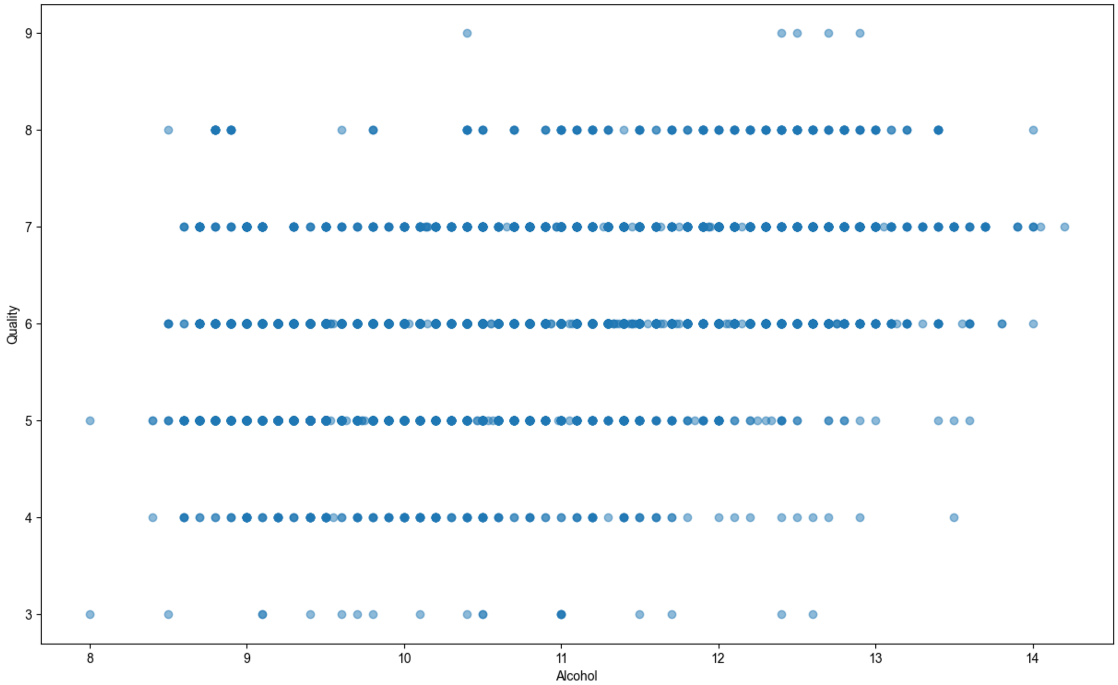
```
outliers = wine[np.abs(z_scores) > threshold]

# create scatter plot
fig, ax = plt.subplots(figsize=(15, 9))
ax.scatter(wine.index, wine['total sulfur dioxide'], alpha=0.5)
ax.scatter(outliers.index, outliers['total sulfur dioxide'], color='r')
plt.show()
```



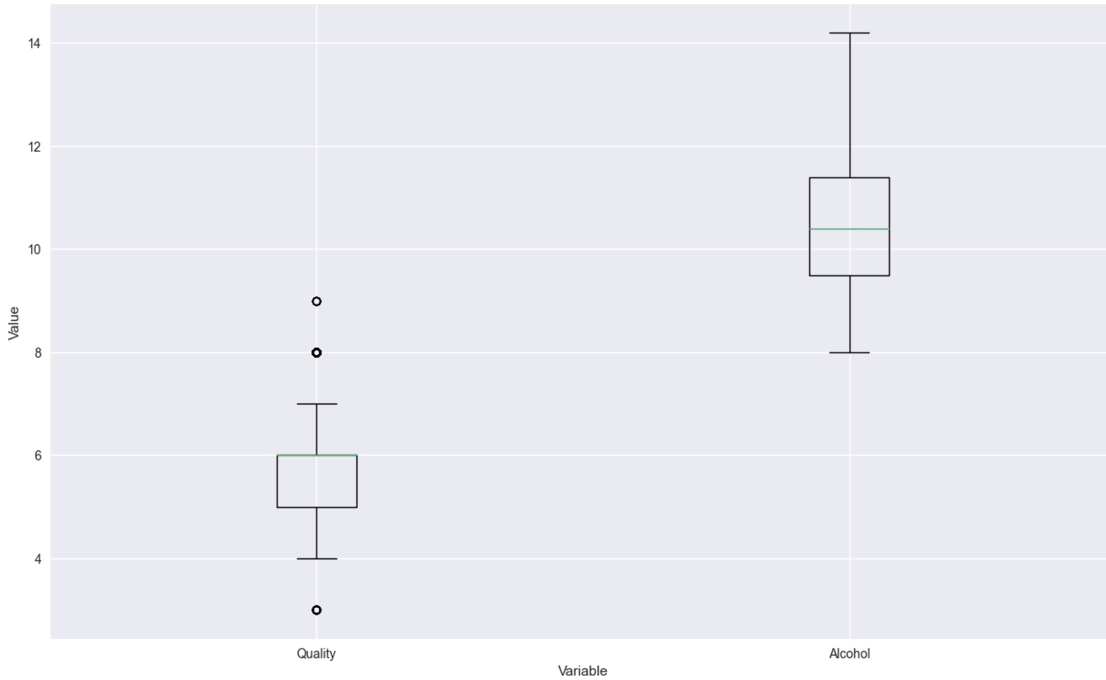
[17]:
```
# create a histogram with larger figure size
fig, ax = plt.subplots(figsize=(15, 9))
ax.hist(white_wine['quality'])
ax.set_xlabel('Quality')
ax.set_ylabel('Frequency')
ax.set_title('Histogram of White Wine Quality')
plt.show()
```



[18]:
```
# create scatter plot with larger figure size
ig, ax = plt.subplots(figsize=(15, 9))
ax.scatter(x='alcohol', y='quality', data=white_wine, alpha=0.5)

# set style property
ax.set(xlabel='Alcohol', ylabel='Quality')
plt.style.use('seaborn')  # set style to seaborn

plt.show()
```



[19]:
```
fig, ax = plt.subplots(figsize=(15, 9))
ax.boxplot([white_wine['quality'], white_wine['alcohol']], labels=['Quality',␣
↪'Alcohol'])

# set style property
ax.set(xlabel='Variable', ylabel='Value')
plt.style.use('seaborn')

plt.show()
```



0.1.1 Splitting the datat

[20]:
```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

[21]:
```
X = wine.drop('quality', axis=1) # Extract the input features
y = wine['quality'] # Extract the target variable
```

[22]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y,  test_size=0.2,␣
↪random_state=42)
```

[23]:
```
sc = StandardScaler()
```

[24]:
```
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

0.2 1. Linear regression model

[25]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

[26]:
```
# Create a linear regression model
lr_model = LinearRegression()
```

[27]:
```
# Fit the model on the training set
lr_model.fit(X_train, y_train)
```

[27]:
```
LinearRegression()
```

[28]:
```
# Make predictions on the testing set
y_pred = lr_model.predict(X_test)
```

[29]:
```
# Calculate RMSE, MAE, MSE, and R2 score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

[30]:
```
print('RMSE:', rmse)
print('MAE:', mae)
print('MSE:', mse)
print('R-squared Score:', r2)
```

```
RMSE: 0.7393892357611412
MAE: 0.5658710079723465
MSE: 0.5466964419594444
R-squared Score: 0.2597673129771396
```

0.3 2. XGBoost

[31]:
```
from xgboost import XGBRegressor
```

[32]:
```
# Create an XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
```

[33]:
```
xgb_model.fit(X_train, y_train)
```

[33]:
```
XGBRegressor(base_score=None, booster=None, callbacks=None,
             colsample_bylevel=None, colsample_bynode=None,
             colsample_bytree=None, early_stopping_rounds=None,
             enable_categorical=False, eval_metric=None, feature_types=None,
             gamma=None, gpu_id=None, grow_policy=None, importance_type=None,
             interaction_constraints=None, learning_rate=None, max_bin=None,
             max_cat_threshold=None, max_cat_to_onehot=None,
             max_delta_step=None, max_depth=None, max_leaves=None,
             min_child_weight=None, missing=nan, monotone_constraints=None,
             n_estimators=100, n_jobs=None, num_parallel_tree=None,
             predictor=None, random_state=42, …)
```

[34]:
```
# Make predictions on the testing set
y_pred = xgb_model.predict(X_test)
```

[35]:
```
# Calculate RMSE, MAE, MSE, and R2 score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

[36]:
```
print('RMSE:', rmse)
print('MAE:', mae)
print('MSE:', mse)
print('R-squared Score:', r2)
```

```
RMSE: 0.6290334979171012
MAE: 0.46276961509998027
MSE: 0.39568314150182365
R-squared Score: 0.4642408975742527
```

0.4 3. ANN

[37]:
```
import keras
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
```

[38]:
```
# Define hyperparameters
learning_rate = 0.001
num_epochs = 100
batch_size = 32
```

[39]:
```
# Create a neural network model
ann_model = Sequential()
ann_model.add(Dense(128, input_dim=X_train.shape[1], activation='relu'))
ann_model.add(Dense(64, activation='relu'))
ann_model.add(Dense(32, activation='relu'))
ann_model.add(Dense(16, activation='relu'))
ann_model.add(Dense(1))
```

[40]:
```
# Compile model with Adam optimizer and custom learning rate
opt = Adam(lr=learning_rate)
ann_model.compile(loss='mean_squared_error', optimizer=opt,␣
 ↪metrics=['mean_squared_error'])
```

[41]:
```
# Train model with specified batch size and number of epochs
history = ann_model.fit(X_train, y_train, batch_size=batch_size,␣
 ↪epochs=num_epochs, verbose=1, validation_split=0.2)
```

```
Epoch 1/100
130/130 [==============================] - 1s 4ms/step - loss: 6.2541 -
mean_squared_error: 6.2541 - val_loss: 1.5355 - val_mean_squared_error: 1.5355
Epoch 2/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 1.2199 -
mean_squared_error: 1.2199 - val_loss: 1.1021 - val_mean_squared_error: 1.1021
Epoch 3/100
130/130 [==============================] - 0s 2ms/step - loss: 0.8532 -
mean_squared_error: 0.8532 - val_loss: 0.8042 - val_mean_squared_error: 0.8042
Epoch 4/100
130/130 [==============================] - 0s 2ms/step - loss: 0.6464 -
mean_squared_error: 0.6464 - val_loss: 0.6210 - val_mean_squared_error: 0.6210
Epoch 5/100
130/130 [==============================] - 0s 2ms/step - loss: 0.5498 -
mean_squared_error: 0.5498 - val_loss: 0.5433 - val_mean_squared_error: 0.5433
Epoch 6/100
130/130 [==============================] - 0s 2ms/step - loss: 0.5227 -
mean_squared_error: 0.5227 - val_loss: 0.5158 - val_mean_squared_error: 0.5158
Epoch 7/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4941 -
mean_squared_error: 0.4941 - val_loss: 0.4747 - val_mean_squared_error: 0.4747
Epoch 8/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4894 -
mean_squared_error: 0.4894 - val_loss: 0.5111 - val_mean_squared_error: 0.5111
Epoch 9/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4689 -
mean_squared_error: 0.4689 - val_loss: 0.4761 - val_mean_squared_error: 0.4761
Epoch 10/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4790 -
mean_squared_error: 0.4790 - val_loss: 0.4642 - val_mean_squared_error: 0.4642
Epoch 11/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4620 -
mean_squared_error: 0.4620 - val_loss: 0.4596 - val_mean_squared_error: 0.4596
Epoch 12/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4494 -
mean_squared_error: 0.4494 - val_loss: 0.4653 - val_mean_squared_error: 0.4653
Epoch 13/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4469 -
mean_squared_error: 0.4469 - val_loss: 0.5055 - val_mean_squared_error: 0.5055
Epoch 14/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4445 -
mean_squared_error: 0.4445 - val_loss: 0.4848 - val_mean_squared_error: 0.4848
Epoch 15/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4559 -
mean_squared_error: 0.4559 - val_loss: 0.5377 - val_mean_squared_error: 0.5377
Epoch 16/100
130/130 [==============================] - 0s 3ms/step - loss: 0.4333 -
mean_squared_error: 0.4333 - val_loss: 0.4706 - val_mean_squared_error: 0.4706
Epoch 17/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4378 -
mean_squared_error: 0.4378 - val_loss: 0.4500 - val_mean_squared_error: 0.4500
Epoch 18/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.4318 -
mean_squared_error: 0.4318 - val_loss: 0.4828 - val_mean_squared_error: 0.4828
Epoch 19/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4246 -
mean_squared_error: 0.4246 - val_loss: 0.4553 - val_mean_squared_error: 0.4553
Epoch 20/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4245 -
mean_squared_error: 0.4245 - val_loss: 0.4597 - val_mean_squared_error: 0.4597
Epoch 21/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4227 -
mean_squared_error: 0.4227 - val_loss: 0.4772 - val_mean_squared_error: 0.4772
Epoch 22/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4118 -
mean_squared_error: 0.4118 - val_loss: 0.4584 - val_mean_squared_error: 0.4584
Epoch 23/100
130/130 [==============================] - 0s 2ms/step - loss: 0.4163 -
mean_squared_error: 0.4163 - val_loss: 0.4633 - val_mean_squared_error: 0.4633
Epoch 24/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3978 -
mean_squared_error: 0.3978 - val_loss: 0.4825 - val_mean_squared_error: 0.4825
Epoch 25/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3949 -
mean_squared_error: 0.3949 - val_loss: 0.4963 - val_mean_squared_error: 0.4963
Epoch 26/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3999 -
mean_squared_error: 0.3999 - val_loss: 0.4645 - val_mean_squared_error: 0.4645
Epoch 27/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3931 -
mean_squared_error: 0.3931 - val_loss: 0.4580 - val_mean_squared_error: 0.4580
Epoch 28/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3891 -
mean_squared_error: 0.3891 - val_loss: 0.4778 - val_mean_squared_error: 0.4778
Epoch 29/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3856 -
mean_squared_error: 0.3856 - val_loss: 0.4630 - val_mean_squared_error: 0.4630
Epoch 30/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3777 -
mean_squared_error: 0.3777 - val_loss: 0.4740 - val_mean_squared_error: 0.4740
Epoch 31/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3792 -
mean_squared_error: 0.3792 - val_loss: 0.4703 - val_mean_squared_error: 0.4703
Epoch 32/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3690 -
mean_squared_error: 0.3690 - val_loss: 0.4700 - val_mean_squared_error: 0.4700
Epoch 33/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3595 -
mean_squared_error: 0.3595 - val_loss: 0.4775 - val_mean_squared_error: 0.4775
Epoch 34/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.3592 -
mean_squared_error: 0.3592 - val_loss: 0.4874 - val_mean_squared_error: 0.4874
Epoch 35/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3687 -
mean_squared_error: 0.3687 - val_loss: 0.4961 - val_mean_squared_error: 0.4961
Epoch 36/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3535 -
mean_squared_error: 0.3535 - val_loss: 0.4768 - val_mean_squared_error: 0.4768
Epoch 37/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3508 -
mean_squared_error: 0.3508 - val_loss: 0.5262 - val_mean_squared_error: 0.5262
Epoch 38/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3428 -
mean_squared_error: 0.3428 - val_loss: 0.4937 - val_mean_squared_error: 0.4937
Epoch 39/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3452 -
mean_squared_error: 0.3452 - val_loss: 0.5408 - val_mean_squared_error: 0.5408
Epoch 40/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3427 -
mean_squared_error: 0.3427 - val_loss: 0.4933 - val_mean_squared_error: 0.4933
Epoch 41/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3418 -
mean_squared_error: 0.3418 - val_loss: 0.5198 - val_mean_squared_error: 0.5198
Epoch 42/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3401 -
mean_squared_error: 0.3401 - val_loss: 0.4910 - val_mean_squared_error: 0.4910
Epoch 43/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3252 -
mean_squared_error: 0.3252 - val_loss: 0.4802 - val_mean_squared_error: 0.4802
Epoch 44/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3223 -
mean_squared_error: 0.3223 - val_loss: 0.5325 - val_mean_squared_error: 0.5325
Epoch 45/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3339 -
mean_squared_error: 0.3339 - val_loss: 0.4926 - val_mean_squared_error: 0.4926
Epoch 46/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3324 -
mean_squared_error: 0.3324 - val_loss: 0.5056 - val_mean_squared_error: 0.5056
Epoch 47/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3126 -
mean_squared_error: 0.3126 - val_loss: 0.5351 - val_mean_squared_error: 0.5351
Epoch 48/100
130/130 [==============================] - 0s 3ms/step - loss: 0.3108 -
mean_squared_error: 0.3108 - val_loss: 0.5218 - val_mean_squared_error: 0.5218
Epoch 49/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3121 -
mean_squared_error: 0.3121 - val_loss: 0.5193 - val_mean_squared_error: 0.5193
Epoch 50/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.3199 -
mean_squared_error: 0.3199 - val_loss: 0.5085 - val_mean_squared_error: 0.5085
Epoch 51/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3040 -
mean_squared_error: 0.3040 - val_loss: 0.5048 - val_mean_squared_error: 0.5048
Epoch 52/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2960 -
mean_squared_error: 0.2960 - val_loss: 0.5012 - val_mean_squared_error: 0.5012
Epoch 53/100
130/130 [==============================] - 0s 2ms/step - loss: 0.3010 -
mean_squared_error: 0.3010 - val_loss: 0.5199 - val_mean_squared_error: 0.5199
Epoch 54/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2932 -
mean_squared_error: 0.2932 - val_loss: 0.5221 - val_mean_squared_error: 0.5221
Epoch 55/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2968 -
mean_squared_error: 0.2968 - val_loss: 0.5047 - val_mean_squared_error: 0.5047
Epoch 56/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2980 -
mean_squared_error: 0.2980 - val_loss: 0.4891 - val_mean_squared_error: 0.4891
Epoch 57/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2800 -
mean_squared_error: 0.2800 - val_loss: 0.5329 - val_mean_squared_error: 0.5329
Epoch 58/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2897 -
mean_squared_error: 0.2897 - val_loss: 0.5072 - val_mean_squared_error: 0.5072
Epoch 59/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2810 -
mean_squared_error: 0.2810 - val_loss: 0.5224 - val_mean_squared_error: 0.5224
Epoch 60/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2908 -
mean_squared_error: 0.2908 - val_loss: 0.5199 - val_mean_squared_error: 0.5199
Epoch 61/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2853 -
mean_squared_error: 0.2853 - val_loss: 0.5009 - val_mean_squared_error: 0.5009
Epoch 62/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2748 -
mean_squared_error: 0.2748 - val_loss: 0.5858 - val_mean_squared_error: 0.5858
Epoch 63/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2782 -
mean_squared_error: 0.2782 - val_loss: 0.5020 - val_mean_squared_error: 0.5020
Epoch 64/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2629 -
mean_squared_error: 0.2629 - val_loss: 0.5143 - val_mean_squared_error: 0.5143
Epoch 65/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2745 -
mean_squared_error: 0.2745 - val_loss: 0.5242 - val_mean_squared_error: 0.5242
Epoch 66/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.2696 -
mean_squared_error: 0.2696 - val_loss: 0.5357 - val_mean_squared_error: 0.5357
Epoch 67/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2733 -
mean_squared_error: 0.2733 - val_loss: 0.5202 - val_mean_squared_error: 0.5202
Epoch 68/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2557 -
mean_squared_error: 0.2557 - val_loss: 0.5172 - val_mean_squared_error: 0.5172
Epoch 69/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2545 -
mean_squared_error: 0.2545 - val_loss: 0.5313 - val_mean_squared_error: 0.5313
Epoch 70/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2620 -
mean_squared_error: 0.2620 - val_loss: 0.4915 - val_mean_squared_error: 0.4915
Epoch 71/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2492 -
mean_squared_error: 0.2492 - val_loss: 0.5281 - val_mean_squared_error: 0.5281
Epoch 72/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2508 -
mean_squared_error: 0.2508 - val_loss: 0.5160 - val_mean_squared_error: 0.5160
Epoch 73/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2527 -
mean_squared_error: 0.2527 - val_loss: 0.5283 - val_mean_squared_error: 0.5283
Epoch 74/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2354 -
mean_squared_error: 0.2354 - val_loss: 0.5270 - val_mean_squared_error: 0.5270
Epoch 75/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2463 -
mean_squared_error: 0.2463 - val_loss: 0.5150 - val_mean_squared_error: 0.5150
Epoch 76/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2412 -
mean_squared_error: 0.2412 - val_loss: 0.4939 - val_mean_squared_error: 0.4939
Epoch 77/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2572 -
mean_squared_error: 0.2572 - val_loss: 0.5339 - val_mean_squared_error: 0.5339
Epoch 78/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2381 -
mean_squared_error: 0.2381 - val_loss: 0.5302 - val_mean_squared_error: 0.5302
Epoch 79/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2293 -
mean_squared_error: 0.2293 - val_loss: 0.5112 - val_mean_squared_error: 0.5112
Epoch 80/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2370 -
mean_squared_error: 0.2370 - val_loss: 0.5078 - val_mean_squared_error: 0.5078
Epoch 81/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2315 -
mean_squared_error: 0.2315 - val_loss: 0.4973 - val_mean_squared_error: 0.4973
Epoch 82/100
```

```
130/130 [==============================] - 0s 3ms/step - loss: 0.2296 -
mean_squared_error: 0.2296 - val_loss: 0.5417 - val_mean_squared_error: 0.5417
Epoch 83/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2241 -
mean_squared_error: 0.2241 - val_loss: 0.5404 - val_mean_squared_error: 0.5404
Epoch 84/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2262 -
mean_squared_error: 0.2262 - val_loss: 0.5882 - val_mean_squared_error: 0.5882
Epoch 85/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2276 -
mean_squared_error: 0.2276 - val_loss: 0.5403 - val_mean_squared_error: 0.5403
Epoch 86/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2213 -
mean_squared_error: 0.2213 - val_loss: 0.5317 - val_mean_squared_error: 0.5317
Epoch 87/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2154 -
mean_squared_error: 0.2154 - val_loss: 0.5465 - val_mean_squared_error: 0.5465
Epoch 88/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2271 -
mean_squared_error: 0.2271 - val_loss: 0.5295 - val_mean_squared_error: 0.5295
Epoch 89/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2131 -
mean_squared_error: 0.2131 - val_loss: 0.5705 - val_mean_squared_error: 0.5705
Epoch 90/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2260 -
mean_squared_error: 0.2260 - val_loss: 0.5359 - val_mean_squared_error: 0.5359
Epoch 91/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2154 -
mean_squared_error: 0.2154 - val_loss: 0.5451 - val_mean_squared_error: 0.5451
Epoch 92/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2021 -
mean_squared_error: 0.2021 - val_loss: 0.5389 - val_mean_squared_error: 0.5389
Epoch 93/100
130/130 [==============================] - 0s 3ms/step - loss: 0.2029 -
mean_squared_error: 0.2029 - val_loss: 0.5346 - val_mean_squared_error: 0.5346
Epoch 94/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2106 -
mean_squared_error: 0.2106 - val_loss: 0.5739 - val_mean_squared_error: 0.5739
Epoch 95/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2074 -
mean_squared_error: 0.2074 - val_loss: 0.5516 - val_mean_squared_error: 0.5516
Epoch 96/100
130/130 [==============================] - 0s 3ms/step - loss: 0.1997 -
mean_squared_error: 0.1997 - val_loss: 0.5699 - val_mean_squared_error: 0.5699
Epoch 97/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2046 -
mean_squared_error: 0.2046 - val_loss: 0.5623 - val_mean_squared_error: 0.5623
Epoch 98/100
```

```
130/130 [==============================] - 0s 2ms/step - loss: 0.1976 -
mean_squared_error: 0.1976 - val_loss: 0.5765 - val_mean_squared_error: 0.5765
Epoch 99/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2179 -
mean_squared_error: 0.2179 - val_loss: 0.5628 - val_mean_squared_error: 0.5628
Epoch 100/100
130/130 [==============================] - 0s 2ms/step - loss: 0.2009 -
mean_squared_error: 0.2009 - val_loss: 0.5468 - val_mean_squared_error: 0.5468
```

[42]:
```
# Make predictions on the testing set
y_pred = ann_model.predict(X_test)
```

```
41/41 [==============================] - 0s 1ms/step
```

[43]:
```
# Calculate RMSE, MAE, MSE, and R2 score
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
mae = mean_absolute_error(y_test, y_pred)
mse = mean_squared_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)
```

[44]:
```
print('RMSE:', rmse)
print('MAE:', mae)
print('MSE:', mse)
print('R-squared Score:', r2)
```

```
RMSE: 0.7477269652378632
MAE: 0.5659084646518414
MSE: 0.5590956145438246
R-squared Score: 0.24297870391632415
```

[45]:
```
# separate the data and Label
x = wine_df.drop('quality',axis=1)
Y = wine_df['quality'].apply(lambda y_value: 1 if y_value>=7 else 0)
```

[46]:
```
X_train, X_test, y_train, y_test = train_test_split(x, Y, test_size=0.2,␣
↪random_state=42)
```

0.5 4. Logistic Regression

[47]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix
```

[48]:
```
sc = StandardScaler()
X_train_std = sc.fit_transform(X_train)
X_test_std = sc.transform(X_test)
```

[49]:
```python
# create and fit the logistic regression model
lr = LogisticRegression(random_state=42)
lr.fit(X_train_std, y_train)
```

[49]:
```
LogisticRegression(random_state=42)
```

[50]:
```python
# make predictions on the testing set
y_pred_lr = lr.predict(X_test_std)

# calculate the accuracy score and confusion matrix
acc_lr = accuracy_score(y_test, y_pred_lr)
cm_lr = confusion_matrix(y_test, y_pred_lr)
print('Accuracy Score (Logistic Regression):', acc_lr)
print('Confusion Matrix (Logistic Regression):\n', cm_lr)
```

```
Accuracy Score (Logistic Regression): 0.8246153846153846
Confusion Matrix (Logistic Regression):
 [[1004   44]
 [ 184   68]]
```

0.6 5. Random Forest model

[51]:
```python
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix
```

[52]:
```python
# Create and train the random forest model
rf_model = RandomForestClassifier()
rf_model.fit(X_train, y_train)
```

[52]:
```
RandomForestClassifier()
```

[53]:
```python
# Evaluate the random forest model
rf_pred = rf_model.predict(X_test)
rf_acc = accuracy_score(y_test, rf_pred)
print("Random Forest accuracy: ", rf_acc)
```

```
Random Forest accuracy:  0.8861538461538462
```

[54]:
```python
# Create confusion matrix
y_true = y_test
y_pred = rf_pred
conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)
```

```
[[1007   41]
 [ 107  145]]
```

0.7 6. Decision Tree

[55]:
```
from sklearn.tree import DecisionTreeRegressor
from sklearn.metrics import mean_squared_error
import matplotlib.pyplot as plt
```

[56]:
```
# Create a decision tree regressor with a maximum depth of 3
tree_model = DecisionTreeRegressor(max_depth=1)
```

[57]:
```
# Fit the model on the training data
tree_model.fit(X_train, y_train)
```

[57]:
```
DecisionTreeRegressor(max_depth=1)
```

[58]:
```
# Make predictions on the test data
dt_pred = tree_model.predict(X_test)
```

[59]:
```
# Evaluate the model
mse = mean_squared_error(y_test, dt_pred)
print("Decision tree MSE: ", mse)
```

```
Decision tree MSE:  0.13583711324969888
```

0.8 LSTM Model

[60]:
```
from sklearn.preprocessing import StandardScaler
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout
```

[61]:
```
X = wine.iloc[:,:-1].values
y = wine.iloc[:,-1].values
```

[62]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[63]:
```
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[64]:
```
X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], 1))
X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], 1))
# reshaping for LSTM model
```

[65]:
```
model = Sequential()
model.add(LSTM(50, return_sequences=True, input_shape=(X_train.shape[1], 1)))
model.add(LSTM(50))
model.add(Dense(1))
```

[66]:
```
model.compile(optimizer='adam', loss='mean_squared_error')
```

[67]:
```
model.fit(X_train, y_train, epochs=100, batch_size=32, validation_data=(X_test,␣
↪y_test))
```

```
Epoch 1/100
163/163 [==============================] - 6s 14ms/step - loss: 5.0015 -
val_loss: 0.6893
Epoch 2/100
163/163 [==============================] - 1s 9ms/step - loss: 0.6909 -
val_loss: 0.6364
Epoch 3/100
163/163 [==============================] - 1s 9ms/step - loss: 0.6278 -
val_loss: 0.5999
Epoch 4/100
163/163 [==============================] - 2s 9ms/step - loss: 0.6144 -
val_loss: 0.6202
Epoch 5/100
163/163 [==============================] - 2s 9ms/step - loss: 0.6094 -
val_loss: 0.5884
Epoch 6/100
163/163 [==============================] - 2s 9ms/step - loss: 0.6027 -
val_loss: 0.5918
Epoch 7/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5995 -
val_loss: 0.5829
Epoch 8/100
163/163 [==============================] - 2s 10ms/step - loss: 0.5935 -
val_loss: 0.5743
Epoch 9/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5942 -
val_loss: 0.5807
Epoch 10/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5865 -
val_loss: 0.5720
Epoch 11/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5804 -
val_loss: 0.5550
Epoch 12/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5765 -
val_loss: 0.6659
Epoch 13/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5790 -
val_loss: 0.5709
Epoch 14/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5725 -
val_loss: 0.5792
Epoch 15/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5662 -
val_loss: 0.5473
```

```
Epoch 16/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5657 -
val_loss: 0.5391
Epoch 17/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5587 -
val_loss: 0.5401
Epoch 18/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5599 -
val_loss: 0.5396
Epoch 19/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5520 -
val_loss: 0.5400
Epoch 20/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5581 -
val_loss: 0.5334
Epoch 21/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5549 -
val_loss: 0.5298
Epoch 22/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5509 -
val_loss: 0.5328
Epoch 23/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5451 -
val_loss: 0.5286
Epoch 24/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5478 -
val_loss: 0.5266
Epoch 25/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5434 -
val_loss: 0.5303
Epoch 26/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5442 -
val_loss: 0.5183
Epoch 27/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5377 -
val_loss: 0.5503
Epoch 28/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5386 -
val_loss: 0.5178
Epoch 29/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5351 -
val_loss: 0.5220
Epoch 30/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5353 -
val_loss: 0.5334
Epoch 31/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5326 -
val_loss: 0.5157
```

```
Epoch 32/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5262 -
val_loss: 0.5191
Epoch 33/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5268 -
val_loss: 0.5166
Epoch 34/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5265 -
val_loss: 0.5188
Epoch 35/100
163/163 [==============================] - 2s 10ms/step - loss: 0.5225 -
val_loss: 0.5467
Epoch 36/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5239 -
val_loss: 0.5194
Epoch 37/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5196 -
val_loss: 0.5249
Epoch 38/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5165 -
val_loss: 0.5272
Epoch 39/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5119 -
val_loss: 0.5111
Epoch 40/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5107 -
val_loss: 0.5139
Epoch 41/100
163/163 [==============================] - 2s 10ms/step - loss: 0.5105 -
val_loss: 0.5167
Epoch 42/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5130 -
val_loss: 0.5200
Epoch 43/100
163/163 [==============================] - 2s 10ms/step - loss: 0.5063 -
val_loss: 0.5036
Epoch 44/100
163/163 [==============================] - 2s 9ms/step - loss: 0.5090 -
val_loss: 0.5062
Epoch 45/100
163/163 [==============================] - 2s 10ms/step - loss: 0.5073 -
val_loss: 0.5052
Epoch 46/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4995 -
val_loss: 0.5070
Epoch 47/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4991 -
val_loss: 0.5002
```

```
Epoch 48/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4947 -
val_loss: 0.5184
Epoch 49/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4921 -
val_loss: 0.5108
Epoch 50/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4924 -
val_loss: 0.4884
Epoch 51/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4887 -
val_loss: 0.4932
Epoch 52/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4875 -
val_loss: 0.4911
Epoch 53/100
163/163 [==============================] - 2s 9ms/step - loss: 0.4844 -
val_loss: 0.4874
Epoch 54/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4813 -
val_loss: 0.4889
Epoch 55/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4794 -
val_loss: 0.4921
Epoch 56/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4800 -
val_loss: 0.4922
Epoch 57/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4715 -
val_loss: 0.4906
Epoch 58/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4716 -
val_loss: 0.4887
Epoch 59/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4697 -
val_loss: 0.4952
Epoch 60/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4629 -
val_loss: 0.4847
Epoch 61/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4667 -
val_loss: 0.5008
Epoch 62/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4609 -
val_loss: 0.4781
Epoch 63/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4590 -
val_loss: 0.4877
```

```
Epoch 64/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4525 -
val_loss: 0.4825
Epoch 65/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4507 -
val_loss: 0.4956
Epoch 66/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4485 -
val_loss: 0.4928
Epoch 67/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4490 -
val_loss: 0.4811
Epoch 68/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4438 -
val_loss: 0.4783
Epoch 69/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4381 -
val_loss: 0.4967
Epoch 70/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4392 -
val_loss: 0.4900
Epoch 71/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4356 -
val_loss: 0.4852
Epoch 72/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4321 -
val_loss: 0.5009
Epoch 73/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4298 -
val_loss: 0.4862
Epoch 74/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4248 -
val_loss: 0.4959
Epoch 75/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4240 -
val_loss: 0.4910
Epoch 76/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4173 -
val_loss: 0.4885
Epoch 77/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4157 -
val_loss: 0.4870
Epoch 78/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4143 -
val_loss: 0.4900
Epoch 79/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4092 -
val_loss: 0.4871
```

```
Epoch 80/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4039 -
val_loss: 0.5206
Epoch 81/100
163/163 [==============================] - 2s 10ms/step - loss: 0.4016 -
val_loss: 0.5032
Epoch 82/100
163/163 [==============================] - 2s 9ms/step - loss: 0.3996 -
val_loss: 0.4990
Epoch 83/100
163/163 [==============================] - 2s 9ms/step - loss: 0.3938 -
val_loss: 0.5020
Epoch 84/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3913 -
val_loss: 0.4963
Epoch 85/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3850 -
val_loss: 0.4938
Epoch 86/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3881 -
val_loss: 0.4916
Epoch 87/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3768 -
val_loss: 0.5014
Epoch 88/100
163/163 [==============================] - 2s 9ms/step - loss: 0.3788 -
val_loss: 0.4978
Epoch 89/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3704 -
val_loss: 0.4857
Epoch 90/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3678 -
val_loss: 0.5123
Epoch 91/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3643 -
val_loss: 0.5088
Epoch 92/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3602 -
val_loss: 0.5078
Epoch 93/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3548 -
val_loss: 0.5048
Epoch 94/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3529 -
val_loss: 0.5147
Epoch 95/100
163/163 [==============================] - 2s 9ms/step - loss: 0.3467 -
val_loss: 0.5100
```

```
Epoch 96/100
163/163 [==============================] - 2s 10ms/step - loss: 0.3405 -
val_loss: 0.5173
Epoch 97/100
163/163 [==============================] - 2s 10ms/step - loss: 0.3411 -
val_loss: 0.5060
Epoch 98/100
163/163 [==============================] - 2s 10ms/step - loss: 0.3360 -
val_loss: 0.5060
Epoch 99/100
163/163 [==============================] - 2s 9ms/step - loss: 0.3306 -
val_loss: 0.5179
Epoch 100/100
163/163 [==============================] - 1s 9ms/step - loss: 0.3279 -
val_loss: 0.5073
```

[67]:
```
<keras.callbacks.History at 0x243360ba830>
```

[68]:
```python
loss = model.evaluate(X_test, y_test)
print('Test Loss:', loss)
```

```
41/41 [==============================] - 0s 4ms/step - loss: 0.5073
Test Loss: 0.5073338150978088
```

0.9 Multilayer Perceptron (MLP)

[69]:
```python
from sklearn.neural_network import MLPRegressor
```

[70]:
```python
# split the data into train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
↪random_state=42)
```

[71]:
```python
# standardize the data
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

[72]:
```python
mlp = MLPRegressor(hidden_layer_sizes=(50, 50), activation='relu',␣
↪solver='adam', max_iter=500, random_state=42)
mlp.fit(X_train, y_train)
```

[72]:
```
MLPRegressor(hidden_layer_sizes=(50, 50), max_iter=500, random_state=42)
```

[73]:
```python
# evaluate the model on the test set
y_pred = mlp.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
print("Test MSE: ", mse)
```

```
Test MSE:  0.4930566425509396
```

0.10 Prediction

[74]:
```
input_data = (14,0.5,0.36,6.1,0.071,17.0,102.0,0.9978,3.35,0.8,10.5)
```

[75]:
```
# changing the input data to a numpy array
input_data_as_numpy_array = np.asarray(input_data)
```

[76]:
```
# reshape the data as we are predicting the label for only one instance
input_data_reshaped = input_data_as_numpy_array.reshape(1,-1)
```

[77]:
```
prediction = rf_model.predict(input_data_reshaped)
print(prediction)

if (prediction[0]==1):
  print('Good Quality Wine')
else:
  print('Bad Quality Wine')
```

```
[0]
Bad Quality Wine
```