red_wine

May 12, 2023

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

[2]:
```python
wine = pd.read_csv('winequality-red.csv')
```

0.1 Understanding the structure of the dataset

[3]:
```python
wine.head()
```

[3]:
```
   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
```

[4]:
```python
wine.tail()
```

[4]:
```
      fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
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  \
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
1594     10.5        5
1595     11.2        6
1596     11.0        6
1597     10.2        5
1598     11.0        6
```

[5]:
```
wine.shape
```

[5]:
```
(1599, 12)
```

[6]:
```
wine.columns
```

[6]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

[7]:
```
wine.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   float64
 2   citric acid           1599 non-null   float64
 3   residual sugar        1599 non-null   float64
 4   chlorides             1599 non-null   float64
 5   free sulfur dioxide   1599 non-null   float64
 6   total sulfur dioxide  1599 non-null   float64
 7   density               1599 non-null   float64
 8   pH                    1599 non-null   float64
 9   sulphates             1599 non-null   float64
 10  alcohol               1599 non-null   float64
 11  quality               1599 non-null   int64
dtypes: float64(11), int64(1)
```

```
memory usage: 150.0 KB
```

[8]:
```
wine.isnull().sum()
```

[8]:
```
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
```

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

[9]:
```
       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
```

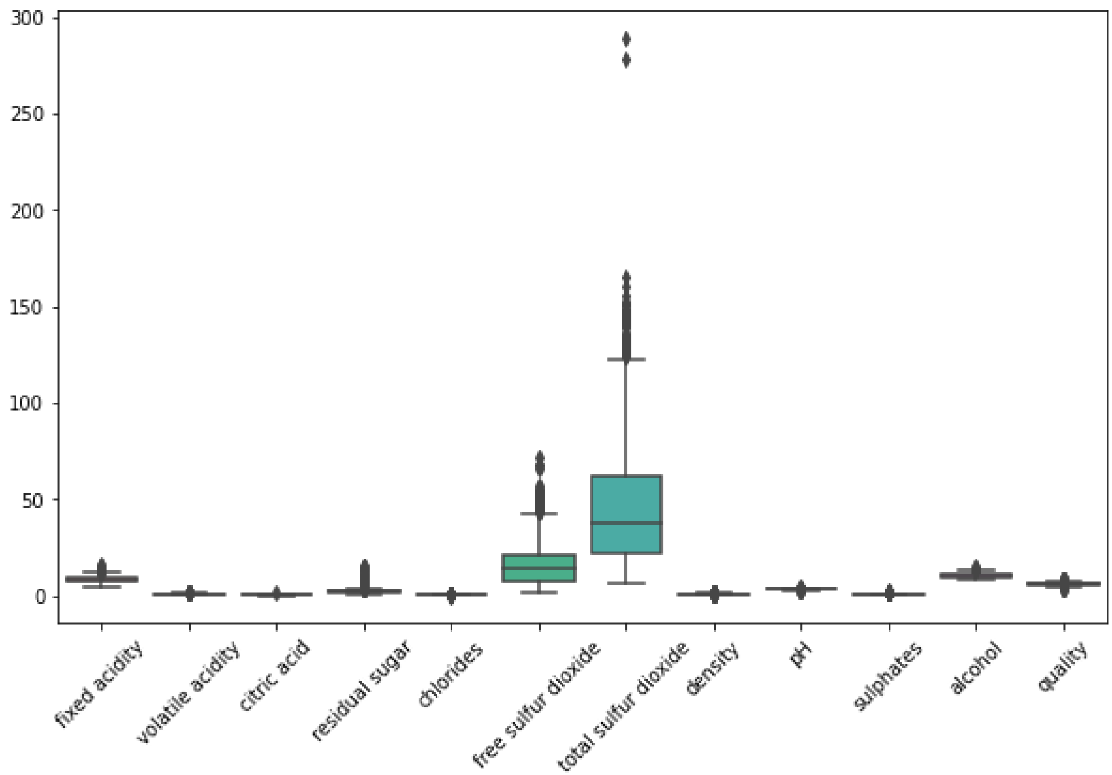
```
max        4.010000      2.000000     14.900000      8.000000
```

0.2 Outlier Detection

0.2.1 1. Boxplot

[10]:
```
import seaborn as sns

plt.subplots(figsize=(10, 6))
plt.xticks(rotation=45)
sns.boxplot(data=wine)
```

[10]: <AxesSubplot:>

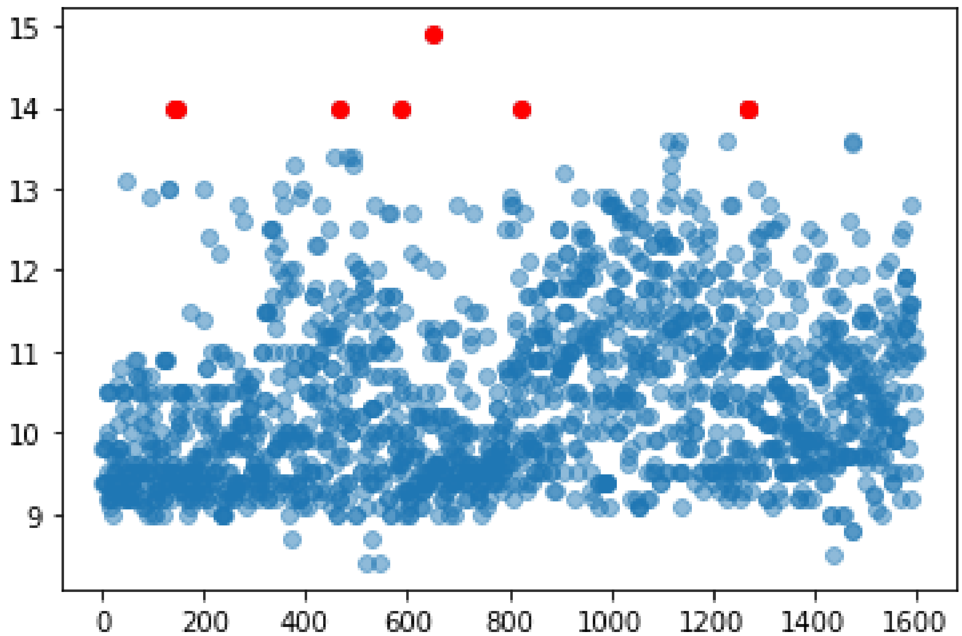


0.3 2. Zscore

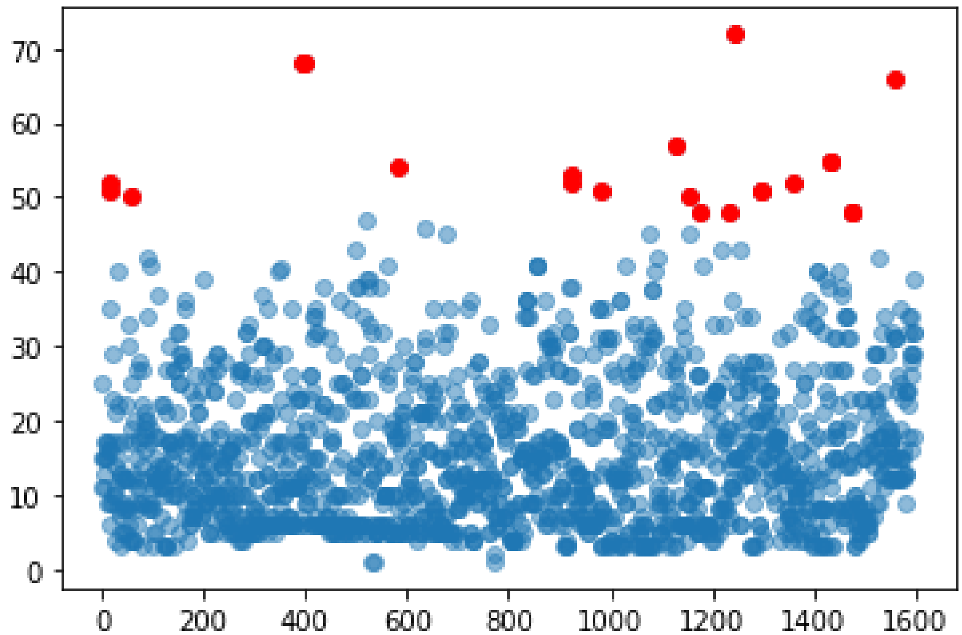
[11]:
```
# In this code, the zscore() function is used to calculate the Z-scores for␣
↪each data point in the "alcohol" column.
# The threshold variable is set to 3, which means that any data point with a␣
↪Z-score greater than 3 or less than -3 is considered an outlier.
from scipy import stats
```

```python
z_scores = stats.zscore(wine['alcohol'])
threshold = 3

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

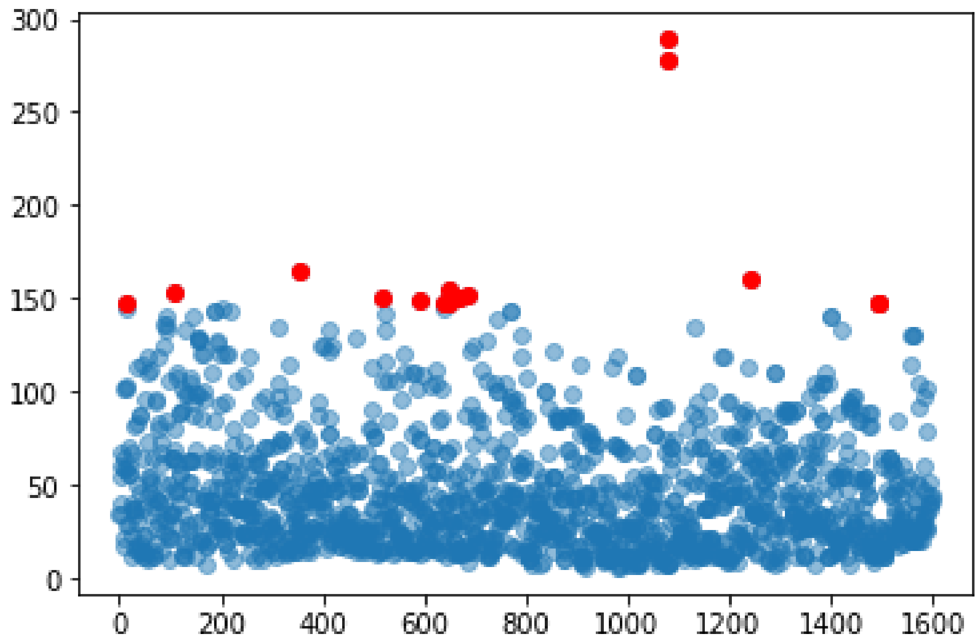
[12]:
```python
import matplotlib.pyplot as plt

plt.scatter(wine.index, wine['alcohol'], alpha=0.5)
plt.scatter(outliers.index, outliers['alcohol'], color='r')
plt.show()
```



[13]:
```python
z_scores = stats.zscore(wine['free sulfur dioxide'])
threshold = 3

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

plt.scatter(wine.index, wine['free sulfur dioxide'], alpha=0.5)
plt.scatter(outliers.index, outliers['free sulfur dioxide'], color='r')
plt.show()
```

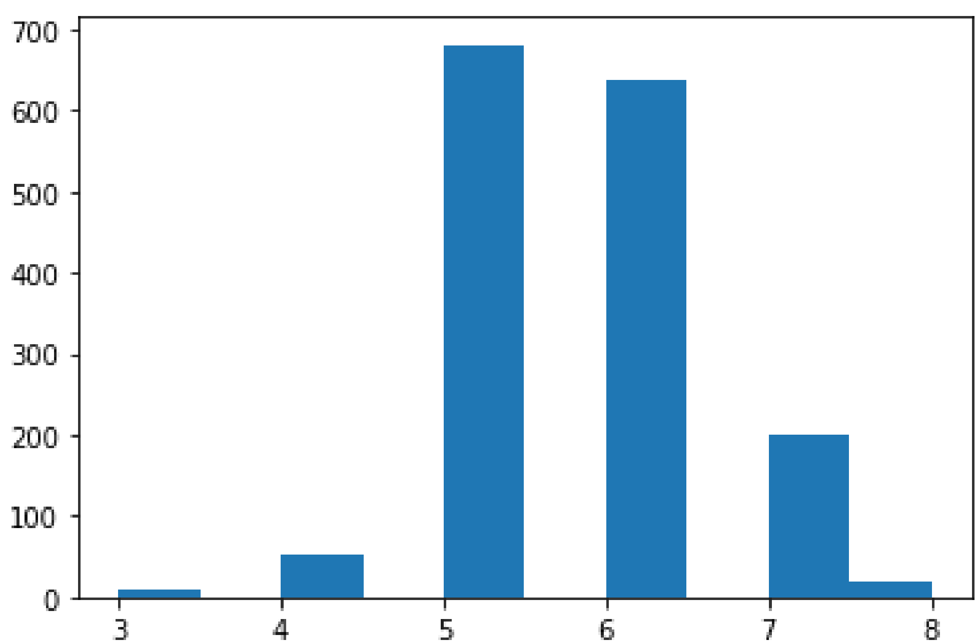


[14]:
```
z_scores = stats.zscore(wine['total sulfur dioxide'])
threshold = 3

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

plt.scatter(wine.index, wine['total sulfur dioxide'], alpha=0.5)
plt.scatter(outliers.index, outliers['total sulfur dioxide'], color='r')
plt.show()
```



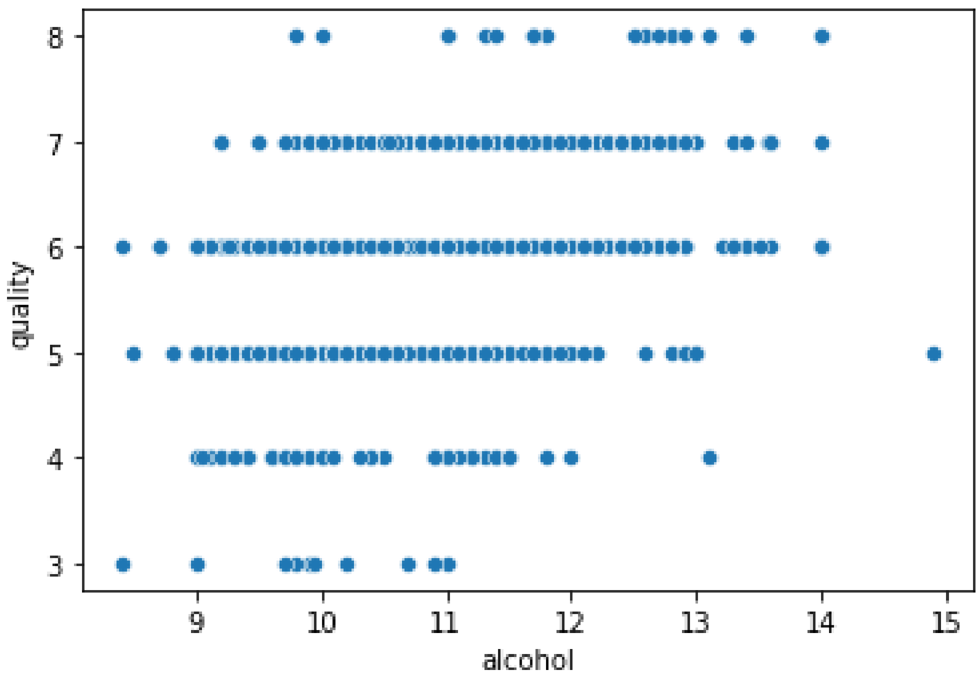
0.4 Data Visualization

[15]:
```
plt.hist(wine['quality'])
plt.show()
```



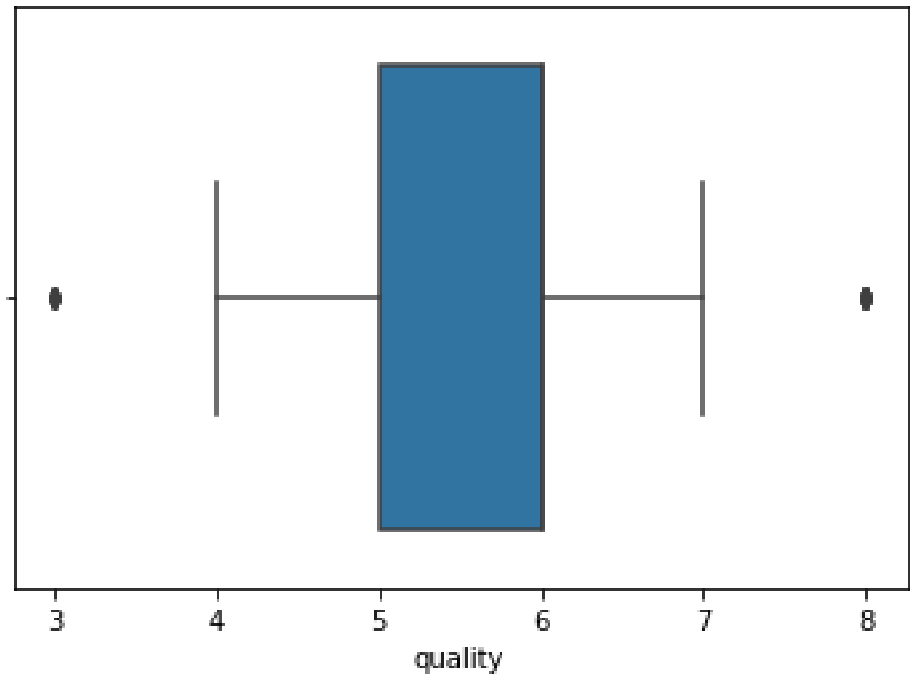
[16]:
```python
import seaborn as sns

sns.scatterplot(x='alcohol', y='quality', data=wine)
```

[16]: <AxesSubplot:xlabel='alcohol', ylabel='quality'>



[17]:
```python
sns.boxplot(x='quality', data=wine)
```

[17]: <AxesSubplot:xlabel='quality'>



0.4.1 Splitting the data and creating regression model

[18]:
```
from sklearn.model_selection import train_test_split
```

[19]:
```
X = wine.drop('quality', axis=1) # Extract the input features
y = wine['quality'] # Extract the target variable
```

[20]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
    ↪random_state=42)
```

1 Regression

1.0.1 1. Regression model

[21]:
```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

[22]:
```
# Create a linear regression model
lr_model = LinearRegression()
```

[23]:
```
# Fit the model on the training set
lr_model.fit(X_train, y_train)
```

[23]:
```
LinearRegression()
```

[24]:

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

[25]:

```
# 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)
```

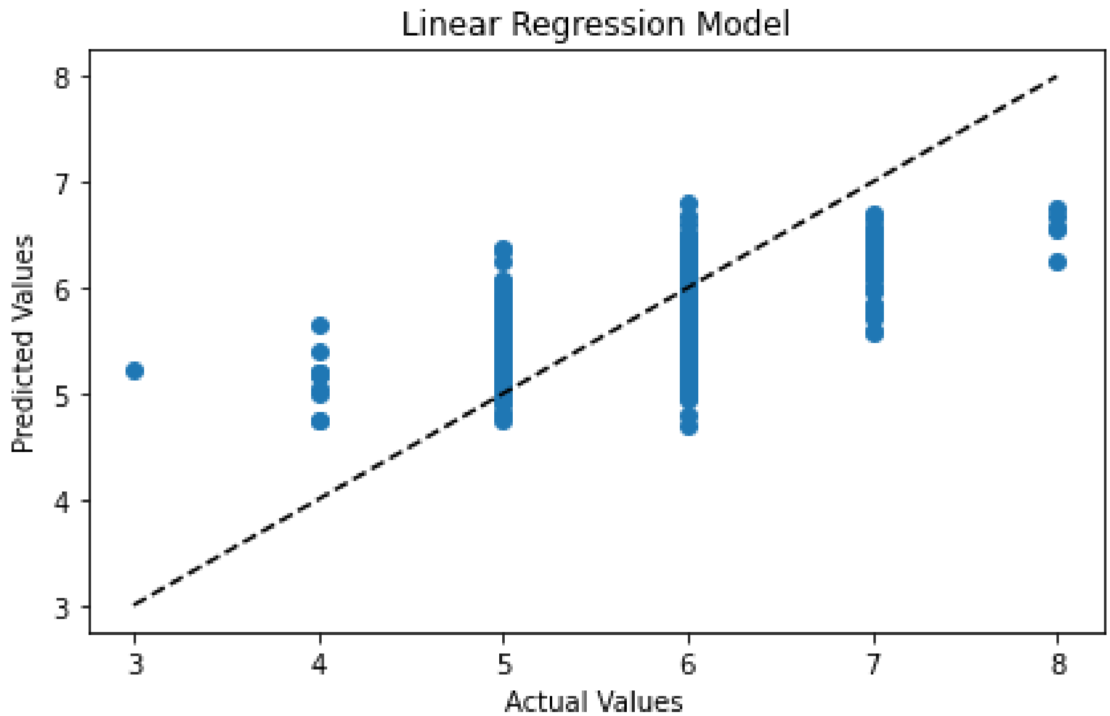
[26]:

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

```
RMSE: 0.6245199307983028
MAE: 0.5035304415524661
MSE: 0.390025143964317
R-squared Score: 0.4031803412790679
```

[28]:

```
# Plot the predicted values against the actual values
plt.scatter(y_test, y_pred)
plt.xlabel('Actual Values')
plt.ylabel('Predicted Values')
plt.title('Linear Regression Model')
# Plot the regression line
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--k')
plt.tight_layout()
plt.show()
```



1.1 2. XGboost

[29]:
```python
from xgboost import XGBRegressor
```

[30]:
```python
# Create an XGBoost model
xgb_model = XGBRegressor(objective='reg:squarederror', random_state=42)
```

[31]:
```python
# Fit the model on the training set
xgb_model.fit(X_train, y_train)
```

[31]:
```
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, …)
```

[32]:
```python
# Make predictions on the testing set
y_pred = xgb_model.predict(X_test)
```

[33]:
```python
# 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)
```

[34]:
```python
print('RMSE:', rmse)
print('MAE:', mae)
print('MSE:', mse)
print('R-squared Score:', r2)
```

```
RMSE: 0.5773221494298576
MAE: 0.40534366890788076
MSE: 0.3333008642223108
R-squared Score: 0.489980297129894
```

1.2 3. Artificial Nural Network

[35]:
```python
import keras
from keras.models import Sequential
from keras.layers import Dense
```

[36]:
```python
# 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))
```

[37]:
```python
# Compile the model
ann_model.compile(loss='mean_squared_error', optimizer='adam')
```

[38]:
```python
# Fit the model on the training set
history = ann_model.fit(X_train, y_train, validation_split=0.2, epochs=50,␣
↪batch_size=32)
```

```
Epoch 1/50
32/32 [==============================] - 1s 8ms/step - loss: 10.0693 - val_loss:
1.1350
Epoch 2/50
32/32 [==============================] - 0s 3ms/step - loss: 0.7753 - val_loss:
0.5668
Epoch 3/50
32/32 [==============================] - 0s 3ms/step - loss: 0.6288 - val_loss:
0.4657
```

```
Epoch 4/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5839 - val_loss:
0.4896
Epoch 5/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5741 - val_loss:
0.4186
Epoch 6/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5662 - val_loss:
0.5640
Epoch 7/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5595 - val_loss:
0.4035
Epoch 8/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5326 - val_loss:
0.4058
Epoch 9/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5734 - val_loss:
0.3937
Epoch 10/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5470 - val_loss:
0.4066
Epoch 11/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5166 - val_loss:
0.3812
Epoch 12/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5999 - val_loss:
0.4069
Epoch 13/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5079 - val_loss:
0.3899
Epoch 14/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5338 - val_loss:
0.3903
Epoch 15/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5049 - val_loss:
0.3846
Epoch 16/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5169 - val_loss:
0.3677
Epoch 17/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5140 - val_loss:
0.4072
Epoch 18/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5359 - val_loss:
0.6050
Epoch 19/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5731 - val_loss:
0.3807
```

```
Epoch 20/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5071 - val_loss:
0.3614
Epoch 21/50
32/32 [==============================] - 0s 2ms/step - loss: 0.4984 - val_loss:
0.4444
Epoch 22/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4912 - val_loss:
0.3637
Epoch 23/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4851 - val_loss:
0.3714
Epoch 24/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4830 - val_loss:
0.3540
Epoch 25/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4951 - val_loss:
0.3838
Epoch 26/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4892 - val_loss:
0.3570
Epoch 27/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4917 - val_loss:
0.3646
Epoch 28/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4940 - val_loss:
0.3626
Epoch 29/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5098 - val_loss:
0.3573
Epoch 30/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5062 - val_loss:
0.3727
Epoch 31/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4939 - val_loss:
0.3534
Epoch 32/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4886 - val_loss:
0.3438
Epoch 33/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5147 - val_loss:
0.3800
Epoch 34/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5011 - val_loss:
0.3563
Epoch 35/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4911 - val_loss:
0.3519
```

```
Epoch 36/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5150 - val_loss:
0.3504
Epoch 37/50
32/32 [==============================] - 0s 2ms/step - loss: 0.4909 - val_loss:
0.3894
Epoch 38/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4982 - val_loss:
0.6311
Epoch 39/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5332 - val_loss:
0.3906
Epoch 40/50
32/32 [==============================] - 0s 2ms/step - loss: 0.5310 - val_loss:
0.4927
Epoch 41/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4895 - val_loss:
0.3380
Epoch 42/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4864 - val_loss:
0.3544
Epoch 43/50
32/32 [==============================] - 0s 2ms/step - loss: 0.4720 - val_loss:
0.4569
Epoch 44/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5148 - val_loss:
0.4784
Epoch 45/50
32/32 [==============================] - 0s 3ms/step - loss: 0.5572 - val_loss:
0.3490
Epoch 46/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4796 - val_loss:
0.3966
Epoch 47/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4920 - val_loss:
0.4803
Epoch 48/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4863 - val_loss:
0.4108
Epoch 49/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4771 - val_loss:
0.3883
Epoch 50/50
32/32 [==============================] - 0s 3ms/step - loss: 0.4494 - val_loss:
0.3348
```

[39]:
```
# Make predictions on the testing set
y_pred = ann_model.predict(X_test)
```

```
10/10 [==============================] - 0s 2ms/step
```

[40]:
```
# 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)
```

[41]:
```
print('RMSE:', rmse)
print('MAE:', mae)
print('MSE:', mse)
print('R-squared Score:', r2)
```

```
RMSE: 0.63806854902311
MAE: 0.5153964817523956
MSE: 0.407131473252457
R-squared Score: 0.37700409657867584
```

2 Classification

[42]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score
```

2.1 1. Logistic Regression model

[43]:
```
# Create and train the logistic regression model
lr_model = LogisticRegression()
lr_model.fit(X_train, y_train)
```

```
C:\Users\Sushan Shivagiri\AppData\Local\Programs\Python\Python310\lib\site-
packages\sklearn\linear_model\_logistic.py:444: 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(
```

[43]: LogisticRegression()

[44]:
```
# Evaluate the logistic regression model
lr_pred = lr_model.predict(X_test)
lr_acc = accuracy_score(y_test, lr_pred)
print("Logistic Regression accuracy: ", lr_acc)
```

```
Logistic Regression accuracy:  0.5625
```

2.2 2. Random forest model

[45]:
```
# Create and train the random forest model
rf_model = RandomForestClassifier(random_state=42)
rf_model.fit(X_train, y_train)
```

[45]: RandomForestClassifier(random_state=42)

[46]:
```
# 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:  0.659375
```

[47]:
```
from sklearn.metrics import confusion_matrix, plot_roc_curve



y_true = y_test
y_pred = lr_model.predict(X_test)

conf_mat = confusion_matrix(y_true, y_pred)
print(conf_mat)
```

```
[[ 0  0  1  0  0  0]
 [ 0  0  9  1  0  0]
 [ 0  0 97 33  0  0]
 [ 0  0 48 82  2  0]
 [ 0  0  4 37  1  0]
 [ 0  0  0  4  1  0]]
```