Wine_Quality_Prediction_ (1)

December 4, 2023

Wine Quality Prediction By Mehedi Hasan Nahid

In this project, I utilized the Red Wine Quality dataset to develop classification models. These models aim to predict if a specific red wine falls under the category of “good quality”. The dataset rates each wine with a “quality” score ranging from 0 to 10. To align with the project’s objectives, I transformed the quality scores into a binary format. Wines with a score of 7 or above are classified as “good quality,” while those with a score below 7 are not. The determination of a wine’s quality is based on 11 input variables.

Project Documentation: Impact of Data Preparation on Classification steps

1:Baseline Model Building

Goal: Create a baseline Logistic Regression model using the wine dataset. Target: ‘Quality’ score.

Model: Logistic Regression with all 11 predictors.

Metric: Model accuracy (Bottom-line accuracy).

2:Predictor Transformation

Task: Modify one predictor at a time and rebuild the model.

Action: Choose a predictor, apply a transformation, and create a new model.

Frequency: Repeat 5 times, with different or same predictors (if applying a new transformation).

3:Feature Selection

Task: Drop 1-3 predictors and build a new model.

Action: Select a subset of predictors, remove them, and construct a new model.

Frequency: Repeat 5 times with varied subsets.

4:Comparative Analysis

Goal: Assess the accuracy of models from steps 2 and 3 against the baseline.

Deadline: 10 new models and accuracies to compare.

5:Concluding Observations

Task: Write observations comparing new model accuracies with the baseline.

Content: Discuss effectiveness and patterns in data preparation.

#Constructing an initial Classification model/Baseline Model Building and preprocessing our data

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

     from warnings import filterwarnings
     filterwarnings(action='ignore')

     #Loading the dataset and getting some information
     csv_file_path = '/content/wine-data-set .csv'
     wine_data = pd.read_csv(csv_file_path)
     print(wine_data.head())
     print("Wine data Shape",wine_data.shape)
```

```
   fixed acidity  volatile acidity  citric acid  residual sugar  chlorides  \
0            7.0              0.27         0.36            20.7      0.045
1            6.3              0.30         0.34             1.6      0.049
2            8.1              0.28         0.40             6.9      0.050
3            7.2              0.23         0.32             8.5      0.058
4            7.2              0.23         0.32             8.5      0.058

   free sulfur dioxide  total sulfur dioxide  density    pH  sulphates  \
0                 45.0                 170.0   1.0010  3.00       0.45
1                 14.0                 132.0   0.9940  3.30       0.49
2                 30.0                  97.0   0.9951  3.26       0.44
3                 47.0                 186.0   0.9956  3.19       0.40
4                 47.0                 186.0   0.9956  3.19       0.40

   alcohol  quality
0      8.8        6
1      9.5        6
2     10.1        6
3      9.9        6
4      9.9        6
Wine data Shape (6463, 12)
```

```
[ ]: #Description of wine data
     wine_data.describe(include='all')
```

```
[ ]:        fixed acidity  volatile acidity  citric acid  residual sugar  \
     count    6463.000000       6463.000000  6463.000000     6463.000000
     mean        7.217755          0.339589     0.318758        5.443958
     std         1.297913          0.164639     0.145252        4.756852
     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  6463.000000          6463.000000           6463.000000  6463.000000
mean      0.056056            30.516865            115.694492     0.994698
std       0.035076            17.758815             56.526736     0.003001
min       0.009000             1.000000              6.000000     0.987110
25%       0.038000            17.000000             77.000000     0.992330
50%       0.047000            29.000000            118.000000     0.994890
75%       0.065000            41.000000            156.000000     0.997000
max       0.611000           289.000000            440.000000     1.038980

                pH    sulphates      alcohol      quality
count  6463.000000  6463.000000  6463.000000  6463.000000
mean      3.218332     0.531150    10.492825     5.818505
std       0.160650     0.148913     1.193128     0.873286
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
```

[ ]:
```
# wine data info
wine_data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 6463 entries, 0 to 6462
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   fixed acidity         6463 non-null   float64
 1   volatile acidity      6463 non-null   float64
 2   citric acid           6463 non-null   float64
 3   residual sugar        6463 non-null   float64
 4   chlorides             6463 non-null   float64
 5   free sulfur dioxide   6463 non-null   float64
 6   total sulfur dioxide  6463 non-null   float64
 7   density               6463 non-null   float64
 8   pH                    6463 non-null   float64
 9   sulphates             6463 non-null   float64
 10  alcohol               6463 non-null   float64
 11  quality               6463 non-null   int64
dtypes: float64(11), int64(1)
memory usage: 606.0 KB
```

[ ]:
```
#Preprocessing the dataset
#checking for null value
```

```
print(wine_data.isna().sum())
```

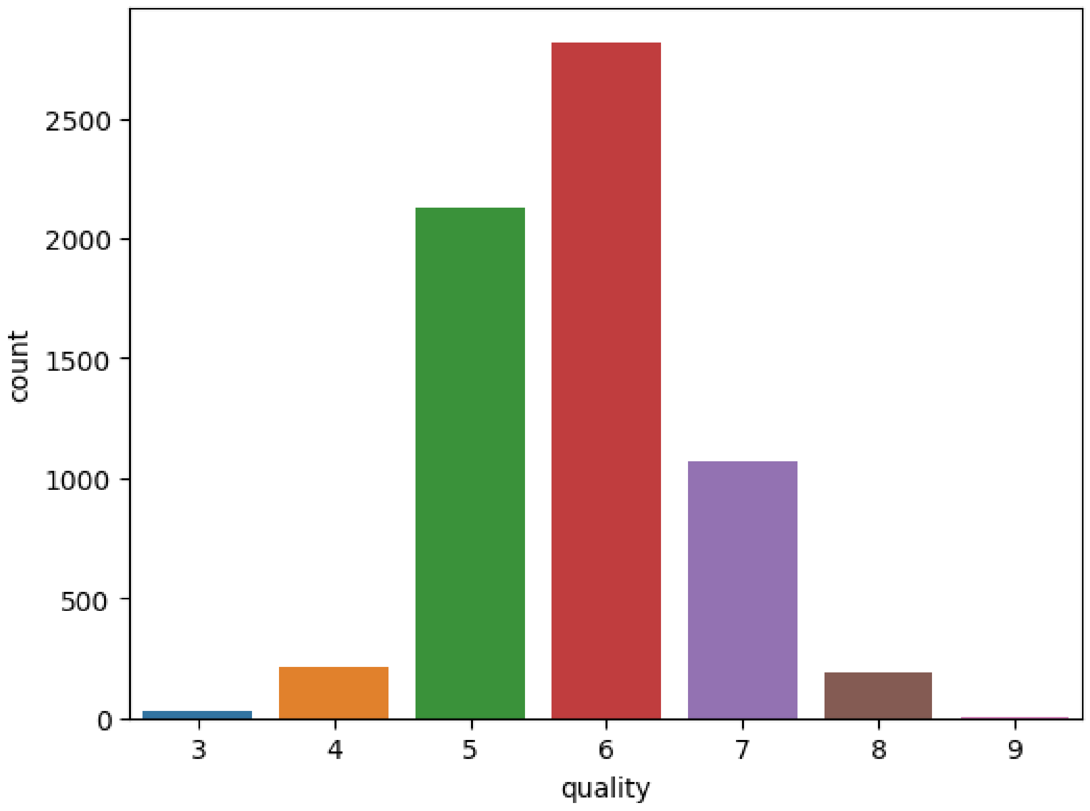
```
fixed acidity           0
volatile acidity        0
citric acid             0
residual sugar          0
chlorides               0
free sulfur dioxide     0
total sulfur dioxide    0
density                 0
pH                      0
sulphates               0
alcohol                 0
quality                 0
dtype: int64
```

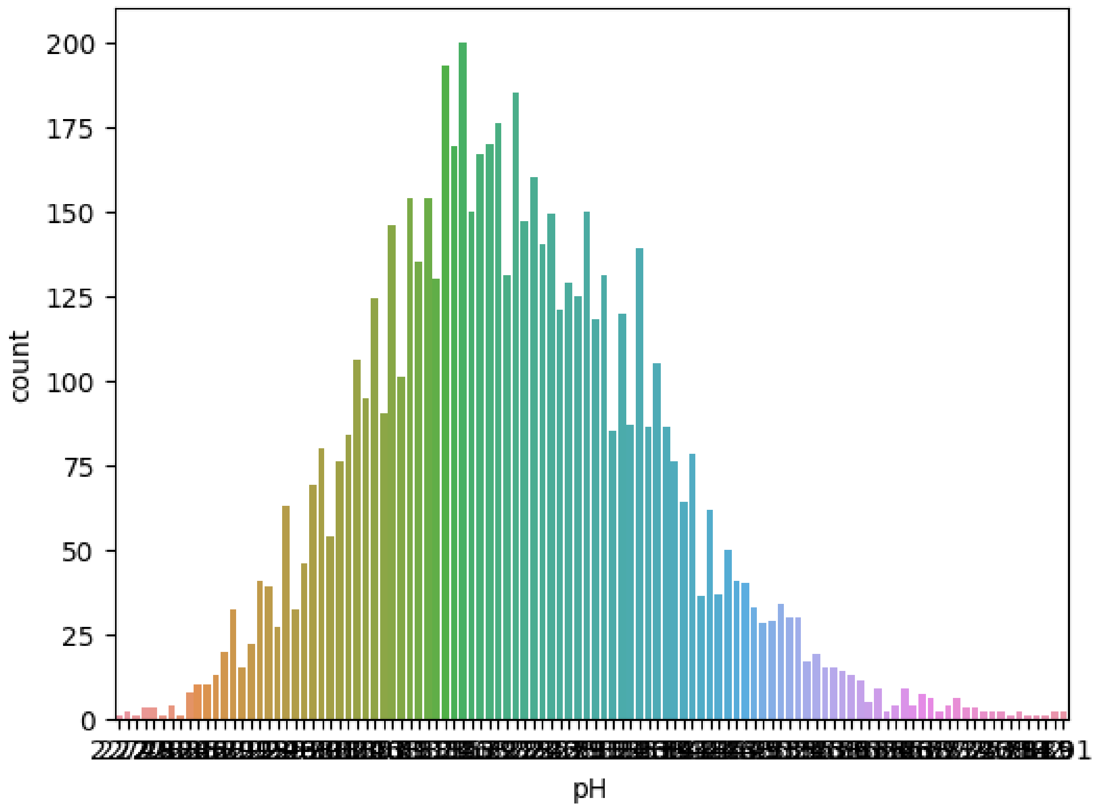
As we do not have null values in our data set we do not need to fill our missing values

1 Data Analysis

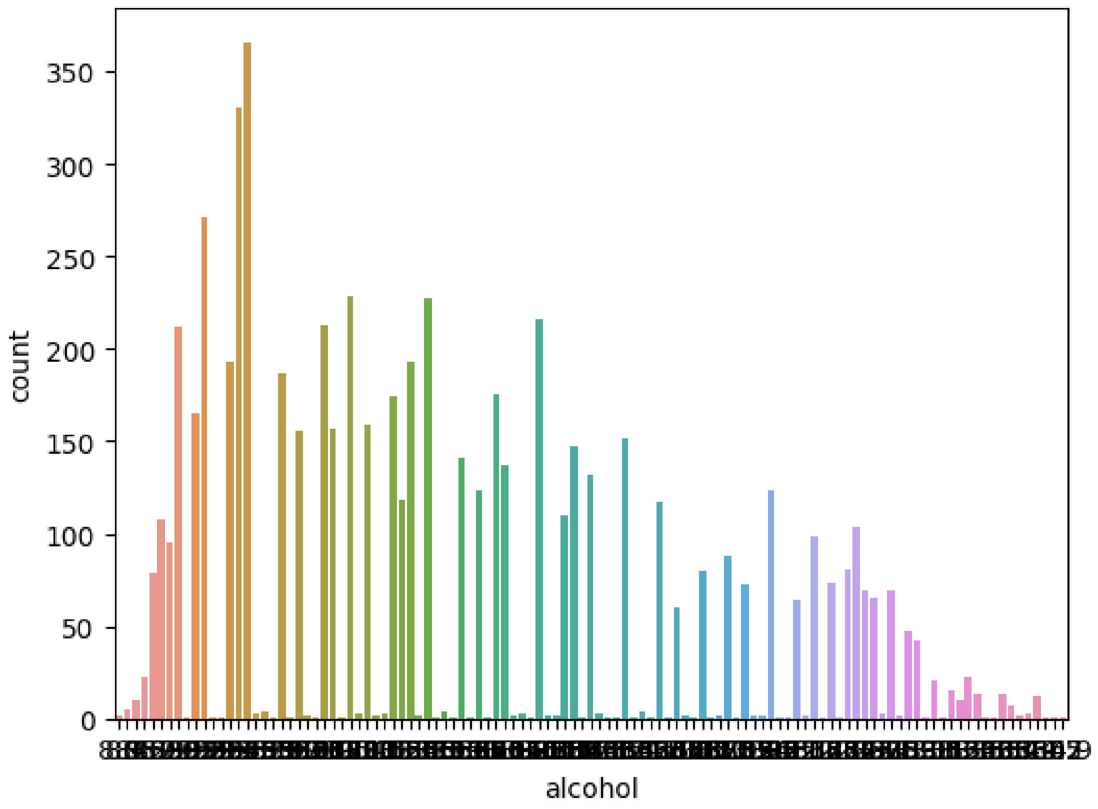
[ ]:
```
#Countplot:
import seaborn as sns
import matplotlib.pyplot as plt

# Creating a count plot for the 'quality' and other value column in the wine␣
  ↪dataset
sns.countplot(x=wine_data['quality'])

# Displaying the plot
plt.show()

sns.countplot(x = wine_data['pH'])
plt.show()

sns.countplot(x=wine_data['alcohol'])
plt.show()
sns.countplot(x = wine_data['fixed acidity'])
plt.show()

sns.countplot(x = wine_data['volatile acidity'])
plt.show()

sns.countplot(x = wine_data['density'])
plt.show()

sns.countplot(x = wine_data['chlorides'])
```

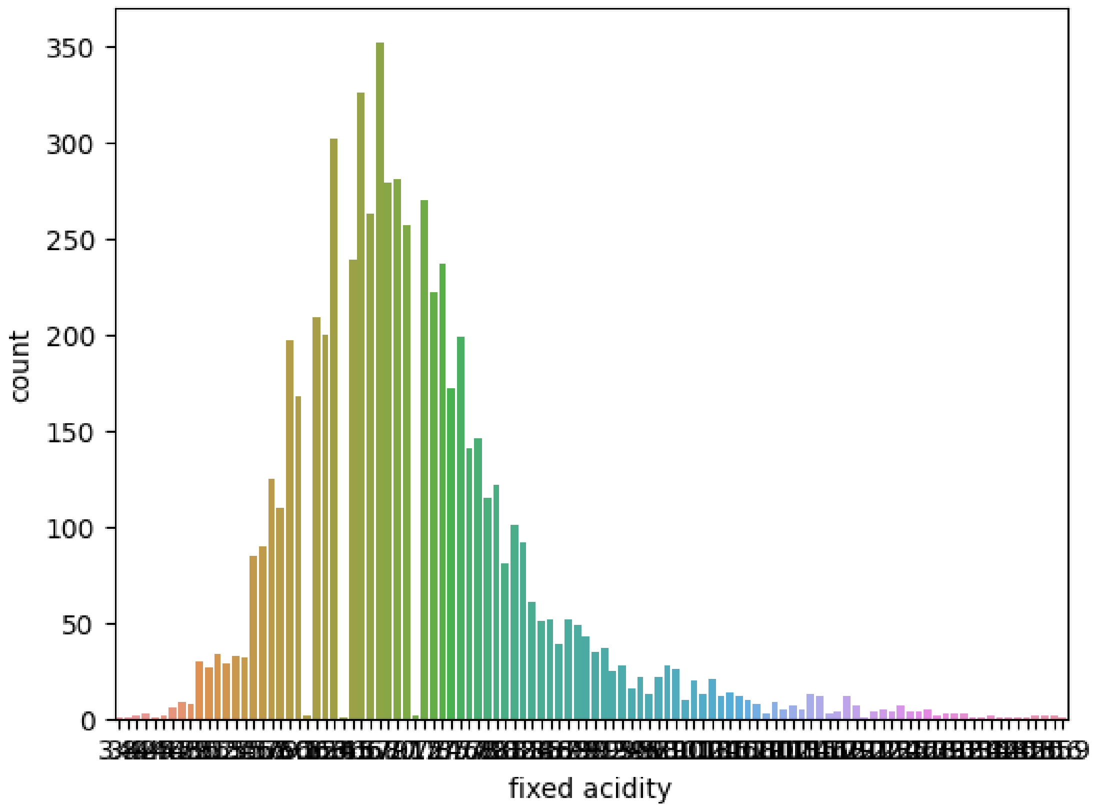
```
plt.show()
```

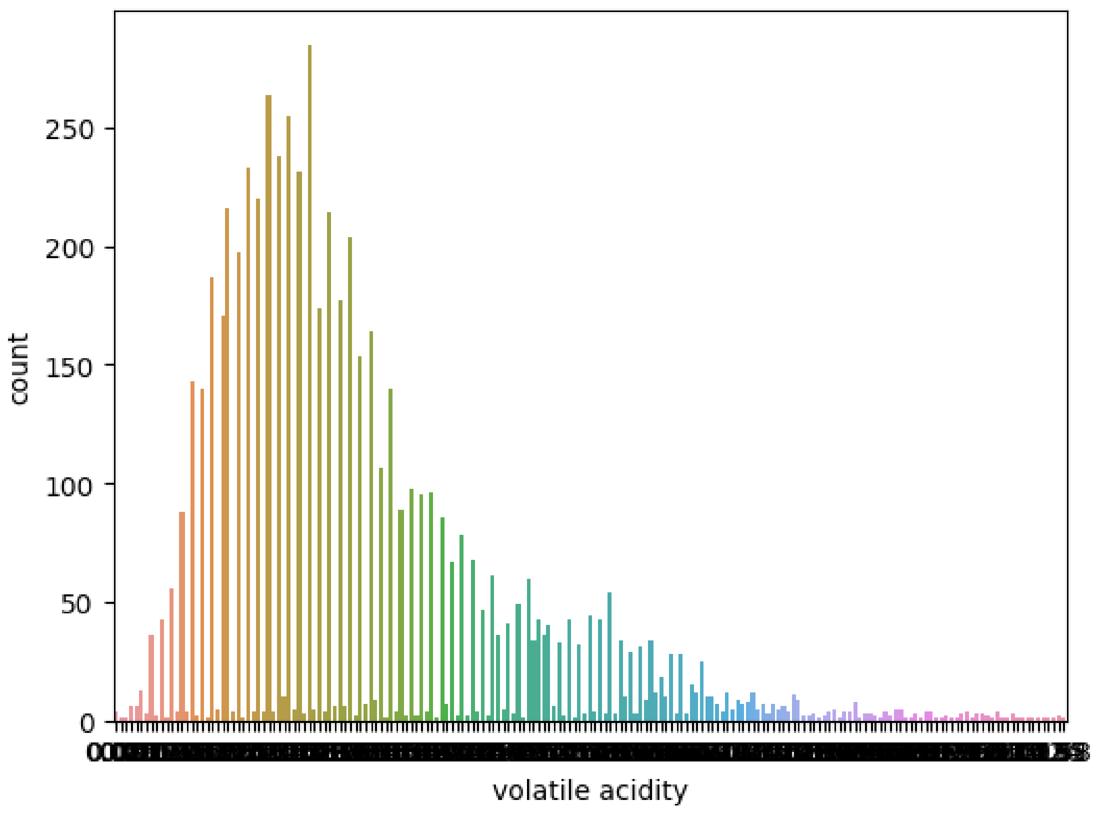




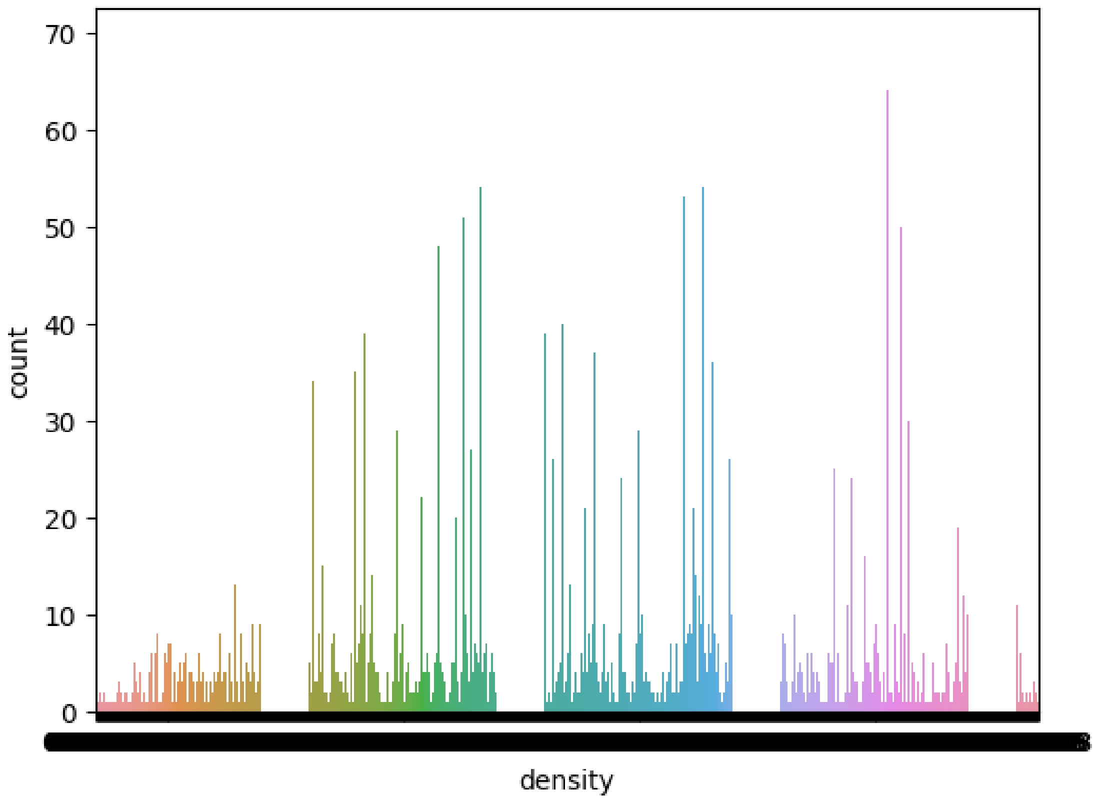
count
pH
0
25
50
75
100
125
150
175
200


count
alcohol
0
50
100
150
200
250
300
350

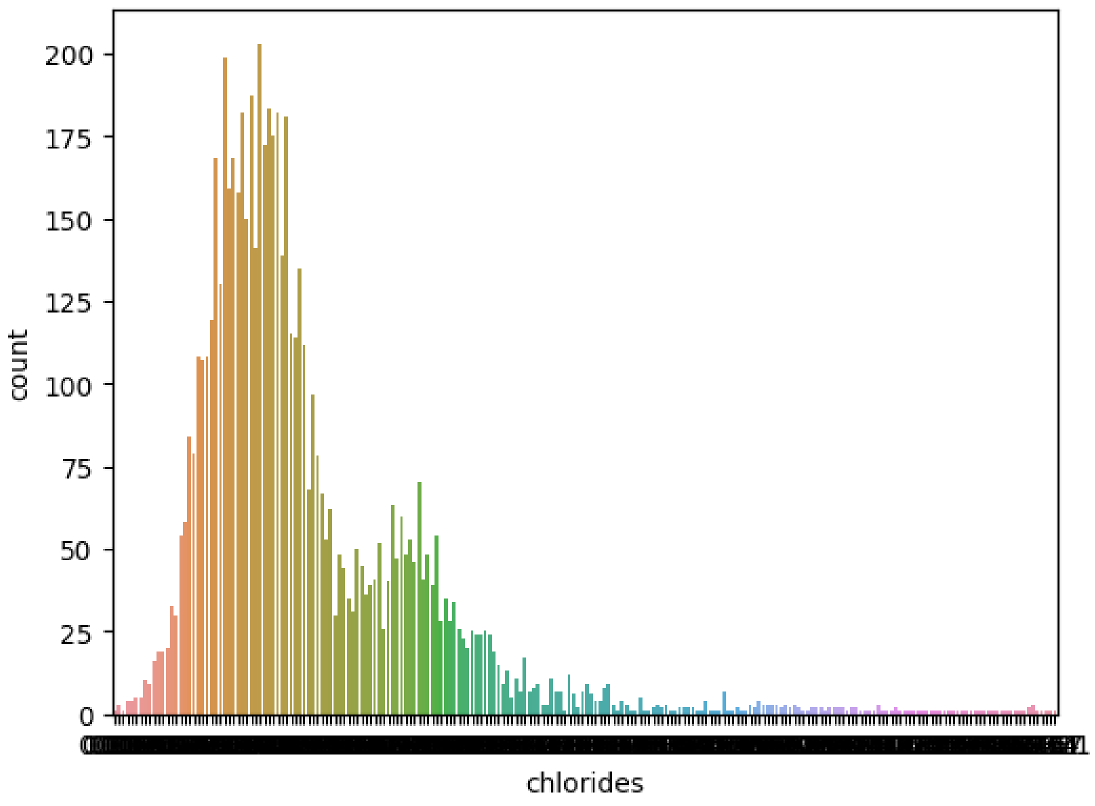

count
fixed acidity
0
50
100
150
200
250
300
350



count
250
200
150
100
50
0
volatile acidity

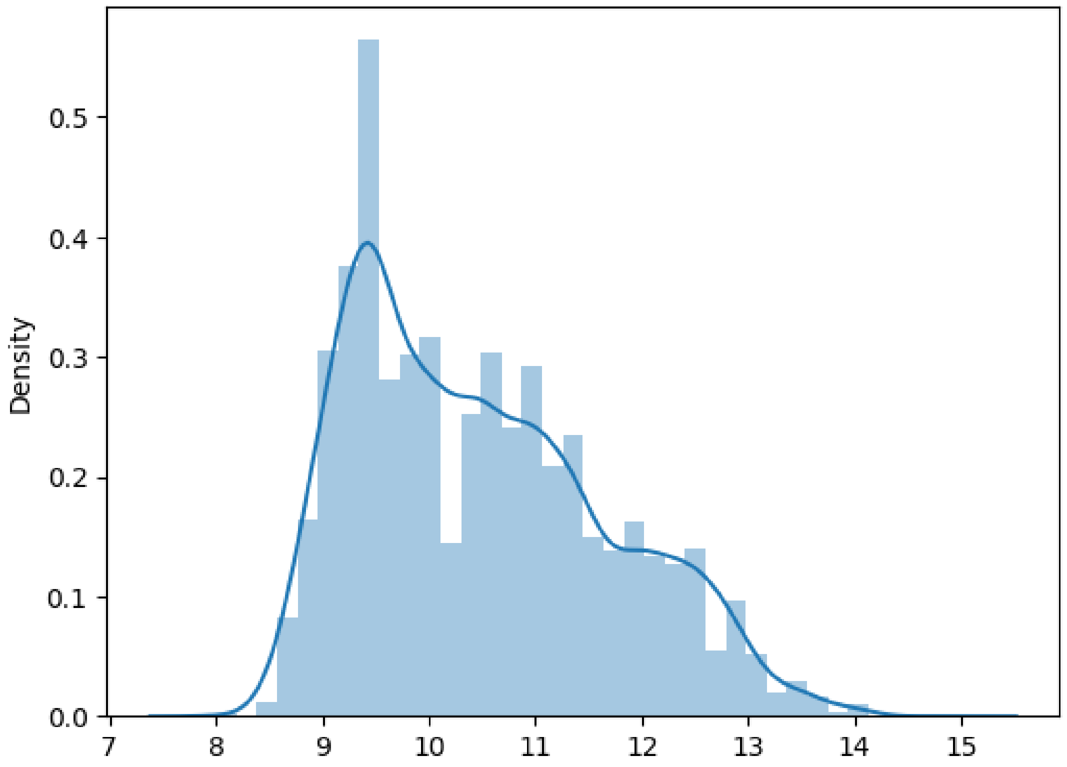


count
density
70
60
50
40
30
20
10
0



```
[ ]: #Distplot
     import seaborn as sns
     import matplotlib.pyplot as plt

     sns.distplot(x = wine_data['alcohol'])
```

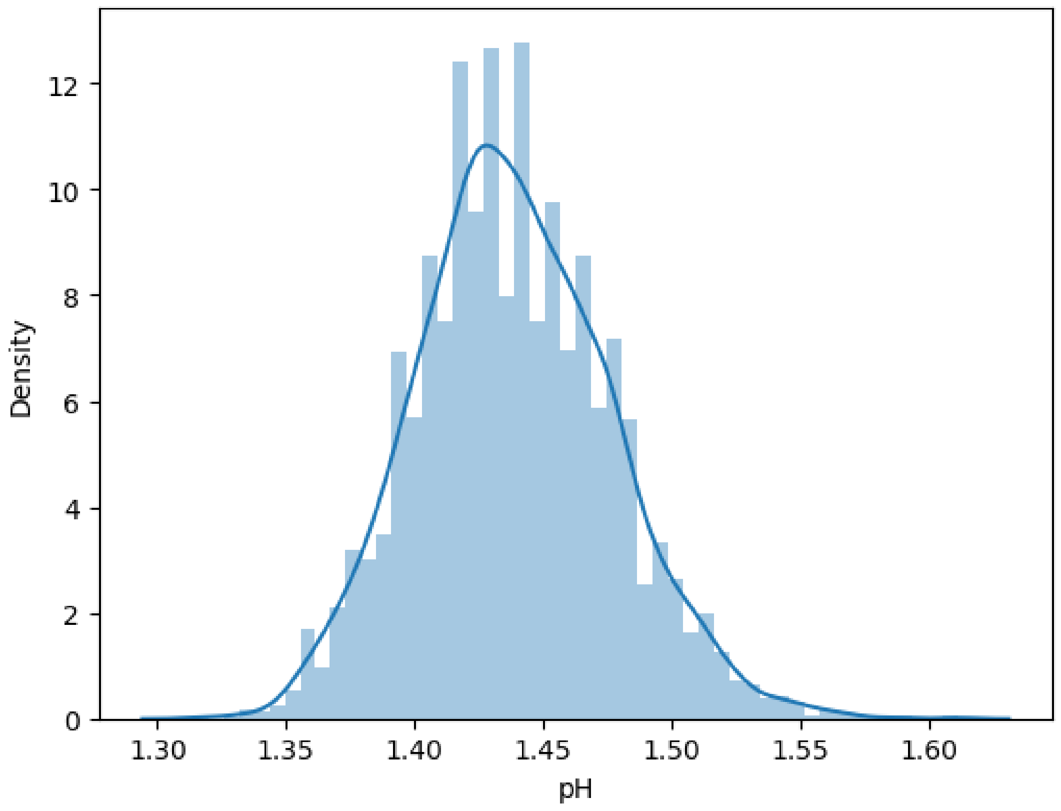
```
[ ]: <Axes: ylabel='Density'>
```



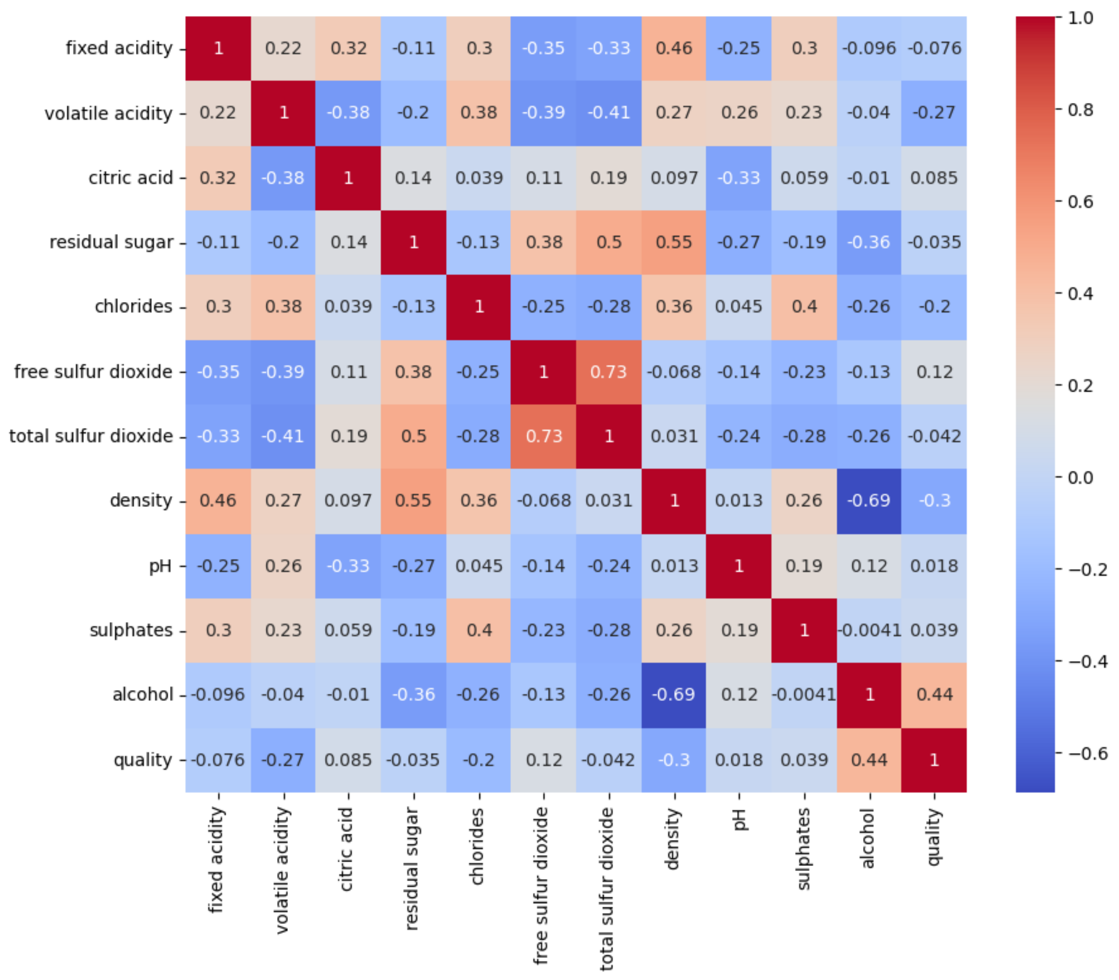
Log transformation

Log transformation helps to make the highly skewed distribution to less skewed.

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



     wine_data['pH'] = np.log(1 + wine_data['pH'])

     # Creating a distribution plot for the transformed 'pH' data
     sns.distplot(wine_data['pH'])

     # Displaying the plot
     plt.show()
```

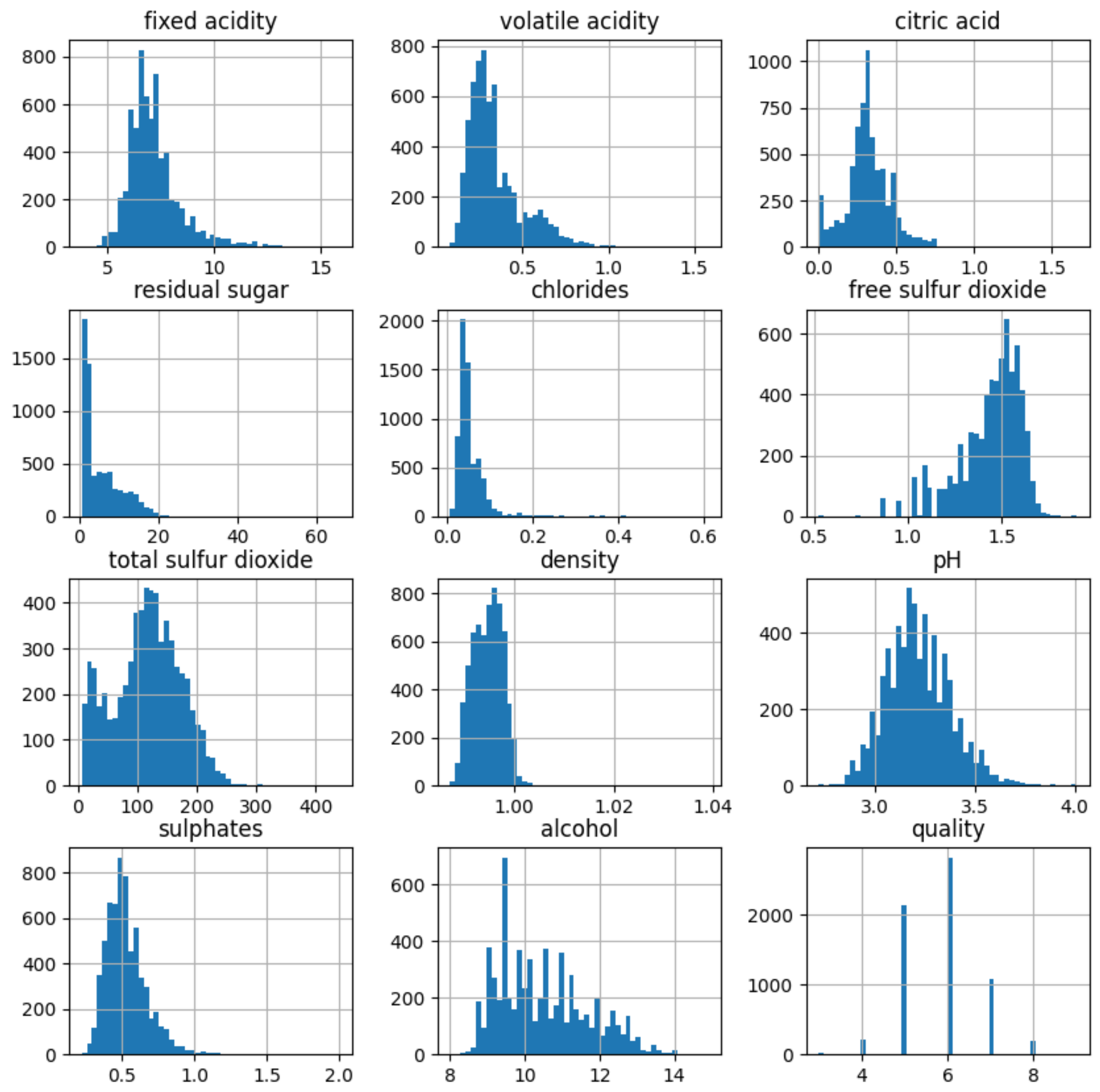


```
[ ]: #Heatmap for expressing correlation

     import pandas as pd
     import seaborn as sns
     import matplotlib.pyplot as plt
     correlation_matrix = wine_data.corr()
     # Plotting the correlation heatmap
     plt.figure(figsize=(10, 8))
     sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
     plt.show()
```



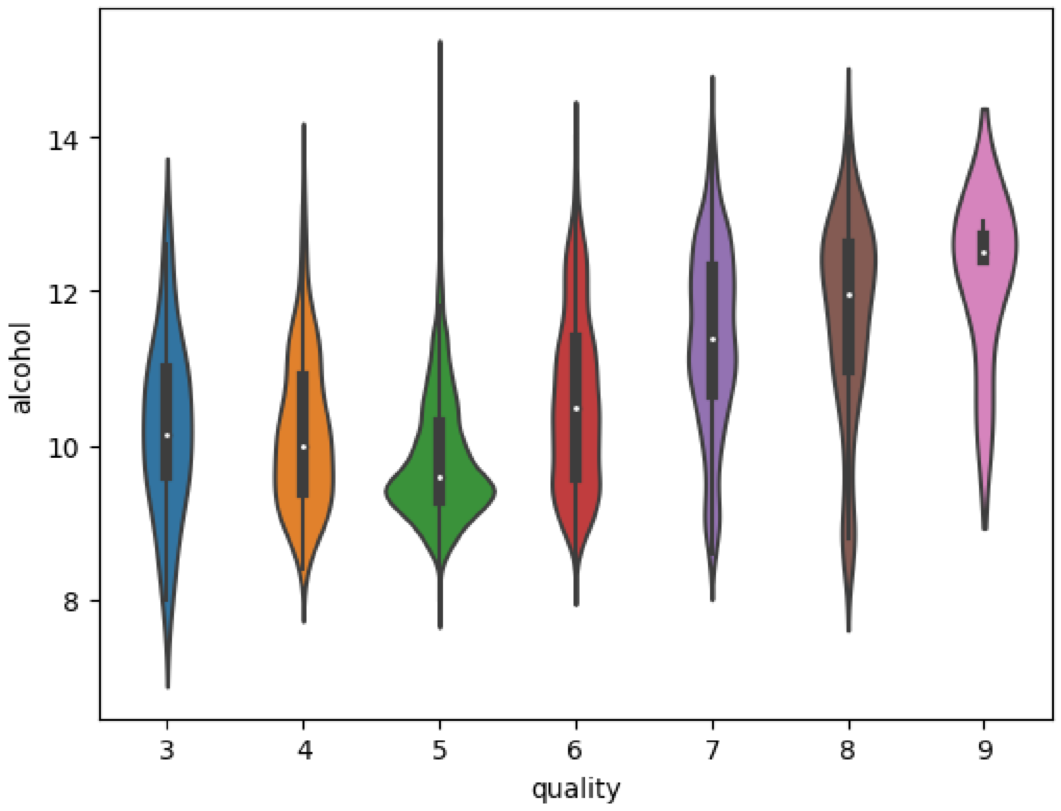
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1	0.22	0.32	-0.11	0.3	-0.35	-0.33	0.46	-0.25	0.3	-0.096	-0.076
volatile acidity	0.22	1	-0.38	-0.2	0.38	-0.39	-0.41	0.27	0.26	0.23	-0.04	-0.27
citric acid	0.32	-0.38	1	0.14	0.039	0.11	0.19	0.097	-0.33	0.059	-0.01	0.085
residual sugar	-0.11	-0.2	0.14	1	-0.13	0.38	0.5	0.55	-0.27	-0.19	-0.36	-0.035
chlorides	0.3	0.38	0.039	-0.13	1	-0.25	-0.28	0.36	0.045	0.4	-0.26	-0.2
free sulfur dioxide	-0.35	-0.39	0.11	0.38	-0.25	1	0.73	-0.068	-0.14	-0.23	-0.13	0.12
total sulfur dioxide	-0.33	-0.41	0.19	0.5	-0.28	0.73	1	0.031	-0.24	-0.28	-0.26	-0.042
density	0.46	0.27	0.097	0.55	0.36	-0.068	0.031	1	0.013	0.26	-0.69	-0.3
pH	-0.25	0.26	-0.33	-0.27	0.045	-0.14	-0.24	0.013	1	0.19	0.12	0.018
sulphates	0.3	0.23	0.059	-0.19	0.4	-0.23	-0.28	0.26	0.19	1	-0.0041	0.039
alcohol	-0.096	-0.04	-0.01	-0.36	-0.26	-0.13	-0.26	-0.69	0.12	-0.0041	1	0.44
quality	-0.076	-0.27	0.085	-0.035	-0.2	0.12	-0.042	-0.3	0.018	0.039	0.44	1

[ ]:
```
#Histogram

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
wine_data.hist(figsize=(10,10),bins=50)
plt.show()
```



[3]:
```
#Violinplot:
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

sns.violinplot(x='quality', y='alcohol', data=wine_data)
```

[3]:
```
<Axes: xlabel='quality', ylabel='alcohol'>
```

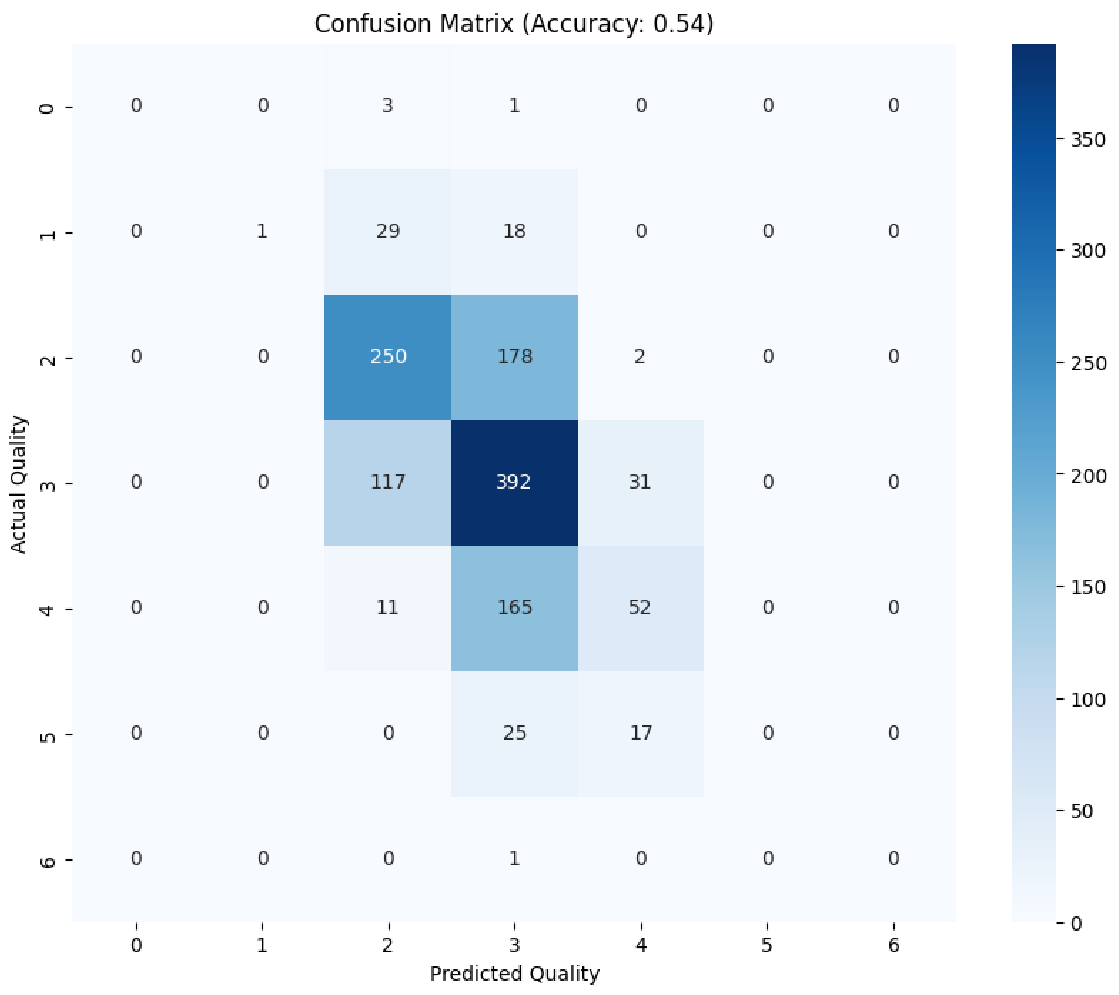


2 BaseLine model

[2]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report,␣
 ↪confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt


# Define 'X' (features) and 'y' (target) from 'data'
X = wine_data.drop('quality', axis=1)
y = wine_data['quality']

# Split the data into train and test sets without scaling
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)

# Train the logistic regression model without scaling
```

```python
model = LogisticRegression(max_iter=1000000)
model.fit(X_train, y_train)

# Make predictions
y_pred = model.predict(X_test)

# Calculate and print model accuracy
model_accuracy_no_scaling = accuracy_score(y_test, y_pred)
print("Accuracy without scaling:", model_accuracy_no_scaling)

# Confusion Matrix
conf_matrix = confusion_matrix(y_test, y_pred)

# Heatmap for Confusion Matrix
plt.figure(figsize=(10, 8))
sns.heatmap(conf_matrix, annot=True, fmt='g', cmap='Blues')
plt.xlabel('Predicted Quality')
plt.ylabel('Actual Quality')
plt.title(f'Confusion Matrix (Accuracy: {model_accuracy_no_scaling:.2f})')
plt.show()

# Print classification report
print(classification_report(y_test, y_pred))
```

```
Accuracy without scaling: 0.5375096674400619
```



```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         4
           4       1.00      0.02      0.04        48
           5       0.61      0.58      0.60       430
           6       0.50      0.73      0.59       540
           7       0.51      0.23      0.32       228
           8       0.00      0.00      0.00        42
           9       0.00      0.00      0.00         1

    accuracy                           0.54      1293
   macro avg       0.37      0.22      0.22      1293
weighted avg       0.54      0.54      0.50      1293
```

Our Initial Baseline accuracy is: 0.53

3 Transformation of the predictors

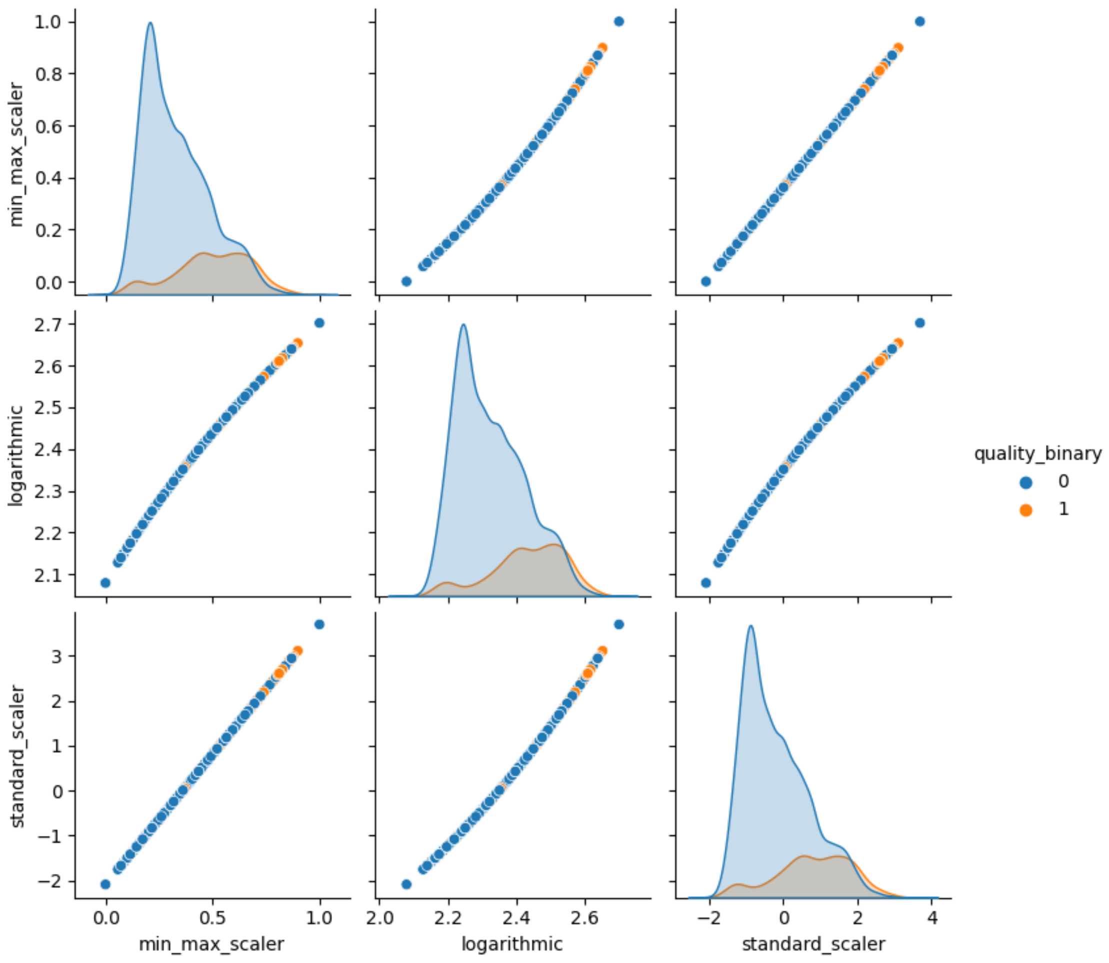
[4]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.preprocessing import MinMaxScaler, StandardScaler, RobustScaler
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt




# Convert 'quality' into a binary target variable
wine_data['quality_binary'] = (wine_data['quality'] >= 7).astype(int)

# Initial predictor
predictor = 'alcohol'

# Defining the transformations
transformations = {
    'min_max_scaler': MinMaxScaler(),
    'logarithmic': lambda x: np.log(x + 1e-5),  # Adding a small constant to␣
 ↪avoid log(0)
    'standard_scaler': StandardScaler(),
    'exponential': np.exp,
    'robust_scaler': RobustScaler()
}

# Placeholder for model results
model_results = {}

for name, transform in transformations.items():
    # Apply the transformation or scaler
    transformed_data = wine_data.copy()
    if callable(transform):
        transformed_data[predictor] = transform(transformed_data[predictor])
    else:
        transformed_data[predictor] = transform.
 ↪fit_transform(transformed_data[[predictor]])

    # Splitting the data into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(
        transformed_data[[predictor]],
        transformed_data['quality_binary'],
        test_size=0.2,
        random_state=42
    )

    # Building the Logistic Regression model
    model = LogisticRegression()
    model.fit(X_train, y_train)

    # Evaluating the model
    train_score = model.score(X_train, y_train)
    test_score = model.score(X_test, y_test)

    # Storing the results
    model_results[name] = {
        'Train Score': train_score,
        'Test Score': test_score
    }

# Display model results
for transformation, scores in model_results.items():
    print(f"{transformation} - Train Score: {scores['Train Score']}, Test Score:
 ↪ {scores['Test Score']}")
```

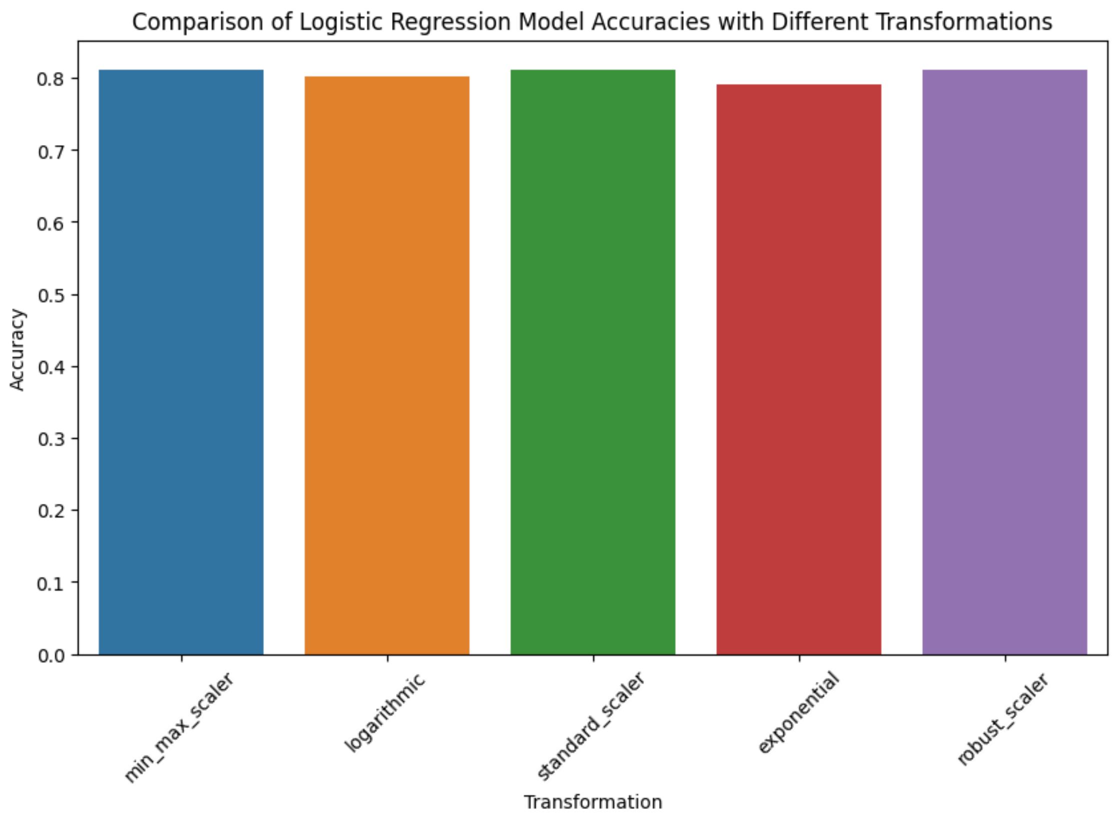
```
min_max_scaler - Train Score: 0.8143133462282398, Test Score: 0.8105181747873164
logarithmic - Train Score: 0.8131528046421663, Test Score: 0.8020108275328693
standard_scaler - Train Score: 0.8168278529980658, Test Score:
0.8120649651972158
exponential - Train Score: 0.8065764023210832, Test Score: 0.7904098994586234
robust_scaler - Train Score: 0.8168278529980658, Test Score: 0.8120649651972158
```

Here are some visualizing for the comparison

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

# For a pair plot, select a few transformations due to space constraints
selected_transformations = ['min_max_scaler', 'logarithmic', 'standard_scaler']
pair_plot_data = wine_data[['quality_binary']].copy()

for name in selected_transformations:
    transformed = transformations[name](wine_data['alcohol'] + 1e-5) if␣
 ↪callable(transformations[name]) else transformations[name].
 ↪fit_transform(wine_data[['alcohol']]).flatten()
    pair_plot_data[name] = transformed

sns.pairplot(pair_plot_data, hue='quality_binary',␣
 ↪vars=selected_transformations)
plt.show()
```

```
import matplotlib.pyplot as plt
import seaborn as sns

# Convert the results to a DataFrame for easier plotting
results_df = pd.DataFrame(model_results).T

# Plotting
plt.figure(figsize=(10, 6))
sns.barplot(data=results_df, x=results_df.index, y='Test Score')
plt.title('Comparison of Logistic Regression Model Accuracies with Different␣
↪Transformations')
plt.ylabel('Accuracy')
plt.xlabel('Transformation')
plt.xticks(rotation=45)
plt.show()
```





4 Feature selection dropping certain predictors

```
[ ]: import pandas as pd
     import numpy as np
     from sklearn.linear_model import LogisticRegression
     from sklearn.model_selection import train_test_split
     from sklearn.metrics import accuracy_score
     from itertools import combinations


     initial_predictors = [
         'fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
         'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
         'pH', 'sulphates', 'alcohol'
     ]

     from itertools import combinations


     # Convert 'quality' into a binary target variable
```

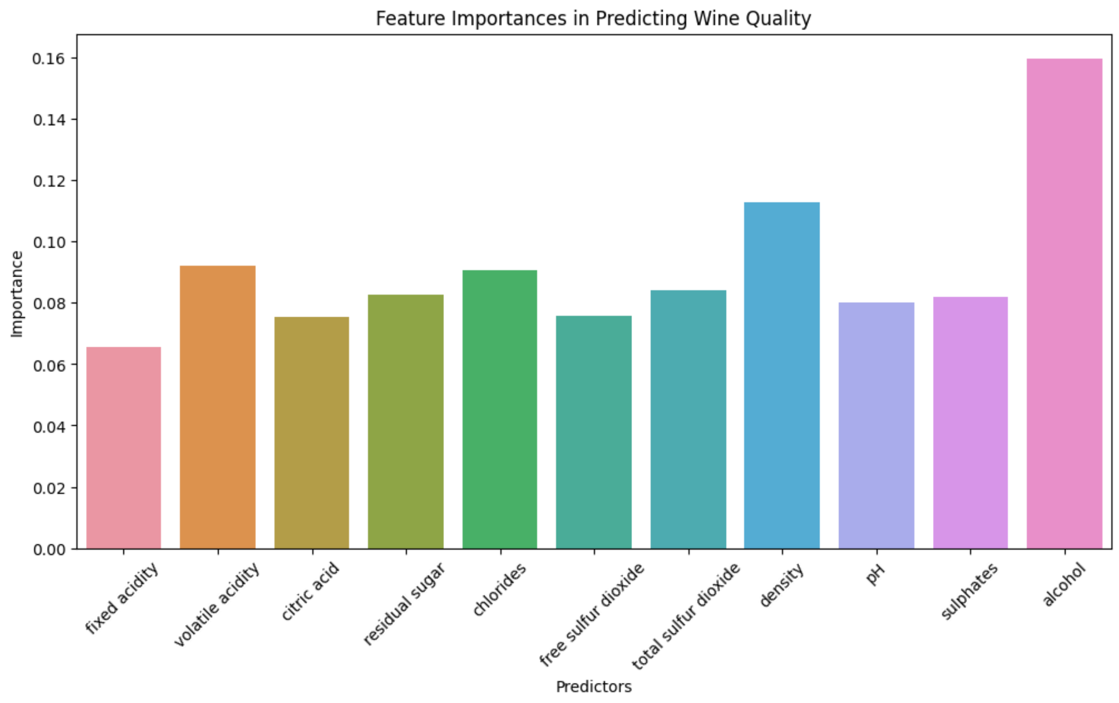
```
wine_data['quality_binary'] = (wine_data['quality'] >= 7).astype(int)

subset_results = {}

# Generate combinations of 1-3 predictors to drop
for r in range(1, 4):
    for subset in combinations(initial_predictors, r):
        # Drop selected predictors
        X_reduced = wine_data.drop(list(subset) + ['quality',␣
 ↪'quality_binary'], axis=1)

        # Split data
        X_train, X_test, y_train, y_test = train_test_split(
            X_reduced, wine_data['quality_binary'], test_size=0.2,␣
 ↪random_state=42)

        # Build Logistic Regression Model
        model = LogisticRegression()
        model.fit(X_train, y_train)

        # Evaluate Model
        accuracy = model.score(X_test, y_test)
        subset_results[', '.join(subset)] = accuracy

        # Limit to 5 different subsets
        if len(subset_results) >= 5:
            break
    if len(subset_results) >= 5:
        break

# Display subset results
for subset, accuracy in subset_results.items():
    print(f"Subset dropped: {subset} - Accuracy: {accuracy}")
```

```
Subset dropped: fixed acidity - Accuracy: 0.8151585460170147
Subset dropped: volatile acidity - Accuracy: 0.805877803557618
Subset dropped: citric acid - Accuracy: 0.8105181747873164
Subset dropped: residual sugar - Accuracy: 0.8105181747873164
Subset dropped: chlorides - Accuracy: 0.8097447795823666
```

Building a Random Forest model to get feature importances

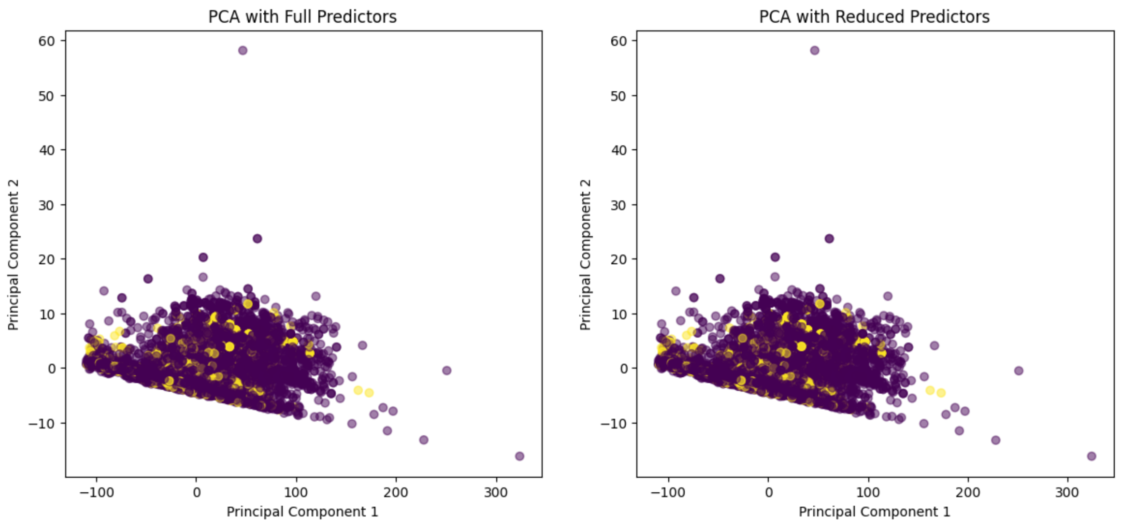
[ ]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.ensemble import RandomForestClassifier
```

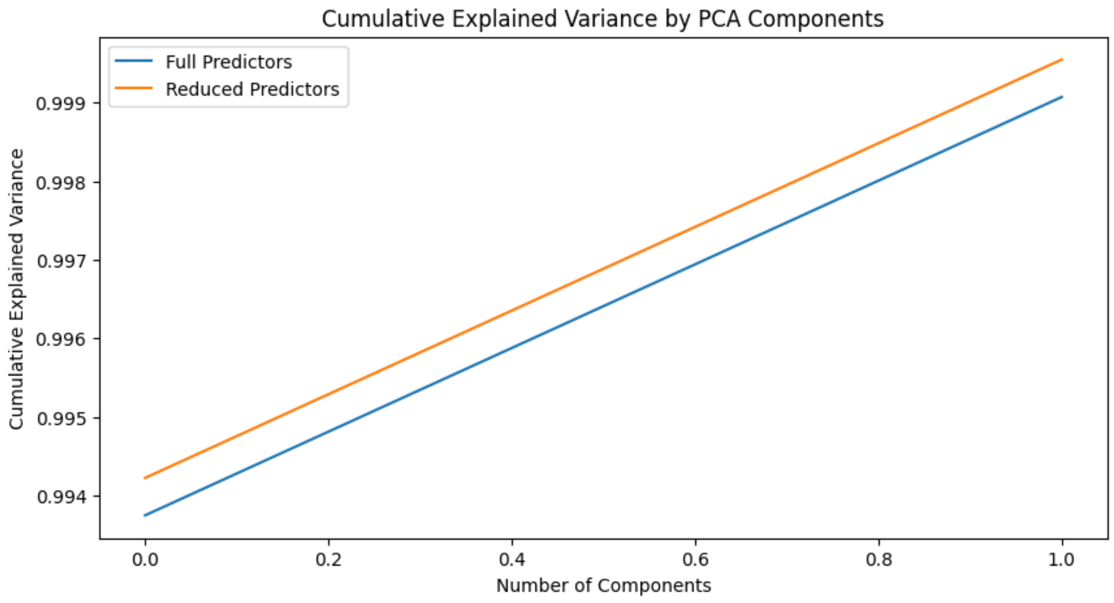
```
# Building a Random Forest model to get feature importances
rf = RandomForestClassifier()
rf.fit(wine_data[initial_predictors], wine_data['quality_binary'])

# Plotting feature importances
plt.figure(figsize=(12, 6))
sns.barplot(x=initial_predictors, y=rf.feature_importances_)
plt.title('Feature Importances in Predicting Wine Quality')
plt.ylabel('Importance')
plt.xlabel('Predictors')
plt.xticks(rotation=45)
plt.show()
```



[ ]:
```
#vizualization after dropping predictors

import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.decomposition import PCA

# PCA with full predictors
```

```
pca_full = PCA(n_components=2)
principal_components_full = pca_full.
 ↪fit_transform(wine_data[initial_predictors])

# PCA with reduced predictors (example: dropping first three predictors)
reduced_predictors = initial_predictors[3:]
pca_reduced = PCA(n_components=2)
principal_components_reduced = pca_reduced.
 ↪fit_transform(wine_data[reduced_predictors])

# Plotting
fig, ax = plt.subplots(1, 2, figsize=(14, 6))

ax[0].scatter(principal_components_full[:, 0], principal_components_full[:, 1],␣
 ↪c=wine_data['quality_binary'], alpha=0.5)
ax[0].set_title('PCA with Full Predictors')
ax[0].set_xlabel('Principal Component 1')
ax[0].set_ylabel('Principal Component 2')

ax[1].scatter(principal_components_reduced[:, 0], principal_components_reduced[:
 ↪, 1], c=wine_data['quality_binary'], alpha=0.5)
ax[1].set_title('PCA with Reduced Predictors')
ax[1].set_xlabel('Principal Component 1')
ax[1].set_ylabel('Principal Component 2')

plt.show()

plt.figure(figsize=(10, 5))

# Full model
plt.plot(np.cumsum(pca_full.explained_variance_ratio_), label='Full Predictors')

# Reduced model
plt.plot(np.cumsum(pca_reduced.explained_variance_ratio_), label='Reduced␣
 ↪Predictors')

plt.xlabel('Number of Components')
plt.ylabel('Cumulative Explained Variance')
plt.legend()
plt.title('Cumulative Explained Variance by PCA Components')
plt.show()
```





5 Remarks

In this phase of the project, we observed how different data preparation techniques influenced the accuracy of Logistic Regression models in predicting wine quality.After implementing various data preparation techniques, we observed notable variations in the accuracy measurements across the new models. Each model, tailored uniquely by preprocessing steps such as normalization, feature selection presented a distinct accuracy score. These variations underscore the profound impact of data quality and structure on model performance.

When these new accuracy scores are contrasted with the bottom-line accuracy, which was achieved prior to any sophisticated data handling, a clear trajectory of improvement is noticeable. This progression highlights the incremental value added through each stage of data preparation. It is, however, crucial to consider that the highest accuracy does not always equate to the best model. Factors such as overfitting, model complexity, and interpretability are vital to consider. In some cases, a slightly lower accuracy might be acceptable if the model is more robust and generalizable to unseen data. Ultimately, these findings reinforce the principle that while data preparation is a complex and iterative process, it is fundamental in developing effective predictive models. Here is our accuracy values after transformation,

”min_max_scaler - Train Score: 0.8143133462282398, Test Score: 0.8105181747873164

logarithmic - Train Score: 0.8131528046421663, Test Score: 0.8020108275328693

standard_scaler - Train Score: 0.8168278529980658, Test Score: 0.8120649651972158

exponential - Train Score: 0.8065764023210832, Test Score: 0.7904098994586234

robust_scaler - Train Score: 0.8168278529980658, Test Score: 0.8120649651972158 ” that scores is much higher that our original bottom line accuracy.We can see that standard_scaler and robust_scaler has the higher accuracy than others.

The feature selection models offer another layer of insight. By reducing the number of predictors, each model tests the hypothesis that a simpler, more focused input can enhance model performance.Comparing these new measurements to each other, it’s evident that some techniques are more effective than others in capturing the underlying patterns of the dataset. For instance, models benefitting from feature engineering—where irrelevant or redundant features were dropped to outperform those without this step. This observation aligns with the common understanding that a more focused and relevant feature set can lead to better model performance.We can observe that after dropping column in our set our accuracy increased by some points.Here is our data after droppping certain predictors.

Subset dropped: fixed acidity - Accuracy: 0.8151585460170147 Subset dropped: volatile acidity - Accuracy: 0.805877803557618 Subset dropped: citric acid - Accuracy: 0.8105181747873164 Subset dropped: residual sugar - Accuracy: 0.8105181747873164 Subset dropped: chlorides - Accuracy: 0.8097447795823666

This accuracy is much higher that our original bottomline accuracy which was Accuracy without scaling: 0.5266821345707656 and in some case this value is slightly higher than transformation values.