Wine Quality Prediction

In [ ]:
```
#importing the libraries
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn import preprocessing
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn import svm
from sklearn import metrics
```

In [ ]:
```
# Loading the dataset
df = pd.read_csv('winequality-red.csv')
df.head()
```

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.6
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.6
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.5
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5

In [ ]:
```
# checking for null values
df.isnull().sum()
```

Out[ ]:
```
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
```

In [ ]:
```
df.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
```

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

Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000

Data Preprocessing

In [ ]:
```
df['quality'].value_counts()
```

Out[ ]:
```
quality
5    681
6    638
7    199
4     53
8     18
3     10
Name: count, dtype: int64
```

In [ ]:
```
df['quality'] = df['quality'].apply(lambda x: 1 if x >= 7 else 0)
df.rename(columns={'quality': 'good-quality'}, inplace=True)
df.head()
```

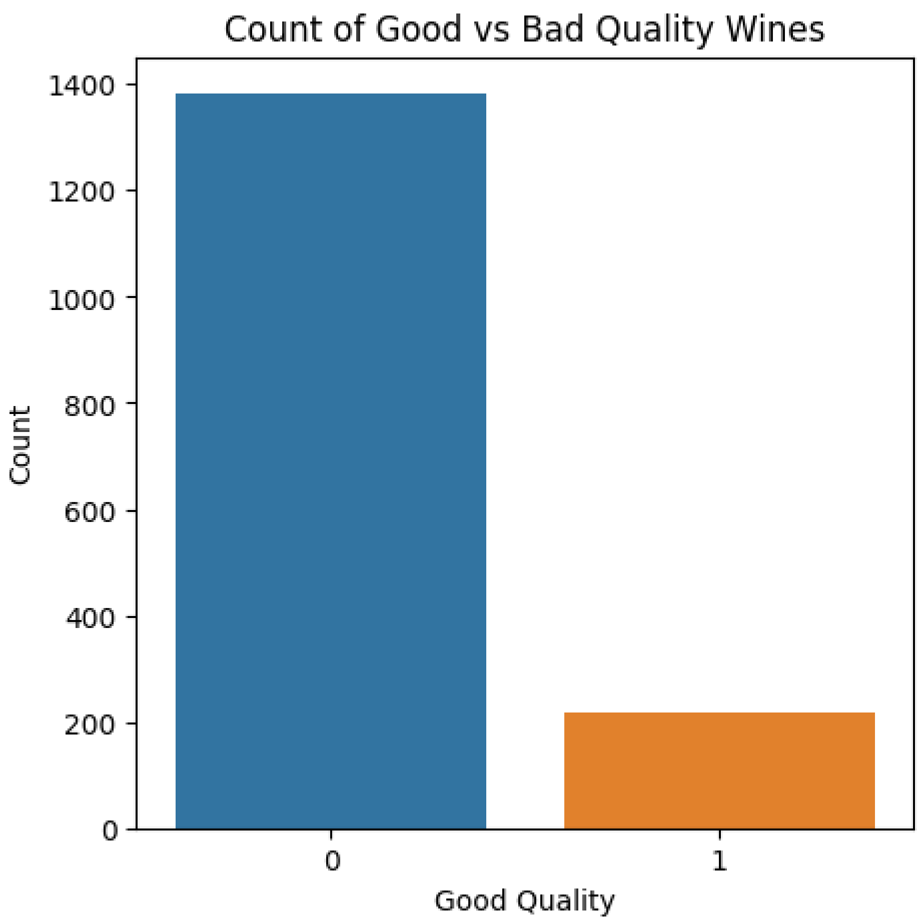
Out[ ]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphate
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.6
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.6
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.5
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.5

Exploratory Data Analysis

In [ ]:

```
plt.figure(figsize=(5,5))
sns.countplot(x='good-quality', data=df)
plt.xlabel('Good Quality')
plt.ylabel('Count')
plt.title('Count of Good vs Bad Quality Wines')
plt.show()
```

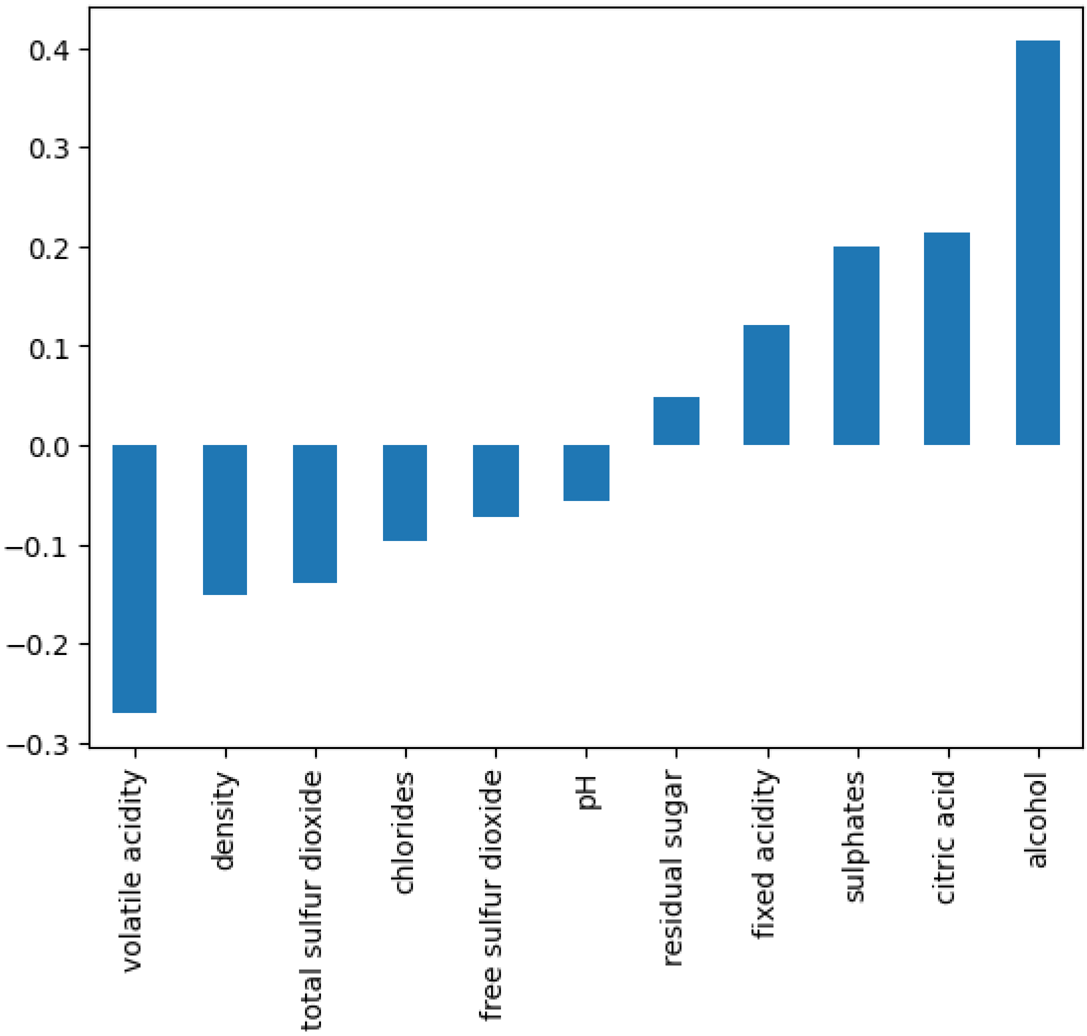


Analysis of coorelation between features

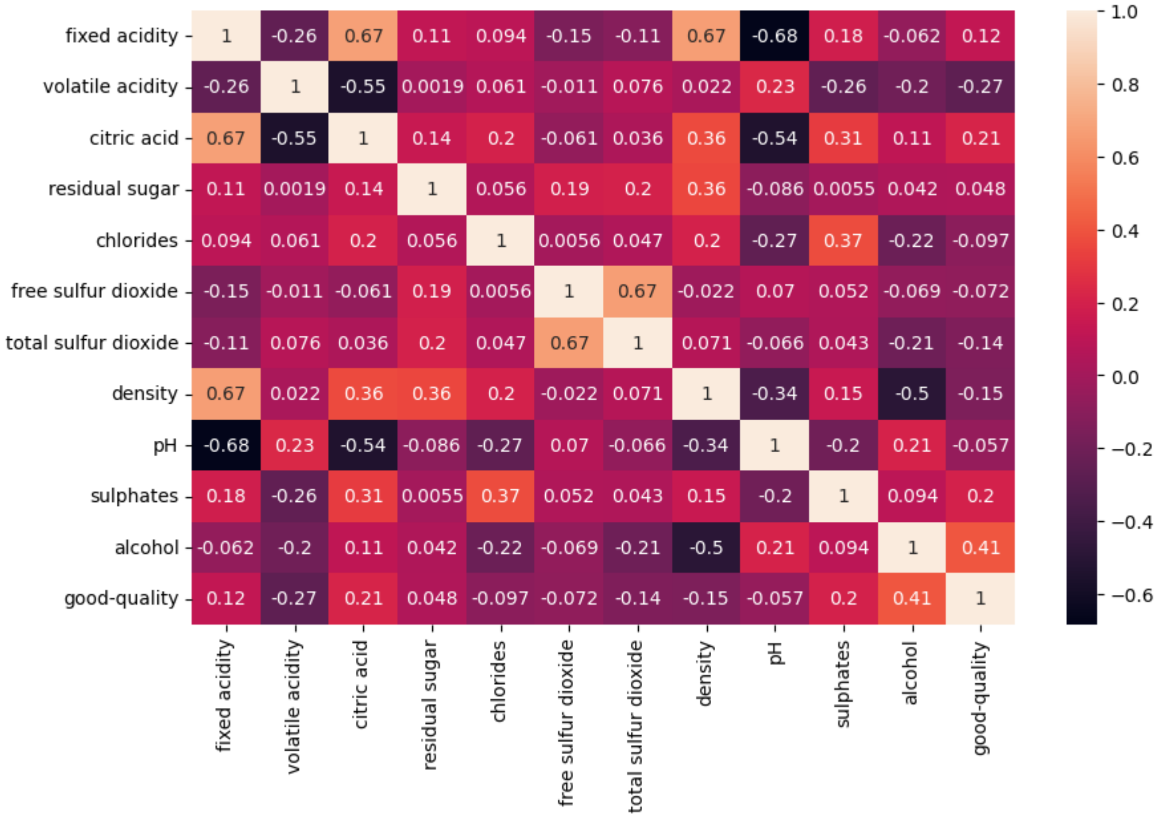
In [ ]:

```
df.corr()['good-quality'][:-1].sort_values().plot(kind='bar')
```

Out[ ]: <Axes: >

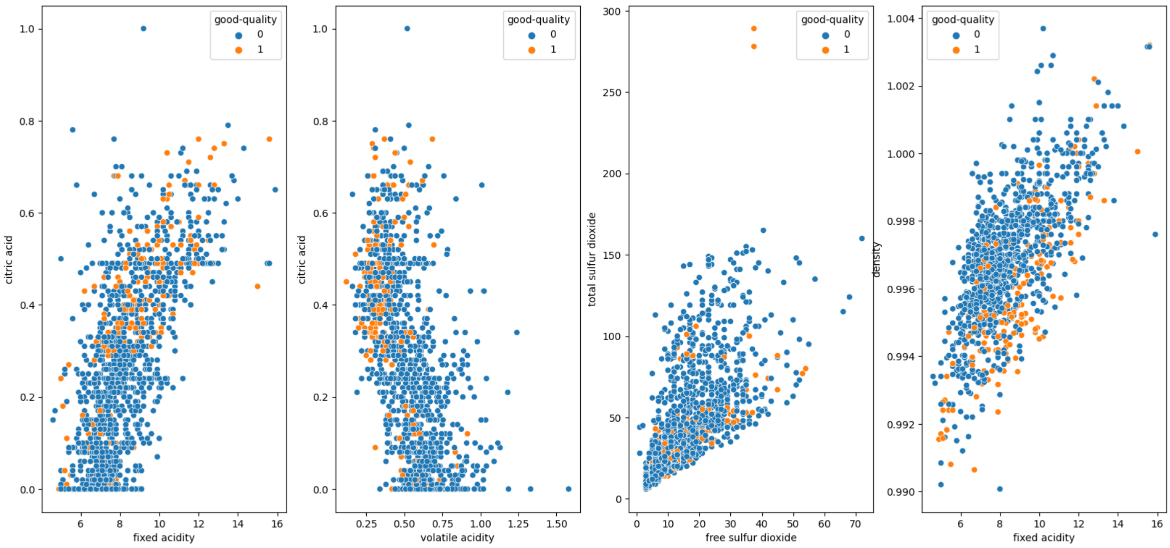


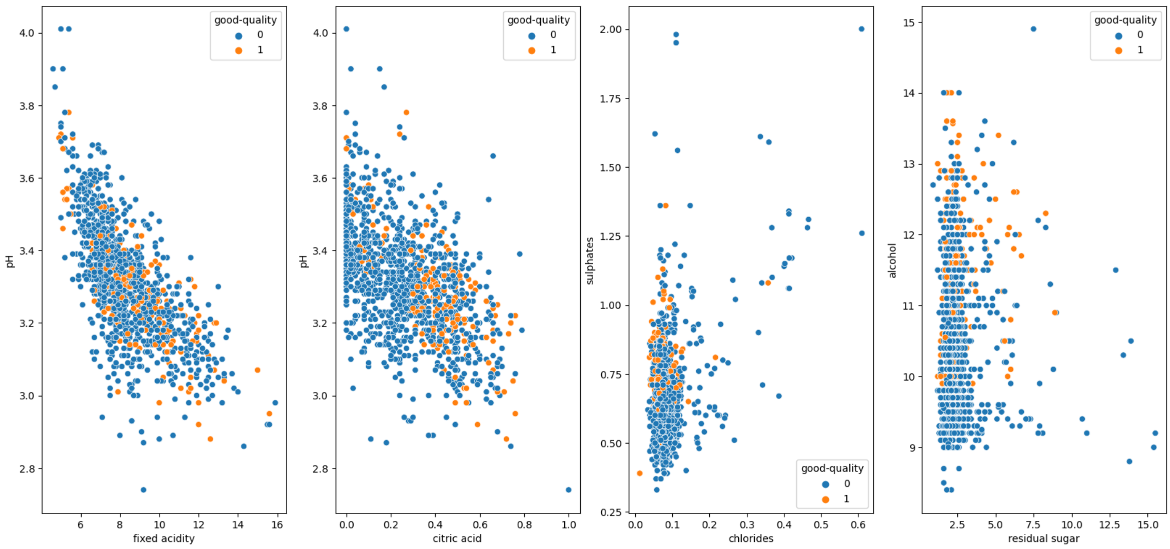
In [ ]:
```
plt.figure(figsize=(10,6))
sns.heatmap(df.corr(), annot=True)
plt.show()
```



In [ ]:
```
fig, ax = plt.subplots(2,4,figsize=(20,20))
sns.scatterplot(x = 'fixed acidity', y = 'citric acid', hue = 'good-quality', da
sns.scatterplot(x = 'volatile acidity', y = 'citric acid', hue = 'good-quality',
sns.scatterplot(x = 'free sulfur dioxide', y = 'total sulfur dioxide', hue = 'gc
sns.scatterplot(x = 'fixed acidity', y = 'density', hue = 'good-quality', data =
sns.scatterplot(x = 'fixed acidity', y = 'pH', hue = 'good-quality', data = df,
sns.scatterplot(x = 'citric acid', y = 'pH', hue = 'good-quality', data = df, ax
sns.scatterplot(x = 'chlorides', y = 'sulphates', hue = 'good-quality', data = c
sns.scatterplot(x = 'residual sugar', y = 'alcohol', hue = 'good-quality', data
```

Out[ ]: <Axes: xlabel='residual sugar', ylabel='alcohol'>





Train Test Split

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(df.drop('good-quality', axis
```

Model Training

Logistic Regression

In [ ]:
```
lr = LogisticRegression()
lr
```

Out[ ]:
```
▾ LogisticRegression
LogisticRegression()
```

In [ ]:
```
# training the model
lr.fit(X_train, y_train)
lr.score(X_train, y_train)
```

```
C:\Users\DELL\AppData\Local\Packages\PythonSoftwareFoundation.Python.3.11_qbz5n2k
fra8p0\LocalCache\local-packages\Python311\site-packages\sklearn\linear_model\_lo
gistic.py:458: 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(
```

Out[ ]: 0.8838248436103664

In [ ]:
```
# testing the model
lr_pred = lr.predict(X_test)
accuracy_score(y_test, lr_pred)
```

Out[ ]: 0.8666666666666667

Support Vector Machine (SVM)

In [ ]:
```
clf = svm.SVC(kernel='rbf')
clf
```

Out[ ]:
```
▾ SVC
SVC()
```

In [ ]:
```
# training the model
clf.fit(X_train, y_train)
clf.score(X_train, y_train)
```

Out[ ]: 0.8668453976764968

In [ ]:
```
# testing the model
sv_pred = clf.predict(X_test)
accuracy_score(y_test, sv_pred)
```

Out[ ]: 0.8625

Decision Tree

In [ ]:
```
dtree = DecisionTreeClassifier()
dtree
```

Out[ ]:
```
▾ DecisionTreeClassifier
DecisionTreeClassifier()
```

In [ ]:
```
# training the model
dtree.fit(X_train, y_train)
dtree.score(X_train, y_train)
```

Out[ ]: 1.0

In [ ]:
```
# testing the model
tr_pred = dtree.predict(X_test)
accuracy_score(y_test, tr_pred)
```

Out[ ]: 0.8645833333333334

K-Nearest Neighbors (KNN)

In [ ]:
```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors=5)
knn
```

Out[ ]:
```
▾ KNeighborsClassifier
KNeighborsClassifier()
```

In [ ]:
```
# training the model
knn.fit(X_train, y_train)
knn.score(X_train, y_train)
```

Out[ ]: 0.9079535299374442

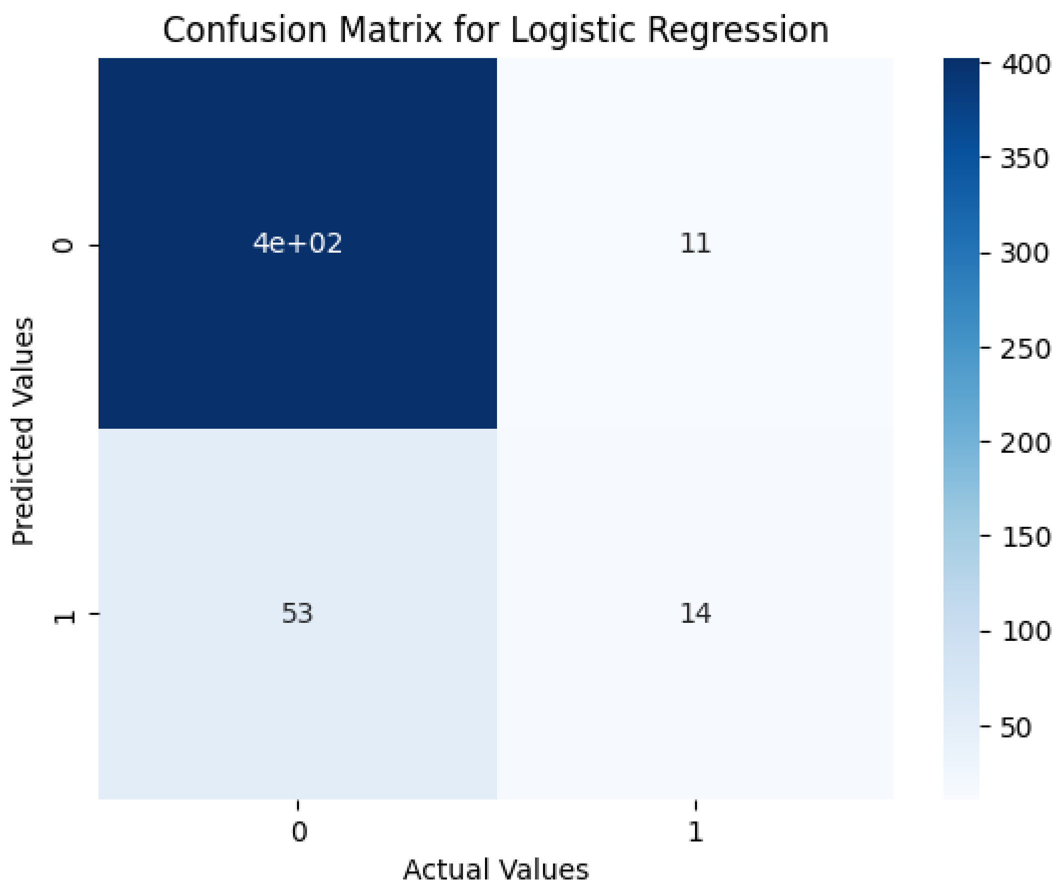
In [ ]:
```
# testing the model
kn_pred = knn.predict(X_test)
accuracy_score(y_test, kn_pred)
```

Out[ ]: 0.8583333333333333

Model Evaluation

Logistic Regression

In [ ]:
```
# logistic regression model evaluation
sns.heatmap(confusion_matrix(y_test, lr_pred), annot=True, cmap='Blues')
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Logistic Regression')
plt.show()
```

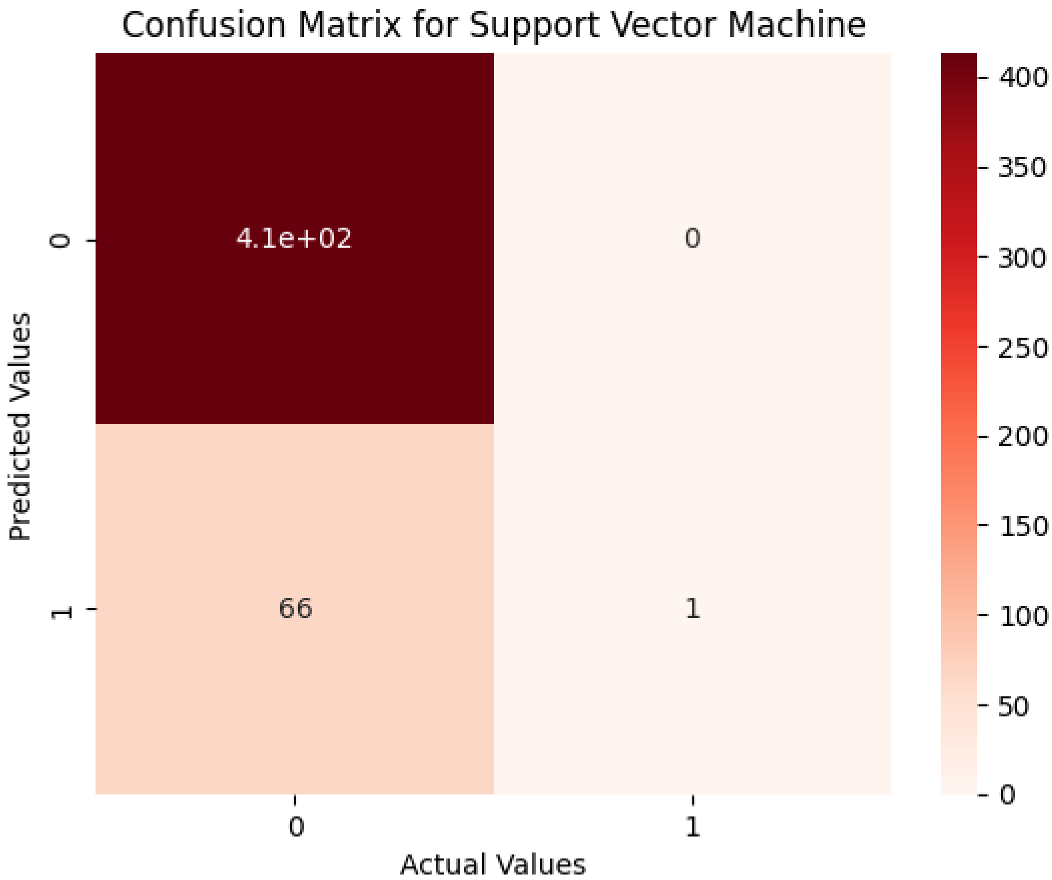


In [ ]:
```
print('Logistic Regression Model Accuracy: ', accuracy_score(y_test, lr_pred))
print('Logistic Regression Model f1 score: ', metrics.f1_score(y_test, lr_pred))
print('Logistic Regression Model MAE: ', metrics.mean_absolute_error(y_test, lr_
print('Logistic Regression Model RMSE: ', np.sqrt(metrics.mean_squared_error(y_t
```

```
Logistic Regression Model Accuracy:  0.8666666666666667
Logistic Regression Model f1 score:  0.30434782608695654
Logistic Regression Model MAE:  0.13333333333333333
Logistic Regression Model RMSE:  0.3651483716701107
```

Support Vector Machine (SVM)

In [ ]:
```
sns.heatmap(confusion_matrix(y_test, sv_pred), annot=True, cmap='Reds')
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Support Vector Machine')
plt.show()
```

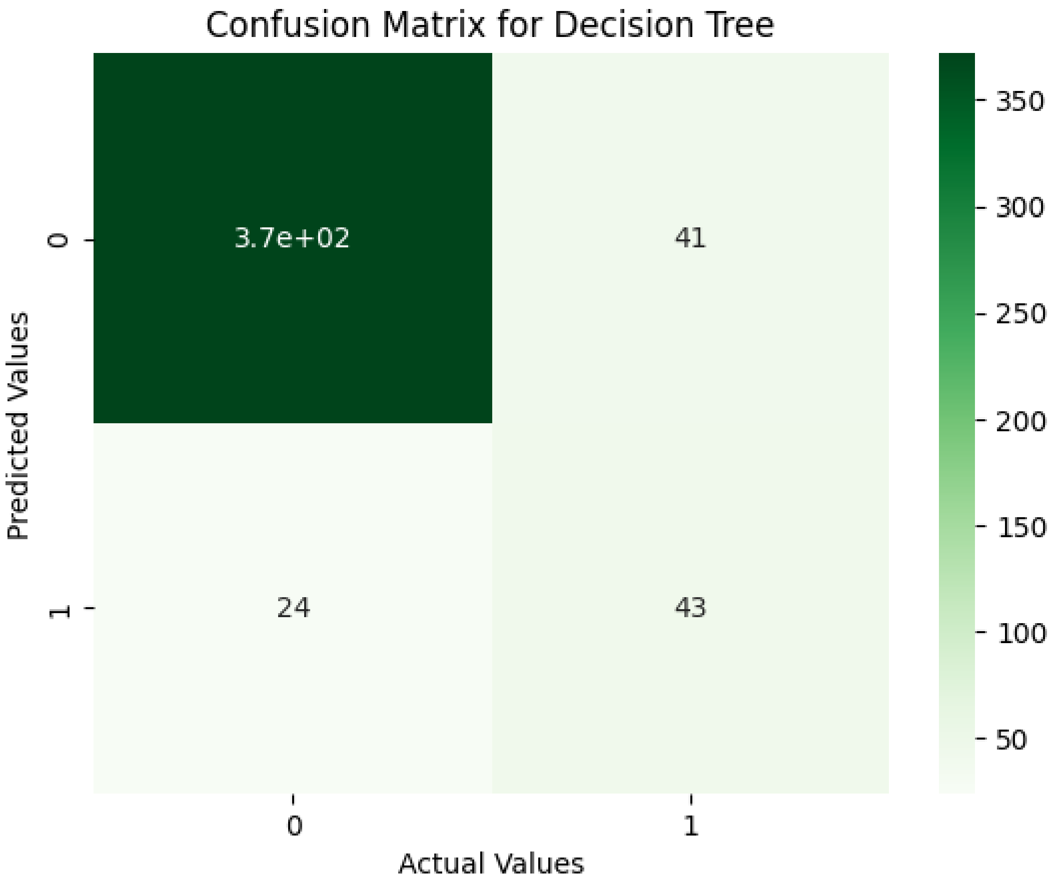


In [ ]:
```
print('Support Vector Machine Model Accuracy: ', accuracy_score(y_test, sv_pred)
print('Support Vector Machine Model f1 score: ', metrics.f1_score(y_test, sv_pre
print('Support Vector Machine Model MAE: ', metrics.mean_absolute_error(y_test,
print('Support Vector Machine Model RMSE: ', np.sqrt(metrics.mean_squared_error(
```

```
Support Vector Machine Model Accuracy:  0.8625
Support Vector Machine Model f1 score:  0.029411764705882353
Support Vector Machine Model MAE:  0.1375
Support Vector Machine Model RMSE:  0.37080992435478316
```

Decision Tree

In [ ]:
```
sns.heatmap(confusion_matrix(y_test, tr_pred), annot=True, cmap='Greens')
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for Decision Tree')
plt.show()
```

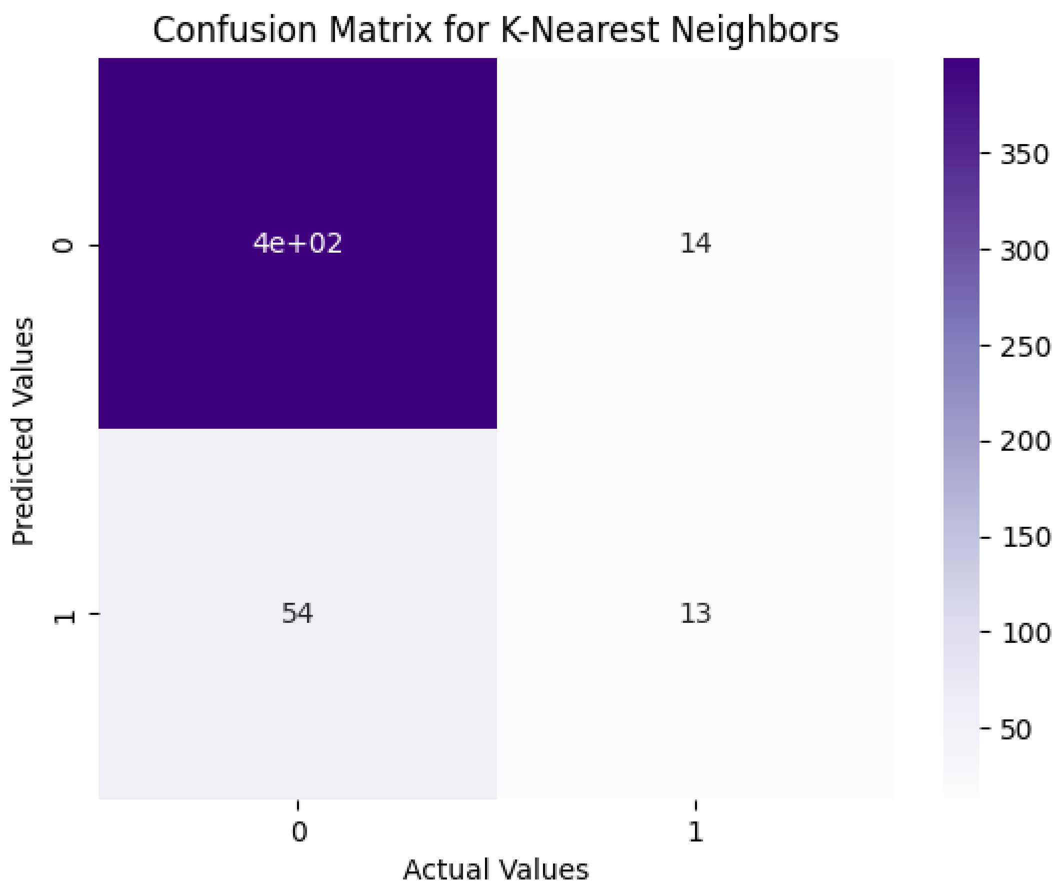


In [ ]:
```
print('Decision Tree Model Accuracy: ', accuracy_score(y_test, tr_pred))
print('Decision Tree Model f1 score: ', metrics.f1_score(y_test, tr_pred))
print('Decision Tree Model MAE: ', metrics.mean_absolute_error(y_test, tr_pred))
print('Decision Tree Model RMSE: ', np.sqrt(metrics.mean_squared_error(y_test, t
```

```
Decision Tree Model Accuracy:  0.8645833333333334
Decision Tree Model f1 score:  0.5695364238410596
Decision Tree Model MAE:  0.13541666666666666
Decision Tree Model RMSE:  0.3679900360969936
```

K-Nearest Neighbors (KNN)

In [ ]:
```
sns.heatmap(confusion_matrix(y_test, kn_pred), annot=True, cmap='Purples')
plt.ylabel('Predicted Values')
plt.xlabel('Actual Values')
plt.title('Confusion Matrix for K-Nearest Neighbors')
plt.show()
```

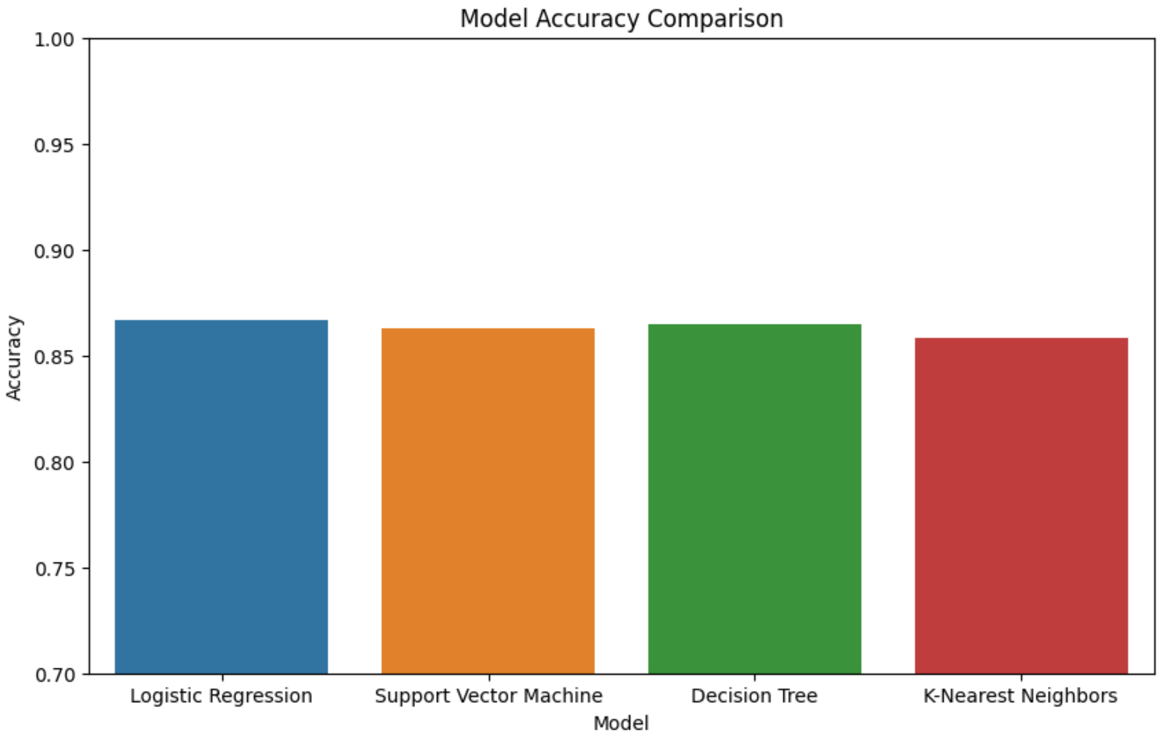


In [ ]:
```
print('K-Nearest Neighbors Model Accuracy: ', accuracy_score(y_test, kn_pred))
print('K-Nearest Neighbors Model f1 score: ', metrics.f1_score(y_test, kn_pred))
print('K-Nearest Neighbors Model MAE: ', metrics.mean_absolute_error(y_test, kn_
print('K-Nearest Neighbors Model RMSE: ', np.sqrt(metrics.mean_squared_error(y_t
```

```
K-Nearest Neighbors Model Accuracy:  0.8583333333333333
K-Nearest Neighbors Model f1 score:  0.276595744680851
K-Nearest Neighbors Model MAE:  0.14166666666666666
K-Nearest Neighbors Model RMSE:  0.3763863263545405
```

Model Comparison

In [ ]:
```
models = ['Logistic Regression', 'Support Vector Machine', 'Decision Tree', 'K-N
accuracy = [accuracy_score(y_test, lr_pred), accuracy_score(y_test, sv_pred), ac
plt.figure(figsize=(10,6))
sns.barplot(x=models, y=accuracy)
plt.title('Model Accuracy Comparison')
plt.xlabel('Model')
plt.ylabel('Accuracy')
plt.ylim(0.7, 1.0)
plt.show()
```



Conclusion

It is observed that the Logistic Regression model performs the best on the test set with an accuracy of 86.67%. The model can predict the quality of the wine based on the given features with an accuracy of 86.67%.