KEY STEPS IN DATA PREPARATION AND INITIAL ANALYSIS:

Importing Libraries:

Before beginning any analysis or modeling tasks, it's essential to import the necessary libraries and tools. These typically include libraries for data manipulation (e.g., pandas), data visualization (e.g., seaborn, matplotlib), and machine learning algorithms (e.g., scikit-learn).

Loading Dataset:

To start the analysis, we load the dataset containing information about red wine attributes into our environment. This dataset serves as the foundation for our predictive modeling tasks and contains features such as acidity levels, alcohol content, pH levels, and quality ratings provided by wine experts.

Description and Finding Null Values:

Upon loading the dataset, we typically inspect its structure and contents using methods like `info()` to understand the data types of each column and check for missing values. Identifying null values is crucial as they can affect the integrity of our analysis and modeling efforts.

Data Analysis:

Data analysis involves exploring the dataset to gain insights into the distribution of features and their relationships with each other. Techniques such as countplots, KDE plots, distplots, histograms, and heatmaps are employed to visualize the data and uncover patterns or correlations that may exist.

Feature Selection:

Feature selection is the process of identifying the most relevant features that significantly influence the target variable (wine quality). This step helps streamline the modeling process by focusing on the most informative attributes while disregarding irrelevant or redundant ones.

Feature Importance:

For certain machine learning algorithms like Random Forest or XGBoost, feature importance scores can be calculated to determine the contribution of each feature to the predictive model's performance. Features with higher importance scores are considered more influential in predicting wine quality.

Splitting Dataset:

Before building predictive models, the dataset is typically split into training and testing sets. The training set is used to train the models, while the testing set is used to evaluate their performance on unseen data. This ensures that the models generalize well to new observations.

MODEL BUILDING:

Various machine learning algorithms are employed to build predictive models for red wine quality prediction. These algorithms include Logistic Regression, K-Nearest Neighbors (KNN), Support Vector Classifier (SVC), Decision Tree, Gaussian Naive Bayes (NB), Random Forest, and XGBoost. Each algorithm has its strengths and weaknesses, making it suitable for different types of data and problem scenarios.

1. Linear Regression:

- Linear regression can be applied to predict the quality of wine as a continuous numerical value based on various input features such as acidity levels, sugar content, pH level, etc.
- It models the linear relationship between the input features and the target variable (wine quality score) and provides insights into how each feature contributes to overall quality.

2. Decision Trees:

- Decision trees can be used to predict wine quality by recursively splitting the dataset based on feature values.
- Each node in the tree represents a feature, and each leaf node corresponds to a predicted quality score.
- Decision trees are interpretable and can capture non-linear relationships between features and wine quality, making them suitable for exploratory analysis and understanding feature importance.

3. Random Forest:

- Random Forests can improve upon decision trees by building an ensemble of multiple trees and combining their predictions.
- They can handle a large number of input features and provide robust predictions by reducing overfitting.
- Random Forests also offer feature importance scores, helping to identify which attributes have the most significant impact on wine quality.

4. Support Vector Machines (SVM):

- SVMs can be used for wine quality prediction by finding the optimal hyperplane that separates different quality classes based on feature values.
- They are effective for classification tasks with complex decision boundaries and can handle high-dimensional data well.
- SVMs are particularly useful when the distribution of classes is not linearly separable in the feature space.

5. K-Nearest Neighbors (KNN):

- KNN can predict wine quality by calculating the similarity between a new wine sample and its nearest neighbors in the feature space.
- It makes predictions based on the average quality scores of the k nearest neighbors, where k is a user-defined parameter.

- KNN is non-parametric and does not make strong assumptions about the underlying data distribution, making it suitable for exploratory analysis and small to moderate-sized datasets.

6. Gaussian Naive Bayes:

- Gaussian Naive Bayes can be used for wine quality prediction by modeling the conditional probability distribution of quality classes given the input features.
- It assumes that features are conditionally independent given the class label and calculates the posterior probability of each class using Bayes' theorem.
- Gaussian Naive Bayes is computationally efficient and works well with continuous features following a Gaussian distribution.

7. XGBoost:

- XGBoost is a powerful gradient boosting algorithm that can be used for wine quality prediction by sequentially building decision trees to minimize a differentiable loss function.
- It provides high predictive accuracy, handles missing values, and incorporates regularization techniques to prevent overfitting.
- XGBoost is widely used in competitions and real-world applications for its efficiency, scalability, and ability to achieve state-of-the-art results in predictive modeling tasks.

These algorithms offer a variety of approaches for predicting wine quality based on different aspects of the data and problem requirements. Experimentation and model evaluation are essential to determine which algorithm performs best for a specific wine quality prediction task.

SOURCE CODE:

Importing Libraries

```
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 Dataset

```
wine = pd.read_csv("winequality-red.csv")
print("Successfully Imported Data!")
wine.head()
print(wine.shape)
```

Description

```
wine.describe(include='all')
```

Finding Null Values

```
print(wine.isna().sum())
wine.corr()
wine.groupby('quality').mean()
```

Data Analysis

Countplot:

```
sns.countplot(wine['quality'])
plt.show()
sns.countplot(wine['pH'])
plt.show()
sns.countplot(wine['alcohol'])
plt.show()
sns.countplot(wine['fixed acidity'])
plt.show()
sns.countplot(wine['volatile acidity'])
plt.show()
sns.countplot(wine['citric acid'])
plt.show()
sns.countplot(wine['density'])
plt.show()
```

KDE plot:

```
sns.kdeplot(wine.query('quality > 2').quality)
```

Distplot:

```
sns.distplot(wine['alcohol'])
wine.plot(kind ='box',subplots = True, layout =(4,4),sharex = False)
wine.plot(kind ='density',subplots = True, layout =(4,4),sharex = False)
```

Histogram

```
wine.hist(figsize=(10,10),bins=50)
plt.show()
```

Heatmap for expressing correlation

```
corr = wine.corr()
sns.heatmap(corr,annot=True)
```

Pair Plot:

```
sns.pairplot(wine)
```

Violinplot:

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

Feature Selection

Create Classification version of target variable

```
wine['goodquality'] = [1 if x >= 7 else 0 for x in wine['quality']]# Separate feature variables and target variable
X = wine.drop(['quality','goodquality'], axis = 1)
Y = wine['goodquality']
```

See proportion of good vs bad wines

```
wine['goodquality'].value_counts()
X
print(Y)
```

Feature Importance

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()

from sklearn.ensemble import ExtraTreesClassifier
classifiern = ExtraTreesClassifier()
classifiern.fit(X,Y)
score = classifiern.feature_importances_
print(score)
```

Splitting Dataset

```
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X,Y,test_size=0.3,random_state=7)
```

LogisticRegression:

```
from sklearn.linear_model import LogisticRegression
model = LogisticRegression()
model.fit(X_train,Y_train)
Y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score,confusion_matrix
print("Accuracy Score:",accuracy_score(Y_test,Y_pred))
confusion_mat = confusion_matrix(Y_test,Y_pred)
print(confusion_mat)
```

Using KNN:

```
from sklearn.neighbors import KNeighborsClassifier
model = KNeighborsClassifier(n_neighbors=3)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred))
```

Using SVC:

```
from sklearn.svm import SVC
model = SVC()
model.fit(X_train,Y_train)
pred_y = model.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,pred_y))
```

Using Decision Tree:

```
from sklearn.tree import DecisionTreeClassifier
model = DecisionTreeClassifier(criterion='entropy',random_state=7)
model.fit(X_train,Y_train)
y_pred = model.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred))
```

Using GaussianNB:

```
from sklearn.naive_bayes import GaussianNB
model3 = GaussianNB()
model3.fit(X_train,Y_train)
y_pred3 = model3.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred3))
```

Using Random Forest:

```
from sklearn.ensemble import RandomForestClassifier
model2 = RandomForestClassifier(random_state=1)
model2.fit(X_train, Y_train)
y_pred2 = model2.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred2))
```

Using Xgboost:

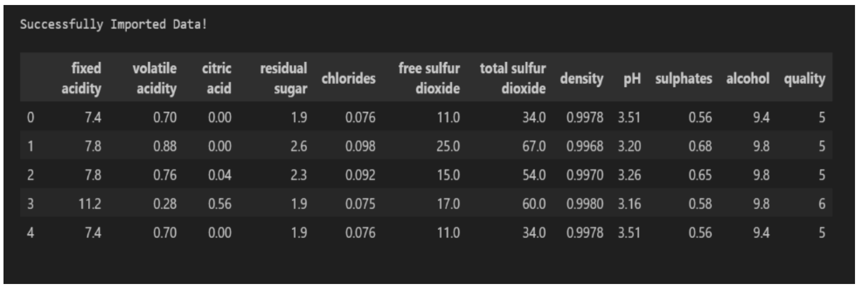
```
import xgboost as xgb
model5 = xgb.XGBClassifier(random_state=1)
model5.fit(X_train, Y_train)
y_pred5 = model5.predict(X_test)

from sklearn.metrics import accuracy_score
print("Accuracy Score:",accuracy_score(Y_test,y_pred5))
results = pd.DataFrame({
    'Model': ['Logistic Regression','KNN', 'SVC','Decision Tree' ,'GaussianNB','Random Forest','Xgboost'],
    'Score': [0.870,0.872,0.868,0.864,0.833,0.893,0.879]})

result_df = results.sort_values(by='Score', ascending=False)
result_df = result_df.set_index('Score')
result_df
```

SCREENSHOTS OF THE RESULT:

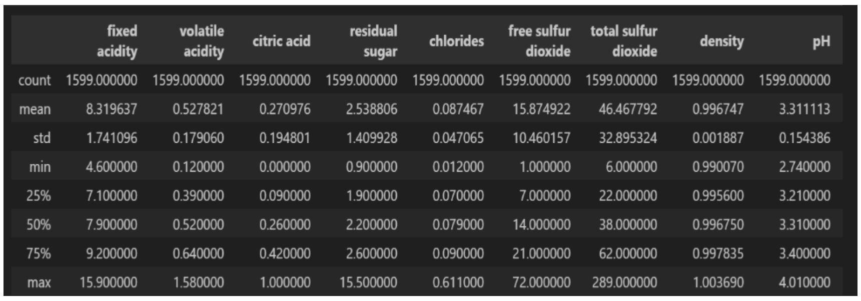
Loading Dataset



```
Successfully Imported Data!
```

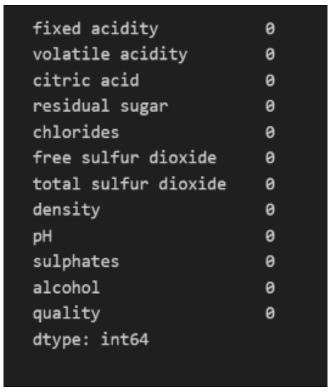
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	5
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	5
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	6
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	5

Description

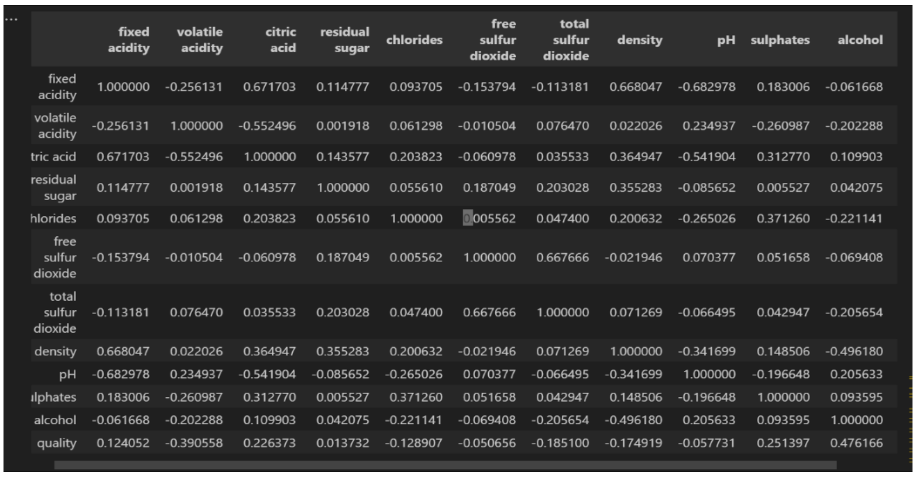


	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000

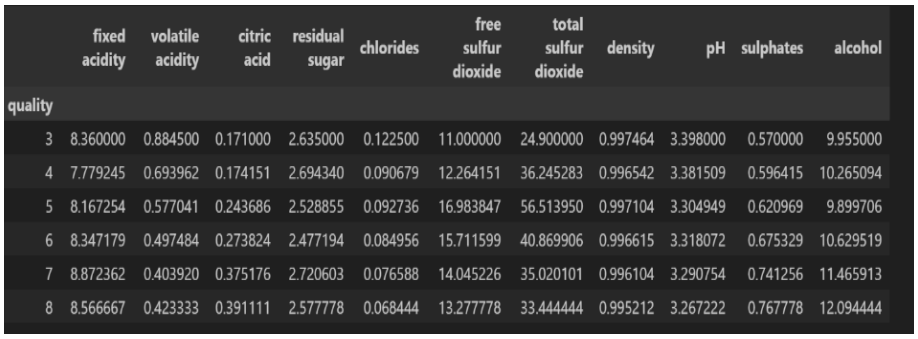
Finding Null Values



```
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
```

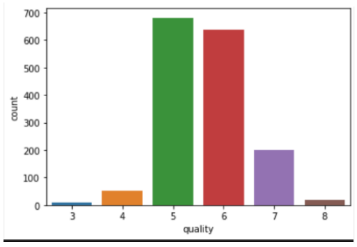


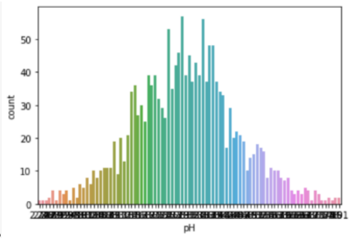
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-0.682978	0.183006	-0.061668
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	0.234937	-0.260987	-0.202288
tric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-0.541904	0.312770	0.109903
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-0.085652	0.005527	0.042075
hlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-0.265026	0.371260	-0.221141
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	0.070377	0.051658	-0.069408
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-0.066495	0.042947	-0.205654
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-0.341699	0.148506	-0.496180
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1.000000	-0.196648	0.205633
ılphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-0.196648	1.000000	0.093595
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	0.205633	0.093595	1.000000
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919	-0.057731	0.251397	0.476166

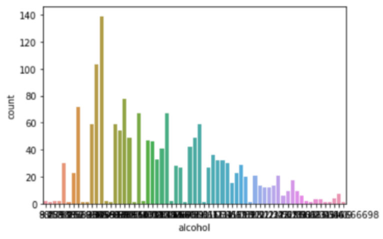


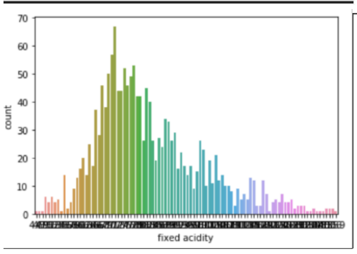
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
quality											
3	8.360000	0.884500	0.171000	2.635000	0.122500	11.000000	24.900000	0.997464	3.398000	0.570000	9.955000
4	7.779245	0.693962	0.174151	2.694340	0.090679	12.264151	36.245283	0.996542	3.381509	0.596415	10.265094
5	8.167254	0.577041	0.243686	2.528855	0.092736	16.983847	56.513950	0.997104	3.304949	0.620969	9.899706
6	8.347179	0.497484	0.273824	2.477194	0.084956	15.711599	40.869906	0.996615	3.318072	0.675329	10.629519
7	8.872362	0.403920	0.375176	2.720603	0.076588	14.045226	35.020101	0.996104	3.290754	0.741256	11.465913
8	8.566667	0.423333	0.391111	2.577778	0.068444	13.277778	33.444444	0.995212	3.267222	0.767778	12.094444

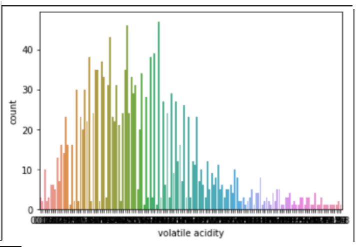
Countplot:

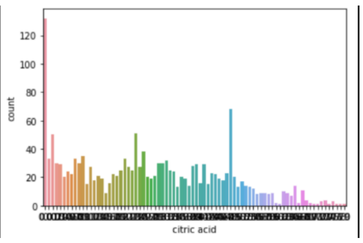


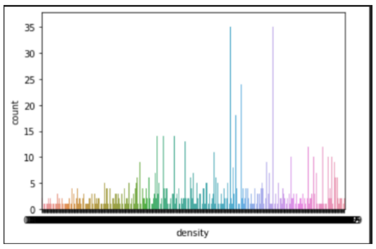




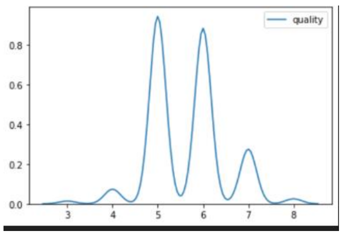




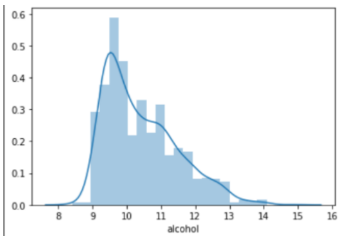


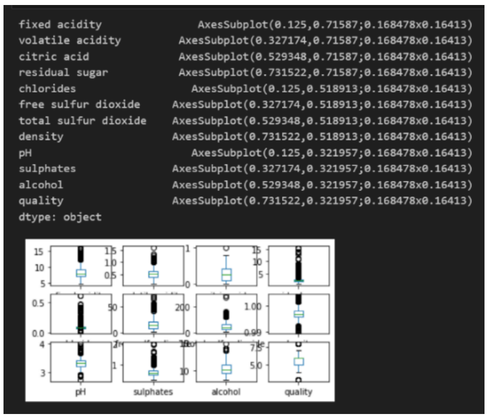


KDE plot:

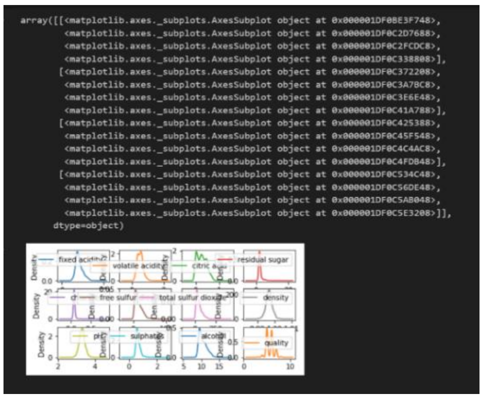


Distplot:



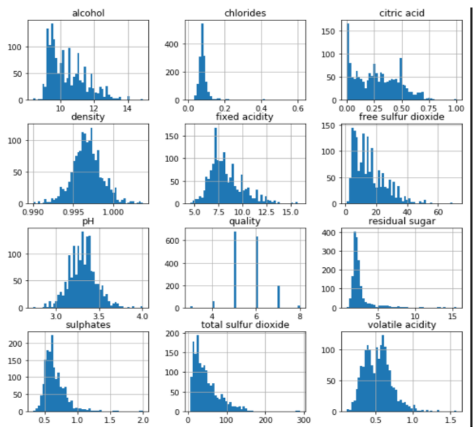


```
fixed acidity                AxesSubplot(0.125,0.71587;0.168478x0.16413)
volatile acidity          AxesSubplot(0.327174,0.71587;0.168478x0.16413)
citric acid               AxesSubplot(0.529348,0.71587;0.168478x0.16413)
residual sugar            AxesSubplot(0.731522,0.71587;0.168478x0.16413)
chlorides                   AxesSubplot(0.125,0.518913;0.168478x0.16413)
free sulfur dioxide      AxesSubplot(0.327174,0.518913;0.168478x0.16413)
total sulfur dioxide     AxesSubplot(0.529348,0.518913;0.168478x0.16413)
density                  AxesSubplot(0.731522,0.518913;0.168478x0.16413)
pH                          AxesSubplot(0.125,0.321957;0.168478x0.16413)
sulphates                AxesSubplot(0.327174,0.321957;0.168478x0.16413)
alcohol                  AxesSubplot(0.529348,0.321957;0.168478x0.16413)
quality                  AxesSubplot(0.731522,0.321957;0.168478x0.16413)
dtype: object
```

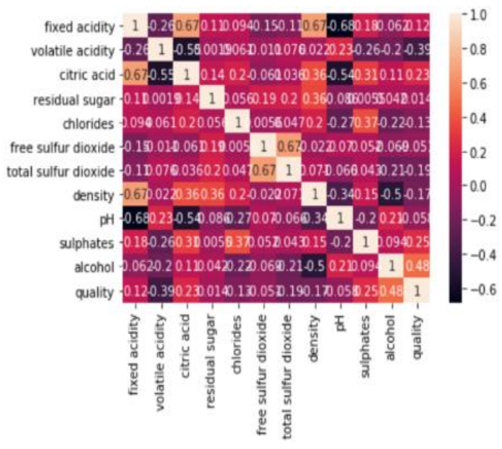


```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0BE3F748>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C2D7688>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C2FCDC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C338808>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C372208>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C3A7BC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C3E6E48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C41A788>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C425388>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C45F548>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C4C4AC8>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C4FDB48>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C534C48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C56DE48>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C5AB048>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x000001DF0C5E3208>]],
      dtype=object)
```

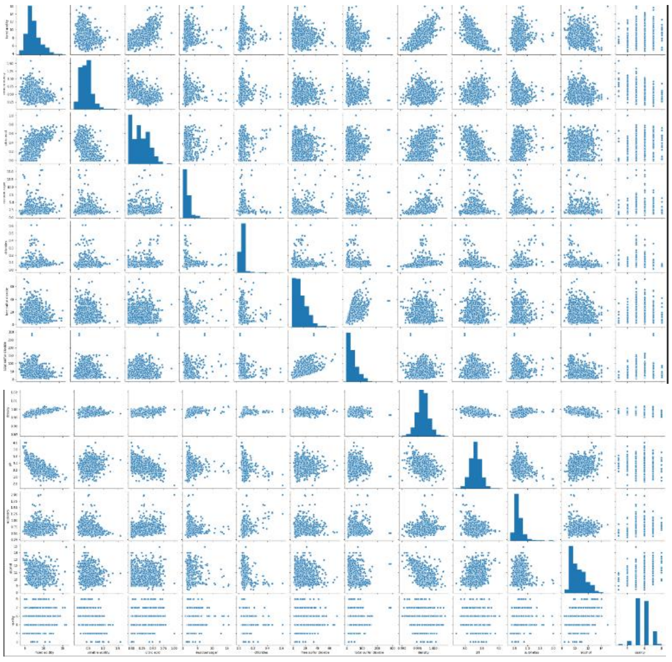
Histogram:



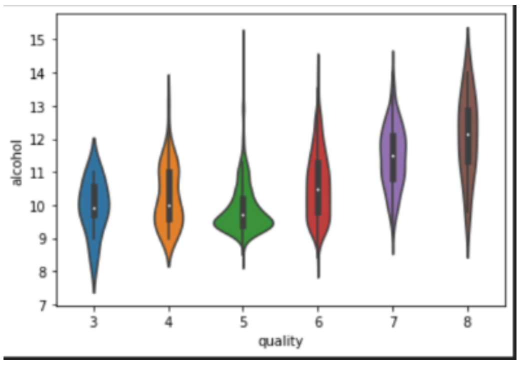
Heatmap for expressing correlation:



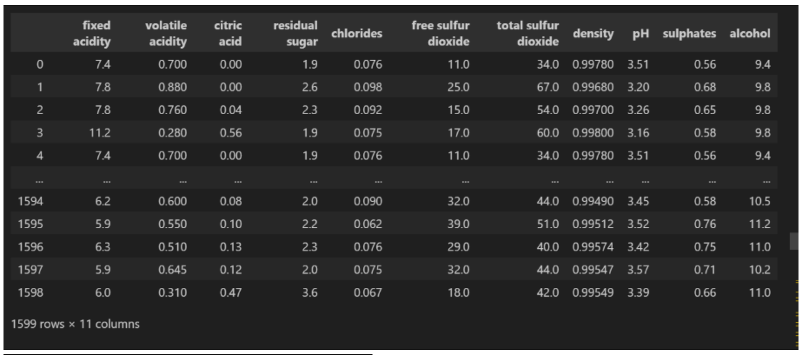
Pair Plot:



Violinplot:

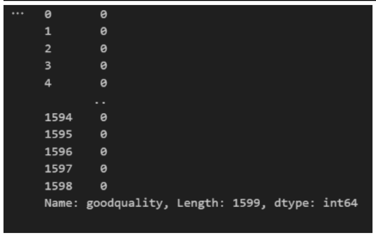


Feature Selection



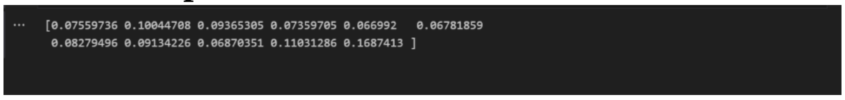
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4
...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0

1599 rows × 11 columns



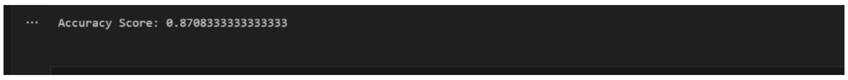
```
...  0       0
     1       0
     2       0
     3       0
     4       0
            ..
     1594    0
     1595    0
     1596    0
     1597    0
     1598    0
     Name: goodquality, Length: 1599, dtype: int64
```

Feature Importance



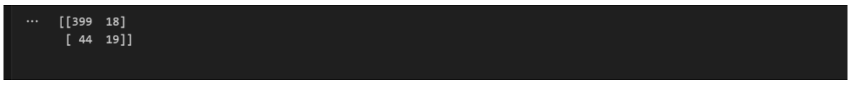
```
···  [0.07559736 0.10044708 0.09365305 0.07359705 0.066992   0.06781859
      0.08279496 0.09134226 0.06870351 0.11031286 0.1687413 ]
```

Splitting Dataset



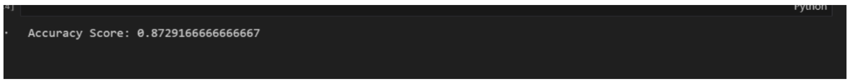
```
···  Accuracy Score: 0.8708333333333333
```

LogisticRegression



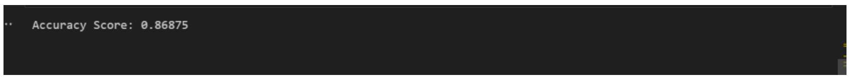
```
···  [[399  18]
      [ 44  19]]
```

Using KNN:



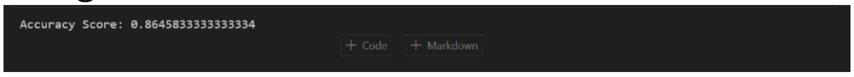
```
·  Accuracy Score: 0.8729166666666667
```

Using SVC:



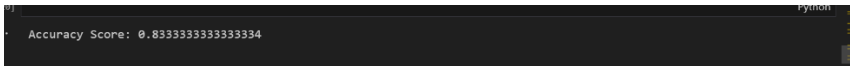
```
··  Accuracy Score: 0.86875
```

Using Decision Tree:



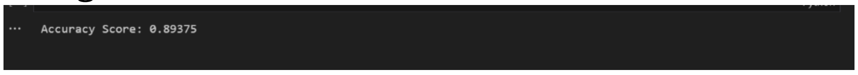
```
Accuracy Score: 0.8645833333333334
```

Using GaussianNB:



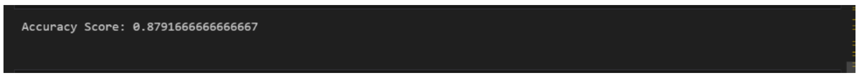
```
·  Accuracy Score: 0.8333333333333334
```

Using Random Forest:

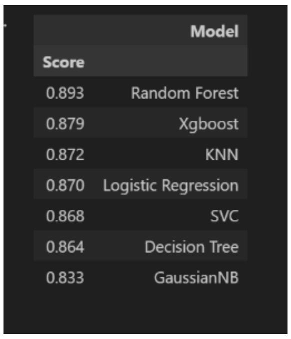


```
···  Accuracy Score: 0.89375
```

Using XgBoost:



```
Accuracy Score: 0.8791666666666667
```



	Model
Score	
0.893	Random Forest
0.879	Xgboost
0.872	KNN
0.870	Logistic Regression
0.868	SVC
0.864	Decision Tree
0.833	GaussianNB