Analysis of Classification Methods on Wine Quality Data

Chenthuran Abeyakaran, Lucas Alcantara, Arron Zheng

Introduction:

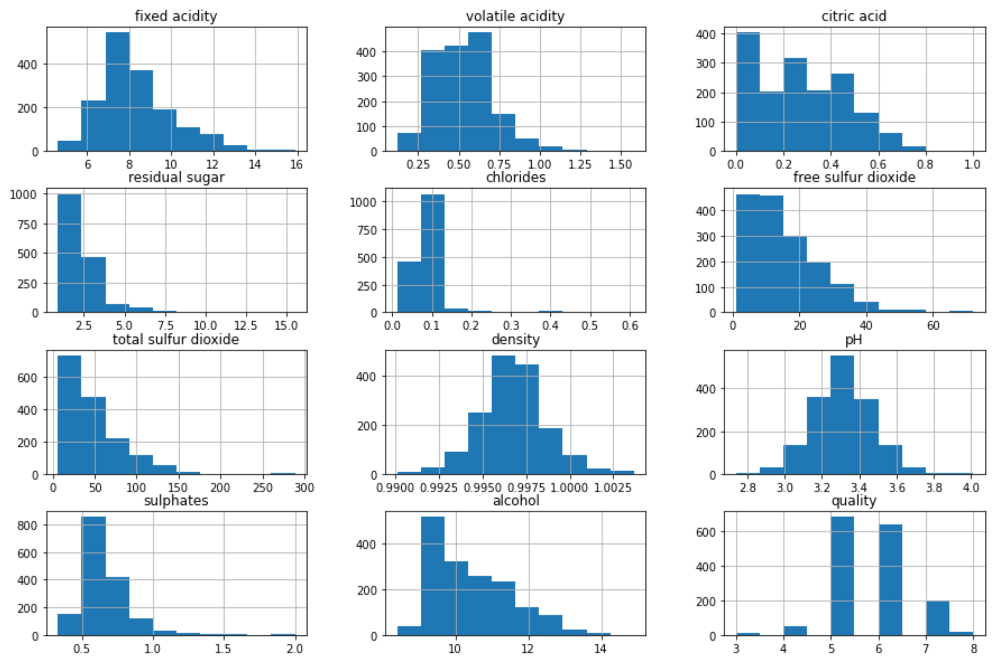
The purpose of this report is to demonstrate the materials and knowledge that reflect the curriculum of Machine Learning and Pattern Classification (ESE417) at Washington University in St. Louis. The curriculum provides a mathematical and practical understanding of machine learning, using real datasets to experiment on with self coded models, such as the iris dataset, a gamma particles dataset, and, for this project, the wine dataset.

We seek to demonstrate a thorough understanding of the material learned throughout the semester, by examining classification methods covered in class. Classification problems revolve around predicting a binary or categorical label given other predictive features. Since our data set contains true labels, we leverage supervised learning techniques. In our paper specifically, we examine the performance of support vector machines (SVM) and random forest classifiers to classify Portuguese wine samples as either high quality or low quality.

Dataset and Data Cleaning:

Looking at the data set, there are 1599 observations and 12 features. The target feature is wine quality which takes on discrete scores from 3-8. The other explanatory features are listed below: 1) Fixed Acidity, 2) Volatile Acidity, 3) Citric Acid, 4) Residual Sugar, 5) Chlorides, 6) Free Sulfur Dioxide, 7) Total Sulfur Dioxide, 8) Density, 9) pH, 10) Sulphates, and 11) Alcohol.

We took a closer look at the features individually to help guide our data cleaning process. Shown below are the empirical distributions for the individual features:



Via data cleaning, it was also determined that no values in any sample had an empty value. With the exception of the target variable, the quality of the wine sample, all variables are continuous and real.

Next, we look for and remove duplicate observations from our data points, since duplicate data points tend to represent oversampling certain wines when collecting data. In total, there are 240 duplicate observations.

Methods:

Support Vector Machines (SVM) - Overview: The Support Vector Machine (SVM) is a classification algorithm that aims to construct a hyperplane to separate data points by their label. Although it is similar to perceptron classification, SVM is significantly stronger than perceptron classifiers. The distance from the closest data point to the hyperplane is commonly known as the **margin.** SVM maximizes the margin, or the distance between the hyperplane and the nearest points on either side; this results in a model that usually does well in avoiding overfitting on training data. Furthermore, SVM can address problems that are non-linearly separable points by taking advantage of feature-space transformations.

Support Vector Machines (SVM) - Implementation: For our project, we used the scikit-learn implementation for SVC (C-Support Vector Classification). To implement the best model, Three hyper parameters were tuned by utilizing a Cross Validation grid search:

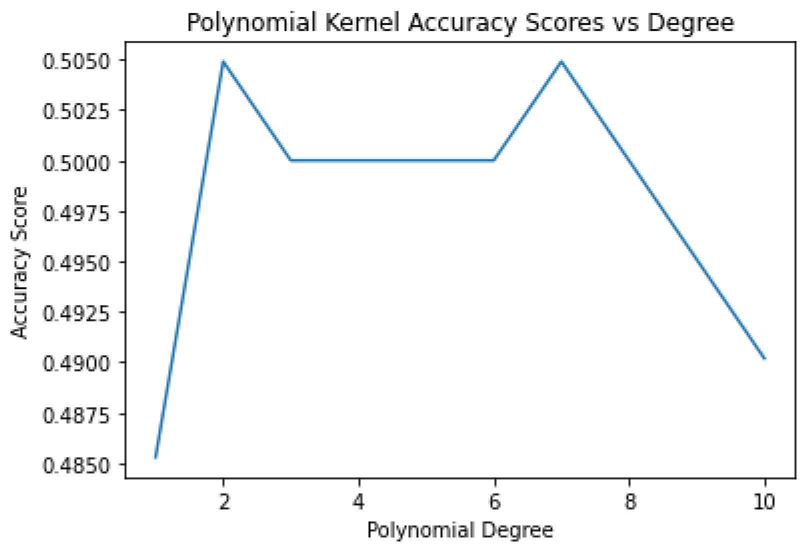
- C (Regularization parameter): C is inversely related to the strength of regularization. As such, C can be any positive real number. By default, SciKitLearn sets C equal to the L2 penalty parameter, and We specifically looked at a range of values between 0.05 and 1.5.
- Kernel (feature-space transformations): There are several different candidate feature-space transformations in the universe of kernel functions. Specifically for the scikit-learn implementation, the main ones are linear, polynomial, RBF (Radial Basis Function), and sigmoid. Polynomial kernels require a degree for the transformation.
- Degree. The degree is only used for a polynomial kernel and represents the polynomial degree transformation applied. We looked specifically at positive integers 1 - 10.

Random Forest Classifier: It is a classification model built on the decision trees classifier which uses multiple trees. The class which is classified the most from these trees is the final class. We used grid search and cross-validation in order to choose the best hyperparameters for our model. Due to the imbalance of the dataset, we used the scorer for the grid search to be a weighted average of the f1-score for each class.
The three hyperparameters we optimized for the random forest classifier are:

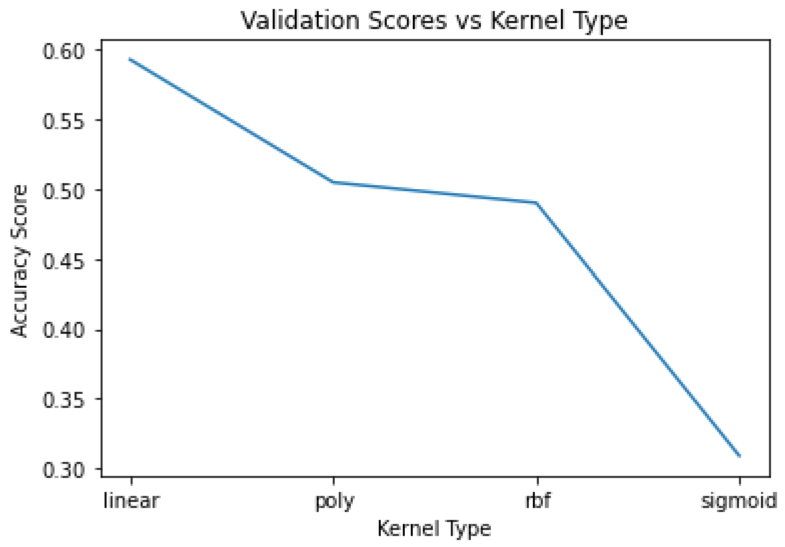
- **Number of Estimators:** This is the number of trees used for the classification of the trees.
- **Max Depth:** This is what represents the maximum depth of each individual tree. By limiting the depth of individual decision trees, 'overfitting' on training data can be avoided.
- **Criterion:** The criterion is the method by which the classifier decides which feature to make the decision feature for the root node of a tree. The two criterions available are ‘entropy’ for using the information gain and ‘gini’ for the Gini impurity.

Results and Analysis:

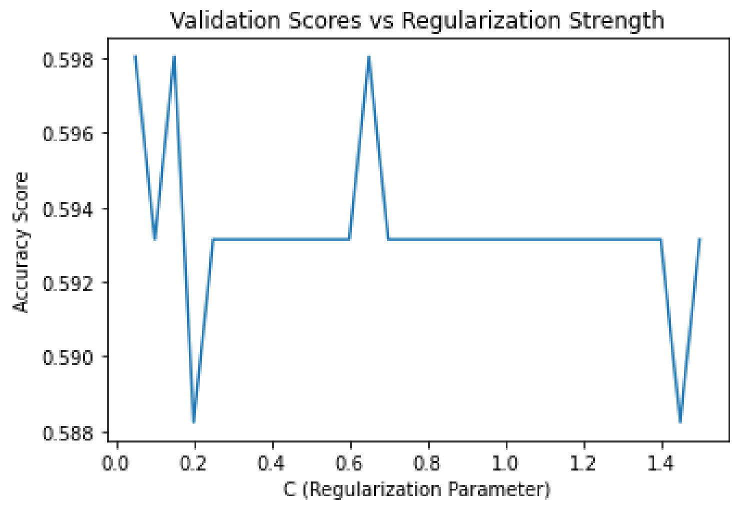
To evaluate the performance of our models, we looked at accuracy first and foremost. Furthermore, we looked at the confusion matrix and weighted f1-score since the target classes are imbalanced.
When implementing SVM, we searched for the optimal combination of hyperparameters listed above. First, we looked at the optimal degree for the polynomial kernel. Looking at the accuracy plot below, we can see that when looking at accuracy and efficiency, the best value is degree=2.



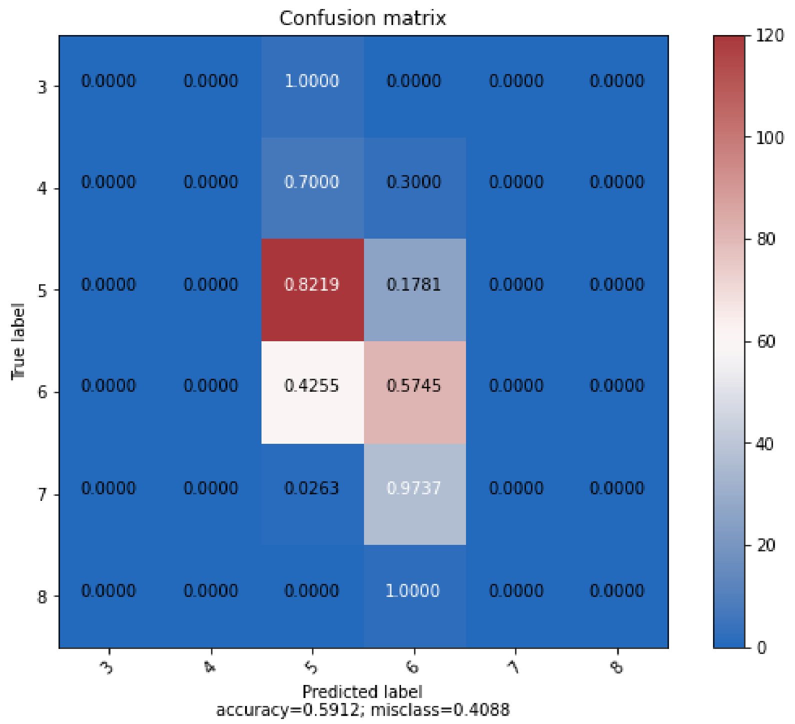
Next, using a polynomial kernel with degree, we look at the accuracy score by kernel types. Looking at the accuracy plot below, we can see that the best performing kernel is the linear kernel.



Finally, having selected the linear kernel, we look at the accuracy scores with different values for the regularization strength parameters. Looking at the accuracy plot below, we can see that the default C=0.65 is one of the best values.



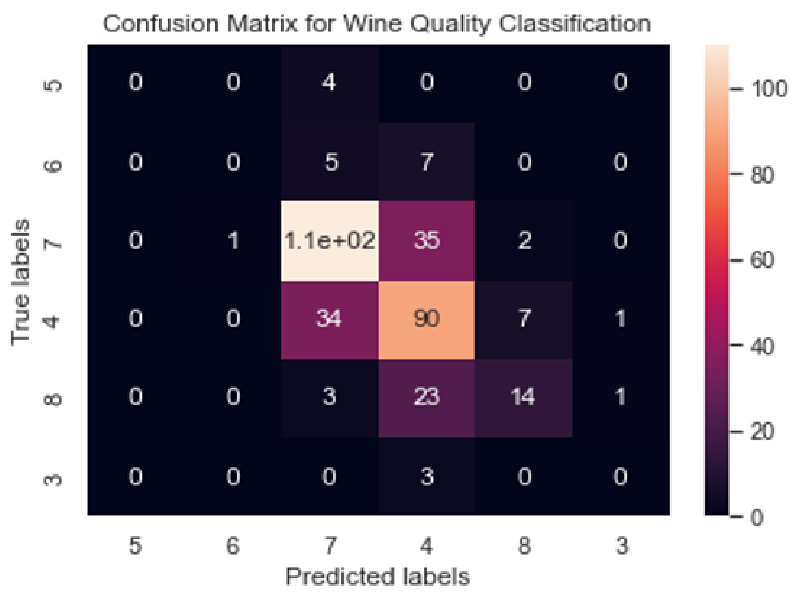
As a result, using the combination of C=0.65 and the linear kernel, we get a test accuracy score of 59.12% and the following confusion matrix.



The results of the grid search for the best parameters that show the best score during cross-validation is {'criterion': 'gini', 'max_depth': 13, 'n_estimators': 100} and the weighted average of the f1 score for each class is 0.6087. The accuracy score for the best parameters random forest model is 0.63. The table below shows the classification report for the model with the best parameters:

Class	Precision	Recall	F1-score	Support
3	0	0	0	4
4	0	0	0	12
5	0.71	0.74	0.72	148
6	0.57	0.68	0.62	132
7	0.61	0.34	0.44	41
8	0	0	0	3

Below we have displayed the confusion matrix for our classifier.



Conclusions:

After analyzing the results of the 2 models used, we can see that the random forest model performed the best as its weighted average of the f1-score is 0.61 compared to 0.52 for the support vector classifier and also the accuracy score of the random forest model is higher with a score of 0.63 compared to 0.57 of the support vector classifier. Both models performed poorly and are clearly under fitted for our data.

Contributions:

Code:

Random Forest: Chenthuran Abeyakaran, Lucas Alcantara, Arron Zheng

SVM: Chenthuran Abeyakaran, Lucas Alcantara, Arron Zheng

Report:

Intro: Lucas

Dataset/Data cleaning: Lucas

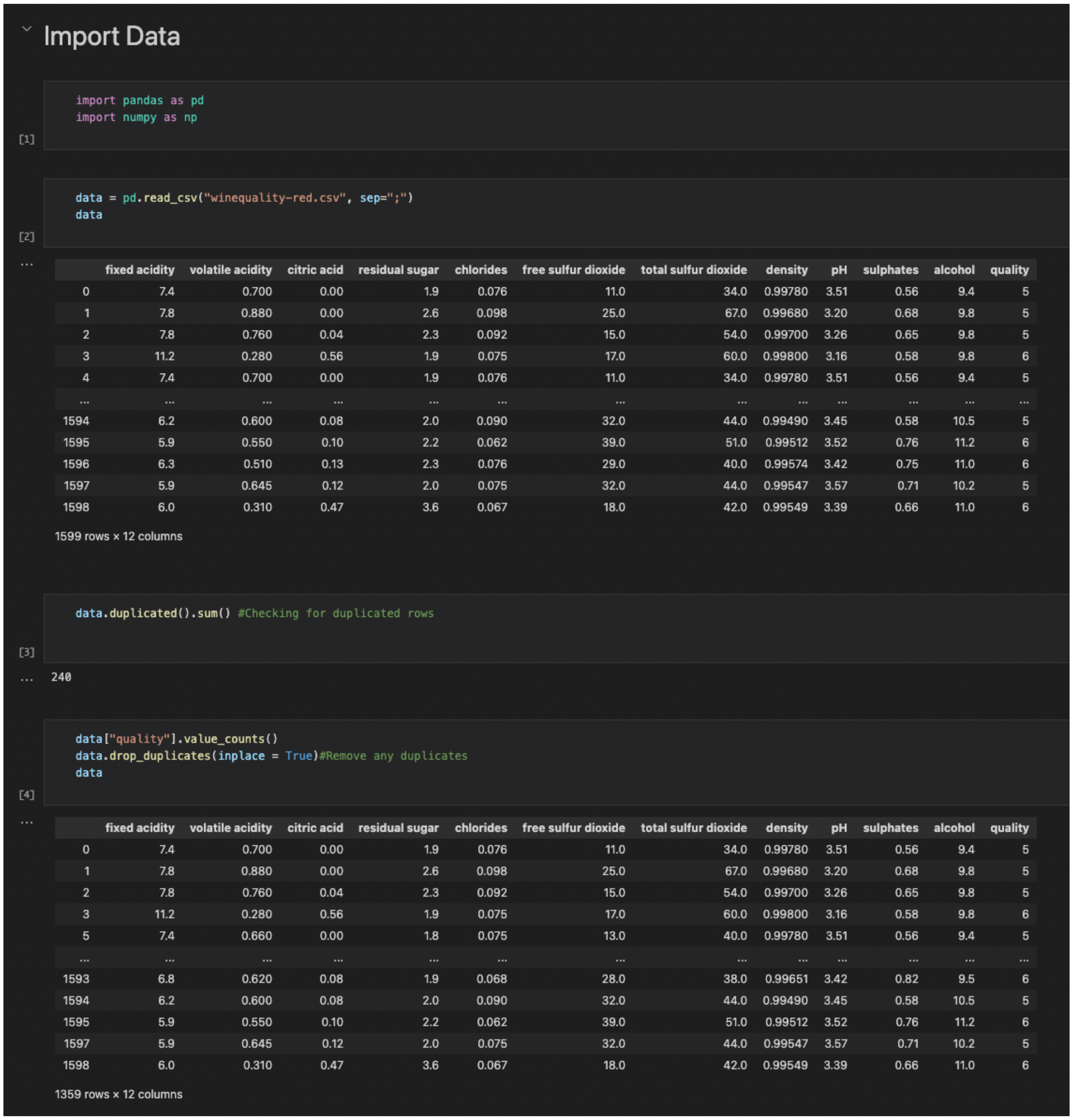
Methods: Chenthuran and Arron

Result and Analysis: Chenthuran and Arron

Conclusion: Chenthuran, Lucas, Arron

Appendix:

SVM Code:



Import Data

```
import pandas as pd
import numpy as np
```
[1]

```
data = pd.read_csv("winequality-red.csv", sep=";")
data
```
[2]

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1599 rows × 12 columns

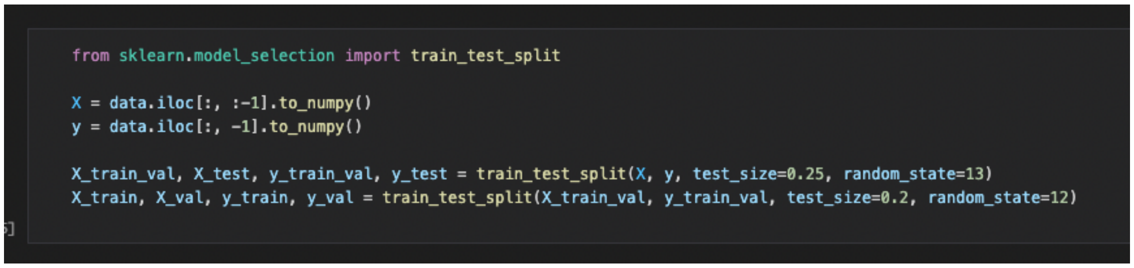
```
data.duplicated().sum() #Checking for duplicated rows
```
[3]

```
240
```

```
data["quality"].value_counts()
data.drop_duplicates(inplace = True)#Remove any duplicates
data
```
[4]

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
5	7.4	0.660	0.00	1.8	0.075	13.0	40.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1593	6.8	0.620	0.08	1.9	0.068	28.0	38.0	0.99651	3.42	0.82	9.5	6
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

1359 rows × 12 columns



```
from sklearn.model_selection import train_test_split

X = data.iloc[:, :-1].to_numpy()
y = data.iloc[:, -1].to_numpy()

X_train_val, X_test, y_train_val, y_test = train_test_split(X, y, test_size=0.25, random_state=13)
X_train, X_val, y_train, y_val = train_test_split(X_train_val, y_train_val, test_size=0.2, random_state=12)
```
5]

Build Model

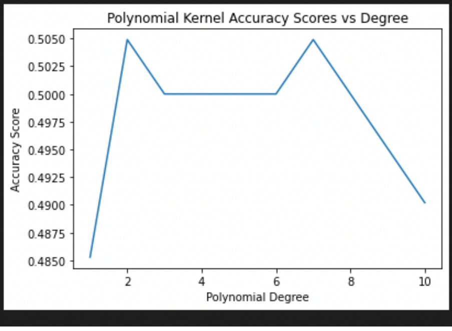
```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC

model = SVC()
params = {"C": np.linspace(0.05, 1.5, 30),
            "kernel": ['linear', 'poly', 'rbf', 'sigmoid'],
            "degree": range(1, 11)}

# search = GridSearchCV(model, params)
# search.fit(X_train, y_train)
```
[6]

Plots

```
from sklearn.model_selection import GridSearchCV
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
```
[7]

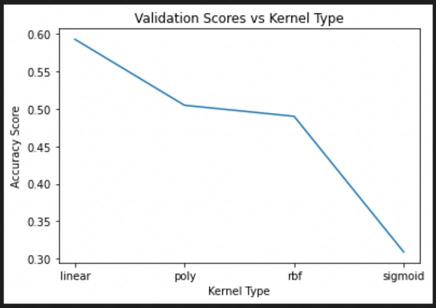
```
acc = []

for d in params["degree"]:
    y_pred = SVC(kernel="poly", degree=d).fit(X_train, y_train).predict(X_val)

    acc.append(accuracy_score(y_val, y_pred))

plt.plot(params["degree"], acc)
plt.xlabel("Polynomial Degree")
plt.ylabel("Accuracy Score")
plt.title("Polynomial Kernel Accuracy Scores vs Degree")
plt.show()
```
[8]

...

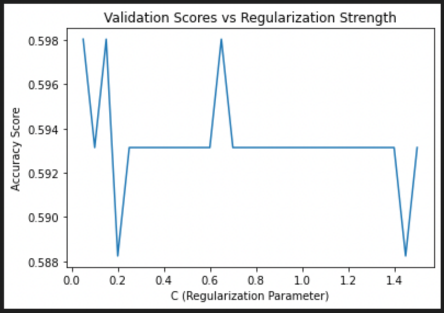


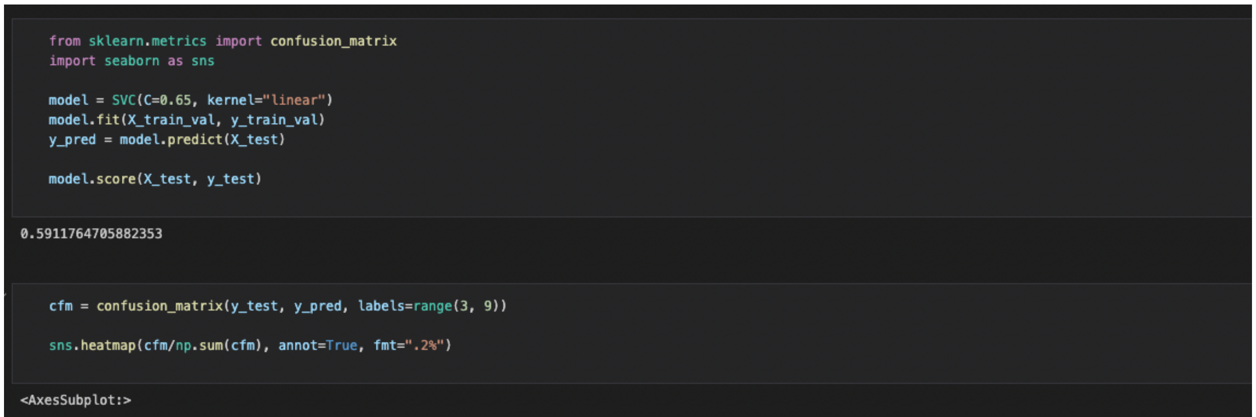
```python
acc = []

for kernel in params["kernel"]:
    y_pred = SVC(kernel=kernel, degree=2).fit(X_train, y_train).predict(X_val)

    acc.append(accuracy_score(y_val, y_pred))

plt.plot(params["kernel"], acc)
plt.xlabel("Kernel Type")
plt.ylabel("Accuracy Score")
plt.title("Validation Scores vs Kernel Type")
plt.show()
```

[11]



```python
acc = []

for c in params["C"]:
    y_pred = SVC(C=c, kernel="linear").fit(X_train, y_train).predict(X_val)

    acc.append(accuracy_score(y_val, y_pred))

plt.plot(params["C"], acc)
plt.xlabel("C (Regularization Parameter)")
plt.ylabel("Accuracy Score")
plt.title("Validation Scores vs Regularization Strength")
plt.show()
```



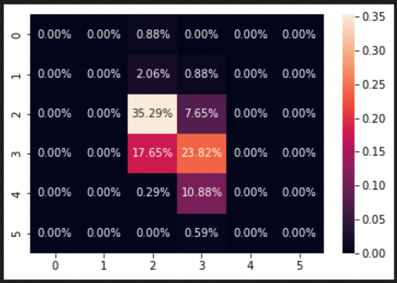


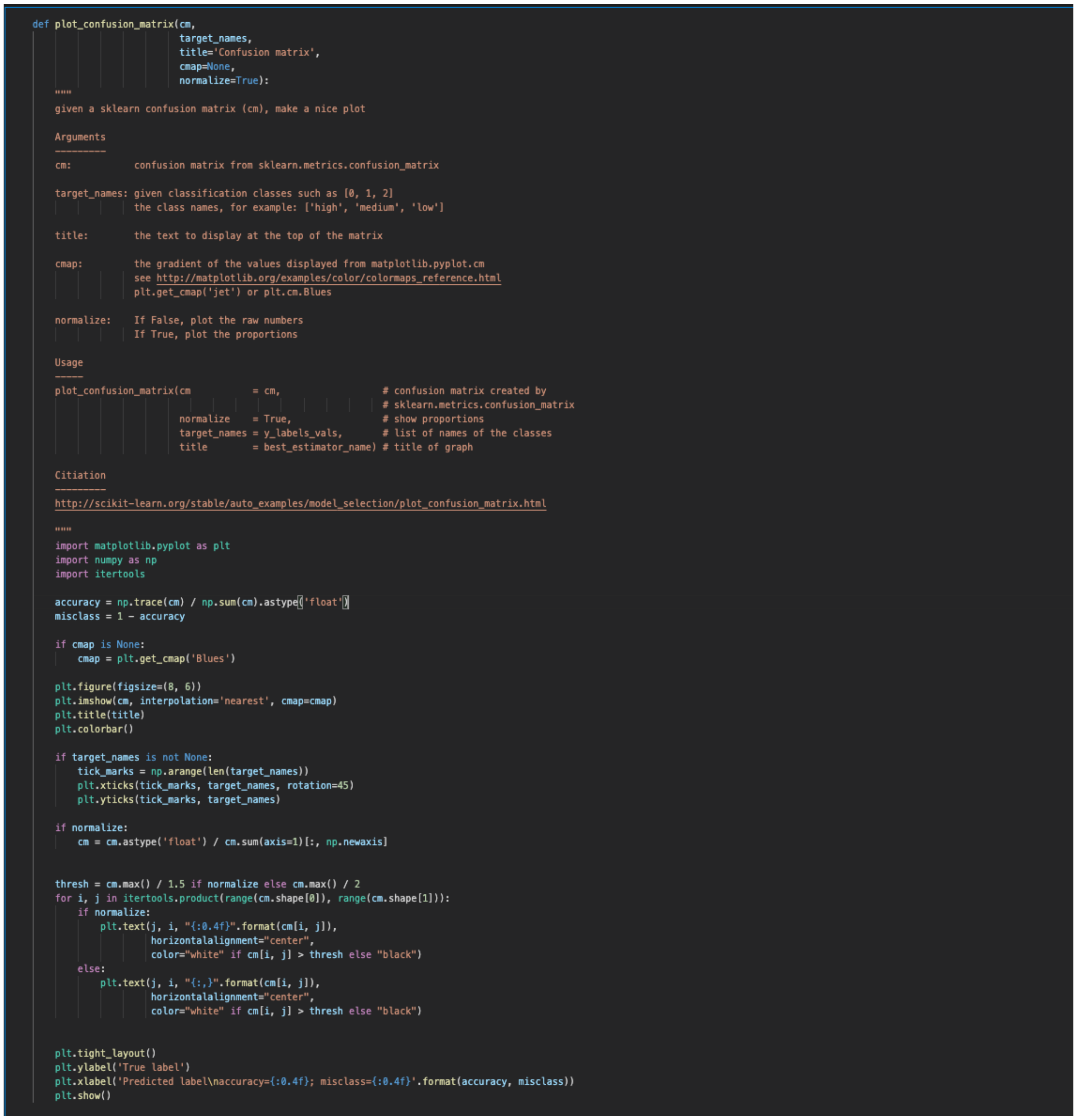
```
from sklearn.metrics import confusion_matrix
import seaborn as sns

model = SVC(C=0.65, kernel="linear")
model.fit(X_train_val, y_train_val)
y_pred = model.predict(X_test)

model.score(X_test, y_test)
```

```
0.5911764705882353
```

```
cfm = confusion_matrix(y_test, y_pred, labels=range(3, 9))

sns.heatmap(cfm/np.sum(cfm), annot=True, fmt=".2%")
```

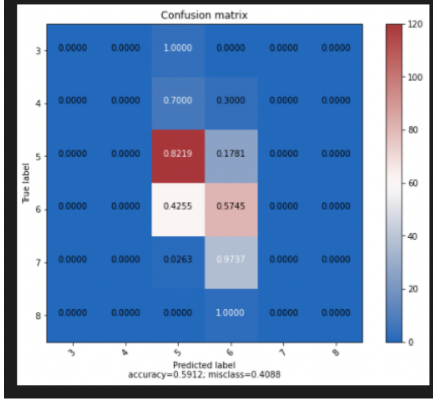
```
<AxesSubplot:>
```



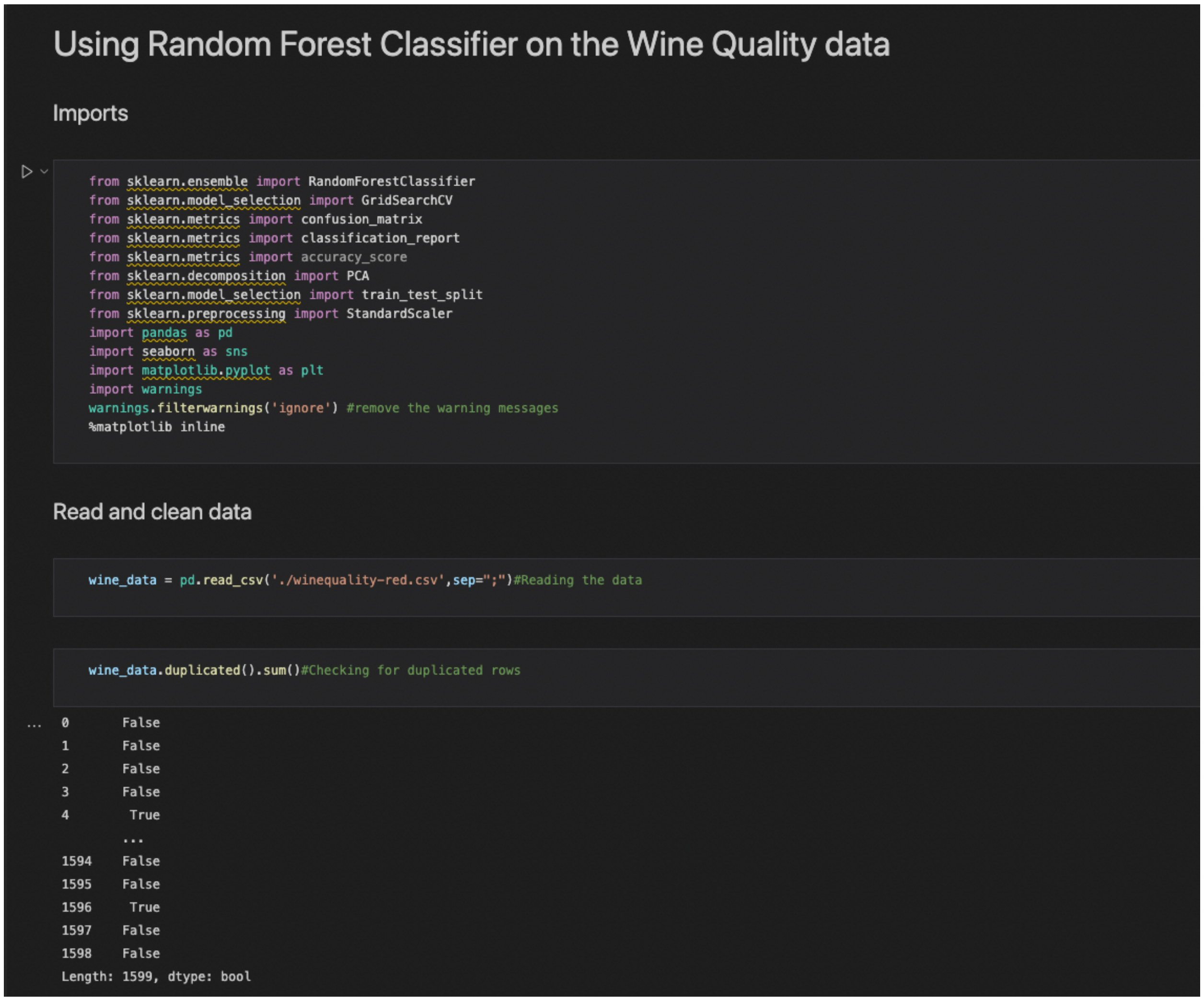


```
def plot_confusion_matrix(cm,
                          target_names,
                          title='Confusion matrix',
                          cmap=None,
                          normalize=True):
    """
    given a sklearn confusion matrix (cm), make a nice plot

    Arguments
    ---------
    cm:           confusion matrix from sklearn.metrics.confusion_matrix

    target_names: given classification classes such as [0, 1, 2]
                  the class names, for example: ['high', 'medium', 'low']

    title:        the text to display at the top of the matrix

    cmap:         the gradient of the values displayed from matplotlib.pyplot.cm
                  see http://matplotlib.org/examples/color/colormaps_reference.html
                  plt.get_cmap('jet') or plt.cm.Blues

    normalize:    If False, plot the raw numbers
                  If True, plot the proportions

    Usage
    -----
    plot_confusion_matrix(cm           = cm,                  # confusion matrix created by
                                                              # sklearn.metrics.confusion_matrix
                          normalize    = True,                # show proportions
                          target_names = y_labels_vals,       # list of names of the classes
                          title        = best_estimator_name) # title of graph

    Citiation
    ---------
    http://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

    """
    import matplotlib.pyplot as plt
    import numpy as np
    import itertools

    accuracy = np.trace(cm) / np.sum(cm).astype('float')
    misclass = 1 - accuracy

    if cmap is None:
        cmap = plt.get_cmap('Blues')

    plt.figure(figsize=(8, 6))
    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title(title)
    plt.colorbar()

    if target_names is not None:
        tick_marks = np.arange(len(target_names))
        plt.xticks(tick_marks, target_names, rotation=45)
        plt.yticks(tick_marks, target_names)

    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]


    thresh = cm.max() / 1.5 if normalize else cm.max() / 2
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        if normalize:
            plt.text(j, i, "{:0.4f}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")
        else:
            plt.text(j, i, "{:,}".format(cm[i, j]),
                     horizontalalignment="center",
                     color="white" if cm[i, j] > thresh else "black")


    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label\naccuracy={:0.4f}; misclass={:0.4f}'.format(accuracy, misclass))
    plt.show()
```

```
plot_confusion_matrix(cfm, range(3, 9), cmap="vlag")
```



Random Forest Classifier Code:



Using Random Forest Classifier on the Wine Quality data

Imports

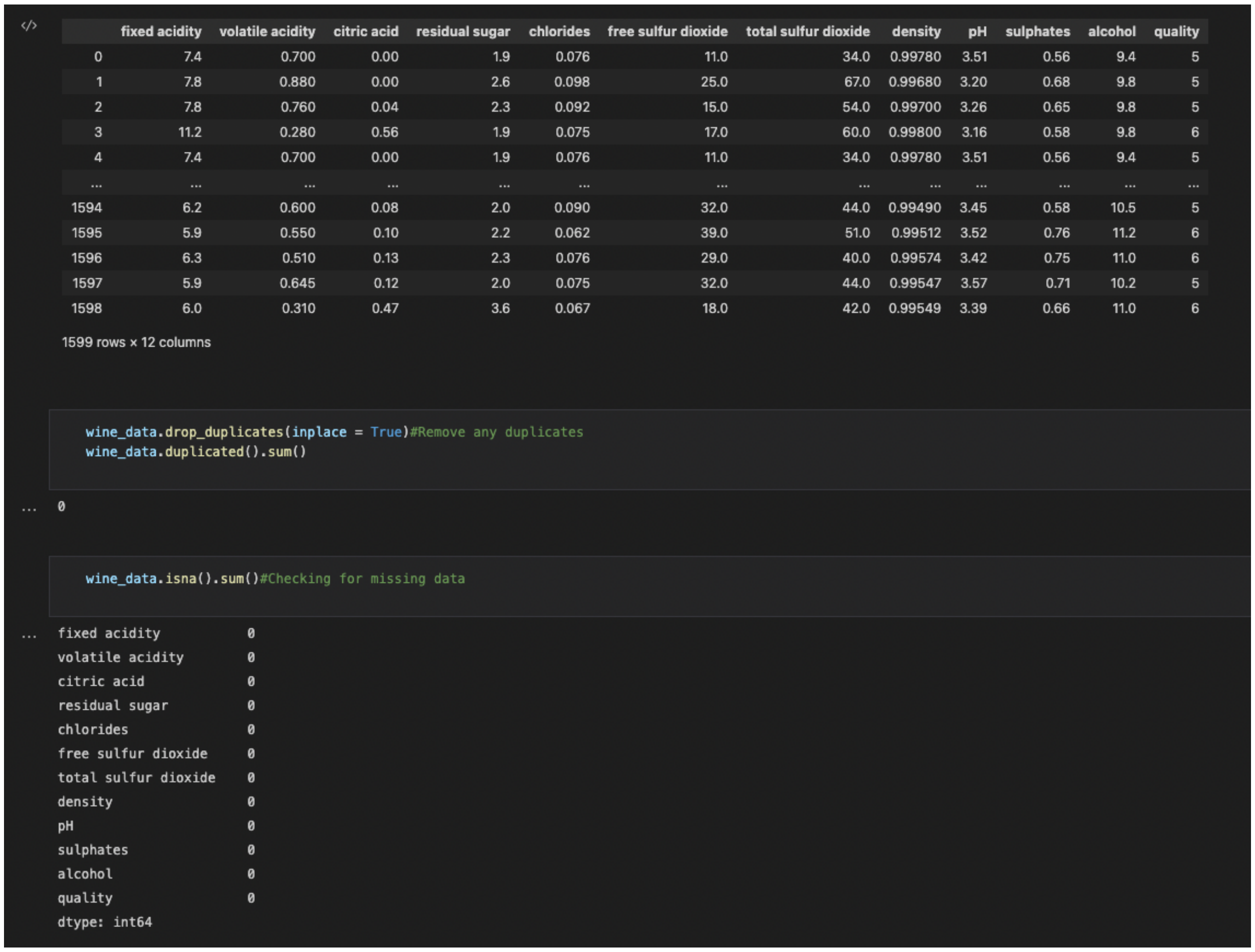
```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore') #remove the warning messages
%matplotlib inline
```

Read and clean data

```
wine_data = pd.read_csv('./winequality-red.csv',sep=";")#Reading the data
```

```
wine_data.duplicated().sum()#Checking for duplicated rows
```

```
0       False
1       False
2       False
3       False
4        True
        ...
1594    False
1595    False
1596     True
1597    False
1598    False
Length: 1599, dtype: bool
```



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
0	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
1	7.8	0.880	0.00	2.6	0.098	25.0	67.0	0.99680	3.20	0.68	9.8	5
2	7.8	0.760	0.04	2.3	0.092	15.0	54.0	0.99700	3.26	0.65	9.8	5
3	11.2	0.280	0.56	1.9	0.075	17.0	60.0	0.99800	3.16	0.58	9.8	6
4	7.4	0.700	0.00	1.9	0.076	11.0	34.0	0.99780	3.51	0.56	9.4	5
...	...	...	...	...	...	...	...	...	...	...	...	...
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10.5	5
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11.2	6
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11.0	6
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10.2	5
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11.0	6

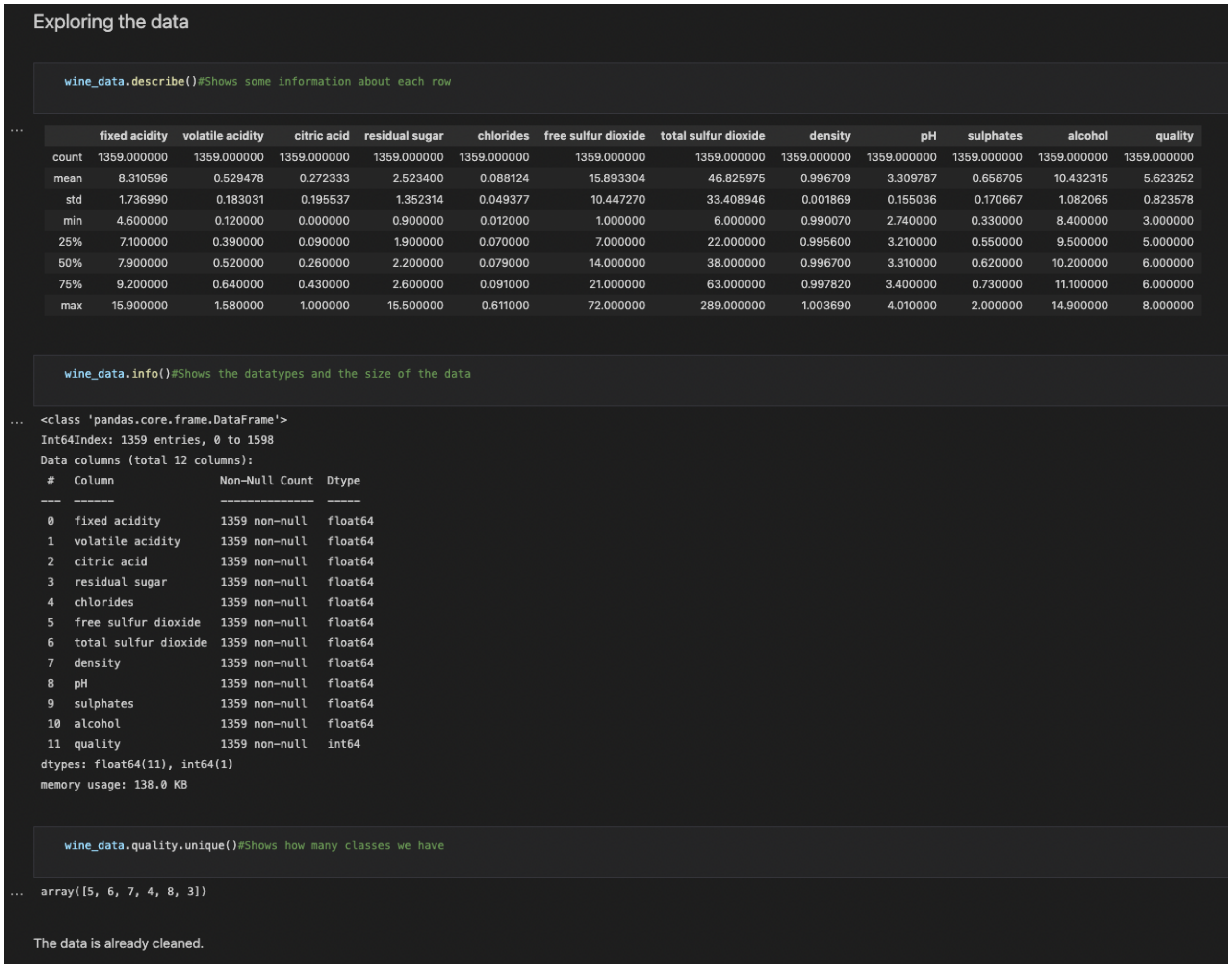
1599 rows × 12 columns

```
wine_data.drop_duplicates(inplace = True)#Remove any duplicates
wine_data.duplicated().sum()
```

```
0
```

```
wine_data.isna().sum()#Checking for missing data
```

```
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
```



Exploring the data

```
wine_data.describe()#Shows some information about each row
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
count	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000	1359.000000
mean	8.310596	0.529478	0.272333	2.523400	0.088124	15.893304	46.825975	0.996709	3.309787	0.658705	10.432315	5.623252
std	1.736990	0.183031	0.195537	1.352314	0.049377	10.447270	33.408946	0.001869	0.155036	0.170667	1.082065	0.823578
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996700	3.310000	0.620000	10.200000	6.000000
75%	9.200000	0.640000	0.430000	2.600000	0.091000	21.000000	63.000000	0.997820	3.400000	0.730000	11.100000	6.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.000000

```
wine_data.info()#Shows the datatypes and the size of the data
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 1359 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1359 non-null   float64
 1   volatile acidity      1359 non-null   float64
 2   citric acid           1359 non-null   float64
 3   residual sugar        1359 non-null   float64
 4   chlorides             1359 non-null   float64
 5   free sulfur dioxide   1359 non-null   float64
 6   total sulfur dioxide  1359 non-null   float64
 7   density               1359 non-null   float64
 8   pH                    1359 non-null   float64
 9   sulphates             1359 non-null   float64
 10  alcohol               1359 non-null   float64
 11  quality               1359 non-null   int64  
dtypes: float64(11), int64(1)
memory usage: 138.0 KB
```

```
wine_data.quality.unique()#Shows how many classes we have
```

```
array([5, 6, 7, 4, 8, 3])
```

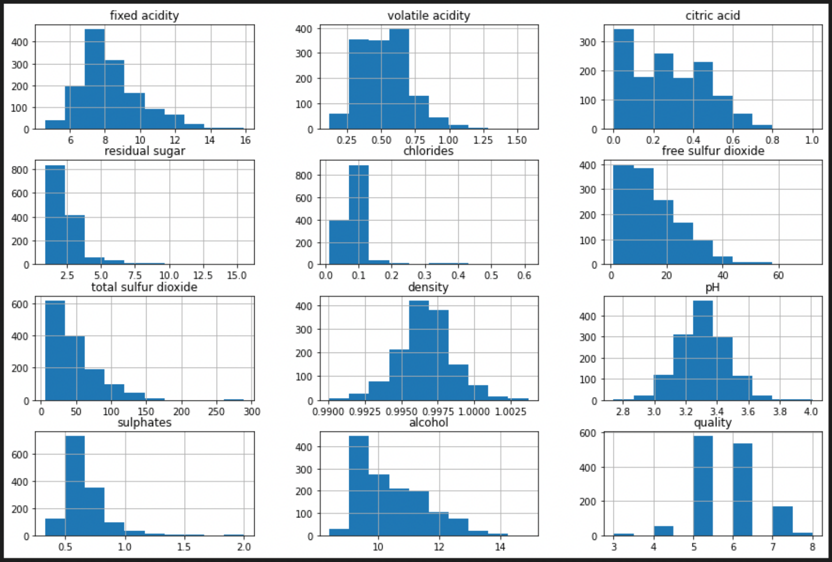
The data is already cleaned.

```
wine_data.columns[:-1]#Shows the features we have
```

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

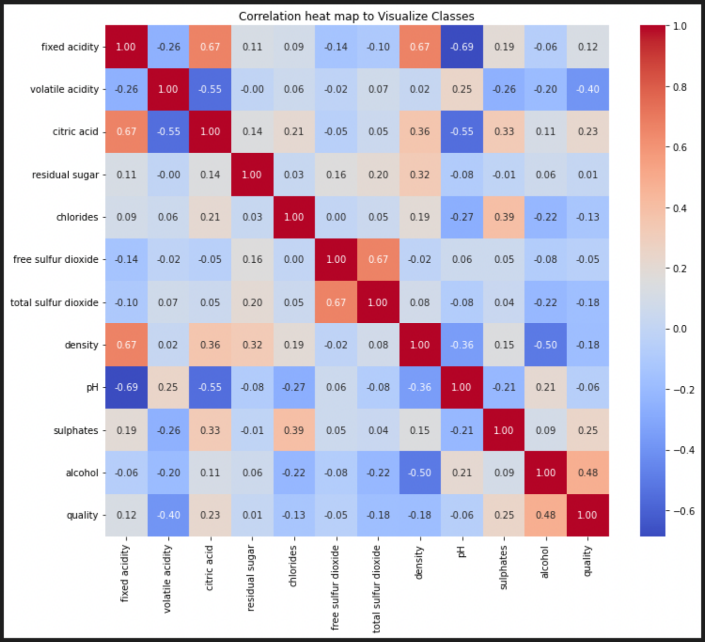
Histograms

```
wine_data.hist(figsize=(15, 10));
```



Correlation heat map

```
plt.figure(figsize=(12,10))
plt.title("Correlation heat map to Visualize Classes")
sns.heatmap(wine_data.corr(),annot=True, cmap='coolwarm',fmt='.2f');
```



	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	quality
fixed acidity	1.00	-0.26	0.67	0.11	0.09	-0.14	-0.10	0.67	-0.69	0.19	-0.06	0.12
volatile acidity	-0.26	1.00	-0.55	-0.00	0.06	-0.02	0.07	0.02	0.25	-0.26	-0.20	-0.40
citric acid	0.67	-0.55	1.00	0.14	0.21	-0.05	0.05	0.36	-0.55	0.33	0.11	0.23
residual sugar	0.11	-0.00	0.14	1.00	0.03	0.16	0.20	0.32	-0.08	-0.01	0.06	0.01
chlorides	0.09	0.06	0.21	0.03	1.00	0.00	0.05	0.19	-0.27	0.39	-0.22	-0.13
free sulfur dioxide	-0.14	-0.02	-0.05	0.16	0.00	1.00	0.67	-0.02	0.06	0.05	-0.08	-0.05
total sulfur dioxide	-0.10	0.07	0.05	0.20	0.05	0.67	1.00	0.08	-0.08	0.04	-0.22	-0.18
density	0.67	0.02	0.36	0.32	0.19	-0.02	0.08	1.00	-0.36	0.15	-0.50	-0.18
pH	-0.69	0.25	-0.55	-0.08	-0.27	0.06	-0.08	-0.36	1.00	-0.21	0.21	-0.06
sulphates	0.19	-0.26	0.33	-0.01	0.39	0.05	0.04	0.15	-0.21	1.00	0.09	0.25
alcohol	-0.06	-0.20	0.11	0.06	-0.22	-0.08	-0.22	-0.50	0.21	0.09	1.00	0.48
quality	0.12	-0.40	0.23	0.01	-0.13	-0.05	-0.18	-0.18	-0.06	0.25	0.48	1.00

Splitting the data into train and test

```
#Split data into training and test sets
X = wine_data.drop(['quality'],axis = 1)
Y = wine_data['quality']
x_train, x_test, y_train, y_test = train_test_split(X, Y, test_size = .25, random_state=42)
```

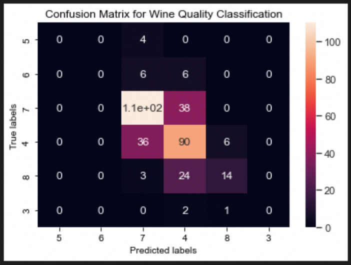
Initializing the model and the parameters for the grid search and training

```
#Initialize the classifier
winequalityclassifier = RandomForestClassifier(random_state=417)
kfold = 3

#Create a list of parameters you want to tune
param_grid_RFC = {
    'n_estimators': [ 3,4,5,6,8,10, 20, 30, 40, 50, 75, 100, 200, 400, 500, 700, 800,1000],
    'max_depth': [7,8,9,10,11,12,13,14,15,30,40,50,100,200,300],
    'criterion' : ['gini', 'entropy']
}

#Fit the model using grid search
CV_rfc = GridSearchCV(estimator=winequalityclassifier, param_grid=param_grid_RFC, cv= kfold)
CV_rfc.fit(x_train, y_train)
#Use the model with the best parameters to test it with the testing data
y_pred = CV_rfc.predict(x_test)
#Print the result of best hyperparameters
print(CV_rfc.best_params_)
```

```
{'criterion': 'gini', 'max_depth': 7, 'n_estimators': 500}
```

Plotting the confusion matrix

```
cm = confusion_matrix(y_test, y_pred)#Get the confusion matrix
df_cm = pd.DataFrame(cm)
ax= plt.subplot()
sns.set()
sns.heatmap(df_cm, annot=True, annot_kws={"size": 12})#Font size
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels');
ax.set_title('Confusion Matrix for Wine Quality Classification');
ax.xaxis.set_ticklabels(wine_data.quality.unique()); ax.yaxis.set_ticklabels(wine_data.quality.unique());
plt.show()
```



The classification report

```
print(classification_report(y_test, y_pred))#Get the classification report
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         4
           4       0.00      0.00      0.00        12
           5       0.69      0.74      0.72       148
           6       0.56      0.68      0.62       132
           7       0.67      0.34      0.45        41
           8       0.00      0.00      0.00         3

    accuracy                           0.63       340
   macro avg       0.32      0.29      0.30       340
weighted avg       0.60      0.63      0.61       340
```