SVC Implementation

1. Problem statement.

- The dataset Comprises of data for red wine quality based on different features that the wine has.
- The problem is a multiclass problem and we need to create a model that predicts the quality of the wine based on other features.
- Prediction result can be used by manufacturer to improve the quality of the wine and the consumer can use it for knowing the qality of wine

2. Data Collection.

Source:

Paulo Cortez, University of Minho, Guimarães, Portugal, http://www3.dsi.uminho.pt/pcortez (http://www3.dsi.uminho.pt/pcortez) A. Cerdeira, F. Almeida, T. Matos and J. Reis, Viticulture Commission of the Vinho Verde Region(CVRVV), Porto, Portugal @2009

Importing Libraries

In [1]:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
import pandas_profiling
import pickle
from pandas_profiling import ProfileReport
from sklearn.preprocessing import StandardScaler
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.model_selection import RandomizedSearchCV, GridSearchCV
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import ConfusionMatrixDisplay
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.metrics import accuracy_score, confusion_matrix, roc_curve, roc
import scikitplot as skl
sns.set()
```

Loading Dataset

In [2]:

```
wine = pd.read_csv("winequality-red.csv")
df = wine.copy()
df.head(10).style.background_gradient(cmap = "Reds_r")
```

Out[2]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH
0	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000
1	7.800000	0.880000	0.000000	2.600000	0.098000	25.000000	67.000000	0.996800	3.200000
2	7.800000	0.760000	0.040000	2.300000	0.092000	15.000000	54.000000	0.997000	3.260000
3	11.200000	0.280000	0.560000	1.900000	0.075000	17.000000	60.000000	0.998000	3.160000
4	7.400000	0.700000	0.000000	1.900000	0.076000	11.000000	34.000000	0.997800	3.510000
5	7.400000	0.660000	0.000000	1.800000	0.075000	13.000000	40.000000	0.997800	3.510000
6	7.900000	0.600000	0.060000	1.600000	0.069000	15.000000	59.000000	0.996400	3.300000
7	7.300000	0.650000	0.000000	1.200000	0.065000	15.000000	21.000000	0.994600	3.390000
8	7.800000	0.580000	0.020000	2.000000	0.073000	9.000000	18.000000	0.996800	3.360000
9	7.500000	0.500000	0.360000	6.100000	0.071000	17.000000	102.000000	0.997800	3.350000

Sample

In [3]:

```
df.sample(8)
```

Out[3]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
1087	7.9	0.19	0.42	1.6	0.057	18.0	30.0	0.99400	3.29	0.69	11
1358	7.4	0.64	0.17	5.4	0.168	52.0	98.0	0.99736	3.28	0.50	9
337	7.8	0.43	0.32	2.8	0.080	29.0	58.0	0.99740	3.31	0.64	10
306	7.6	0.62	0.32	2.2	0.082	7.0	54.0	0.99660	3.36	0.52	9
481	9.4	0.30	0.56	2.8	0.080	6.0	17.0	0.99640	3.15	0.92	11
1525	6.7	0.48	0.08	2.1	0.064	18.0	34.0	0.99552	3.33	0.64	9
289	11.6	0.42	0.53	3.3	0.105	33.0	98.0	1.00100	3.20	0.95	9
97	7.0	0.50	0.25	2.0	0.070	3.0	22.0	0.99630	3.25	0.63	9

Columns/ Features

In [4]:
```
df.columns
```

Out[4]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

Feature Information :

Input Features:

- **fixed acidity** : most acids involved with wine or fixed or nonvolatile (do not evaporate readily)
- **volatile acidity** : the amount of acetic acid in wine, which at too high of levels can lead to an unpleasant, vinegar taste
- **citric acid** : found in small quantities, citric acid can add 'freshness' and flavor to wines
- **residual sugar** : the amount of sugar remaining after fermentation stops, it's rare to find wines with less than 1 gram/liter and
- **chlorides** : the amount of salt in the wine
- **free sulfur dioxide** : the free form of SO2 exists in equilibrium between molecular SO2 (as a dissolved gas) and bisulfite ion; it prevents
- **total sulfur dioxide** : amount of free and bound forms of S02; in low concentrations, SO2 is mostly undetectable in wine, but at free SO2
- **density** : the density of water is close to that of water depending on the percent alcohol and sugar content
- **ph** : describes how acidic or basic a wine is on a scale from 0 (very acidic) to 14 (very basic); most wines are between 3-4 on the
- **sulphates** : a wine additive which can contribute to sulfur dioxide gas (S02) levels, wich acts as an antimicrobial and
- **alcohol** : the percent alcohol content of the wine

Output Feature :

- **quality** : scale that describes the quality of the red wine.

Basic Info

In [5]:
```
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
```

Descriptive Statistics of Numeric Variables

In [6]:

```
df.describe().T.style.background_gradient(cmap = "magma")
```

Out[6]:

	count	mean	std	min	25%	50%	75%	ma
fixed acidity	1599.000000	8.319637	1.741096	4.600000	7.100000	7.900000	9.200000	15.9000(
volatile acidity	1599.000000	0.527821	0.179060	0.120000	0.390000	0.520000	0.640000	1.5800(
citric acid	1599.000000	0.270976	0.194801	0.000000	0.090000	0.260000	0.420000	1.0000(
residual sugar	1599.000000	2.538806	1.409928	0.900000	1.900000	2.200000	2.600000	15.5000(
chlorides	1599.000000	0.087467	0.047065	0.012000	0.070000	0.079000	0.090000	0.6110(
free sulfur dioxide	1599.000000	15.874922	10.460157	1.000000	7.000000	14.000000	21.000000	72.0000(
total sulfur dioxide	1599.000000	46.467792	32.895324	6.000000	22.000000	38.000000	62.000000	289.0000(
density	1599.000000	0.996747	0.001887	0.990070	0.995600	0.996750	0.997835	1.0036(
pH	1599.000000	3.311113	0.154386	2.740000	3.210000	3.310000	3.400000	4.0100(
sulphates	1599.000000	0.658149	0.169507	0.330000	0.550000	0.620000	0.730000	2.0000(
alcohol	1599.000000	10.422983	1.065668	8.400000	9.500000	10.200000	11.100000	14.9000(
quality	1599.000000	5.636023	0.807569	3.000000	5.000000	6.000000	6.000000	8.0000(

Profiling Report

In [7]:

```
#profiling = pandas_profiling.ProfileReport(df)
#profiling
```

3. Cleaning dataset

Null Values

In [8]:
```
df.isnull().sum()
```

Out[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
```

There are **No null values** in the dataframe.

Duplicate entries

In [9]:
```
df.duplicated().sum()
```

Out[9]:
```
240
```

Observations :
There are 240 duplicates. The quality ratings for the same/similar wine were given by different wine tasters so there is a possibility of similar reviews. We can thus keep these duplicates.

4. EDA

Unique Values of Quality(Target Variable)

In [10]:
```
df['quality'].unique()
```

Out[10]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

Observations:

Target variable/Dependent variable is discrete and categorical in nature.
“quality” score scale ranges from 1 to 10; 1 being poor and 10 being the best.
1,2,9 & 10 Quality ratings are not given by any observation. Only scores obtained are between 3 to 8.

Frequency Counts of each Quality Value

In [11]:
```
df['quality'].value_counts()
```

Out[11]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

Observations:

“quality” has most values concentrated in the categories 5, 6 and 7.
Only a few observations made for the categories 3 & 8

Renaming Columns

In [12]:
```
df.rename(columns={'fixed acidity':'fixed_acidity', 'volatile acidity':'vola
       'free sulfur dioxide':'free_sulfur_dioxide', 'total sulfur dioxide':'
```

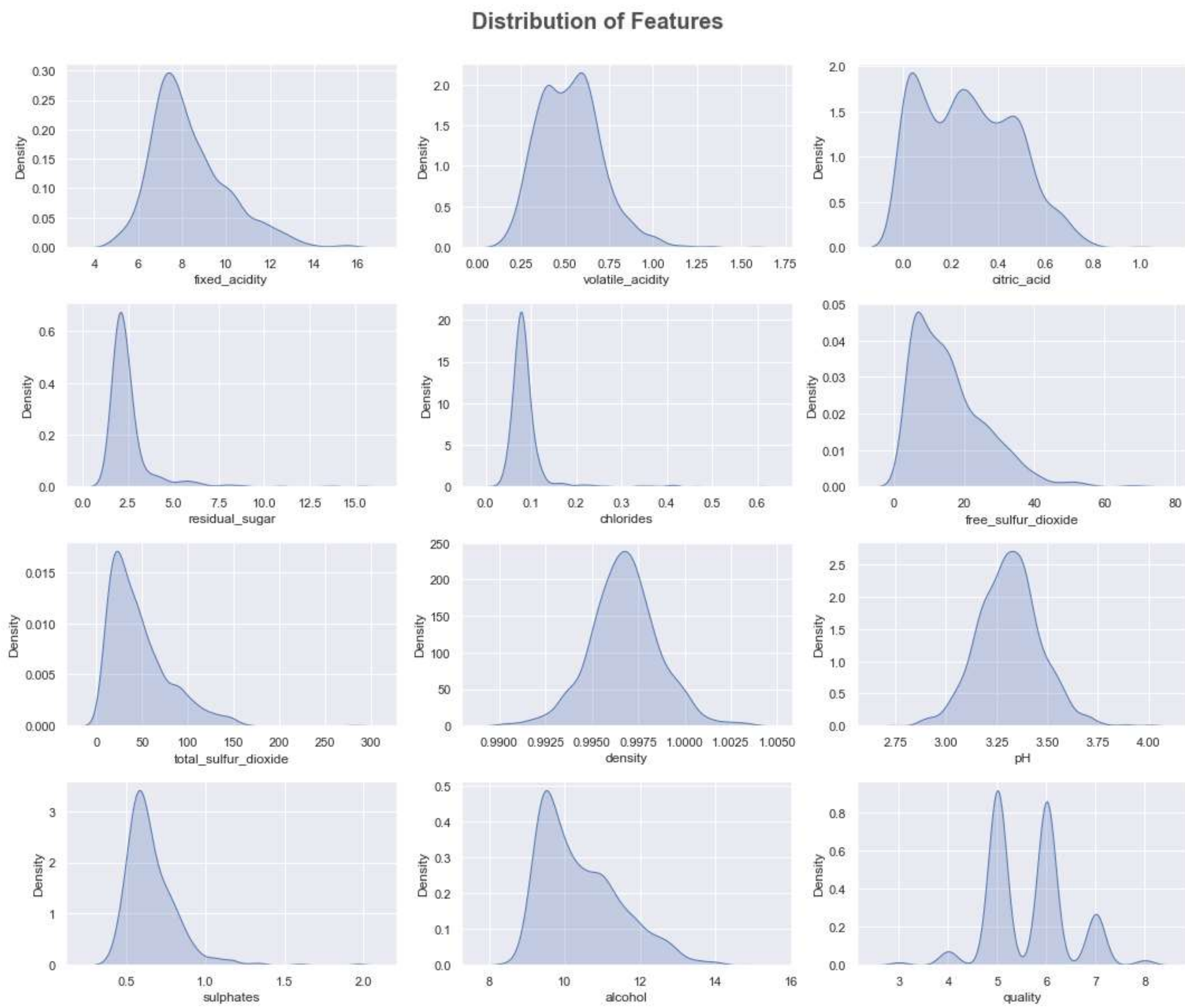
In [13]:
```
df.columns
```

Out[13]:
```
Index(['fixed_acidity', 'volatile_acidity', 'citric_acid', 'residual_sugar',
       'chlorides', 'free_sulfur_dioxide', 'total_sulfur_dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [14]:

```
plt.figure(figsize=(15, 15))
plt.suptitle('Distribution of Features', fontsize=20, fontweight='bold', alp

for i in range(0,len(df.columns)):
    plt.subplot(5, 3, i+1)
    sns.kdeplot(x=df[df.columns[i]],shade=True, color='b')
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



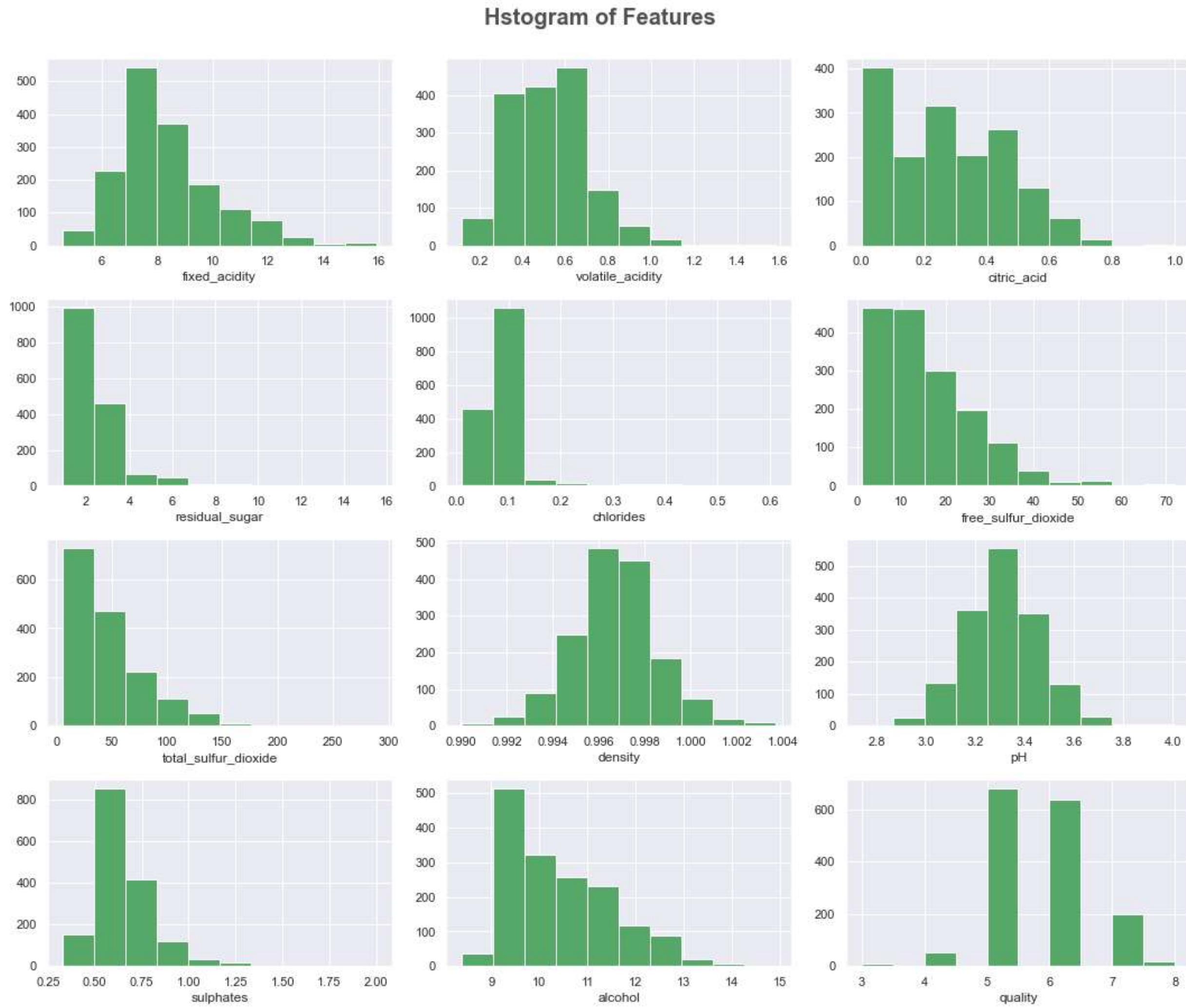
observation:

Analyzing the graphs here, it turns out that the values of the variable 'fixed_acidity' are relatively normally distributed (but a bit left skewed). But there are two peaks in the distributions of other 'volatile_acidity' and 'citric_acid' variables.Features 'residual_sugar' , 'chlorides' , 'free_sulfur_dioxide' , 'sulphates' and 'alcohol' are hoghly skewed to left , showing non normality sign. 'density' and 'ph' columns are almost normally disrtibuted.

In [15]:

```
plt.figure(figsize=(15, 15))
plt.suptitle('Hstogram of Features', fontsize=20, fontweight='bold', alpha=0

for i in range(0,len(df.columns)):
    plt.subplot(5, 3, i+1)
    plt.hist(x = df[df.columns[i]] , color = 'g')
    plt.xlabel(df.columns[i])
    plt.tight_layout()
```



Observations:

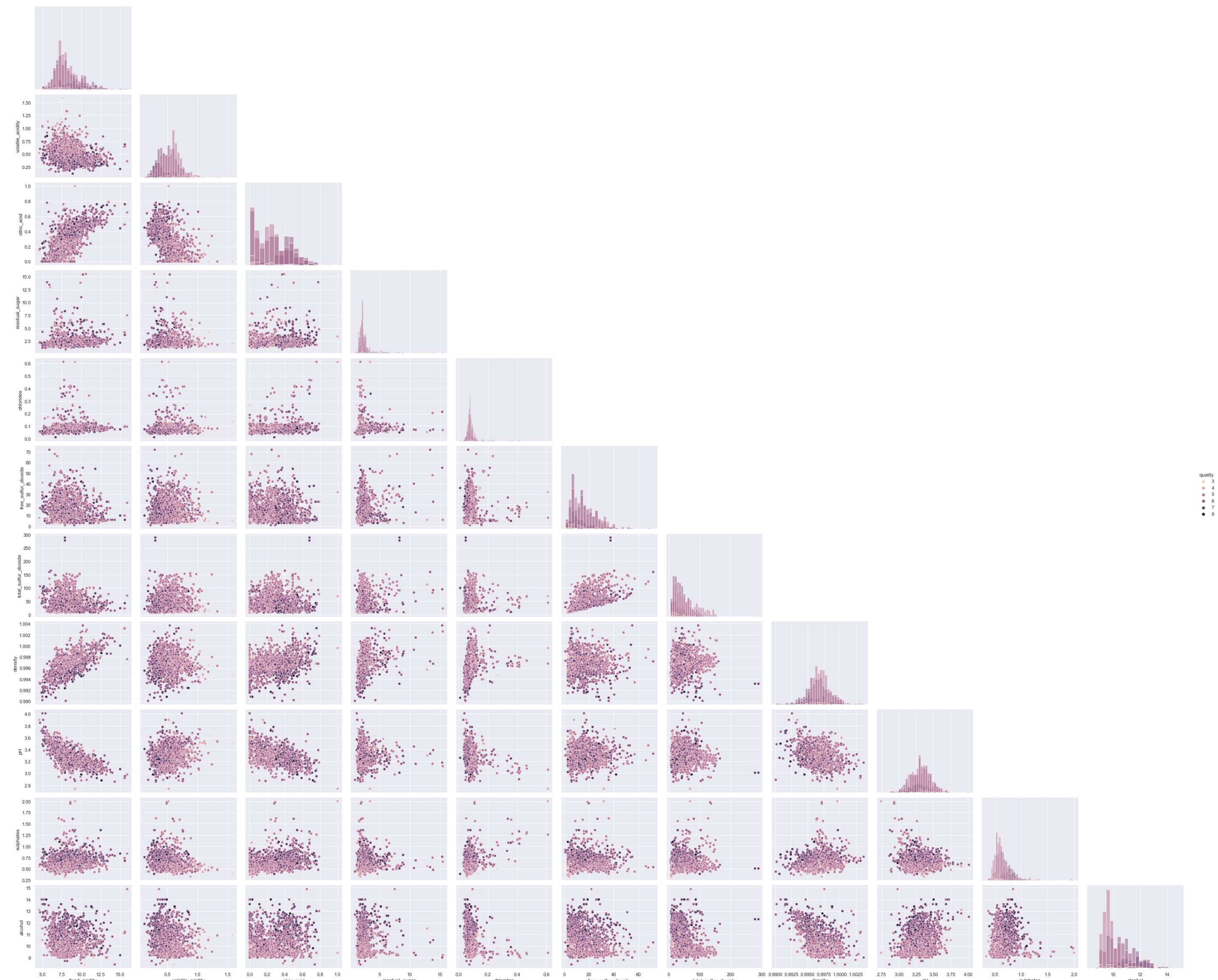
The distribution of the attribute “alcohol” seems to be positively skewed i.e the curve is shifted towards the left. The attributes 'density' and 'pH' are quite normally distributed.

Now looking at the attribute quality, we can observe that the wines with average quality (i.e. quality rating 5 to 7) are more than wines with bad(1-4) or good(8-10) quality.

Pairplot

In [16]:
```
1 sns.pairplot(df, diag_kind = "hist", hue = "quality", height = 3, aspect = 1
```

Out[16]: <seaborn.axisgrid.PairGrid at 0x1d181b49040>

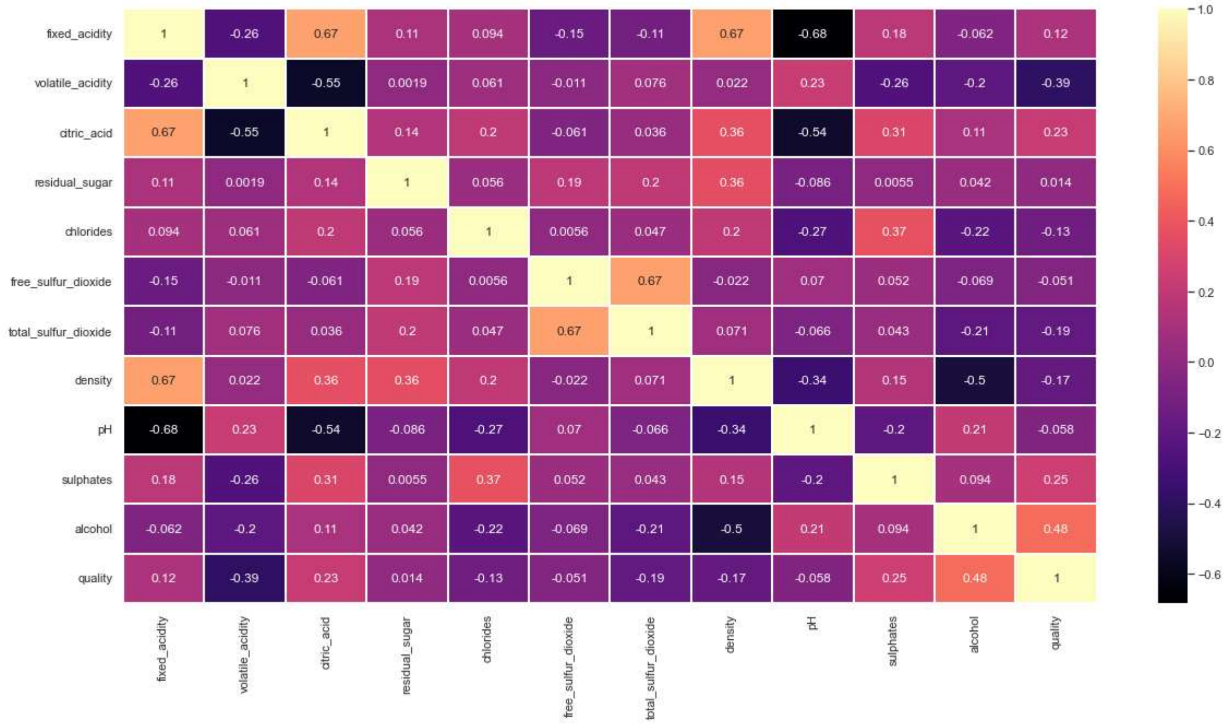


We see here that there is correlation between some variables. And this is what we don't want. This problem is called 'multicollinearity'

Heatmap for multicollinearity

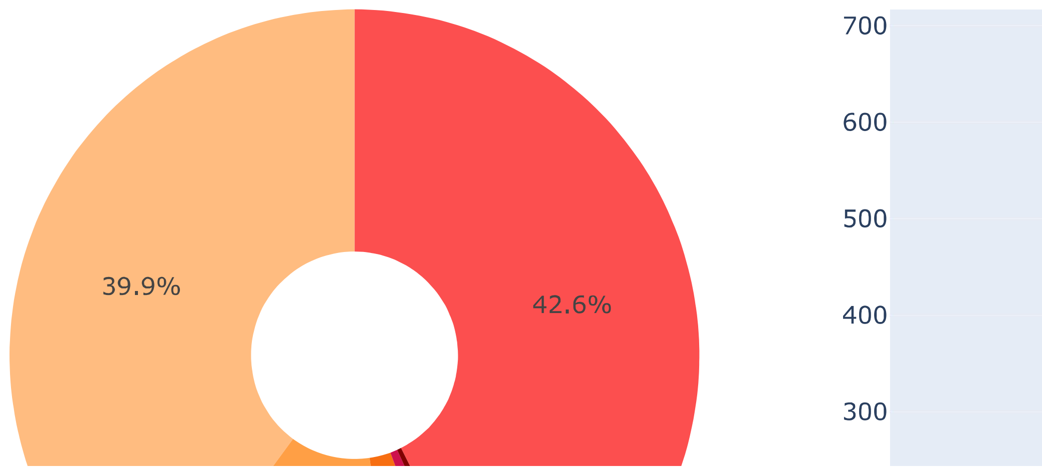
In [17]:

```
plt.figure(figsize = [20, 10], facecolor = 'white')
sns.heatmap(df.corr(), annot = True, linewidths = 2, cmap = "magma");
```



In [18]:

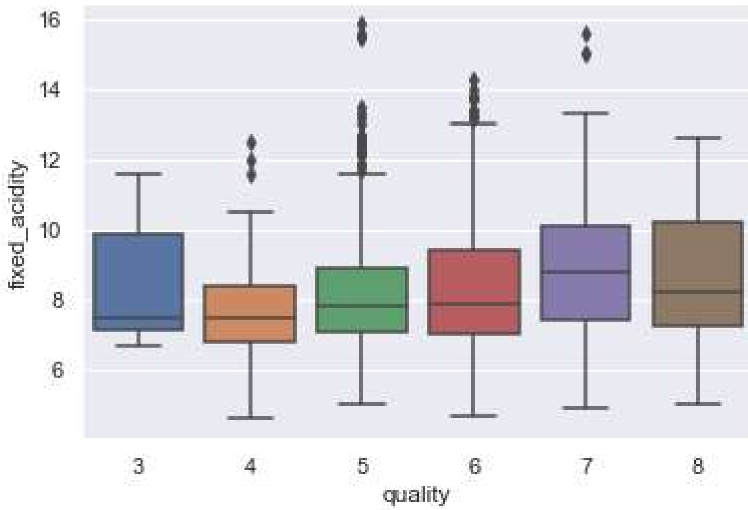
```
from plotly.subplots import make_subplots
import plotly.graph_objects as go
colors = ['#FC4F4F', '#FFBC80', '#FF9F45', '#F76E11', '#CD104D', '#820000']
quality = [3,4,5,6,7,8]
fig = make_subplots(rows = 1, cols = 2, specs = [[{"type": "pie"}, {"type":
fig.add_trace(go.Pie(values = df.quality.value_counts(), labels = df.quality
                     marker = dict(colors = colors), hole = .3, name=''), ro
fig.add_trace(go.Bar(x = df.quality.value_counts().index, y = df.quality.val
                     colorscale = colors)), row = 1, col = 2)
fig.update_layout(showlegend = False)
fig.show()
```

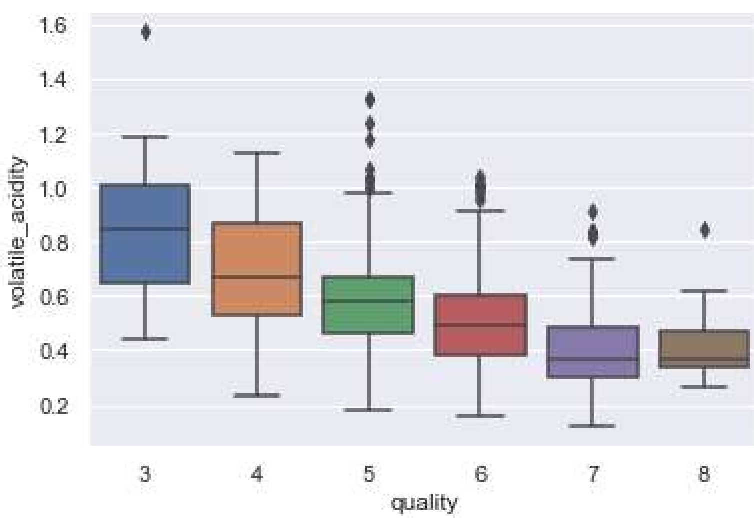


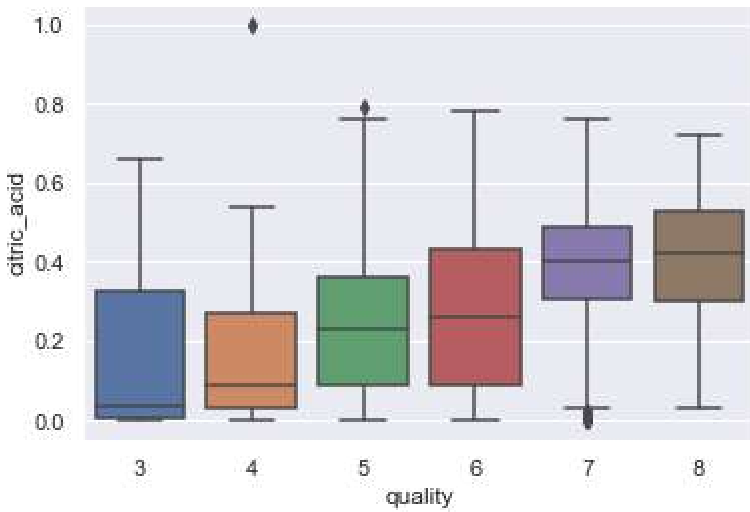
Outliers

In [19]:

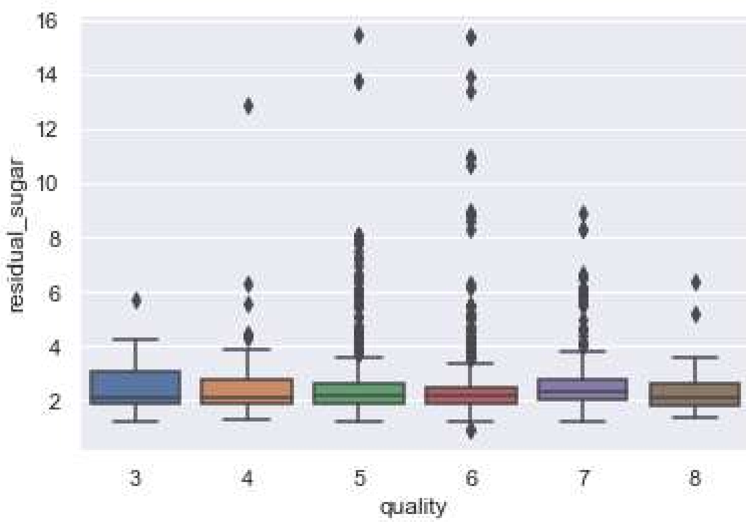
```
for i in df.columns:
    if i =="quality":
        break
    sns.boxplot("quality",i,data=df)
    plt.show()
```



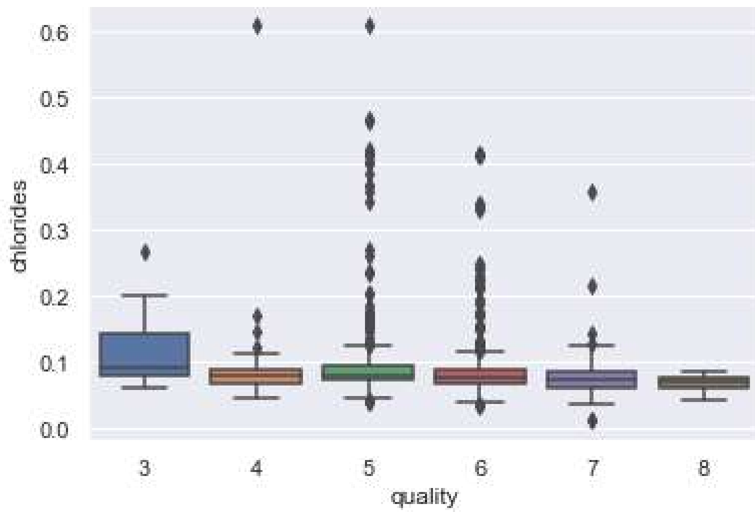


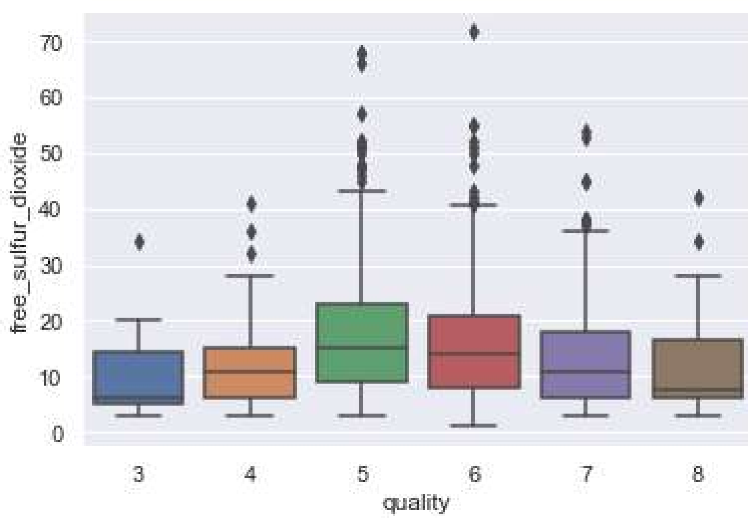


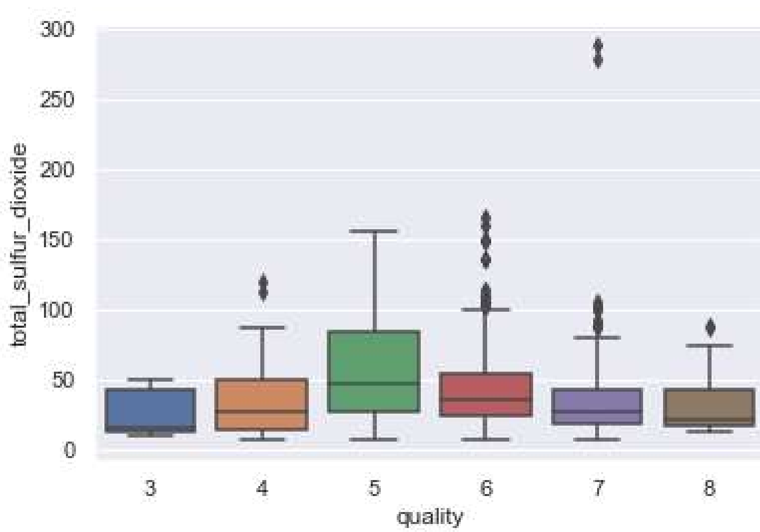
1.0
0.8
0.6
0.4
0.2
0.0
citric_acid
3
4
5
6
7
8
quality

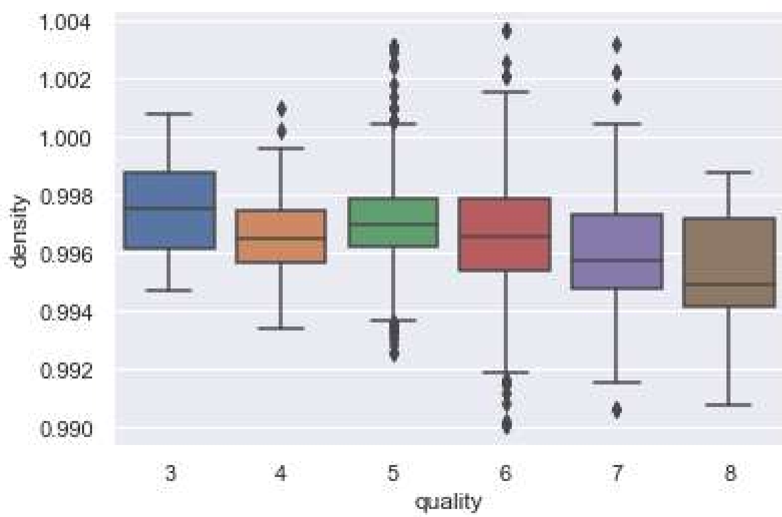


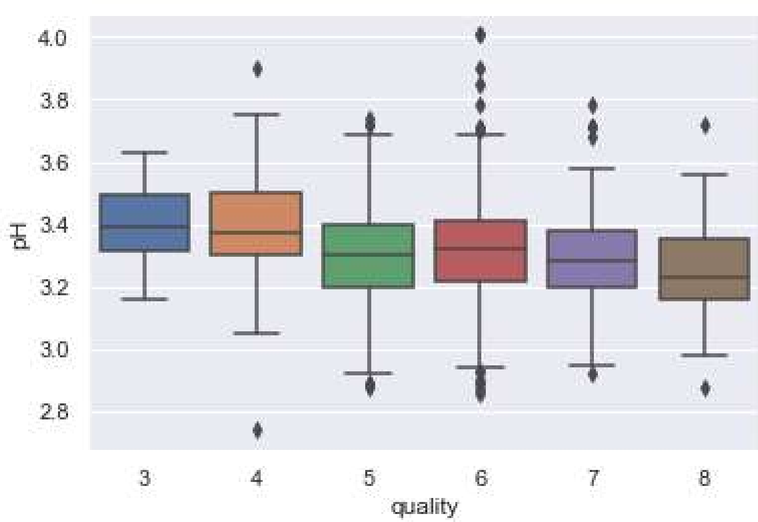
16
14
12
10
8
6
4
2
residual_sugar
3
4
5
6
7
8
quality

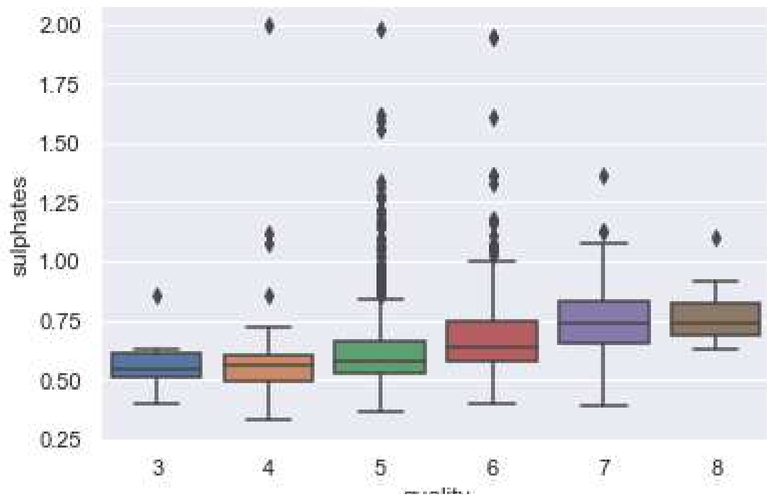

0.6
0.5
0.4
0.3
0.2
0.1
0.0
chlorides
3
4
5
6
7
8
quality

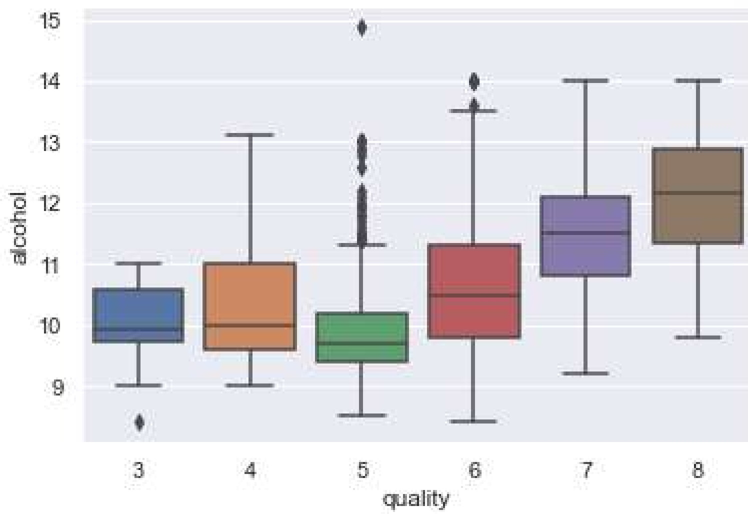

70
60
50
40
30
20
10
0
free_sulfur_dioxide
3
4
5
6
7
8
quality


300
250
200
150
100
50
0
total_sulfur_dioxide
3
4
5
6
7
8
quality


1.004
1.002
1.000
0.998
0.996
0.994
0.992
0.990
density
3
4
5
6
7
8
quality


4.0
3.8
3.6
3.4
3.2
3.0
2.8
pH
3
4
5
6
7
8
quality





5. Data Preprocessing

selecting dependent and independent columns

In [20]:

```
# we select dependent variable (label)
y = df["quality"]

# we select independent variable
x = df.drop("quality", axis = 1)
```

Splitting training and test data

In [21]:
```
x_train, x_test, y_train, y_test = train_test_split(x, y,
                                                    test_size = 0.25,
                                                    shuffle = True,
                                                    random_state = 15)
```

In [22]:
```
print("shape of train input data:",x_train.shape,"\n shape of train output d
      "\nshape of test input data ",x_test.shape,"\nshape of test output dat
```

```
shape of train input data: (1199, 11)
 shape of train output data (1199,)
shape of test input data  (400, 11)
shape of test output data (400,)
```

Feature scaling

In [23]:
```
def scaler_standard(X_train, X_test):
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    return X_train_scaled, X_test_scaled
```

In [24]:
```
x_train_scaled, x_test_scaled = scaler_standard(x_train, x_test)
```

In [25]:
```
x_train_scaled
```

Out[25]:
```
array([[-0.08984359,  0.37434335, -1.41618799, ...,  1.26371731,
        -0.0589426 , -1.06214829],
       [-0.03317738,  0.20506646, -0.28926993, ...,  0.41840095,
        -0.23984409, -0.87351189],
       [ 1.32681172, -0.52846674,  0.58153039, ..., -2.70276713,
        -0.96345004, -0.77919369],
       ...,
       [-0.37317466, -0.07706169,  0.01807136, ...,  0.74352263,
         0.48376186, -1.1564665 ],
       [-0.31650845,  0.6564715 , -0.90395251, ...,  0.41840095,
         0.18225938, -0.30760268],
       [-1.56316512,  0.54362024, -1.26251735, ...,  2.30410667,
        -0.23984409, -0.30760268]])
```

In [26]:
```
x_test_scaled
```

Out[26]:
```
array([[-0.82650436,  1.36179189, -0.49416412, ...,  0.61347396,
        -0.36044508, -0.11896627],
       [-0.03317738,  1.33357907,  0.06929491, ...,  0.93859563,
         0.12195888,  1.01285215],
       [-0.65650572,  0.54362024, -1.36496444, ...,  0.09327928,
        -0.90314954,  1.29580676],
       ...,
       [ 1.04348066, -1.65697935,  0.47908329, ..., -0.49193974,
        -1.02375053, -1.06214829],
       [-0.59983951, -1.14914868, -0.23804638, ...,  0.80854696,
        -0.23984409, -0.30760268],
       [ 0.25015368,  0.37434335, -0.49416412, ..., -1.07715875,
        -0.72224806, -0.96783009]])
```

Variance Inflation Factor

In [27]:
```
vif = pd.DataFrame()
vif["vif"] = [variance_inflation_factor(x_train_scaled,i) for i in range(x_t
vif["Features"] = x_train.columns

#Let's check the values
vif
```

Out[27]:

	vif	Features
0	8.214155	fixed_acidity
1	1.800195	volatile_acidity
2	3.300791	citric_acid
3	1.642138	residual_sugar
4	1.598401	chlorides
5	1.910315	free_sulfur_dioxide
6	2.143245	total_sulfur_dioxide
7	6.385594	density
8	3.403990	pH
9	1.512904	sulphates
10	2.931158	alcohol

6. ML Model : Support Vector Classifier

In [28]:
```
from sklearn.svm import SVC
svc_model=SVC()
```

In [29]:
```
svc_model.fit(x_train_scaled,y_train)
```

Out[29]:
```
SVC()
```

model score

In [30]:
```
svc_model.score(x_train_scaled,y_train)
```

Out[30]:
```
0.6755629691409508
```

Saving the model

In [31]:
```
import pickle
# Writing different model files to file
with open( 'svc_modelForPrediction.sav', 'wb') as f:
    pickle.dump(svc_model,f)
```

Prediction

In [32]:
```
y_pred = svc_model.predict(x_test_scaled)
```

In [33]:
```
y_pred
```

Out[33]:
```
array([5, 6, 6, 6, 6, 5, 7, 6, 5, 6, 7, 5, 6, 6, 5, 7, 5, 6, 6, 6, 6, 5,
       6, 6, 5, 5, 5, 5, 6, 6, 5, 5, 6, 5, 6, 6, 6, 6, 5, 5, 6, 6, 6, 7,
       5, 5, 5, 5, 5, 5, 5, 6, 5, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 6, 5, 6,
       7, 6, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 5, 6, 5, 6, 5, 6, 5, 6, 6, 6,
       5, 6, 6, 5, 6, 5, 7, 6, 6, 5, 6, 6, 6, 6, 6, 6, 6, 5, 5, 7, 5, 6,
       5, 7, 5, 5, 6, 5, 6, 6, 6, 6, 6, 5, 6, 5, 5, 5, 5, 6, 6, 6, 6, 6,
       5, 5, 6, 5, 6, 6, 5, 6, 5, 5, 5, 5, 6, 5, 6, 5, 5, 5, 6, 5, 6, 6,
       5, 5, 5, 6, 5, 6, 6, 6, 5, 5, 5, 6, 7, 6, 6, 5, 5, 6, 6, 5, 6, 7,
       5, 6, 5, 6, 5, 6, 5, 6, 5, 5, 5, 5, 5, 6, 6, 6, 7, 5, 6, 5, 6, 5,
       5, 6, 6, 5, 5, 5, 5, 5, 5, 5, 6, 6, 6, 6, 6, 5, 5, 5, 5, 5, 5, 6,
       6, 5, 5, 5, 6, 5, 6, 5, 5, 6, 7, 5, 6, 6, 7, 6, 5, 6, 6, 6, 6, 6,
       5, 5, 5, 5, 6, 7, 5, 5, 5, 6, 5, 6, 5, 5, 5, 6, 5, 6, 6, 6, 6, 5,
       6, 6, 6, 6, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5, 7, 5, 6, 5, 5, 5, 7, 5,
       6, 5, 5, 5, 5, 6, 6, 5, 5, 6, 5, 5, 5, 6, 6, 6, 5, 5, 5, 5, 6, 5,
       6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 6, 5, 7, 5, 5, 6, 5, 5, 5, 7, 5, 6,
       5, 5, 6, 7, 5, 6, 5, 6, 6, 5, 6, 5, 6, 6, 6, 6, 5, 5, 7, 5, 5, 5,
       6, 5, 6, 5, 5, 5, 5, 7, 6, 6, 7, 6, 6, 5, 6, 5, 6, 5, 5, 5, 5, 6,
       6, 5, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5, 6, 5, 5, 6, 5, 6, 5, 6, 5, 5,
       5, 5, 6, 5], dtype=int64)
```

Performence Metrics

In [34]:
```python
# Accuracy score
accuracy = accuracy_score(y_test,y_pred)
accuracy
```

Out[34]:
```
0.6225
```

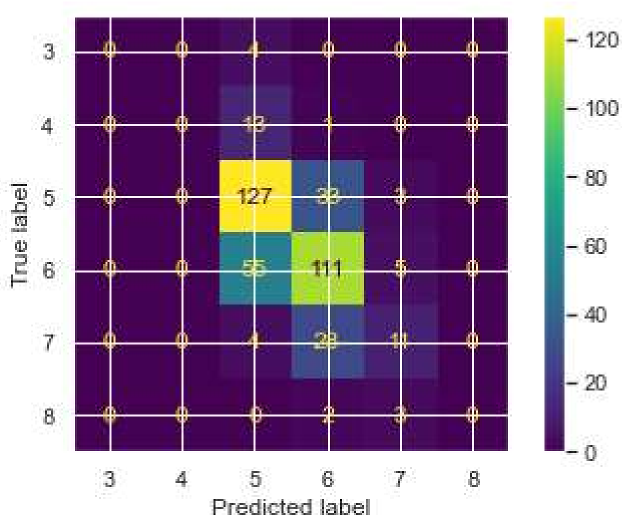
In [35]:
```python
score = accuracy_score(y_test,y_pred)
cr = classification_report(y_test,y_pred)
print("SVC ")
print ("Accuracy Score value: {:.4f}".format(score))
print (cr)
```

```
SVC
Accuracy Score value: 0.6225
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         4
           4       0.00      0.00      0.00        14
           5       0.63      0.78      0.69       163
           6       0.63      0.65      0.64       171
           7       0.50      0.26      0.34        43
           8       0.00      0.00      0.00         5

    accuracy                           0.62       400
   macro avg       0.29      0.28      0.28       400
weighted avg       0.58      0.62      0.59       400
```

In [36]:
```python
svc_cm = ConfusionMatrixDisplay.from_estimator(svc_model, x_test_scaled, y_t
```



observation:

The accuracy of our model is **62.25%**.

Hyperparameter Tuning

Grid Search CV

In [112]:
```
#parameters
param= {'C':[1.5,1.6,2,2.5],
        'kernel':["linear","rbf"],
        'gamma':[0.1,0.5,0.6,1]}
```

In [113]:
```
grid_svc= GridSearchCV(svc_model, param_grid=param, scoring='accuracy', cv=5
grid_svc.fit(x_train_scaled,y_train)
```

Out[113]:
```
GridSearchCV(cv=5, estimator=SVC(),
             param_grid={'C': [1.5, 1.6, 2, 2.5], 'gamma': [0.1, 0.5, 0.6, 1],
                         'kernel': ['linear', 'rbf']},
             scoring='accuracy')
```

Best parameters

In [114]:
```
grid_svc.best_params_
```

Out[114]:
```
{'C': 1.6, 'gamma': 1, 'kernel': 'rbf'}
```

Using the best parameters and creating new model

In [115]:
```
svc_model2=SVC(C= 1.6, gamma= 1, kernel='rbf')
svc_model2.fit(x_train_scaled,y_train)
```

Out[115]:
```
SVC(C=1.6, gamma=1)
```

In [116]:

```
# new predictions
y_pred_new=svc_model2.predict(x_test_scaled)
y_pred_new
```

Out[116]:

```
array([5, 5, 5, 6, 7, 6, 7, 6, 5, 6, 6, 5, 6, 6, 5, 7, 5, 5, 5, 7, 5, 5,
       6, 7, 5, 5, 5, 5, 6, 6, 5, 6, 6, 6, 6, 6, 6, 5, 5, 6, 6, 6, 6, 7,
       5, 5, 5, 6, 6, 6, 5, 7, 5, 6, 6, 5, 5, 5, 5, 7, 5, 5, 5, 6, 5, 6,
       7, 5, 6, 5, 5, 5, 5, 6, 5, 5, 5, 5, 6, 6, 5, 6, 6, 6, 5, 6, 5, 5,
       5, 7, 6, 6, 5, 5, 6, 6, 5, 5, 6, 6, 6, 5, 8, 6, 6, 5, 6, 6, 6, 5,
       5, 7, 5, 5, 6, 5, 7, 5, 6, 6, 6, 5, 6, 5, 5, 6, 6, 5, 6, 5, 5, 5,
       5, 5, 6, 5, 5, 6, 5, 6, 5, 5, 5, 6, 6, 5, 6, 5, 5, 5, 7, 5, 6, 5,
       6, 5, 5, 6, 6, 5, 6, 6, 5, 5, 5, 6, 7, 7, 6, 5, 5, 6, 5, 5, 7, 7,
       6, 6, 5, 5, 5, 6, 6, 6, 6, 5, 6, 5, 5, 6, 6, 6, 6, 6, 6, 5, 6, 5,
       5, 6, 6, 5, 6, 6, 5, 5, 6, 5, 6, 6, 5, 6, 6, 5, 6, 5, 5, 5, 5, 5,
       6, 5, 5, 5, 6, 5, 6, 5, 5, 6, 7, 5, 6, 6, 6, 5, 5, 6, 7, 6, 6, 5,
       6, 5, 5, 5, 5, 6, 5, 5, 6, 5, 5, 6, 6, 5, 5, 5, 5, 6, 6, 6, 6, 5,
       6, 6, 7, 6, 5, 6, 6, 5, 5, 5, 6, 6, 5, 5, 7, 5, 5, 5, 5, 5, 7, 6,
       5, 5, 5, 5, 5, 6, 6, 5, 5, 7, 5, 5, 5, 6, 5, 6, 6, 5, 6, 6, 6, 5,
       6, 6, 5, 6, 5, 5, 6, 6, 7, 6, 6, 5, 7, 5, 5, 6, 5, 5, 5, 7, 6, 6,
       5, 5, 5, 7, 5, 5, 5, 6, 6, 5, 7, 5, 6, 6, 6, 6, 5, 5, 6, 5, 5, 6,
       6, 5, 6, 5, 5, 5, 6, 7, 6, 6, 6, 7, 6, 5, 6, 5, 6, 5, 5, 5, 6, 6,
       6, 6, 5, 6, 6, 5, 5, 5, 5, 6, 5, 5, 7, 5, 5, 6, 5, 6, 5, 7, 5, 5,
       5, 6, 6, 6], dtype=int64)
```

In [117]:

```
print("Accuracy Score:",accuracy_score(y_pred_new,y_test))
print("Classification Report:\n",classification_report(y_pred_new,y_test))
print("Confusion Matrix:\n",confusion_matrix(y_pred_new,y_test))
```

```
Accuracy Score: 0.6475
Classification Report:
               precision    recall  f1-score   support

           3       0.00      0.00      0.00         0
           4       0.00      0.00      0.00         0
           5       0.79      0.65      0.71       199
           6       0.64      0.65      0.64       168
           7       0.49      0.66      0.56        32
           8       0.00      0.00      0.00         1

    accuracy                           0.65       400
   macro avg       0.32      0.33      0.32       400
weighted avg       0.70      0.65      0.67       400

Confusion Matrix:
 [[  0   0   0   0   0   0]
 [  0   0   0   0   0   0]
 [  3   8 129  52   6   1]
 [  1   6  34 109  15   3]
 [  0   0   0  10  21   1]
 [  0   0   0   0   1   0]]
```

observation:

We are able to Increase the accuracy of our model from **62.25%** to **64.75%** by Hyperparameter Tuning the model