ML_SVC Parctical Implementation on Wine_Quality Dataset

In [71]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

In [72]:
```
import pandas as pd
df=pd.read_csv(r"https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Qu
```

In [ ]:
```
```

In [73]:
```
df.head()
```

Out[73]:

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

In [74]:
```
df.tail()
```

Out[74]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
1594	6.2	0.600	0.08	2.0	0.090	32.0	44.0	0.99490	3.45	0.58	10
1595	5.9	0.550	0.10	2.2	0.062	39.0	51.0	0.99512	3.52	0.76	11
1596	6.3	0.510	0.13	2.3	0.076	29.0	40.0	0.99574	3.42	0.75	11
1597	5.9	0.645	0.12	2.0	0.075	32.0	44.0	0.99547	3.57	0.71	10
1598	6.0	0.310	0.47	3.6	0.067	18.0	42.0	0.99549	3.39	0.66	11

In [75]:
```
df.columns
```

Out[75]:
```
Index(['fixed acidity', 'volatile acidity', 'citric acid', 'residual sugar',
       'chlorides', 'free sulfur dioxide', 'total sulfur dioxide', 'density',
       'pH', 'sulphates', 'alcohol', 'quality'],
      dtype='object')
```

In [76]:
```
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 [77]:
```
df.shape
```

Out[77]:
```
(1599, 12)
```

In [78]:
```
df.quality
```

Out[78]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

In [79]:
```
df.quality.unique()
```

Out[79]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

In [80]:
```
df['quality'].value_counts()
```

Out[80]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

In [81]:
```
df.describe()
```

Out[81]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide
count	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
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000

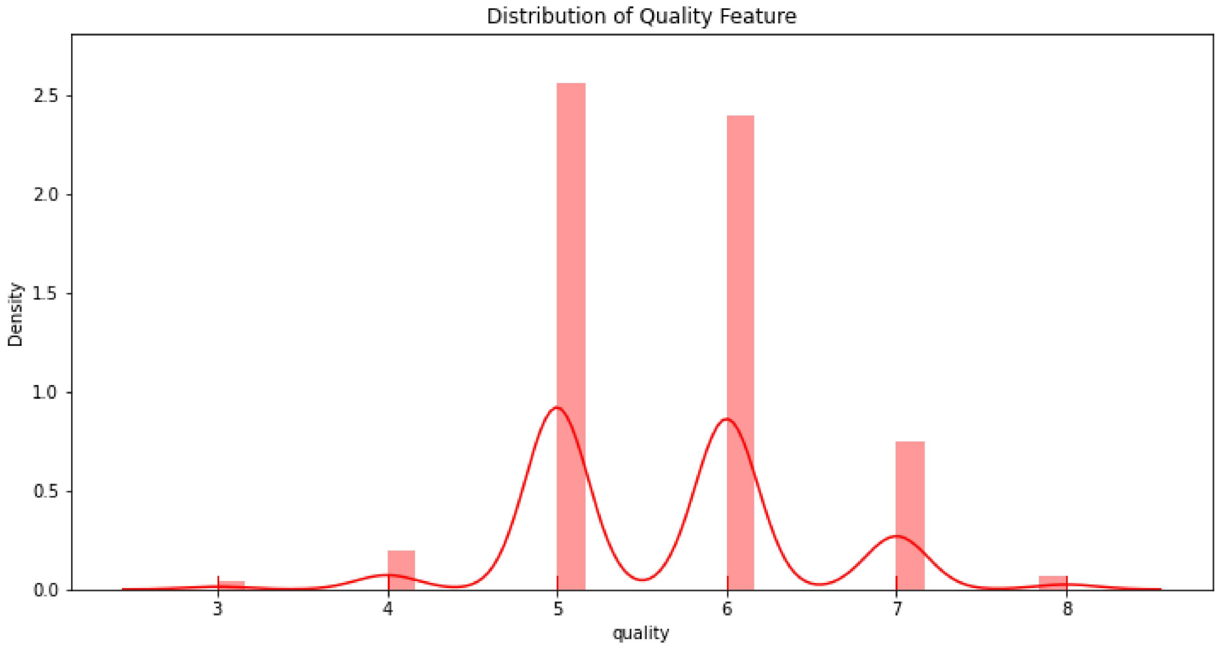
In [82]:
```
df.describe().T
```

Out[82]:

	count	mean	std	min	25%	50%	75%	max
fixed acidity	1599.0	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	1599.0	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	1599.0	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	1599.0	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	1599.0	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	1599.0	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	1599.0	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	1599.0	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	1599.0	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	1599.0	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	1599.0	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	1599.0	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

In [86]:
```
# Checking the distribution of Target Feature
plt.figure(figsize=(12,6))
plt.title('Distribution of Quality Feature')
sns.distplot(df['quality'],bins=30,rug=True,color='Red')
```

Out[86]:
```
<AxesSubplot:title={'center':'Distribution of Quality Feature'}, xlabel='qualit
y', ylabel='Density'>
```



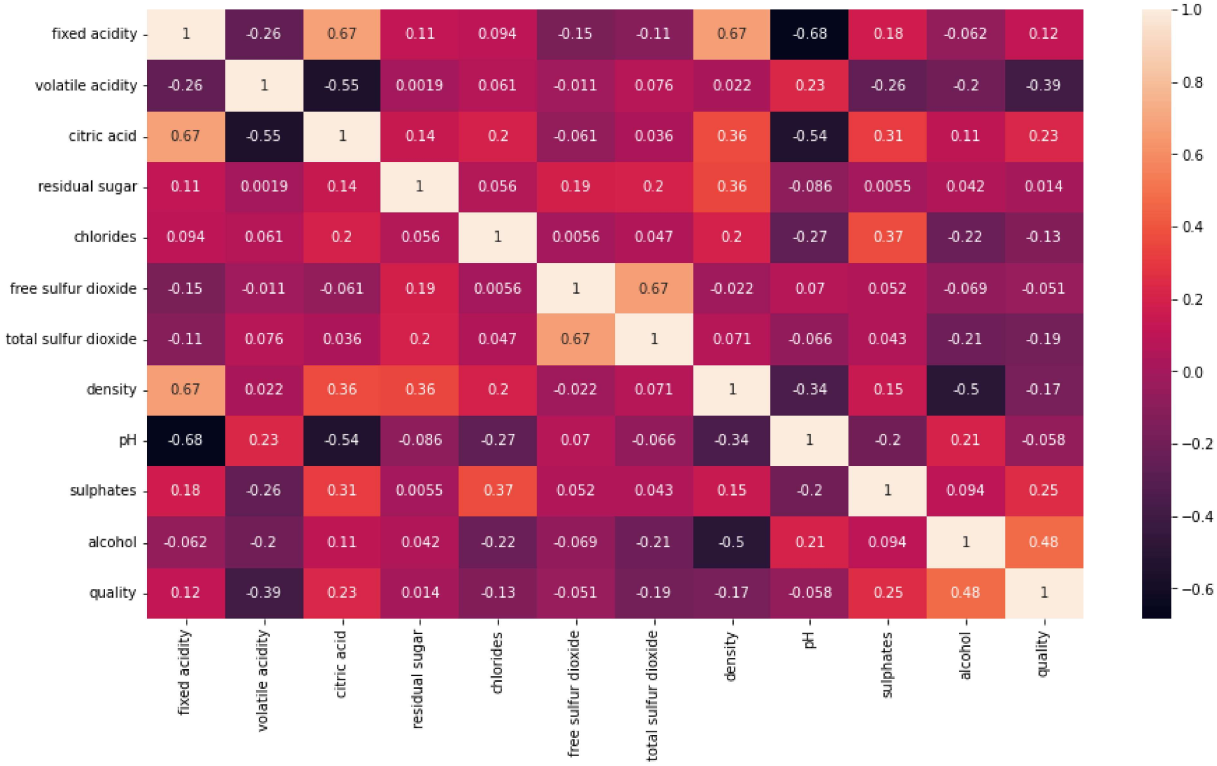
In [84]:
```
df.corr()
```

Out[84]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
fixed acidity	1.000000	-0.256131	0.671703	0.114777	0.093705	-0.153794	-0.113181	0.668047	-(
volatile acidity	-0.256131	1.000000	-0.552496	0.001918	0.061298	-0.010504	0.076470	0.022026	(
citric acid	0.671703	-0.552496	1.000000	0.143577	0.203823	-0.060978	0.035533	0.364947	-(
residual sugar	0.114777	0.001918	0.143577	1.000000	0.055610	0.187049	0.203028	0.355283	-(
chlorides	0.093705	0.061298	0.203823	0.055610	1.000000	0.005562	0.047400	0.200632	-(
free sulfur dioxide	-0.153794	-0.010504	-0.060978	0.187049	0.005562	1.000000	0.667666	-0.021946	(
total sulfur dioxide	-0.113181	0.076470	0.035533	0.203028	0.047400	0.667666	1.000000	0.071269	-(
density	0.668047	0.022026	0.364947	0.355283	0.200632	-0.021946	0.071269	1.000000	-(
pH	-0.682978	0.234937	-0.541904	-0.085652	-0.265026	0.070377	-0.066495	-0.341699	1
sulphates	0.183006	-0.260987	0.312770	0.005527	0.371260	0.051658	0.042947	0.148506	-(
alcohol	-0.061668	-0.202288	0.109903	0.042075	-0.221141	-0.069408	-0.205654	-0.496180	(
quality	0.124052	-0.390558	0.226373	0.013732	-0.128907	-0.050656	-0.185100	-0.174919	-(

In [85]:
```
plt.figure(figsize=(15,8))
sns.heatmap(df.corr(),annot=True)
```

Out[85]: <AxesSubplot:>



Creatingb the Independent Feature

In [17]:
```
x=df.drop('quality',axis=1)
```

In [ ]:

In [18]:
```
x.head()
```

Out[18]:

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

Creating the Dependent Feature

In [19]:
```
y=df['quality']
```

In [20]:
```
y
```

Out[20]:
```
0       5
1       5
2       5
3       6
4       5
       ..
1594    5
1595    6
1596    6
1597    5
1598    6
Name: quality, Length: 1599, dtype: int64
```

Creating SVM Model Training

In [88]:
```
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.33,random_state=35
```

In [89]:
```
x_test.head()
```

Out[89]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcoh
1536	6.1	0.53	0.08	1.9	0.077	24.0	45.0	0.99528	3.60	0.68	10
617	11.5	0.31	0.51	2.2	0.079	14.0	28.0	0.99820	3.03	0.93	9
660	7.2	0.52	0.07	1.4	0.074	5.0	20.0	0.99730	3.32	0.81	9
657	12.0	0.50	0.59	1.4	0.073	23.0	42.0	0.99800	2.92	0.68	10
1489	6.2	0.57	0.10	2.1	0.048	4.0	11.0	0.99448	3.44	0.76	10

In [32]:
```
y_train.head()
```

Out[32]:
```
548     6
355     6
1296    5
209     7
140     5
Name: quality, dtype: int64
```

Standardizing the model

In [92]:
```
from sklearn.preprocessing import StandardScaler
scaler =StandardScaler()
x_train=scaler.fit_transform(x_train)
x_test=scaler.transform(x_test)
```

In [93]:
```
scaler=StandardScaler()
```

In [94]:
```
scaler.fit(x_train)
```

Out[94]:
```
StandardScaler()
```

In [95]:
```
print(scaler.mean_)
```

```
[-3.98063157e-16 -2.23910526e-16 -3.53281052e-16  1.54249473e-16
 -8.79056139e-17 -8.29298244e-19  1.65859649e-17 -2.57059650e-14
 -3.25748350e-15 -1.22736140e-16 -5.67239999e-16]
```

In [96]:
```
x_train_tf=scaler.transform(x_train)
```

In [97]:
```
x_train_tf
```

Out[97]:
```
array([[-0.29458893, -0.07144684, -0.10770066, ...,  0.77756879,
        -0.03112232, -1.29807621],
       [ 0.227194  ,  0.50492002, -0.57281118, ..., -1.12824789,
        -0.48461901,  0.56762073],
       [-0.4105407 , -0.7850439 ,  0.20237302, ...,  1.36902914,
        -0.03112232, -0.64508228],
       ...,
       [-0.4105407 , -0.62036765, -0.10770066, ..., -1.06253008,
        -0.48461901, -1.20479137],
       [-0.00470953, -1.38885679,  1.0809151 , ..., -0.33963409,
        -0.20118358,  1.87360859],
       [-0.99029951,  0.53236606, -1.39967431, ..., -0.73394099,
        -1.16486404, -0.83165198]])
```

In [98]:
```
x_test
```

Out[98]:
```
array([[-1.28017892, -0.01655475, -0.98624274, ...,  1.89477167,
         0.13893894, -0.0853732 ],
       [ 1.85051868, -1.22418055,  1.23595194, ..., -1.85114387,
         1.55611608, -0.55179744],
       [-0.64244422, -0.07144684, -1.03792169, ...,  0.05467281,
         0.87587105, -0.73836713],
       ...,
       [-0.17863717,  0.47747398,  0.30573091, ..., -0.01104501,
        -0.42793192,  0.28776619],
       [ 0.40112164,  1.38319332, -0.15937961, ..., -0.14248064,
         0.08225185, -0.36522774],
       [ 0.74897693, -0.181231  ,  0.46076775, ..., -0.86537663,
         0.13893894,  0.47433588]])
```

Train the Support Vector Classifier Without Hyperparameter Tuning

In [101]:
```
# fitting Kernal SVM  to the Training Set
from sklearn.svm import SVC
from sklearn.metrics import classification_report , confusion_matrix

# train the model on train set
model=SVC()
model.fit(x_train,y_train)

# print prediction
predictions=model.predict(x_test)
print(classification_report(y_test , predictions))
```

```
              precision    recall  f1-score   support

           3       0.00      0.00      0.00         2
           4       0.00      0.00      0.00        21
           5       0.68      0.70      0.69       220
           6       0.55      0.72      0.62       211
           7       0.55      0.25      0.34        64
           8       0.00      0.00      0.00        10

    accuracy                           0.61       528
   macro avg       0.30      0.28      0.28       528
weighted avg       0.57      0.61      0.58       528
```

```
Observations:
Here Accuracy of Model is 60%
```

Applying GridSearchCV for better Accuracy

In [105]:
```
from sklearn.model_selection import GridSearchCV

# defining parameter range
param_grid={'C':[0.1,1,10,100,1000],
            'gamma':[1,0.1,0.01,0.001,0.0001],
            'kernel':['rbf']}

grid=GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(x_train, y_train)
```

```
0.0s
[CV 3/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.435 total time=
0.0s
[CV 4/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.430 total time=
0.1s
[CV 5/5] END ........C=0.1, gamma=1, kernel=rbf;, score=0.430 total time=
0.0s
[CV 1/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.586 total time=
0.0s
[CV 2/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.636 total time=
0.0s
[CV 3/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.607 total time=
0.0s
[CV 4/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.612 total time=
0.0s
[CV 5/5] END ......C=0.1, gamma=0.1, kernel=rbf;, score=0.556 total time=
0.0s
[CV 1/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.544 total time=
0.0s
[CV 2/5] END .....C=0.1, gamma=0.01, kernel=rbf;, score=0.598 total time=
```

In [107]:
```
# print best parameterafter tuning
print(grid.best_params_)

#print how our model looks after hyper-parameter tuning
print(grid.best_estimator_)
```

```
{'C': 1, 'gamma': 0.1, 'kernel': 'rbf'}
SVC(C=1, gamma=0.1)
```

In [ ]: