Support Vector Classifier Practical Implementation - Wine Quality Dataset

Linkedin: https://www.linkedin.com/in/satya-nerurkar-9b0655190/ (https://www.linkedin.com/in/satya-nerurkar-9b0655190/)
Github: https://github.com/SatyaNerurkar (https://github.com/SatyaNerurkar)

In [2]:
```
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

Data ingestion

In [3]:
```
raw_df = pd.read_csv("https://raw.githubusercontent.com/aniruddhachoudhury/Red-Wine-Quality/master/winequality-red.csv")
```

Profile of the data

In [3]:
```
raw_df.head()
```

Out[3]:

	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

In [4]:
```
print("This dataset contains {} rows and {} columns.".format(raw_df.shape[0],raw_df.shape[1]))
```

```
This dataset contains 1599 rows and 12 columns.
```

In [5]:
```
raw_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 [6]:
```
raw_df['quality'].value_counts()
```

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

In [7]:
```
raw_df.describe()
```

Out[7]:

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	sulphates	alcohol	c
count	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.000000	1599.0
mean	8.319637	0.527821	0.270976	2.538806	0.087467	15.874922	46.467792	0.996747	3.311113	0.658149	10.422983	5.6
std	1.741096	0.179060	0.194801	1.409928	0.047065	10.460157	32.895324	0.001887	0.154386	0.169507	1.065668	0.8
min	4.600000	0.120000	0.000000	0.900000	0.012000	1.000000	6.000000	0.990070	2.740000	0.330000	8.400000	3.0
25%	7.100000	0.390000	0.090000	1.900000	0.070000	7.000000	22.000000	0.995600	3.210000	0.550000	9.500000	5.0
50%	7.900000	0.520000	0.260000	2.200000	0.079000	14.000000	38.000000	0.996750	3.310000	0.620000	10.200000	6.0
75%	9.200000	0.640000	0.420000	2.600000	0.090000	21.000000	62.000000	0.997835	3.400000	0.730000	11.100000	6.0
max	15.900000	1.580000	1.000000	15.500000	0.611000	72.000000	289.000000	1.003690	4.010000	2.000000	14.900000	8.0

In [8]:
```
raw_df.quality.unique()
```

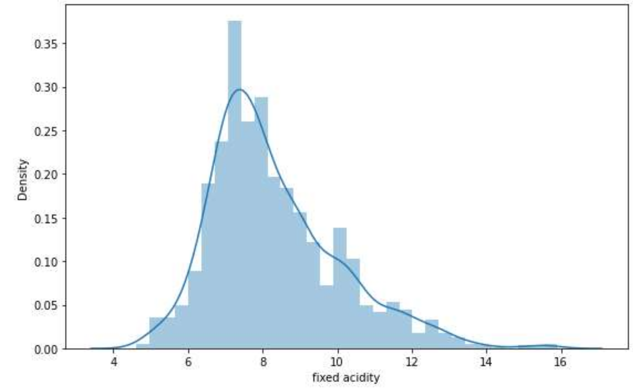
Out[8]:
```
array([5, 6, 7, 4, 8, 3], dtype=int64)
```

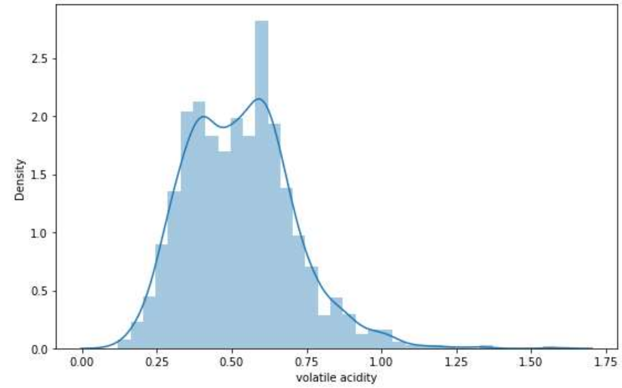
In [9]:
```
raw_df['quality'].value_counts()
```

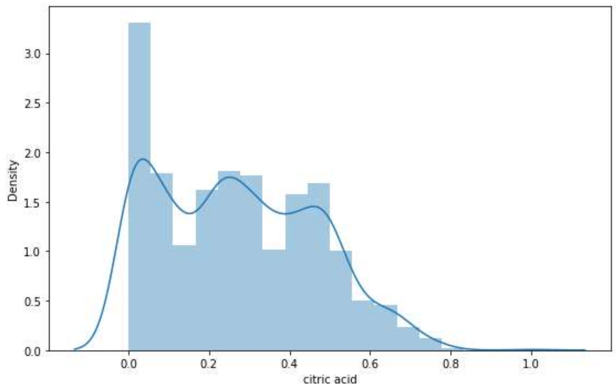
Out[9]:
```
5    681
6    638
7    199
4     53
8     18
3     10
Name: quality, dtype: int64
```

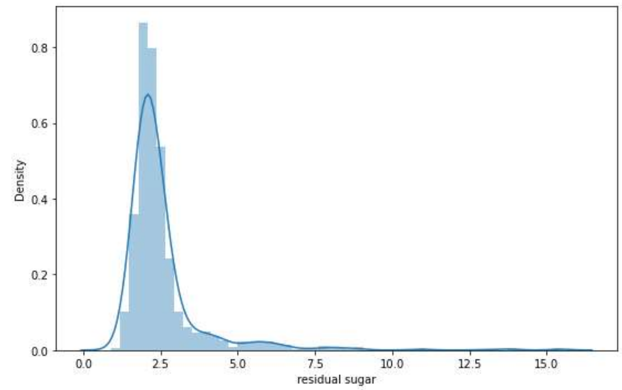
Exploratory Data Analysis

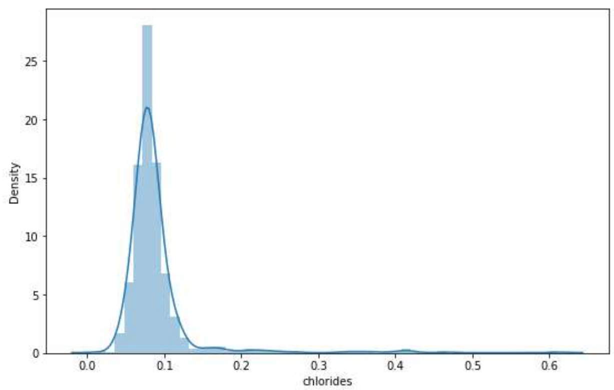
In [10]:
```
cols = raw_df.columns
plt.figure(figsize=(20,40), facecolor='white')

for i in range(0, len(cols)):
    plt.subplot(6, 2, i+1)
    sns.distplot(x=raw_df[cols[i]],kde=True)
    plt.xlabel(cols[i])
```

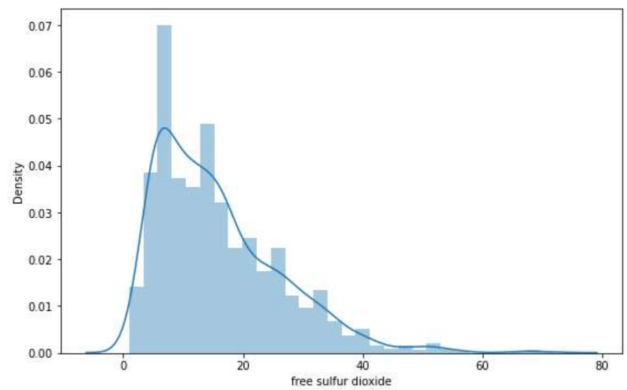

Density
fixed acidity

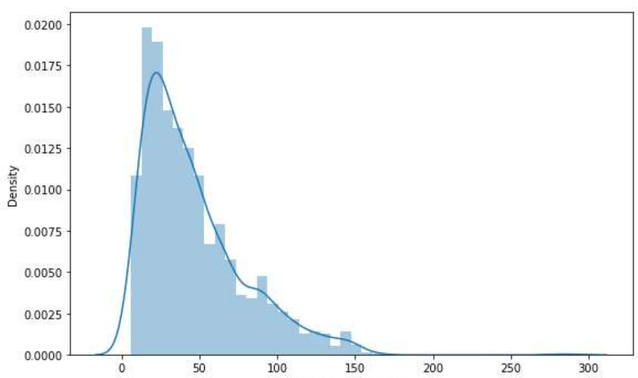

Density
volatile acidity

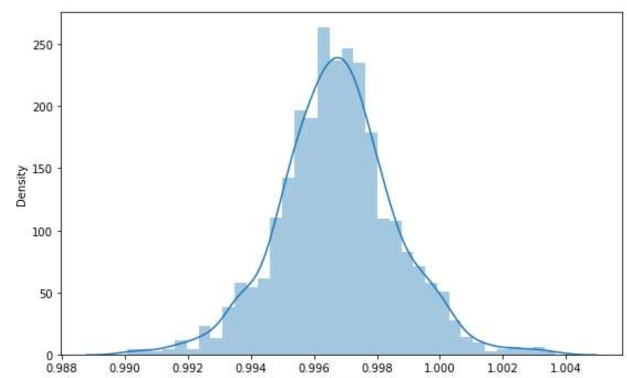

Density
citric acid

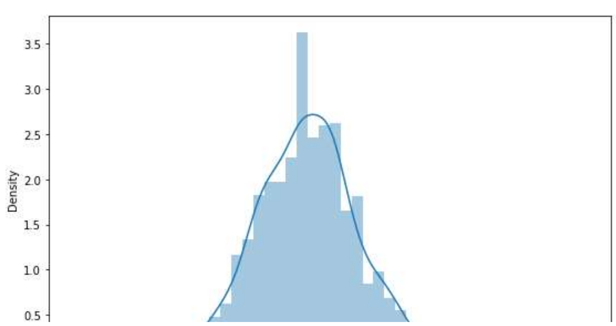

Density
residual sugar

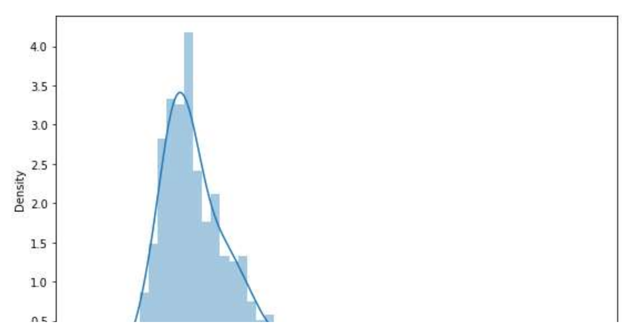

Density
chlorides


Density
free sulfur dioxide


Density
total sulfur dioxide

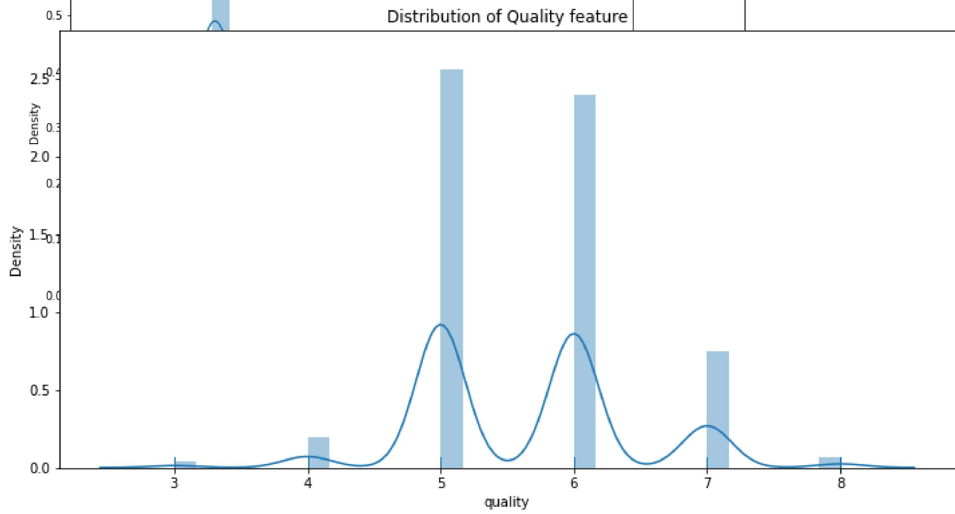

Density
density


Density


Density

In [11]:
```
plt.figure(figsize=(12,6))
plt.title('Distribution of Quality feature')
sns.distplot(raw_df['quality'],bins = 30,rug=True)
```

Out[11]: <AxesSubplot:title={'center':'Distribution of Quality feature'}, xlabel='quality', ylabel='Density'>



In [4]:

```python
from pandas_profiling import ProfileReport

# EDA using pandas-profiling
profile = ProfileReport(raw_df, explorative=True)

# Displaying report in notebook cell.
profile.to_notebook_iframe()
```

Summarize dataset: 100% 170/170 [01:18<00:00, 1.92it/s, Completed]

Generate report structure: 100% 1/1 [00:16<00:00, 16.34s/it]

Render HTML: 100% 1/1 [00:15<00:00, 15.56s/it]

Overview

Dataset statistics

Number of variables	12
Number of observations	1599
Missing cells	0
Missing cells (%)	0.0%
Duplicate rows	220
Duplicate rows (%)	13.8%
Total size in memory	150.0 KiB
Average record size in memory	96.1 B

Variable types

Numeric	12

Alerts

Dataset has 220 (13.8%) duplicate rows	Duplicates
`fixed acidity` is highly correlated with `citric acid` and 3 other fields (citric acid, density, pH, alcohol)	High correlation
`volatile acidity` is highly correlated with `citric acid`	High correlation
`citric acid` is highly correlated with `fixed acidity` and 3 other fields (fixed acidity, chlorides, pH, sulphates)	High correlation
`free sulfur dioxide` is highly correlated with `residual sugar` and 1 other fields (residual sugar, total sulfur dioxide)	High correlation

In [12]:

```python
X=raw_df.drop("quality",axis=1)
```

In [13]:

```python
y=raw_df["quality"]
```

In [14]:

```python
X.head()
```

Out[14]:

	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 [15]:
```
y.head()
```

Out[15]:
```
0    5
1    5
2    5
3    6
4    5
Name: quality, dtype: int64
```

In [16]:
```
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=42)
```

In [17]:
```
print("Train dataset contains {} rows and {} columns.".format(X_train.shape[0],X_train.shape[1]))
print("Test dataset contains {} rows and {} columns.".format(X_test.shape[0],X_test.shape[1]))
print("Train dataset contains {} rows.".format(y_train.shape[0]))
print("Test dataset contains {} rows.".format(y_test.shape[0]))
```

```
Train dataset contains 1071 rows and 11 columns.
Test dataset contains 528 rows and 11 columns.
Train dataset contains 1071 rows.
Test dataset contains 528 rows.
```

Feature Scaling

In [18]:
```
from sklearn.preprocessing import StandardScaler
```

In [19]:
```
scaler = StandardScaler()
```

In [20]:
```
scaler.fit(X_train)##calculate the mean and std dev
```

Out[20]:
```
▾ StandardScaler
StandardScaler()
```

In [21]:
```
print(scaler.mean_)
```

```
[ 8.30345472  0.53246499  0.26933707  2.54691877  0.08772736 15.91223156
 46.76330532  0.99677933  3.31453782  0.65881419 10.41521942]
```

In [22]:
```
X_train_scaled=scaler.transform(X_train)
```

In [23]:
```
X_train_scaled
```

Out[23]:
```
array([[ 2.40069523, -1.03103722,  1.12742595, ..., -1.26096312,
         0.52726134, -0.01431863],
       [-0.93967131,  1.22920403, -1.32502245, ...,  1.52622836,
        -0.28225704,  2.24363201],
       [-0.99827424,  0.55113165, -1.37611513, ..., -0.74241587,
        -1.20742091, -0.86105011],
       ...,
       [-0.6466567 ,  0.49462562, -1.06955908, ...,  1.26695473,
        -0.68701624, -0.86105011],
       [-0.23643625, -1.87862768,  0.4121285 , ...,  0.03540501,
         0.81637505,  1.39690052],
       [-1.46709761, -1.3700734 , -0.04770558, ...,  0.48913386,
        -0.68701624,  2.90220094]])
```

In [24]:
```
X_test_scaled=scaler.transform(X_test)
```

In [25]:
```
X_test_scaled
```

Out[25]:
```
array([[-3.53642095e-01,  1.55589436e-01, -9.67373729e-01, ...,
        -4.83142240e-01,  6.85666499e-03, -7.66968836e-01],
       [-2.95039173e-01, -1.83446751e-01, -5.07539654e-01, ...,
         4.89133857e-01, -1.03395269e+00, -8.61050113e-01],
       [ 1.40444556e+00,  7.77155778e-01, -2.52076279e-01, ...,
        -2.23868614e-01,  1.85718440e+00, -4.84725007e-01],
       ...,
       [-2.02456406e-03, -1.25706134e+00,  6.16499196e-01, ...,
        -2.94133945e-02,  6.42906824e-01,  1.96138818e+00],
       [-6.06274859e-02,  4.50655383e+00, -1.37611513e+00, ...,
         1.39659155e+00, -9.76129945e-01,  4.56087756e-01],
       [ 4.66798811e-01,  7.20649747e-01, -6.09725004e-01, ...,
        -2.23868614e-01, -6.87016236e-01, -7.66968836e-01]])
```

Model Building

Support Vector Classifier

In [26]:
```
from sklearn.svm import SVC
SVC_model=SVC()
```

In [27]:
```
SVC_model.fit(X_train_scaled,y_train)
```

Out[27]:
```
▾ SVC
SVC()
```

In [28]:
```
SVC_model.score(X_train_scaled,y_train)
```

Out[28]:
```
0.6778711484593838
```

In [29]:
```
SVC_predict=SVC_model.predict(X_test_scaled)
```

In [30]:
```
from sklearn.metrics import accuracy_score
```

In [31]:
```
accuracy_score(y_test,SVC_predict)
```

Out[31]:
```
0.5984848484848485
```

Logistic Regression

In [32]:
```
from sklearn.linear_model import LogisticRegression
```

In [33]:
```
LR_model=LogisticRegression()
```

In [34]:
```
LR_model.fit(X_train_scaled,y_train)
```

Out[34]:
```
▾ LogisticRegression
LogisticRegression()
```

In [35]:
```
LR_predict=LR_model.predict(X_test_scaled)
```

In [36]:
```
accuracy_score(y_test,LR_predict)
```

Out[36]:
```
0.571969696969697
```

In [37]:
```
X_test_scaled[0]
```

Out[37]:
```
array([-0.3536421 ,  0.15558944, -0.96737373, -0.03334372,  0.55556956,
       -0.18596079, -0.02314512,  0.1740298 , -0.48314224,  0.00685666,
       -0.76696884])
```

Single point prediction

In [38]:
```
SVC_model.predict([[-0.3536421 ,  0.15558944, -0.96737373, -0.03334372,  0.55556956,
       -0.18596079, -0.02314512,  0.1740298 , -0.48314224,  0.00685666,
       -0.76696884]])
```

Out[38]:
```
array([5], dtype=int64)
```

In [39]:
```
LR_model.predict([[-0.3536421 ,  0.15558944, -0.96737373, -0.03334372,  0.55556956,
       -0.18596079, -0.02314512,  0.1740298 , -0.48314224,  0.00685666,
       -0.76696884]])
```

Out[39]:
```
array([5], dtype=int64)
```