NAME : NIMMAGADDA NAGA VENKATA SAI KUMAR

UNIVERSITY : SATHYABAMA INSTITUTE OF SCIENCE AND TECHNOLOGY

GMAIL : NNVSAIKUMAR@GMAIL.COM

MOBILE : 7569943524

……………………………………………………………………………………

RED WINE QUALITY PREDICTION

TASK :

```
        Build a model to predict the quality of wine baseed on different input variables like fixed acidity of the
   wine,residual sugar content , chlorides ,  free sulpher dioxide , pH values , alcohol content etc on which wine's              quality is
   dependent
```

Data Description : Use machine learning to determine which physiochemical properties make a wine 'good'!

STEPS :

(1) Importing packages (2) Loading Dataset (3) Plotting Graphs (4) Pre-processing dataset (5) creating ML Model (6) Tuning

IMPORTING REQUIRED PACKAGES

import pandas as pd import numpy as np import matplotlib.pyplot as plt import seaborn as sns

from sklearn.model_selection import train_test_split, GridSearchCV,cross_val_score from sklearn.preprocessing import StandardScaler from sklearn.ensemble import RandomForestClassifier from sklearn.svm import SVC from sklearn.metrics import confusion_matrix

LOADING DATASET

In [9]:

```
#loading dataset
df = pd.read_csv(R'C:\Users\srira\Downloads\winequality-red.csv')
df.head()
```

Out[9]:

	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 [10]:

```
df.info()
```

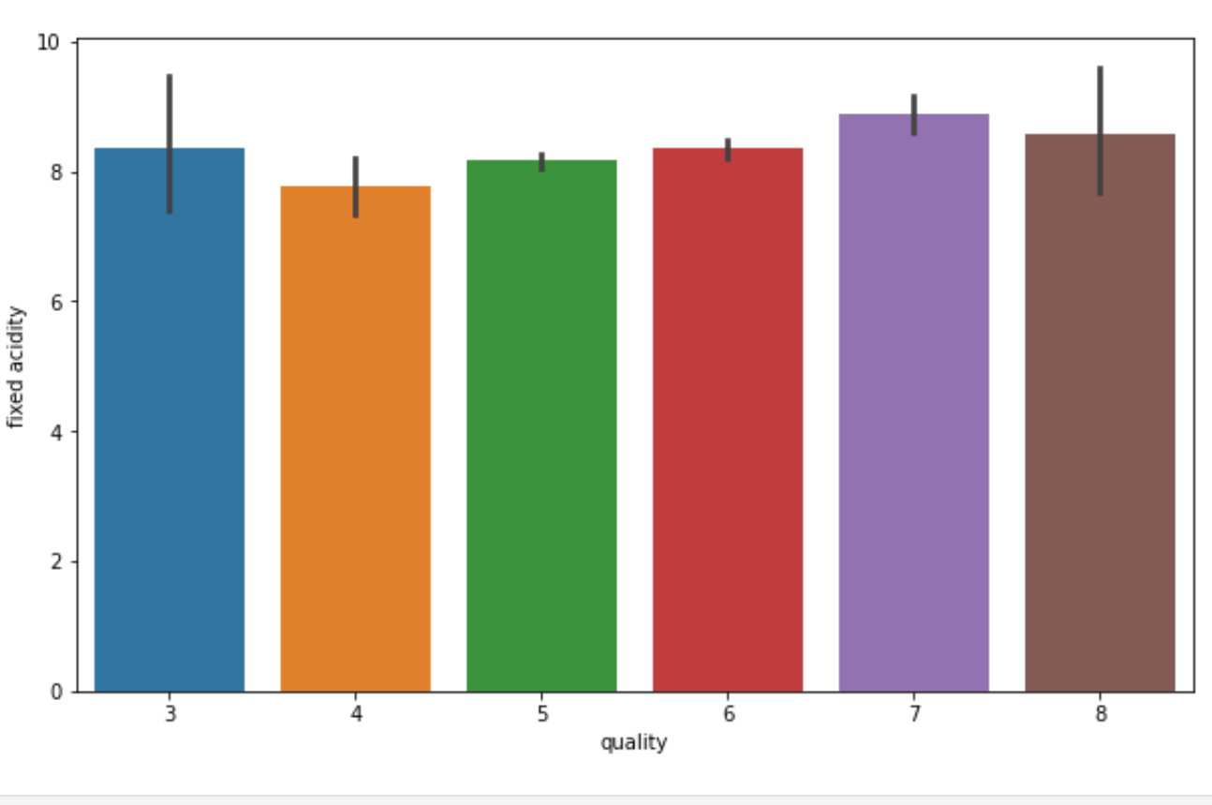
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   fixed acidity         1599 non-null   float64
 1   volatile acidity      1599 non-null   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
```

PLOTING GRAPHS FOR BETTER INSIGHTS TO DATA DESTRIBUTION

In [14]:

```
# fixed acidity v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'fixed acidity', data = df)
```

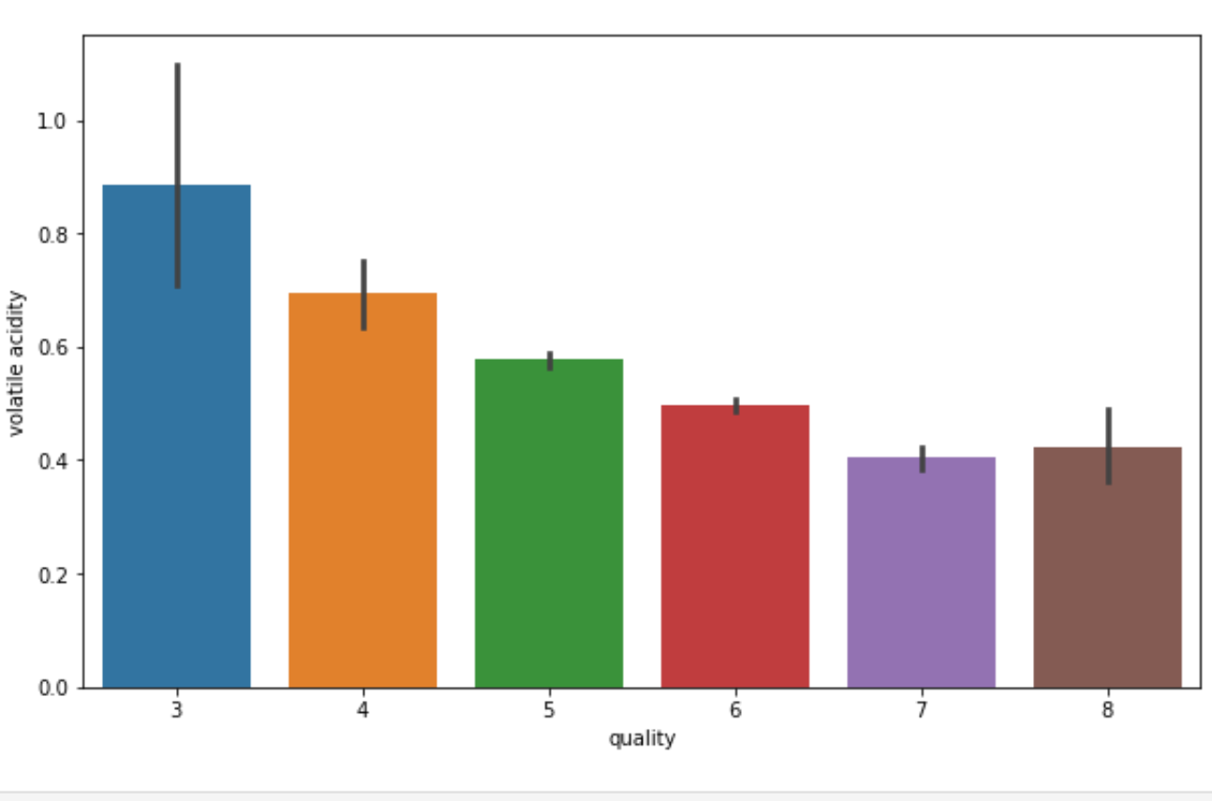
Out[14]: `<AxesSubplot:xlabel='quality', ylabel='fixed acidity'>`



In [15]:

```
# volatile acidity v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'volatile acidity', data = df)
```

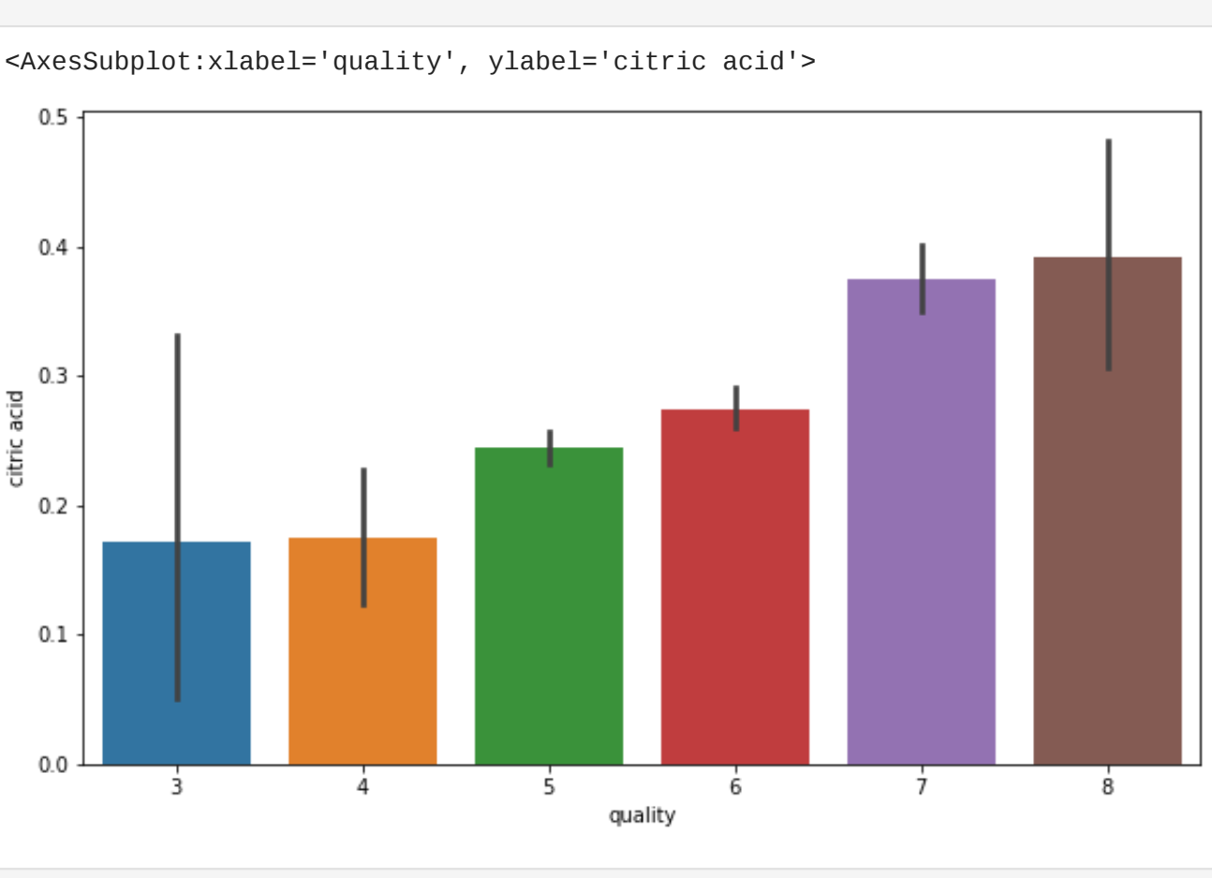
Out[15]: `<AxesSubplot:xlabel='quality', ylabel='volatile acidity'>`



In [17]:

```
# citric acid v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'citric acid', data = df)
```

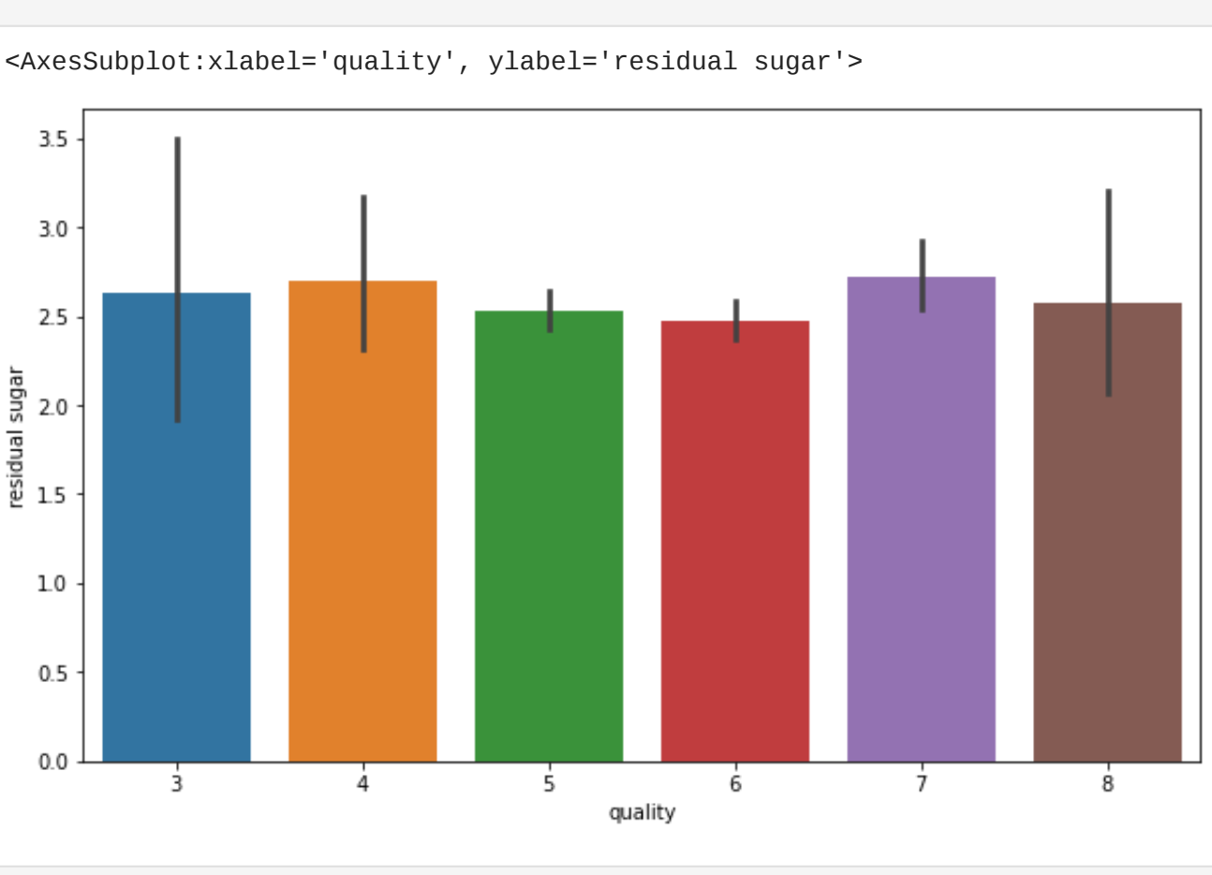
Out[17]: `<AxesSubplot:xlabel='quality', ylabel='citric acid'>`



In [18]:

```
# residual sugar v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'residual sugar', data = df)
```

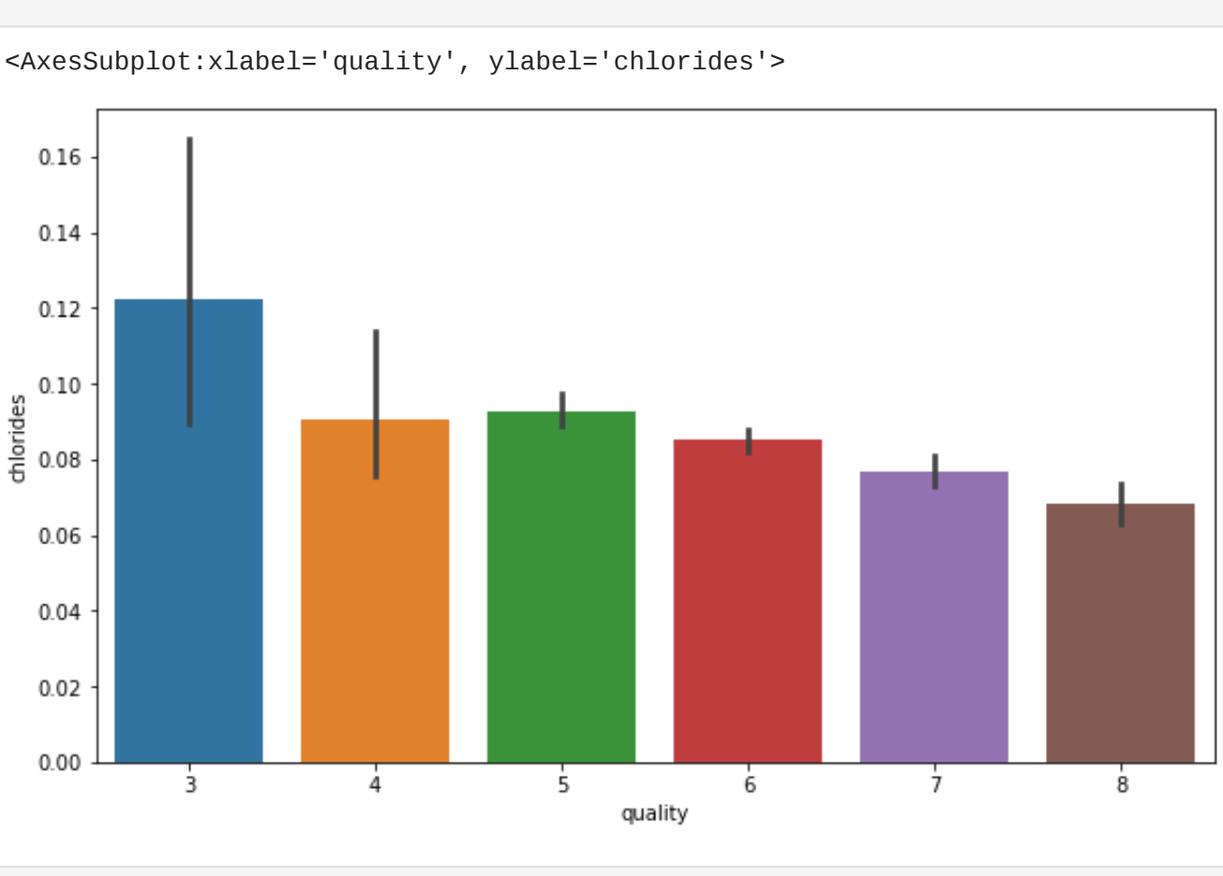
Out[18]: `<AxesSubplot:xlabel='quality', ylabel='residual sugar'>`



In [19]:

```
# chlorides v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'chlorides', data = df)
```

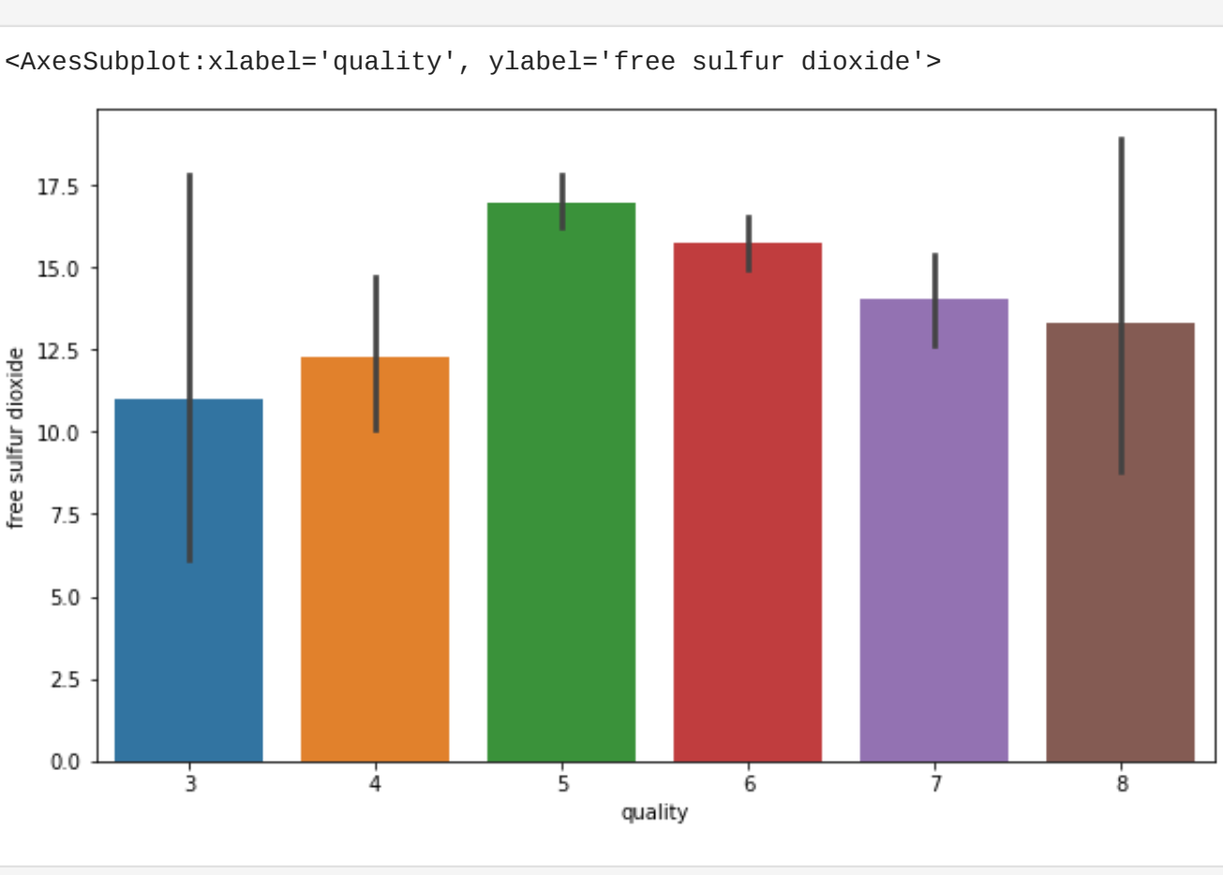
Out[19]: `<AxesSubplot:xlabel='quality', ylabel='chlorides'>`



In [20]:

```
# free sulfur dioxide v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'free sulfur dioxide', data = df)
```

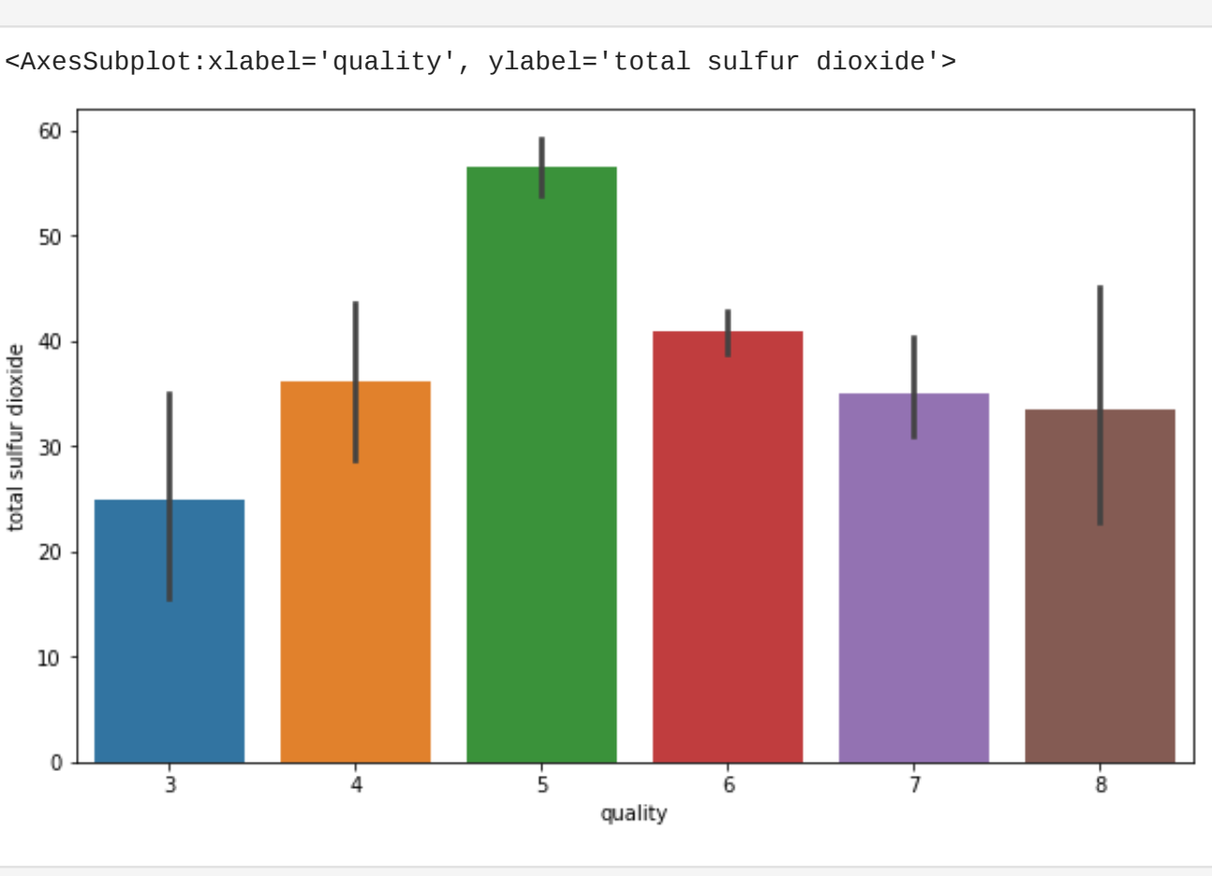
Out[20]: `<AxesSubplot:xlabel='quality', ylabel='free sulfur dioxide'>`



In [21]:

```
# total sulfur dioxide v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'total sulfur dioxide', data = df)
```

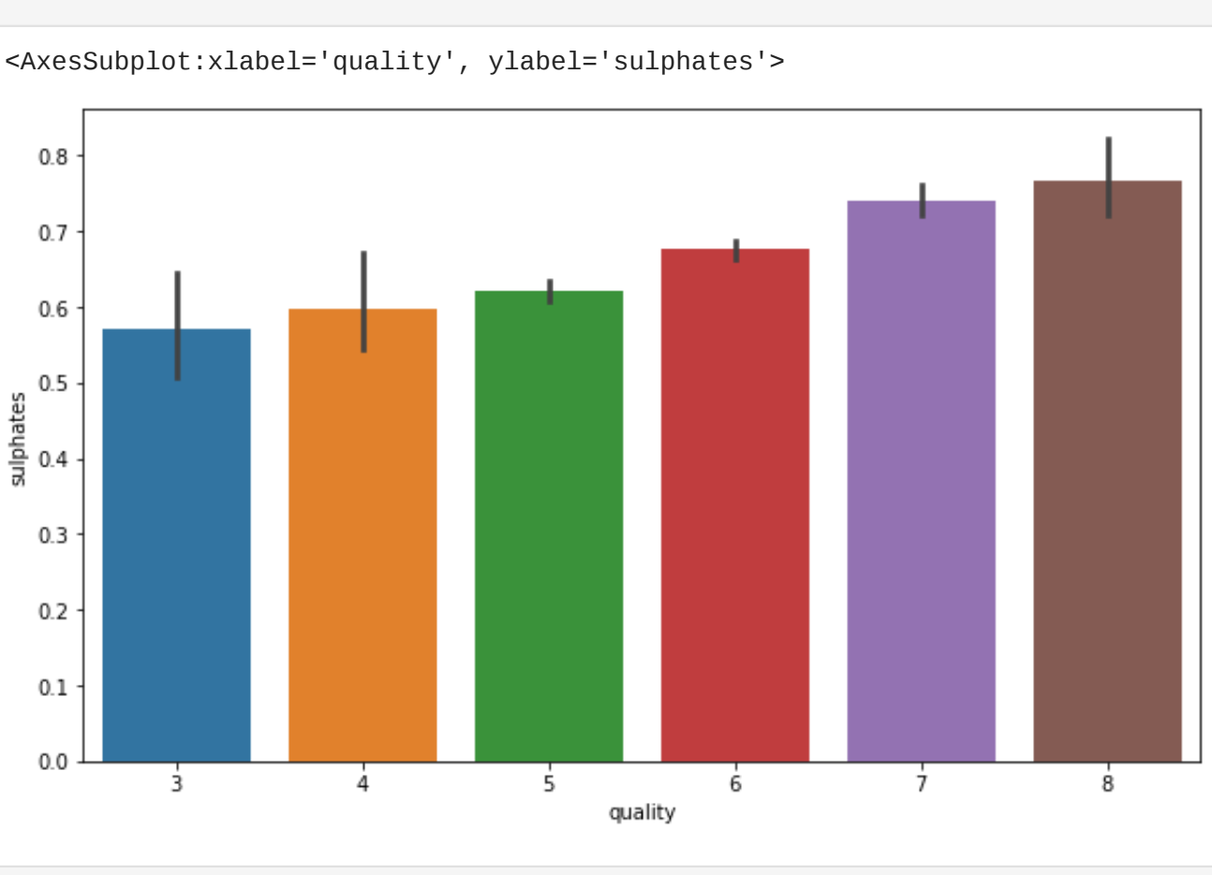
Out[21]: `<AxesSubplot:xlabel='quality', ylabel='total sulfur dioxide'>`



In [22]:

```
# sulphates v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'sulphates', data = df)
```

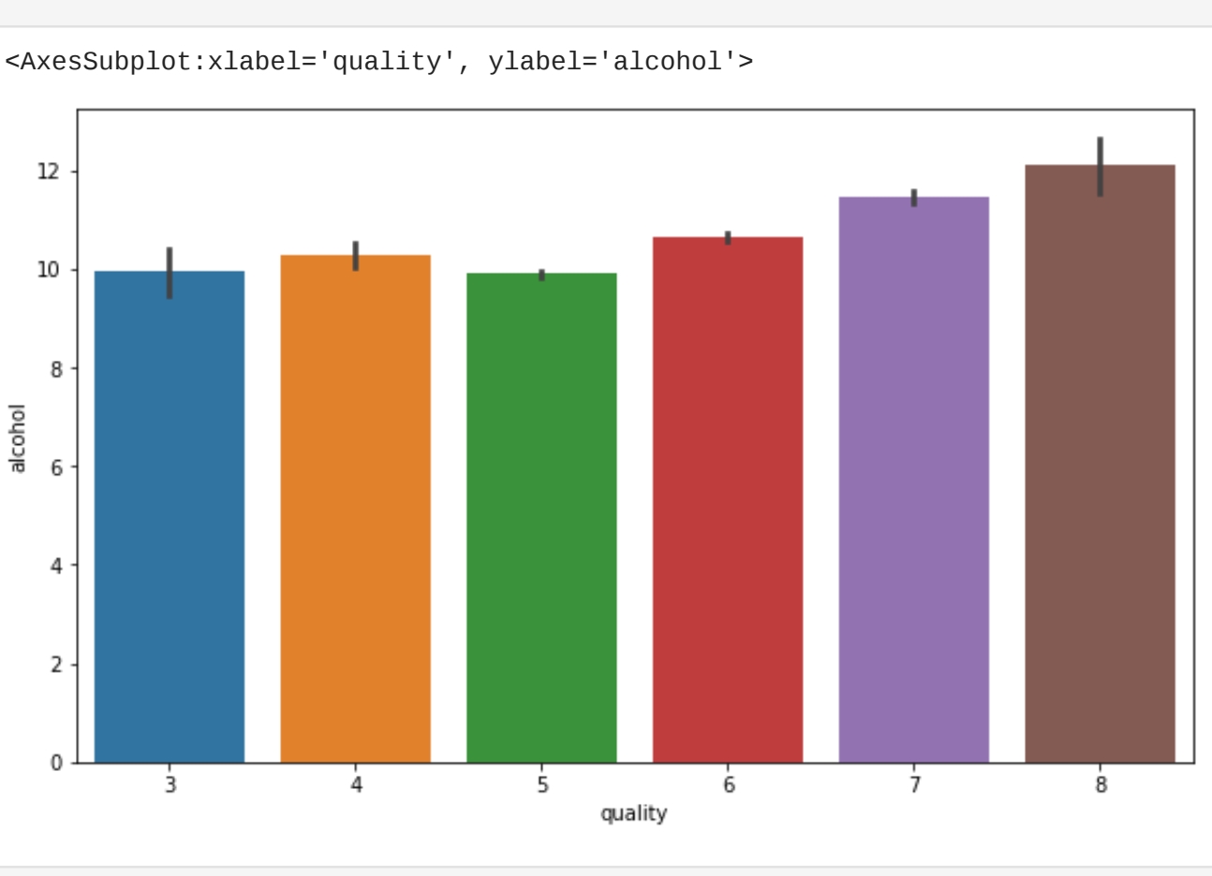
Out[22]: `<AxesSubplot:xlabel='quality', ylabel='sulphates'>`



In [23]:

```
# alcohol v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'alcohol', data = df)
```

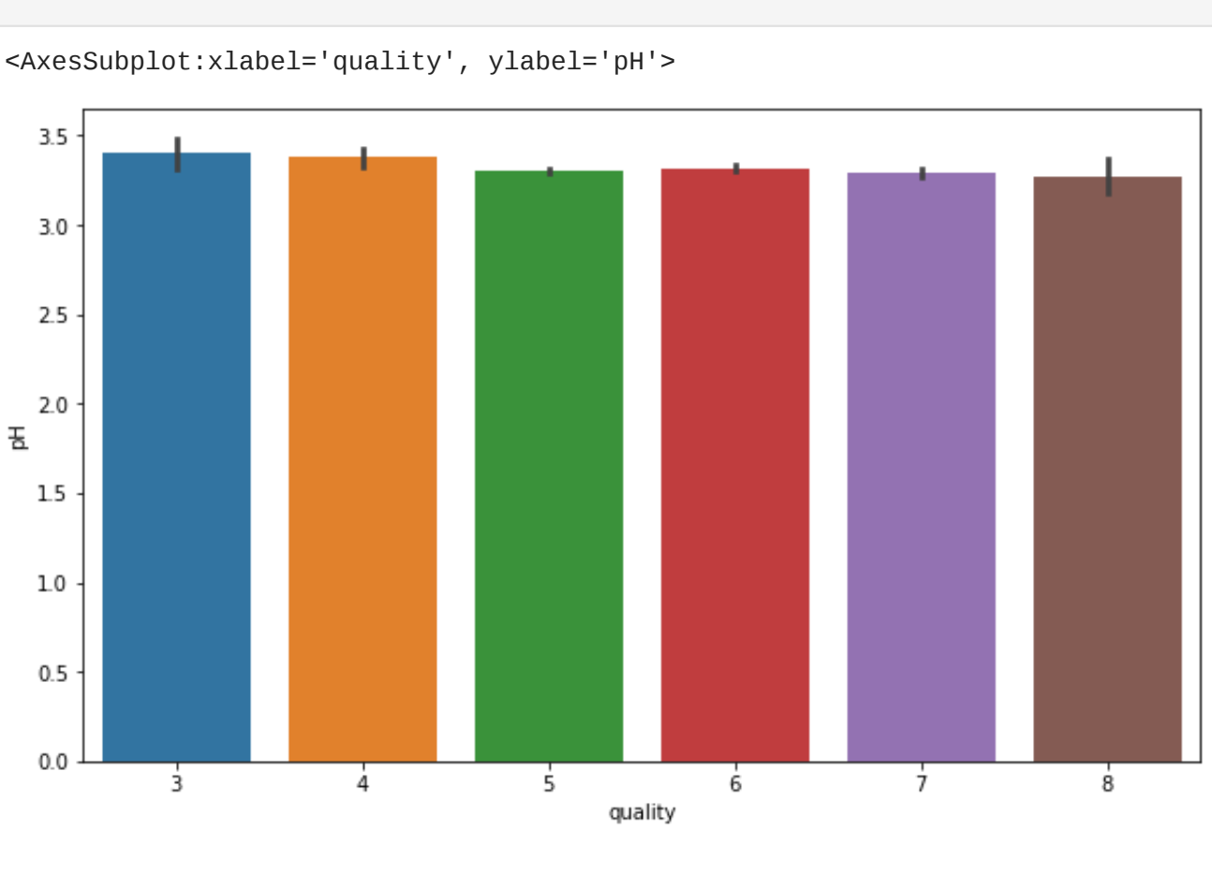
Out[23]: `<AxesSubplot:xlabel='quality', ylabel='alcohol'>`



In [25]:

```
# ph v/s quality

plt.figure(figsize= (10, 6))
sns.barplot (x = 'quality', y = 'pH', data = df)
```

Out[25]: `<AxesSubplot:xlabel='quality', ylabel='pH'>`



PRE-PROCESSING DATA FOR ML ALGORITHM

In [26]:

```
df["quality"].value_counts()
```

Out[26]:

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

In [27]:

```
#Classifying the wine quality as good or bad based on its quality

#"Bad" or e if quality of wine Lies in the range (3, 6) (inclusive)
# "Good" or 1 if quality of wine lies in the range (7, 8) (inclusive)

df['quality'] = df['quality'].apply(lambda x:1 if x>6.5 else 0)
df.head()
```

Out[27]:

	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	0
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	0.68	9.8	0
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	0.65	9.8	0
3	11.2	0.28	0.56	1.9	0.075	17.0	60.0	0.9980	3.16	0.58	9.8	0
4	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	0.56	9.4	0

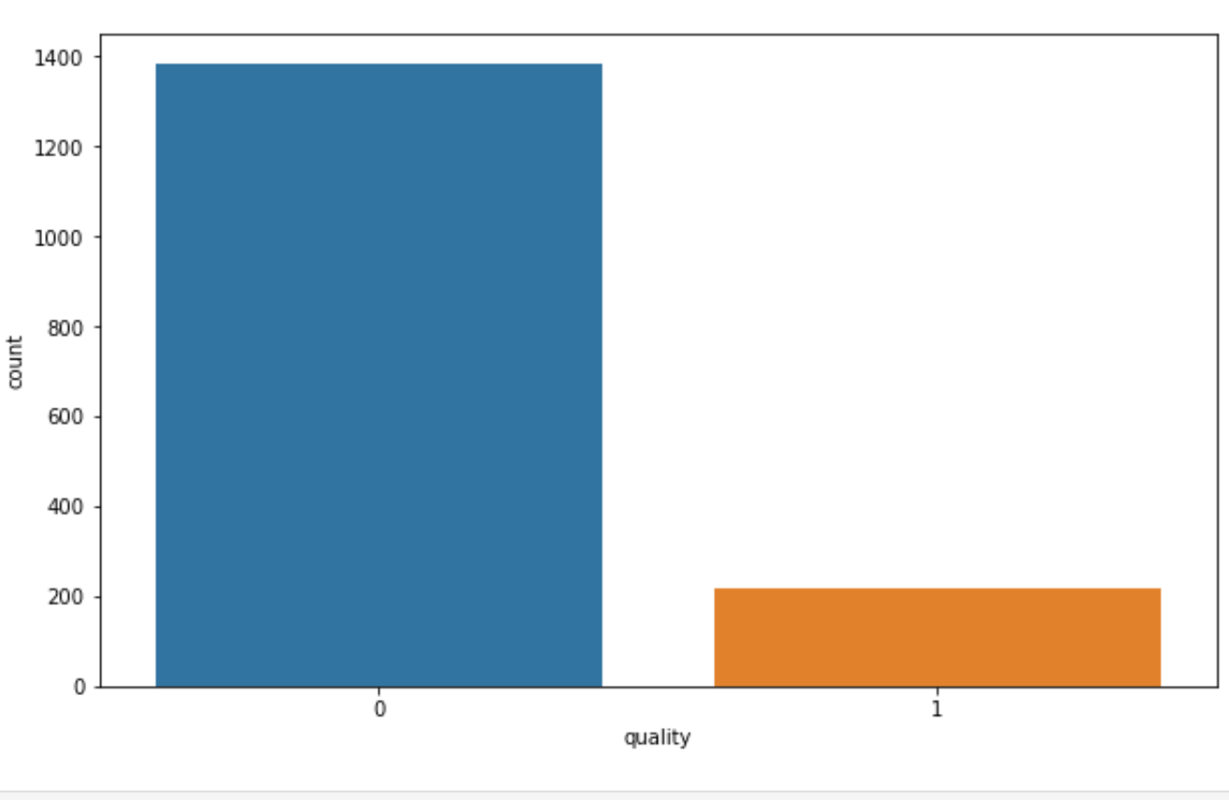
In [31]:

```
print (df['quality'].value_counts())

fig = plt.figure(figsize= (10,6))
sns.countplot(df['quality'])
```

```
0    1382
1     217
Name: quality, dtype: int64
```

```
C:\ProgramData\Anaconda3\lib\site-packages\seaborn\_decorators.py:36: FutureWarning: Pass the following variable as a keyword arg: x. From version 0.12, the o
nly valid positional argument will be `data`, and passing other arguments without an explicit keyword will result in an error or misinterpretation.
  warnings.warn(
```

Out[31]: `<AxesSubplot:xlabel='quality', ylabel='count'>`



In [41]:

```
#Separating to Dependent and Independent variables

X = df.iloc[:, :-1].values

Y = df.iloc[:, -1].values
```

In [42]:

```
X
```

Out[42]:

```
array([[ 7.4  ,  0.7  ,  0.   , ...,  3.51 ,  0.56 ,  9.4  ],
       [ 7.8  ,  0.88 ,  0.   , ...,  3.2  ,  0.68 ,  9.8  ],
       [ 7.8  ,  0.76 ,  0.04 , ...,  3.26 ,  0.65 ,  9.8  ],
       ...,
       [ 6.3  ,  0.51 ,  0.13 , ...,  3.42 ,  0.75 , 11.   ],
       [ 5.9  ,  0.645,  0.12 , ...,  3.57 ,  0.71 , 10.2  ],
       [ 6.   ,  0.31 ,  0.47 , ...,  3.39 ,  0.66 , 11.   ]])
```

In [43]:

```
Y
```

Out[43]:

```
array([0, 0, 0, ..., 0, 0, 0], dtype=int64)
```

In [44]:

```
# Splitting into train and test sets

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size = 0.2, random_state = 42)

print("X-train Shape: ", X_train.shape)

print("X-test Shape: ", X_test.shape)

print("Y-train Shape: ", Y_train.shape)

print("Y-test Shape: ", Y_test.shape)
```

```
X-train Shape:  (1279, 11)
X-test Shape:  (320, 11)
Y-train Shape:  (1279,)
Y-test Shape:  (320,)
```

In [45]:

```
# applying Standard Scaling to the dataset to scale all the field values to same scale (approx.)

sc = StandardScaler ()

X_train = sc.fit_transform(X_train)

X_test= sc.fit_transform(X_test)
```

CREATING ML ALGORITHMS

RANDOM FOREST CLASSIFIER

In [51]:

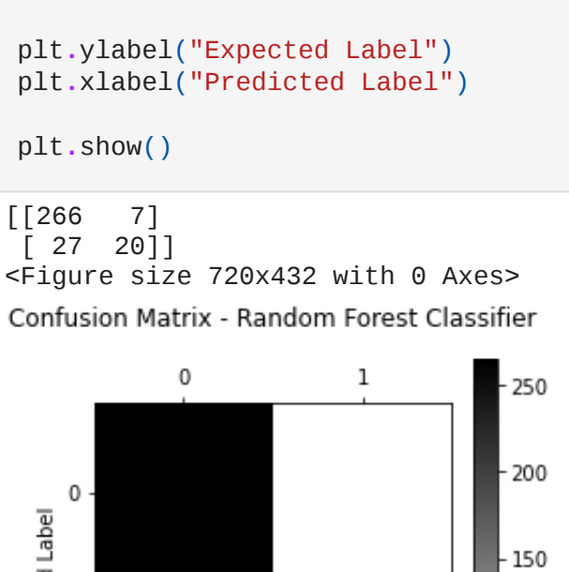
```
# Creating Random Forest Classifier Instance

rf_regressor = RandomForestClassifier(n_estimators=100)

# Fitting the classifier with training data
rf_regressor.fit(X_train, Y_train)

# Predicting using classifier

Y_pred = rf_regressor.predict (X_test)
```

In [56]:

```
# Confusion Matrix using the Random Forest Classifier

cm = confusion_matrix(Y_test, Y_pred)
print(cm)

#Matplot for Confusion Matrix of Random Forest Classifier

fig= plt.figure(figsize=(10,6))

plt.matshow(cm, cmap = plt.cm.binary, interpolation = 'nearest')

plt.title("Confusion Matrix - Random Forest Classifier\n")

plt.colorbar()

plt.ylabel("Expected Label")
plt.xlabel("Predicted Label")

plt.show()
```

```
[[266   7]
 [ 27  20]]
<Figure size 720x432 with 0 Axes>
```

Confusion Matrix - Random Forest Classifier



In [60]:

```
print("Total Predictions :", len(Y_pred))
print("Correct Prediction :", sum(Y_pred==Y_test))
print("Incorrect Prediction:", sum(Y_pred != Y_test))
print("Accuracy:", sum(Y_pred ==Y_test)*100 / len(Y_pred), "%")
```

```
Total Predictions : 320
Correct Prediction : 286
Incorrect Prediction: 34
Accuracy: 89.375 %
```

SUPPORT VECTOR CLASSIFIER(SVC)

In [63]:

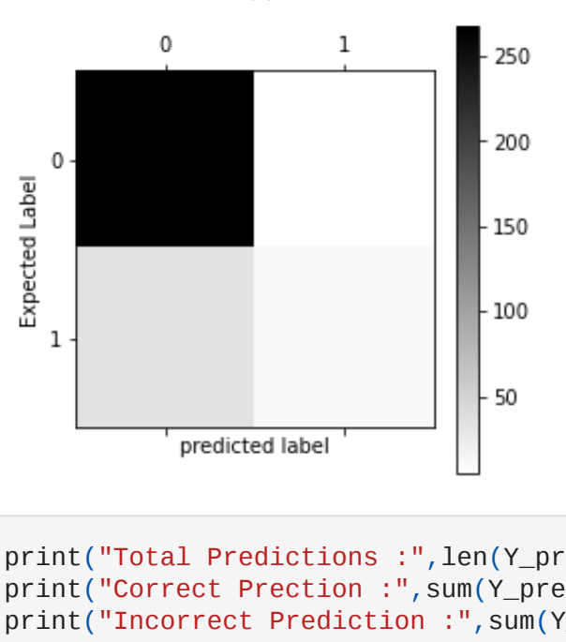
```
svc_regressor= SVC()
svc_regressor.fit(X_train, Y_train)
Y_pred = svc_regressor.predict(X_test)
```

In [64]:

```
cm = confusion_matrix(Y_test,Y_pred)
print(cm)
fig = plt.figure(figsize=(10,6))
plt.matshow(cm,cmap=plt.cm.binary,interpolation = 'nearest')
plt.title("confusion matrix - support vector classifier\n")
plt.colorbar()
plt.ylabel("Expected Label")
plt.xlabel("predicted label")
plt.show()
```

```
[[268   5]
 [ 35  12]]
<Figure size 720x432 with 0 Axes>
```

confusion matrix - support vector classifier



In [65]:

```
print("Total Predictions :",len(Y_pred))
print("Correct Prection :",sum(Y_pred==Y_test))
print("Incorrect Prediction :",sum(Y_pred!=Y_test))
print("Accuracy :",sum(Y_pred==Y_test)*100/len(Y_pred),"%")
```

```
Total Predictions : 320
Correct Prection : 280
Incorrect Prediction : 40
Accuracy : 87.5 %
```

TUMING ML ALGORITHM

SVC

In [80]:

```
#Finding parameters for SVC models

param = {
    'c': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
    'kernel': ['linear', 'rbf'],
    'gamma': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4]

}

grid_svc = GridSearchCV(svc_regressor, param_grid = param, scoring = 'accuracy', cv = 10)
```

In [81]:

```
print(grid_svc)
```

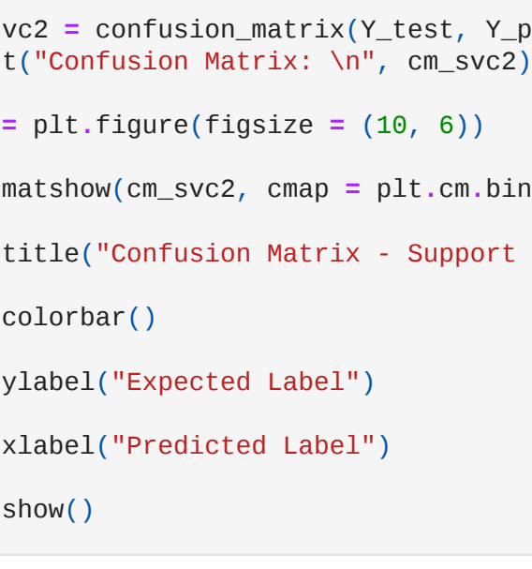
```
GridSearchCV(cv=10, estimator=SVC(),
             param_grid={'c': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'gamma': [0.1, 0.8, 0.9, 1, 1.1, 1.2, 1.3, 1.4],
                         'kernel': ['linear', 'rbf']},
             scoring='accuracy')
```

In [83]:

```
svc_regressor2 = SVC(C= 1.2, gamma=0.9, kernel = 'rbf')

svc_regressor2.fit(X_train, Y_train)

Y_pred_svc= svc_regressor2.predict (X_test)

cm_svc2 = confusion_matrix(Y_test, Y_pred_svc)
print("Confusion Matrix: \n", cm_svc2)

fig = plt.figure(figsize = (10, 6))

plt.matshow(cm_svc2, cmap = plt.cm.binary, interpolation = 'nearest')

plt.title("Confusion Matrix - Support Vecotr Classifier (Improved)\n")

plt.colorbar()

plt.ylabel("Expected Label")

plt.xlabel("Predicted Label")

plt.show()
```

```
Confusion Matrix:
 [[271   2]
 [ 31  16]]
<Figure size 720x432 with 0 Axes>
```

Confusion Matrix - Support Vecotr Classifier (Improved)



In [86]:

```
print("Total Predictions :",len(Y_pred_svc))
print("Correct Prection :",sum(Y_pred_svc==Y_test))
print("Incorrect Prediction :",sum(Y_pred_svc!=Y_test))
print("Accuracy :",sum(Y_pred_svc==Y_test)*100/len(Y_pred_svc),"%")
```

```
Total Predictions : 320
Correct Prection : 287
Incorrect Prediction : 33
Accuracy : 89.6875 %
```

In [97]:

```
rf_eval = cross_val_score(estimator = rf_regressor, X = X_train, y = Y_train,cv = 10)
rf_eval.mean()
```

Out[97]:

```
0.9085322342519685
```

RESULT :

(1) We created ML classifier for classifying wine as good or bad.

(2) The dataset contained 1599 records and 12 fields.

(3) Random Forest Classifier Classified with an accuracy of 89.5%(approx) , and 90%(approx) after tuning>

(4) SVC classifier with and accuracy of 88%(approx) ,and 90%(approx) after tuning.