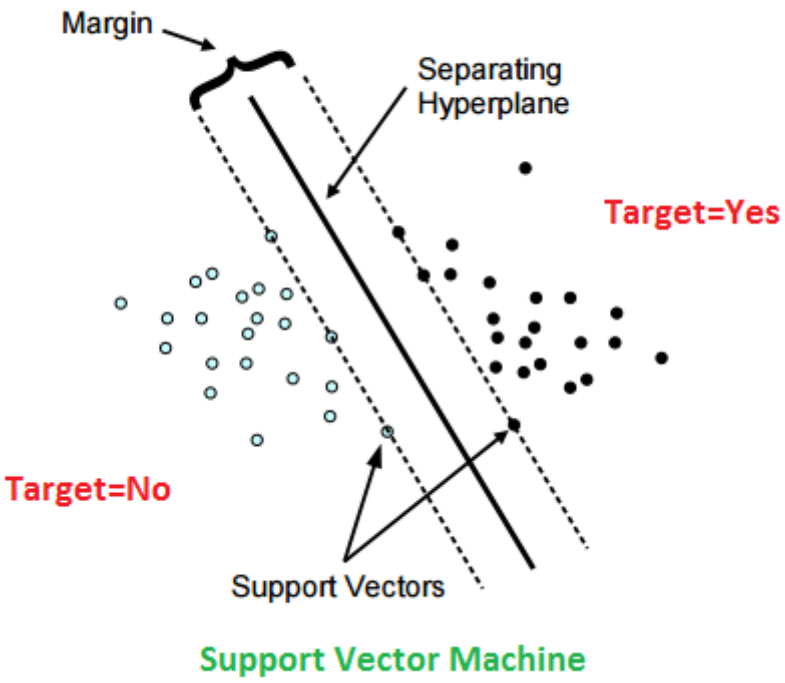
Lab - 9 - Support Vector Machines

> Support Vector Machines (SVMs in short) are machine learning algorithms that are used for classification and regression purposes. SVMs are one of the powerful machine learning algorithms for classification, regression and outlier detection purposes.

> An SVM classifier builds a model that assigns new data points to one of the given categories. Thus, it can be viewed as a non-probabilistic binary linear classifier



Objective -

Demonstrate-

- Classification using Linear, Polynomial and Radial basis function kernels. * Demonstrate the impact of regularization.
- Demonstrate GridSearchCv method for obtaining optimal hyperparameters for classification using RBF kernel. Any suitable dataset(s) of your choice can be used for the experiments. Following link contains details of various functions required for the implementation.

```
from google.colab import drive
```

```
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

```
# !ls "/content/drive/My Drive/Colab ML"
```

Import the Primary packages

```
import numpy as np
import sklearn
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

wine = pd.read_csv('/content/drive/My Drive/Colab ML/winequality-red.csv')
```

Exploratory Data Analysis

```
wine.head()
```

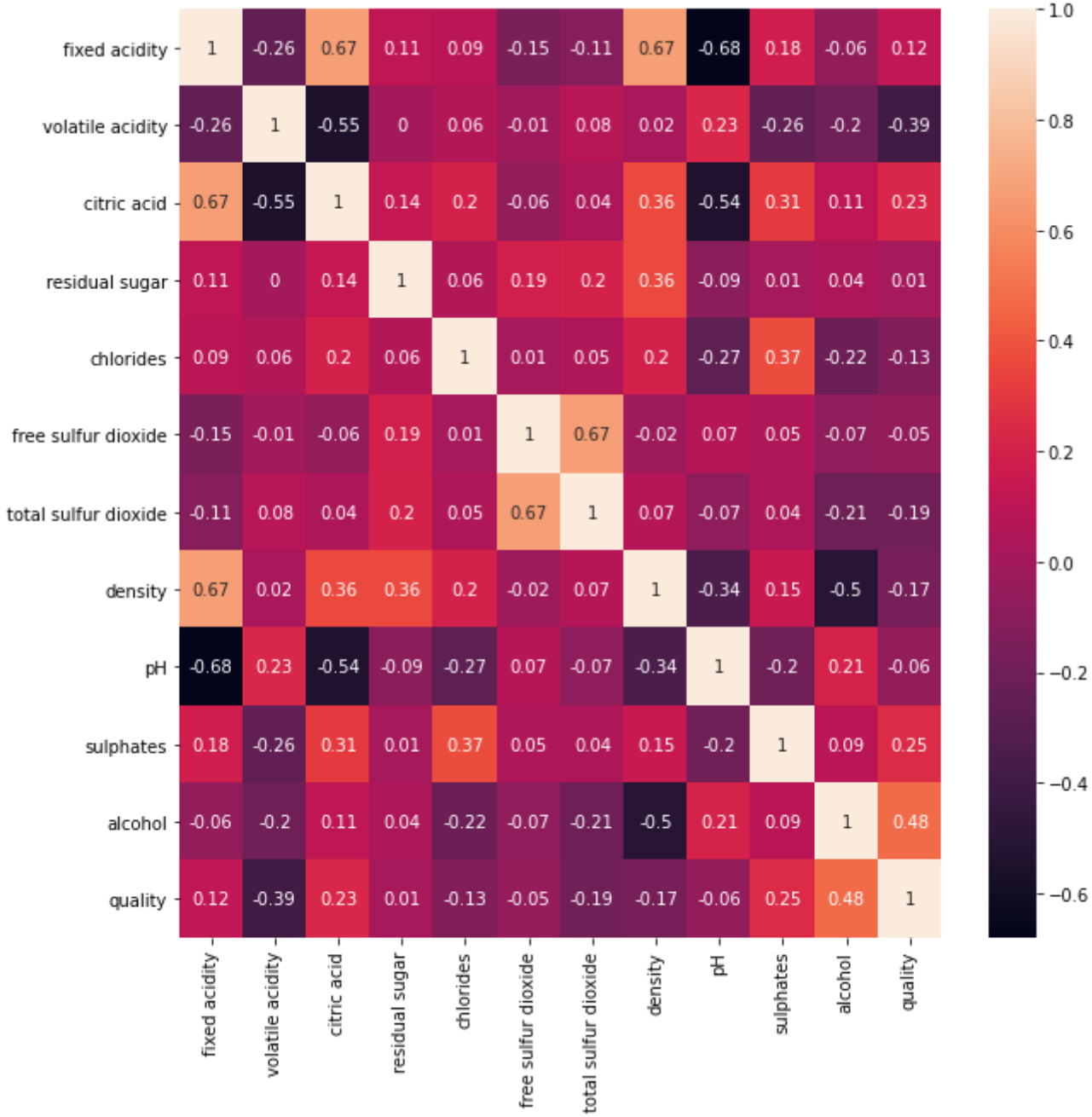
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	pH	s
0	7.4	0.70	0.00	1.9	0.076	11.0	34.0	0.9978	3.51	
1	7.8	0.88	0.00	2.6	0.098	25.0	67.0	0.9968	3.20	
2	7.8	0.76	0.04	2.3	0.092	15.0	54.0	0.9970	3.26	
3	11.2	0.28	0.56	1.9	0.075	17.0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

```
print('Wine Quality classes are - \n',np.unique(wine['quality']))
wine.info()
```

```
Wine Quality classes are -
 [3 4 5 6 7 8]
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1599 entries, 0 to 1598
Data columns (total 12 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
```

```
import seaborn as sns

fig, ax = plt.subplots(figsize=(10,10))         # Sample figsize in inches
correlation_matrix = wine.corr().round(2)
sns.heatmap(data=correlation_matrix, annot=True, ax = ax)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22cdd3c080>
```




```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size =0.1, random_state= 1)
```

Making binary classificaion for the response variable.

Dividing wine as good and bad by giving the limit for the quality.

```
bins = (2, 6.5, 8)
group_names = ['bad', 'good']
wine['quality'] = pd.cut(wine['quality'], bins = bins, labels = group_names)
```

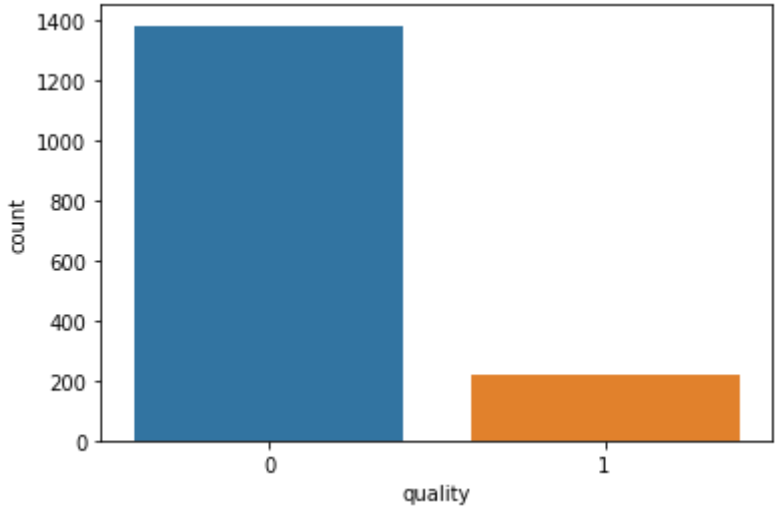
Now lets assign a labels to our quality variable

```
from sklearn.preprocessing import StandardScaler, LabelEncoder

label_quality = LabelEncoder()


wine['quality'] = label_quality.fit_transform(wine['quality'])



wine['quality'].value_counts()
```

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

```
sns.countplot(wine['quality'])
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f22da002710>
```



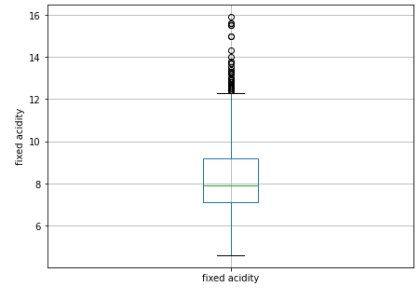
The target class is divided into two categories

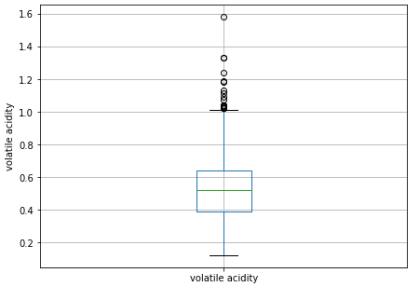
```
round(wine.describe(),2)
```

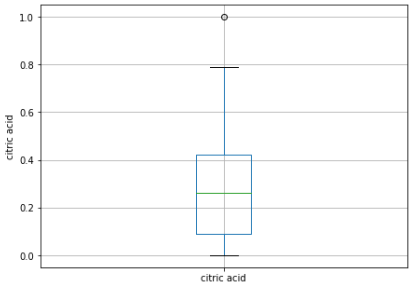
	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free sulfur dioxide	total sulfur dioxide	density	
count	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	1599.00	15
mean	8.32	0.53	0.27	2.54	0.09	15.87	46.47	1.00	
std	1.74	0.18	0.19	1.41	0.05	10.46	32.90	0.00	
min	4.60	0.12	0.00	0.90	0.01	1.00	6.00	0.99	
25%	7.10	0.39	0.09	1.90	0.07	7.00	22.00	1.00	
50%	7.90	0.52	0.26	2.20	0.08	14.00	38.00	1.00	
75%	9.20	0.64	0.42	2.60	0.09	21.00	62.00	1.00	

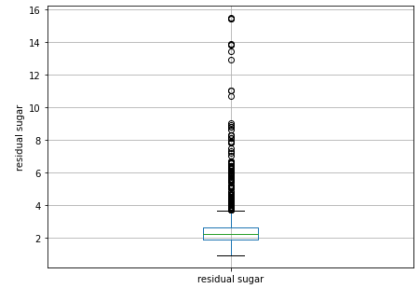
Boxplots to visualize outliers

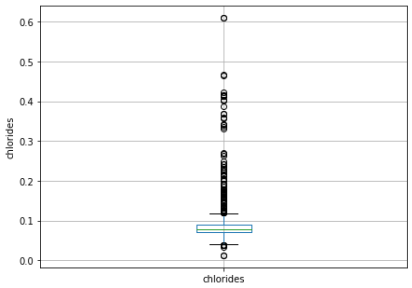
```
plt.figure(figsize=(24,30))
k = wine.columns
for i in range(11):
# plt.figure(figsize=(24,30))

  plt.subplot(5, 3, i+1)
  fig = wine.boxplot(column=k[i])
  fig.set_title('')
  fig.set_ylabel(k[i])
```

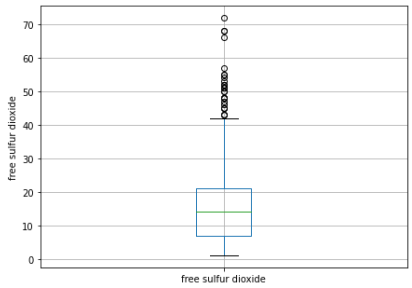


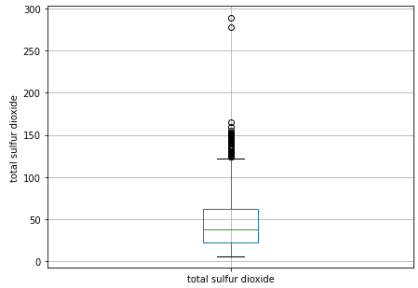


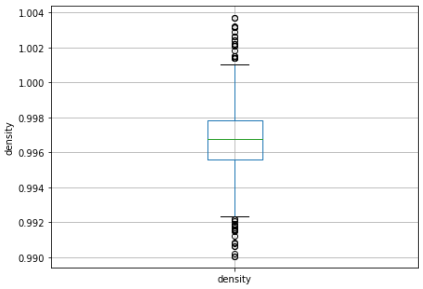


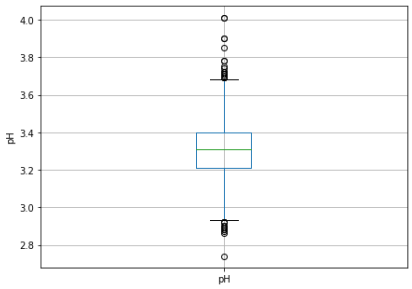


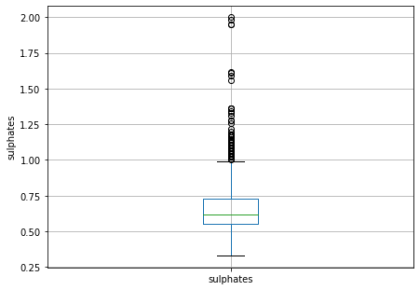


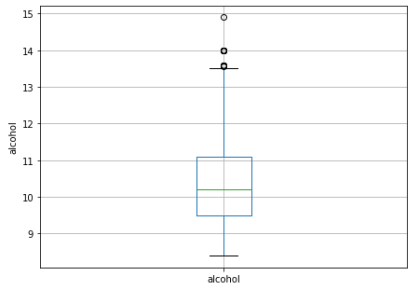








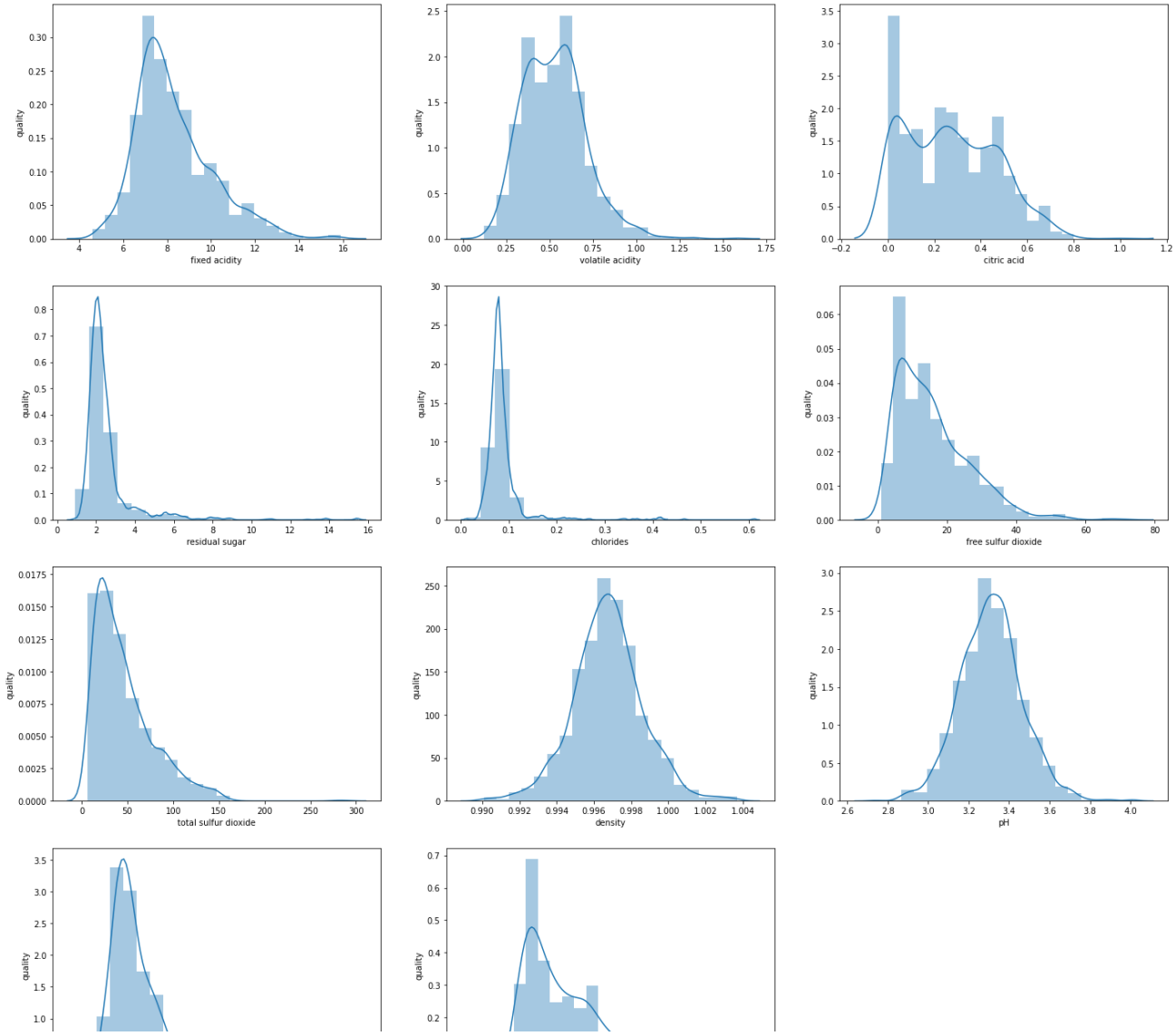




> **The above boxplots confirm that there are lot of outliers in these variables which will require carefull fitting of model.**

Handling Outliers

```
plt.figure(figsize=(24,30))
k = wine.columns
for i in range(11):

  plt.subplot(5, 3, i+1)
  fig = sns.distplot(wine[k[i]], bins =20, hist = True)
  fig.set_xlabel(k[i])
  fig.set_ylabel('quality')
```



We observe that except Density and pH all the remaining are Skewed.

Declaring Feature vectors and Target Variables

```
X = wine.drop(['quality'], axis=1)

y = wine['quality']
```

Split the data into training and testing

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state =


X_train.shape, X_test.shape
```

```
((1279, 11), (320, 11))
```

Feature Scaling

```
cols = X_train.columns


from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

X_train = scaler.fit_transform(X_train)

X_test = scaler.transform(X_test)


X_train = pd.DataFrame(X_train, columns=[cols])
X_test = pd.DataFrame(X_test, columns=[cols])
X_train.describe()
```

	fixed acidity	volatile acidity	citric acid	residual sugar	chlorides	free s dioxid
count	1.279000e+03	1.279000e+03	1.279000e+03	1.279000e+03	1.279000e+03	1.279(
mean	-3.644900e-16	4.437420e-16	-1.012135e-16	1.022768e-16	1.848057e-16	-1.098
std	1.000391e+00	1.000391e+00	1.000391e+00	1.000391e+00	1.000391e+00	1.0003
min	-2.097363e+00	-2.246915e+00	-1.392011e+00	-1.138559e+00	-1.570125e+00	-1.4294
25%	-7.134876e-01	-7.672394e-01	-9.317263e-01	-4.498451e-01	-3.562873e-01	-7.638(
50%	-2.521957e-01	-5.480282e-02	-6.229902e-02	-2.432309e-01	-1.679332e-01	-1.932
75%	5.550651e-01	5.891303e-01	7.559855e-01	3.225488e-02	6.227746e-02	4.723(

Run SVM with default hyperparameters - RBF

```
Default hyperparameter means C=1.0, kernel=rbf and gamma=auto among other parameters.


from sklearn.svm import SVC
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix


svc=SVC()

svc.fit(X_train,y_train)
```

```
y_pred=svc.predict(X_test)

print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score

C_parameter = [100, 1000, 10000]

for i in range(3):
  svc=SVC(C = C_parameter[i])

  svc.fit(X_train,y_train)

  y_pred=svc.predict(X_test)

  # plot_confusion_matrix(svc, X_test, y_test, display_labels=wine['quality'])
  cm = confusion_matrix(y_test, y_pred)
  print(cm)
  print('Model accuracy score with hyperparameters: {0:0.4f}'. format(accuracy_score(y_tes
```

```
Model accuracy score with default hyperparameters: 0.9187
[[267  23]
 [ 10  20]]
Model accuracy score with hyperparameters: 0.8969 100
[[271  19]
 [  9  21]]
Model accuracy score with hyperparameters: 0.9125 1000
[[267  23]
 [ 10  20]]
Model accuracy score with hyperparameters: 0.8969 10000
```

Run SVM with linear kernel

```
C_parameter = [1, 100]

for i in range(2):
  svc=SVC(kernel = 'linear', C = C_parameter[i])

  svc.fit(X_train,y_train)

  y_pred=svc.predict(X_test)

  print('Model accuracy score with hyperparameters: {0:0.4f}'. format(accuracy_score(y_tes

svc=SVC(kernel = 'linear', C = 1)
svc.fit(X_train,y_train)
y_pred_train = svc.predict(X_train)

y_pred_train

print('Training-set accuracy score: {0:0.4f}'. format(accuracy_score(y_train, y_pred_train
```

```
print('Training set score: {:.4f}'.format(svc.score(X_train, y_train)))
print('Test set score: {:.4f}'.format(svc.score(X_test, y_test)))
```

```
Model accuracy score with hyperparameters: 0.9062 1
Model accuracy score with hyperparameters: 0.9062 100
Training-set accuracy score: 0.8538
Training set score: 0.8538
Test set score: 0.9062
```

Observed that the model's performance remains same with different hyperparameters. Also the training and test score are fair enough not to be over or under fitted.

Run SVM with polynomial kernel

```
C_parameter = [1, 100, 1000]

for i in range(3):

  poly_svc=SVC(kernel='poly', C=C_parameter[i])

  # fit classifier to training set
  poly_svc.fit(X_train,y_train)


  # make predictions on test set
  y_pred=poly_svc.predict(X_test)


  # compute and print accuracy score
  print('Model accuracy score with polynomial kernel and changing C: {0:0.4f}'. format(acc
```

```
Model accuracy score with polynomial kernel and changing C: 0.9031
Model accuracy score with polynomial kernel and changing C: 0.8812
Model accuracy score with polynomial kernel and changing C: 0.8719
```

> The model failed terribly.

> Polynomial kernel gives poor performance. It may be overfitting the training set.

Hyperparameter Optimization using GridSearch CV

```
from sklearn.model_selection import GridSearchCV

from sklearn.svm import SVC
```

```
# instantiate classifier with default hyperparameters with kernel=rbf, C=1.0 and gamma=aut
svc=SVC()

# declare parameters for hyperparameter tuning
parameters = [ {'C':[1, 10, 100, 1000], 'kernel':['linear']},
               {'C':[1, 10, 100, 1000], 'kernel':['rbf'], 'gamma':[0.1, 0.2, 0.3, 0.4, 0.5
               {'C':[1, 10, 100, 1000], 'kernel':['poly'], 'degree': [2,3,4] ,'gamma':[0.0
              ]



grid_search = GridSearchCV(estimator = svc,
                           param_grid = parameters,
                           scoring = 'accuracy',
                           cv = 5,
                           verbose=0)


grid_search.fit(X_train, y_train)
```

```
GridSearchCV(cv=5, error_score=nan,
                 estimator=SVC(C=1.0, break_ties=False, cache_size=200,
                               class_weight=None, coef0=0.0,
                               decision_function_shape='ovr', degree=3,
                               gamma='scale', kernel='rbf', max_iter=-1,
                               probability=False, random_state=None, shrinking=True,
                               tol=0.001, verbose=False),
                 iid='deprecated', n_jobs=None,
                 param_grid=[{'C': [1, 10, 100, 1000], 'kernel': ['linear']},
                             {'C': [1, 10, 100, 1000],
                              'gamma': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8,
                                        0.9],
                              'kernel': ['rbf']},
                             {'C': [1, 10, 100, 1000], 'degree': [2, 3, 4],
                              'gamma': [0.01, 0.02, 0.03, 0.04, 0.05],
                              'kernel': ['poly']}],
                 pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
                 scoring='accuracy', verbose=0)
```

Examine the best model

```
# best score achieved during the GridSearchCV
print('GridSearch CV best score : {:.4f}\n\n'.format(grid_search.best_score_))


# print parameters that give the best results
print('Parameters that give the best results :','\n\n', (grid_search.best_params_))


# print estimator that was chosen by the GridSearch
print('\n\nEstimator that was chosen by the search :','\n\n', (grid_search.best_estimator_
```

```
GridSearch CV best score : 0.8898


Parameters that give the best results :

 {'C': 1, 'gamma': 0.9, 'kernel': 'rbf'}


Estimator that was chosen by the search :

 SVC(C=1, break_ties=False, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma=0.9, kernel='rbf',
    max_iter=-1, probability=False, random_state=None, shrinking=True,
    tol=0.001, verbose=False)
```

Calculate GridSearch CV score on test set

```
print('GridSearch CV score on test set: {0:0.4f}'.format(grid_search.score(X_test, y_test)
```

```
GridSearch CV score on test set: 0.9344
```

Observations -

- Our original model test accuracy is 0.9187 while GridSearch CV score on test-set is 0.9344.
- So, GridSearch CV helps to identify the parameters that will improve the performance for this particular model.
- Here, we should not confuse best_score_ attribute of grid_search with the score method on the test-set.
- The score method on the test-set gives the generalization performance of the model. Using the score method, we employ a model trained on the whole training set.
- The best_score_ attribute gives the mean cross-validation accuracy, with cross-validation performed on the training set.
- There are outliers in our dataset. So, as I increase the value of C to limit fewer outliers, the accuracy increased. This is true with different kinds of kernels.
- We get maximum accuracy with rbf and linear kernel with C=1 and the accuracy is 0.9344. So, we can conclude that our model is doing a very good job in terms of predicting the class labels. But, this is not true. Here, we have an imbalanced dataset. Accuracy is an inadequate measure for quantifying predictive performance in the imbalanced dataset problem. So, we must explore confusion matrix that provide better guidance in selecting models.