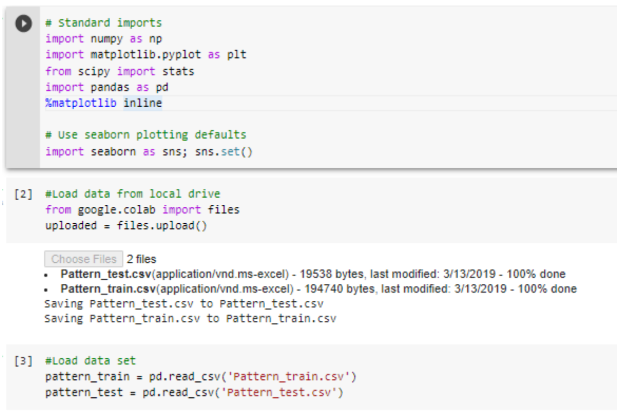
Pattern Recognition Multi-Output Classification Problem Using SVM:

This problem is a multi-input and multi-output classification issue that we would like to solve by the SVM algorithm. This is a classification problem that involves 7 classes as outputs and 10 features or inputs. So, this is a 10-dimensional problem. The task here is to design and train an SVM to minimize the error between actual and predicted output values. We have 2500 data points as the training dataset and 250 data points as the testing dataset. The steps I did from beginning to end will be explained in the following. Note: Here, data is not linearly separable, so, we need to combine SVM with kernels that help SVM become extremely powerful. Although we have 7 outputs or classes, after encoding these outputs, we have a multi-class classification problem that needs to solve as 26 binary classification problems.

Import required libraries and Load data from local drive and load dataset



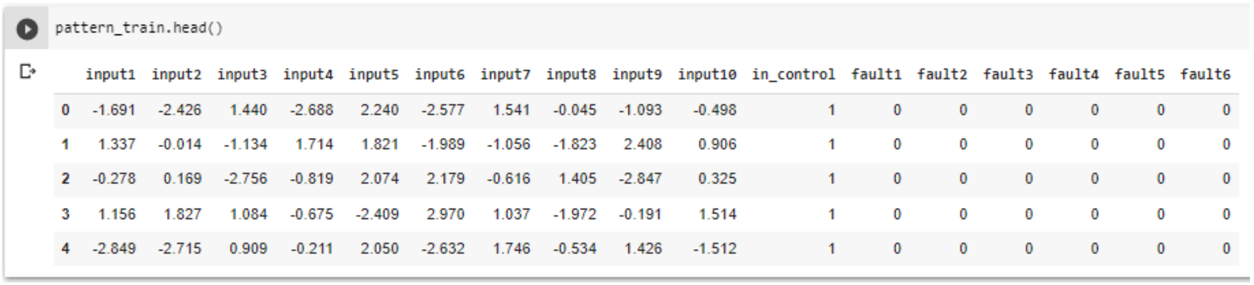
```
# Standard imports
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats
import pandas as pd
%matplotlib inline

# Use seaborn plotting defaults
import seaborn as sns; sns.set()
```

```
[2] #Load data from local drive
    from google.colab import files
    uploaded = files.upload()
```

```
Choose Files 2 files
• Pattern_test.csv(application/vnd.ms-excel) - 19538 bytes, last modified: 3/13/2019 - 100% done
• Pattern_train.csv(application/vnd.ms-excel) - 194740 bytes, last modified: 3/13/2019 - 100% done
Saving Pattern_test.csv to Pattern_test.csv
Saving Pattern_train.csv to Pattern_train.csv
```

```
[3] #Load data set
    pattern_train = pd.read_csv('Pattern_train.csv')
    pattern_test = pd.read_csv('Pattern_test.csv')
```

1. **See how the train data is:**

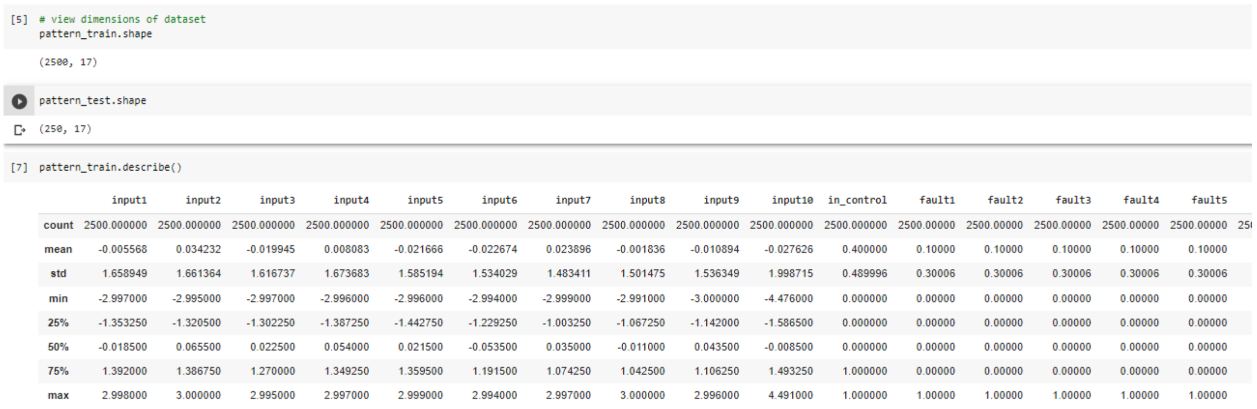


```
pattern_train.head()
```

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	in_control	fault1	fault2	fault3	fault4	fault5	fault6
0	-1.691	-2.426	1.440	-2.688	2.240	-2.577	1.541	-0.045	-1.093	-0.498	1	0	0	0	0	0	0
1	1.337	-0.014	-1.134	1.714	1.821	-1.989	-1.056	-1.823	2.408	0.906	1	0	0	0	0	0	0
2	-0.278	0.169	-2.756	-0.819	2.074	2.179	-0.616	1.405	-2.847	0.325	1	0	0	0	0	0	0
3	1.156	1.827	1.084	-0.675	-2.409	2.970	1.037	-1.972	-0.191	1.514	1	0	0	0	0	0	0
4	-2.849	-2.715	0.909	-0.211	2.050	-2.632	1.746	-0.534	1.426	-1.512	1	0	0	0	0	0	0

2. **Exploratory data analysis**

Now, I will explore the data to gain insights about the data.



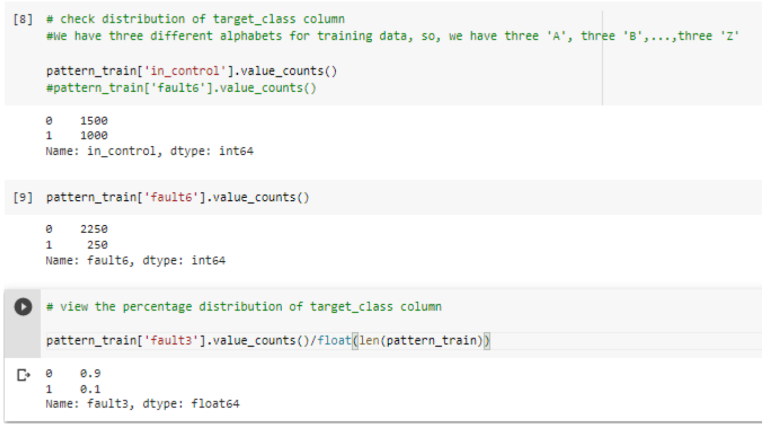
```
[5] # view dimensions of dataset
    pattern_train.shape

    (2500, 17)
```

```
pattern_test.shape

(250, 17)
```

```
[7] pattern_train.describe()
```

	input1	input2	input3	input4	input5	input6	input7	input8	input9	input10	in_control	fault1	fault2	fault3	fault4	fault5
count	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.000000	2500.00000	2500.00000	2500.00000	2500.00000	2500.00000
mean	-0.005568	0.034232	-0.019945	0.008083	-0.021666	-0.022674	0.023896	-0.001836	-0.010894	-0.027626	0.400000	0.10000	0.10000	0.10000	0.10000	0.10000
std	1.658949	1.661364	1.616737	1.673683	1.585194	1.534029	1.483411	1.501475	1.536349	1.998715	0.489996	0.30006	0.30006	0.30006	0.30006	0.30006
min	-2.997000	-2.995000	-2.997000	-2.996000	-2.996000	-2.994000	-2.999000	-2.991000	-3.000000	-4.476000	0.000000	0.00000	0.00000	0.00000	0.00000	0.00000
25%	-1.353250	-1.320500	-1.302250	-1.387250	-1.442750	-1.229250	-1.003250	-1.067250	-1.142000	-1.586500	0.000000	0.00000	0.00000	0.00000	0.00000	0.00000
50%	-0.018500	0.065500	0.022500	0.054000	0.021500	-0.053500	0.035000	-0.011000	0.043500	-0.008500	0.000000	0.00000	0.00000	0.00000	0.00000	0.00000
75%	1.392000	1.386750	1.270000	1.349250	1.359500	1.191500	1.074250	1.042500	1.106250	1.493250	1.000000	0.00000	0.00000	0.00000	0.00000	0.00000
max	2.998000	3.000000	2.995000	2.997000	2.999000	2.994000	2.997000	3.000000	2.996000	4.491000	1.000000	1.00000	1.00000	1.00000	1.00000	1.00000

3. I will check target distribution to know whether I have class imbalanced or not.



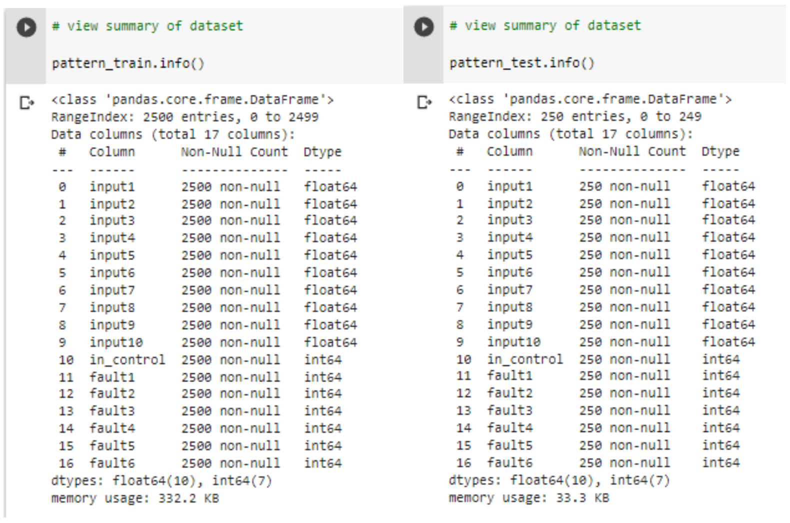
```
[8] # check distribution of target_class column
    #We have three different alphabets for training data, so, we have three 'A', three 'B',...,three 'Z'

    pattern_train['in_control'].value_counts()
    #pattern_train['fault6'].value_counts()

    0    1500
    1    1000
    Name: in_control, dtype: int64
```

```
[9] pattern_train['fault6'].value_counts()

    0    2250
    1     250
    Name: fault6, dtype: int64
```

```
# view the percentage distribution of target_class column

pattern_train['fault3'].value_counts()/float(len(pattern_train))

0    0.9
1    0.1
Name: fault3, dtype: float64
```

So, this is a class imbalanced problem.

4. view summary of dataset



```
# view summary of dataset

pattern_train.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2500 entries, 0 to 2499
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   input1      2500 non-null   float64
 1   input2      2500 non-null   float64
 2   input3      2500 non-null   float64
 3   input4      2500 non-null   float64
 4   input5      2500 non-null   float64
 5   input6      2500 non-null   float64
 6   input7      2500 non-null   float64
 7   input8      2500 non-null   float64
 8   input9      2500 non-null   float64
 9   input10     2500 non-null   float64
 10  in_control  2500 non-null   int64
 11  fault1      2500 non-null   int64
 12  fault2      2500 non-null   int64
 13  fault3      2500 non-null   int64
 14  fault4      2500 non-null   int64
 15  fault5      2500 non-null   int64
 16  fault6      2500 non-null   int64
dtypes: float64(10), int64(7)
memory usage: 332.2 KB
```

```
# view summary of dataset

pattern_test.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 250 entries, 0 to 249
Data columns (total 17 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   input1      250 non-null    float64
 1   input2      250 non-null    float64
 2   input3      250 non-null    float64
 3   input4      250 non-null    float64
 4   input5      250 non-null    float64
 5   input6      250 non-null    float64
 6   input7      250 non-null    float64
 7   input8      250 non-null    float64
 8   input9      250 non-null    float64
 9   input10     250 non-null    float64
 10  in_control  250 non-null    int64
 11  fault1      250 non-null    int64
 12  fault2      250 non-null    int64
 13  fault3      250 non-null    int64
 14  fault4      250 non-null    int64
 15  fault5      250 non-null    int64
 16  fault6      250 non-null    int64
dtypes: float64(10), int64(7)
memory usage: 33.3 KB
```

We can see that there are no missing values in the dataset and all the variables are numerical variables.

We can see that there are no missing values in the dataset and all the variables are numerical variables.

Summary of numerical variables

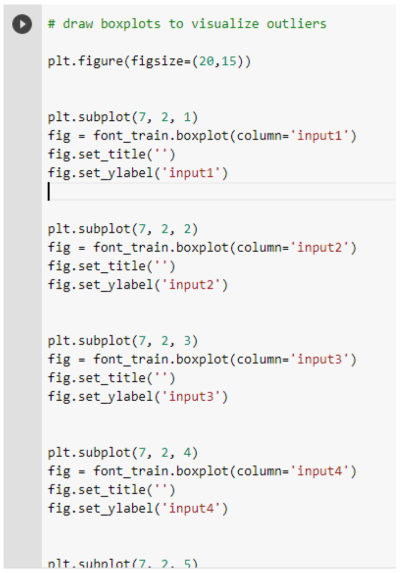
There are 2500 numerical variables in train and 250 in test datasets.

10 are continuous variables and 7 are discrete variable.

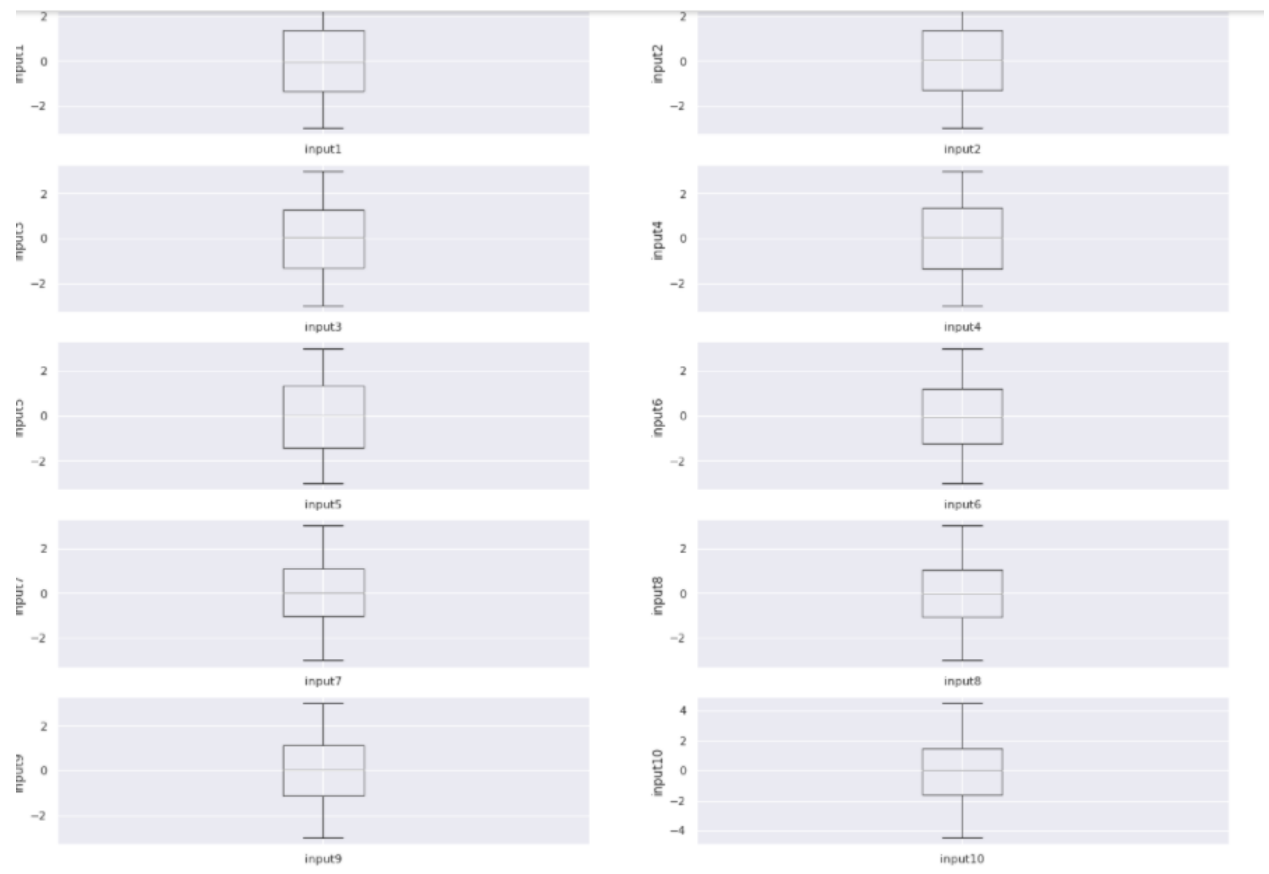
The discrete variables are target variables.

There are no missing values in the dataset.

5. **Outliers:**
   On closer inspection, we can suspect that all the continuous variables may contain outliers. I will draw boxplots to visualize outliers in the above variables.



```
# draw boxplots to visualize outliers

plt.figure(figsize=(20,15))


plt.subplot(7, 2, 1)
fig = font_train.boxplot(column='input1')
fig.set_title('')
fig.set_ylabel('input1')


plt.subplot(7, 2, 2)
fig = font_train.boxplot(column='input2')
fig.set_title('')
fig.set_ylabel('input2')


plt.subplot(7, 2, 3)
fig = font_train.boxplot(column='input3')
fig.set_title('')
fig.set_ylabel('input3')


plt.subplot(7, 2, 4)
fig = font_train.boxplot(column='input4')
fig.set_title('')
fig.set_ylabel('input4')


plt.subplot(7, 2, 5)
```



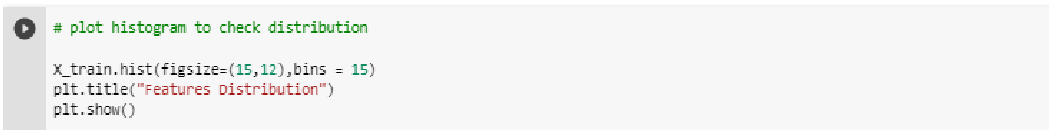
The above boxplots confirm that there are no outliers in these variables.

Handle outliers with SVM:

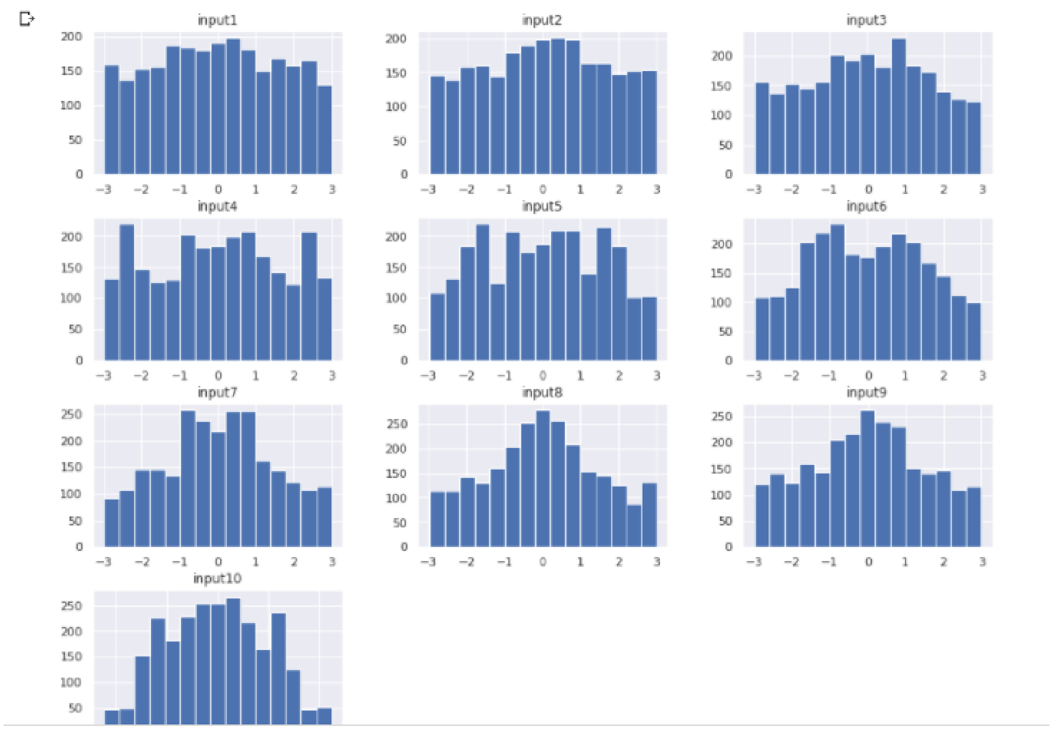
There are 2 variants of SVMs. They are hard-margin variant of SVM and soft-margin variant of SVM.

One version of SVM is called soft-margin variant of SVM. In this case, we can have a few points incorrectly classified or classified with a margin less than 1. But for every such point, we have to pay a penalty in the form of C parameter, which controls the outliers. Here, the message is that since the dataset contains outliers, so the value of C should be high while training the model and the margin should be soft.

6. **Declare feature vector and target variable**



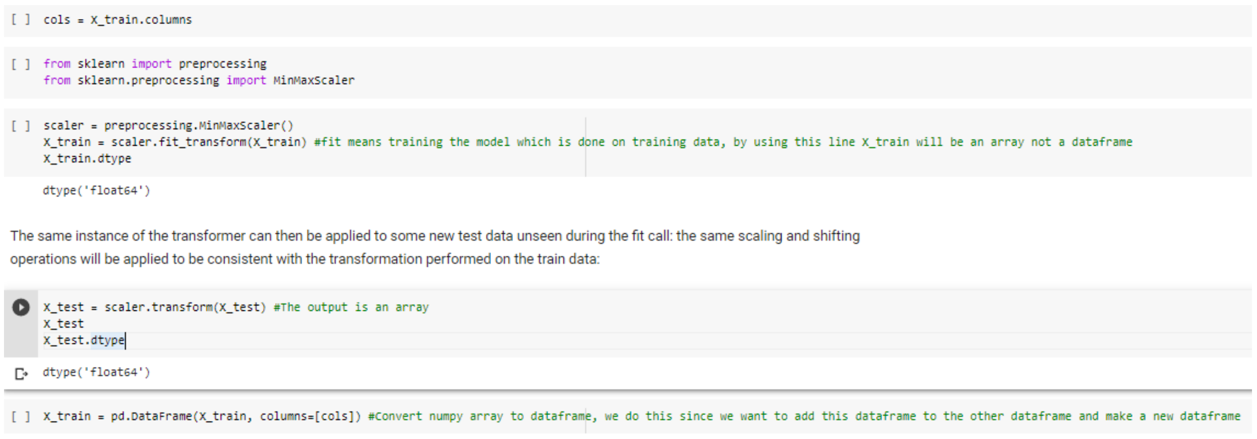
```
# plot histogram to check distribution

X_train.hist(figsize=(15,12),bins = 15)
plt.title("Features Distribution")
plt.show()
```



Also, here, We can see that some of the 10 continuous variables are skewed.

7. **Feature Scaling**

A standardization is scaling features to lie between a given minimum and maximum value, often between zero and one, or so that the maximum absolute value of each feature is scaled to unit size. This can be achieved using MinMaxScaler or MaxAbsScaler, respectively. The motivation to use this scaling include robustness to very small standard deviations of features and preserving zero entries in sparse data.



```
[ ] cols = X_train.columns
```

```
[ ] from sklearn import preprocessing
    from sklearn.preprocessing import MinMaxScaler
```

```
[ ] scaler = preprocessing.MinMaxScaler()
    X_train = scaler.fit_transform(X_train) #fit means training the model which is done on training data, by using this line X_train will be an array not a dataframe
    X_train.dtype

    dtype('float64')
```

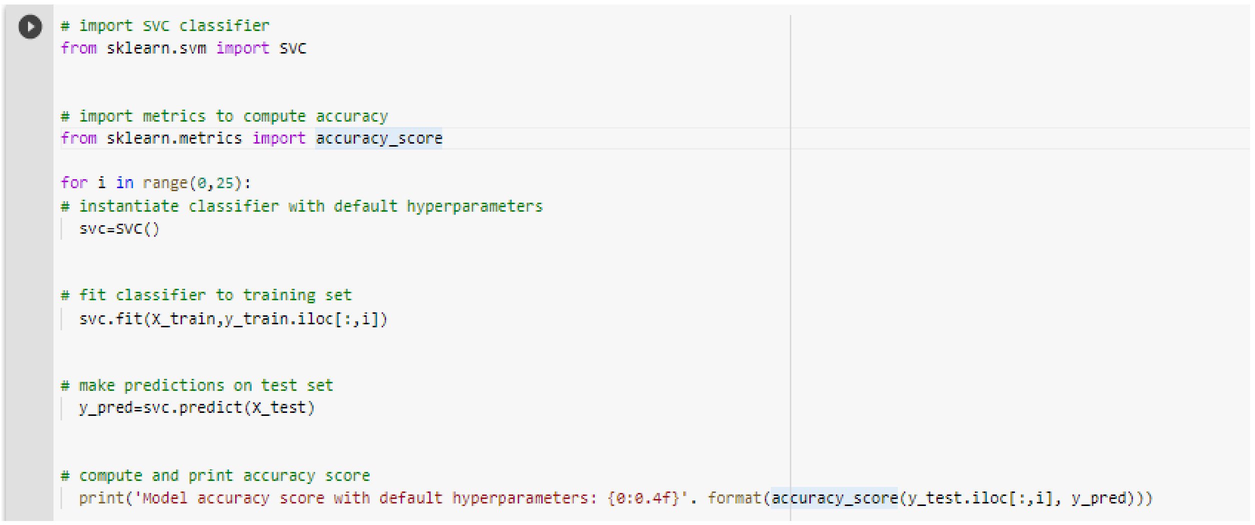
The same instance of the transformer can then be applied to some new test data unseen during the fit call: the same scaling and shifting operations will be applied to be consistent with the transformation performed on the train data:

```
X_test = scaler.transform(X_test) #The output is an array
X_test
X_test.dtype

dtype('float64')
```

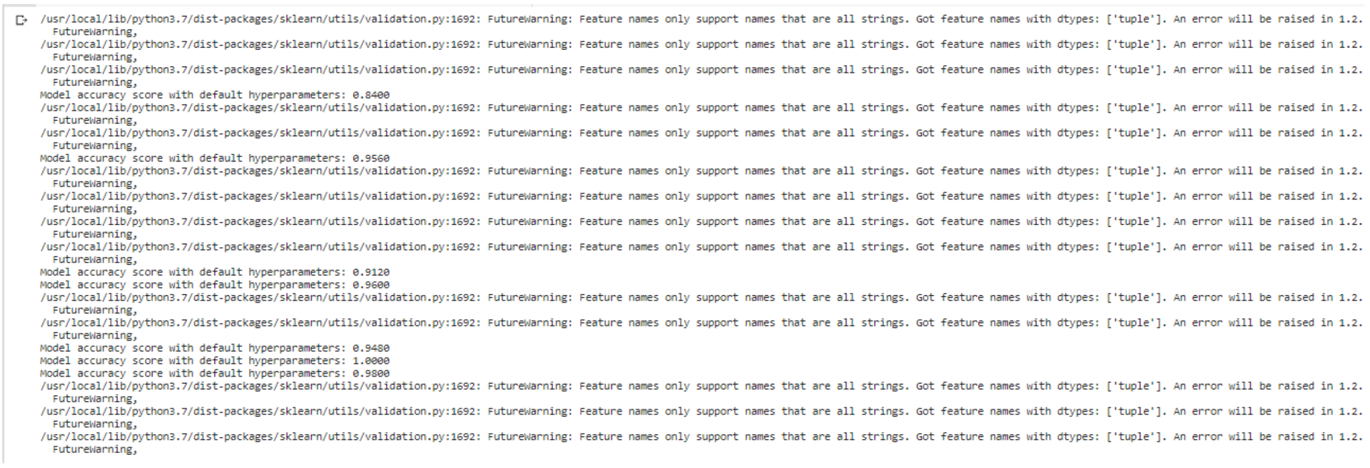
```
[ ] X_train = pd.DataFrame(X_train, columns=[cols]) #Convert numpy array to dataframe, we do this since we want to add this dataframe to the other dataframe and make a new dataframe
```

8. **Run SVM with default hyperparameters**

We have support vector classifiers. I will use two of them: 'linear' and 'RBF' classifiers



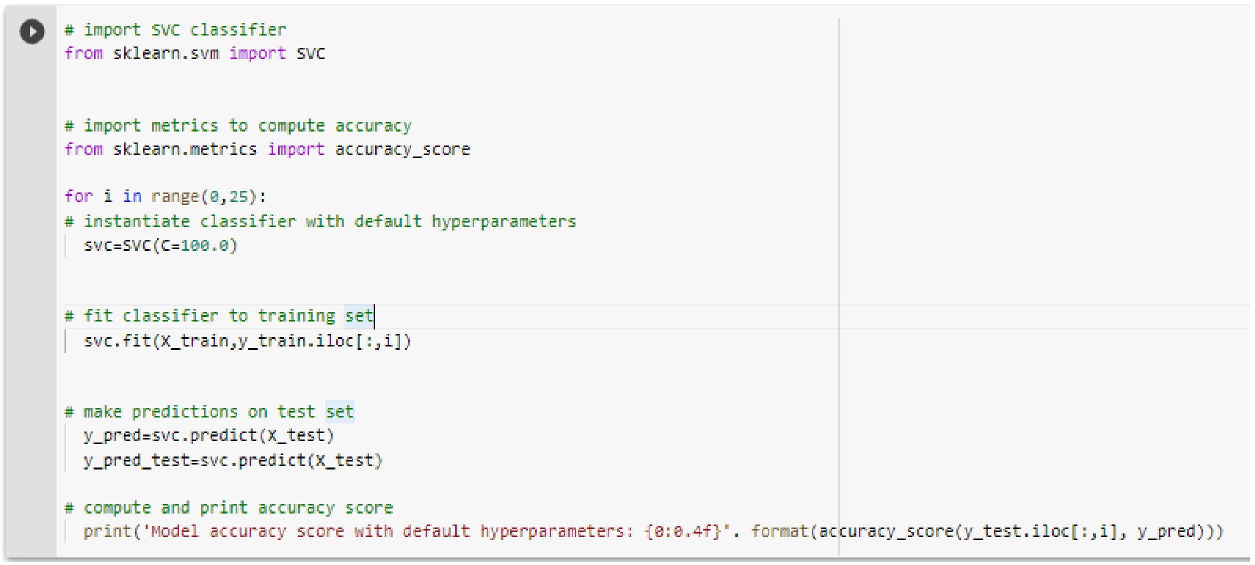
```python
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score

for i in range(0,25):
# instantiate classifier with default hyperparameters
  svc=SVC()


# fit classifier to training set
  svc.fit(X_train,y_train.iloc[:,i])


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


# compute and print accuracy score
  print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test.iloc[:,i], y_pred)))
```



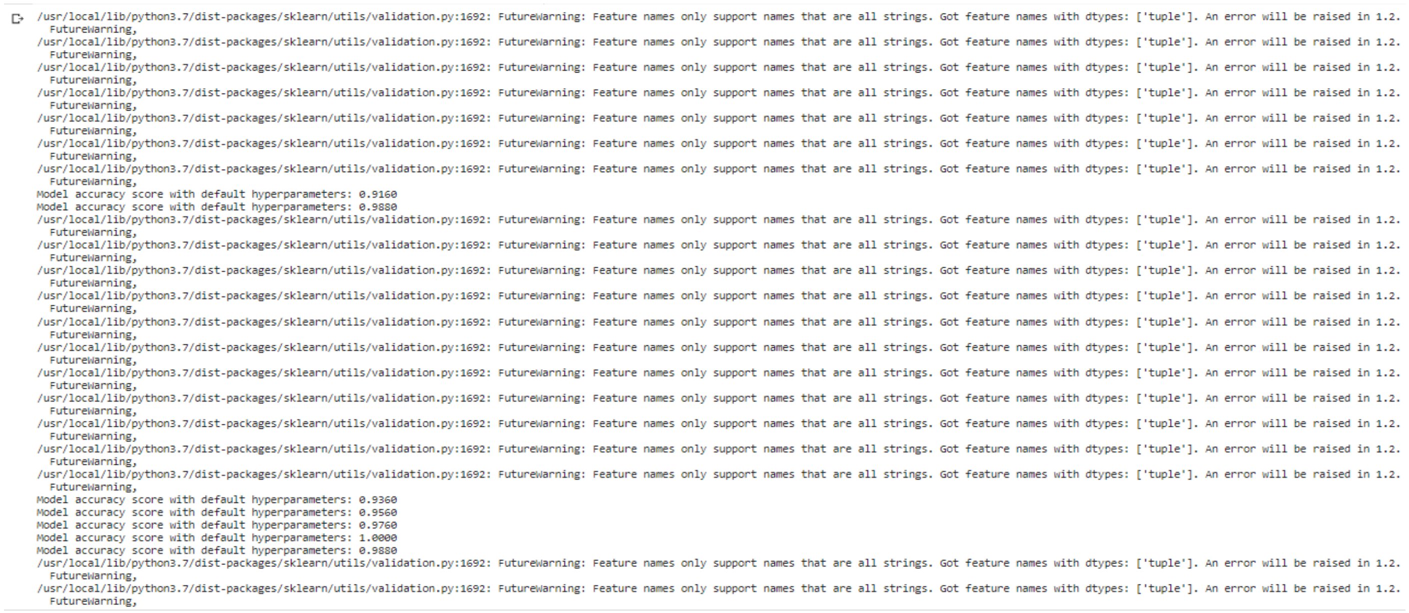
```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.8400
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9560
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9120
Model accuracy score with default hyperparameters: 0.9600
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9480
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 0.9800
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
```

Multi-class classification problem was solved as multiple binary classification problems by using for loop(instead we could use one vs rest or one vs one methods). Model accuracy score with default hyperparameters for different target columns are: 0.9615, 0.9872, 1

9. **Run SVM with RBF kernel and C=100.0**
   We have seen that there are outliers in our dataset. So, we should increase the value of C as higher C means fewer outliers. So, I will run SVM with kernel=rbf and C=100.0.



```python
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score

for i in range(0,25):
# instantiate classifier with default hyperparameters
  svc=SVC(C=100.0)


# fit classifier to training set
  svc.fit(X_train,y_train.iloc[:,i])


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

# compute and print accuracy score
  print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test.iloc[:,i], y_pred)))
```



```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9160
Model accuracy score with default hyperparameters: 0.9880
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9360
Model accuracy score with default hyperparameters: 0.9560
Model accuracy score with default hyperparameters: 0.9760
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 0.9880
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all strings. Got feature names with dtypes: ['tuple']. An error will be raised in 1.2.
  FutureWarning,
```

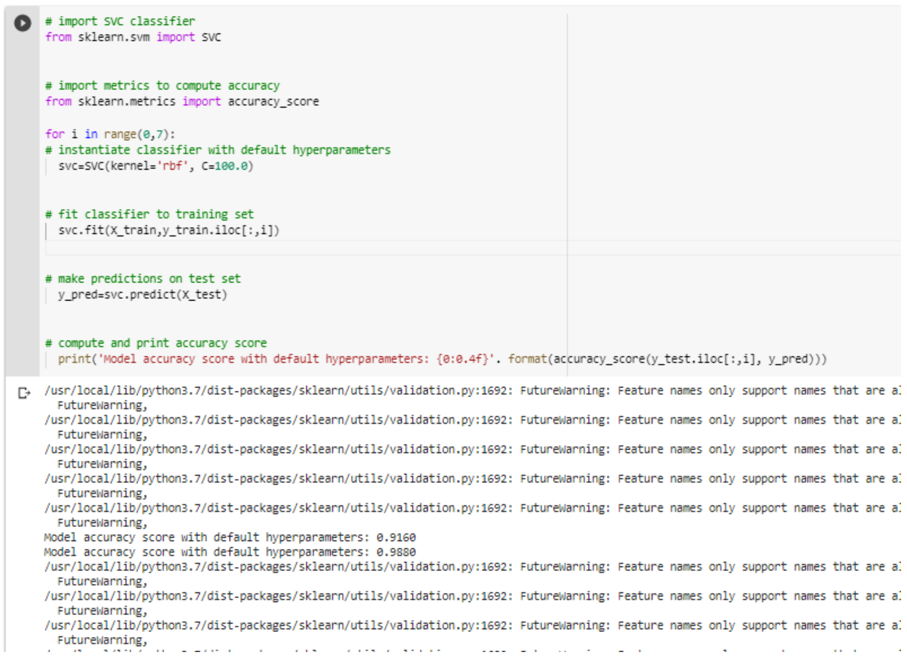
Model accuracy score with rbf kernel and C=100.0 was increased. In other words, the error or difference between the actual and predicted test results is decreased.

We can see that we obtain a higher accuracy with C=100.0 as higher C means less outliers and accuracy made better for almost all of the classes except one of them.

Now, I will further increase the value of C=1000.0 and check accuracy.

I also do linear kernel with C=1, 100, and 1000

10. Run SVM with linear kernel and C=10000:

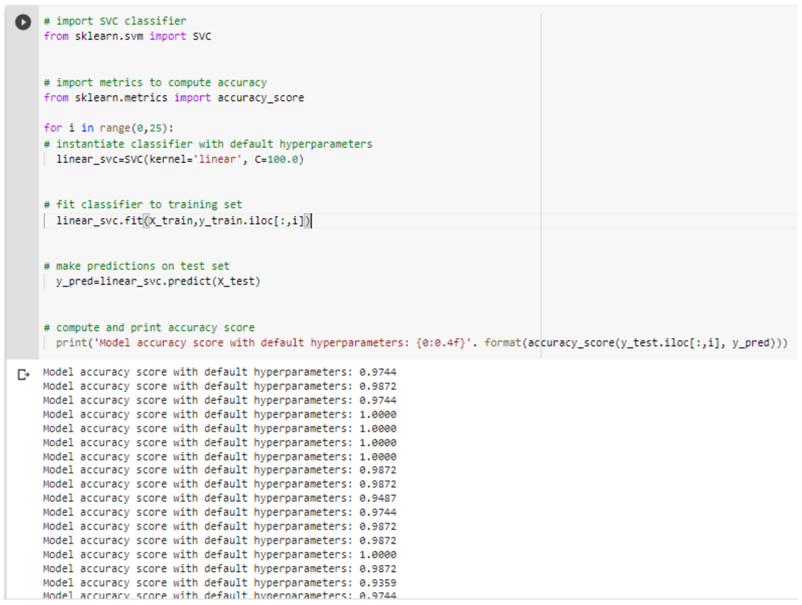


```
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score

for i in range(0,7):
# instantiate classifier with default hyperparameters
  svc=SVC(kernel='rbf', C=100.0)


# fit classifier to training set
  svc.fit(X_train,y_train.iloc[:,i])


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


# compute and print accuracy score
  print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test.iloc[:,i], y_pred)))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
Model accuracy score with default hyperparameters: 0.9160
Model accuracy score with default hyperparameters: 0.9880
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are a
  FutureWarning,
```

We can see that perfomance of 'rbf' was so better. This is because non-linear separable dataset

11. **Run SVM with linear kernel and C=100.0**



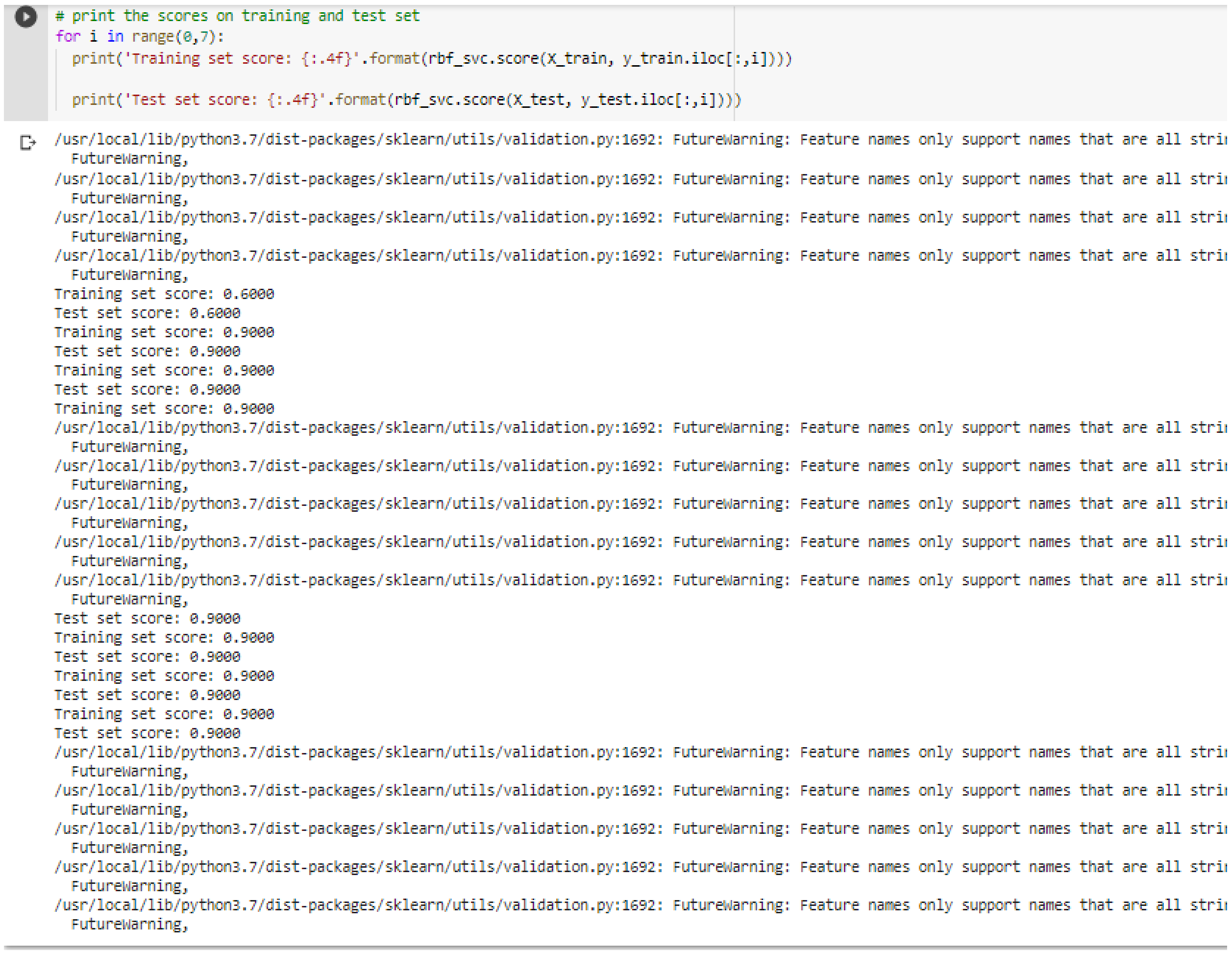
```
# import SVC classifier
from sklearn.svm import SVC


# import metrics to compute accuracy
from sklearn.metrics import accuracy_score

for i in range(0,25):
# instantiate classifier with default hyperparameters
  linear_svc=SVC(kernel='linear', C=100.0)


# fit classifier to training set
  linear_svc.fit(X_train,y_train.iloc[:,i])


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


# compute and print accuracy score
  print('Model accuracy score with default hyperparameters: {0:0.4f}'. format(accuracy_score(y_test.iloc[:,i], y_pred)))
```

```
Model accuracy score with default hyperparameters: 0.9744
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 0.9744
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 0.9487
Model accuracy score with default hyperparameters: 0.9744
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 1.0000
Model accuracy score with default hyperparameters: 0.9872
Model accuracy score with default hyperparameters: 0.9359
Model accuracy score with default hyperparameters: 0.9744
```

We see that we can obtain higher accuracy with C=100.0 and C=1000.0 as compared to C=1.0. But 'RBF' kernel has better performance than the linear for this non-linear separable dataset.

12. **Compare the train-set and test-set accuracy**

Now, I will compare the train-set and test-set accuracy to check for overfitting.

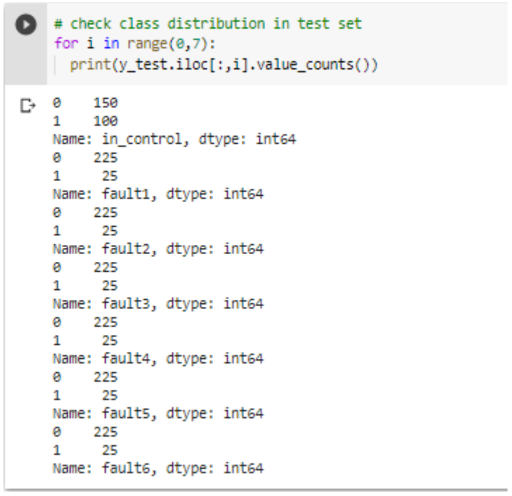


```
# print the scores on training and test set
for i in range(0,7):
  print('Training set score: {:.4f}'.format(rbf_svc.score(X_train, y_train.iloc[:,i])))

  print('Test set score: {:.4f}'.format(rbf_svc.score(X_test, y_test.iloc[:,i])))
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
Training set score: 0.6000
Test set score: 0.6000
Training set score: 0.9000
Test set score: 0.9000
Training set score: 0.9000
Test set score: 0.9000
Training set score: 0.9000
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
Test set score: 0.9000
Training set score: 0.9000
Test set score: 0.9000
Training set score: 0.9000
Test set score: 0.9000
Training set score: 0.9000
Test set score: 0.9000
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
/usr/local/lib/python3.7/dist-packages/sklearn/utils/validation.py:1692: FutureWarning: Feature names only support names that are all stri
  FutureWarning,
```

If differences between test score and training score in the above result are zero which mean it is a good model/fit. or the first class, train and test scores are 0.6. Very low training score and low test score means under-fitting in this case which has high bias and low variance.

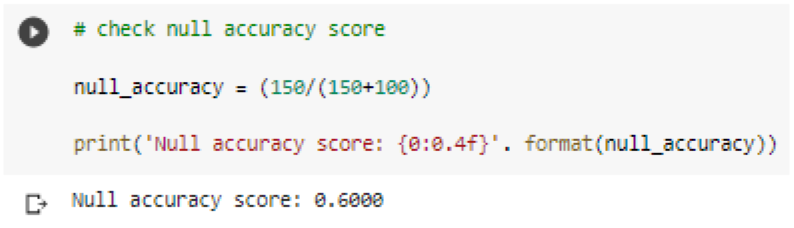
13. Compare model accuracy with null accuracy



```
# check class distribution in test set
for i in range(0,7):
  print(y_test.iloc[:,i].value_counts())
```

```
0    150
1    100
Name: in_control, dtype: int64
0    225
1     25
Name: fault1, dtype: int64
0    225
1     25
Name: fault2, dtype: int64
0    225
1     25
Name: fault3, dtype: int64
0    225
1     25
Name: fault4, dtype: int64
0    225
1     25
Name: fault5, dtype: int64
0    225
1     25
Name: fault6, dtype: int64
```

We can see that the occurences of most frequent class 0 is 150 for the first class and the least frequence class 1 is 100. This can also be seen for other classes. So, we can calculate null accuracy by dividing 150 by total number of occurences.



```
# check null accuracy score

null_accuracy = (150/(150+100))

print('Null accuracy score: {0:0.4f}'. format(null_accuracy))
```

```
Null accuracy score: 0.6000
```

We can see that our model accuracy score is 0.6 for the first class and null accuracy score is 0.6. So, we can conclude that our SVM classifier is doing a very good job in predicting the class labels.

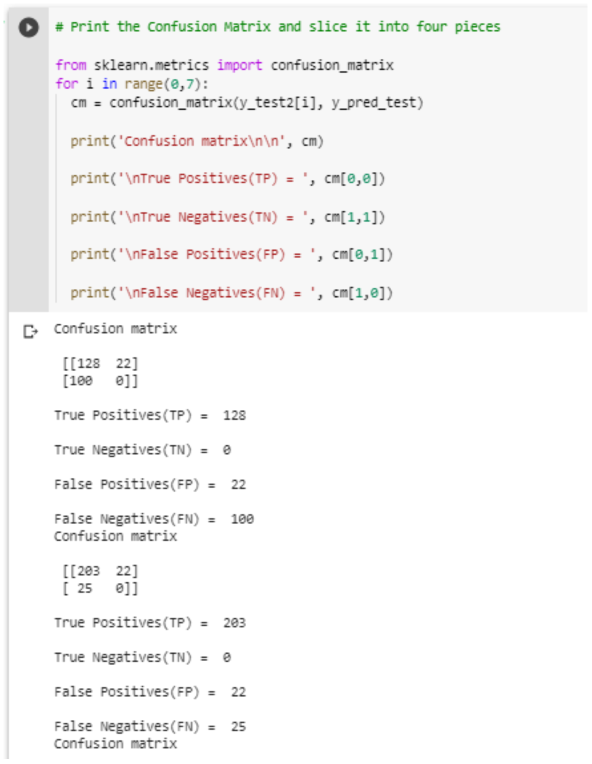
14. **Comments**

We get maximum accuracy with RBFkernel with C=100.0. Based on the above analysis we can conclude that our classification model accuracy is very good. Our model is doing a very good job in terms of predicting the class labels.

But we should note that here, we have an imbalanced dataset. The problem is that accuracy is an inadequate measure for quantifying predictive performance in the imbalanced dataset problem.

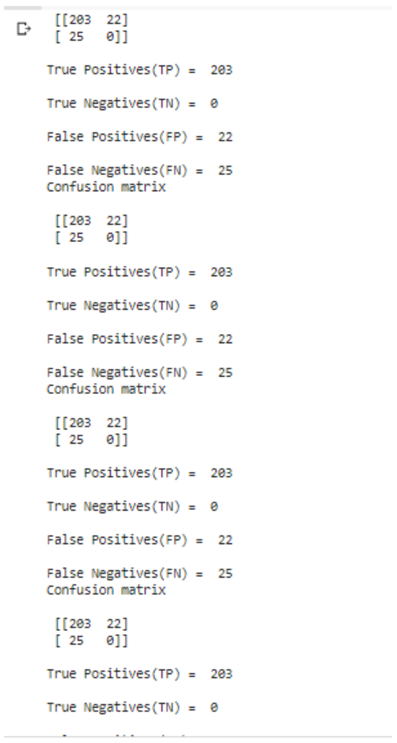
So, we must explore alternative metrics that provide better guidance in selecting models. We would like to know the underlying distribution of values and the type of errors our classifier is making.

One such metric to analyze the model performance in imbalanced classes problem is Confusion matrix.



```
# Print the Confusion Matrix and slice it into four pieces

from sklearn.metrics import confusion_matrix
for i in range(0,7):
  cm = confusion_matrix(y_test2[i], y_pred_test)

  print('Confusion matrix\n\n', cm)

  print('\nTrue Positives(TP) = ', cm[0,0])

  print('\nTrue Negatives(TN) = ', cm[1,1])

  print('\nFalse Positives(FP) = ', cm[0,1])

  print('\nFalse Negatives(FN) = ', cm[1,0])
```

```
Confusion matrix

 [[128  22]
 [100   0]]

True Positives(TP) =  128

True Negatives(TN) =  0

False Positives(FP) =  22

False Negatives(FN) =  100
Confusion matrix

 [[203  22]
 [ 25   0]]

True Positives(TP) =  203

True Negatives(TN) =  0

False Positives(FP) =  22

False Negatives(FN) =  25
Confusion matrix
```



```
 [[203  22]
 [ 25   0]]

True Positives(TP) =  203

True Negatives(TN) =  0

False Positives(FP) =  22

False Negatives(FN) =  25
Confusion matrix

 [[203  22]
 [ 25   0]]

True Positives(TP) =  203

True Negatives(TN) =  0

False Positives(FP) =  22

False Negatives(FN) =  25
Confusion matrix

 [[203  22]
 [ 25   0]]

True Positives(TP) =  203

True Negatives(TN) =  0

False Positives(FP) =  22

False Negatives(FN) =  25
Confusion matrix

 [[203  22]
 [ 25   0]]

True Positives(TP) =  203

True Negatives(TN) =  0
```

The confusion matrix for the last class shows 225 + 22 = 247 correct predictions and 0 + 3 = 3 incorrect predictions.

In this case, we have:

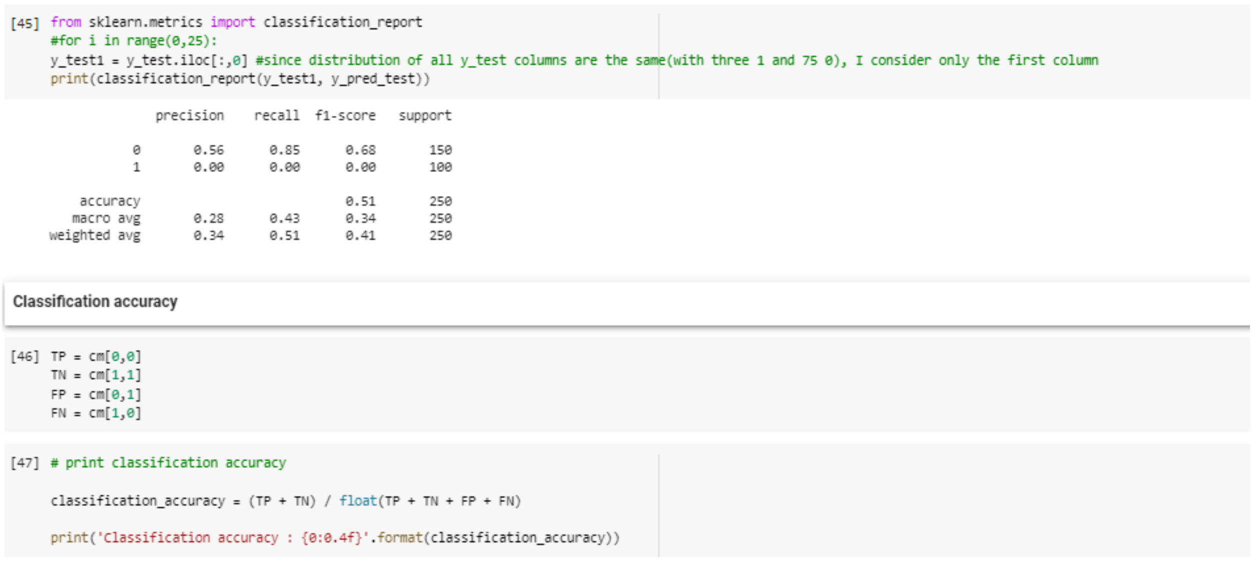
True Positives (Actual Positive:1 and Predict Positive:1) - 225

True Negatives (Actual Negative:0 and Predict Negative:0) - 22

False Positives (Actual Negative:0 but Predict Positive:1) - 0 (Type I error)

False Negatives (Actual Positive:1 but Predict Negative:0) - 3 (Type II error)

15. **Classification Report**
    Classification report is another way to evaluate the classification model performance. It displays the precision, recall, f1 and support scores for the model. I have described these terms in later.
    We can print a classification report as follows:



```
[45] from sklearn.metrics import classification_report
     #for i in range(0,25):
     y_test1 = y_test.iloc[:,0] #since distribution of all y_test columns are the same(with three 1 and 75 0), I consider only the first column
     print(classification_report(y_test1, y_pred_test))

                   precision    recall  f1-score   support

                0       0.56      0.85      0.68       150
                1       0.00      0.00      0.00       100

         accuracy                           0.51       250
        macro avg       0.28      0.43      0.34       250
     weighted avg       0.34      0.51      0.41       250
```

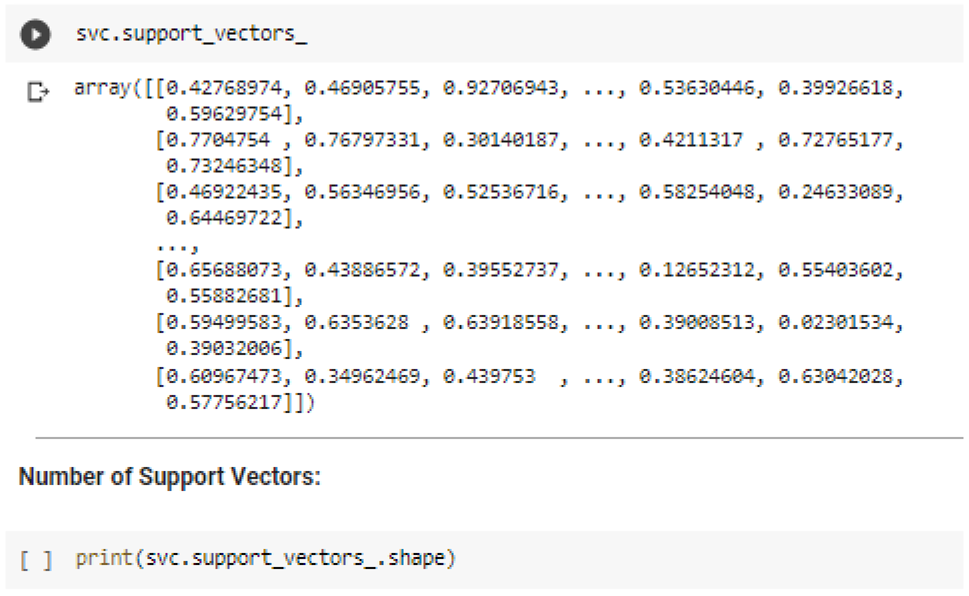
Classification accuracy

```
[46] TP = cm[0,0]
     TN = cm[1,1]
     FP = cm[0,1]
     FN = cm[1,0]
```

```
[47] # print classification accuracy

     classification_accuracy = (TP + TN) / float(TP + TN + FP + FN)

     print('Classification accuracy : {0:0.4f}'.format(classification_accuracy))
```

16. Support Vectors:

In Scikit-Learn, the identity of these points are stored in the support_vectors_ attribute of the classifier:



```
svc.support_vectors_
```

```
array([[0.42768974, 0.46905755, 0.92706943, ..., 0.53630446, 0.39926618,
        0.59629754],
       [0.7704754 , 0.76797331, 0.30140187, ..., 0.4211317 , 0.72765177,
        0.73246348],
       [0.46922435, 0.56346956, 0.52536716, ..., 0.58254048, 0.24633089,
        0.64469722],
       ...,
       [0.65688073, 0.43886572, 0.39552737, ..., 0.12652312, 0.55403602,
        0.55882681],
       [0.59499583, 0.6353628 , 0.63918558, ..., 0.39008513, 0.02301534,
        0.39032006],
       [0.60967473, 0.34962469, 0.439753  , ..., 0.38624604, 0.63042028,
        0.57756217]])
```

Number of Support Vectors:

```
[ ] print(svc.support_vectors_.shape)
```

So, the number of support vectors that RBF classifier was found is 125.

The number of support vectors depends on how much slack we allow and the distribution of the data. If we allow a large amount of slack, we will have many support vectors. If we allow very little slack, we will have very few support vectors. The accuracy depends on finding the right level of slack for the data being analyzed. For some data it will not be possible to get a high level of accuracy, we must simply find the best fit we can.