In [35]:
```
import pandas as pd
import numpy as np
import seaborn as sns
from scipy import stats
from matplotlib import pyplot as plt
```

In [24]:
```
df=pd.read_csv("https://raw.githubusercontent.com/dsrscientist/dataset3/refs/heads/main/glass
df
```

Out[24]:

	0	1	2	3	4	5	6	7	8	9	10
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.00	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.00	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.00	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.00	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.00	0.0	1
...	...	...	...	...	...	...	...	...	...	...	...
209	210	1.51623	14.14	0.00	2.88	72.61	0.08	9.18	1.06	0.0	7
210	211	1.51685	14.92	0.00	1.99	73.06	0.00	8.40	1.59	0.0	7
211	212	1.52065	14.36	0.00	2.02	73.42	0.00	8.44	1.64	0.0	7
212	213	1.51651	14.38	0.00	1.94	73.61	0.00	8.48	1.57	0.0	7
213	214	1.51711	14.23	0.00	2.08	73.36	0.00	8.62	1.67	0.0	7

214 rows × 11 columns

In [25]:
```
df.shape
```

Out[25]:
```
(214, 11)
```

In [26]:
```
df.isnull().sum()
```

Out[26]:
```
0     0
1     0
2     0
3     0
4     0
5     0
6     0
7     0
8     0
9     0
10    0
dtype: int64
```

In [27]:
```
df[10].value_counts()
```

Out[27]:
```
10
2    76
1    70
7    29
3    17
5    13
6     9
Name: count, dtype: int64
```

In [28]:
```
df.describe()
```

Out[28]:

	0	1	2	3	4	5	6	7	8
count	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000	214.000000
mean	107.500000	1.518365	13.407850	2.684533	1.444907	72.650935	0.497056	8.956963	0.175047
std	61.920648	0.003037	0.816604	1.442408	0.499270	0.774546	0.652192	1.423153	0.497219
min	1.000000	1.511150	10.730000	0.000000	0.290000	69.810000	0.000000	5.430000	0.000000
25%	54.250000	1.516522	12.907500	2.115000	1.190000	72.280000	0.122500	8.240000	0.000000
50%	107.500000	1.517680	13.300000	3.480000	1.360000	72.790000	0.555000	8.600000	0.000000
75%	160.750000	1.519157	13.825000	3.600000	1.630000	73.087500	0.610000	9.172500	0.000000
max	214.000000	1.533930	17.380000	4.490000	3.500000	75.410000	6.210000	16.190000	3.150000

Above statistics shows that data is across all attributes is not in same range, so will normalize tha data first.

Preparing Dataset

Adding meaningful column/attribute names

In [29]:
```
names=['Id', 'RI','Na', 'Mg','Al', 'Si', 'K', 'Ca', 'Ba', 'Fe', 'glass_type']
df.columns=names
df.head()
```

Out[29]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	glass_type
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
1	2	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1
4	5	1.51742	13.27	3.62	1.24	73.08	0.55	8.07	0.0	0.0	1

Removing unnecessary columns

In [22]:
```
df=df.drop("Id")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Cell In[22], line 1
----> 1 df=df.drop("Id")

File ~\anaconda3\Lib\site-packages\pandas\core\frame.py:5258, in DataFrame.drop(self, lab
els, axis, index, columns, level, inplace, errors)
   5110 def drop(
   5111     self,
   5112     labels: IndexLabel = None,
   (...)
   5119     errors: IgnoreRaise = "raise",
   5120 ) -> DataFrame | None:
   5121     """
   5122     Drop specified labels from rows or columns.
   5123 
   (...)
   5256             weight  1.0     0.8
   5257     """
```

In [21]:
```
df.head(3)
```

Out[21]:

	Id	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe	glass_type
0	1	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0	1
2	3	1.51618	13.53	3.55	1.54	72.99	0.39	7.78	0.0	0.0	1
3	4	1.51766	13.21	3.69	1.29	72.61	0.57	8.22	0.0	0.0	1

Checking outliers through Z-score

In [31]:
```
z=abs(stats.zscore(df))

#np.where(z>3)

df=df[(z<3).all(axis=1)]

#df.shape
```

Separating Features and Label

In [32]:
```
features=['RI','Na','Mg','Al','Si','K','Ca','Ba','Fe']
label=['glass_type']

X=df[features]

Y=df[label]
```

In [33]:
```
X.shape
```

Out[33]: (194, 9)

In [34]:
```
type(X)
```

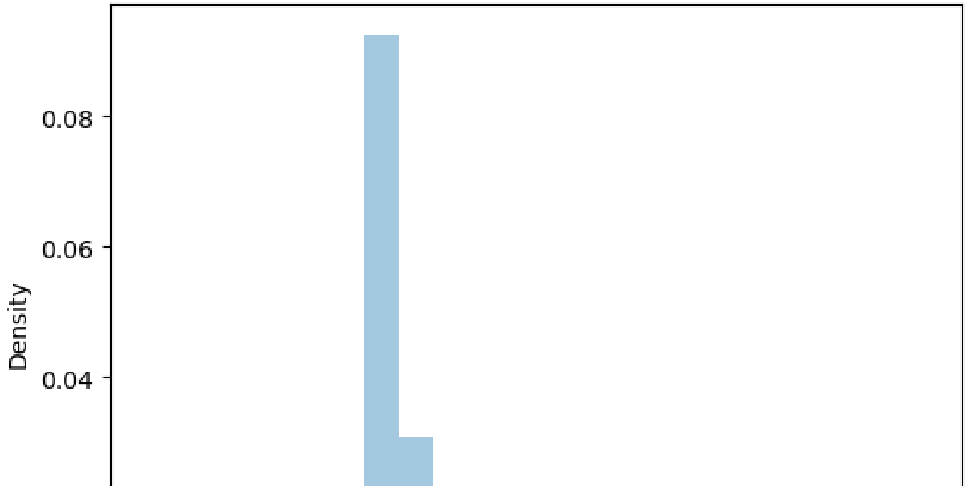
Out[34]: pandas.core.frame.DataFrame

Data Visualization

In [36]:
```
x2=X.values

for i in range(1,9):
    sns.distplot(x2[i])
    plt.xlabel(features[i])
    plt.show()
```

```
sns.distplot(x2[i])
```

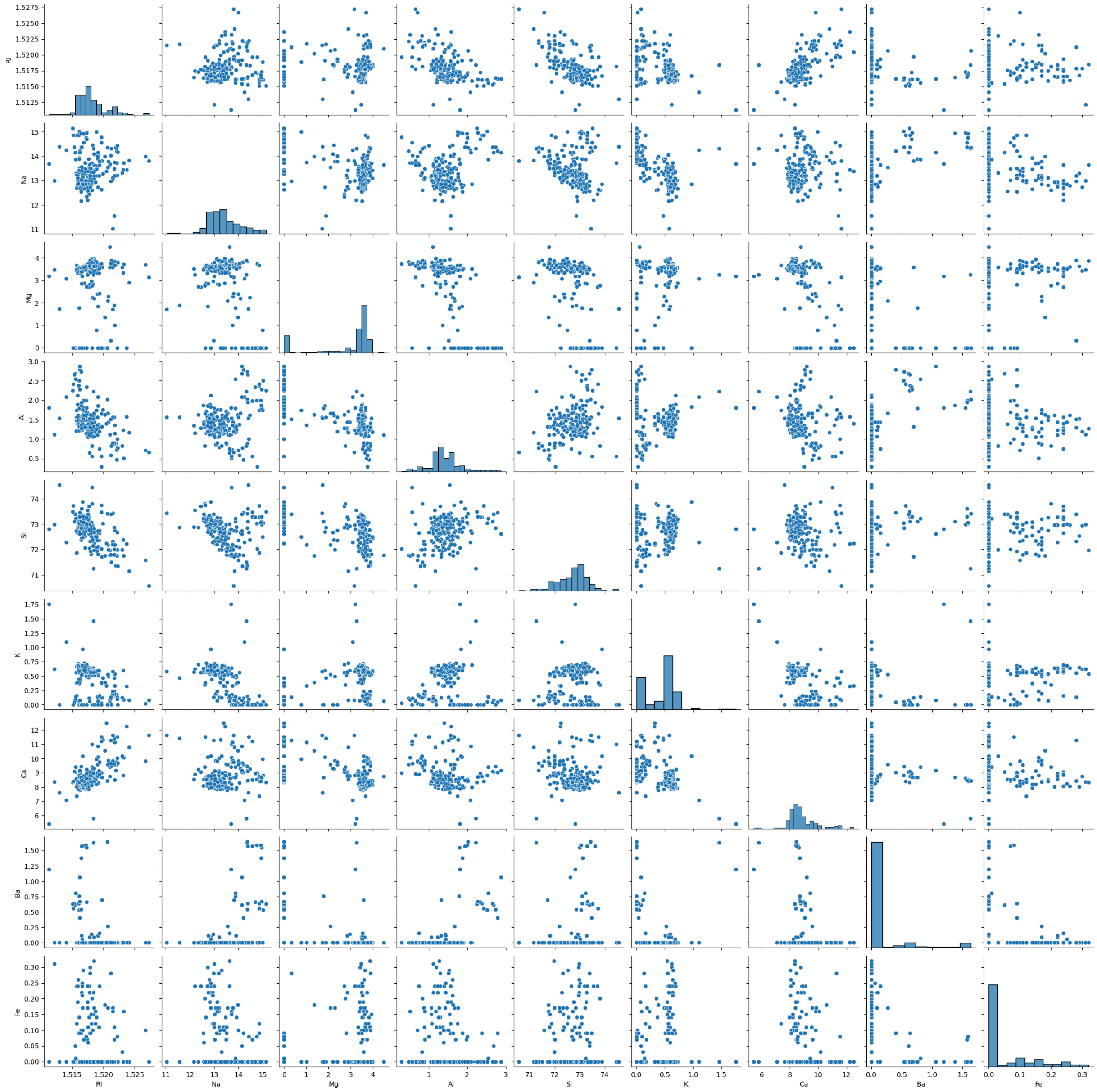


Above diagrams shows that our dataset is skewed either on positive side or negative side and data is not normalized.

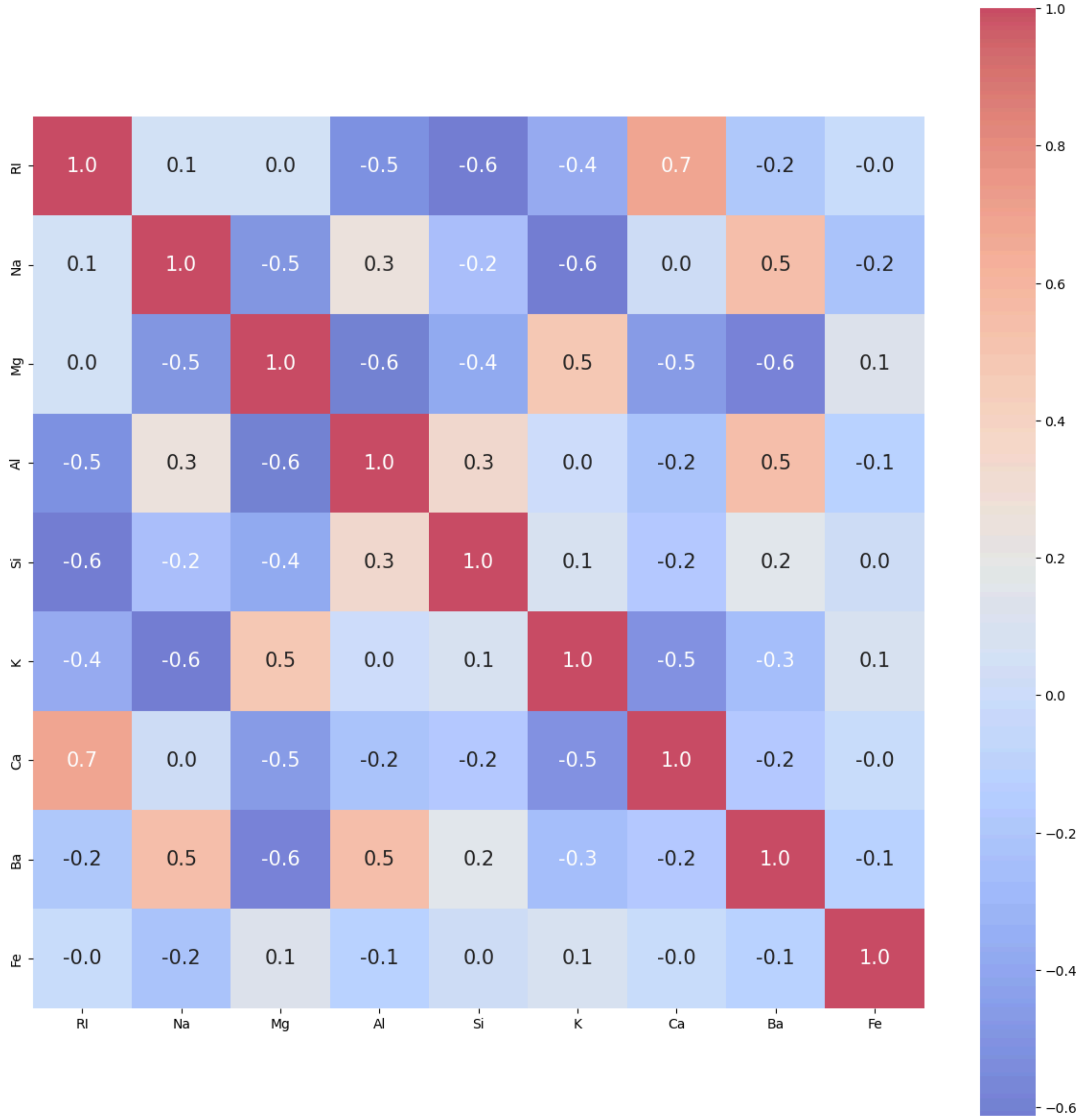
In [38]:
```
x2=pd.DataFrame(X)
plt.figure(figsize=(8,8))
sns.pairplot(data=x2)
plt.show()
```

```
C:\Users\Rashmi\anaconda3\Lib\site-packages\seaborn\axisgrid.py:118: UserWarning: The figure
layout has changed to tight
  self._figure.tight_layout(*args, **kwargs)
```

```
<Figure size 800x800 with 0 Axes>
```



In [40]:
```
ation=X.corr()
igure(figsize=(15,15))
eatmap(corelation,cbar=True,square=True,annot=True,fmt='.1f',annot_kws={'size':15},xticklabels
how()
```



Our Diagram shows correlation between different features conclusion:

1. RI and Ca have strong correlation between each other
2. Al and ba have intermediate correlation between each other

Scaling tha data(1-0 range)

In [41]:
```
# normalizing/Scaling the data

from sklearn.preprocessing import MinMaxScaler
scaler= MinMaxScaler()
#scaler.fit(X)
#X=Scaler.transform(X)
#X=pd.DataFrame(X)
```

In [42]:
```
X.head(2)
```

Out[42]:

	RI	Na	Mg	Al	Si	K	Ca	Ba	Fe
0	1.52101	13.64	4.49	1.10	71.78	0.06	8.75	0.0	0.0
1	1.51761	13.89	3.60	1.36	72.73	0.48	7.83	0.0	0.0

In [43]:
```
Y.head(2)
```

Out[43]:

	glass_type
0	1
1	1

Scalling the features

In [44]:
```
from sklearn import preprocessing
X=preprocessing.scale(X)
```

Visualizing Data after Preprocessing

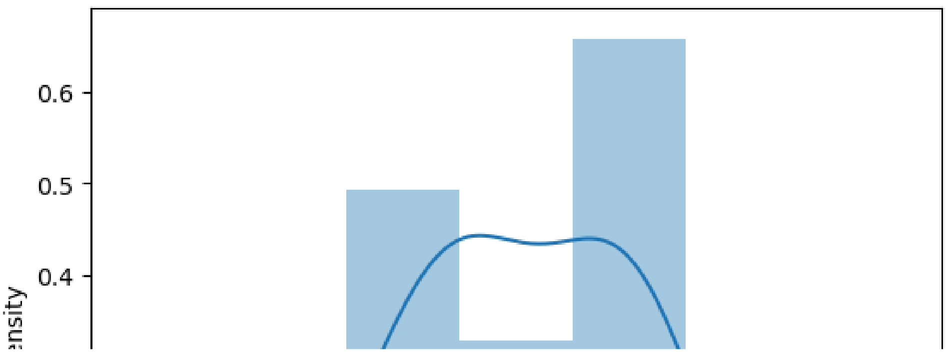
In [46]:
```
x2=X

from matplotlib import pyplot as plt
import seaborn as sns
for i in range(1,9):
    sns.distplot(x2[i])
    plt.xlabel(features[i])
    plt.show()
```

```
similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see
https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751 (https://gist.github.co
m/mwaskom/de44147ed2974457ad6372750bbe5751)

  sns.distplot(x2[i])
```



Above diagrams show that after preprocessing skewness is reduced and data is more normalized.

Train Test Split

In [47]:
```
from sklearn.model_selection import train_test_split
```

In [51]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.25, random_state=0,strati
```

In [52]:
```
##Flattening the array

y_train=y_train.values.ravel()
y_test=y_test.values.ravel()
```

In [53]:
```
print('Shape of X_train= ' + str(X_train.shape))
print('Shape of X_test= ' + str(X_test.shape))
print('Shape of y_train= ' + str(y_train.shape))
print('Shape of y_test= ' + str(y_test.shape))
```

```
Shape of X_train= (145, 9)
Shape of X_test= (49, 9)
Shape of y_train= (145,)
Shape of y_test= (49,)
```

Applying Different Machine Models

1. KNN

In [54]:
```
from sklearn.metrics import accuracy_score
from sklearn.neighbors import KNeighborsClassifier
```

In [57]:
```
Scores=[]

for i in range(2,11):
    knn=KNeighborsClassifier(n_neighbors=i)
    knn.fit(X_train, y_train)
    score=knn.score(X_test,y_test)
    Scores.append(score)

print(knn.score(X_train,y_train))
print(Scores)
```

```
0.6896551724137931
[0.7142857142857143, 0.6530612244897959, 0.7346938775510204, 0.7142857142857143, 0.673469387
755102, 0.6530612244897959, 0.6938775510204082, 0.6938775510204082, 0.6938775510204082]
```

2. Decision Tree

In [58]:
```
from sklearn.tree import DecisionTreeClassifier

Scores=[]

for i in range(1):
    tree=DecisionTreeClassifier(random_state=0)
    tree.fit(X_train,y_train)
    score=tree.score(X_test,y_test)
    Scores.append(score)

print(tree.score(X_train,y_train))
print(Scores)
```

```
1.0
[0.5510204081632653]
```

3. Logistic Regression

In [59]:
```
from sklearn.linear_model import LogisticRegression

Scores=[]

for i in range(1):
    logistic =LogisticRegression(random_state=0, solver='lbfgs', multi_class='multinomial',ma
    logistic.fit(X_train, y_train)
    score = logistic.score(X_test,y_test)
    Scores.append(score)

print(logistic.score(X_train, y_train))
print(Scores)
```

```
0.7517241379310344
[0.6938775510204082]
```

4. SVC Classifier (Non-Linear Kernel)

In [61]:
```
from sklearn.svm import SVC

Scores=[]

for i in range(1):
    svc=SVC(gamma='auto')
    svc.fit(X_train,y_train)
    score=svc.score(X_test, y_test)
    Scores.append(score)

print(svc.score(X_train, y_train))
print(Scores)
```

```
0.7517241379310344
[0.7551020408163265]
```

5. SVC Classifier (Linear Kernel)

In [62]:
```
from sklearn.svm import LinearSVC

Scores=[]

for i in range(1):
    svc=LinearSVC(random_state=0)
    svc.fit(X_train, y_train)
    score=svc.score(X_test, y_test)
    Scores.append(score)

print(svc.score(X_train, y_train))
print(Scores)
```

```
0.7517241379310344
[0.6938775510204082]

C:\Users\Rashmi\anaconda3\Lib\site-packages\sklearn\svm\_classes.py:32: FutureWarning: The d
efault value of `dual` will change from `True` to `'auto'` in 1.5. Set the value of `dual` e
xplicitly to suppress the warning.
  warnings.warn(
C:\Users\Rashmi\anaconda3\Lib\site-packages\sklearn\svm\_base.py:1242: ConvergenceWarning: L
iblinear failed to converge, increase the number of iterations.
  warnings.warn(
```

6. Random Forest

In [63]:
```
from sklearn.ensemble import RandomForestClassifier

Scores=[]
Range=[10,20,30,50,70,80,100,120]

for i in range(1):
    forest=RandomForestClassifier(criterion='gini',n_estimators=10, min_samples_leaf=1, min_s
    forest.fit(X_train, y_train)
    score=forest.score(X_test, y_test)

print(forest.score(X_train, y_train))
print(score)
```

```
0.9724137931034482
0.7755102040816326
```

In [ ]:

7. Gradient Decent Tree Boosting

In [64]:
```
from sklearn.ensemble import GradientBoostingClassifier

gd=GradientBoostingClassifier(n_estimators=100, learning_rate=1.0, max_depth=1, random_state=
gd.fit(X_train, y_train)
score=gd.score(X_test, y_test)

print(gd.score(X_train, y_train))
print(score)
```

```
0.9724137931034482
0.6326530612244898
```

Summary

```
Out of all above models:
    1. Random forest is giving best result with:
```

```
        Training accuracy: 0.9724137931034482
        Testing accuracy: 0.7755102040816326

But since it is overfitting we will choose next best model that is:

    2. SVM (Non Linear Kernel)
        Training accuracy: 0.7517241379310344
        Testing accuracy: 0.7551020408163265
```

In [ ]: