SDSC3006 Steel Plates Faults Detection

In [84]:
```
import math
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn import svm
from sklearn import metrics
from xgboost import XGBClassifier
from sklearn import preprocessing
from sklearn.decomposition import PCA
from imblearn.over_sampling import SMOTE
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import mean_squared_error
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.metrics import classification_report, confusion_matrix,accuracy
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report
```

Data Import & Description

In [7]:
```
df = pd.read_csv('faults.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1941 entries, 0 to 1940
Data columns (total 34 columns):
 #   Column                 Non-Null Count  Dtype  
---  ------                 --------------  -----  
 0   X_Minimum              1941 non-null   int64  
 1   X_Maximum              1941 non-null   int64  
 2   Y_Minimum              1941 non-null   int64  
 3   Y_Maximum              1941 non-null   int64  
 4   Pixels_Areas           1941 non-null   int64  
 5   X_Perimeter            1941 non-null   int64  
 6   Y_Perimeter            1941 non-null   int64  
 7   Sum_of_Luminosity      1941 non-null   int64  
 8   Minimum_of_Luminosity  1941 non-null   int64  
 9   Maximum_of_Luminosity  1941 non-null   int64  
 10  Length_of_Conveyer     1941 non-null   int64  
 11  TypeOfSteel_A300       1941 non-null   int64  
 12  TypeOfSteel_A400       1941 non-null   int64  
 13  Steel_Plate_Thickness  1941 non-null   int64  
 14  Edges_Index            1941 non-null   float64
 15  Empty_Index            1941 non-null   float64
 16  Square_Index           1941 non-null   float64
 17  Outside_X_Index        1941 non-null   float64
 18  Edges_X_Index          1941 non-null   float64
 19  Edges_Y_Index          1941 non-null   float64
 20  Outside_Global_Index   1941 non-null   float64
 21  LogOfAreas             1941 non-null   float64
 22  Log_X_Index            1941 non-null   float64
 23  Log_Y_Index            1941 non-null   float64
 24  Orientation_Index      1941 non-null   float64
 25  Luminosity_Index       1941 non-null   float64
 26  SigmoidOfAreas         1941 non-null   float64
 27  Pastry                 1941 non-null   int64  
 28  Z_Scratch              1941 non-null   int64  
 29  K_Scatch               1941 non-null   int64  
 30  Stains                 1941 non-null   int64  
 31  Dirtiness              1941 non-null   int64  
 32  Bumps                  1941 non-null   int64  
 33  Other_Faults           1941 non-null   int64  
dtypes: float64(13), int64(21)
memory usage: 515.7 KB
```

In [8]:
```
df.head()
```

Out[8]:

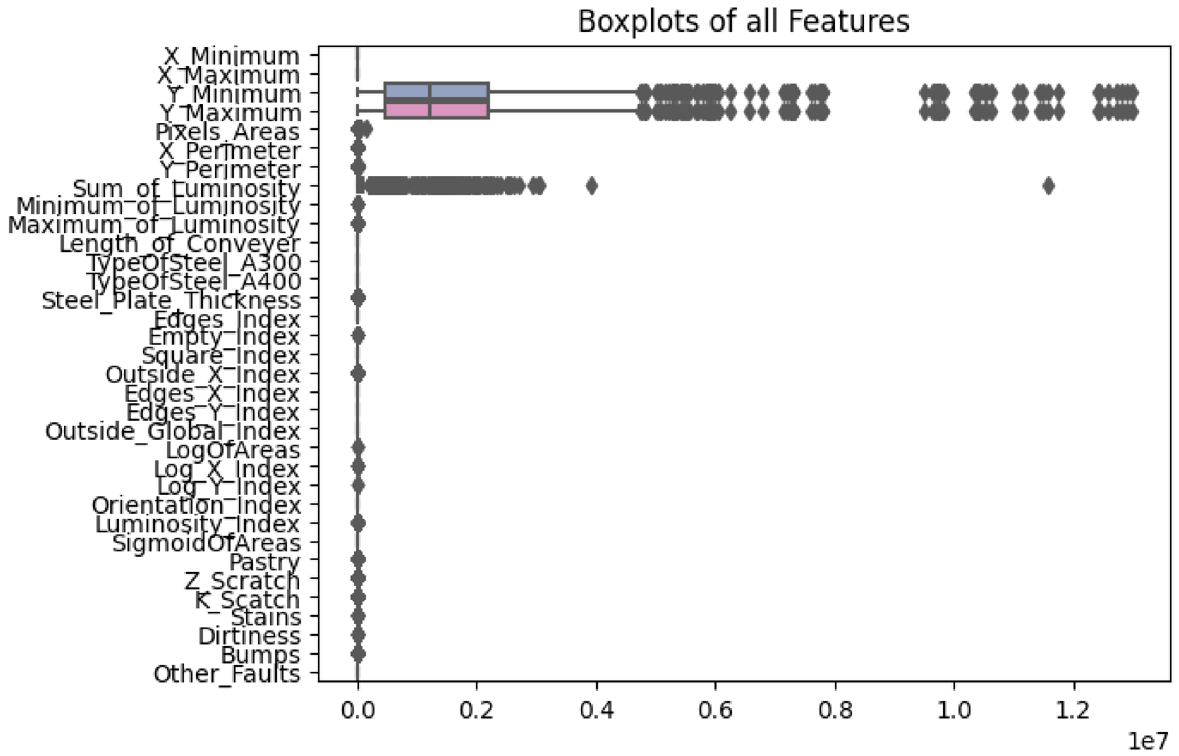
	X_Minimum	X_Maximum	Y_Minimum	Y_Maximum	Pixels_Areas	X_Perimeter	Y_Perim
0	42	50	270900	270944	267	17	
1	645	651	2538079	2538108	108	10	
2	829	835	1553913	1553931	71	8	
3	853	860	369370	369415	176	13	
4	1289	1306	498078	498335	2409	60	

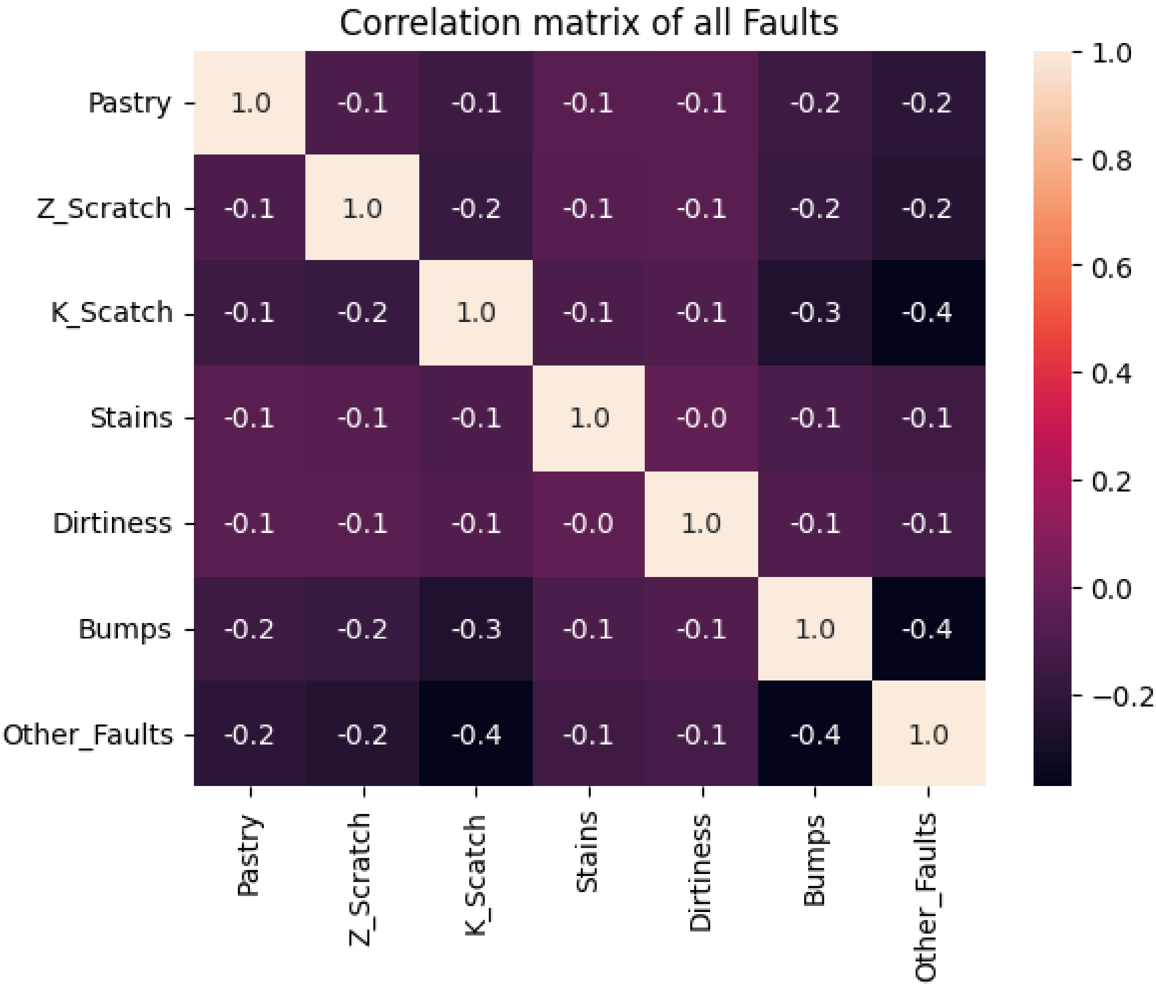
5 rows × 34 columns

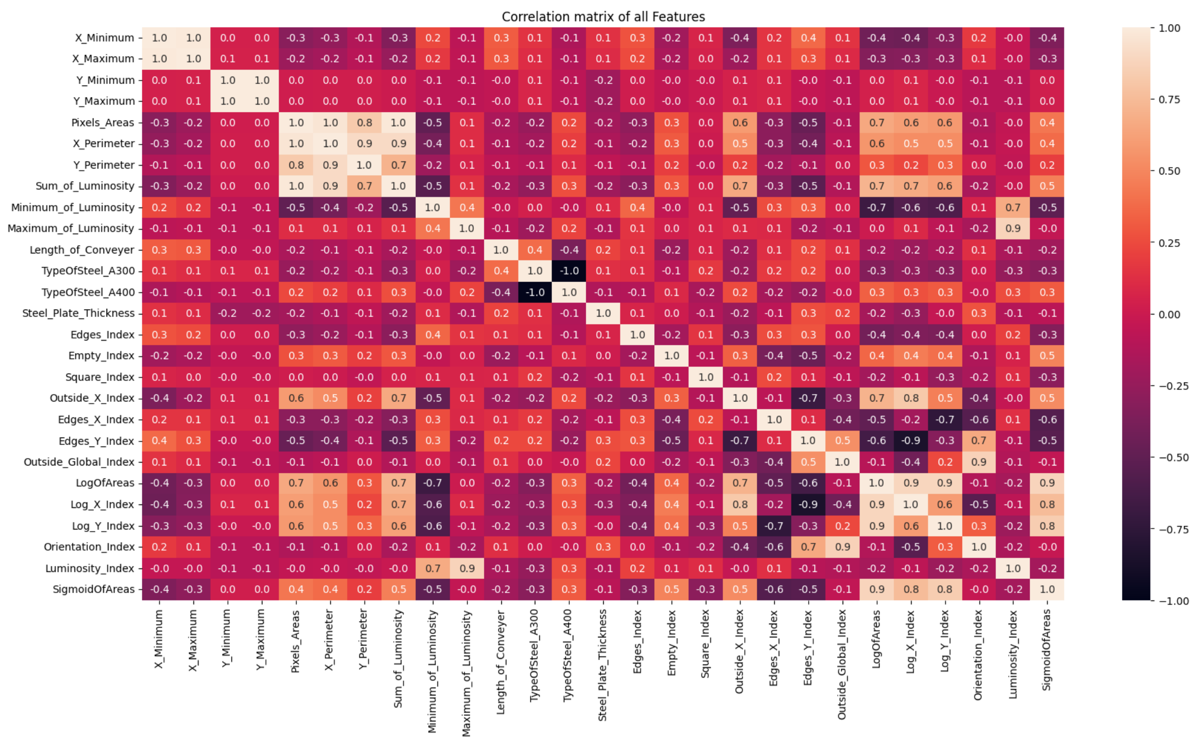
Data Exploration and Visualization

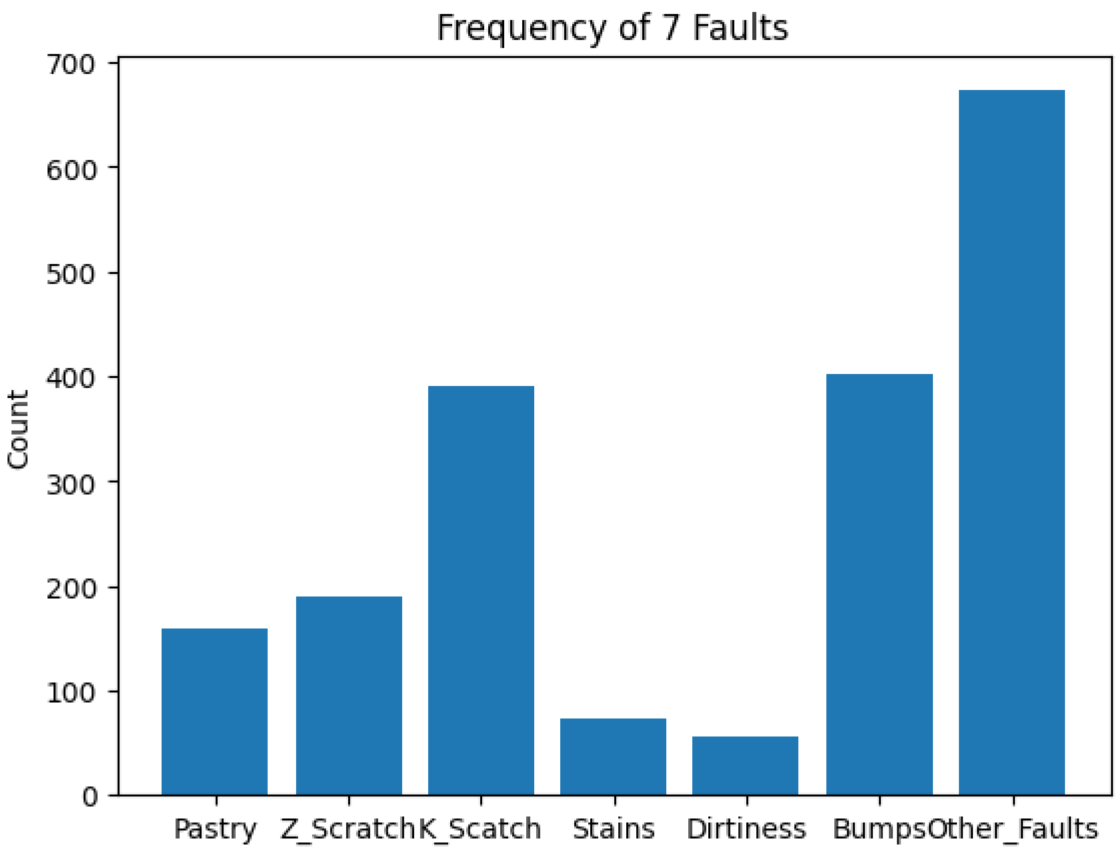
In [9]:
```
def DataExploration(dataframe):
```

```
    df = dataframe
    X=df.iloc[:,:27]
    Y=df.iloc[:,27:]

    # Box Plots
    ax = sns.boxplot(data=df, orient="h", palette="Set2")
    ax.set_title('Boxplots of all Features')
    plt.show()


    Pastry = df.query('Pastry==1')
    Z_Scratch = df.query('Z_Scratch==1')
    K_Scatch = df.query('K_Scatch==1')
    Stains =df.query('Stains==1')
    Dirtiness = df.query('Dirtiness==1')
    Bumps = df.query('Bumps==1')
    Other_Faults = df.query('Other_Faults==1')

    # Correlation matrix of all faults
    ax = sns.heatmap(Y.corr(), annot=True, fmt=".1f")
    ax.set_title('Correlation matrix of all Faults')
    plt.show()

    # Correlation matrix of all features
    plt.figure(figsize=(20, 10))
    ax = sns.heatmap(X.corr(), annot=True, fmt=".1f")
    ax.set_title('Correlation matrix of all Features')
    plt.show()

    # Frequency of 7 Faults
    fault_name=['Pastry', 'Z_Scratch', 'K_Scatch', 'Stains', 'Dirtiness', 'E
    value=[len(Pastry), len(Z_Scratch), len(K_Scatch), len(Stains), len(Dirt
    plt.bar(fault_name,value,width=0.8)
    plt.title('Frequency of 7 Faults')
    plt.ylabel('Count')
    plt.show()
DataExploration(df)
```




Correlation matrix of all Faults
Pastry
Z_Scratch
K_Scatch
Stains
Dirtiness
Bumps
Other_Faults
1.0
0.8
0.6
0.4
0.2
0.0
−0.2


Correlation matrix of all Features
X_Minimum
X_Maximum
Y_Minimum
Y_Maximum
Pixels_Areas
X_Perimeter
Y_Perimeter
Sum_of_Luminosity
Minimum_of_Luminosity
Maximum_of_Luminosity
Length_of_Conveyer
TypeOfSteel_A300
TypeOfSteel_A400
Steel_Plate_Thickness
Edges_Index
Empty_Index
Square_Index
Outside_X_Index
Edges_X_Index
Edges_Y_Index
Outside_Global_Index
LogOfAreas
Log_X_Index
Log_Y_Index
Orientation_Index
Luminosity_Index
SigmoidOfAreas
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00



No outliers: "X_Minimum","X_Maximum","Length_of_Conveyer","TypeOfSteel_A300","TypeOfSteel_A40

Have outliers: "Y_Minimum","Y_Maximum","Sum_of_Luminosity","Pixels_Areas","X_Perimeter","Y_Perimete

We don't handle the outliers since they may belong to the fewer Faults classes (e.g Stains/ Dirtiness/...)

Data Preprocessing

In [39]:
```
def DataPreprocessing(df):

    # Remove missing value
    df = df.dropna() # But no missing value, df do not change

    # Divide the dateset into features and faults
    faults =df[["Pastry","Z_Scratch","K_Scatch","Stains","Dirtiness","Bumps"
    X = df.drop(["Pastry","Z_Scratch","K_Scatch","Stains","Dirtiness","Bumps
    y = []
    for i in range(faults.shape[0]):
        if faults["Pastry"].values[i] == 1:
            y.append("Pastry")
        elif faults["Z_Scratch"].values[i] == 1:
            y.append("Z_Scratch")
        elif faults["K_Scatch"].values[i] == 1:
            y.append("K_Scatch")
        elif faults["Stains"].values[i] == 1:
            y.append("Stains")
        elif faults["Dirtiness"].values[i] == 1:
            y.append("Dirtiness")
```

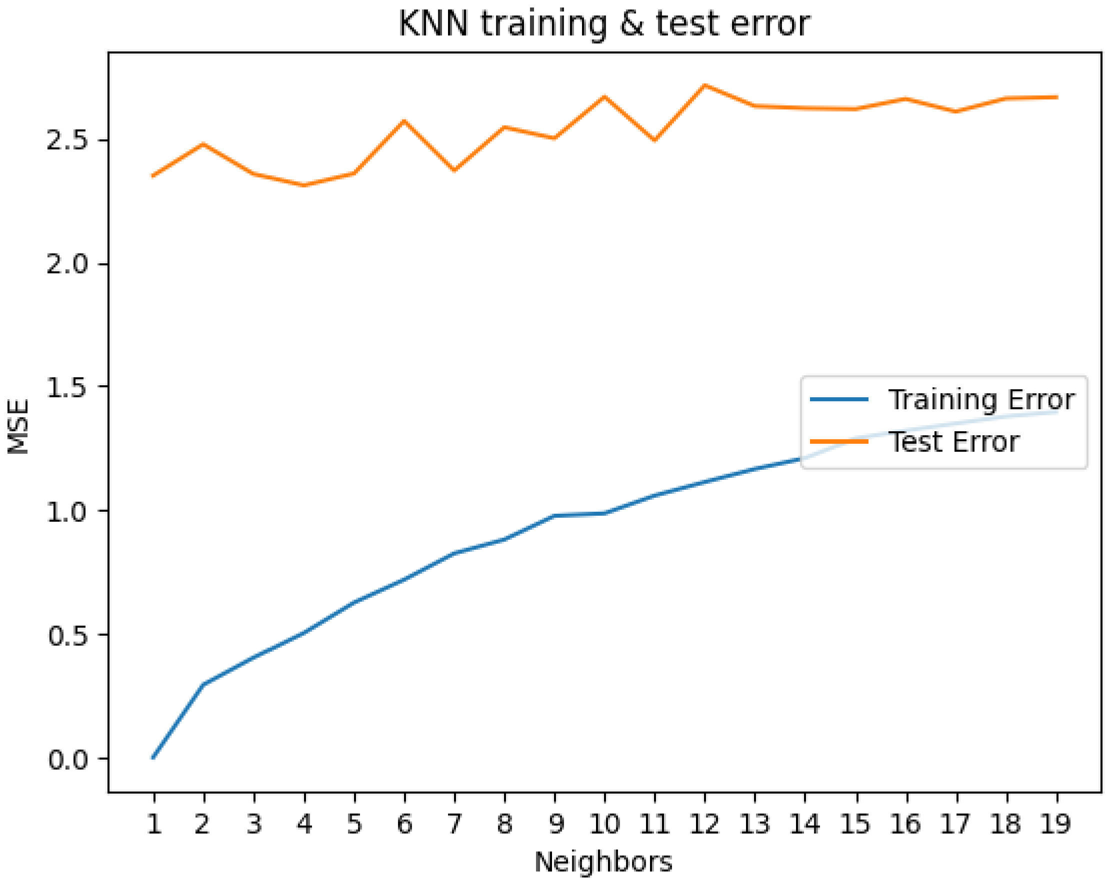
```
        elif faults["Bumps"].values[i] == 1:
            y.append("Bumps")
        else:
            y.append("Other_Faults")

    y=np.array(y)
    faultstype= pd.DataFrame({'faults':y})

    # Label Encoder
    le=LabelEncoder()
    y=le.fit_transform(y)


    # train test split
    X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, te

    # Min-max normalization (after spliting)
    X_train_minmax = pd.DataFrame(
        MinMaxScaler().fit_transform(X_train),
        columns = X_train.columns
    )
    X_test_minmax = pd.DataFrame(
        MinMaxScaler().fit_transform(X_test),
        columns = X_test.columns
    )

    # Normalization after spliting
    X_train_normalized = pd.DataFrame(
        StandardScaler().fit_transform(X_train),
        columns = X_train.columns
    )

    X_test_normalized = pd.DataFrame(
        StandardScaler().fit_transform(X_test),
        columns = X_test.columns
    )
    #X.plot(kind="density", layout=(6,5),subplots=True,sharex=False, sharey=
    #plt.show()
    #X.head()

    # oversample
    oversample = SMOTE()
    X_train_normalized, y_train = oversample.fit_resample(X_train_normalized

    return X_train_normalized, X_test_normalized, y_train, y_test
target_names=["Bump","Dirtiness","K_Scatch","Other_Faults","Pastry","Stains"
X_train_normalized, X_test_normalized, y_train, y_test = DataPreprocessing(d
```

Model Selection

In [67]:

```
params = range(1,20)
training_errors = []
test_errors = []
for p in params:
    clf = KNeighborsClassifier(n_neighbors=p)
    clf.fit(X_train_normalized, y_train)
    y_pred = clf.predict(X_train_normalized)
    training_err = mean_squared_error(y_train, y_pred)
    training_errors.append(training_err)
```

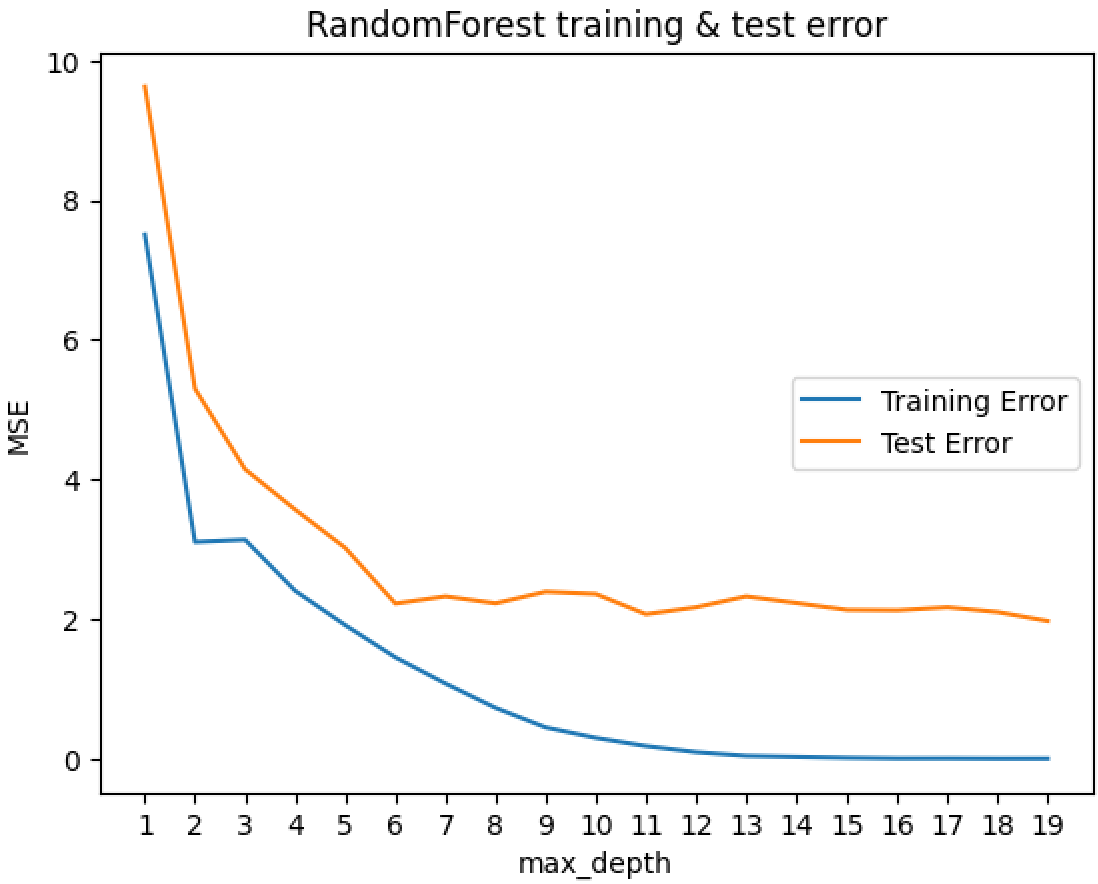
```
    y_pred = clf.predict(X_test_normalized)
    test_err = mean_squared_error(y_test, y_pred)
    test_errors.append(test_err)

error_table = pd.DataFrame()
error_table["degree"] = params
error_table["training_error"] = training_errors
error_table["test_error"] = test_errors

import matplotlib.pyplot as plt
plt.plot(error_table['degree'], error_table['training_error'], label = 'Trai
plt.plot(error_table['degree'], error_table['test_error'], label = 'Test Err
plt.title("KNN training & test error")
plt.ylabel("MSE")
plt.xlabel("Neighbors")
plt.legend(loc = 'center right')
plt.xticks(params, params)
plt.show()
```



In [72]:

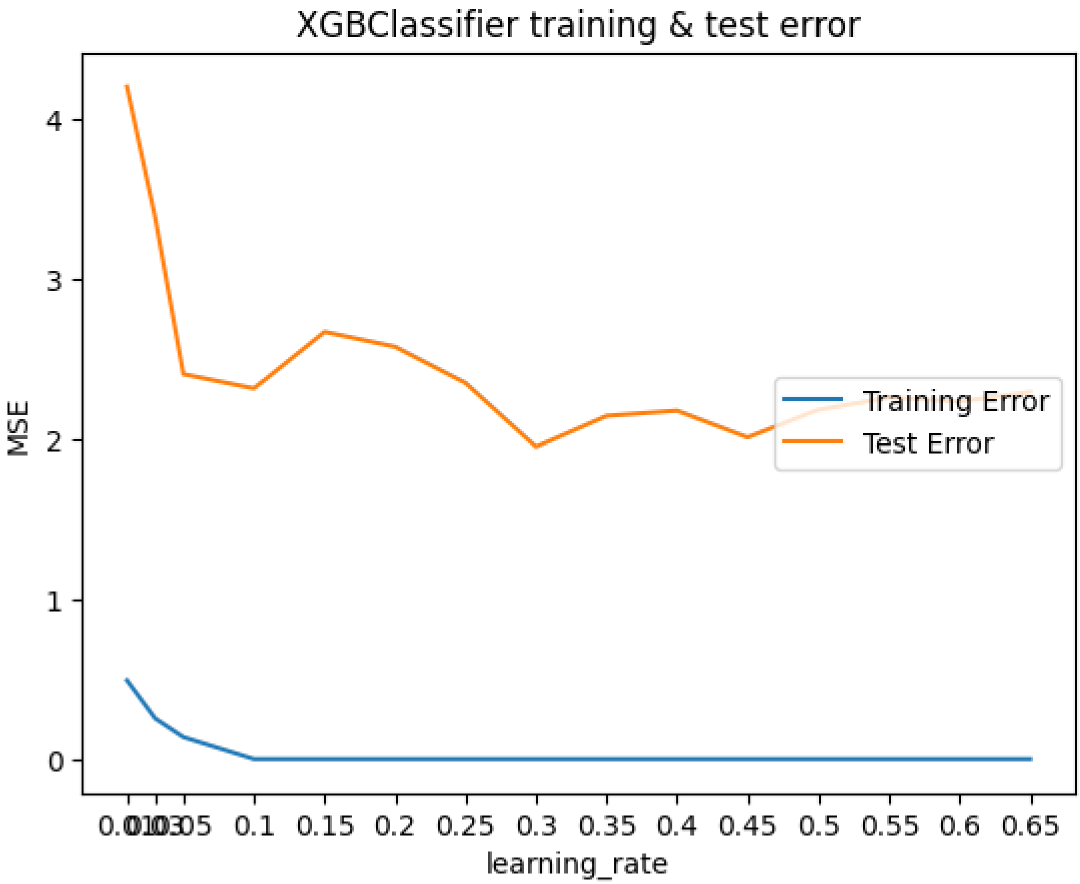
```
params = range(1,20)
training_errors = []
test_errors = []
for p in params:
    clf = RandomForestClassifier(max_depth =p, random_state=42)
    clf.fit(X_train_normalized, y_train)
    y_pred = clf.predict(X_train_normalized)
    training_err = mean_squared_error(y_train, y_pred)
    training_errors.append(training_err)
    y_pred = clf.predict(X_test_normalized)
    test_err = mean_squared_error(y_test, y_pred)
    test_errors.append(test_err)

error_table = pd.DataFrame()
error_table["degree"] = params
```

```
error_table["training_error"] = training_errors
error_table["test_error"] = test_errors

import matplotlib.pyplot as plt
plt.plot(error_table['degree'], error_table['training_error'], label = 'Trai
plt.plot(error_table['degree'], error_table['test_error'], label = 'Test Err
plt.title("RandomForest training & test error")
plt.ylabel("MSE")
plt.xlabel("max_depth")
plt.legend(loc = 'center right')
plt.xticks(params, params)
plt.show()
```



In [73]:
```
params = [0.01,0.03,0.05,0.1,0.15,0.2,0.25,0.3,0.35,0.4,0.45,0.5,0.55,0.6,0.
training_errors = []
test_errors = []
for p in params:
    clf = XGBClassifier(n_estimators=100, learning_rate= p)
    clf.fit(X_train_normalized, y_train)
    y_pred = clf.predict(X_train_normalized)
    training_err = mean_squared_error(y_train, y_pred)
    training_errors.append(training_err)
    y_pred = clf.predict(X_test_normalized)
    test_err = mean_squared_error(y_test, y_pred)
    test_errors.append(test_err)

error_table = pd.DataFrame()
error_table["degree"] = params
error_table["training_error"] = training_errors
error_table["test_error"] = test_errors

import matplotlib.pyplot as plt
plt.plot(error_table['degree'], error_table['training_error'], label = 'Trai
plt.plot(error_table['degree'], error_table['test_error'], label = 'Test Err
```

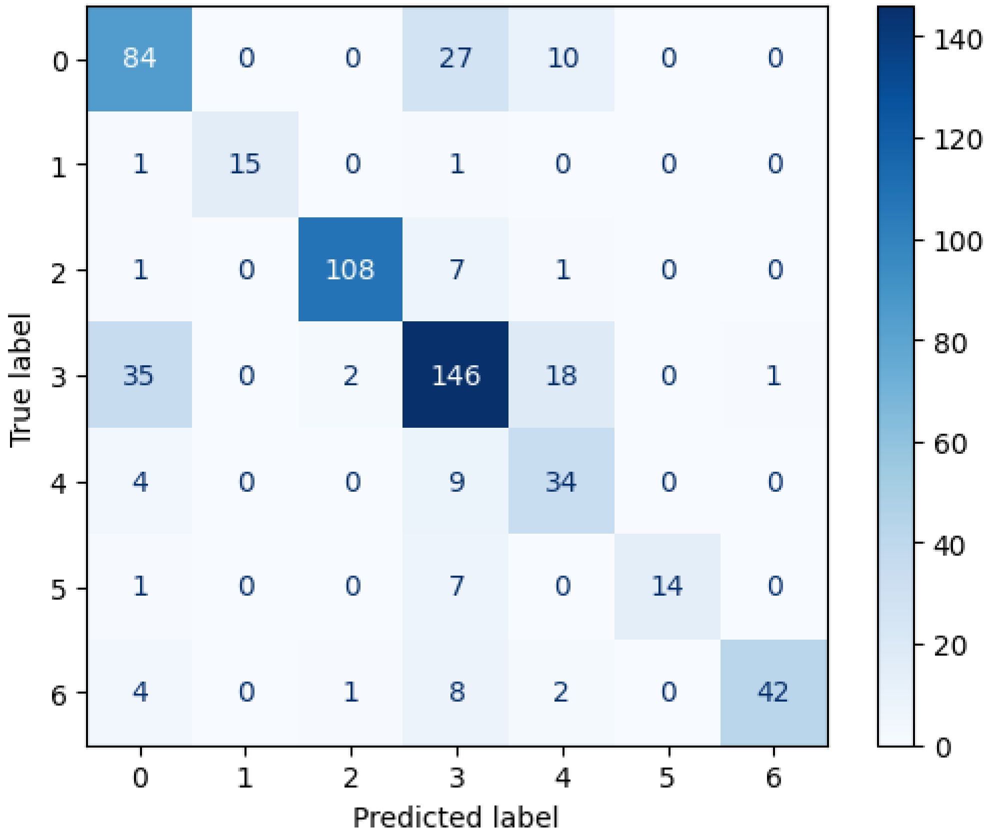
```
plt.title("XGBClassifier training & test error")
plt.ylabel("MSE")
plt.xlabel("learning_rate")
plt.legend(loc = 'center right')
plt.xticks(params, params)
plt.show()
```



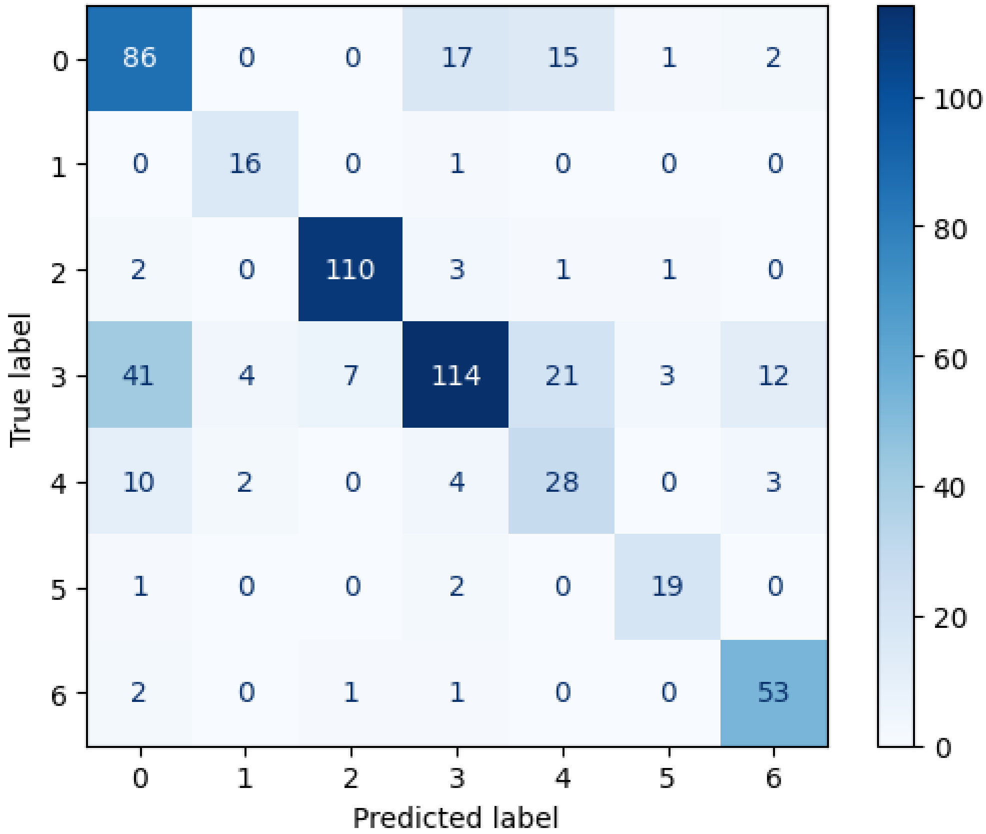
In [82]:

```
ClassifierDict = {
    "RandomForest":RandomForestClassifier(max_depth =20, random_state=42),
    "KNN": KNeighborsClassifier(n_neighbors=3),
    "LogisticRegression":LogisticRegression(random_state=42),
    "XGBoost":XGBClassifier(n_estimators=100, learning_rate= 0.3)

}
for j in ClassifierDict:
    clf=ClassifierDict.get(j)
    clf.fit(X_train_normalized,y_train)

    RSS = mean_squared_error(y_test, clf.predict(X_test_normalized))
    print(j,"with no PCA | accuary rate : ",clf.score(X_test_normalized,y_te
    print(classification_report(y_test, clf.predict(X_test_normalized), targ
    confusion_matrix = metrics.confusion_matrix(y_test, clf.predict(X_test_r
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusior
    cm_display.plot(cmap=plt.cm.Blues)

    plt.show()
```

```
RandomForest with no PCA | accuary rate :  0.7598627787307033 | RSE :  1.94
33962264150944
              precision    recall  f1-score   support

        Bump       0.65      0.69      0.67       121
   Dirtiness       1.00      0.88      0.94        17
    K_Scatch       0.97      0.92      0.95       117
Other_Faults       0.71      0.72      0.72       202
      Pastry       0.52      0.72      0.61        47
      Stains       1.00      0.64      0.78        22
    Z_Scatch       0.98      0.74      0.84        57

    accuracy                           0.76       583
   macro avg       0.83      0.76      0.79       583
weighted avg       0.78      0.76      0.77       583
```

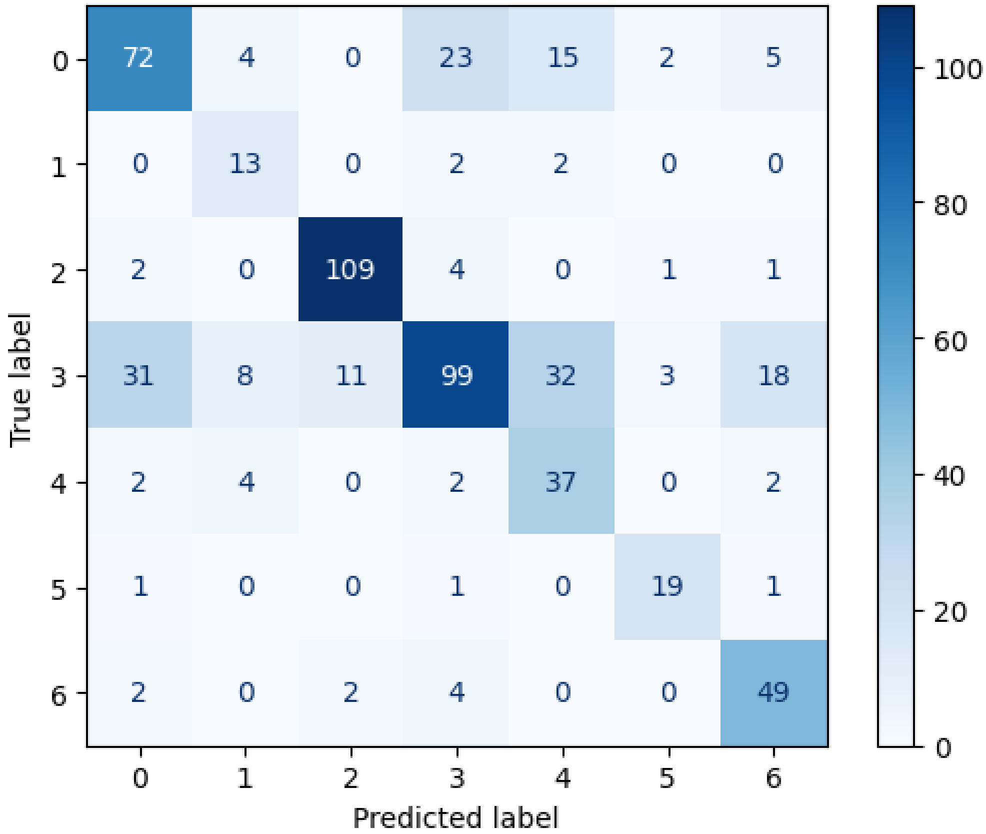



```
KNN with no PCA | accuary rate :  0.7307032590051458 | RSE :  2.35849056603
7736
              precision    recall  f1-score   support

        Bump       0.61      0.71      0.65       121
   Dirtiness       0.73      0.94      0.82        17
    K_Scatch       0.93      0.94      0.94       117
Other_Faults       0.80      0.56      0.66       202
      Pastry       0.43      0.60      0.50        47
      Stains       0.79      0.86      0.83        22
    Z_Scatch       0.76      0.93      0.83        57

    accuracy                           0.73       583
   macro avg       0.72      0.79      0.75       583
weighted avg       0.75      0.73      0.73       583
```

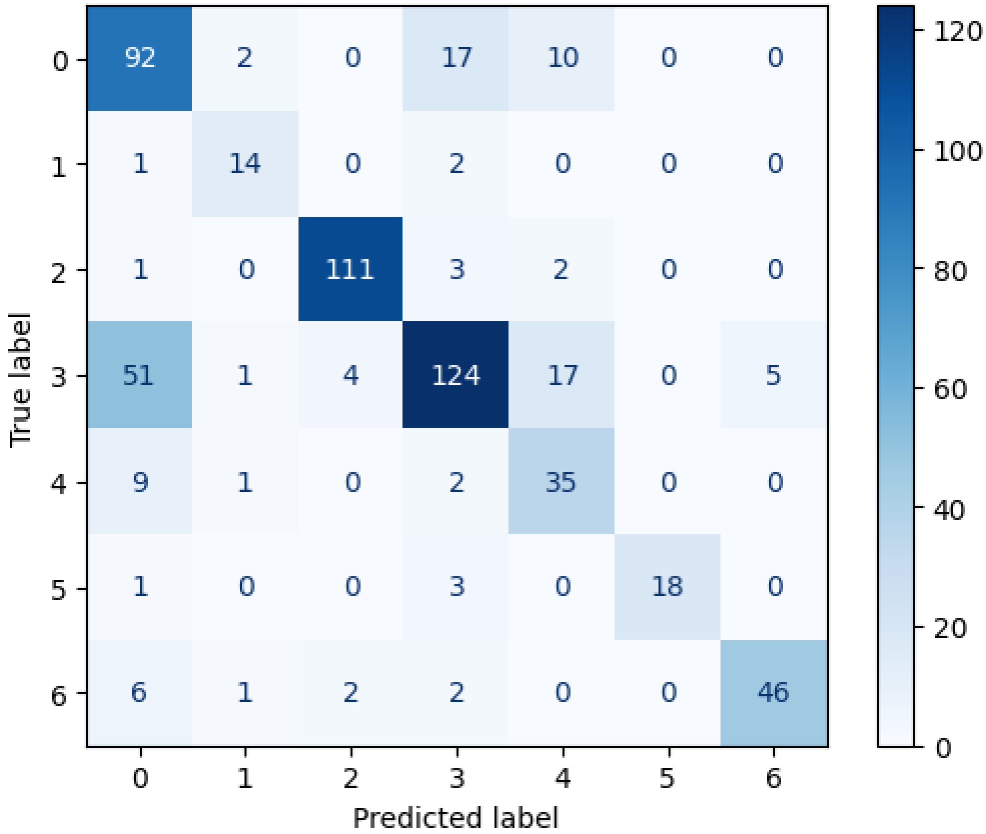



```
LogisticRegression with no PCA | accuary rate :  0.6826758147512865 | RSE : 
2.607204116638079
              precision    recall  f1-score   support

        Bump       0.65      0.60      0.62       121
   Dirtiness       0.45      0.76      0.57        17
   K_Scatch       0.89      0.93      0.91       117
Other_Faults       0.73      0.49      0.59       202
      Pastry       0.43      0.79      0.56        47
      Stains       0.76      0.86      0.81        22
    Z_Scatch       0.64      0.86      0.74        57

    accuracy                           0.68       583
   macro avg       0.65      0.76      0.68       583
weighted avg       0.71      0.68      0.68       583
```

```
/Users/sapphire/miniforge3/envs/tf/lib/python3.9/site-packages/sklearn/line
ar_model/_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (st
atus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession
  n_iter_i = _check_optimize_result(
```



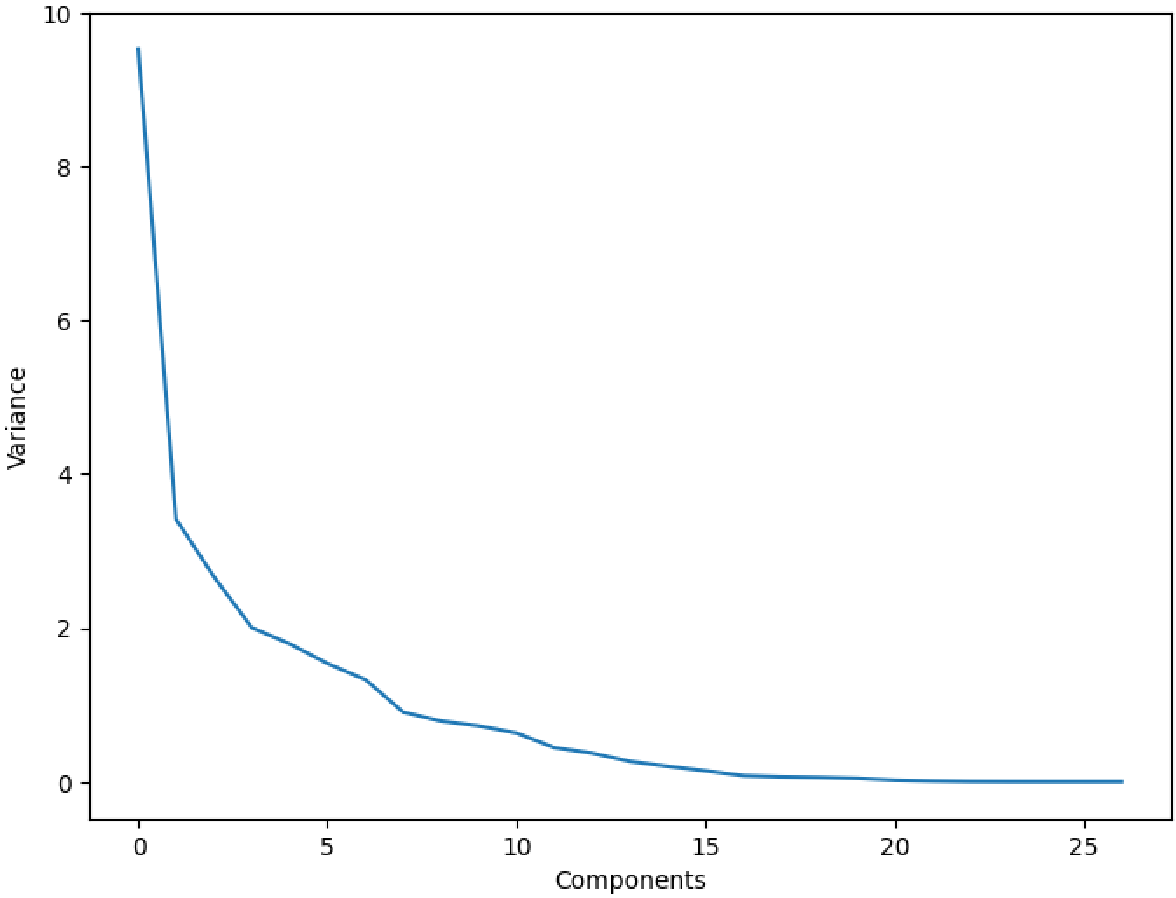

```
XGBoost with no PCA | accuary rate :  0.7547169811320755 | RSE :  2.3173241
852487134
              precision    recall  f1-score   support

        Bump       0.57      0.76      0.65       121
   Dirtiness       0.74      0.82      0.78        17
    K_Scatch       0.95      0.95      0.95       117
Other_Faults       0.81      0.61      0.70       202
      Pastry       0.55      0.74      0.63        47
      Stains       1.00      0.82      0.90        22
    Z_Scatch       0.90      0.81      0.85        57

    accuracy                           0.75       583
   macro avg       0.79      0.79      0.78       583
weighted avg       0.78      0.75      0.76       583
```



Dimensionality reduction

In [35]:
```
from sklearn.decomposition import PCA
pca=PCA(whiten=True)
pca.fit(X_train_normalized)
plt.figure(figsize=(8,6))
plt.plot(pca.explained_variance_)
plt.ylabel("Variance")
plt.xlabel("Components")
```

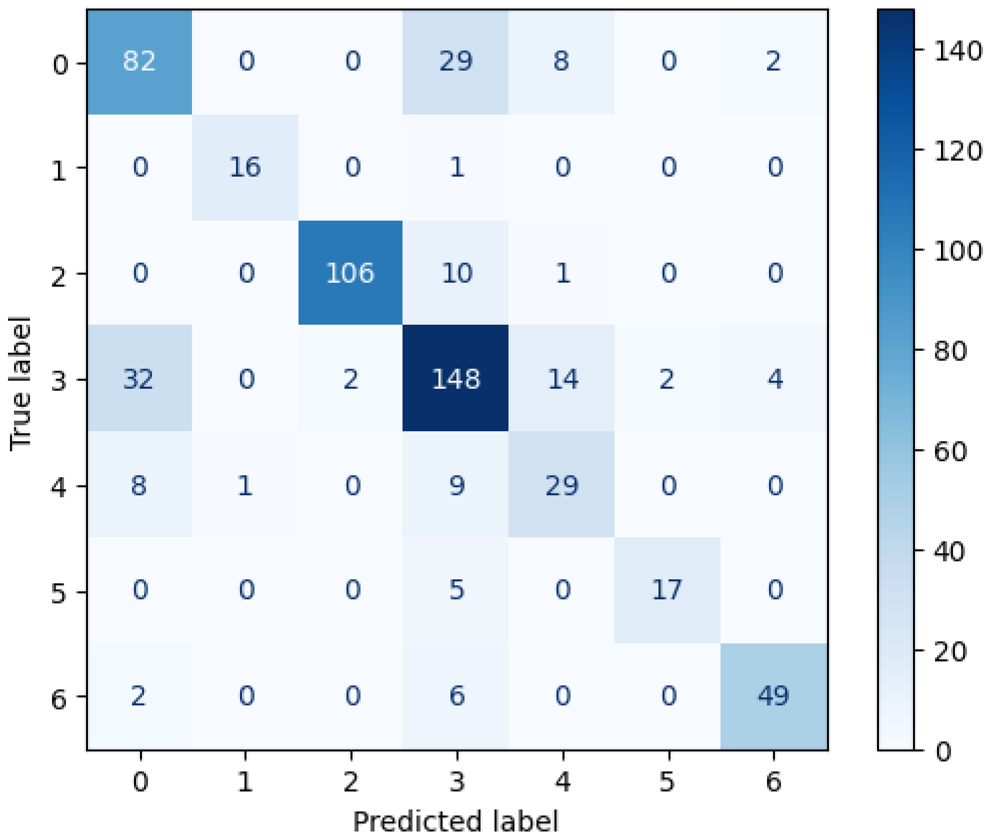
Out[35]:
```
Text(0.5, 0, 'Components')
```



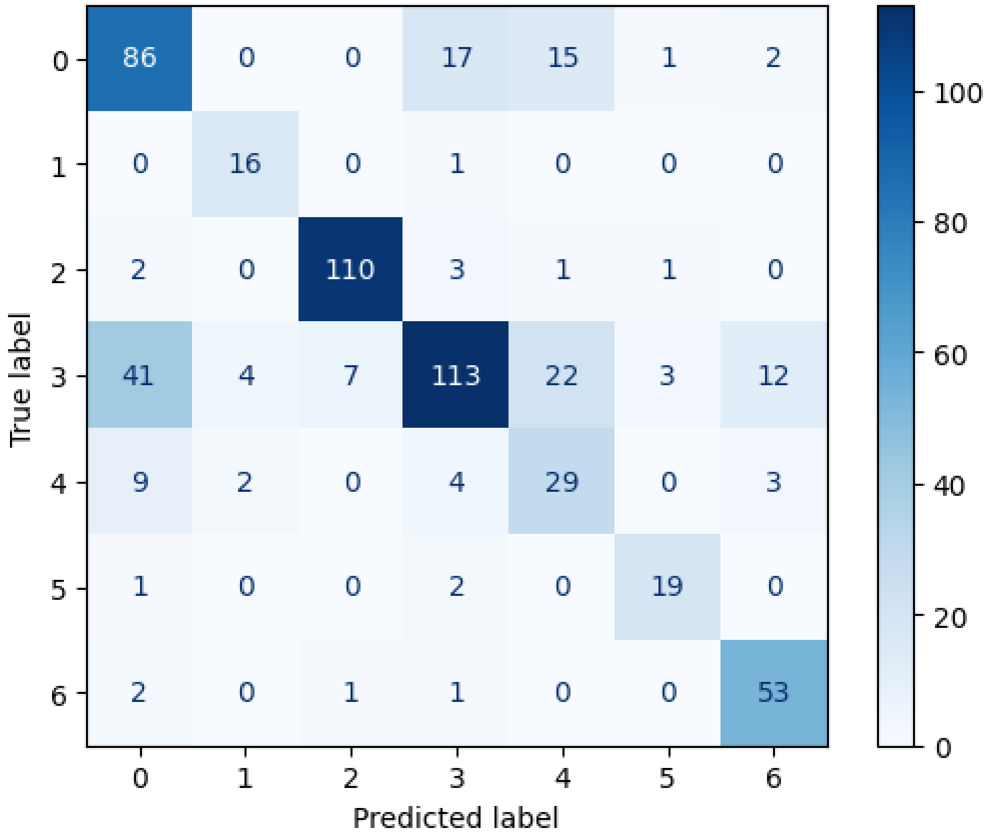
The graph above shows that the feature vector of 10-20 principal components can be represented. Let's do the PCA conversion based on 19 key components

In [99]:
```
pca=PCA(n_components=19)
X_train_pca=pca.fit_transform(X_train_normalized)
X_test_pca=pca.transform(X_test_normalized)

for j in ClassifierDict:
    clf=ClassifierDict.get(j)
    clf.fit(X_train_pca,y_train)

    RSS = mean_squared_error(y_test, clf.predict(X_test_pca))
    print(j,"with PCA | accuary rate : ",clf.score(X_test_pca,y_test), "| RS
    print(classification_report(y_test, clf.predict(X_test_pca), target_name
    confusion_matrix = metrics.confusion_matrix(y_test, clf.predict(X_test_p
    cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusior
    cm_display.plot(cmap=plt.cm.Blues)
    plt.show()
```

```
RandomForest with PCA | accuary rate :  0.7667238421955404 | RSS :  1.91938
25042881647
              precision    recall  f1-score   support

        Bump       0.66      0.68      0.67       121
   Dirtiness       0.94      0.94      0.94        17
    K_Scatch       0.98      0.91      0.94       117
Other_Faults       0.71      0.73      0.72       202
      Pastry       0.56      0.62      0.59        47
      Stains       0.89      0.77      0.83        22
    Z_Scatch       0.89      0.86      0.88        57

    accuracy                           0.77       583
   macro avg       0.81      0.79      0.79       583
weighted avg       0.77      0.77      0.77       583
```

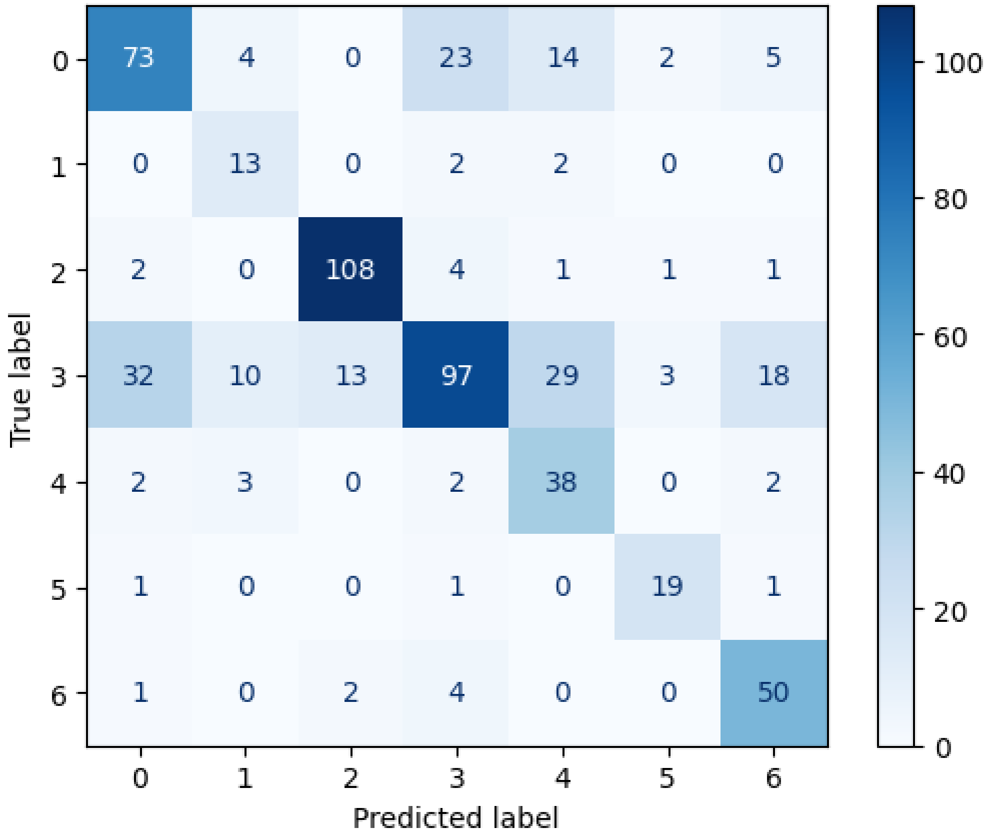



```
KNN with PCA | accuary rate :  0.7307032590051458 | RSS :  2.33276157804459
67
              precision    recall  f1-score   support

        Bump       0.61      0.71      0.66       121
   Dirtiness       0.73      0.94      0.82        17
    K_Scatch       0.93      0.94      0.94       117
Other_Faults       0.80      0.56      0.66       202
      Pastry       0.43      0.62      0.51        47
      Stains       0.79      0.86      0.83        22
    Z_Scatch       0.76      0.93      0.83        57

    accuracy                           0.73       583
   macro avg       0.72      0.79      0.75       583
weighted avg       0.75      0.73      0.73       583
```

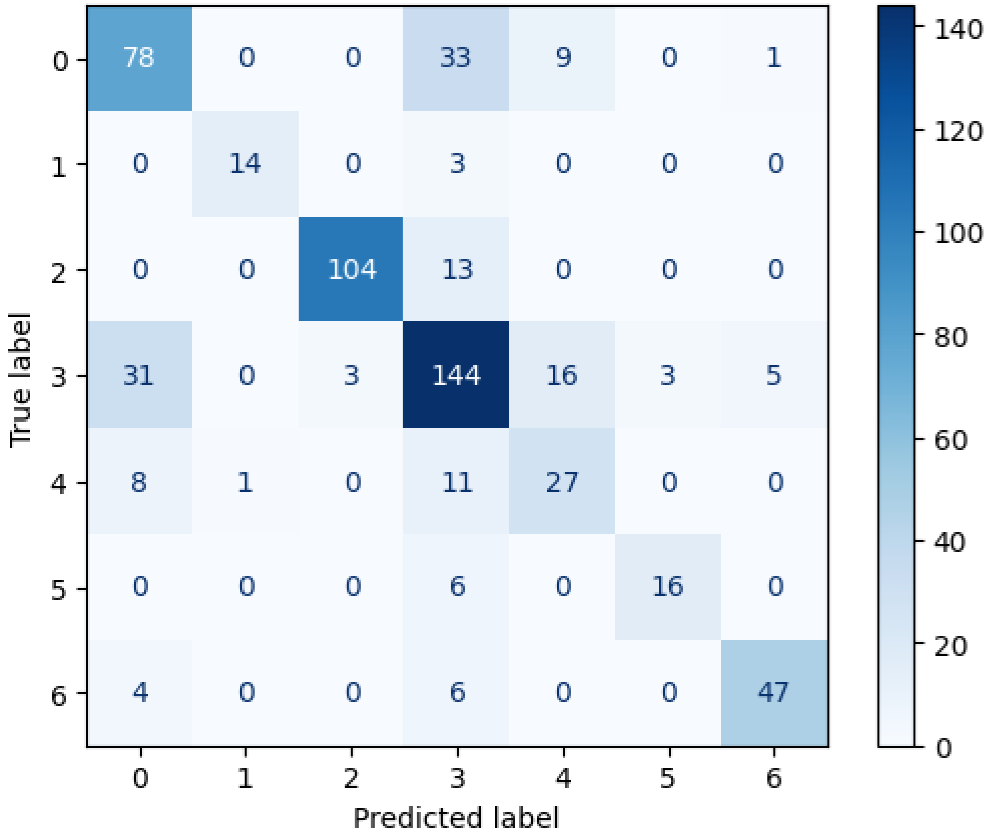



```
LogisticRegression with PCA | accuary rate :  0.6826758147512865 | RSS :
2.5368782161234993
              precision    recall  f1-score   support

        Bump       0.66      0.60      0.63       121
   Dirtiness       0.43      0.76      0.55        17
    K_Scatch       0.88      0.92      0.90       117
Other_Faults       0.73      0.48      0.58       202
      Pastry       0.45      0.81      0.58        47
      Stains       0.76      0.86      0.81        22
    Z_Scatch       0.65      0.88      0.75        57

    accuracy                           0.68       583
   macro avg       0.65      0.76      0.69       583
weighted avg       0.71      0.68      0.68       583
```

```
/Users/sapphire/miniforge3/envs/tf/lib/python3.9/site-packages/sklearn/line
ar_model/_logistic.py:444: ConvergenceWarning: lbfgs failed to converge (st
atus=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regr
ession
  n_iter_i = _check_optimize_result(
```




```
XGBoost with PCA | accuary rate :  0.7375643224699828 | RSS :  2.1046312178
38765
              precision    recall  f1-score   support

        Bump       0.64      0.64      0.64       121
   Dirtiness       0.93      0.82      0.87        17
    K_Scatch       0.97      0.89      0.93       117
Other_Faults       0.67      0.71      0.69       202
      Pastry       0.52      0.57      0.55        47
      Stains       0.84      0.73      0.78        22
    Z_Scatch       0.89      0.82      0.85        57

    accuracy                           0.74       583
   macro avg       0.78      0.74      0.76       583
weighted avg       0.75      0.74      0.74       583
```



True label
Predicted label
0
1
2
3
4
5
6
78 0 0 33 9 0 1
0 14 0 3 0 0 0
0 0 104 13 0 0 0
31 0 3 144 16 3 5
8 1 0 11 27 0 0
0 0 0 6 0 16 0
4 0 0 6 0 0 47
140
120
100
80
60
40
20
0