In [215]:

*#import require python classes and packages*  
**import** pandas **as** pd *#pandas to read and explore dataset*  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt *#use to visualize dataset vallues*  
**import** seaborn **as** sns  
**from** sklearn.preprocessing **import** MinMaxScaler  
**from** sklearn.model\_selection **import** train\_test\_split  
**from** sklearn **import** svm *#SVM class*  
**from** sklearn.tree **import** DecisionTreeClassifier  
**from** sklearn.naive\_bayes **import** GaussianNB  
**from** sklearn.ensemble **import** RandomForestClassifier  
**from** sklearn.metrics **import** precision\_score  
**from** sklearn.metrics **import** recall\_score  
**from** sklearn.metrics **import** f1\_score  
**from** sklearn.metrics **import** accuracy\_score  
**from** sklearn.metrics **import** confusion\_matrix  
**from** sklearn.metrics **import** roc\_curve  
**from** sklearn.metrics **import** roc\_auc\_score  
**from** sklearn **import** metrics   
**from** sklearn.preprocessing **import** MinMaxScaler  
**from** sklearn.preprocessing **import** LabelEncoder  
**from** sklearn.neighbors **import** KNeighborsClassifier  
**from** sklearn.ensemble **import** RandomForestRegressor  
**import** os  
**from** keras.utils.np\_utils **import** to\_categorical  
**from** keras.layers **import** MaxPooling2D  
**from** keras.layers **import** Dense, Dropout, Activation, Flatten, GlobalAveragePooling2D, BatchNormalization  
**from** keras.layers **import** Convolution2D  
**from** keras.models **import** Sequential  
**from** keras.callbacks **import** ModelCheckpoint  
**import** pickle

In [270]:

*#loading and displaying heart disease dataset*  
dataset **=** pd**.**read\_csv("Dataset/predictive\_maintenance.csv")  
dataset

Out[270]:

|  | **UDI** | **Product\_ID** | **Type** | **Air\_temperature\_[K]** | **Process\_temperature\_[K]** | **Rotational\_speed\_[rpm]** | **Torque\_[Nm]** | **Tool\_wear\_[min]** | **Target** | **Failure\_Type** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 51 | L47230 | L | 298.9 | 309.1 | 2861 | 4.6 | 143 | 1 | Power Failure |
| **1** | 70 | L47249 | L | 298.9 | 309.0 | 1410 | 65.7 | 191 | 1 | Power Failure |
| **2** | 78 | L47257 | L | 298.8 | 308.9 | 1455 | 41.3 | 208 | 1 | Tool Wear Failure |
| **3** | 161 | L47340 | L | 298.4 | 308.2 | 1282 | 60.7 | 216 | 1 | Overstrain Failure |
| **4** | 162 | L47341 | L | 298.3 | 308.1 | 1412 | 52.3 | 218 | 1 | Overstrain Failure |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| **514** | 9759 | L56938 | L | 298.6 | 309.8 | 2271 | 16.2 | 218 | 1 | Tool Wear Failure |
| **515** | 9765 | L56944 | L | 298.5 | 309.5 | 1294 | 66.7 | 12 | 1 | Power Failure |
| **516** | 9823 | L57002 | L | 298.5 | 309.4 | 1360 | 60.9 | 187 | 1 | Overstrain Failure |
| **517** | 9831 | L57010 | L | 298.3 | 309.3 | 1337 | 56.1 | 206 | 1 | Overstrain Failure |
| **518** | 9975 | L57154 | L | 298.6 | 308.2 | 1361 | 68.2 | 172 | 1 | Power Failure |

519 rows × 10 columns

In [257]:

*#describing dataset with details like count, mean, standard deviation of each dataset attributes*  
dataset**.**describe()

Out[257]:

|  | **UDI** | **Air\_temperature\_[K]** | **Process\_temperature\_[K]** | **Rotational\_speed\_[rpm]** | **Torque\_[Nm]** | **Tool\_wear\_[min]** | **Target** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **count** | 519.000000 | 519.000000 | 519.000000 | 519.000000 | 519.000000 | 519.000000 | 519.000000 |
| **mean** | 3266.921002 | 300.264740 | 309.901927 | 1507.757225 | 46.484008 | 129.485549 | 0.635838 |
| **std** | 2826.032654 | 1.983029 | 1.334179 | 322.753559 | 15.110089 | 72.415010 | 0.481659 |
| **min** | 1.000000 | 295.600000 | 306.100000 | 1181.000000 | 3.800000 | 0.000000 | 0.000000 |
| **25%** | 130.500000 | 298.800000 | 308.850000 | 1345.500000 | 36.300000 | 66.500000 | 0.000000 |
| **50%** | 3696.000000 | 299.600000 | 309.800000 | 1398.000000 | 48.200000 | 135.000000 | 1.000000 |
| **75%** | 4775.000000 | 302.100000 | 310.900000 | 1542.000000 | 57.100000 | 200.000000 | 1.000000 |
| **max** | 9975.000000 | 304.400000 | 313.700000 | 2886.000000 | 76.600000 | 253.000000 | 1.000000 |

In [219]:

*#visualizing distribution of numerical data*  
dataset**.**hist(figsize**=**(10, 8))  
plt**.**title("Representation of Dataset Attributes")  
plt**.**show()

In [220]:

*#finding and displaying count of missing or null values*  
dataset**.**isnull()**.**sum()

Out[220]:

UDI 0  
Product\_ID 0  
Type 0  
Air\_temperature\_[K] 0  
Process\_temperature\_[K] 0  
Rotational\_speed\_[rpm] 0  
Torque\_[Nm] 0  
Tool\_wear\_[min] 0  
Target 0  
Failure\_Type 0  
dtype: int64

In [221]:

*#finding & plotting graph of failure machine parts which required maintenance*  
*#visualizing class labels count found in dataset*  
labels, count **=** np**.**unique(dataset['Failure\_Type']**.**ravel(), return\_counts **=** **True**)  
height **=** count  
bars **=** labels  
y\_pos **=** np**.**arange(len(bars))  
plt**.**figure(figsize **=** (4, 3))   
plt**.**bar(y\_pos, height)  
plt**.**xticks(y\_pos, bars)  
plt**.**xlabel("Dataset Class Label Graph")  
plt**.**ylabel("Count")  
plt**.**xticks(rotation**=**90)  
plt**.**show()

In [222]:

*#visualizing product quality as number of Low, high and medium quality*  
*#describe and plotting graph of various Product Current Quality % found in dataset*   
dataset**.**groupby("Type")**.**size()**.**plot**.**pie(autopct**=**'%.0f%%', figsize**=**(4, 4))  
plt**.**title("Product Type % graph")  
plt**.**xlabel("Product Condition Type")  
plt**.**ylabel("Condition %")  
plt**.**show()

In [223]:

*#visualizing tool life with different failure conditions*  
data **=** dataset**.**groupby(['Failure\_Type', 'Type'])['Tool\_wear\_[min]']**.**sum()**.**sort\_values(ascending**=False**)**.**reset\_index()  
sns**.**catplot(x**=**"Type", y**=**"Tool\_wear\_[min]", hue**=**'Failure\_Type', data**=**data, kind**=**'point')  
plt**.**title("Product Life Time Based on Failure Type and Product Condition Type")  
plt**.**show()

In [224]:

dataset**.**groupby('Type')['Process\_temperature\_[K]']**.**plot(legend**=True**, figsize**=**(6,3))  
plt**.**title("Process Temperature Available in All Products Quality")  
plt**.**show()

In [227]:

data **=** dataset[['Failure\_Type', 'Air\_temperature\_[K]', 'Rotational\_speed\_[rpm]', 'Torque\_[Nm]']]  
plt**.**figure(figsize**=**(12,4))  
sns**.**boxplot(data**=**data, x**=**'Failure\_Type', y**=**'Air\_temperature\_[K]', palette**=**'rainbow')  
plt**.**title("Machine Air Temperature Available in Normal & Failure Conditions")  
plt**.**show()

In [228]:

plt**.**figure(figsize**=**(12,4))  
sns**.**violinplot(data**=**data, x**=**'Failure\_Type', y**=**'Rotational\_speed\_[rpm]', palette**=**'rainbow')  
plt**.**title("Machine Rotational speed Available in Normal & Failure Conditions")  
plt**.**show()

In [271]:

*#using label encoder converting non-numeric values to numeric values*  
encoder1 **=** LabelEncoder()  
encoder2 **=** LabelEncoder()  
encoder3 **=** LabelEncoder()  
dataset['Product\_ID'] **=** pd**.**Series(encoder1**.**fit\_transform(dataset['Product\_ID']**.**astype(str)))*#encode all str columns to numeric*  
dataset['Type'] **=** pd**.**Series(encoder2**.**fit\_transform(dataset['Type']**.**astype(str)))*#encode all str columns to numeric*  
dataset['Failure\_Type'] **=** pd**.**Series(encoder3**.**fit\_transform(dataset['Failure\_Type']**.**astype(str)))*#encode all str columns to numeric*  
*#dataset pre-processing like removing irrelevant features and selecting relevant features from the dataset*  
dataset**.**drop(['UDI', 'Target'], axis **=** 1,inplace**=True**)  
print("Dataset After Cleaning & Processing")  
dataset

Dataset After Cleaning & Processing

Out[271]:

|  | **Product\_ID** | **Type** | **Air\_temperature\_[K]** | **Process\_temperature\_[K]** | **Rotational\_speed\_[rpm]** | **Torque\_[Nm]** | **Tool\_wear\_[min]** | **Failure\_Type** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **0** | 65 | 1 | 298.9 | 309.1 | 2861 | 4.6 | 143 | 3 |
| **1** | 78 | 1 | 298.9 | 309.0 | 1410 | 65.7 | 191 | 3 |
| **2** | 85 | 1 | 298.8 | 308.9 | 1455 | 41.3 | 208 | 5 |
| **3** | 131 | 1 | 298.4 | 308.2 | 1282 | 60.7 | 216 | 2 |
| **4** | 132 | 1 | 298.3 | 308.1 | 1412 | 52.3 | 218 | 2 |
| **...** | ... | ... | ... | ... | ... | ... | ... | ... |
| **514** | 366 | 1 | 298.6 | 309.8 | 2271 | 16.2 | 218 | 5 |
| **515** | 367 | 1 | 298.5 | 309.5 | 1294 | 66.7 | 12 | 3 |
| **516** | 368 | 1 | 298.5 | 309.4 | 1360 | 60.9 | 187 | 2 |
| **517** | 369 | 1 | 298.3 | 309.3 | 1337 | 56.1 | 206 | 2 |
| **518** | 370 | 1 | 298.6 | 308.2 | 1361 | 68.2 | 172 | 3 |

519 rows × 8 columns

In [272]:

*#dataset shuffling and normalization*  
rul **=** dataset['Tool\_wear\_[min]']**.**ravel()*#represents life of machine (rul = remaining useful life)*  
Y **=** dataset['Failure\_Type']**.**ravel()*#represents machine failure or normal*  
data **=** dataset**.**values  
X **=** data[:,0:dataset**.**shape[1]**-**1]  
indices **=** np**.**arange(X**.**shape[0])  
np**.**random**.**shuffle(indices)*#shuffling dataset values*  
X **=** X[indices]  
Y **=** Y[indices]  
rul **=** rul[indices]  
*#normalizing dataset values*  
scaler **=** MinMaxScaler(feature\_range **=** (0, 1))  
X **=** scaler**.**fit\_transform(X)*#normalize train features*  
print("Normalize Training Features")  
print(X)

Normalize Training Features  
[[0.65922921 0.5 0.57954545 ... 0.13782991 0.7967033 0.82608696]  
 [0.76470588 1. 0.34090909 ... 0.07624633 0.58791209 0.17391304]  
 [0.03245436 0. 0.34090909 ... 0.13255132 0.64148352 0.64822134]  
 ...  
 [0.64908722 0.5 0.63636364 ... 0.16422287 0.67445055 0.82213439]  
 [0.31643002 0.5 0.38636364 ... 0.11143695 0.68681319 0.90118577]  
 [0.78904665 1. 0.375 ... 0.13548387 0.55357143 0.55335968]]

In [273]:

*#split dataset into train and test*  
X\_train, X\_test, y\_train, y\_test **=** train\_test\_split(X, Y, test\_size **=** 0.2)  
print("Total records found in dataset = "**+**str(X**.**shape[0]))  
print("Total features found in dataset= "**+**str(X**.**shape[1]))  
print("80% dataset for training : "**+**str(X\_train**.**shape[0]))  
print("20% dataset for testing : "**+**str(X\_test**.**shape[0]))

Total records found in dataset = 519  
Total features found in dataset= 7  
80% dataset for training : 415  
20% dataset for testing : 104

In [274]:

*#define global variables to save accuracy and other metrics*  
accuracy **=** []  
precision **=** []  
recall **=** []  
fscore **=** []

In [275]:

*#function to calculate all metrics*  
**def** calculateMetrics(algorithm, testY, predict):  
 p **=** precision\_score(testY, predict,average**=**'macro') **\*** 100  
 r **=** recall\_score(testY, predict,average**=**'macro') **\*** 100  
 f **=** f1\_score(testY, predict,average**=**'macro') **\*** 100  
 a **=** accuracy\_score(testY,predict)**\***100  
 accuracy**.**append(a)  
 precision**.**append(p)  
 recall**.**append(r)  
 fscore**.**append(f)  
 print(algorithm**+**" Accuracy : "**+**str(a))  
 print(algorithm**+**" Precision : "**+**str(p))  
 print(algorithm**+**" Recall : "**+**str(r))  
 print(algorithm**+**" FSCORE : "**+**str(f))  
 conf\_matrix **=** confusion\_matrix(testY, predict)  
 fig, axs **=** plt**.**subplots(1,2,figsize**=**(10, 3))  
 ax **=** sns**.**heatmap(conf\_matrix, xticklabels **=** labels, yticklabels **=** labels, annot **=** **True**, cmap**=**"viridis" ,fmt **=**"g", ax**=**axs[0]);  
 ax**.**set\_ylim([0,len(labels)])  
 axs[0]**.**set\_title(algorithm**+**" Confusion matrix")   
  
 random\_probs **=** [0 **for** i **in** range(len(testY))]  
 p\_fpr, p\_tpr, \_ **=** roc\_curve(testY, random\_probs, pos\_label**=**1)  
 plt**.**plot(p\_fpr, p\_tpr, linestyle**=**'--', color**=**'orange',label**=**"True classes")  
 ns\_fpr, ns\_tpr, \_ **=** roc\_curve(testY, predict, pos\_label**=**1)  
 axs[1]**.**plot(ns\_tpr, ns\_fpr, linestyle**=**'--', label**=**'Predicted Classes')  
 axs[1]**.**set\_title(algorithm**+**" ROC AUC Curve")  
 axs[1]**.**set\_xlabel('False Positive Rate')  
 axs[1]**.**set\_ylabel('True Positive rate')  
 plt**.**show()

In [276]:

*#training and evaluating performance of SVM algorithm*  
svm\_cls **=** svm**.**SVC(C**=**50)  
svm\_cls**.**fit(X\_train, y\_train)*#train algorithm using training features and target value*  
predict **=** svm\_cls**.**predict(X\_test) *#perform prediction on test data*  
*#call this function with true and predicted values to calculate accuracy and other metrics*  
calculateMetrics("SVM Algorithm", y\_test, predict)

SVM Algorithm Accuracy : 92.3076923076923  
SVM Algorithm Precision : 92.63018027723909  
SVM Algorithm Recall : 89.33277370777371  
SVM Algorithm FSCORE : 90.44081630294446

In [277]:

*#training and evaluating performance of decision tree algorithm*  
dt\_cls **=** DecisionTreeClassifier()  
dt\_cls**.**fit(X\_train, y\_train)*#train algorithm using training features and target value*  
predict **=**dt\_cls**.**predict(X\_test)*#perform prediction on test data*  
*#call this function with true and predicted values to calculate accuracy and other metrics*  
calculateMetrics("Decision Tree Algorithm", y\_test, predict)

Decision Tree Algorithm Accuracy : 88.46153846153845  
Decision Tree Algorithm Precision : 85.1615704556881  
Decision Tree Algorithm Recall : 85.75371387871388  
Decision Tree Algorithm FSCORE : 85.42395588907218

In [278]:

*#training and evaluating performance of RandomForestClassifier algorithm*  
regressor **=** RandomForestRegressor()  
regressor**.**fit(X, rul)  
rf\_cls **=** RandomForestClassifier(max\_depth**=**10)  
rf\_cls**.**fit(X\_train, y\_train)*#train algorithm using training features and target value*  
predict **=** rf\_cls**.**predict(X\_test)*#perform prediction on test data*  
*#call this function with true and predicted values to calculate accuracy and other metrics*  
calculateMetrics("Random Forest", y\_test, predict)

Random Forest Accuracy : 92.3076923076923  
Random Forest Precision : 90.41867954911433  
Random Forest Recall : 89.00335775335776  
Random Forest FSCORE : 89.60848682907505

In [279]:

*#training and evaluating performance of RandomForestClassifier algorithm*  
knn\_cls **=** KNeighborsClassifier(n\_neighbors**=**2)  
knn\_cls**.**fit(X\_train, y\_train)*#train algorithm using training features and target value*  
predict **=** knn\_cls**.**predict(X\_test)*#perform prediction on test data*  
*#call this function with true and predicted values to calculate accuracy and other metrics*  
calculateMetrics("KNN", y\_test, predict)

KNN Accuracy : 82.6923076923077  
KNN Precision : 86.14996114996116  
KNN Recall : 74.15165852665852  
KNN FSCORE : 76.6430971044633

In [280]:

*#training CNN deep learning algorithm to predict factory maintenaance*  
*#converting dataset shape for CNN comptaible format as 4 dimension array*  
X\_train1 **=** np**.**reshape(X\_train, (X\_train**.**shape[0], X\_train**.**shape[1], 1, 1))  
X\_test1 **=** np**.**reshape(X\_test, (X\_test**.**shape[0], X\_test**.**shape[1], 1, 1))  
y\_train1 **=** to\_categorical(y\_train)  
y\_test1 **=** to\_categorical(y\_test)  
*#creating deep learning cnn model object*  
cnn\_model **=** Sequential()  
*#defining CNN layer wwith 32 neurons of size 1 X 1 to filter dataset features 32 times*  
cnn\_model**.**add(Convolution2D(32, (1 , 1), input\_shape **=** (X\_train1**.**shape[1], X\_train1**.**shape[2], X\_train1**.**shape[3]), activation **=** 'relu'))  
*#defining maxpool layet to collect relevant filtered features from previous CNN layer*  
cnn\_model**.**add(MaxPooling2D(pool\_size **=** (1, 1)))  
*#creating another CNN layer with 16 neurons to optimzed features 16 times*  
cnn\_model**.**add(Convolution2D(16, (1, 1), activation **=** 'relu'))  
*#max layet to collect relevant features*  
cnn\_model**.**add(MaxPooling2D(pool\_size **=** (1, 1)))  
*#convert multidimension features to single flatten size*  
cnn\_model**.**add(Flatten())  
*#define output prediction layer*  
cnn\_model**.**add(Dense(units **=** 256, activation **=** 'relu'))  
cnn\_model**.**add(Dense(units **=** y\_train1**.**shape[1], activation **=** 'softmax'))  
*#compile, train and load CNN model*  
cnn\_model**.**compile(optimizer **=** 'adam', loss **=** 'categorical\_crossentropy', metrics **=** ['accuracy'])  
**if** os**.**path**.**exists("model/cnn\_weights.hdf5") **==** **False**:  
 model\_check\_point **=** ModelCheckpoint(filepath**=**'model/cnn\_weights.hdf5', verbose **=** 1, save\_best\_only **=** **True**)  
 hist **=** cnn\_model**.**fit(X\_train1, y\_train1, batch\_size **=** 4, epochs **=** 50, validation\_data**=**(X\_test1, y\_test1), callbacks**=**[model\_check\_point], verbose**=**1)  
 f **=** open('model/cnn\_history.pckl', 'wb')  
 pickle**.**dump(hist**.**history, f)  
 f**.**close()   
**else**:  
 cnn\_model**.**load\_weights("model/cnn\_weights.hdf5")  
*#perform prediction on test data*   
predict **=** cnn\_model**.**predict(X\_test1)  
predict **=** np**.**argmax(predict, axis**=**1)  
y\_test1 **=** np**.**argmax(y\_test1, axis**=**1)  
*#call this function to calculate accuracy and other metrics*  
calculateMetrics("CNN", y\_test1, predict)

CNN Accuracy : 97.11538461538461  
CNN Precision : 96.75716440422323  
CNN Recall : 95.82443019943021  
CNN FSCORE : 96.22240504593447

In [281]:

*#comparison graph between all algorithms*  
df **=** pd**.**DataFrame([['SVM','Accuracy',accuracy[0]],['SVM','Precision',precision[0]],['SVM','Recall',recall[0]],['SVM','FSCORE',fscore[0]],  
 ['Decision Tree','Accuracy',accuracy[1]],['Decision Tree','Precision',precision[1]],['Decision Tree','Recall',recall[1]],['Decision Tree','FSCORE',fscore[1]],  
 ['Random Forest','Accuracy',accuracy[2]],['Random Forest','Precision',precision[2]],['Random Forest','Recall',recall[2]],['Random Forest','FSCORE',fscore[2]],  
 ['KNN','Accuracy',accuracy[3]],['KNN','Precision',precision[3]],['KNN','Recall',recall[3]],['KNN','FSCORE',fscore[3]],  
 ['Deep Learning CNN','Accuracy',accuracy[4]],['Deep Learning CNN','Precision',precision[4]],['Deep Learning CNN','Recall',recall[4]],['Deep Learning CNN','FSCORE',fscore[4]],  
 ],columns**=**['Parameters','Algorithms','Value'])  
df**.**pivot("Parameters", "Algorithms", "Value")**.**plot(kind**=**'bar', figsize**=**(6, 3))  
plt**.**title("All Algorithms Performance Graph")  
plt**.**show()

In [282]:

*#display all algorithm performnace*  
algorithms **=** ['SVM', 'Decision Tree', 'Random Forest', 'KNN', 'Deep Learning CNN']  
data **=** []  
**for** i **in** range(len(accuracy)):  
 data**.**append([algorithms[i], accuracy[i], precision[i], recall[i], fscore[i]])  
data **=** pd**.**DataFrame(data, columns**=**['Algorithm Name', 'Accuracy', 'Precision', 'Recall', 'FSCORE'])  
data

Out[282]:

|  | **Algorithm Name** | **Accuracy** | **Precision** | **Recall** | **FSCORE** |
| --- | --- | --- | --- | --- | --- |
| **0** | SVM | 92.307692 | 92.630180 | 89.332774 | 90.440816 |
| **1** | Decision Tree | 88.461538 | 85.161570 | 85.753714 | 85.423956 |
| **2** | Random Forest | 92.307692 | 90.418680 | 89.003358 | 89.608487 |
| **3** | KNN | 82.692308 | 86.149961 | 74.151659 | 76.643097 |
| **4** | Deep Learning CNN | 97.115385 | 96.757164 | 95.824430 | 96.222405 |

In [283]:

test\_data **=** pd**.**read\_csv("Dataset/testData.csv")  
temp **=** test\_data**.**values  
*#using label encoder converting non-numeric values to numeric values*  
test\_data['Product\_ID'] **=** pd**.**Series(encoder1**.**transform(test\_data['Product\_ID']**.**astype(str)))*#encode all str columns to numeric*  
test\_data['Type'] **=** pd**.**Series(encoder2**.**transform(test\_data['Type']**.**astype(str)))*#encode all str columns to numeric*  
*#dataset pre-processing like removing irrelevant features and selecting relevant features from the dataset*  
test\_data**.**drop(['UDI'], axis **=** 1,inplace**=True**)  
test\_data **=** test\_data**.**values  
test\_data **=** scaler**.**transform(test\_data)  
*#life prediction before maintenance*  
life **=** regressor**.**predict(test\_data)  
test\_data **=** np**.**reshape(test\_data, (test\_data**.**shape[0], test\_data**.**shape[1], 1, 1))  
*#failure prediction*  
predict **=** cnn\_model**.**predict(test\_data)  
**for** i **in** range(len(predict)):  
 pred **=** np**.**argmax(predict[i])  
 print("Test Data : "**+**str(temp[i])**+**" ====> Predicted Failure : "**+**labels[pred])  
 print("Available Life Maintenance = "**+**str(100 **-** (life[i]**/**10))**+**"\n")

Test Data : [195 'M15054' 'M' 298.2 308.5 2678 10.7 86] ====> Predicted Failure : Power Failure  
Available Life Maintenance = 91.441  
  
Test Data : [208 'M15067' 'M' 298.4 308.7 1421 60.7 119] ====> Predicted Failure : Power Failure  
Available Life Maintenance = 88.123  
  
Test Data : [243 'L47422' 'L' 298.0 308.2 1348 58.8 202] ====> Predicted Failure : Overstrain Failure  
Available Life Maintenance = 79.79599999999999  
  
Test Data : [249 'L47428' 'L' 298.0 308.3 1362 56.8 216] ====> Predicted Failure : Overstrain Failure  
Available Life Maintenance = 78.396  
  
Test Data : [17 'M14876' 'M' 298.6 309.2 1311 46.6 44] ====> Predicted Failure : No Failure  
Available Life Maintenance = 95.587  
  
Test Data : [18 'M14877' 'M' 298.7 309.2 1410 45.6 47] ====> Predicted Failure : No Failure  
Available Life Maintenance = 95.311  
  
Test Data : [1510 'L48689' 'L' 298.0 308.5 1429 37.7 220] ====> Predicted Failure : Tool Wear Failure  
Available Life Maintenance = 78.00800000000001  
  
Test Data : [1683 'H31096' 'H' 297.9 307.4 1604 36.1 225] ====> Predicted Failure : Tool Wear Failure  
Available Life Maintenance = 77.474  
  
Test Data : [2073 'L49252' 'L' 299.6 309.5 1570 35.5 189] ====> Predicted Failure : Random Failures  
Available Life Maintenance = 81.1  
  
Test Data : [2073 'L49252' 'L' 299.6 309.5 1570 35.5 189] ====> Predicted Failure : Random Failures  
Available Life Maintenance = 81.1

In [ ]: