In [1]:
```
import numpy as np
import pandas as pd
import sklearn
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_squared_error, r2_score
import os
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import MinMaxScaler
import random
import warnings
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay, accuracy_scor
from sklearn import metrics
np.random.seed(34)
warnings.filterwarnings('ignore')
```

In [2]:
```
index_names = ['ENGINE', 'CYCLE']
setting_names = ['SET1', 'SET2', 'SET3']
sensor_names=[ "INLET_TEMP",
"LPC_OUT_TEMP",
"HPC_OUT_TEMP",
"LPT_OUT_TEMP",
"FAN_IN_PR",
"BYPASS-PR",
"HPC_OUT_PR",
"FAN_RPM",
"CORE_RPM",
"ENGINE_PR",
"HPC_OUT",
"FUEL_RATIO",
"FAN_RPM_CORR",
"CORE_RPM_CORR",
"BYPASS RATIO",
"FUEL_RATIO_BURNER",
"ENTHALPY_BLEED",
"FAN_SPEED_REQ",
"CONV_FAN_SPEED",
"HP_AIRFLOW",
"LPC_AIRFLOW" ]
col_names = index_names + setting_names + sensor_names
```

In [3]:
```
data = pd.read_csv('CMaps\\train_FD001.txt',sep= " ",header=None,index_col=False,na
```

In [4]:
```
pd.set_option('display.max_columns', None)
df = data.copy()
df
```

Out[4]:

| | ENGINE | CYCLE | SET1 | SET2 | SET3 | INLET_TEMP | LPC_OUT_TEMP | HPC_OUT_TE |
|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | -0.0007 | -0.0004 | 100.0 | 518.67 | 641.82 | 158 |
| **1** | 1 | 2 | 0.0019 | -0.0003 | 100.0 | 518.67 | 642.15 | 159 |
| **2** | 1 | 3 | -0.0043 | 0.0003 | 100.0 | 518.67 | 642.35 | 158 |
| **3** | 1 | 4 | 0.0007 | 0.0000 | 100.0 | 518.67 | 642.35 | 158 |
| **4** | 1 | 5 | -0.0019 | -0.0002 | 100.0 | 518.67 | 642.37 | 158 |
| **...** | ... | ... | ... | ... | ... | ... | ... | |
| **20626** | 100 | 196 | -0.0004 | -0.0003 | 100.0 | 518.67 | 643.49 | 159 |
| **20627** | 100 | 197 | -0.0016 | -0.0005 | 100.0 | 518.67 | 643.54 | 160 |
| **20628** | 100 | 198 | 0.0004 | 0.0000 | 100.0 | 518.67 | 643.42 | 160 |
| **20629** | 100 | 199 | -0.0011 | 0.0003 | 100.0 | 518.67 | 643.23 | 160 |
| **20630** | 100 | 200 | -0.0032 | -0.0005 | 100.0 | 518.67 | 643.85 | 160 |

20631 rows × 26 columns

In [5]:

```
df.shape
```

Out[5]:

```
(20631, 26)
```

In [6]:

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20631 entries, 0 to 20630
Data columns (total 26 columns):
 #   Column              Non-Null Count  Dtype  
---  ------              --------------  -----  
 0   ENGINE              20631 non-null  int64  
 1   CYCLE               20631 non-null  int64  
 2   SET1                20631 non-null  float64
 3   SET2                20631 non-null  float64
 4   SET3                20631 non-null  float64
 5   INLET_TEMP          20631 non-null  float64
 6   LPC_OUT_TEMP        20631 non-null  float64
 7   HPC_OUT_TEMP        20631 non-null  float64
 8   LPT_OUT_TEMP        20631 non-null  float64
 9   FAN_IN_PR           20631 non-null  float64
 10  BYPASS-PR           20631 non-null  float64
 11  HPC_OUT_PR          20631 non-null  float64
 12  FAN_RPM             20631 non-null  float64
 13  CORE_RPM            20631 non-null  float64
 14  ENGINE_PR           20631 non-null  float64
 15  HPC_OUT             20631 non-null  float64
 16  FUEL_RATIO          20631 non-null  float64
 17  FAN_RPM_CORR        20631 non-null  float64
 18  CORE_RPM_CORR       20631 non-null  float64
 19  BYPASS RATIO        20631 non-null  float64
 20  FUEL_RATIO_BURNER   20631 non-null  float64
 21  ENTHALPY_BLEED      20631 non-null  int64  
 22  FAN_SPEED_REQ       20631 non-null  int64  
 23  CONV_FAN_SPEED      20631 non-null  float64
 24  HP_AIRFLOW          20631 non-null  float64
 25  LPC_AIRFLOW         20631 non-null  float64
dtypes: float64(22), int64(4)
memory usage: 4.1 MB
```

In [7]:
```
df.loc[:,['ENGINE','CYCLE']].describe()
```

Out[7]:

| | ENGINE | CYCLE |
|---|---|---|
| **count** | 20631.000000 | 20631.000000 |
| **mean** | 51.506568 | 108.807862 |
| **std** | 29.227633 | 68.880990 |
| **min** | 1.000000 | 1.000000 |
| **25%** | 26.000000 | 52.000000 |
| **50%** | 52.000000 | 104.000000 |
| **75%** | 77.000000 | 156.000000 |
| **max** | 100.000000 | 362.000000 |

In [8]:
```
df.loc[:,['SET1','SET2','SET3']].describe()
```

Out[8]:

| | SET1 | SET2 | SET3 |
|---|---|---|---|
| **count** | 20631.000000 | 20631.000000 | 20631.0 |
| **mean** | -0.000009 | 0.000002 | 100.0 |
| **std** | 0.002187 | 0.000293 | 0.0 |
| **min** | -0.008700 | -0.000600 | 100.0 |
| **25%** | -0.001500 | -0.000200 | 100.0 |
| **50%** | 0.000000 | 0.000000 | 100.0 |
| **75%** | 0.001500 | 0.000300 | 100.0 |
| **max** | 0.008700 | 0.000600 | 100.0 |

In [9]:
```
df.loc[:,'INLET_TEMP':'LPC_AIRFLOW'].describe()
```

Out[9]:

| | INLET_TEMP | LPC_OUT_TEMP | HPC_OUT_TEMP | LPT_OUT_TEMP | FAN_IN_PR | BY |
|---|---|---|---|---|---|---|
| **count** | 2.063100e+04 | 20631.000000 | 20631.000000 | 20631.000000 | 2.063100e+04 | 2063 |
| **mean** | 5.186700e+02 | 642.680934 | 1590.523119 | 1408.933782 | 1.462000e+01 | 2 |
| **std** | 6.537152e-11 | 0.500053 | 6.131150 | 9.000605 | 3.394700e-12 | |
| **min** | 5.186700e+02 | 641.210000 | 1571.040000 | 1382.250000 | 1.462000e+01 | 2 |
| **25%** | 5.186700e+02 | 642.325000 | 1586.260000 | 1402.360000 | 1.462000e+01 | 2 |
| **50%** | 5.186700e+02 | 642.640000 | 1590.100000 | 1408.040000 | 1.462000e+01 | 2 |
| **75%** | 5.186700e+02 | 643.000000 | 1594.380000 | 1414.555000 | 1.462000e+01 | 2 |
| **max** | 5.186700e+02 | 644.530000 | 1616.910000 | 1441.490000 | 1.462000e+01 | 2 |

In [10]:
```
unwanted=[]
for i in df.select_dtypes(include=np.number):
    if df[i].nunique()==1:
        unwanted.append(i)

unwanted
```

Out[10]:
```
['SET3',
 'INLET_TEMP',
 'FAN_IN_PR',
 'ENGINE_PR',
 'FUEL_RATIO_BURNER',
 'FAN_SPEED_REQ',
 'CONV_FAN_SPEED']
```

In [11]:
```
df.drop(columns=unwanted, inplace=True)
print(df.shape)
```

```
df
```

(20631, 19)

Out[11]:

| | ENGINE | CYCLE | SET1 | SET2 | LPC_OUT_TEMP | HPC_OUT_TEMP | LPT_OUT_TEMP |
|---|---|---|---|---|---|---|---|
| **0** | 1 | 1 | -0.0007 | -0.0004 | 641.82 | 1589.70 | 1400.60 |
| **1** | 1 | 2 | 0.0019 | -0.0003 | 642.15 | 1591.82 | 1403.14 |
| **2** | 1 | 3 | -0.0043 | 0.0003 | 642.35 | 1587.99 | 1404.20 |
| **3** | 1 | 4 | 0.0007 | 0.0000 | 642.35 | 1582.79 | 1401.87 |
| **4** | 1 | 5 | -0.0019 | -0.0002 | 642.37 | 1582.85 | 1406.22 |
| **...** | ... | ... | ... | ... | ... | ... | ... |
| **20626** | 100 | 196 | -0.0004 | -0.0003 | 643.49 | 1597.98 | 1428.63 |
| **20627** | 100 | 197 | -0.0016 | -0.0005 | 643.54 | 1604.50 | 1433.58 |
| **20628** | 100 | 198 | 0.0004 | 0.0000 | 643.42 | 1602.46 | 1428.18 |
| **20629** | 100 | 199 | -0.0011 | 0.0003 | 643.23 | 1605.26 | 1426.53 |
| **20630** | 100 | 200 | -0.0032 | -0.0005 | 643.85 | 1600.38 | 1432.14 |

20631 rows × 19 columns

In [12]:

```
data_RUL = df.groupby(['ENGINE']).agg({'CYCLE':'max'})
data_RUL.rename(columns={'CYCLE':'LIFE'},inplace=True)
data_RUL.head()
```

Out[12]:

| | LIFE |
|---|---|
| **ENGINE** | |
| **1** | 192 |
| **2** | 287 |
| **3** | 179 |
| **4** | 189 |
| **5** | 269 |

In [13]:

```
df_RUL=df.merge(data_RUL,how='left',on=['ENGINE'])
df_RUL['RUL']=df_RUL['LIFE']-df_RUL['CYCLE']
df_RUL.drop(['LIFE'],axis=1,inplace=True)
df_RUL.drop(['CYCLE'],axis=1,inplace=True)
df_RUL
```

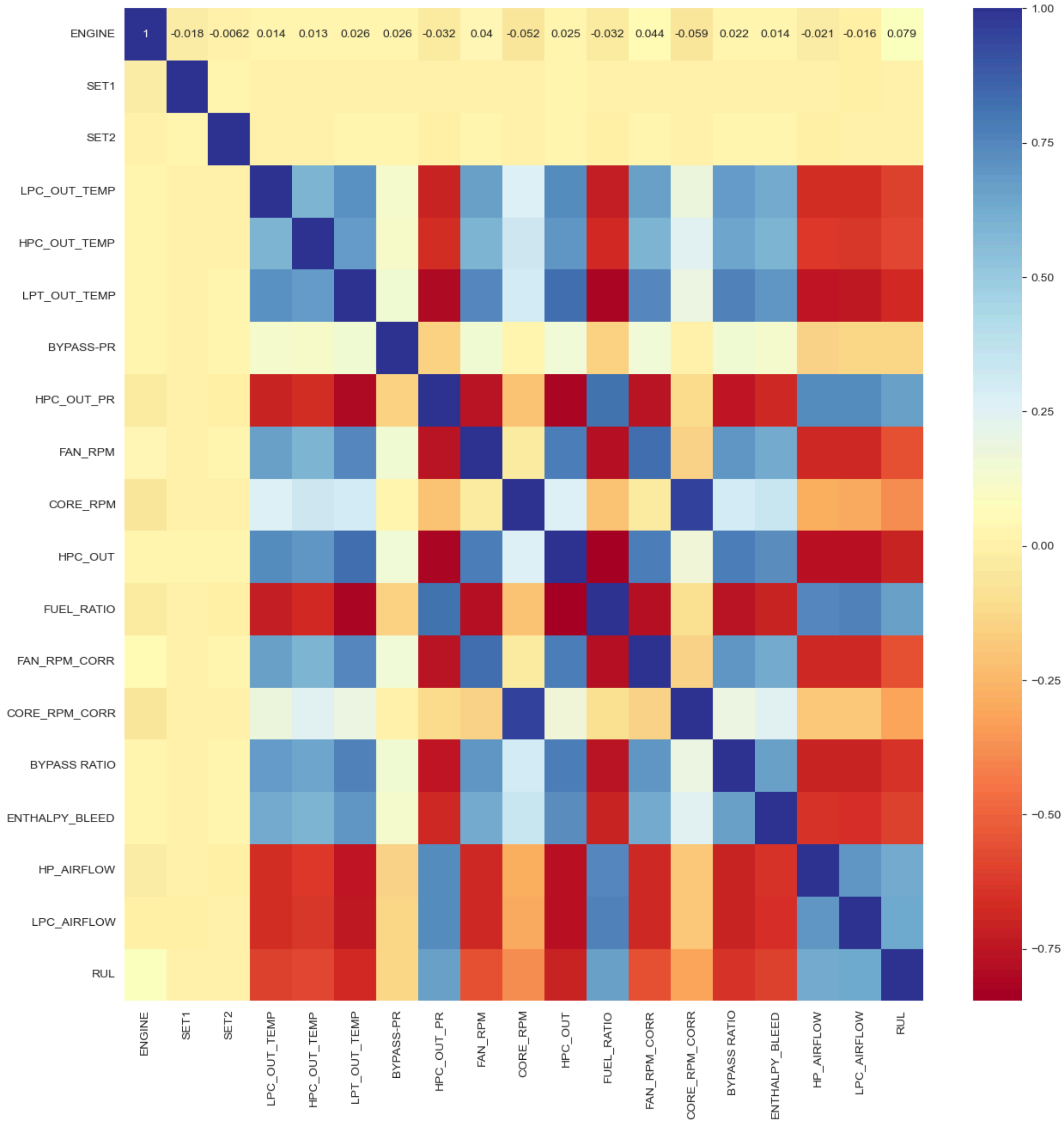
Out[13]:

| | ENGINE | SET1 | SET2 | LPC_OUT_TEMP | HPC_OUT_TEMP | LPT_OUT_TEMP | BYPAS F |
|---|---|---|---|---|---|---|---|
| **0** | 1 | -0.0007 | -0.0004 | 641.82 | 1589.70 | 1400.60 | 21.( |
| **1** | 1 | 0.0019 | -0.0003 | 642.15 | 1591.82 | 1403.14 | 21.( |
| **2** | 1 | -0.0043 | 0.0003 | 642.35 | 1587.99 | 1404.20 | 21.( |
| **3** | 1 | 0.0007 | 0.0000 | 642.35 | 1582.79 | 1401.87 | 21.( |
| **4** | 1 | -0.0019 | -0.0002 | 642.37 | 1582.85 | 1406.22 | 21.( |
| **...** | ... | ... | ... | ... | ... | ... | |
| **20626** | 100 | -0.0004 | -0.0003 | 643.49 | 1597.98 | 1428.63 | 21.( |
| **20627** | 100 | -0.0016 | -0.0005 | 643.54 | 1604.50 | 1433.58 | 21.( |
| **20628** | 100 | 0.0004 | 0.0000 | 643.42 | 1602.46 | 1428.18 | 21.( |
| **20629** | 100 | -0.0011 | 0.0003 | 643.23 | 1605.26 | 1426.53 | 21.( |
| **20630** | 100 | -0.0032 | -0.0005 | 643.85 | 1600.38 | 1432.14 | 21.( |

20631 rows × 19 columns

In [14]:

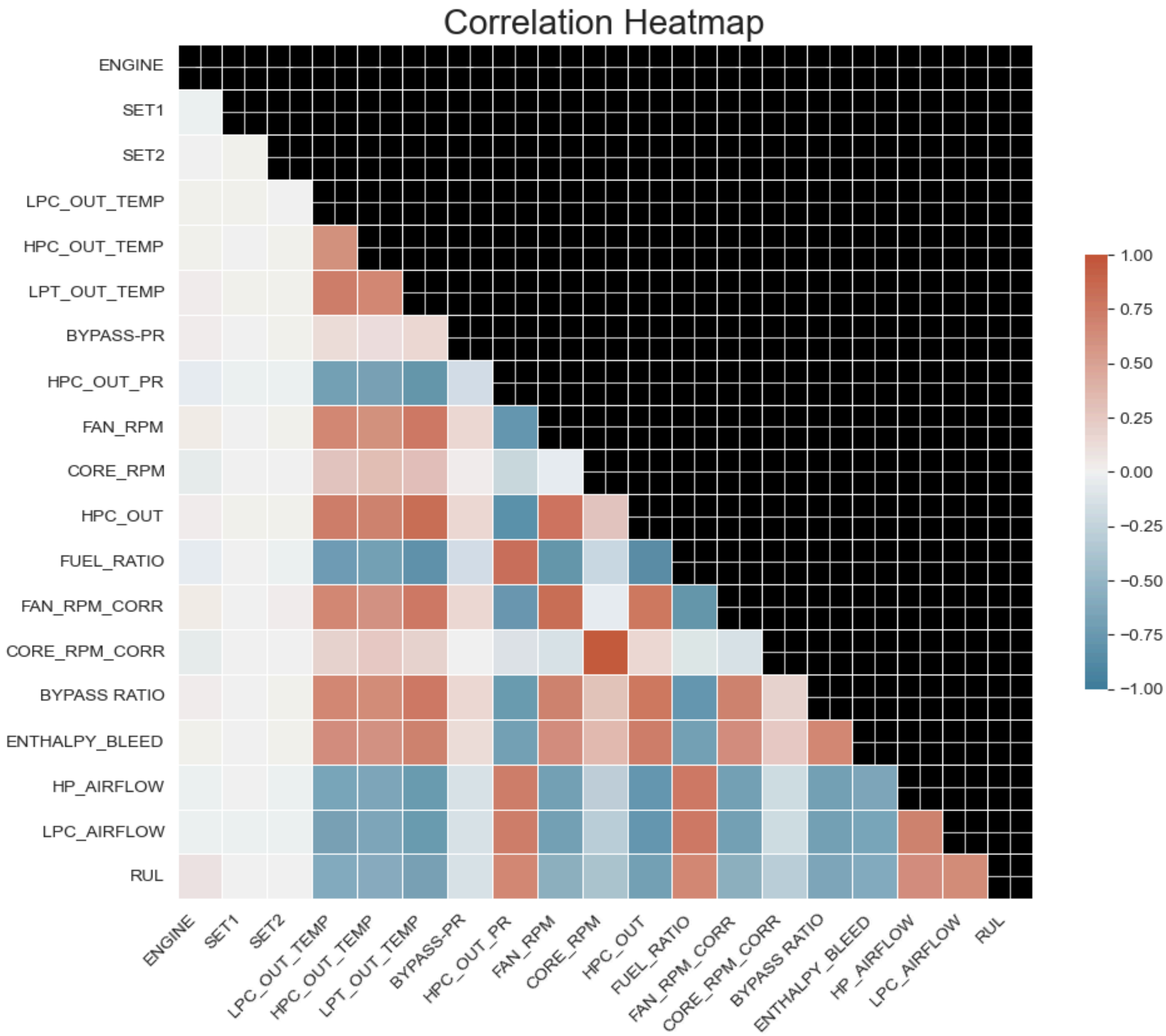
```
plt.figure(figsize=(15,15))
sns.set_style("whitegrid", {"axes.facecolor": ".0"})
df_cluster2 = df_RUL.corr()
plot_kws={"s": 1}
sns.heatmap(df_RUL.corr(),
            cmap='RdYlBu',
            annot=True,
            linecolor='lightgrey').set_facecolor('white')
```




In [15]:

```python
corr = df_RUL.corr()

# Generate a mask for the upper triangle
mask = np.triu(np.ones_like(corr, dtype=bool))

# Set up the matplotlib figure
f, ax = plt.subplots(figsize=(11, 9))

# Generate a custom diverging colormap
cmap = sns.diverging_palette(230, 20, as_cmap=True)

# Draw the heatmap with the mask and correct aspect ratio
sns.heatmap(corr, mask=mask, cmap=cmap, vmax=1, vmin=-1, center=0,
            square=True, linewidths=.5, cbar_kws={"shrink": .5}, annot=True, fmt=".
            annot_kws={"size": 8})

# Enhance readability
```

```
plt.xticks(fontsize=10, rotation=45, ha='right')
plt.yticks(fontsize=10)
plt.title('Correlation Heatmap', fontsize=20)

# Show the heatmap
plt.show()
```




In [16]:
```
correlations_with_RUL = corr['RUL'].drop('RUL')

# Calculate the absolute values of the correlations
absolute_correlations = correlations_with_RUL.abs()

# Sort the absolute correlations in descending order to identify the strongest corr
sorted_indices = absolute_correlations.sort_values(ascending=False).index

# Retrieve the original correlation values for the sorted indices
sorted_correlations = correlations_with_RUL.loc[sorted_indices]

# Print the sorted correlations with their original signs
print(sorted_correlations)
```

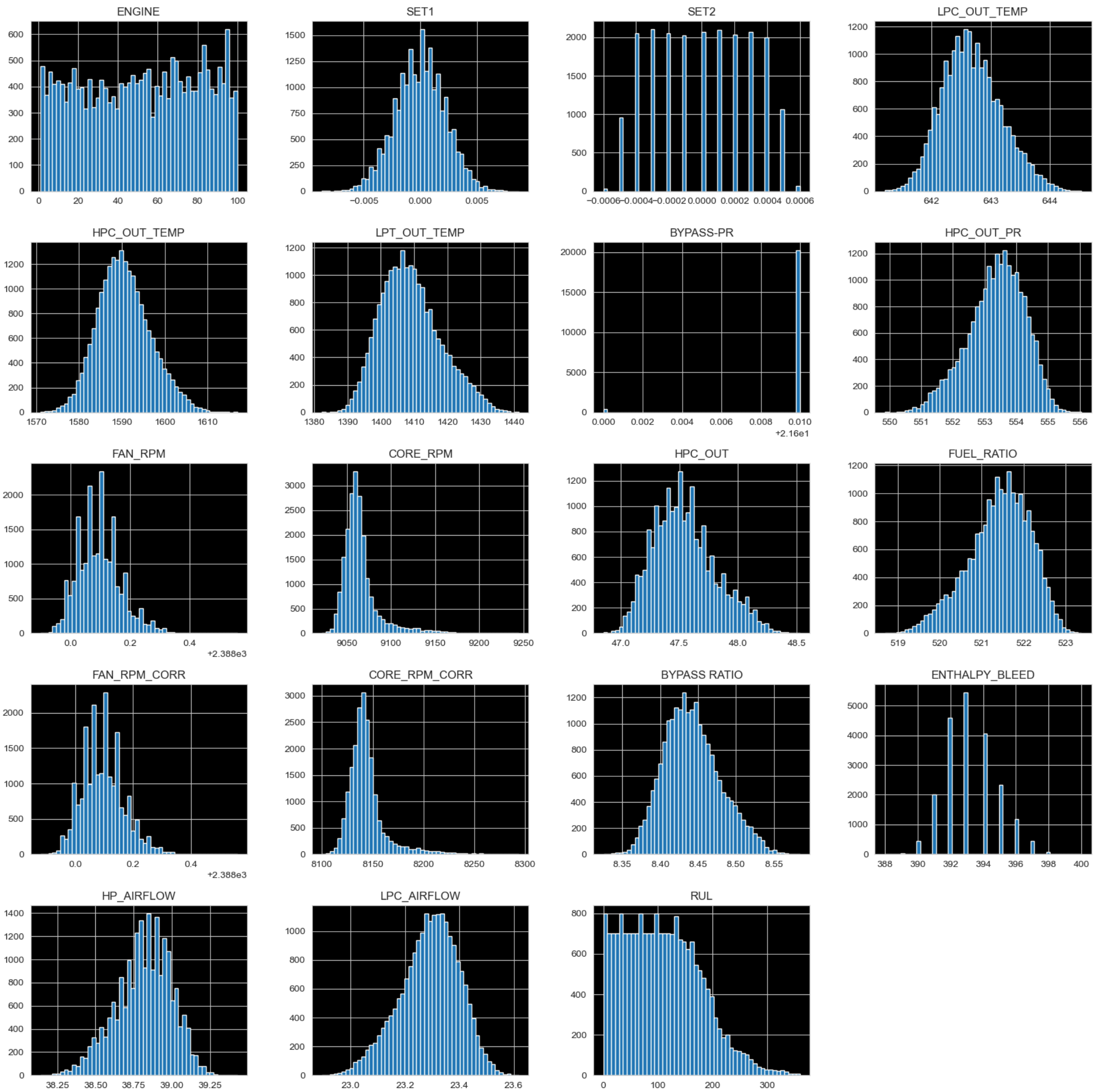
```
HPC_OUT          -0.696228
LPT_OUT_TEMP     -0.678948
FUEL_RATIO        0.671983
HPC_OUT_PR        0.657223
BYPASS RATIO     -0.642667
LPC_AIRFLOW       0.635662
HP_AIRFLOW        0.629428
LPC_OUT_TEMP     -0.606484
ENTHALPY_BLEED   -0.606154
HPC_OUT_TEMP     -0.584520
FAN_RPM          -0.563968
FAN_RPM_CORR     -0.562569
CORE_RPM         -0.390102
CORE_RPM_CORR    -0.306769
BYPASS-PR        -0.128348
ENGINE            0.078753
SET1             -0.003198
SET2             -0.001948
Name: RUL, dtype: float64
```

In [17]:

```
df_RUL.hist(bins=50, figsize=(20, 20))
```

Out[17]:

```
array([[<Axes: title={'center': 'ENGINE'}>,
        <Axes: title={'center': 'SET1'}>,
        <Axes: title={'center': 'SET2'}>,
        <Axes: title={'center': 'LPC_OUT_TEMP'}>],
       [<Axes: title={'center': 'HPC_OUT_TEMP'}>,
        <Axes: title={'center': 'LPT_OUT_TEMP'}>,
        <Axes: title={'center': 'BYPASS-PR'}>,
        <Axes: title={'center': 'HPC_OUT_PR'}>],
       [<Axes: title={'center': 'FAN_RPM'}>,
        <Axes: title={'center': 'CORE_RPM'}>,
        <Axes: title={'center': 'HPC_OUT'}>,
        <Axes: title={'center': 'FUEL_RATIO'}>],
       [<Axes: title={'center': 'FAN_RPM_CORR'}>,
        <Axes: title={'center': 'CORE_RPM_CORR'}>,
        <Axes: title={'center': 'BYPASS RATIO'}>,
        <Axes: title={'center': 'ENTHALPY_BLEED'}>],
       [<Axes: title={'center': 'HP_AIRFLOW'}>,
        <Axes: title={'center': 'LPC_AIRFLOW'}>,
        <Axes: title={'center': 'RUL'}>, <Axes: >]], dtype=object)
```

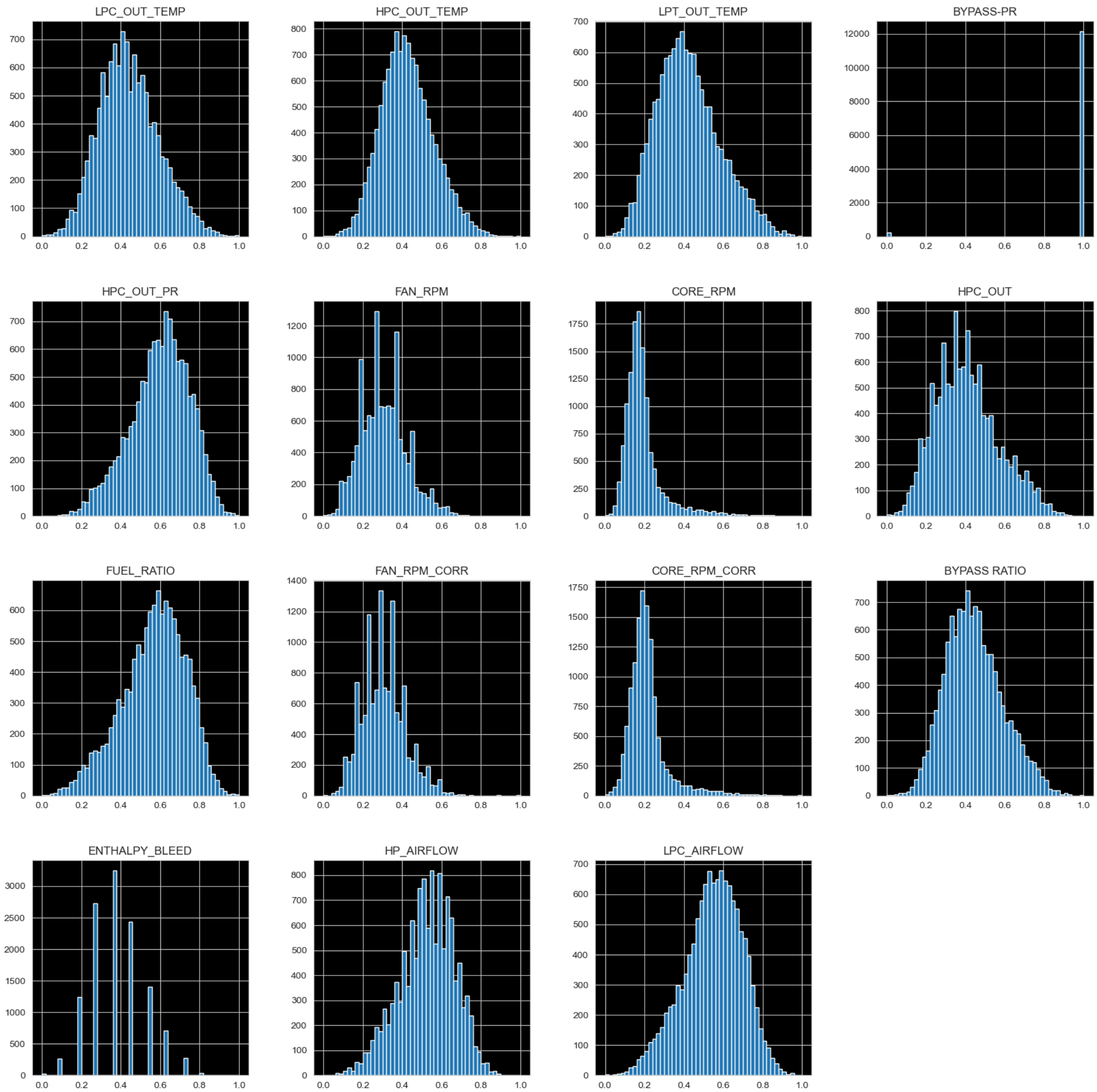

ENGINE
SET1
SET2
LPC_OUT_TEMP
HPC_OUT_TEMP
LPT_OUT_TEMP
BYPASS-PR
HPC_OUT_PR
FAN_RPM
CORE_RPM
HPC_OUT
FUEL_RATIO
FAN_RPM_CORR
CORE_RPM_CORR
BYPASS RATIO
ENTHALPY_BLEED
HP_AIRFLOW
LPC_AIRFLOW
RUL
+2.16e1
+2.388e3
+2.388e3

In [18]:
```
labels={1:"DANGER",2:"NOT DANGER YET",3:"SAFE"}

label=[]

#--Transforming rul values to classes :
for i in df_RUL['RUL']:
    if i<=69:
        label.append(1)
    elif i>69 and i<=135:
        label.append(2)
    else:
        label.append(3)
label=np.array(label)

drop_labels = ['ENGINE', 'SET1', 'SET2']
df_train_test=df_RUL.drop(columns=drop_labels).copy()
# Make a copy of the dataset to apply transformations
df_transformed = df_train_test.copy()
X_train, X_test, y_train, y_test=train_test_split(df_train_test,np.array(label), te
```

In [19]:
```
#scaler = StandardScaler()
scaler = MinMaxScaler()
#Droping the target variable
X_train.drop(columns=['RUL'], inplace=True)
X_test.drop(columns=['RUL'], inplace=True)

X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.fit_transform(X_test)
```

In [20]:
```
X_train_scaled_df = pd.DataFrame(X_train, columns=X_train.columns)

X_train_scaled_df = pd.DataFrame(X_train_scaled, columns=X_train.columns)

X_train_scaled_df.hist(bins=50, figsize=(20, 20))
```

Out[20]:
```
array([[<Axes: title={'center': 'LPC_OUT_TEMP'}>,
        <Axes: title={'center': 'HPC_OUT_TEMP'}>,
        <Axes: title={'center': 'LPT_OUT_TEMP'}>,
        <Axes: title={'center': 'BYPASS-PR'}>],
       [<Axes: title={'center': 'HPC_OUT_PR'}>,
        <Axes: title={'center': 'FAN_RPM'}>,
        <Axes: title={'center': 'CORE_RPM'}>,
        <Axes: title={'center': 'HPC_OUT'}>],
       [<Axes: title={'center': 'FUEL_RATIO'}>,
        <Axes: title={'center': 'FAN_RPM_CORR'}>,
        <Axes: title={'center': 'CORE_RPM_CORR'}>,
        <Axes: title={'center': 'BYPASS RATIO'}>],
       [<Axes: title={'center': 'ENTHALPY_BLEED'}>,
        <Axes: title={'center': 'HP_AIRFLOW'}>,
        <Axes: title={'center': 'LPC_AIRFLOW'}>, <Axes: >]], dtype=object)
```




In [21]:
```
def mean_absolute_error(y_true, y_pred):
    y_true, y_pred = np.array(y_true), np.array(y_pred)
    return np.mean(np.abs((y_true - y_pred) / (y_true)))
```

In [22]:
```
from sklearn.svm import SVC
classifier_SVC = SVC(kernel = 'linear',random_state = 42)
classifier=classifier_SVC
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()
```

```
# Measure the performance
print('SVM-linear')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```

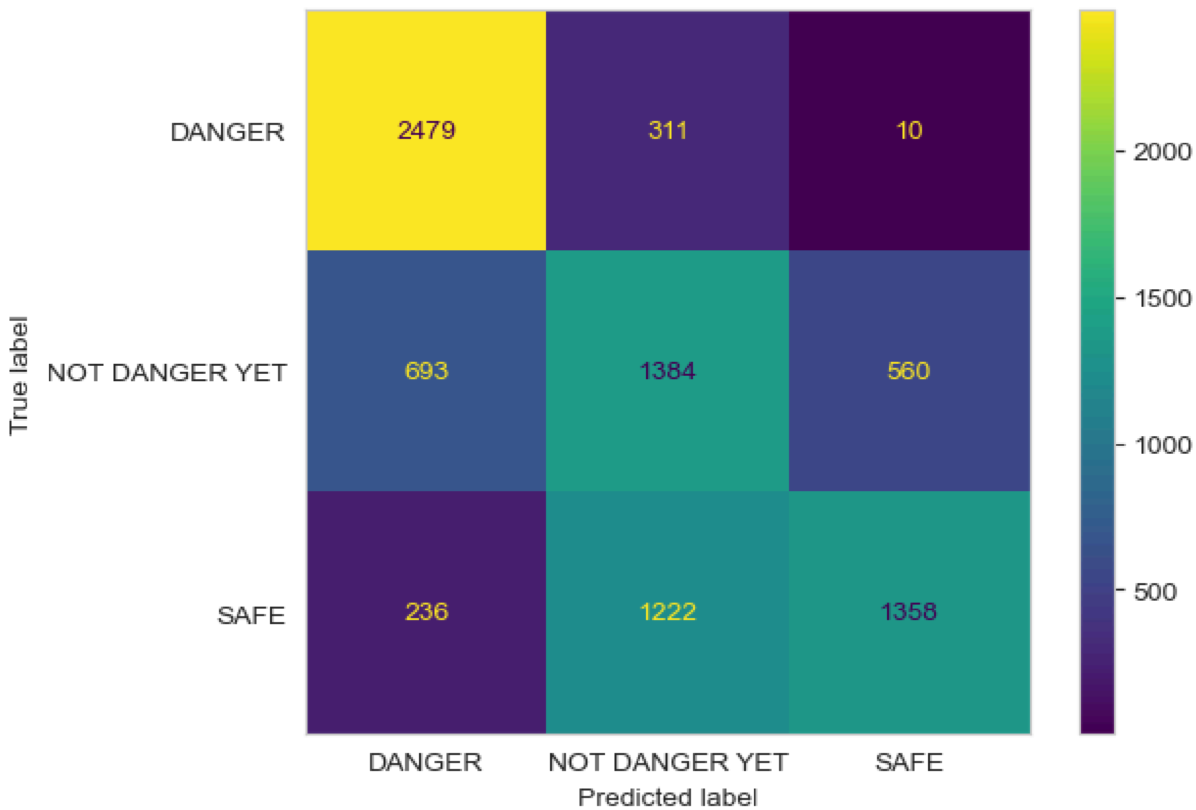

```
SVM-linear
Accuracy score of training 0.686
Error rate of training 0.181
Accuracy score of test 0.660
Error rate of test 0.168
              precision    recall  f1-score   support

           1       0.80      0.88      0.84      2800
           2       0.49      0.57      0.52      2637
           3       0.71      0.53      0.61      2816

    accuracy                           0.66      8253
   macro avg       0.67      0.66      0.66      8253
weighted avg       0.67      0.66      0.66      8253
```

In [23]:
```
from sklearn.ensemble import RandomForestClassifier
classifier_RF=RandomForestClassifier(n_estimators=10)
classifier=classifier_RF
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)
```

```
cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('RandomForestClassifier')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```


```
RandomForestClassifier
Accuracy score of training 0.989
Error rate of training 0.006
Accuracy score of test 0.633
Error rate of test 0.187
              precision    recall  f1-score   support

           1       0.76      0.87      0.81      2800
           2       0.47      0.54      0.50      2637
           3       0.68      0.48      0.57      2816

    accuracy                           0.63      8253
   macro avg       0.64      0.63      0.63      8253
weighted avg       0.64      0.63      0.63      8253
```

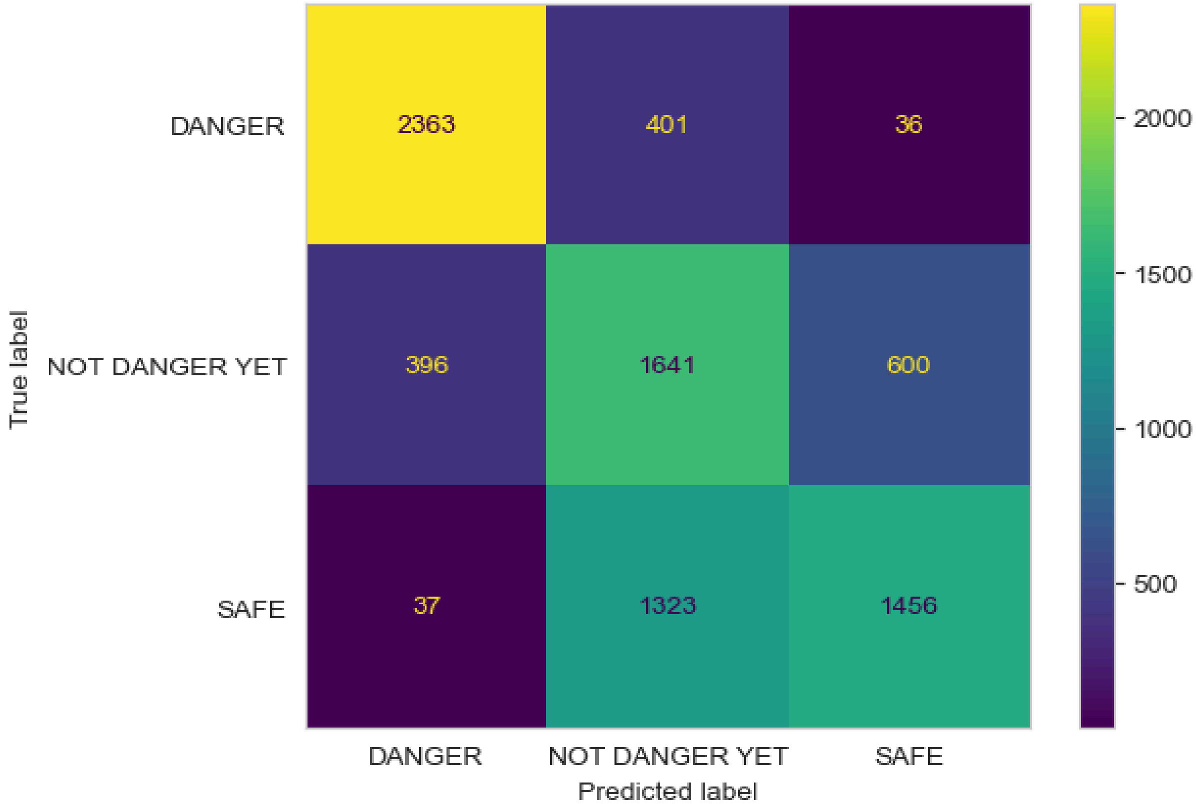
In [24]:
```
from sklearn.naive_bayes import GaussianNB
gnb = GaussianNB()
classifier=gnb
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('GaussianNB')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```



```
GaussianNB
Accuracy score of training 0.647
Error rate of training 0.191
Accuracy score of test 0.633
Error rate of test 0.184
              precision    recall  f1-score   support

           1       0.73      0.89      0.80      2800
           2       0.47      0.52      0.50      2637
           3       0.70      0.48      0.57      2816

    accuracy                           0.63      8253
   macro avg       0.64      0.63      0.62      8253
weighted avg       0.64      0.63      0.63      8253
```

In [25]:

```
from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier(n_neighbors=100)
classifier=knn
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('KNeighborsClassifier')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```




```
KNeighborsClassifier
Accuracy score of training 0.696
Error rate of training 0.187
Accuracy score of test 0.662
Error rate of test 0.174
              precision    recall  f1-score   support

           1       0.85      0.84      0.84      2800
           2       0.49      0.62      0.55      2637
           3       0.70      0.52      0.59      2816

    accuracy                           0.66      8253
   macro avg       0.68      0.66      0.66      8253
weighted avg       0.68      0.66      0.66      8253
```

In [26]:

```
from sklearn.linear_model import LogisticRegression
logistic_regression_model = LogisticRegression(max_iter=1000, random_state=42)
classifier=logistic_regression_model
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()
```

```
# Measure the performance
print('LogisticRegression')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```


```
LogisticRegression
Accuracy score of training 0.684
Error rate of training 0.179
Accuracy score of test 0.655
Error rate of test 0.170
              precision    recall  f1-score   support

           1       0.77      0.89      0.82      2800
           2       0.49      0.55      0.51      2637
           3       0.72      0.52      0.61      2816

    accuracy                           0.66      8253
   macro avg       0.66      0.65      0.65      8253
weighted avg       0.66      0.66      0.65      8253
```

In [27]:

```
from sklearn.dummy import DummyClassifier
stratified_clf = DummyClassifier(strategy='stratified', random_state=42)
most_frequent_clf = DummyClassifier(strategy='most_frequent', random_state=42)
uniform_clf = DummyClassifier(strategy='uniform', random_state=42)

baseline_classifiers = {
    'Stratified': stratified_clf,
```

```
    'Most Frequent': most_frequent_clf,
    'Uniform': uniform_clf
}
for name, baseline_clf in baseline_classifiers.items():
    baseline_clf.fit(X_train_scaled, y_train)  # Note: Baseline classifiers do not
    y_baseline_pred_test = baseline_clf.predict(X_test_scaled)

    print(f"Evaluation of {name} Classifier:")
    print("Accuracy score of test: %.3f" % accuracy_score(y_test, y_baseline_pred_t
    print(classification_report(y_test, y_baseline_pred_test))
    print("-" * 80)
```

```
Evaluation of Stratified Classifier:
Accuracy score of test: 0.344
              precision    recall  f1-score   support

           1       0.36      0.34      0.35      2800
           2       0.34      0.35      0.34      2637
           3       0.34      0.34      0.34      2816

    accuracy                           0.34      8253
   macro avg       0.34      0.34      0.34      8253
weighted avg       0.34      0.34      0.34      8253

--------------------------------------------------------------------------------
Evaluation of Most Frequent Classifier:
Accuracy score of test: 0.341
              precision    recall  f1-score   support

           1       0.00      0.00      0.00      2800
           2       0.00      0.00      0.00      2637
           3       0.34      1.00      0.51      2816

    accuracy                           0.34      8253
   macro avg       0.11      0.33      0.17      8253
weighted avg       0.12      0.34      0.17      8253

--------------------------------------------------------------------------------
Evaluation of Uniform Classifier:
Accuracy score of test: 0.329
              precision    recall  f1-score   support

           1       0.33      0.33      0.33      2800
           2       0.32      0.33      0.32      2637
           3       0.33      0.32      0.33      2816

    accuracy                           0.33      8253
   macro avg       0.33      0.33      0.33      8253
weighted avg       0.33      0.33      0.33      8253

--------------------------------------------------------------------------------
```

In [28]:

```
# def update_rolling_mean(data, mask):
#     for x, group in mask.groupby("ENGINE"):
#         for x in X_train.columns:
#             data.loc[group.index[10:], x+"_rolling"] = data.loc[group.index, x].r
```

```
#              data.loc[group.index[:10], x+"_rolling"] = data.loc[group.index[:10],
# drop_labels = ['BYPASS-PR', 'SET1', 'SET2']
# df_rolling=df_RUL.drop(columns=drop_labels).copy()
# update_rolling_mean(df_rolling, df_rolling)
# df_rolling
def update_rolling_mean(data):
    # Define a function to apply to each group
    def apply_rolling(group):
        for col in group.columns:
            if col != 'ENGINE' and col != 'RUL':
                # Apply rolling mean with a window of 10, min_periods=1 ensures we
                group[col + "_rolling"] = group[col].rolling(window=10, min_periods
        return group

    # Group by 'ENGINE' and apply the rolling function
    data = data.groupby('ENGINE').apply(apply_rolling)
    data.reset_index(inplace=True, drop=True)
    # Optional: drop original columns if you only want to keep the rolling features
    # for col in data.columns:
    #     if not col.endswith('_rolling') and col != 'ENGINE' and col != 'RUL' :
    #         data.drop(col, inplace=True, axis=1)

    return data
df_rolling = df_RUL.copy()  # Assuming df_RUL is your initial DataFrame
df_rolling = update_rolling_mean(df_rolling)
df_rolling
```

Out[28]:

| | ENGINE | SET1 | SET2 | LPC_OUT_TEMP | HPC_OUT_TEMP | LPT_OUT_TEMP | BYPAS F |
|---|---|---|---|---|---|---|---|
| **0** | 1 | -0.0007 | -0.0004 | 641.82 | 1589.70 | 1400.60 | 21.( |
| **1** | 1 | 0.0019 | -0.0003 | 642.15 | 1591.82 | 1403.14 | 21.( |
| **2** | 1 | -0.0043 | 0.0003 | 642.35 | 1587.99 | 1404.20 | 21.( |
| **3** | 1 | 0.0007 | 0.0000 | 642.35 | 1582.79 | 1401.87 | 21.( |
| **4** | 1 | -0.0019 | -0.0002 | 642.37 | 1582.85 | 1406.22 | 21.( |
| **...** | ... | ... | ... | ... | ... | ... | |
| **20626** | 100 | -0.0004 | -0.0003 | 643.49 | 1597.98 | 1428.63 | 21.( |
| **20627** | 100 | -0.0016 | -0.0005 | 643.54 | 1604.50 | 1433.58 | 21.( |
| **20628** | 100 | 0.0004 | 0.0000 | 643.42 | 1602.46 | 1428.18 | 21.( |
| **20629** | 100 | -0.0011 | 0.0003 | 643.23 | 1605.26 | 1426.53 | 21.( |
| **20630** | 100 | -0.0032 | -0.0005 | 643.85 | 1600.38 | 1432.14 | 21.( |

20631 rows × 36 columns

In [29]:

```
corr = df_rolling.corr()
correlations_with_RUL = corr['RUL'].drop('RUL')
```

```
# Calculate the absolute values of the correlations
absolute_correlations = correlations_with_RUL.abs()

# Sort the absolute correlations in descending order to identify the strongest corr
sorted_indices = absolute_correlations.sort_values(ascending=False).index

# Retrieve the original correlation values for the sorted indices
sorted_correlations = correlations_with_RUL.loc[sorted_indices]
features_to_drop = absolute_correlations[absolute_correlations < 0.1].index
# Print the sorted correlations with their original signs
print(sorted_correlations)
features_to_drop
```

```
LPT_OUT_TEMP_rolling     -0.732868
LPC_AIRFLOW_rolling       0.729701
HPC_OUT_rolling          -0.728727
ENTHALPY_BLEED_rolling   -0.728268
BYPASS RATIO_rolling     -0.727228
HPC_OUT_TEMP_rolling     -0.726007
HP_AIRFLOW_rolling        0.723255
LPC_OUT_TEMP_rolling     -0.721540
HPC_OUT_PR_rolling        0.711189
FUEL_RATIO_rolling        0.710306
HPC_OUT                  -0.696228
LPT_OUT_TEMP             -0.678948
FUEL_RATIO                0.671983
HPC_OUT_PR                0.657223
BYPASS RATIO             -0.642667
LPC_AIRFLOW               0.635662
HP_AIRFLOW                0.629428
LPC_OUT_TEMP             -0.606484
ENTHALPY_BLEED           -0.606154
FAN_RPM_rolling          -0.592877
FAN_RPM_CORR_rolling     -0.591497
HPC_OUT_TEMP             -0.584520
FAN_RPM                  -0.563968
FAN_RPM_CORR             -0.562569
CORE_RPM_rolling         -0.393039
CORE_RPM                 -0.390102
CORE_RPM_CORR            -0.306769
CORE_RPM_CORR_rolling    -0.305386
BYPASS-PR_rolling        -0.301141
BYPASS-PR                -0.128348
ENGINE                    0.078753
SET2_rolling             -0.016361
SET1_rolling             -0.012197
SET1                     -0.003198
SET2                     -0.001948
Name: RUL, dtype: float64
```

Out[29]:
```
Index(['ENGINE', 'SET1', 'SET2', 'SET1_rolling', 'SET2_rolling'], dtype='object')
```

In [30]:
```
df_rolling.drop(columns=[ 'SET1', 'SET2', 'SET1_rolling', 'SET2_rolling','BYPASS-PR
X_train, X_test, y_train, y_test=train_test_split(df_rolling.drop(columns=['ENGINE'
```

```
df_rolling.iloc[-1,-14:]=df_rolling.iloc[-2,-14:]
df_rolling
```

Out[30]:

| | ENGINE | LPC_OUT_TEMP | HPC_OUT_TEMP | LPT_OUT_TEMP | HPC_OUT_PR | FAN_RPM |
|---|---|---|---|---|---|---|
| **0** | 1 | 641.82 | 1589.70 | 1400.60 | 554.36 | 2388.06 |
| **1** | 1 | 642.15 | 1591.82 | 1403.14 | 553.75 | 2388.04 |
| **2** | 1 | 642.35 | 1587.99 | 1404.20 | 554.26 | 2388.08 |
| **3** | 1 | 642.35 | 1582.79 | 1401.87 | 554.45 | 2388.11 |
| **4** | 1 | 642.37 | 1582.85 | 1406.22 | 554.00 | 2388.06 |
| **...** | ... | ... | ... | ... | ... | ... |
| **20626** | 100 | 643.49 | 1597.98 | 1428.63 | 551.43 | 2388.19 |
| **20627** | 100 | 643.54 | 1604.50 | 1433.58 | 550.86 | 2388.23 |
| **20628** | 100 | 643.42 | 1602.46 | 1428.18 | 550.94 | 2388.24 |
| **20629** | 100 | 643.23 | 1605.26 | 1426.53 | 550.68 | 2388.25 |
| **20630** | 100 | 643.85 | 1600.38 | 1432.14 | 550.79 | 2388.26 |

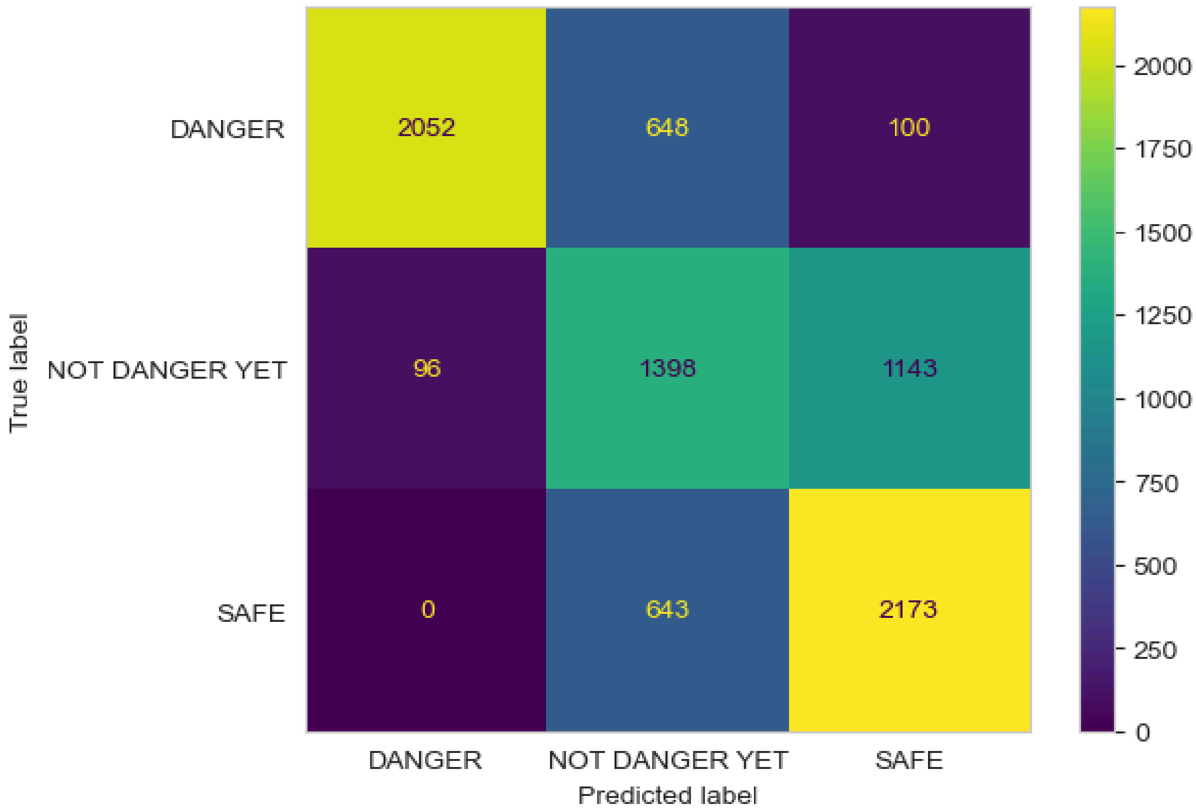
20631 rows × 30 columns

In [31]:

```
X_train_scaled=scaler.fit_transform(X_train)
X_test_scaled=scaler.fit_transform(X_test)
```

In [32]:

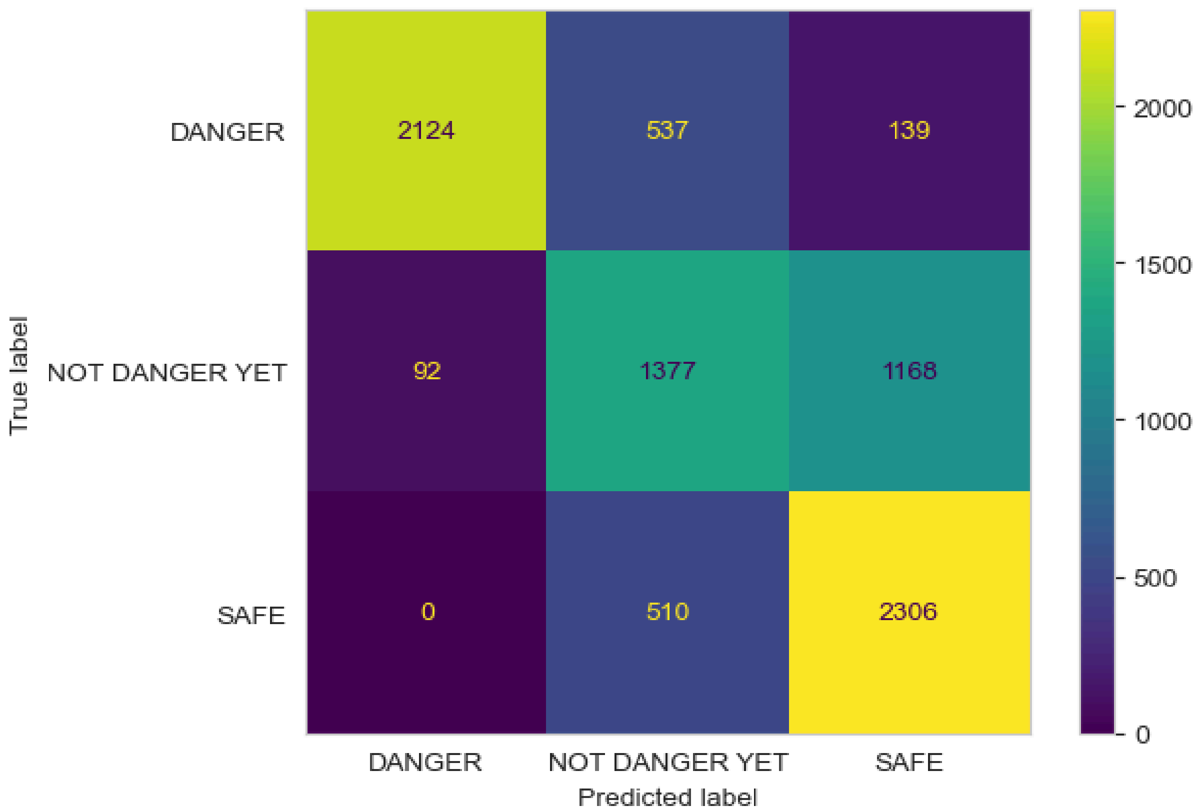
```
classifier_SVC = SVC(kernel = 'linear',random_state = 42)
classifier=classifier_SVC
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('SVM-linear')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```




```
SVM-linear
Accuracy score of training 0.691
Error rate of training 0.178
Accuracy score of test 0.681
Error rate of test 0.204
              precision    recall  f1-score   support

           1       0.96      0.73      0.83      2800
           2       0.52      0.53      0.52      2637
           3       0.64      0.77      0.70      2816

    accuracy                           0.68      8253
   macro avg       0.70      0.68      0.68      8253
weighted avg       0.71      0.68      0.69      8253
```

In [33]:
```
from sklearn.svm import SVC
classifier_SVC = SVC(kernel = 'poly',random_state = 42)
classifier=classifier_SVC
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()
```

```
# Measure the performance
print('SVM-Ploy')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```

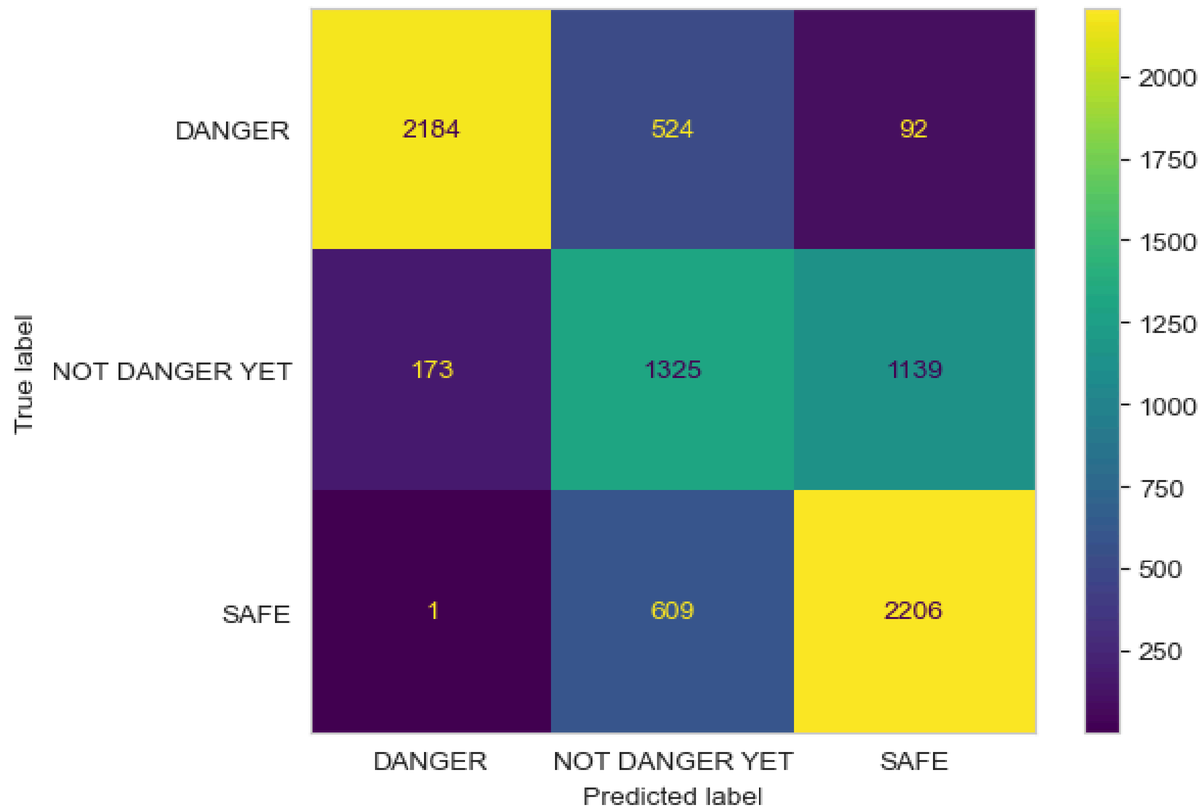



```
SVM-Ploy
Accuracy score of training 0.725
Error rate of training 0.172
Accuracy score of test 0.704
Error rate of test 0.196
              precision    recall  f1-score   support

           1       0.96      0.76      0.85      2800
           2       0.57      0.52      0.54      2637
           3       0.64      0.82      0.72      2816

    accuracy                           0.70      8253
   macro avg       0.72      0.70      0.70      8253
weighted avg       0.72      0.70      0.71      8253
```

In [34]:

```
logistic_regression_model = LogisticRegression(max_iter=1000, random_state=42)
classifier=logistic_regression_model
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)
```

```
cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('LogisticRegression')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```

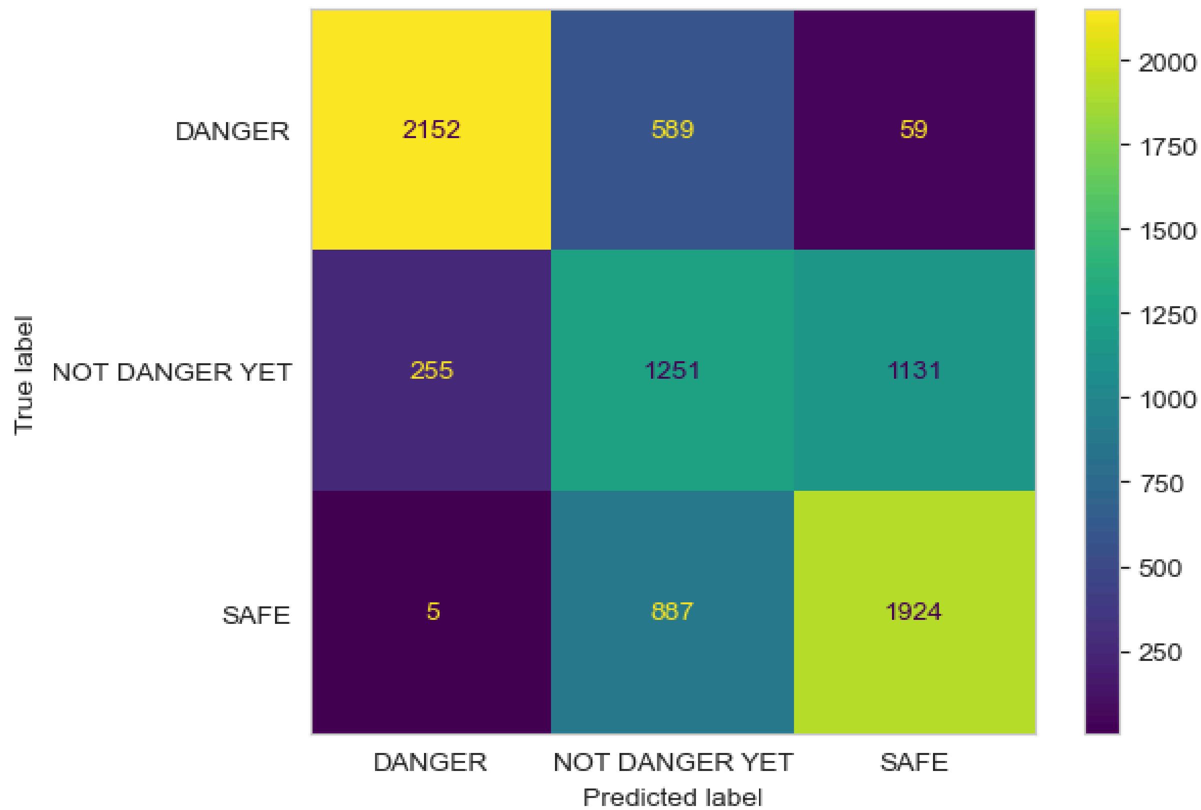



```
LogisticRegression
Accuracy score of training 0.690
Error rate of training 0.174
Accuracy score of test 0.692
Error rate of test 0.190
              precision    recall  f1-score   support

           1       0.93      0.78      0.85      2800
           2       0.54      0.50      0.52      2637
           3       0.64      0.78      0.71      2816

    accuracy                           0.69      8253
   macro avg       0.70      0.69      0.69      8253
weighted avg       0.71      0.69      0.69      8253
```

In [35]:
```
classifier_RF=RandomForestClassifier(n_estimators=10)
classifier=classifier_RF
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('RandomForestClassifier')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```

```
RandomForestClassifier
Accuracy score of training 0.993
Error rate of training 0.004
Accuracy score of test 0.678
Error rate of test 0.189
              precision    recall  f1-score   support

           1       0.86      0.82      0.84      2800
           2       0.53      0.51      0.52      2637
           3       0.65      0.70      0.67      2816

    accuracy                           0.68      8253
   macro avg       0.68      0.67      0.68      8253
weighted avg       0.68      0.68      0.68      8253
```

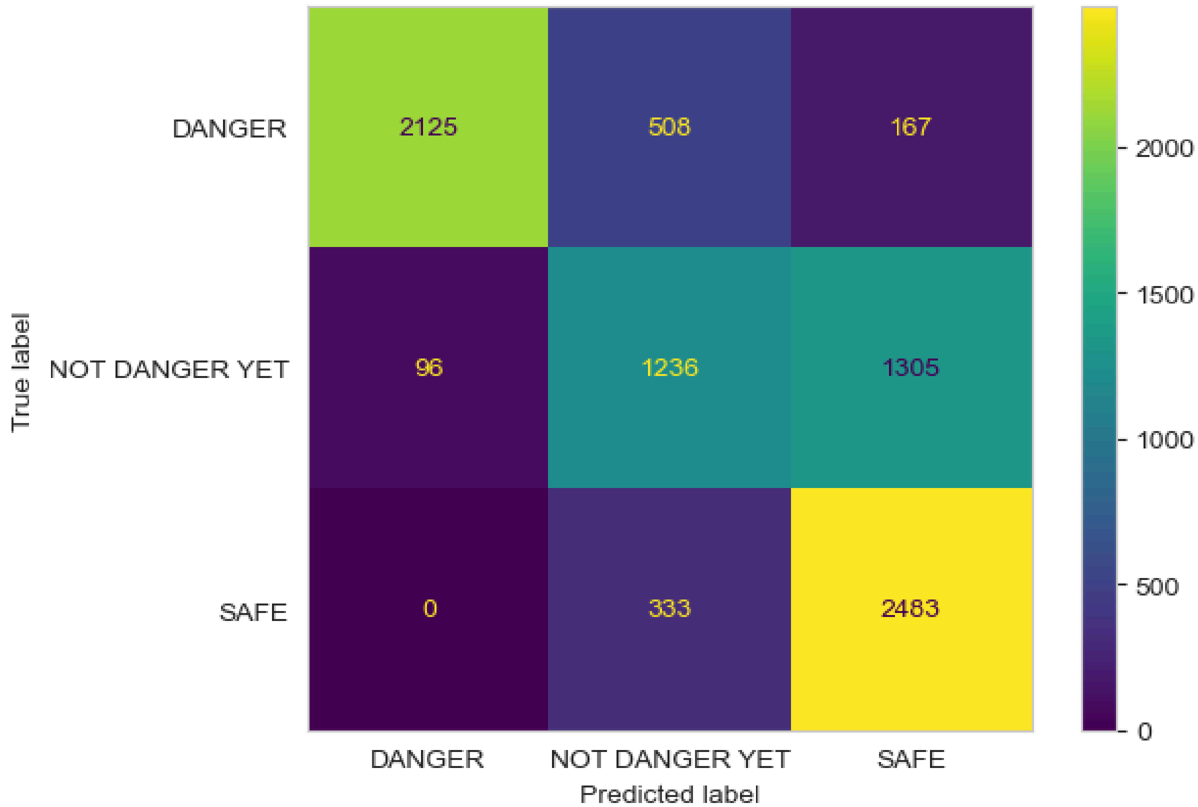
In [36]:
```
gnb = GaussianNB()
classifier=gnb
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
print('GaussianNB')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```




```
GaussianNB
Accuracy score of training 0.638
Error rate of training 0.200
Accuracy score of test 0.645
Error rate of test 0.206
              precision    recall  f1-score   support

           1       0.89      0.77      0.83      2800
           2       0.46      0.47      0.47      2637
           3       0.62      0.68      0.65      2816

    accuracy                           0.65      8253
   macro avg       0.66      0.64      0.65      8253
weighted avg       0.66      0.65      0.65      8253
```

In [37]:
```
knn=KNeighborsClassifier(n_neighbors=100)
classifier=knn
classifier.fit(X_train_scaled,np.array(y_train))
y_svc_train=classifier.predict(X_train_scaled)

y_svc_test=classifier.predict(X_test_scaled)


cm= confusion_matrix(y_test, y_svc_test, labels=classifier.classes_)
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=[labels[classifie
disp.plot()
plt.grid(False)
plt.show()

# Measure the performance
```

```
print('KNeighborsClassifier')
print("Accuracy score of training %.3f" %metrics.accuracy_score(y_train, y_svc_trai
print("Error rate of training %.3f" %mean_absolute_error(y_train,y_svc_train))
print("Accuracy score of test %.3f" %metrics.accuracy_score(y_test, y_svc_test))
print("Error rate of test %.3f" %mean_absolute_error(y_test,y_svc_test))
print(classification_report(y_test,y_svc_test))
```




```
KNeighborsClassifier
Accuracy score of training 0.723
Error rate of training 0.172
Accuracy score of test 0.708
Error rate of test 0.200
              precision    recall  f1-score   support

           1       0.96      0.76      0.85      2800
           2       0.60      0.47      0.52      2637
           3       0.63      0.88      0.73      2816

    accuracy                           0.71      8253
   macro avg       0.73      0.70      0.70      8253
weighted avg       0.73      0.71      0.70      8253
```