Prédiction des défauts de roulement des moteurs à induction

Ce projet consiste à prédire les défauts de roulement dans les moteurs inductions par Machine Learning. Pour ce faire, nous disposons d'une base de données préalablement constituée. Cette base de données a été constitué à partir des différents états (sain et avec défaut) d'un roulement de type **6205-2RS 52x25x15mm**. On a trois états : état sain, avec défaut sur la cage extérieur et avec défaut sur la cage intérieur. Les mésures ont été faites pour trois vitesses(v1=1060 rpm, v2=1480 rpm, v3=2000 rpm) différentes du roulement. Nous avons donc ici un problème de classification multiclasse. Nous allons d'abord traiter la donnée pour la rendre exploitable et ensuite nous appliquerons trois modèles de Machine Learning et un modèle de Deep Learning à cette dernière. Nous utiliserons l'**accuracy** et le **weighted avg** commes métriques pour la validation.

Entrée [58]:

```
#importation des bibliothèques
import numpy as np
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn import preprocessing
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split,cross_validate
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import LinearSVC
from sklearn.neural_network import MLPClassifier
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
from sklearn.metrics import plot_confusion_matrix
Fs=25600 # Fs est fréquence d'échantillonnage des signaux
```

Chargement de la base de données

Entrée [59]:

```
# Les signaux pour la vitesse v1
df_h=pd.read_excel('v1_healthy_vib_0.xlsx')
df_CE=pd.read_excel('v1_CE_fault_vib_0.xlsx')
df_CI=pd.read_excel('v1_CI_fault_vib_0.xlsx')
```

Entrée [60]:

```
# Les signaux pour la vitesse v2
df_hv2=pd.read_excel('v2_healthy_vib_0.xlsx')
df_CE_v2=pd.read_excel('v2_CE_fault_vib_0.xlsx')
df_CI_v2=pd.read_excel('v2_CI_fault_vib_0.xlsx')
```

Entrée [61]:

```
# Les signaux pour la vitesse v3
df_hv3=pd.read_excel('v3_healthy_vib_0.xlsx')
df_CE_v3=pd.read_excel('v3_CE_fault_vib_0.xlsx')
df_CI_v3=pd.read_excel('v3_CI_fault_vib_0.xlsx')
```

Entrée [62]:

```
# Affichage du signal sain pour la vitesse v1
df_h
```

Out[62]:

	vib_0	vib_0.1	vib_0.2	vib_0.3	vib_0.4	vib_0.5	vib_0.6	vib_0.7	vib_
0	0.000069	0.000102	0.000140	0.000154	0.000030	0.000154	0.000113	0.000093	0.000
1	0.000148	0.000080	0.000079	0.000058	0.000114	0.000013	0.000129	0.000122	0.000
2	0.000068	0.000136	0.000121	0.000130	0.000156	0.000229	0.000075	0.000155	0.000
3	0.000073	0.000130	0.000181	0.000068	0.000104	0.000105	0.000046	0.000093	0.000
4	0.000066	0.000098	0.000085	0.000121	0.000130	0.000122	0.000138	0.000109	0.000
...	...	...	...	...	...	...	...	...	
76795	0.000118	0.000121	0.000109	0.000104	0.000051	0.000166	0.000149	0.000037	0.000
76796	0.000142	0.000150	0.000076	0.000135	0.000048	0.000070	0.000063	0.000113	0.000
76797	0.000073	0.000051	0.000068	0.000141	0.000132	0.000066	0.000097	0.000123	0.000
76798	0.000140	0.000069	0.000175	0.000095	0.000089	0.000163	0.000110	0.000108	0.000
76799	0.000142	0.000155	0.000112	0.000096	0.000209	0.000154	0.000124	0.000122	0.000

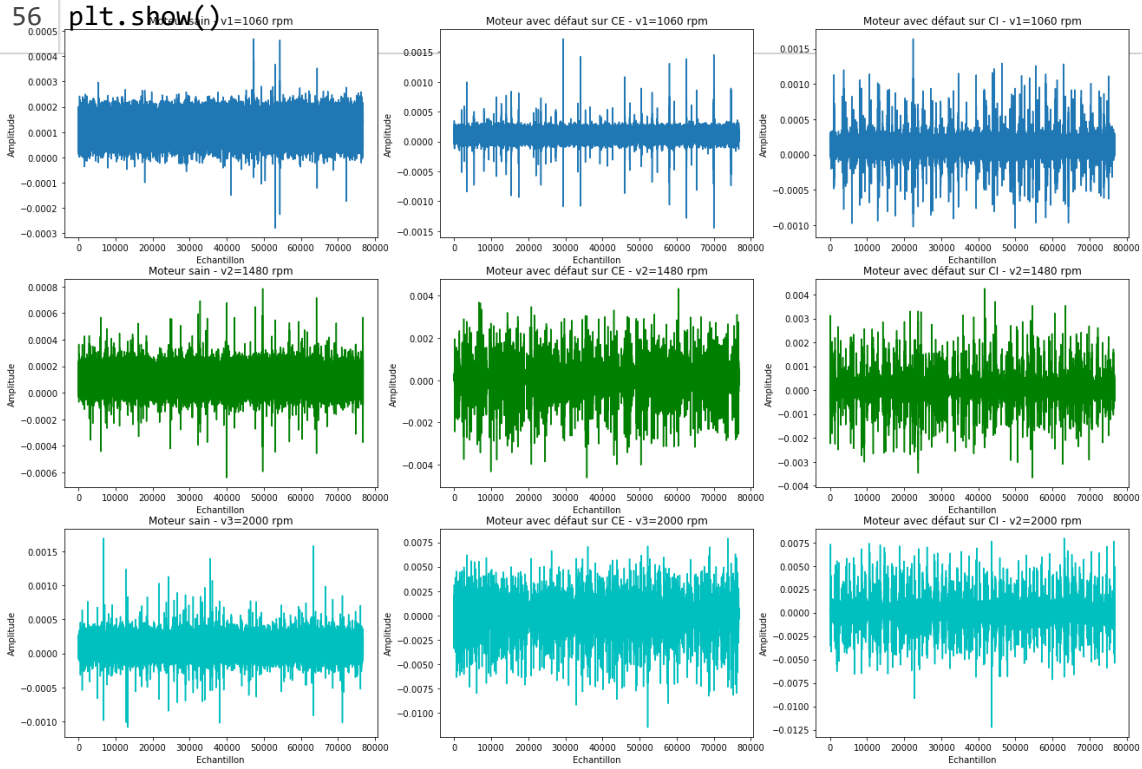
76800 rows × 70 columns

Visualisation des premiers signaux pour chaque état et à chaque vitesse

Entrée [63]:

```
plt.figure(figsize=(22,15))
plt.subplot(3,3,1)
plt.plot(df_h["vib_0"])
plt.title('Moteur sain - v1=1060 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,2)
plt.plot(df_CE["vib_0"])
plt.title('Moteur avec défaut sur CE - v1=1060 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,3)
plt.plot(df_CI["vib_0"])
plt.title('Moteur avec défaut sur CI - v1=1060 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,4)
plt.plot(df_hv2["vib_0"],c='g')
plt.title('Moteur sain - v2=1480 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,5)
plt.plot(df_CE_v2["vib_0"],c='g')
plt.title('Moteur avec défaut sur CE - v2=1480 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,6)
plt.plot(df_CI_v2["vib_0"],c='g')
plt.title('Moteur avec défaut sur CI - v2=1480 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,7)
plt.plot(df_hv3["vib_0"],c='c')
plt.title('Moteur sain - v3=2000 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,8)
plt.plot(df_CE_v3["vib_0"],c='c')
plt.title('Moteur avec défaut sur CE - v3=2000 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

plt.subplot(3,3,9)
plt.plot(df_CI_v3["vib_0"],c='c')
plt.title('Moteur avec défaut sur CI - v2=2000 rpm')
plt.xlabel('Echantillon')
plt.ylabel('Amplitude')

```

```
56 plt.show()
```



On remarque une différence entre les signaux pour différents états. Il y a aussi une différences entre les signaux selon la vitesse pour le même état. Le challenge lors de la classification serait de pouvoir classifier les états quelle que soit la vitesse.

Nous allons maintenant extraire les caratéristiques des signaux et constituer un dataframe avant de passer à la classification

Caractéristique à extraire :

- max : le maximum du signal;
- min : le minimum du signal;
- mean : la moyenne du signal;
- sd : la variance du signal;
- frequence: La fréquence moyenne du signal;
- rms : la valeur efficace du signal;
- skewness : l'asymétrie du signal;
- kurtosis : l'applatissement du signal;
- crest : le facteur de crête du signal;
- form : indice de forme du signal;
- etat : c'est le label prenant 0, 1 ou 2.

Ecriture de la fonctin d'extraction

Nous allons définir ici la fonction *feature_extract* qui prend en paramètre *sig* : un signal sous forme de colonne pandas ; *Fs* : la fréquence d'échantillonnage du signal; *df* : un dataframe pandas donc les colonnes sont les caractéristiques (celles énumerées ci-dessous) ; *etat* : un nombre entier. Cette fonction va calculer les caractéristiques (celles énumerées ci-dessous) du signal *sig*, les ajouter à la ligne suivante du dataframe *df* (la valeur etat sera ajouté dans la colonne *etat*) et retourner le ce dernier.

Entrée [64]:

```python
def feature_extract(sig,Fs,df,etat:int):
    # Extraction des valeurs de la colonne pandas
    x = sig.values

    # calcul des indicateurs
    rms = np.sqrt(np.mean(x**2))  # RMS
    mean = np.mean(x)  # Mean
    sd = np.std(x)  # Sd
    skewness = np.mean((x - np.mean(x))**3) / np.std(x)**3  # Skewness
    kurtosis = np.mean((x - np.mean(x))**4) / np.std(x)**4  # Kurtosis
    crest = np.max(np.abs(x)) / rms  # Crest
    form = rms / np.mean(np.abs(x))  # Form
    max_val = np.max(x)  # valeur maximum
    min_val = np.min(x)  # valeur minimum
    freqs = np.fft.fftfreq(len(x), d=1/Fs)
    fft = np.fft.fft(x)
    mean_freq = np.sum(np.abs(freqs) * np.abs(fft)**2) / np.sum(np.abs(fft)**2)  # f

    # Creation d'un ligne pandas avec les caractéristiques
    new_row = pd.DataFrame({"max": max_val,
                            "min": min_val,
                            "mean": mean,
                            "sd": sd,
                            "frequence": mean_freq,
                            "rms": rms,
                            "skewness": skewness,
                            "kurtosis": kurtosis,
                            "crest": crest,
                            "form": form,
                            "etat": etat},
                           index=[0])

    # Ajout de cette nouvelle ligne au dataframe
    df = pd.concat([df, new_row], ignore_index=True)
    # Convertion de la colonne etat en type int
    df['etat']=df['etat'].astype(int)
    return df
```

Création de notre dataset

On va appliquer de la fonction *feature_extract* en boucle sur les colonnes des dataframes contenants les signaux pour créer les datasets selon les vitesses(*df* pour *v1*, *df_v2* pour *v2* et *df_v3* pour *v3*) et le dataset générale *data* pour toute la donnée.

Entrée [66]:

```python
# Creation d'un dataframe pandas vide
df=pd.DataFrame(columns=["max","min","mean","sd","frequence","rms","skewness","kurto
```

Entrée [67]:

```
# remplissage de df
for el in df_h.keys():
    df=feature_extract(df_h[el],Fs,df,0)
for el in df_CE.keys():
    df=feature_extract(df_CE[el],Fs,df,1)
for el in df_CI.keys():
    df=feature_extract(df_CI[el],Fs,df,2)
```

Entrée [68]:

```
# Creation d'un dataframe pandas vide
df_v2=pd.DataFrame(columns=["max","min","mean","sd","frequence","rms","skewness","ku
```

Entrée [69]:

```
# remplissage de df_v2
for el in df_hv2.keys():
    df_v2=feature_extract(df_hv2[el],Fs,df,0)
for el in df_CE_v2.keys():
    df_v2=feature_extract(df_CE_v2[el],Fs,df,1)
for el in df_CI_v2.keys():
    df_v2=feature_extract(df_CI_v2[el],Fs,df,2)
```

Entrée [70]:

```
# Creation d'un dataframe pandas vide
df_v3=pd.DataFrame(columns=["max","min","mean","sd","frequence","rms","skewness","ku
```

Entrée [71]:

```
# remplissage de df_v3
for el in df_hv3.keys():
    df_v3=feature_extract(df_hv3[el],Fs,df,0)
for el in df_CE_v3.keys():
    df_v3=feature_extract(df_CE_v3[el],Fs,df,1)
for el in df_CI_v3.keys():
    df_v3=feature_extract(df_CI_v3[el],Fs,df,2)
```

Entrée [72]:

```
# Creation d'un dataframe pandas vide
data=pd.DataFrame(columns=["max","min","mean","sd","frequence","rms","skewness","kur
```

Entrée [73]:

```
# remplissage de data
for el in df_h.keys():
    data=feature_extract(df_h[el],Fs,data,0)
for el in df_CE.keys():
    data=feature_extract(df_CE[el],Fs,data,1)
for el in df_CI.keys():
    data=feature_extract(df_CI[el],Fs,data,2)

for el in df_hv2.keys():
    data=feature_extract(df_hv2[el],Fs,data,0)
for el in df_CE_v2.keys():
    data=feature_extract(df_CE_v2[el],Fs,data,1)
for el in df_CI_v2.keys():
    data=feature_extract(df_CI_v2[el],Fs,data,2)

for el in df_hv3.keys():
    data=feature_extract(df_hv3[el],Fs,data,0)
for el in df_CE_v3.keys():
    data=feature_extract(df_CE_v3[el],Fs,data,1)
for el in df_CI_v3.keys():
    data=feature_extract(df_CI_v3[el],Fs,data,2)
```

Entrée [74]:

```
# Affichage de data
data
```

Out[74]:

	max	min	mean	sd	frequence	rms	skewness	kurtosis	
0	0.000469	-0.000280	0.000113	0.000039	721.538003	0.000120	-0.005562	3.838162	
1	0.000405	-0.000204	0.000112	0.000039	749.342031	0.000118	-0.003809	3.531972	
2	0.000464	-0.000229	0.000107	0.000039	795.562371	0.000114	-0.001383	3.426398	
3	0.000451	-0.000190	0.000106	0.000039	812.867831	0.000113	-0.014720	3.565592	
4	0.000524	-0.000156	0.000105	0.000039	825.512765	0.000112	0.023891	3.972612	
...	...	...	...	...	...	...	...	...	
685	0.013379	-0.013062	0.000096	0.000777	7229.633307	0.000783	-0.073209	25.430421	1
686	0.010664	-0.013971	0.000095	0.000769	7168.555094	0.000775	-0.293040	25.937289	1
687	0.015981	-0.013970	0.000095	0.000759	7233.003003	0.000765	-0.197505	31.205422	2
688	0.014933	-0.016291	0.000100	0.000762	7146.076829	0.000769	-0.175210	29.294700	2
689	0.012021	-0.018345	0.000099	0.000772	7281.171899	0.000779	-0.389410	31.818530	2

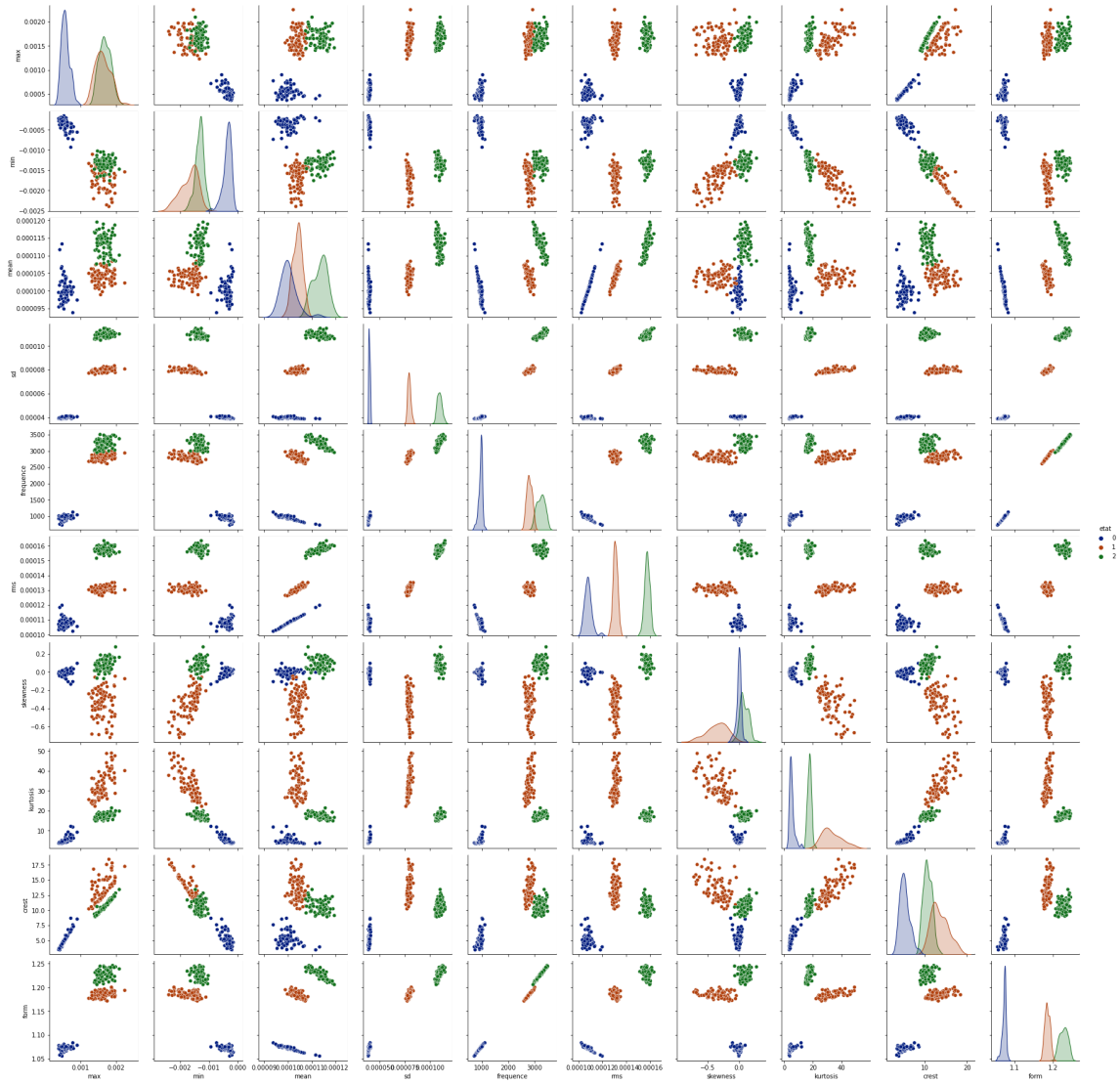
690 rows × 11 columns

Entrée [75]:

```
# Visualisation de df
sns.pairplot(df,hue="etat",palette='dark')
```

Out[75]:

```
<seaborn.axisgrid.PairGrid at 0x1966fc48e80>
```

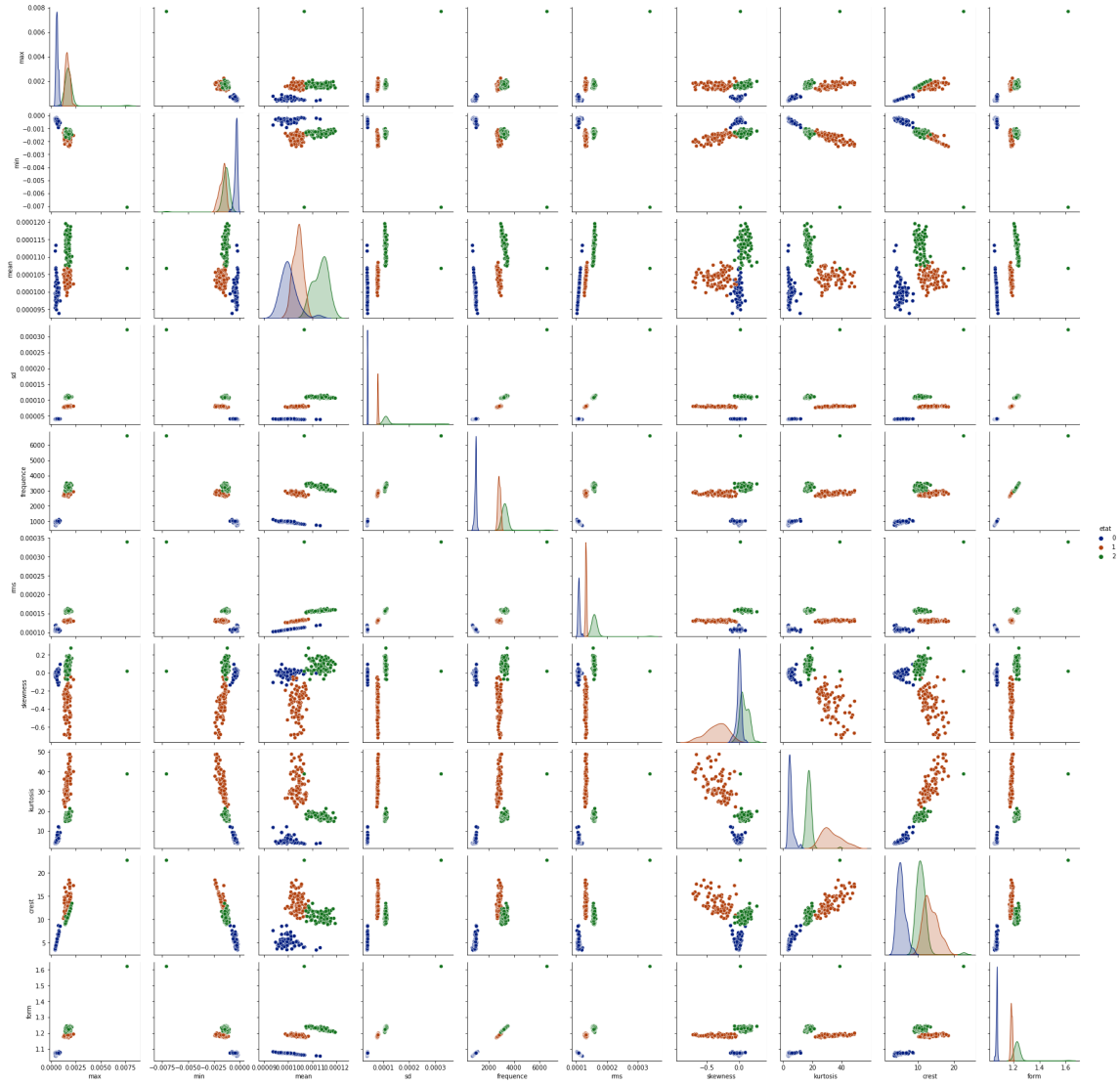


Entrée [76]:

```
# Visualisation de df_v2
sns.pairplot(df_v2,hue="etat",palette='dark')
```

Out[76]:

```
<seaborn.axisgrid.PairGrid at 0x1965007ddc0>
```

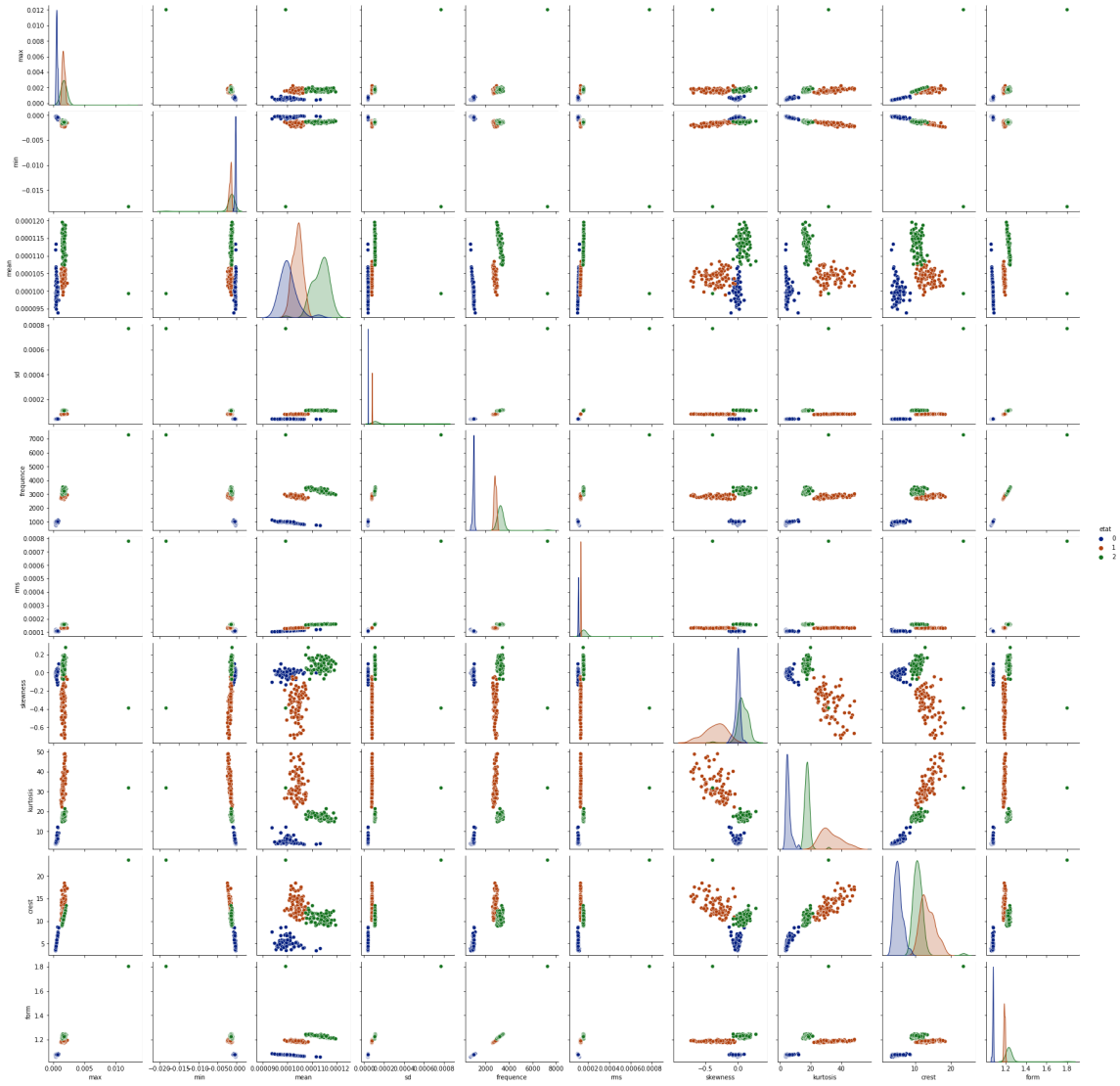


Entrée [77]:

```
# Visualisation de df_v3
sns.pairplot(df_v3,hue="etat",palette='dark')
```

Out[77]:

```
<seaborn.axisgrid.PairGrid at 0x19642fc3610>
```

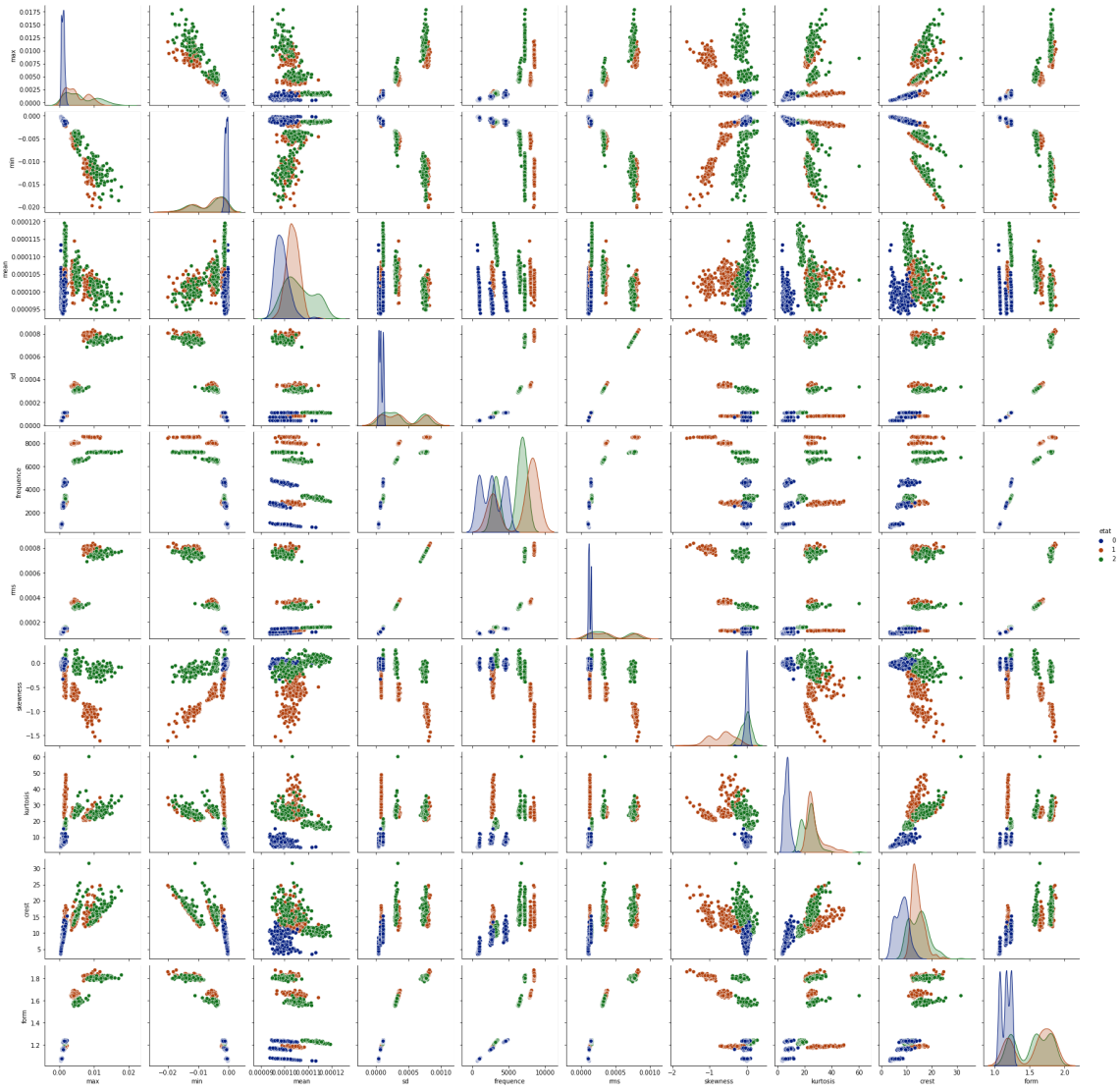


Entrée [78]:

```
# Visualisation de data
sns.pairplot(data,hue="etat",palette='dark')
```

Out[78]:

```
<seaborn.axisgrid.PairGrid at 0x196430493a0>
```



**Séparation des features et du label

Entrée [79]:

```
X=data.drop(['etat'],axis=1)
Y=data['etat']
```

Normalisation des features

Entrée [82]:

```
Nor_MinMax=MinMaxScaler()
X[["max","min","mean","sd","frequence","rms","skewness","kurtosis","crest","form"]]=
```

Division des données en données de test et données d'entrainement

Entrée [83]:

```
Xtrain,Xtest,Ytrain,Ytest=train_test_split(X,Y,test_size=0.25,random_state=10)
```

Les modèles Machine Learning et Deep Learning

1. Arbre de décision

Entrée [86]:

```
dt=DecisionTreeClassifier(max_depth=10)
dt.fit(Xtrain,Ytrain)
ypred_dt=dt.predict(Xtest)
print(classification_report(Ytest,ypred_dt))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        67
           1       1.00      1.00      1.00        51
           2       1.00      1.00      1.00        55

    accuracy                           1.00       173
   macro avg       1.00      1.00      1.00       173
weighted avg       1.00      1.00      1.00       173
```

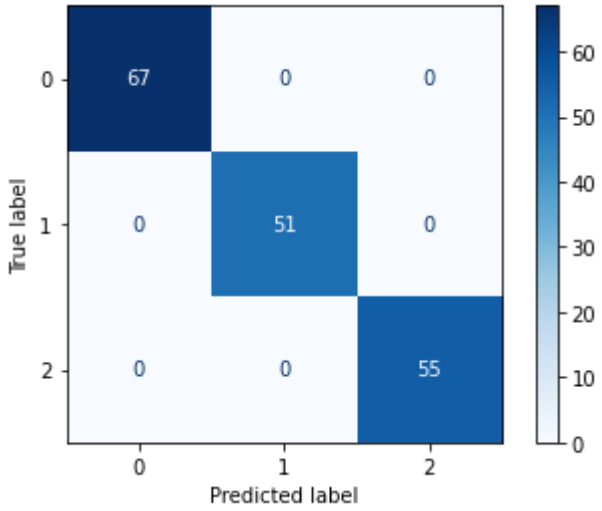
Rien à signaler, le modèle à une performance de 100%

Entrée [87]:

```
plot_confusion_matrix(dt,Xtest,Ytest,cmap=plt.cm.Blues)
```

Out[87]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x196497
67790>
```



2. SVM linéaire

Entrée [88]:

```
lsvm=LinearSVC(random_state=5)
lsvm.fit(Xtrain,Ytrain)
ypred_lsvm=lsvm.predict(Xtest)
print(classification_report(Ytest,ypred_lsvm))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        67
           1       1.00      1.00      1.00        51
           2       1.00      1.00      1.00        55

    accuracy                           1.00       173
   macro avg       1.00      1.00      1.00       173
weighted avg       1.00      1.00      1.00       173
```

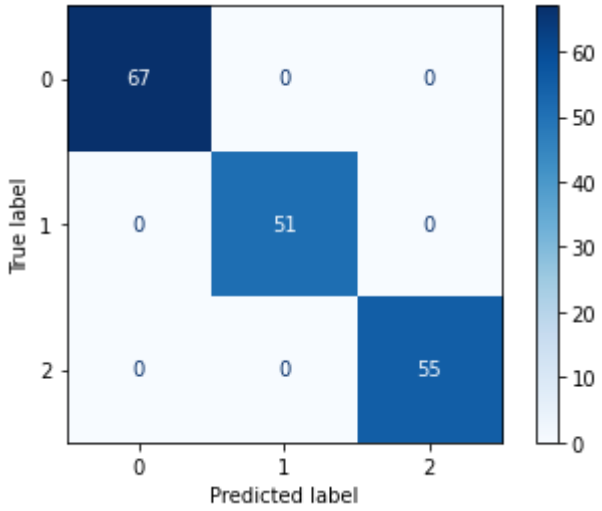
```
*Rien à signaler, le modèle à une performance de 100%*
```

Entrée [90]:

```
plot_confusion_matrix(lsvm,Xtest,Ytest,cmap=plt.cm.Blues)
```

Out[90]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19649b
1b640>
```



3. KNN

Entrée [91]:

```
knn=KNeighborsClassifier(n_neighbors=2)
knn.fit(Xtrain,Ytrain)
ypred_knn=knn.predict(Xtest)
print(classification_report(Ytest,ypred_knn))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        67
           1       1.00      1.00      1.00        51
           2       1.00      1.00      1.00        55

    accuracy                           1.00       173
   macro avg       1.00      1.00      1.00       173
weighted avg       1.00      1.00      1.00       173
```

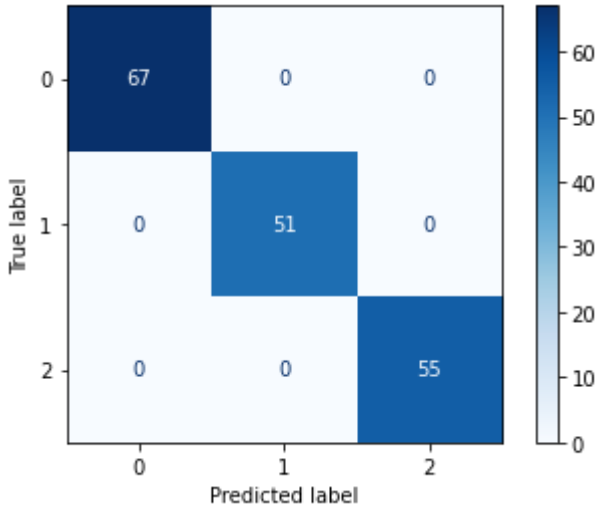
```
*Rien à signaler, le modèle à une performance de 100%*
```

Entrée [93]:

```
1 plot_confusion_matrix(knn,Xtest,Ytest,cmap=plt.cm.Blues)
```

Out[93]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19649c
2fc40>
```



4. Deep learning

Entrée [94]:

```
1 # Création du reseau de neurone
2 MLP=MLPClassifier(hidden_layer_sizes=(50,50,50),activation='relu',max_iter=1000)
3 MLP.out_activation="softmax"
4 # Entrainement du modèle
5 MLP.fit(Xtrain,Ytrain)
```

Out[94]:

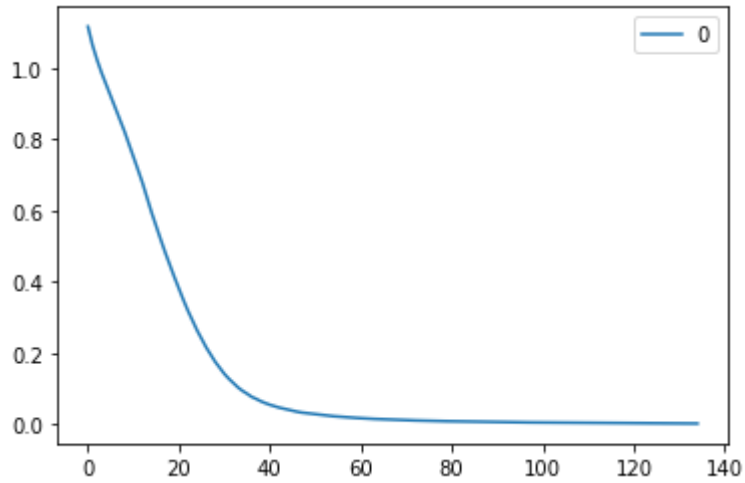
```
MLPClassifier(hidden_layer_sizes=(50, 50, 50), max_iter=1000)
```

Entrée [95]:

```
1 pd.DataFrame(MLP.loss_curve_).plot()
```

Out[95]:

<AxesSubplot:>



Entrée [96]:

```
1 ypred_mod=MLP.predict(Xtest)
2 print(classification_report(Ytest,ypred_mod))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00        67
           1       1.00      1.00      1.00        51
           2       1.00      1.00      1.00        55

    accuracy                           1.00       173
   macro avg       1.00      1.00      1.00       173
weighted avg       1.00      1.00      1.00       173
```

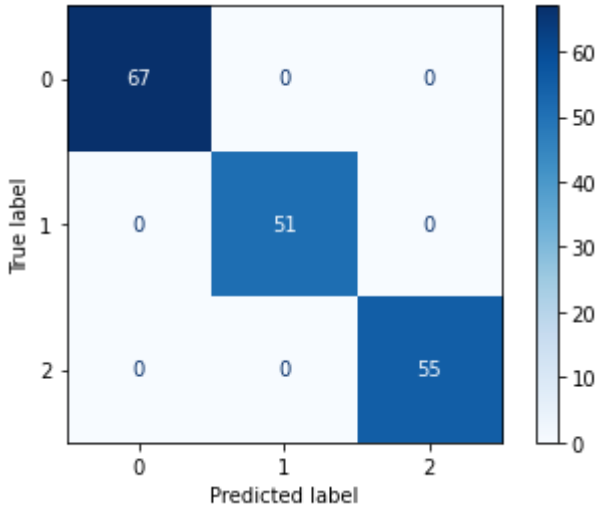
```
1 *Rien à signaler, le modèle à une performance de 100%*
```

Entrée [98]:

```
1 plot_confusion_matrix(MLP,Xtest,Ytest,cmap=plt.cm.Blues)
```

Out[98]:

```
<sklearn.metrics._plot.confusion_matrix.ConfusionMatrixDisplay at 0x19649d
08760>
```




```
1 # Conclusion :
2
```

Entrée [ ]:

```
1
```