Projeto - Aplicação de Redes Neurais Objetivo: O aluno deverá ser capaz de realizar uma análise completa dos dados e projetar uma rede neural para resolver o problema. Avaliar os resultados obtidos através das métricas de classificação Problema: De posse de dados que correspondem a sinais de transitórios de eletrodomésticos (sinais obtidos em uma janela de 2s ao se ligar equipamento) e que foram rotulados em 7 diferentes classes, o aluno deverá realizar os seguintes passos: 1) Carregar os dados e realizar a limpeza dos dados (se necessário) 2) Visualizar os dados para compreensão (dica: plotar 1 exemplo de cada Classe). Como na Figura 1 , abaixo, que representa um eletrodoméstico da Classe 1 3) Como é um problema muticlasse, o aluno deverá transformar os labels para uma representação correta. 4) Preparar os dados para se apresentados à ML 5) Construir a rede neural com seus respectivos parâmetros (taxa de aprendizado, número de camadas intermediárias, número de neurônios, batch_size etc). O aluno deve propor uma estratégia para determinar esses parâmetros. 6) Testar e validar os resultados 7) Avaliar o usa de PCA (Análise de Componentes Principais) para visualização dos dados e também como speed-up da ML (para fins de classificação). 8) Conclusão

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from ydata_profiling import ProfileReport
import matplotlib.pyplot as plt
import seaborn as sns; sns.set()
from sklearn.preprocessing import OneHotEncoder, StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.decomposition import PCA
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from scipy import stats
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.metrics import multilabel_confusion_matrix
```

```
data = "C:/Users/laiss/OneDrive/Arquivos/analista de dados/Redes Neurais/projeto/db.csv"
```

```
df = pd.read_csv(data) # carregando os dados
```

```
df.head()
```

	t0	t1	t2	t3	t4	t5	t6	t7	t8	t9	...	t191	t192	t193	t194	t195	t196	t197	t198	t199	Classe
0	24.00	24.00	23.00	25.00	24.00	25.00	24.00	24.00	22.00	25.00	...	1.00	-1.00	1.00	-1.00	1.00	0.00	1.00	0.00	0.00	
1	23.00	23.00	22.00	21.00	21.00	22.00	23.00	23.00	22.00	21.00	...	-1.00	1.00	0.00	1.00	-1.00	0.00	-1.00	1.00	0.00	
2	-0.55	-0.55	-0.55	3.45	13.45	11.45	18.45	18.45	20.45	20.45	...	-0.55	0.45	-0.55	0.45	-0.55	0.45	-0.55	0.45	-0.55	
3	12.30	10.30	15.30	15.30	16.30	15.30	17.30	16.30	17.30	15.30	...	-0.70	0.30	-0.70	0.30	-0.70	1.30	-0.70	1.30	-0.70	
4	24.85	2.85	5.85	-1.15	2.85	-1.15	1.85	-1.15	0.85	-1.15	...	-0.15	0.85	-1.15	-0.15	-1.15	0.85	-0.15	0.85	-0.15	

5 rows × 201 columns

```
df.shape
```

```
(100, 201)
```

```
df.isnull().sum()#verificando se tem nulos. Não irei dropar duplicata porque pelo problema verifico que o eletrodomestico pode se compor
```

```
t0         0
t1         0
t2         0
t3         0
t4         0
          ..
t196       0
t197       0
t198       0
t199       0
Classes    0
Length: 201, dtype: int64
```

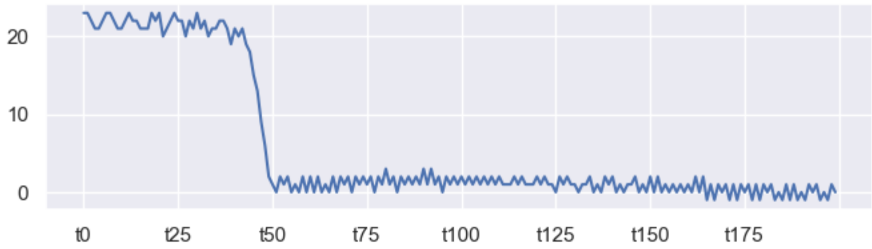
```
labels = df['Classes'].unique() # observando os grupos de classes
```

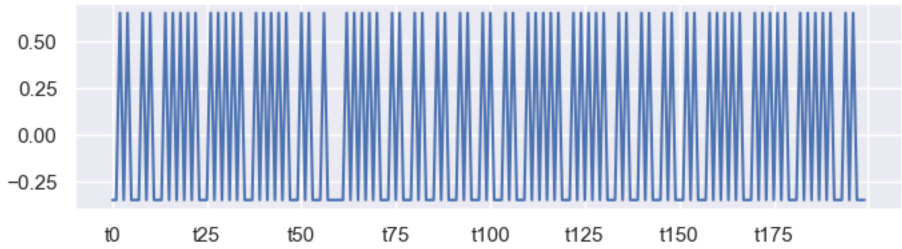
```
labels
```

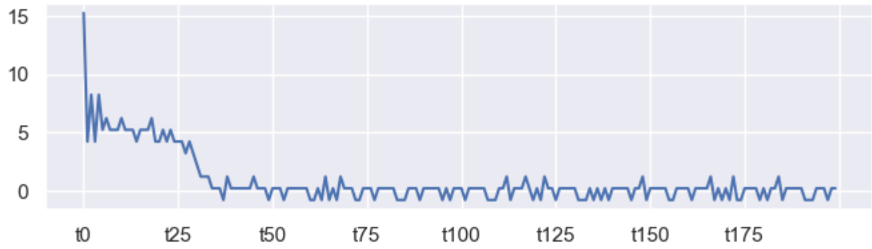
```
array([1, 2, 3, 4, 5, 6, 7], dtype=int64)
```

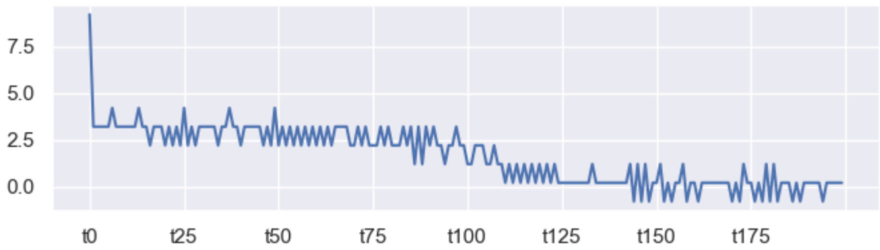
```
def PlotGraf(df,classe,linha):  #código para plotar os gáficos
    df_fig = df[df['Classes'] == classe]
    fig = df_fig.iloc[linha,:-1].plot(figsize = (8, 2))
    plt.show()
```

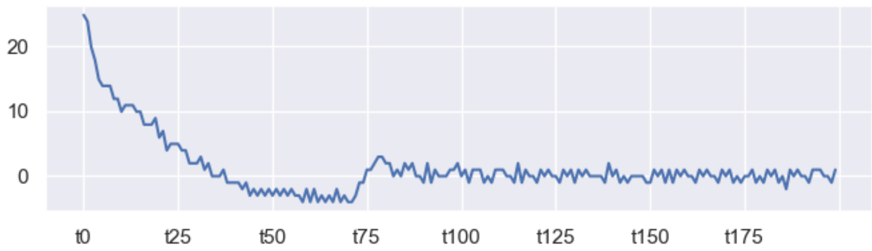
```
for classe in range (1,8):
    PlotGraf(df, classe, 1)
```

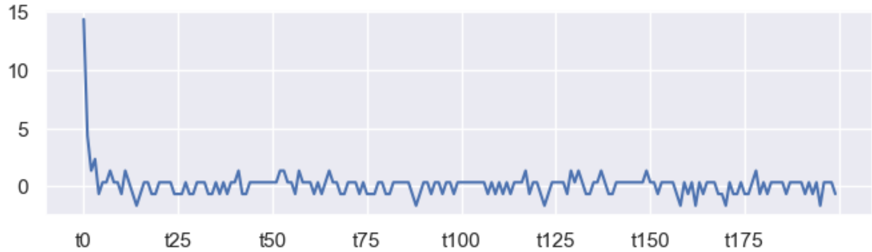


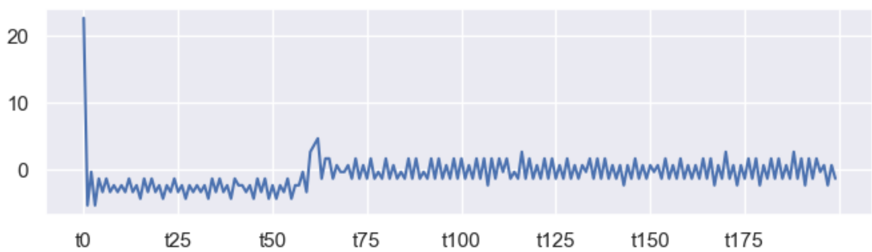













```
labels_1 = df['Classes'].values.reshape(-1, 1)
encoder = OneHotEncoder(sparse_output=False)  # sparse=False para retornar um array
labels_ = encoder.fit_transform(labels_1)
```

```
labels_ #transformou em multiclasses
```

```
array([[1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 1., 0., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 1., 0., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 0., 0., 1.],
       [0., 0., 0., 0., 0., 0., 1.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [1., 0., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0., 0.],
```

```
X = df.drop('Classes', axis=1)
y = labels_
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=20)
```

```
scaler = StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
X_test = scaler.transform(X_test)
```

```
clf = MLPClassifier(random_state=1,hidden_layer_sizes= (20,),learning_rate_init=0.001, max_iter=100,
                    verbose=True).fit(X_train, y_train)
```

```
Iteration 1, loss = 5.07403074
Iteration 2, loss = 4.89271494
Iteration 3, loss = 4.72610018
Iteration 4, loss = 4.57205651
Iteration 5, loss = 4.43120527
Iteration 6, loss = 4.30106028
Iteration 7, loss = 4.18284716
Iteration 8, loss = 4.07320849
Iteration 9, loss = 3.96994410
Iteration 10, loss = 3.87227308
Iteration 11, loss = 3.77949565
Iteration 12, loss = 3.69284025
Iteration 13, loss = 3.61125360
Iteration 14, loss = 3.53394922
```

```
Iteration 15, loss = 3.45953658
Iteration 16, loss = 3.38765512
Iteration 17, loss = 3.31848998
Iteration 18, loss = 3.25204292
Iteration 19, loss = 3.18766788
Iteration 20, loss = 3.12662673
Iteration 21, loss = 3.06869714
Iteration 22, loss = 3.01271854
Iteration 23, loss = 2.95833778
Iteration 24, loss = 2.90578025
Iteration 25, loss = 2.85494645
Iteration 26, loss = 2.80579842
Iteration 27, loss = 2.75803062
Iteration 28, loss = 2.71176072
Iteration 29, loss = 2.66682382
Iteration 30, loss = 2.62287886
Iteration 31, loss = 2.58043517
Iteration 32, loss = 2.53882909
Iteration 33, loss = 2.49818183
Iteration 34, loss = 2.45838545
Iteration 35, loss = 2.41943469
Iteration 36, loss = 2.38140739
Iteration 37, loss = 2.34399006
Iteration 38, loss = 2.30709030
Iteration 39, loss = 2.27065896
Iteration 40, loss = 2.23488004
Iteration 41, loss = 2.19973602
Iteration 42, loss = 2.16461889
Iteration 43, loss = 2.13004281
Iteration 44, loss = 2.09598318
Iteration 45, loss = 2.06234287
Iteration 46, loss = 2.02913181
Iteration 47, loss = 1.99650449
Iteration 48, loss = 1.96457064
Iteration 49, loss = 1.93340371
Iteration 50, loss = 1.90302647
Iteration 51, loss = 1.87330672
Iteration 52, loss = 1.84408230
Iteration 53, loss = 1.81576400
Iteration 54, loss = 1.78818547
Iteration 55, loss = 1.76107452
Iteration 56, loss = 1.73443853
Iteration 57. loss = 1.70830537
```

```
accuracy = clf.score(X_test, y_test)
print(accuracy)
```

```
0.6666666666666666
```

```
#testando no gridsearch
parameters = {
    'hidden_layer_sizes': [(10,), (20,), (30,), (50)],  # Testando diferentes tamanhos de camadas ocultas
    'learning_rate_init': [0.001, 0.01, 0.1],     # Testando diferentes taxas de aprendizagem inicial
    'max_iter': [200, 300, 500]                   # Testando diferentes números máximos de iterações
}
```

```
grid_search = GridSearchCV(clf, parameters, n_jobs=-1, cv=5)
```

```
grid_search.fit(X_train, y_train)
```

```
Iteration 1, loss = 5.07403074
Iteration 2, loss = 3.92832264
Iteration 3, loss = 3.26801233
Iteration 4, loss = 2.75828290
Iteration 5, loss = 2.36110416
Iteration 6, loss = 2.03904842
Iteration 7, loss = 1.77396865
Iteration 8, loss = 1.55294414
Iteration 9, loss = 1.36575174
Iteration 10, loss = 1.20107556
Iteration 11, loss = 1.05878425
Iteration 12, loss = 0.93872903
Iteration 13, loss = 0.83952393
Iteration 14, loss = 0.75065686
Iteration 15, loss = 0.66768138
Iteration 16, loss = 0.59092168
Iteration 17, loss = 0.52443433
Iteration 18, loss = 0.47221059
Iteration 19, loss = 0.42818634
Iteration 20, loss = 0.39048834
Iteration 21, loss = 0.35645209
Iteration 22, loss = 0.32564241
Iteration 23, loss = 0.29749850
Iteration 24, loss = 0.27139104
Iteration 25, loss = 0.24711442
Iteration 26, loss = 0.22445926
Iteration 27, loss = 0.20390234
Iteration 28, loss = 0.18544917
Iteration 29, loss = 0.16921871
Iteration 30, loss = 0.15500418
Iteration 31, loss = 0.14253389
Iteration 32, loss = 0.13136153
Iteration 33, loss = 0.12110361
Iteration 34, loss = 0.11126704
Iteration 35, loss = 0.10180993
Iteration 36, loss = 0.09293755
Iteration 37, loss = 0.08515154
Iteration 38, loss = 0.07829661
Iteration 39, loss = 0.07232468
Iteration 40, loss = 0.06716109
Iteration 41, loss = 0.06260206
Iteration 42, loss = 0.05846062
Iteration 43, loss = 0.05461539
Iteration 44, loss = 0.05105453
Iteration 45, loss = 0.04778848
Iteration 46, loss = 0.04487442
Iteration 47, loss = 0.04228847
Iteration 48, loss = 0.03989200
Iteration 49, loss = 0.03768066
Iteration 50, loss = 0.03566995
Iteration 51, loss = 0.03386105
Iteration 52, loss = 0.03229639
Iteration 53, loss = 0.03090951
Iteration 54, loss = 0.02963864
Iteration 55, loss = 0.02843744
Iteration 56, loss = 0.02728620
Iteration 57, loss = 0.02618013
Iteration 58, loss = 0.02512511
Iteration 59, loss = 0.02412861
Iteration 60, loss = 0.02321382
Iteration 61, loss = 0.02240099
Iteration 62, loss = 0.02168539
Iteration 63, loss = 0.02101525
Iteration 64, loss = 0.02037482
Iteration 65, loss = 0.01975240
Iteration 66, loss = 0.01914676
Iteration 67, loss = 0.01855822
Iteration 68, loss = 0.01799252
Iteration 69, loss = 0.01748249
Iteration 70, loss = 0.01699272
Iteration 71, loss = 0.01653426
Iteration 72, loss = 0.01610994
Iteration 73, loss = 0.01569912
Iteration 74, loss = 0.01530187
Iteration 75, loss = 0.01493808
Iteration 76, loss = 0.01458873
Iteration 77, loss = 0.01424925
Iteration 78, loss = 0.01391938
Iteration 79, loss = 0.01359928
Iteration 80, loss = 0.01329742
Iteration 81, loss = 0.01301042
Iteration 82, loss = 0.01273662
Iteration 83, loss = 0.01247858
Iteration 84, loss = 0.01222593
Iteration 85, loss = 0.01197969
Iteration 86, loss = 0.01173928
Iteration 87, loss = 0.01150524
Iteration 88, loss = 0.01128603
Iteration 89, loss = 0.01107891
Iteration 90, loss = 0.01087859
```

```
Iteration 91, loss = 0.01068513
Iteration 92, loss = 0.01049676
Iteration 93, loss = 0.01031138
Iteration 94, loss = 0.01012949
Iteration 95, loss = 0.00995499
Iteration 96, loss = 0.00978898
Iteration 97, loss = 0.00963000
Iteration 98, loss = 0.00947528
Iteration 99, loss = 0.00932232
Iteration 100, loss = 0.00917326
Iteration 101, loss = 0.00902776
Iteration 102, loss = 0.00888509
Iteration 103, loss = 0.00875077
Iteration 104, loss = 0.00862043
Iteration 105, loss = 0.00849162
Iteration 106, loss = 0.00836445
Iteration 107, loss = 0.00824497
Iteration 108, loss = 0.00812632
Iteration 109, loss = 0.00801013
Iteration 110, loss = 0.00789537
Iteration 111, loss = 0.00778506
Iteration 112, loss = 0.00767815
Iteration 113, loss = 0.00757365
Iteration 114, loss = 0.00747032
Iteration 115, loss = 0.00736880
Iteration 116, loss = 0.00727191
Iteration 117, loss = 0.00717707
Iteration 118, loss = 0.00708399
Iteration 119, loss = 0.00699159
Iteration 120, loss = 0.00690052
Iteration 121, loss = 0.00681420
Iteration 122, loss = 0.00672908
Iteration 123, loss = 0.00664465
Iteration 124, loss = 0.00656204
Iteration 125, loss = 0.00648312
Iteration 126, loss = 0.00640504
Training loss did not improve more than tol=0.000100 for 10 consecutive epochs. Stopping.
```

GridSearchCV

estimator: MLPClassifier

MLPClassifier

```
# Imprimindo os melhores parâmetros encontrados
print("Melhores parâmetros encontrados:")
print(grid_search.best_params_)

# Avaliando o desempenho do modelo com os melhores parâmetros no conjunto de teste
best_clf = grid_search.best_estimator_
accuracy = best_clf.score(X_test, y_test)
print("Acurácia no conjunto de teste com melhores parâmetros:", accuracy)
```

```
Melhores parâmetros encontrados:
{'hidden_layer_sizes': (20,), 'learning_rate_init': 0.01, 'max_iter': 200}
Acurácia no conjunto de teste com melhores parâmetros: 0.8666666666666667
```

Comparando o antigo ML de 0,66 com novo proposto com os parâmetros do GridSearch tivemos um aumento de 0,2 na acurácia.

```
# Avaliando o desempenho do modelo com os melhores parâmetros no conjunto de teste
y_pred = best_clf.predict(X_test) # relatório de classificação para avaliar as outras métricas
print(classification_report(y_test, y_pred, zero_division=1))
```

```
              precision    recall  f1-score   support

           0       1.00      1.00      1.00         7
           1       1.00      0.67      0.80         6
           2       1.00      0.78      0.88         9
           3       1.00      1.00      1.00         0
           4       0.75      1.00      0.86         6
           5       1.00      1.00      1.00         2
           6       1.00      1.00      1.00         0

   micro avg       0.93      0.87      0.90        30
   macro avg       0.96      0.92      0.93        30
weighted avg       0.95      0.87      0.89        30
 samples avg       0.93      0.87      0.87        30
```

Esses resultados servem para avaliar o modelo em várias métricas. No geral o modelo atende bem em várias classes, apenas na classe 2 e na Classe 1 que não consegue corresponder em todas as amostras.

PCA