Projeto MLP: Sistemas Inteligentes

Grupo: Kailane Felix (kefs), Gabriel Lopes (gls6), Camila Xavier (cxm), Lucas Souza (lsb4), Luisa Mendes (mlmsp), Lucas Mota (lmm3)

🎯 Objetivos:

Neste projeto, usaremos um classificador MLP em um conjunto de dados sobre Lombalgia com o objetivo de reconhecer pessoas que possuem a doença através de dados físicos da sua coluna vertebral.

🔁 Conectando o Google Colab ao Google Drive

In [43]:
```
# from google.colab import drive
# drive.mount('/content/drive')
```

📚 Importando as bibliotecas necessárias

In [44]:
```
import pandas as pd
import random
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np
from sklearn.neural_network import MLPClassifier
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from math import ceil
```

In [45]:
```
from matplotlib import rcParams

rcParams['figure.figsize'] = 15, 10
```

🎲 Importando os dados

In [46]:
```
df = pd.read_csv('Dataset_spine.csv')
```

👀 Dando uma olhada na estrutura geral dos dados

In [47]:
```
df.head()
```

Out[47]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	
0	63.027817	22.552586	39.609117	40.475232	98.672917	-0.254400	0.744503	12.5661	14.5386	15.
1	39.056951	10.060991	25.015378	28.995960	114.405425	4.564259	0.415186	12.8874	17.5323	16.
2	68.832021	22.218482	50.092194	46.613539	105.985135	-3.530317	0.474889	26.8343	17.4861	16.

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	
3	69.297008	24.652878	44.311238	44.644130	101.868495	11.211523	0.369345	23.5603	12.7074	11.
4	49.712859	9.652075	28.317406	40.060784	108.168725	7.918501	0.543360	35.4940	15.9546	8.

In [48]:
```
mask = df['Class_att'] == 'Abnormal'
abnormal_df = df[mask]
normal_df = df[~mask]
```

In [49]:
```
abnormal_df.describe()
```

Out[49]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Co
count	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.000000	210.00000
mean	64.692562	19.791111	55.925370	44.901450	115.077713	37.777705	0.483979	21.08587
std	17.662129	10.515871	19.669471	14.515560	14.090605	40.696741	0.286319	8.55036
min	26.147921	-6.554948	14.000000	13.366931	70.082575	-10.675871	0.003220	7.02700
25%	50.102507	13.048130	41.116964	34.380345	107.309280	7.263227	0.250212	13.05440
50%	65.274888	18.798899	56.150000	44.639597	115.650323	31.946516	0.501280	21.75060
75%	77.593672	24.815515	68.102805	55.146868	123.133365	55.371614	0.708476	28.19497
max	129.834041	49.431864	125.742385	121.429566	163.071041	418.543082	0.998827	36.74390

In [50]:
```
normal_df.describe()
```

Out[50]:

	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Co
count	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.000000	100.00000
mean	51.685244	12.821414	43.542605	38.863830	123.890834	2.186572	0.449880	21.81639
std	12.368161	6.778503	12.361388	9.624004	9.014246	6.307483	0.284711	8.84648
min	30.741938	-5.845994	19.071075	17.386972	100.501192	-11.058179	0.005045	7.39070
25%	42.817849	8.799951	35.000000	32.340487	118.182659	-1.511360	0.201843	13.16422
50%	50.123115	13.482435	42.638923	37.059694	123.874328	1.152710	0.418732	22.41460
75%	61.470097	16.785953	51.602346	44.608788	129.040401	4.968807	0.695981	29.76062
max	89.834676	29.894119	90.563461	67.195460	147.894637	31.172767	0.997247	36.61940

Dicionário de dados:

- Col1: incidência pélvica
- Col2: inclinação pélvica
- Col3: ângulo lombar lordosis
- Col4: inclinação sacral

- Col5: raio pélvico
- Col6: grau espondilolistese
- Col7: inclinação pélvica
- Col8: Inclinação direta
- Col9: inclinação torácica
- Col10: inclinação cervical
- Col11: ângulo do sacro
- Col12: inclinação escoliose
- Class_att = anormal, normal

🗂️ Definindo a matriz de atributos e o vetor target:

In [51]:
```
X = df.drop(columns = ['Class_att', 'Unnamed: 13'])
y = df['Class_att']
```

✂️ Separando nosso dataset em um set de treino e um set de teste

In [52]:
```
X_train, X_test, y_train, y_test = train_test_split(X, y, random_state=42)
```

🏋️ Treinando...

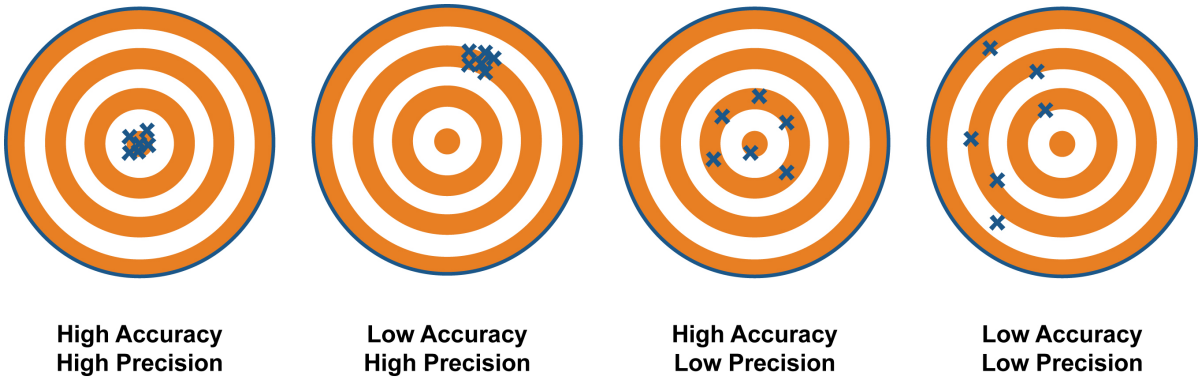
In [53]:
```
clf = MLPClassifier().fit(X_train, y_train)
```

🧐 Analisando o Classification Report do modelo

In [54]:
```
y_pred = clf.predict(X_test)
```

In [55]:
```
class_report = classification_report(y_test, y_pred)
print(class_report)
```

```
              precision    recall  f1-score   support

    Abnormal       0.84      0.86      0.85        57
      Normal       0.60      0.57      0.59        21

    accuracy                           0.78        78
   macro avg       0.72      0.72      0.72        78
weighted avg       0.78      0.78      0.78        78
```



- A precisão mede quantas das previsões positivas do modelo são realmente corretas

- A acurácia é uma medida geral de avaliação que mede a proporção de previsões corretas
- O F1-Score é uma medida de avaliação que considera tanto a precisão quanto o recall.
- O recall mede a capacidade do modelo em detectar todos os exemplos positivos

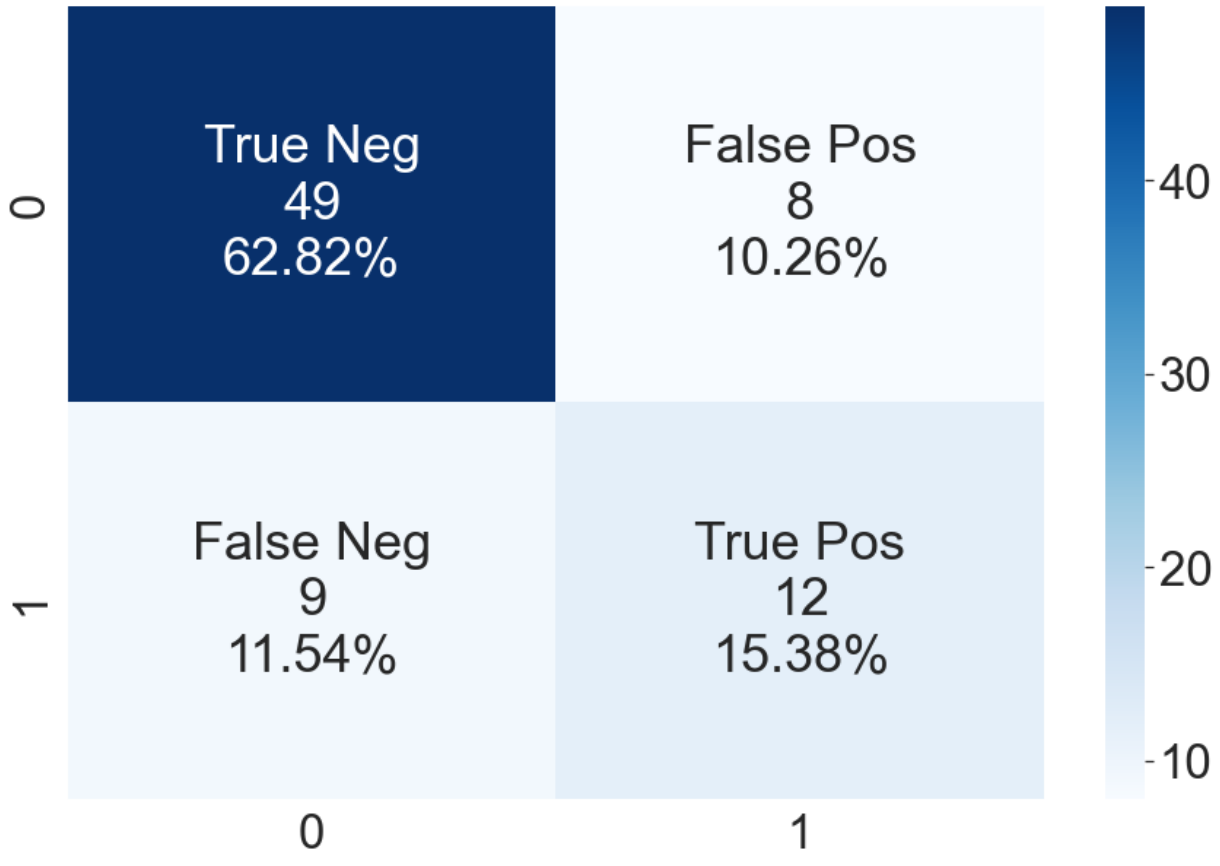
🧐 Analisando a matriz de confusão do modelo

In [56]:
```
cf_matrix_nb = confusion_matrix(y_test, y_pred)
print(cf_matrix_nb)
```

```
[[49  8]
 [ 9 12]]
```

In [57]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix_nb.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix_nb.flatten() / n
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group
labels = np.asarray(labels).reshape(2, 2)

sns.set(font_scale= 3)
sns.heatmap(cf_matrix_nb, annot=labels, fmt= '', cmap='Blues')
```

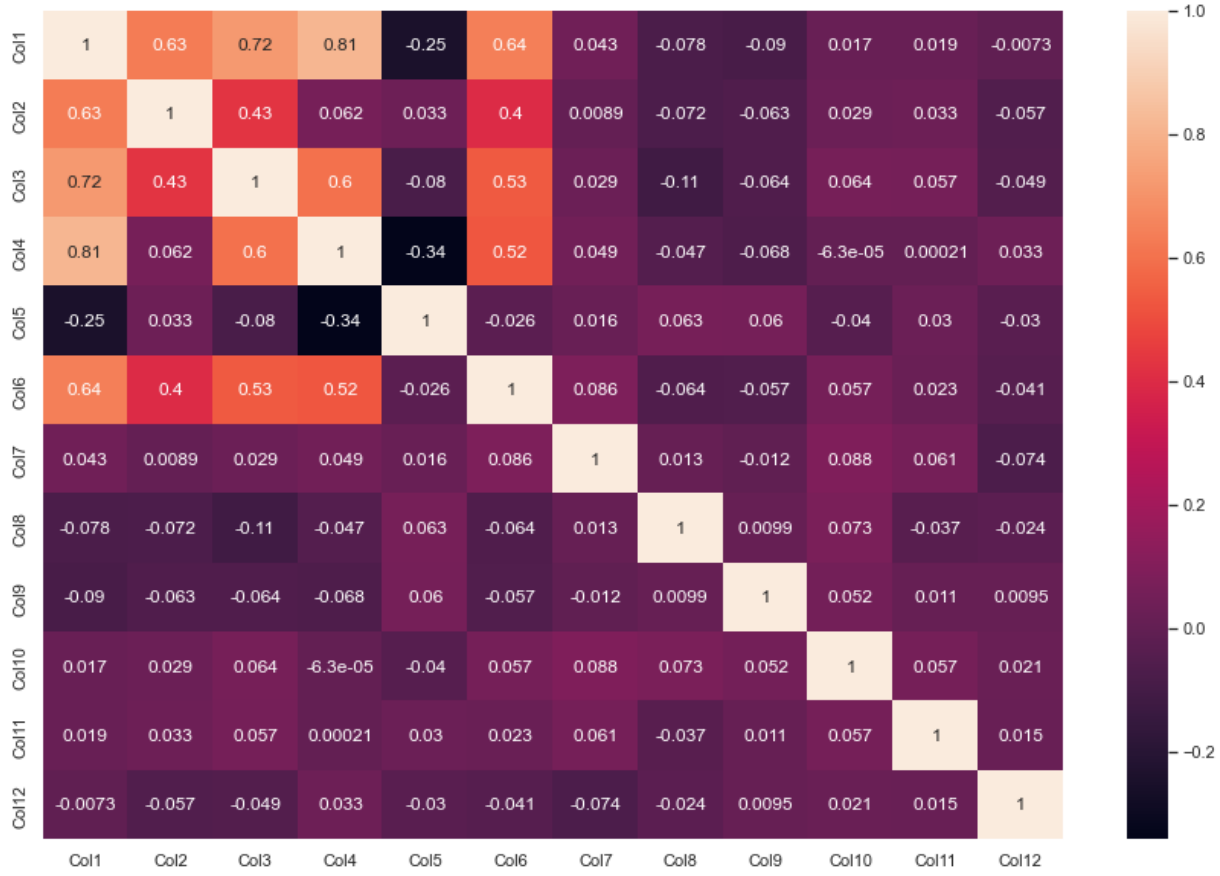
Out[57]: <AxesSubplot:>



📊 Análise exploratória

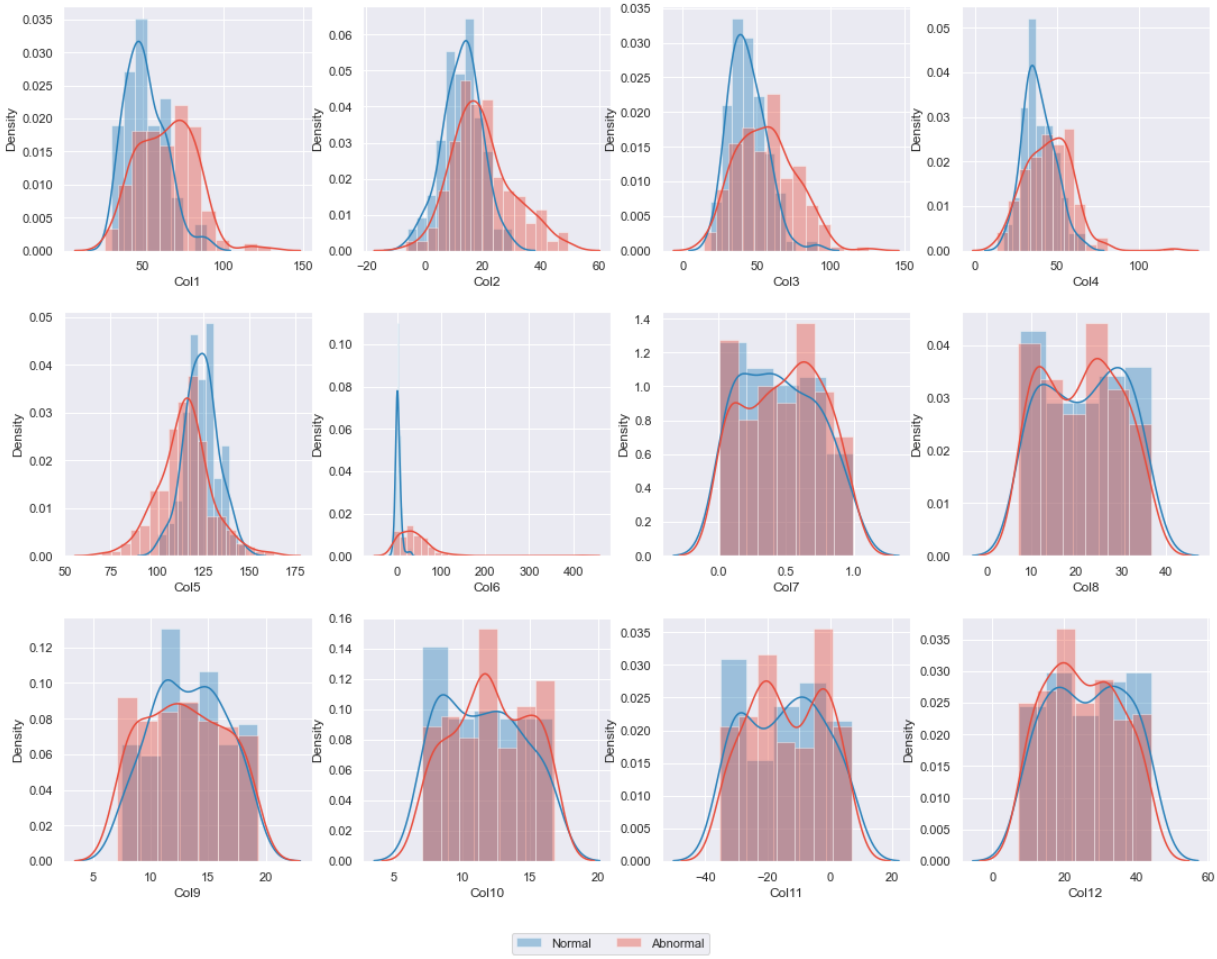
In [58]:
```
sns.set(font_scale= 1)
sns.heatmap(data = df.corr(), annot=True)
```

Out[58]: <AxesSubplot:>



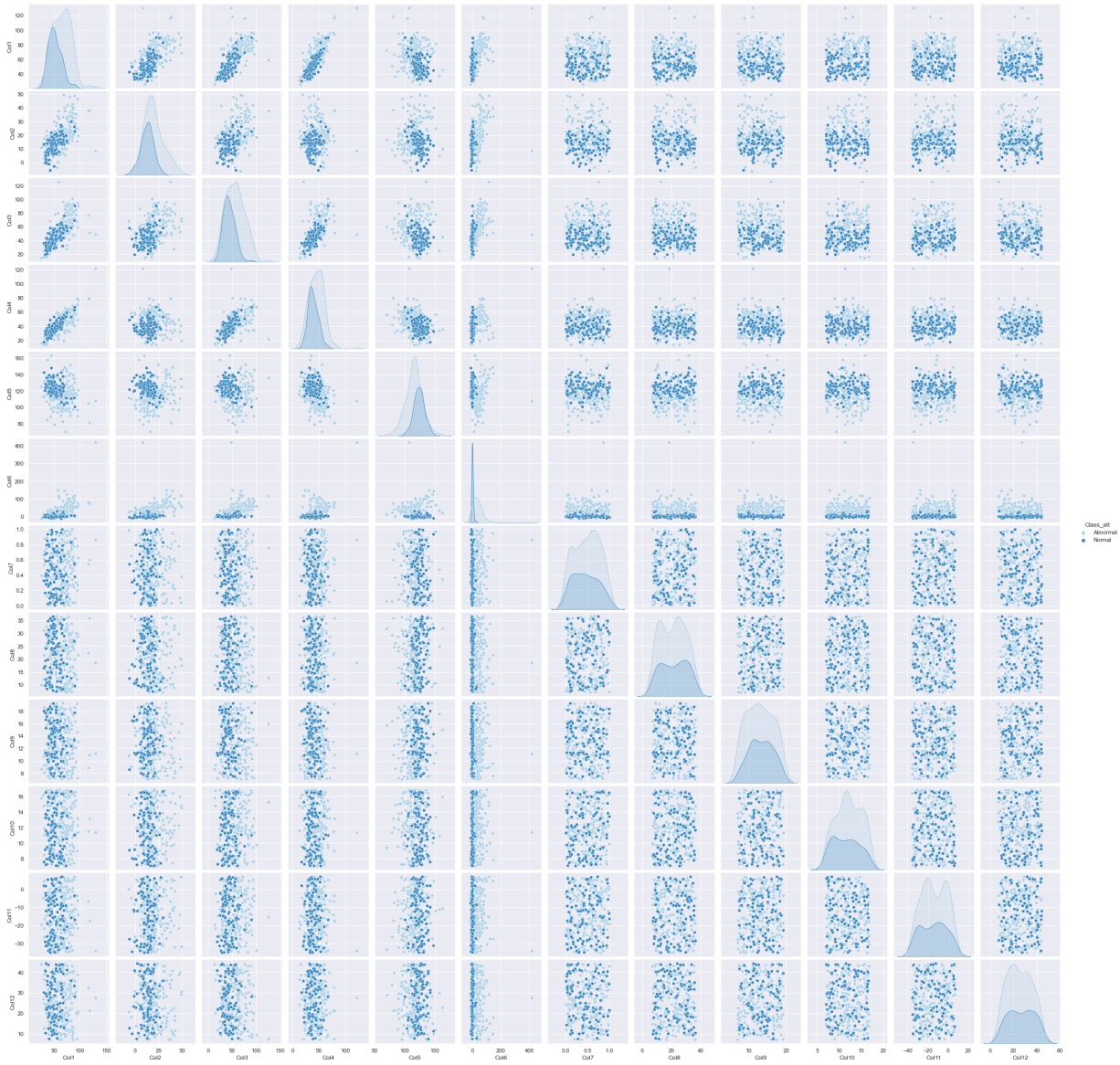
In [59]:

```
import warnings
warnings.filterwarnings('ignore')

fg, ax = plt.subplots(ncols=4, nrows=3, sharex=False, sharey=False, figsize=(15,12))
fg.tight_layout()

plt.subplots_adjust(hspace=0.25, bottom = 0.1)

for index, axes in enumerate(ax.reshape(-1)):
    normal = sns.distplot(df[df.Class_att == 'Normal'].iloc[:, index], ax=axes, colo
    abnormal = sns.distplot(df[df.Class_att == 'Abnormal'].iloc[:, index],ax=axes, c

h, l = axes.get_legend_handles_labels()
fg.legend(h, l, loc='lower center', ncol = 2)
plt.show()
```



In [60]:
```
sns.pairplot(df, hue='Class_att', palette='Blues')
```

Out[60]:
```
<seaborn.axisgrid.PairGrid at 0x1bdde8987f0>
```



Podemos observar que talvez pudessemos retirar algumas features com a intenção de melhorar o modelo. Isso porque variáveis podem ter alta correlação entre si (resultando em problemas de multicolinearidade), ou podem não ser explicativas o suficiente para a classificação.

Uma alternativa é utilizar um algoritmo de feature selection, mas iremos dar foco ao melhoramento através de otimização de hiperparâmetros.

🔋 Melhorando o modelo: fine tuning

Existem inúmeros hiperparâmetros que podem ser ajustados em modelos, e encontrar a combinação ideal deles pode ser considerado um desafio de busca. Atualmente, há diversos algoritmos de otimização disponíveis para essa tarefa, mas uma técnica simples é a GridSearch!

Para ajustar os hiperparâmetros no scikit-learn, pode-se utilizar o método GridSearchCV, que encontra de maneira eficiente os valores ideais dos hiperparâmetros dentre os fornecidos.

In [61]:

```
parameter_space = {
    'hidden_layer_sizes': [(10, 30, 10),(20,), (100, 100, 100), (100,), (30, 40, 20)
    'activation': ['identity', 'logistic', 'tanh', 'relu'],
    'solver': ['sgd', 'adam','lbfgs'],
    'alpha': [0.0001, 0.05],
    'learning_rate': ['constant','adaptive', 'invscaling'],
}
```

```
from sklearn.model_selection import GridSearchCV
```

- hidden_layer_sizes: define o número de camadas e o número de nós que desejamos ter no classificador de rede neural. Assim, o comprimento da tupla denota o número total de camadas ocultas na rede.
- activation: indica função de ativação para as camadas ocultas.
- solver: especifica o algoritmo para otimização de peso entre os nós.
- alpha: termo de regularização, ou termo de penalidade, que combate o overfitting restringindo o tamanho dos pesos
- learning_rate: taxa de aprendizado para atualizações de peso

In [62]:
```
clf_tuned = GridSearchCV(clf, parameter_space, n_jobs=-1, cv=5)
clf_tuned.fit(X, y)
```

Out[62]:
```
GridSearchCV(cv=5, estimator=MLPClassifier(), n_jobs=-1,
             param_grid={'activation': ['identity', 'logistic', 'tanh', 'relu'],
                         'alpha': [0.0001, 0.05],
                         'hidden_layer_sizes': [(10, 30, 10), (20,),
                                                (100, 100, 100), (100,),
                                                (30, 40, 20), (50, 80, 100)],
                         'learning_rate': ['constant', 'adaptive',
                                           'invscaling'],
                         'solver': ['sgd', 'adam', 'lbfgs']})
```

🔎 Observando a melhor combinação de parâmetros encontrada

In [63]:
```
print('Best parameters found:\n', clf_tuned.best_params_)
```

```
Best parameters found:
 {'activation': 'identity', 'alpha': 0.0001, 'hidden_layer_sizes': (10, 30, 10), 'le
arning_rate': 'constant', 'solver': 'lbfgs'}
```

In [70]:
```
clf_tuned = MLPClassifier(hidden_layer_sizes=(10, 30, 10),
                          random_state = 21,
                          activation='identity',
                          alpha=0.0001,
                          solver= 'lbfgs',
                          early_stopping=True,
                          learning_rate= 'constant')
```

🏋️ Treinando...

In [71]:
```
clf_tuned = clf_tuned.fit(X_train, y_train)
```

In [72]:
```
y_pred_tuned = clf_tuned.predict(X_test)
```

🧐 Analisando o Classification Report do modelo

In [73]:
```
class_report = classification_report(y_test, y_pred_tuned)
print(class_report)
```

```
              precision    recall  f1-score   support

    Abnormal       0.88      0.93      0.91        57
```

```
              Normal       0.78      0.67      0.72        21

            accuracy                           0.86        78
           macro avg       0.83      0.80      0.81        78
        weighted avg       0.85      0.86      0.86        78
```

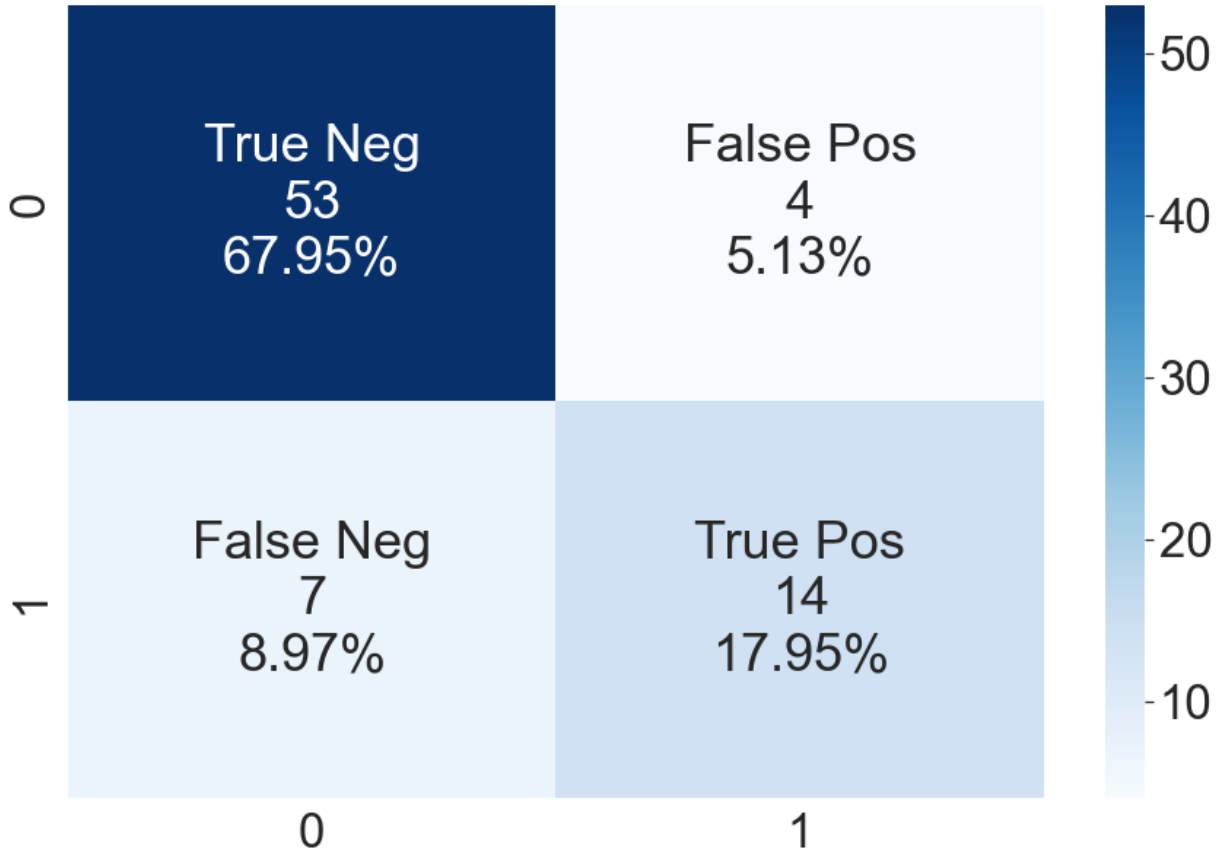
🧐 Analisando a matriz de confusão do modelo

In [74]:
```
cf_matrix_nb_tuned = confusion_matrix(y_test, y_pred_tuned)
print(cf_matrix_nb_tuned)
```

```
[[53  4]
 [ 7 14]]
```

In [75]:
```
group_names = ['True Neg','False Pos','False Neg','True Pos']
group_counts = ["{0:0.0f}".format(value) for value in cf_matrix_nb_tuned.flatten()]
group_percentages = ["{0:.2%}".format(value) for value in cf_matrix_nb_tuned.flatten
labels = [f"{v1}\n{v2}\n{v3}" for v1, v2, v3 in zip(group_names, group_counts, group
labels = np.asarray(labels).reshape(2, 2)

sns.set(font_scale= 3)
sns.heatmap(cf_matrix_nb_tuned, annot=labels, fmt= '', cmap='Blues')
```

Out[75]: <AxesSubplot:>



Podemos observar uma melhora nas métricas do modelo, como um aumento nos verdadeiros negativos e uma diminuição dos falsos positivos. Algumas otimizações ainda podem ser feitas, como tratamento dos dados e feature selection e cross validation.

In [ ]:
```
```