

✓ Disciplina de Inteligência Computacional em Saúde

Avaliação da saúde docente com base em um dataset de pesquisa sobre problemas osteomusculares

O dataset deste trabalho foi importado de uma planilha de dados do questionário nórdico com o seguinte formato:

Detalhamento das Colunas de Dados

#	Coluna	Contagem Não-Nula	Tipo de Dado
0	Cidade	218	object
1	Sexo	218	object
2	Idade	218	object
3	Estado Civil	218	object
4	Filhos	218	object
5	Escolaridade	218	object
6	Tempo de atuação	218	object
7	Vínculo de trabalho	218	object
8	Carga horária semanal	218	object
9	Turnos de trabalho semanal	218	object
10	Número de alunos por turma	218	object
11	Pluriemprego	218	object
12	Possui Sintomas Osteomusculares?	218	object

Transformação dos dados:

em seguida os dados foram transformados para dados numéricos, com os dados numéricos foi executado o algoritmo One-Hot Encoding para converter as variáveis em um formato numérico binário

(https://github.com/paaatcha/MetaBlock/blob/main/benchmarks/pad/preprocess/prepare_data.py)

Tabela com os campos transformados com a execução do One-Hot Encoding

#	Column	Non-Null Count	Dtype
0	Lebon Regis	218 non-null	int64
1	Paulo Lopes	218 non-null	int64
2	São Ludjero	218 non-null	int64
3	Videira	218 non-null	int64
4	Feminino	218 non-null	int64
5	Masculino	218 non-null	int64
6	Entre 31 e 40 anos	218 non-null	int64
7	Acima de 40 anos	218 non-null	int64
8	Até 30 anos	218 non-null	int64
9	estado_civil	218 non-null	int64
10	filhos	218 non-null	int64
11	Especialização	218 non-null	int64
12	Graduação	218 non-null	int64
13	Mestrado/Doutorado	218 non-null	int64
14	Até 10 anos	218 non-null	int64
15	Acima de 10 anos	218 non-null	int64
16	Efetivo	218 non-null	int64
17	ACT	218 non-null	int64
18	Até 20 horas	218 non-null	int64
19	De 21 a 40 horas	218 non-null	int64
20	Acima de 40 horas	218 non-null	int64
21	2 turnos	218 non-null	int64
22	1 turnos	218 non-null	int64
23	3 turnos	218 non-null	int64
24	Até 30 alunos	218 non-null	int64
25	Acima de 30 alunos	218 non-null	int64
26	pluriemprego	218 non-null	int64
27	sintomas_osteomusculares	218 non-null	int64

Observação: Todos os 28 campos são do tipo `int64` e têm 218 entradas não nulas (nenhum dado faltante). A tabela acima representa a estrutura do DataFrame (dtypes, contagem de não nulos).

```
import pandas as pd
data = pd.read_csv('nordico.csv', sep=';')
data.head()
```

	Cidade	Sexo	Idade	Estado Civil	Filhos	Escolaridade	Tempo de atuação	Vínculo de trabalho	Carga horária semanal	Turnos de trabalho semanal	Número de alunos por turma	Pluriemprego
0	Lebon Regis	Feminino	Entre 31 e 40 anos	Com companheiro	Sim	Especialização	Até 10 anos	ACT	Até 20 horas	2 turnos	Até 30 alunos	
1	Lebon Regis	Feminino	Entre 31 e 40 anos	Com companheiro	Sim	Graduação	Até 10 anos	ACT	Até 20 horas	1 turno	Até 30 alunos	
2	Lebon Regis	Feminino	Acima de 40 anos	Com companheiro	Sim	Graduação	Até 10 anos	ACT	Até 20 horas	2 turnos	Até 30 alunos	
3	Lebon Regis	Masculino	Acima de 40 anos	Com companheiro	Sim	Especialização	Acima de 10 anos	ACT	De 21 a 40 horas	2 turnos	Até 30 alunos	
4	Lebon Regis	Feminino	Até 30 anos	Com companheiro	Não	Especialização	Até 10 anos	ACT	De 21 a 40 horas	2 turnos	Até 30 alunos	

Próximas etapas: [Gerar código com data](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

```
import pandas as pd
data = pd.read_csv('nordico_num.csv', sep=';')
data.head()
```

	cidade	sexo	idade	estado_civil	filhos	escolaridade	tempo_atuacao	vinculo	carga_horaria	turnos	alunos_tu
0	1	1	2	1	1	2	1	2	1	2	
1	1	1	2	1	1	1	1	2	1	1	
2	1	1	3	1	1	1	1	2	1	2	
3	1	2	3	1	1	2	2	2	2	2	
4	1	1	1	1	0	2	1	2	2	2	

Próximas etapas: [Gerar código com data](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

Clique duas vezes (ou pressione "Enter") para editar

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 218 entries, 0 to 217
Data columns (total 13 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   cidade                                218 non-null    int64
1   sexo                                  218 non-null    int64
2   idade                                218 non-null    int64
3   estado_civil                          218 non-null    int64
4   filhos                                218 non-null    int64
5   escolaridade                          218 non-null    int64
6   tempo_atuacao                         218 non-null    int64
7   vinculo                               218 non-null    int64
8   carga_horaria                         218 non-null    int64
9   turnos                                218 non-null    int64
10  alunos_turma                          218 non-null    int64
11  pluriemprego                          218 non-null    int64
12  sintomas_osteomusculares             218 non-null    int64
dtypes: int64(13)
memory usage: 22.3 KB
```

```

data_nordico_num = pd.read_csv('nordico_num.csv', sep=';').fillna("EMPTY")

clin_feats = ["cidade", "sexo", "idade", "estado_civil", "filhos", "escolaridade",
              "tempo_atuacao", "vinculo", "carga_horaria", "turnos",
              "alunos_turma", "pluriemprego", "sintomas_osteomusculares"]

new_cli_cols = list()
for c in clin_feats:
    if c in ["estado_civil", "filhos", "pluriemprego", "sintomas_osteomusculares"]:
        new_cli_cols += [c]
        continue
    vals = [c+"_"+str(v) for v in data_nordico_num[c].unique()]
    try:
        vals.remove(c+"_EMPTY")
    except:
        pass
    new_cli_cols += vals
new_df = {c: list() for c in new_cli_cols}

for idx, row in data_nordico_num.iterrows():
    _aux = list()
    _aux_in = list()
    for col in clin_feats:
        data_row = row[col]

        if data_row == 'EMPTY':
            pass
        elif col in ["estado_civil", "filhos", "pluriemprego", "sintomas_osteomusculares"]:
            _aux_in.append(col)
            new_df[col].append(data_row)
            continue
        else:
            _aux.append(col+"_"+str(data_row))

    for x in new_df:
        if x in _aux:
            new_df[x].append(1)
        elif x not in _aux_in:
            new_df[x].append(0)

new_df = pd.DataFrame.from_dict(new_df)

#new_df

# Salvar o DataFrame em um arquivo CSV
new_df.to_csv('nv_saude_docente.csv')

# Identificação das colunas

df_one_hot = new_df.rename(columns={'cidade_1': 'Lebon Regis',
                                   'cidade_2': 'Paulo Lopes',
                                   'cidade_3': 'São Lujero',
                                   'cidade_4': 'Videira',
                                   'sexo_1': 'Feminino',
                                   'sexo_2': 'Masculino',
                                   'idade_1': 'Até 30 anos',
                                   'idade_2': 'Entre 31 e 40 anos',
                                   'idade_3': 'Acima de 40 anos',
                                   'carga_horaria_1': 'Até 20 horas',
                                   'carga_horaria_2': 'De 21 a 40 horas',
                                   'carga_horaria_3': 'Acima de 40 horas',
                                   'turnos_1': '1 turnos',
                                   'turnos_2': '2 turnos',
                                   'turnos_3': '3 turnos',
                                   'alunos_turma_1': 'Até 30 alunos',
                                   'alunos_turma_2': 'Acima de 30 alunos',
                                   'escolaridade_1': 'Graduação',
                                   'escolaridade_2': 'Especialização',
                                   'escolaridade_3': 'Mestrado/Doutorado',
                                   'tempo_atuacao_1': 'Até 10 anos',
                                   'tempo_atuacao_2': 'Acima de 10 anos',
                                   'vinculo_1': 'ACT',
                                   'vinculo_2': 'Efetivo'})

df_one_hot.head(30)

```



	Lebon Regis	Paulo Lopes	São Ludjero	Videira	Feminino	Masculino	Entre 31 e 40 anos	Acima de 40 anos	Até 30 anos	estado_civil	...	Até 20 horas	De 21 a 40 horas	Acima de 40 horas
0	1	0	0	0	1	0	1	0	0	1	...	1	0	0
1	1	0	0	0	1	0	1	0	0	1	...	1	0	0
2	1	0	0	0	1	0	0	1	0	1	...	1	0	0
3	1	0	0	0	0	1	0	1	0	1	...	0	1	0
4	1	0	0	0	1	0	0	0	1	1	...	0	1	0
5	1	0	0	0	1	0	0	1	0	0	...	0	1	0
6	1	0	0	0	1	0	0	1	0	1	...	0	1	0
7	1	0	0	0	1	0	1	0	0	1	...	0	0	1
8	1	0	0	0	0	1	0	0	1	1	...	0	0	1
9	1	0	0	0	1	0	0	1	0	1	...	0	1	0
10	1	0	0	0	1	0	0	1	0	0	...	0	1	0
11	1	0	0	0	1	0	0	1	0	0	...	0	1	0
12	1	0	0	0	1	0	0	1	0	0	...	0	1	0
13	1	0	0	0	1	0	0	1	0	0	...	0	0	1
14	1	0	0	0	1	0	0	1	0	0	...	0	0	1
15	1	0	0	0	1	0	1	0	0	1	...	0	0	1
16	1	0	0	0	1	0	0	0	1	0	...	0	1	0
17	1	0	0	0	1	0	1	0	0	1	...	0	1	0
18	1	0	0	0	1	0	0	1	0	1	...	0	1	0
19	1	0	0	0	1	0	0	1	0	0	...	0	1	0
20	1	0	0	0	1	0	0	1	0	0	...	0	1	0
21	1	0	0	0	1	0	0	1	0	0	...	0	1	0
22	1	0	0	0	1	0	0	1	0	0	...	0	1	0
23	1	0	0	0	1	0	1	0	0	1	...	0	0	1
24	1	0	0	0	1	0	0	1	0	0	...	1	0	0
25	1	0	0	0	1	0	0	1	0	1	...	1	0	0
26	1	0	0	0	1	0	0	1	0	1	...	1	0	0
27	1	0	0	0	1	0	0	0	1	0	...	0	1	0
28	1	0	0	0	0	1	0	0	1	1	...	0	1	0
29	1	0	0	0	0	1	0	0	1	1	...	0	1	0

30 rows x 28 columns

df_one_hot.info()



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 218 entries, 0 to 217
Data columns (total 28 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Lebon Regis                          218 non-null   int64
1   Paulo Lopes                          218 non-null   int64
2   São Ludjero                          218 non-null   int64
3   Videira                             218 non-null   int64
4   Feminino                            218 non-null   int64
5   Masculino                           218 non-null   int64
6   Entre 31 e 40 anos                  218 non-null   int64
7   Acima de 40 anos                    218 non-null   int64
8   Até 30 anos                         218 non-null   int64
9   estado_civil                        218 non-null   int64
10  filhos                              218 non-null   int64
11  Especialização                      218 non-null   int64
12  Graduação                           218 non-null   int64
13  Mestrado/Doutorado                  218 non-null   int64
```

```

14 Até 10 anos                218 non-null    int64
15 Acima de 10 anos          218 non-null    int64
16 Efetivo                   218 non-null    int64
17 ACT                       218 non-null    int64
18 Até 20 horas              218 non-null    int64
19 De 21 a 40 horas          218 non-null    int64
20 Acima de 40 horas         218 non-null    int64
21 2 turnos                  218 non-null    int64
22 1 turnos                  218 non-null    int64
23 3 turnos                  218 non-null    int64
24 Até 30 alunos             218 non-null    int64
25 Acima de 30 alunos        218 non-null    int64
26 pluriemprego              218 non-null    int64
27 sintomas_osteomusculares  218 non-null    int64
dtypes: int64(28)
memory usage: 47.8 KB

```

```

# Importando as bibliotecas necessárias
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.neural_network import MLPClassifier # Importando o MLPClassifier
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import KFold
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import numpy as np

```

```

# Carregando os dados
data = df_one_hot
for column in data.select_dtypes(include=['object']).columns:
    label_encoder = LabelEncoder()
    data[column] = label_encoder.fit_transform(data[column])
# Separando as features (X) e o target (y)

X = data.drop('sintomas_osteomusculares', axis=1)
y = data['sintomas_osteomusculares']

data.head()

```

↕

	Lebon Regis	Paulo Lopes	São Ludjero	Videira	Feminino	Masculino	Entre 31 e 40 anos	Acima de 40 anos	Até 30 anos	estado_civil	...	Até 20 horas	De 21 a 40 horas	Acima de 40 horas	t
0	1	0	0	0	1	0	1	0	0	1	...	1	0	0	
1	1	0	0	0	1	0	1	0	0	1	...	1	0	0	
2	1	0	0	0	1	0	0	1	0	1	...	1	0	0	
3	1	0	0	0	0	1	0	1	0	1	...	0	1	0	
4	1	0	0	0	1	0	0	0	1	1	...	0	1	0	

5 rows x 28 columns

1. **Decision Tree:** min_samples_leaf = 5 melhorou o resultado, o a definição de um min_samples_leaf pode prevenir overfitting, e a definição do random_state assegura a reprodutividade do modelo.

DecisionTreeClassifier(random_state=2025, max_depth=10, min_samples_leaf=5)

2. **SVM.SVC:** as alterações nos parâmetros default não surtiram efeito, foram mantidos os valores padrões. **SVC(random_state = 2025, C=1.0, kernel='rbf')**

3. **Logistic Regression**: solver **liblinear** é recomendado para pequenos datasets, embora outros algoritmos de otimização como **lbfgs**, **gama** e **newton-cg** também apresentaram "resultados similares". **LogisticRegression(random_state=2025, penalty='l2', solver='liblinear')**
4. **Random Forest**: Os hyperparâmetros **max_depth**, **min_samples_split** e **min_samples_leaf** ajudaram na melhora da acurácia **RandomForestClassifier(random_state=2025, n_estimators=100, max_depth=10, min_samples_split=5, min_samples_leaf=3)**
5. **KNN**: **KNeighborsClassifier(n_neighbors=20, weights='distance', metric='minkowski')**
6. **Gaussian NB**: **GaussianNB(var_smoothing=0.65)**,
7. **MLPClassifier (Scikit-learn)**: **MLPClassifier(random_state=2025, hidden_layer_sizes=(20,), max_iter=500)**

```
# Separação treinamento e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)

# Definindo os modelos de ML
models = {
    'Decision Tree': DecisionTreeClassifier(random_state=2025, max_depth=10, min_samples_leaf=5),
    'SVM': SVC(random_state = 2025, C=1.0, kernel='rbf'),
    'Logistic Regression': LogisticRegression(random_state=2025, penalty='l2', solver='liblinear'),
    'Random Forest': RandomForestClassifier(random_state=2025, n_estimators=100, max_depth=10, min_samples_split=5, n
    'KNN': KNeighborsClassifier(n_neighbors=20, weights='distance', metric='minkowski'),
    'Gaussian NB': GaussianNB(var_smoothing=0.65),
    'MLPClassifier (Scikit-learn)': MLPClassifier(random_state=2025, hidden_layer_sizes=(20, ), max_iter=500)
}

# Treinando e avaliando os modelos
for name, model in models.items():
    print(f'Treinando modelo: {name}')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    print(f'Recall...: {recall:.4f}')
    print(f'Precision: {precision:.4f}')
    print(f'F1 Score.: {f1:.4f}')
    print(f'Acurácia: {accuracy:.4f}')

    conf_matrix = confusion_matrix(y_test, y_pred)
    # Normalize the confusion matrix
    conf_matrix_normalized = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]
    print('\nMatriz de Confusão Normalizada:')
    print(conf_matrix_normalized)

    # Visualizando a matriz de confusão com um heatmap
    plt.figure(figsize=(6, 5))
    sns.heatmap(conf_matrix_normalized, annot=True, fmt='.2f', cmap='Blues',
                xticklabels=['Sem sintomas', 'Com sintomas'],
                yticklabels=['Sem sintomas', 'Com sintomas'],
                annot_kws={"size": 16},
                cbar_kws={"shrink": .95}
                )
    plt.title(f'\nMatriz de Confusão Normalizada - {name}', fontsize=14)

    plt.xticks(fontsize=14)
    plt.yticks(fontsize=14)
    plt.xlabel('Predição', fontsize=15)
    plt.ylabel('Valor Real', fontsize=15)

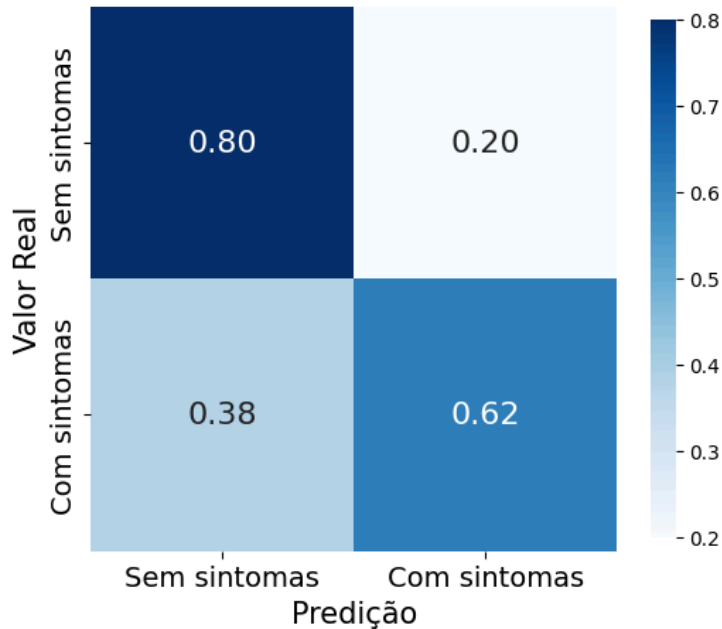
    plt.show()

    print('\n')
```

Treinando modelo: Decision Tree
Recall....: 0.6154
Precision: 0.6667
F1 Score.: 0.6400
Acurácia,: 0.7273

Matriz de Confusão Normalizada:
[[0.8 0.2]
 [0.38461538 0.61538462]]

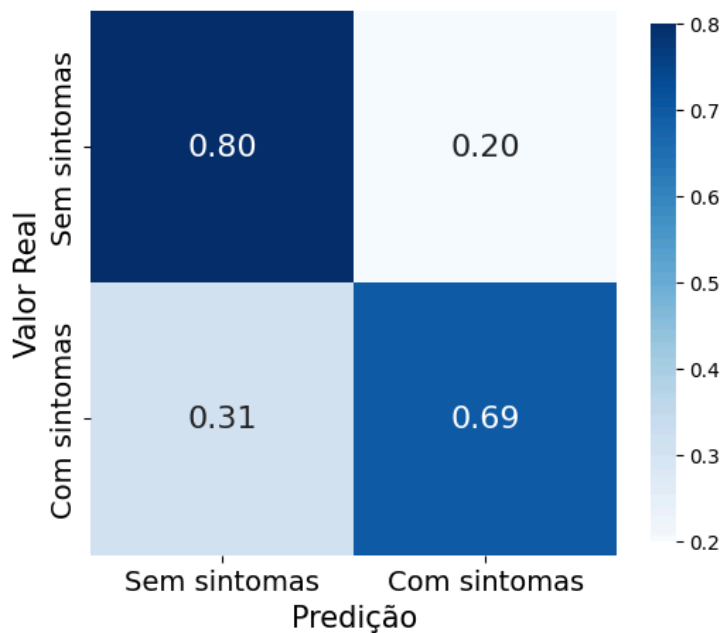
Matriz de Confusão Normalizada - Decision Tree



Treinando modelo: SVM
Recall....: 0.6923
Precision: 0.6923
F1 Score.: 0.6923
Acurácia,: 0.7576

Matriz de Confusão Normalizada:
[[0.8 0.2]
 [0.30769231 0.69230769]]

Matriz de Confusão Normalizada - SVM

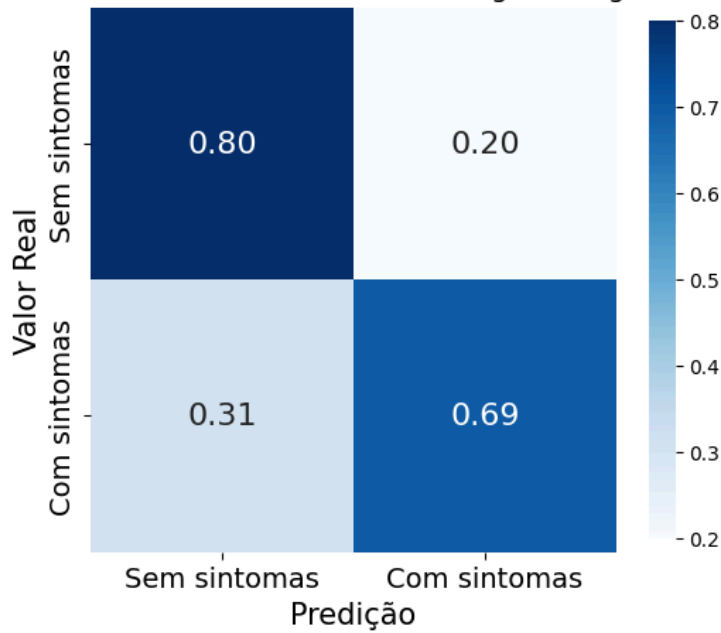


Treinando modelo: Logistic Regression
Recall....: 0.6923
Precision: 0.6923
F1 Score.: 0.6923
Acurácia,: 0.7576

Matriz de Confusão Normalizada:

```
[[0.8      0.2      ]  
 [0.30769231 0.69230769]]
```

Matriz de Confusão Normalizada - Logistic Regression



Treinando modelo: Random Forest

Recall...: 0.7692

Precision: 0.7692

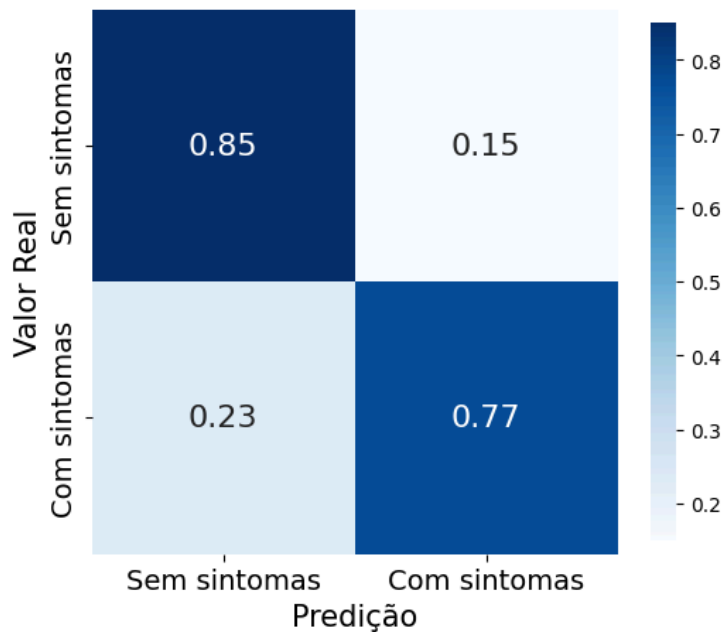
F1 Score.: 0.7692

Acurácia,: 0.8182

Matriz de Confusão Normalizada:

```
[[0.85      0.15      ]  
 [0.23076923 0.76923077]]
```

Matriz de Confusão Normalizada - Random Forest



Treinando modelo: KNN

Recall...: 0.6923

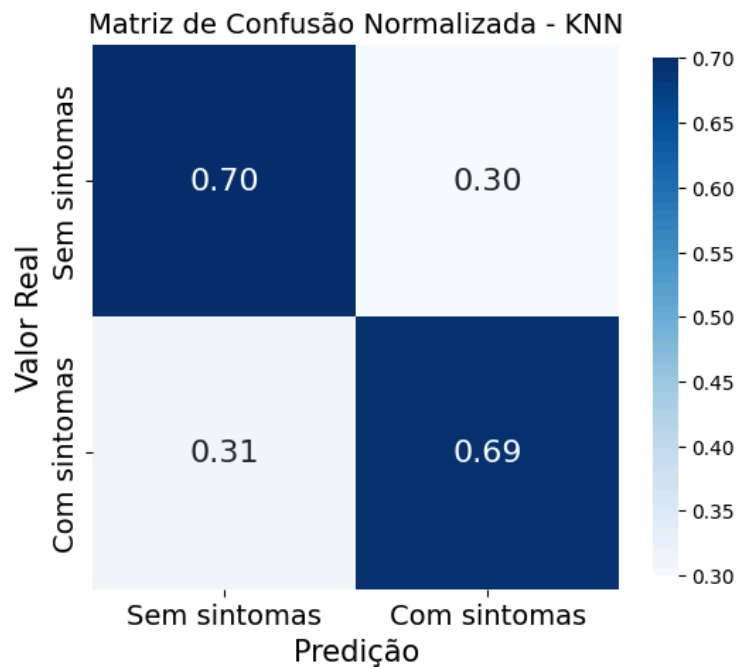
Precision: 0.6000

F1 Score.: 0.6429

Acurácia,: 0.6970

Matriz de Confusão Normalizada:

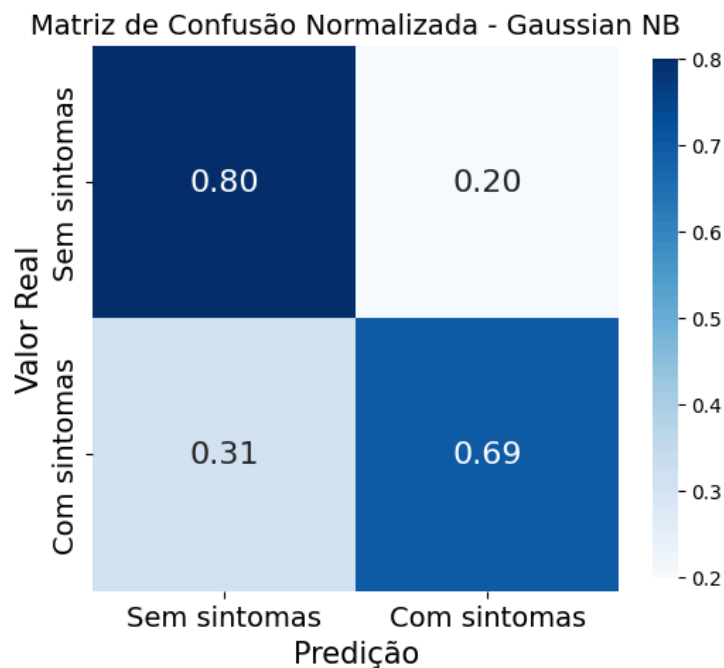
```
[[0.7      0.3      ]  
 [0.30769231 0.69230769]]
```

```
Treinando modelo: Gaussian NB
Recall...: 0.6923
Precision: 0.6923
F1 Score.: 0.6923
Acurácia.: 0.7576
```

Matriz de Confusão Normalizada:

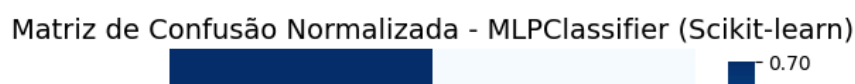
```
[[0.8          0.2       ]
 [0.30769231  0.69230769]]
```

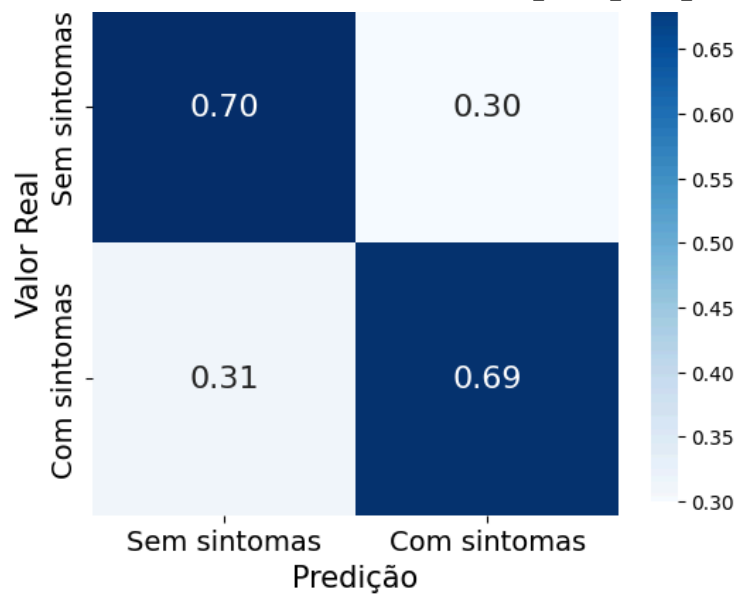


```
Treinando modelo: MLPClassifier (Scikit-learn)
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning:
  warnings.warn(
Recall...: 0.6923
Precision: 0.6000
F1 Score.: 0.6429
Acurácia,: 0.6970
```

Matriz de Confusão Normalizada:

```
[[0.7          0.3       ]
 [0.30769231  0.69230769]]
```





```
#from google.colab import drive
#drive.mount('/content/drive')

import pandas as pd
# DataFrame para armazenar os resultados
results_df = pd.DataFrame(columns=['Modelo', 'Acurácia TR', 'Acurácia TE', 'Precisão', 'Recall', 'f1_score'])

# Treinando e avaliando cada modelo e armazenando os resultados no DataFrame
for name, model in models.items():
    print(f'Treinando modelo: {name}')

    #teste de overfitting:
    model.fit(X_train, y_train)
    y_train_pred = model.predict(X_train) # Predição no conjunto de treinamento
    train_accuracy = accuracy_score(y_train, y_train_pred)

    y_pred = model.predict(X_test) # Predição no conjunto de testes
    test_accuracy = accuracy_score(y_test, y_pred)

    precision = precision_score(y_test, y_pred)
    recall = recall_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred)
    accuracy_test = accuracy_score(y_test, y_pred)

    results_df = pd.concat([results_df, pd.DataFrame({'Modelo': [name], 'Acurácia TR': [train_accuracy], 'Acurácia TE'

# Tabela comparativa
print("\nTabela Comparativa dos Modelos:")
results_df
```

```
Treinando modelo: Decision Tree
Treinando modelo: SVM
Treinando modelo: Logistic Regression
Treinando modelo: Random Forest
/tmp/ipython-input-44-2822709808.py:22: FutureWarning: The behavior of DataFrame concatenation with empty or all-
results_df = pd.concat([results_df, pd.DataFrame({'Modelo': [name], 'Acurácia TR': [train_accuracy], 'Acurácia
Treinando modelo: KNN
Treinando modelo: Gaussian NB
Treinando modelo: MLPClassifier (Scikit-learn)
```

```
Tabela Comparativa dos Modelos:
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning:
warnings.warn()
```

	Modelo	Acurácia TR	Acurácia TE	Precisão	Recall	f1_score	
0	Decision Tree	0.783784	0.727273	0.666667	0.615385	0.640000	
1	SVM	0.816216	0.757576	0.692308	0.692308	0.692308	
2	Logistic Regression	0.751351	0.757576	0.692308	0.692308	0.692308	
3	Random Forest	0.800000	0.818182	0.769231	0.769231	0.769231	
4	KNN	0.962162	0.696970	0.600000	0.692308	0.642857	
5	Gaussian NB	0.691892	0.757576	0.692308	0.692308	0.692308	
6	MLPClassifier (Scikit-learn)	0.854054	0.696970	0.600000	0.692308	0.642857	

Próximas etapas: [Gerar código com results_df](#) [Ver gráficos recomendados](#) [New interactive sheet](#)

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
# ... (Seu código anterior)

# DataFrame para armazenar os resultados
results_df = pd.DataFrame(columns=['Modelo', 'Acurácia'])

# Treinando e avaliando cada modelo e armazenando os resultados no DataFrame
for name, model in models.items():
    print(f'Treinando modelo: {name}')
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    accuracy = accuracy_score(y_test, y_pred)
```

```
results_df = pd.concat([results_df, pd.DataFrame({'Modelo': [name], 'Acurácia': [accuracy]})], ignore_index=True)
print(f'Acurácia: {accuracy}')

# Exibindo a tabela comparativa
print("\nTabela Comparativa dos Modelos:")
print(results_df)

# Criando o gráfico de barras
plt.figure(figsize=(10, 6))
ax = sns.barplot(x='Modelo', y='Acurácia', data=results_df)
plt.title('Comparação da Acurácia dos Modelos')
plt.xlabel('Modelo')
plt.ylabel('Acurácia')
plt.xticks(rotation=45, ha='right')

# Definir limite fixo para o eixo y
plt.ylim(0, 1) # Limites mínimo e máximo desejados

# Destacando o melhor resultado
best_model = results_df.loc[results_df['Acurácia'].idxmax()]
ax.text(best_model.name, best_model['Acurácia'], f"{best_model['Acurácia']:.2f}", ha='center', va='bottom', color='red')
ax.annotate('Melhor Resultado', xy=(best_model.name, best_model['Acurácia']), xytext=(best_model.name, best_model['Acurácia'] + 0.05),
           arrowprops=dict(facecolor='blue', shrink=2), fontsize=10, color='red')

plt.tight_layout()
plt.show()
```

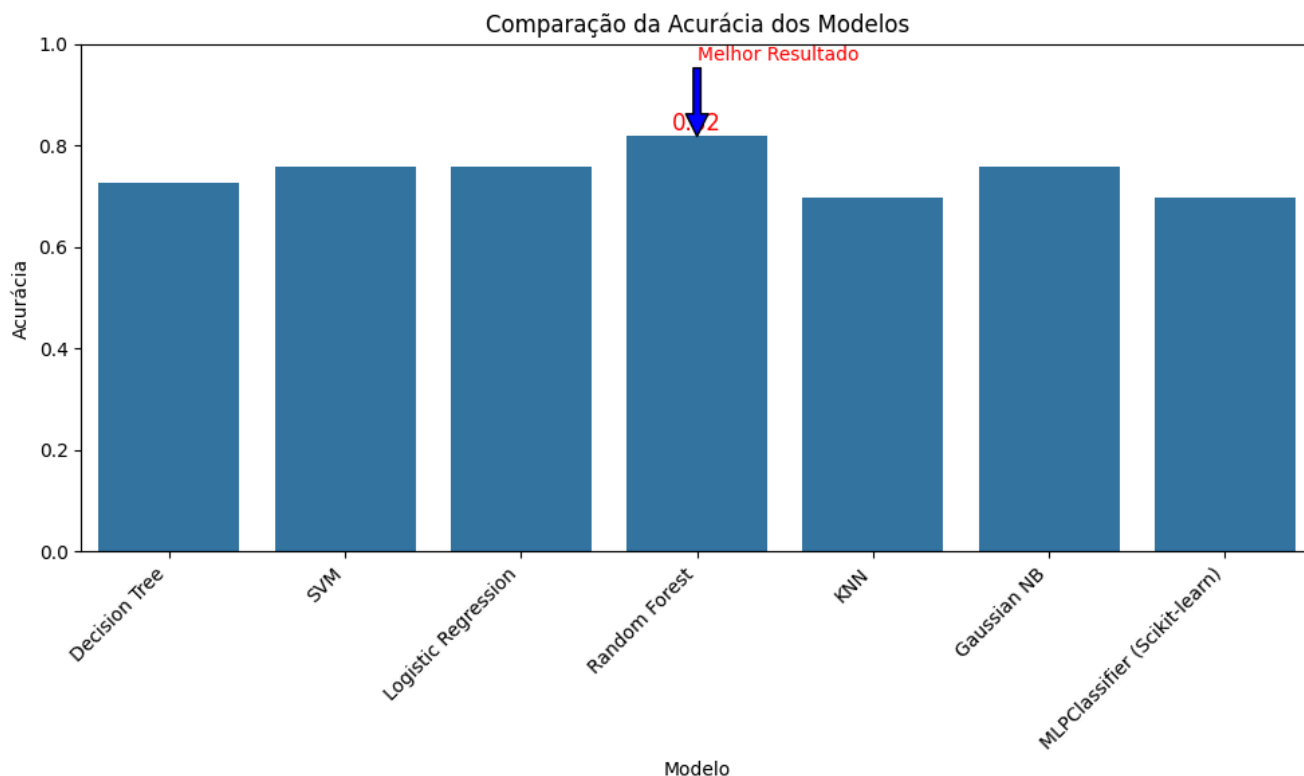
```

/tmp/ipython-input-13-2514689953.py:15: FutureWarning: The behavior of DataFrame concatenation with empty or al
results_df = pd.concat([results_df, pd.DataFrame({'Modelo': [name], 'Acurácia': [accuracy]})], ignore_index=T
Treinando modelo: Decision Tree
Acurácia: 0.72727272727273
Treinando modelo: SVM
Acurácia: 0.75757575757576
Treinando modelo: Logistic Regression
Acurácia: 0.75757575757576
Treinando modelo: Random Forest
Acurácia: 0.81818181818182
Treinando modelo: KNN
Acurácia: 0.6969696969697
Treinando modelo: Gaussian NB
Acurácia: 0.75757575757576
Treinando modelo: MLPClassifier (Scikit-learn)
/usr/local/lib/python3.11/dist-packages/sklearn/neural_network/_multilayer_perceptron.py:691: ConvergenceWarning
warnings.warn(
Acurácia: 0.6969696969697

```

Tabela Comparativa dos Modelos:

	Modelo	Acurácia
0	Decision Tree	0.727273
1	SVM	0.757576
2	Logistic Regression	0.757576
3	Random Forest	0.818182
4	KNN	0.696970
5	Gaussian NB	0.757576
6	MLPClassifier (Scikit-learn)	0.696970



```
pip install shap
```

```

Requirement already satisfied: shap in /usr/local/lib/python3.11/dist-packages (0.48.0)
Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (from shap) (2.0.2)
Requirement already satisfied: scipy in /usr/local/lib/python3.11/dist-packages (from shap) (1.16.0)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.11/dist-packages (from shap) (1.6.1)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (from shap) (2.2.2)
Requirement already satisfied: tqdm>=4.27.0 in /usr/local/lib/python3.11/dist-packages (from shap) (4.67.1)
Requirement already satisfied: packaging>20.9 in /usr/local/lib/python3.11/dist-packages (from shap) (25.0)
Requirement already satisfied: slicer==0.0.8 in /usr/local/lib/python3.11/dist-packages (from shap) (0.0.8)
Requirement already satisfied: numba>=0.54 in /usr/local/lib/python3.11/dist-packages (from shap) (0.60.0)
Requirement already satisfied: cloudpickle in /usr/local/lib/python3.11/dist-packages (from shap) (3.1.1)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.11/dist-packages (from shap) (4.14.1)
Requirement already satisfied: llvmlite<0.44,>=0.43.0dev0 in /usr/local/lib/python3.11/dist-packages (from numba>
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (from pandas->sh
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (2025
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (from pandas->shap) (202
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn->shap)
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.11/dist-packages (from scikit-learn

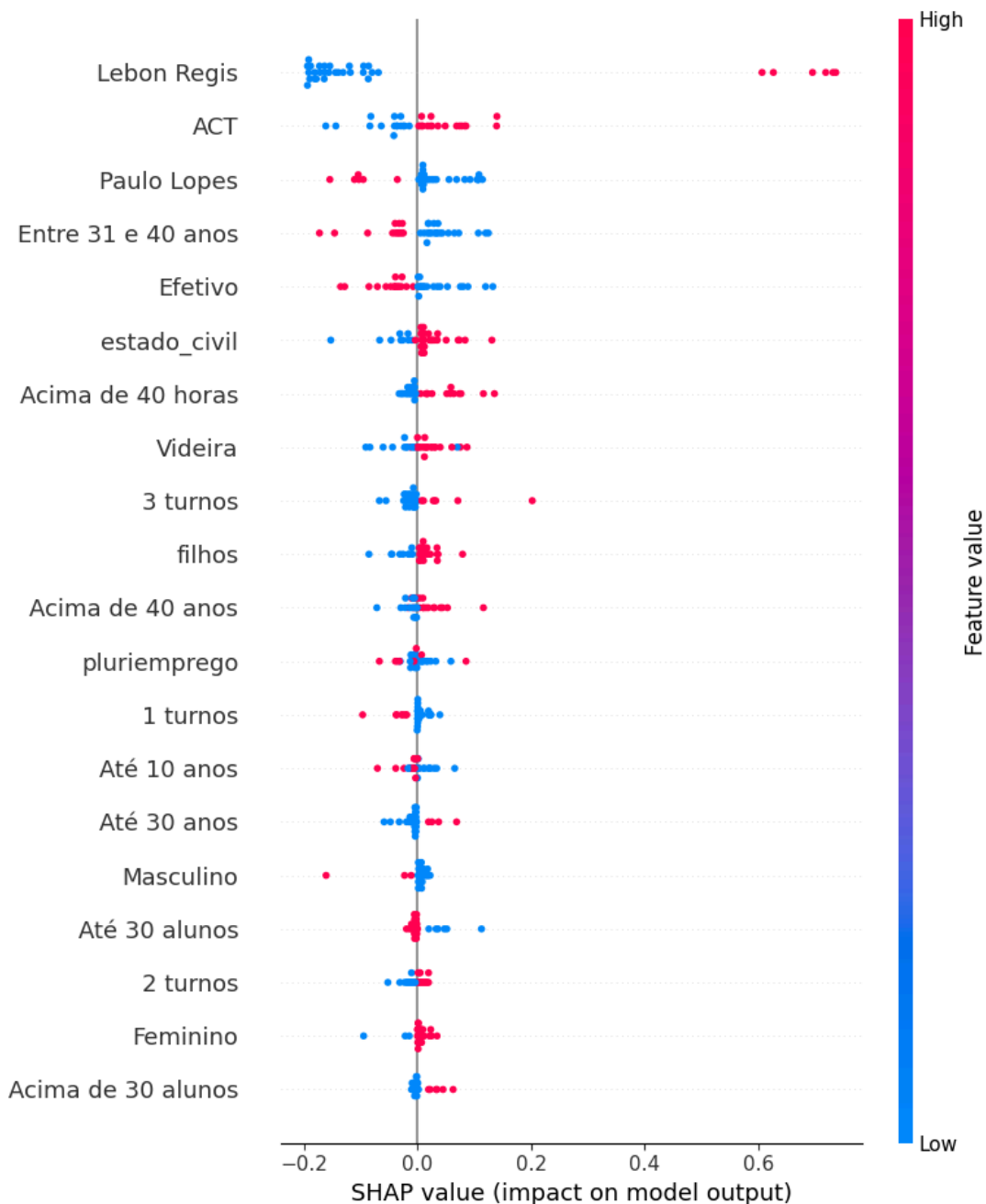
```

Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (from python-dateutil>=2.8.2->


```
import shap
best_model = RandomForestClassifier(random_state=2025, n_estimators=100, max_depth=10, min_samples_split=5, min_samp1
best_model.fit(X_train, y_train)
explainer = shap.Explainer(best_model.predict, X_train)
shap_values = explainer(X_test)
shap.summary_plot(shap_values, X_test, feature_names=data.drop('sintomas_osteomusculares', axis=1).columns)

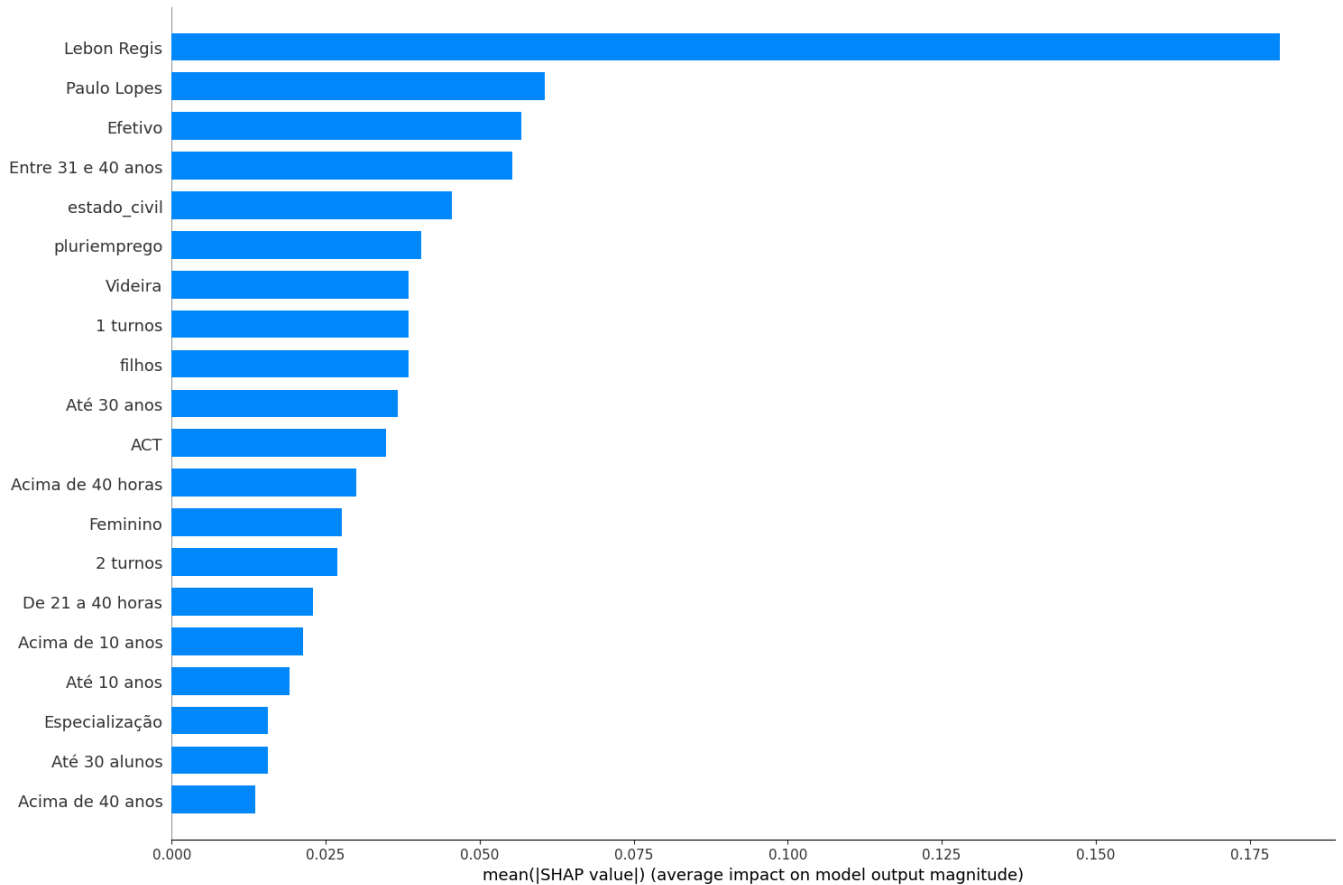
# Impacto das características individuais
#shap.dependence_plot("professor", shap_values.values, X_test, feature_names=data.drop('qualidade_sono', axis=1).colu

↔ PermutationExplainer explainer: 34it [00:10, 10.02s/it]
/tmp/ipython-input-46-3325463486.py:6: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`
shap.summary_plot(shap_values, X_test, feature_names=data.drop('sintomas_osteomusculares', axis=1).columns)
```



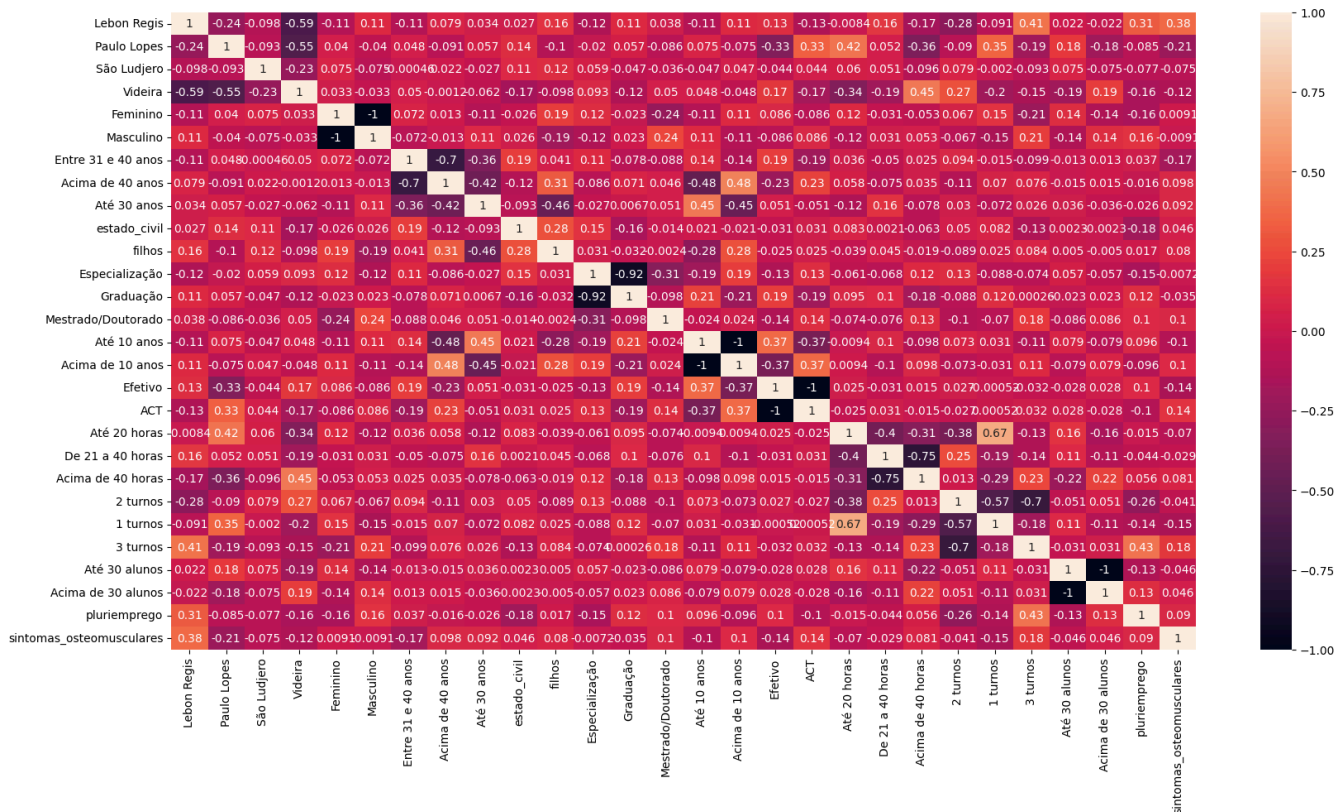
```
shap.summary_plot(shap_values, X_train, plot_type="bar", plot_size=(15,10))
```

 /tmp/ipython-input-16-2254383541.py:1: FutureWarning: The NumPy global RNG was seeded by calling `np.random.seed`
shap.summary_plot(shap_values, X_train, plot_type="bar", plot_size=(15,10))



```
import seaborn as sns
import matplotlib.pyplot as plt
plt.figure(figsize=(20, 10))
sns.heatmap(data.corr(), annot=True)
```

<Axes: >



✎ Daqui para baixo é rascunho, usado para testar, nenhum funcionou melhor que o ajuste manual

```
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression

# definição dos parâmetros
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],
    'penalty': ['l1', 'l2'],
    'solver': ['liblinear']
}

logreg = LogisticRegression(random_state=2025)
grid_search = GridSearchCV(logreg, param_grid, cv=15, scoring='accuracy')

grid_search.fit(X_train, y_train)

print("Melhor parâmetro encontrado: ", grid_search.best_params_)
print("Melhor acurácia da cross-validation: ", grid_search.best_score_)

best_logreg_model = grid_search.best_estimator_
y_pred = best_logreg_model.predict(X_test)
test_accuracy = accuracy_score(y_test, y_pred)
print("Acurácia do conjunto de teste com os melhores parâmetros: ", test_accuracy)

conf_matrix = confusion_matrix(y_test, y_pred)
conf_matrix_normalized = conf_matrix.astype('float') / conf_matrix.sum(axis=1)[:, np.newaxis]
print('Matriz de Confusão normalizada com os melhores parâmetros:')
print(conf_matrix_normalized)

# Visualização
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_normalized, annot=True, fmt='.2f', cmap='Blues',
            xticklabels=['Sem sintomas', 'Com sintomas'],
            yticklabels=['Sem sintomas', 'Com sintomas'])
plt.title('Matriz de Confusão Proporcional - Melhor Logistic Regression')
plt.xlabel('Predição')
plt.ylabel('Valor Real')
plt.show()
```