UNIVERSIDADE PRESBITERIANA MACKENZIE

Integrantes:

Bruno Zovaro Nascimento - 10424880

Douglas Novaes Dias – 14023666

Milan Mirco Moraes Mazur – 10363757

Paulo Cesar Masson Junior - 10416023

PROJETO APLICADO II

CLASSIFICAÇÃO ETÁRIA A PARTIR DE ÍNDICES DE SAÚDE

São Paulo

2024

SUMÁRIO

1. Bibliotecas utilizadas

Primeiro importamos as bibliotecas a serem utilizadas no código python:

```
import pandas as pd
import seaborn as sns
from matplotlib import pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import make_scorer, recall_score
from sklearn.preprocessing import LabelEncoder
```

2. Base de dados

O dataset utilizado está disponível no repositório UCI de aprendizado de máquina. Ele se chama Health Data Set e contém informações de saúde para análises. Especificamente, esse dataset é composto por várias características relacionadas à saúde de pacientes, como variáveis que podem influenciar o diagnóstico ou o tratamento, como idade, tipo de doença, entre outros.

Esses dados podem ser utilizados para modelagem preditiva, como classificação e regressão, a fim de prever resultados relacionados à saúde ou descobrir padrões e insights importantes.

Há uma importante classificação que separa os adultos dos idosos (Adults x Seniors) sendo que essa divisão é o foco do presente estudo.

Assim, foi realizada a importação do Dataset:

df_health = pd.read_csv('https://archive.ics.uci.edu/static/public/887/data.csv', sep=",", encoding='latin-1')

#print(df_health.head)

print(df_health.dtypes)

Saída:

```
In [58]: df_health = pd.read_csv('https://archive.ics.uci.edu/static/public/887/
data.csv', sep=",", encoding='latin-1')
    ...: print(df_health.head)
    ...: print(df_health.dtypes)
    ...:
    ...:
    ...:
<bound method NDFrame.head of          SEQN age_group  RIDAGEYR  RIAGENDR  ...
LBXGLU  DIQ010  LBXGLT  LBXIN
0     73564.0     Adult      61.0       2.0  ...    110.0     2.0   150.0  14.91
1     73568.0     Adult      26.0       2.0  ...     89.0     2.0    80.0   3.85
2     73576.0     Adult      16.0       1.0  ...     89.0     2.0    68.0   6.14
3     73577.0     Adult      32.0       1.0  ...    104.0     2.0    84.0  16.15
4     73580.0     Adult      38.0       2.0  ...    103.0     2.0    81.0  10.92
...       ...       ...       ...       ...  ...      ...     ...     ...    ...
2273  83711.0     Adult      38.0       2.0  ...    100.0     2.0    73.0   6.53
2274  83712.0     Adult      61.0       1.0  ...     93.0     2.0   208.0  13.02
2275  83713.0     Adult      34.0       1.0  ...    103.0     2.0   124.0  21.41
2276  83718.0     Adult      60.0       2.0  ...     90.0     2.0   108.0   4.99
2277  83727.0     Adult      26.0       1.0  ...    108.0     2.0   108.0   3.76

[2278 rows x 10 columns]>
SEQN         float64
age_group     object
RIDAGEYR     float64
RIAGENDR     float64
PAQ605       float64
BMXBMI       float64
LBXGLU       float64
DIQ010       float64
LBXGLT       float64
LBXIN        float64
dtype: object

In [59]:
```

3. Análises Exploratórias

Foi feita verificação se haviam valores ausentes:

```
df_proporcao=df_health.isnull().sum()/len(df_health)
print(df_proporcao)
```

Saída:

```
print(df_proporcao)
SEQN          0.0
age_group     0.0
RIDAGEYR      0.0
RIAGENDR      0.0
PAQ605        0.0
BMXBMI        0.0
LBXGLU        0.0
DIQ010        0.0
LBXGLT        0.0
LBXIN         0.0
dtype: float64
```

Foi feita verificação se haviam linhas duplicadas:

```
total_duplicados = df_health.duplicated(keep=False).sum()
```

print(f'Total de linhas duplicadas: {total_duplicados}')

Saída:

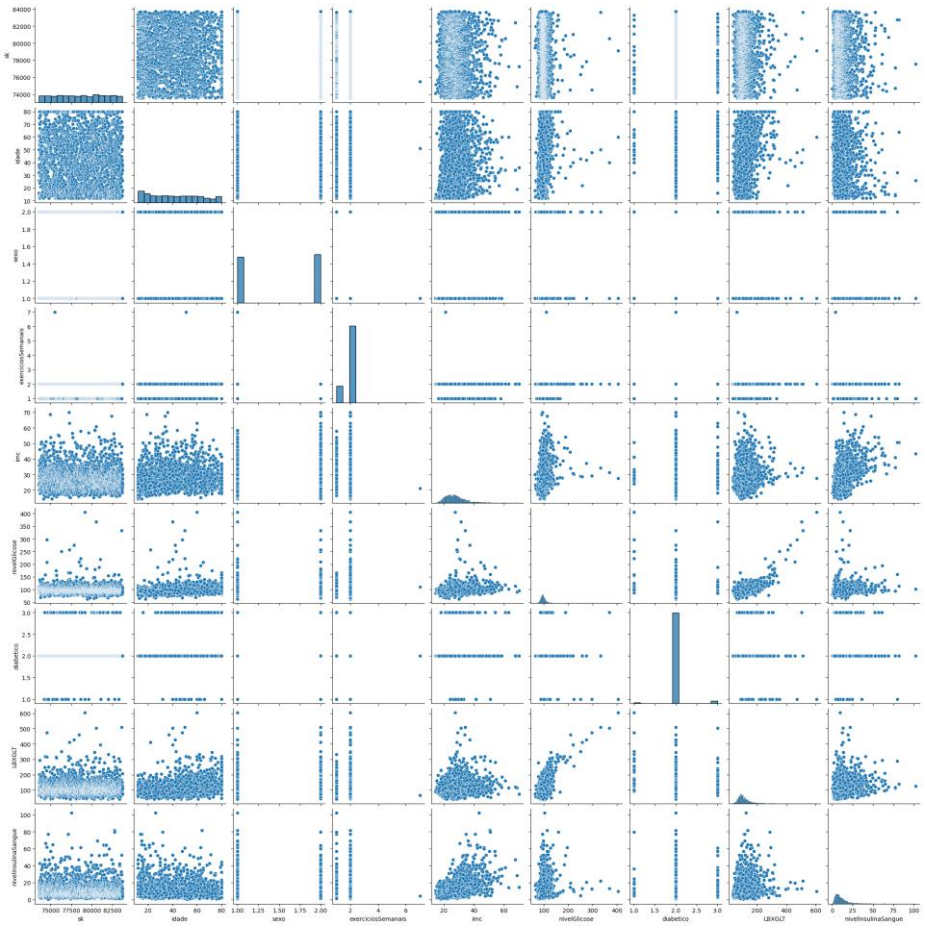
Total de linhas duplicadas: 0

Foi produzido um Gráfico de dispersão (pairplot):

sns.pairplot(df_health)

plt.show()

Saída:



O pair plot é uma ferramenta visual usada na análise exploratória de dados para observar as relações entre múltiplas variáveis numéricas. Ele cria uma matriz de gráficos de dispersão, mostrando como cada par de variáveis se comporta entre si, e, no eixo diagonal, apresenta distribuições univariadas das variáveis. Essa visualização ajuda a identificar padrões, correlações, clusters, e outliers, além de auxiliar na seleção de variáveis relevantes para modelos preditivos. É particularmente útil para explorar rapidamente a estrutura dos dados e ganhar insights sobre a interdependência entre as variáveis.

Para preparar uma matriz de correlação, foi feito o encode dos dados com drop_first=False para permitir que todas as colunas, incluindo as variáveis categóricas, fossem visualizadas na análise. Ao definir drop_first=False, todas as categorias são mantidas, criando variáveis binárias para cada categoria da variável original. Isso facilita a análise da correlação entre todas as variáveis, mas em um cenário de treinamento de modelo, esse procedimento pode não ser recomendado, pois pode resultar em multicolinearidade. A multicolinearidade ocorre quando duas ou mais variáveis independentes estão altamente correlacionadas, o que pode distorcer os resultados de alguns algoritmos, como a regressão linear. Para evitar esse problema, em geral, utiliza-se drop_first=True, o que elimina uma categoria para evitar a criação de variáveis redundantemente correlacionadas.

No mais, foi preparada a matriz de correlação e gráfico com mapa de calor, conforme segue:

```
# Usar one-hot encoding para colunas categóricas
df_encoded = pd.get_dummies(df_health, drop_first=False)
# Calcular a matriz de correlação
correlation_matrix = df_encoded.corr()
#correlation_matrix = df_sample.corr()
print(correlation_matrix)
# Criar um mapa de calor da matriz de correlação
plt.figure(figsize=(12, 10))
```

sns.heatmap(correlation_matrix, annot=True, fmt=".2f", cmap="coolwarm", square=True, cbar_kws={"shrink": .8})

plt.title("Matriz de Correlação")

plt.show()

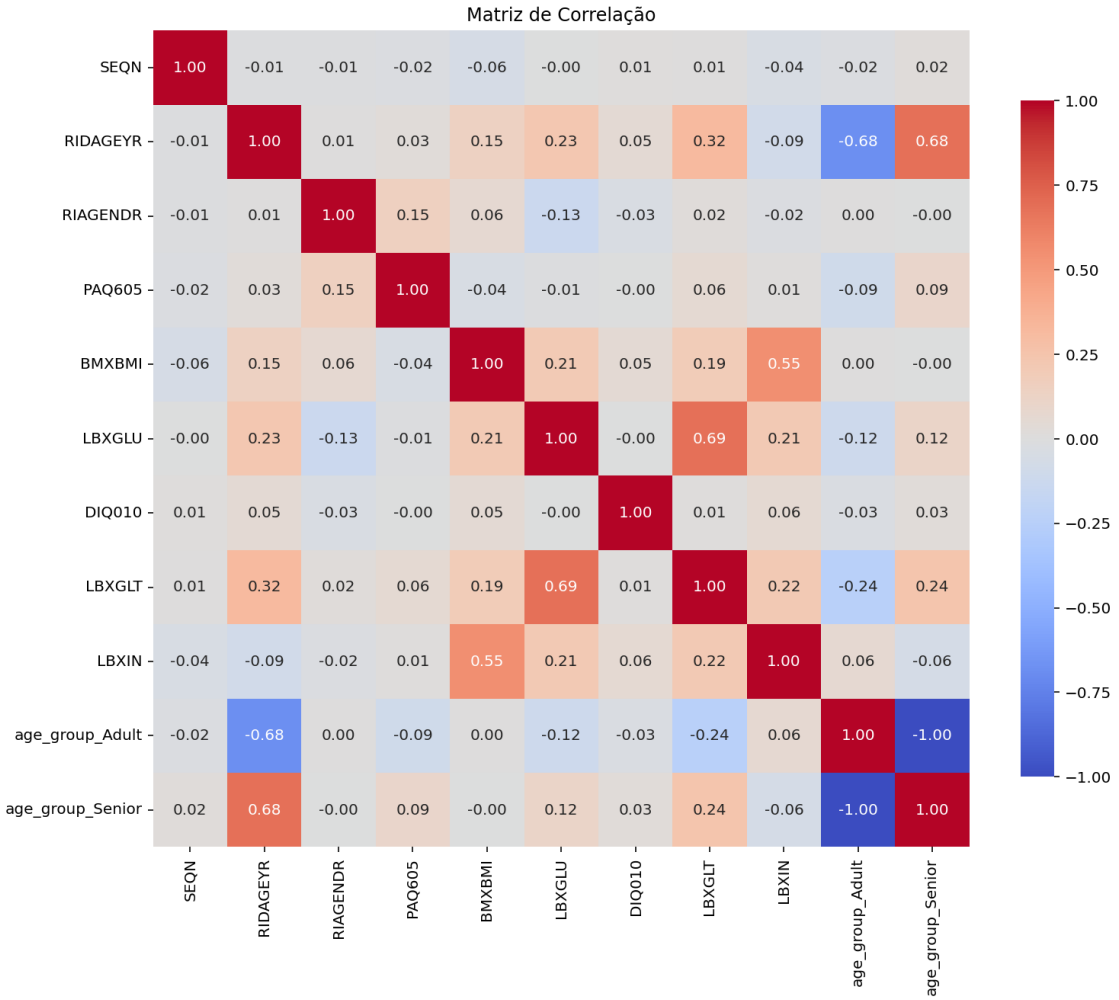
Saída:

Cálculo da correlação:

	SEQN	RIDAGEYR	...	age_group_Adult	age_group_Senior
SEQN	1.000000	-0.008806	...	-0.018257	0.018257
RIDAGEYR	-0.008806	1.000000	...	-0.684490	0.684490
RIAGENDR	-0.012962	0.006398	...	0.002767	-0.002767
PAQ605	-0.019701	0.025973	...	-0.094789	0.094789
BMXBMI	-0.061343	0.147163	...	0.004147	-0.004147
LBXGLU	-0.004147	0.229624	...	-0.116462	0.116462
DIQ010	0.014102	0.049970	...	-0.026399	0.026399
LBXGLT	0.006036	0.318044	...	-0.243113	0.243113
LBXIN	-0.040028	-0.091879	...	0.064159	-0.064159
age_group_Adult	-0.018257	-0.684490	...	1.000000	-1.000000
age_group_Senior	0.018257	0.684490	...	-1.000000	1.000000

[11 rows x 11 columns]

Mapa de calor da correlação:



Um mapa de calor de uma matriz de correlação é uma ferramenta visual usada para representar graficamente as correlações entre diferentes variáveis em um conjunto de dados. Ele utiliza cores para mostrar a intensidade e a direção da correlação entre as variáveis, facilitando a identificação de padrões, como relações fortes ou fracas. Valores próximos de 1 indicam uma correlação positiva forte, valores próximos de -1 indicam uma correlação negativa forte, e valores próximos de 0 indicam pouca ou nenhuma correlação. Esse tipo de visualização é útil para detectar variáveis que estão altamente correlacionadas entre si, o que pode ser relevante para eliminar redundâncias em modelos de machine learning, ou para identificar relações que podem ser exploradas em análises mais profundas. Além disso, um mapa de calor ajuda a simplificar a interpretação de matrizes de correlação complexas, tornando o processo mais eficiente e acessível para tomada de decisões.

Na sequência foi verificado o tipo de dado de cada atributo preparando um label encode da coluna age_group, para evitar qualquer problema mais à frente no aprendizado de máquina.

```
print(df_encoded.dtypes)
```

Saída:

```
SEQN                float64
RIDAGEYR            float64
RIAGENDR            float64
PAQ605              float64
BMXBMI              float64
LBXGLU              float64
DIQ010              float64
LBXGLT              float64
LBXIN               float64
age_group_Adult        bool
age_group_Senior       bool
dtype: object
```

Para preparar o label encode foi consultado o seguinte:

```
print(df_health['age_group'].unique())
```

Saída:

```
['Adult' 'Senior']
```

E feito o label encode:

```
label_encoder = LabelEncoder()
# Aplicando o Label Encoding na coluna 'age_group'
```

```
df_health['age_group_encoded'] = label_encoder.fit_transform(df_health['age_group'])
print(df_health.head())
```

Saída (com nova coluna ao final):

```
      SEQN age_group  RIDAGEYR  ...  LBXGLT  LBXIN  age_group_encoded
0  73564.0     Adult      61.0  ...   150.0  14.91                 0
1  73568.0     Adult      26.0  ...    80.0   3.85                 0
2  73576.0     Adult      16.0  ...    68.0   6.14                 0
3  73577.0     Adult      32.0  ...    84.0  16.15                 0
4  73580.0     Adult      38.0  ...    81.0  10.92                 0

[5 rows x 11 columns]
```

Avaliar os outliers em modelos de classificação é crucial, pois esses valores extremos podem distorcer os resultados, afetando a precisão e a generalização do modelo. Outliers podem influenciar a escolha de limites de decisão, gerar viés nas métricas de desempenho e reduzir a eficiência do modelo. Dependendo do tipo de modelo, como em árvores de decisão ou regressão logística, a presença de outliers pode resultar em previsões imprecisas. Identificar e tratar os outliers adequadamente ajuda a melhorar a robustez do modelo, assegurando que ele seja mais eficaz em lidar com dados reais e representativos.

Assim foi feita estrutura de código em python para verificação de valores em cada coluna, no sentido de buscar Outliers. No caso abaixo foi feita análise da coluna com mais correlação com a variável alvo:

```
print(df_health.columns)
# Avaliacao de outliers
# Para calcular múltiplos quantis (ex: 0.25, 0.5, 0.75)
print('Coluna com mais correlação com Senior x Adult = LBXGLT:')
quantis_coluna=df_health['LBXGLT'].quantile([0.25, 0.5, 0.75])
print(quantis_coluna)
resumo_coluna=df_health['LBXGLT'].describe()
print(resumo_coluna)
```

Saída:

```
Index(['SEQN', 'age_group', 'RIDAGEYR', 'RIAGENDR', 'PAQ605', 'BMXBMI',
       'LBXGLU', 'DIQ010', 'LBXGLT', 'LBXIN', 'age_group_encoded'],
      dtype='object')
Coluna com mais correlação com Senior x Adult = LBXGLT:
0.25     87.0
0.50    105.0
0.75    130.0
Name: LBXGLT, dtype: float64
count    2278.000000
mean      114.978929
std        47.061239
min        40.000000
25%        87.000000
50%       105.000000
75%       130.000000
max       604.000000
Name: LBXGLT, dtype: float64
```

E na sequência foi feita a análise de quantidade de outliers nessa coluna 'LBXGLT' (concentração de glutationa peroxidase no sangue):

```
# Calcular o intervalo interquartil (IQR)
IQR = quantis_coluna[0.75] - quantis_coluna[0.25]
# Definir os limites inferior e superior
limite_inferior = quantis_coluna[0.25] - 1.5 * IQR
limite_superior = quantis_coluna[0.75] + 1.5 * IQR
# Identificar os outliers
outliers = df_health[(df_health['LBXGLT'] < limite_inferior) | (df_health['LBXGLT'] > limite_superior)]
print(outliers)
# Contar a quantidade de outliers
```

```
quantidade_outliers = outliers.shape[0]
print("Quantidade de outliers detectados:", quantidade_outliers)
```

Saída:

```
         SEQN age_group  RIDAGEYR  ...  LBXGLT  LBXIN  age_group_encoded
6     73587.0     Adult      14.0  ...   202.0  21.11                 0
14    73639.0    Senior      71.0  ...   295.0  22.92                 1
55    73816.0     Adult      64.0  ...   203.0  15.49                 0
82    73975.0     Adult      64.0  ...   215.0   2.10                 0
93    74033.0    Senior      80.0  ...   215.0   3.62                 1
...       ...       ...       ...  ...     ...    ...               ...
2196  83358.0     Adult      37.0  ...   197.0  40.33                 0
2220  83448.0     Adult      43.0  ...   287.0  13.64                 0
2221  83450.0    Senior      68.0  ...   231.0   9.65                 1
2253  83624.0     Adult      50.0  ...   510.0  21.87                 0
2274  83712.0     Adult      61.0  ...   208.0  13.02                 0

[121 rows x 11 columns]
Quantidade de outliers detectados: 121
```

4. Treinamento e Resultados

Realizou-se a preparação dos dados para o treinamento de um modelo de machine learning, com o objetivo de classificar os indivíduos em duas categorias de idade, representadas pela coluna 'age_group'. Primeiramente, as variáveis independentes foram isoladas ao remover colunas irrelevantes para a análise, como 'age_group', 'age_group_encoded' e 'RIDAGEYR', sendo 'age_group_encoded' a variável de destino. Em seguida, aplicou-se o MinMaxScaler aos dados de entrada, que normalizou os valores das variáveis para um intervalo de 0 a 1, garantindo que todas as variáveis estivessem em uma escala comum, o que é crucial para a performance de muitos algoritmos de aprendizado de máquina, especialmente aqueles que dependem da distância entre pontos. Para validar o modelo, os dados foram divididos em dois conjuntos: treino e teste, utilizando uma divisão de 70% e 30%, respectivamente, com a divisão estratificada para preservar a proporção das classes em ambos os subconjuntos. Esse processo visa otimizar o treinamento do modelo, garantindo uma avaliação justa de sua performance.

```
X=df_health.drop(columns=['age_group', 'age_group_encoded', 'RIDAGEYR'])

y=df_health['age_group_encoded']

# define o scaler, prepara e executa normalização

scaler = MinMaxScaler()

scaler.fit(X)

X=scaler.transform(X)

X_train, X_test, y_train, y_test=train_test_split(X, y, stratify=y, test_size=0.3, random_state=123)
```

A seguir passou-se para o treinamento dos modelos.

Tentou-se aplicar uma Regressão Logística para classificação de dados, utilizando a técnica de GridSearchCV com o objetivo de otimizar os parâmetros do modelo. Para isso, foi definida uma configuração inicial para o estimador de regressão logística, ajustando os parâmetros de regularização, representados pela variável C, e as opções de penalização, como l1 e l2. A escolha do parâmetro scoring='accuracy' teve como propósito otimizar o modelo com base na precisão geral das previsões.

O modelo foi treinado com o conjunto de dados de treinamento, seguido da avaliação de sua performance no conjunto de teste. A performance foi analisada por meio de classification report, que fornece métricas como precisão, recall e f1-score, e da matriz de confusão, para visualizar a eficácia do modelo na previsão de ambas as classes.

######### TENTATIVA 1 DE REGRESSAO LOGISTICA COM SCORING = ACCURACY

```
# Logistic regression com grid search:
base_estimator = LogisticRegression(max_iter=1000, solver='liblinear', class_weight='balanced')
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularização
    'penalty': ['l1', 'l2']         # Tipos de penalidade
}
# Configurando o GridSearchCV
clf = GridSearchCV(base_estimator, param_grid, cv=5, scoring='accuracy')
clf.fit(X_train, y_train)
# Resultados
print("Melhores parâmetros:", clf.best_params_)
print("Acurácia no conjunto de teste:", clf.score(X_test, y_test))
print(clf.best_estimator_)
print('\nclassification report:\n')
y_pred=clf.predict(X_test)
print(classification_report(y_test, y_pred, zero_division=0))
# Matriz de Confusão
conf_matrix = confusion_matrix(y_test, y_pred)
# Exibindo a matriz de confusão com Seaborn
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão - LogisticRegression - accuracy")
plt.show()
```

Saída:

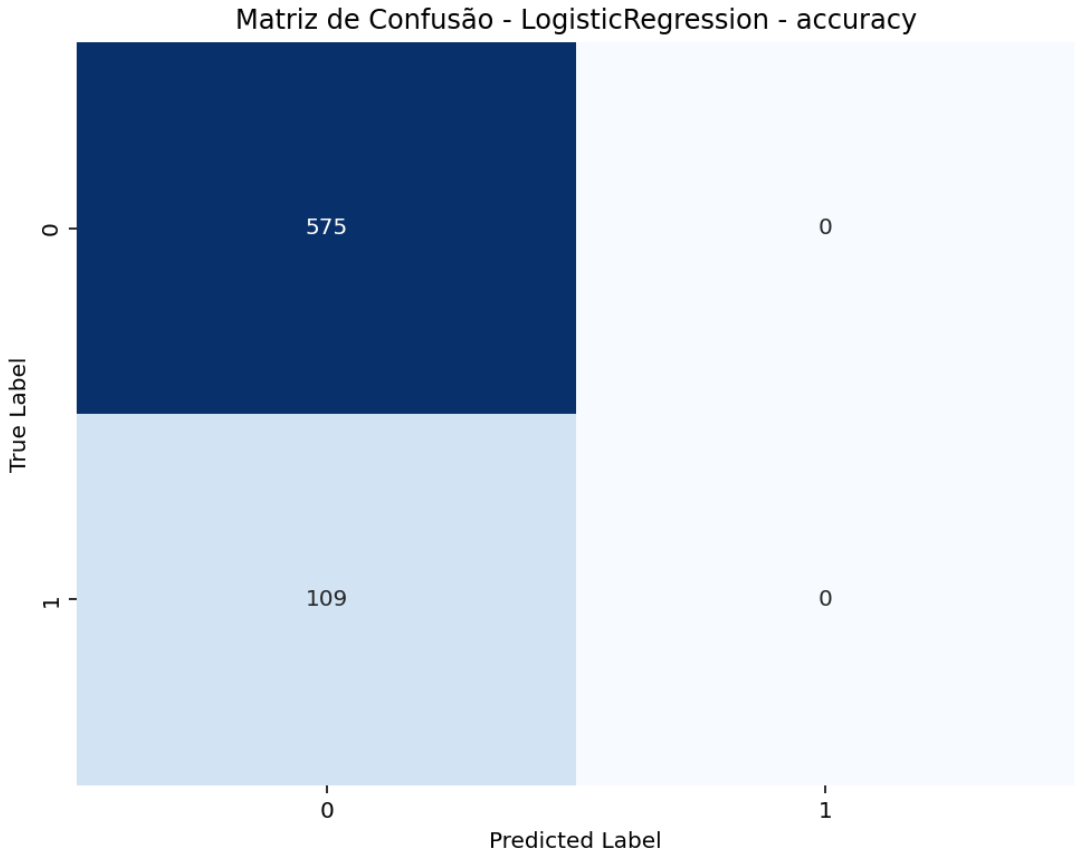
```
Melhores parâmetros: {'C': 0.01, 'penalty': 'l1'}
Acurácia no conjunto de teste: 0.8406432748538012
LogisticRegression(C=0.01, class_weight='balanced', max_iter=1000, penalty='l1',
                   solver='liblinear')

classification report:

              precision    recall  f1-score   support

           0       0.84      1.00      0.91       575
           1       0.00      0.00      0.00       109

    accuracy                           0.84       684
   macro avg       0.42      0.50      0.46       684
weighted avg       0.71      0.84      0.77       684
```



O classification report apresentado mostra a avaliação do modelo de classificação, incluindo métricas como precision (precisão), recall (revocação), f1-score e support (suporte) para cada classe, além de médias gerais.

Classe 0: O modelo teve um desempenho bastante bom na previsão da classe 0, com uma precisão de 0.84, significando que 84% das previsões feitas como classe 0 estavam corretas. O recall foi de 1.00, ou seja, o modelo foi capaz de identificar todas as instâncias reais da classe 0. O f1-score de 0.91 é uma média harmônica entre precisão e recall, indicando um bom equilíbrio entre as duas métricas.

Classe 1: O modelo, no entanto, teve um desempenho muito ruim na classe 1. O recall é 0.00, o que significa que nenhuma instância real da classe 1 foi corretamente identificada pelo modelo. A precisão também foi 0.00, indicando que, entre as previsões feitas para a classe 1, todas estavam incorretas. O f1-score também é 0.00, refletindo essa falta de desempenho.

Acurácia geral: O modelo obteve uma acurácia de 0.84, que é o percentual de previsões corretas no total. No entanto, essa métrica não reflete o desempenho desequilibrado entre as classes.

Médias:

Macro avg: Calcula a média simples das métricas para cada classe, sem considerar o número de instâncias em cada classe. Aqui, as métricas são 0.42 (precisão), 0.50 (recall) e 0.46 (f1-score), refletindo o desempenho global do modelo em ambas as classes.

Weighted avg: Calcula a média ponderada das métricas, levando em conta o número de instâncias de cada classe. Neste caso, a média ponderada para a precisão foi 0.71, para o recall foi 0.84 e o f1-score foi 0.77, indicando que o modelo foi muito bom na identificação da classe 0, mas ruim na identificação da classe 1.

A matriz de confusão indica que todas as instâncias reais da classe 0 foram corretamente classificadas como 0, enquanto todas as instâncias reais da classe 1 foram mal classificadas (nenhuma foi predita corretamente como 1, todas foram preditas como 0). Este tipo de resultado é típico em situações de desequilíbrio de classes, onde o modelo pode simplesmente "adivinhar" a classe majoritária (classe 0) para obter uma boa acurácia, mas falha em classificar corretamente as instâncias da classe minoritária (classe 1).

Essa situação sugere que o modelo precisa ser melhor ajustado.

Assim, em virtude dessa necessidade de ajustes passamos ao segundo teste com o mesmo modelo.

######### **TENTATIVA 2 DE REGRESSAO LOGISTICA COM SCORING = F1score**

```
# Logistic regression com grid search usando F1-score como métrica
clf_f1 = GridSearchCV(base_estimator, param_grid, cv=5, scoring='f1_weighted')  # Alterado para f1_weighted
param_grid = {
    'C': [0.01, 0.1, 1, 10, 100],  # Regularização
    'penalty': ['l1', 'l2']        # Tipos de penalidade
}
```

```
clf_f1.fit(X_train, y_train)
# Resultados
print("Melhores parâmetros com base no F1-score:", clf_f1.best_params_)
print("F1-score no conjunto de teste:", clf_f1.score(X_test, y_test))
print(clf_f1.best_estimator_)
print('\nclassification report:\n')
y_pred_f1 = clf_f1.predict(X_test)
print(classification_report(y_test, y_pred_f1, zero_division=0))
# Matriz de Confusão
conf_matrix_f1 = confusion_matrix(y_test, y_pred_f1)
# Exibindo a matriz de confusão com Seaborn
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_f1, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão - LogisticRegression - (F1-score)")
plt.show()
```

Saída:

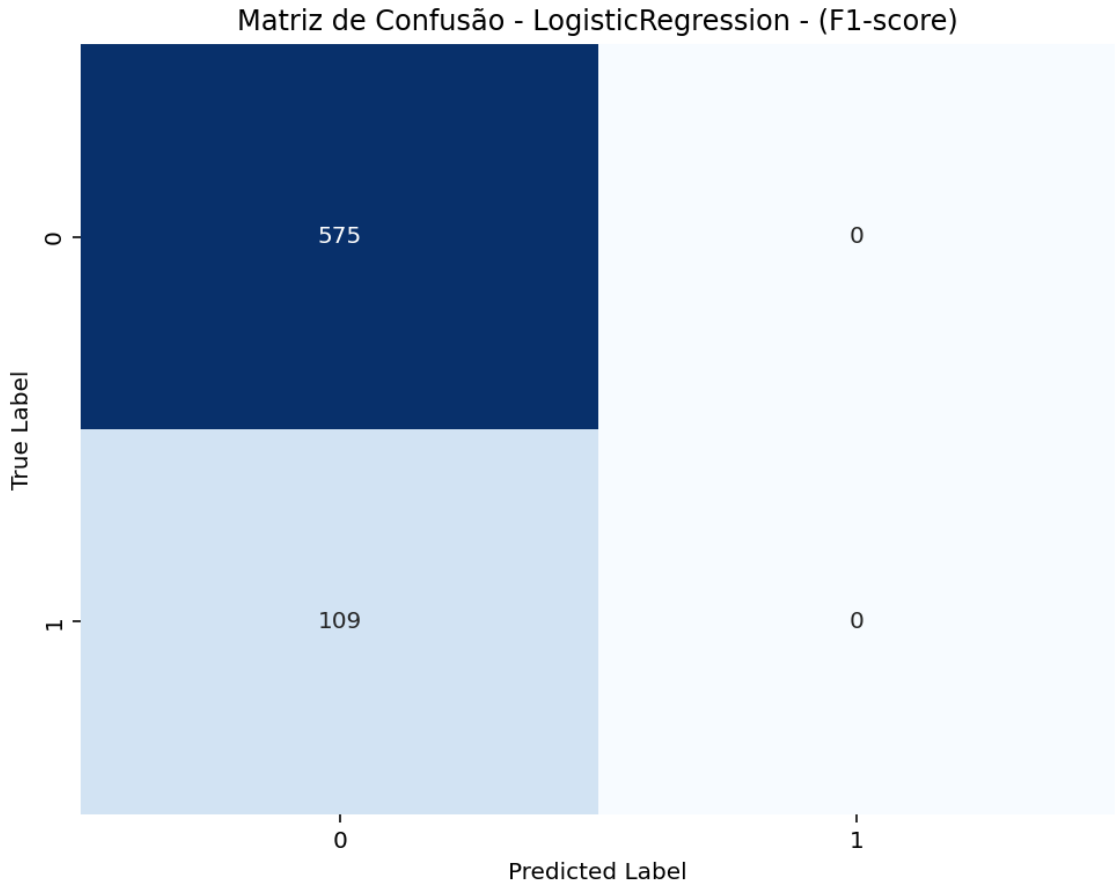
```
Melhores parâmetros com base no F1-score: {'C': 0.01, 'penalty': 'l1'}
F1-score no conjunto de teste: 0.7678631978410415
LogisticRegression(C=0.01, class_weight='balanced', max_iter=1000, penalty='l1',
                   solver='liblinear')

classification report:

              precision    recall  f1-score   support

           0       0.84      1.00      0.91       575
           1       0.00      0.00      0.00       109

    accuracy                           0.84       684
   macro avg       0.42      0.50      0.46       684
weighted avg       0.71      0.84      0.77       684
```



Vê-se que os resultados não tiveram alteração apesar da alteração da seleção do modelo de Regressão Logística que passou a usar no score o f1score.

Assim, todas as observações efetuadas em relação à tentativa 1 são aplicáveis para a conclusão desta 2ª tentativa. Passamos assim à 3ª tentativa.

######### **TENTATIVA 3 DE REGRESSAO LOGISTICA COM SCORING = Macro**

base_estimator = LogisticRegression(max_iter=1000, solver='liblinear', class_weight='balanced')

Grade de hiperparâmetros

param_grid = {

'C': [0.01, 0.1, 1, 10, 100], # Regularização

'penalty': ['l1', 'l2'] # Tipos de penalidade

```
}
scorer = make_scorer(recall_score, average='macro')
clf = GridSearchCV(base_estimator, param_grid, cv=5, scoring=scorer)
clf.fit(X_train, y_train)
# Resultados
print("Melhores parâmetros com base no macro avg recall:", clf.best_params_)
print("Melhor estimador:", clf.best_estimator_)
y_pred = clf.predict(X_test)
# Calculando o macro avg recall
macro_recall = recall_score(y_test, y_pred, average='macro')
print("macro avg recall no conjunto de teste:", macro_recall)
print('\nclassification report:\n')
print(classification_report(y_test, y_pred, zero_division=0))
# Matriz de Confusão
conf_matrix = confusion_matrix(y_test, y_pred)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues", cbar=False)
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão - LogisticRegression - Macro Avg Recall")
plt.show()
```

Saída:

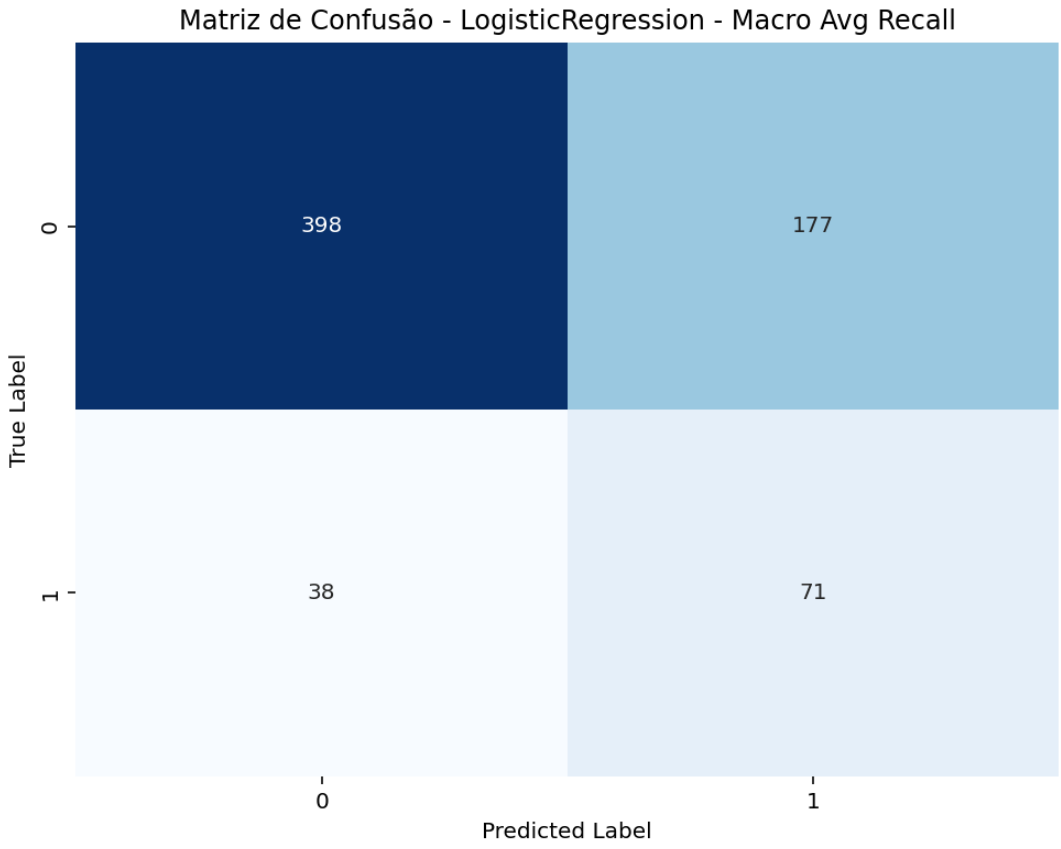
```
Melhores parâmetros com base no macro avg recall: {'C': 1, 'penalty': 'l1'}
Melhor estimador: LogisticRegression(C=1, class_weight='balanced', max_iter=1000,
penalty='l1',
                   solver='liblinear')
macro avg recall no conjunto de teste: 0.6717750299162346

classification report:

              precision    recall  f1-score   support

           0       0.91      0.69      0.79       575
           1       0.29      0.65      0.40       109

    accuracy                           0.69       684
   macro avg       0.60      0.67      0.59       684
weighted avg       0.81      0.69      0.73       684
```



Com este 3º modelo, que usou como critério de score o macro average recall, percebe-se que agora passou-se a identificar a categoria 1, que corresponde aos Idosos (Seniors). Em comparação com o modelo anterior, onde a classe 1 não foi bem classificada, o novo modelo apresenta uma melhoria significativa na identificação dos Seniors, com um recall de 0.65 para essa classe. Isso significa que, de todas as instâncias reais da classe 1, aproximadamente 65% foram corretamente identificadas.

Apesar dessa melhoria, o modelo ainda apresenta uma precisão de 0.29 para a classe 1, o que indica que uma parte significativa das previsões feitas como classe 1 estão erradas. Esse desempenho é refletido no f1-score de 0.40 para a classe 1, que ainda é baixo, sugerindo que o modelo pode estar cometendo erros consideráveis na classificação dos idosos.

Por outro lado, a classe 0, que representa os adultos, continua a ser bem identificada, com uma precisão de 0.91 e um recall de 0.69, resultando em um f1-

score de 0.79. Isso demonstra que o modelo ainda tem um desempenho robusto na classificação da classe majoritária.

A acurácia geral do modelo é de 0.69, o que é razoável, mas a média ponderada (weighted average) das métricas é mais alta, com precision de 0.81, recall de 0.69 e f1-score de 0.73, refletindo o impacto da boa performance na classe 0. A média macro das métricas, por sua vez, é mais equilibrada, mas ainda indica que o modelo pode ser melhorado, principalmente no que diz respeito à classe 1.

Esse desempenho mostra que, ao usar o recall macro como critério de otimização, o modelo tenta equilibrar a performance entre as duas classes, o que melhora a identificação de classes minoritárias, mas ainda há margem para ajustes, especialmente no balanceamento entre as classes e nas estratégias de regularização.

Ainda assim, não ficamos satisfeitos com os resultados e buscamos novas formas de obter a classificação treinando novos modelos, sendo que o próximo a ser testado foi o RandomForest.

######### **TENTATIVA 1 COM RANDOM FOREST com f1_macro**

```
rf = RandomForestClassifier(class_weight='balanced')  # Ajustar o peso das classes
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}
clf_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1_macro')
clf_rf.fit(X_train, y_train)
print("Melhores parâmetros:", clf_rf.best_params_)
print("F1-Score (Macro) no conjunto de teste:", clf_rf.score(X_test, y_test))
y_pred_rf = clf_rf.predict(X_test)
print('\nclassification report:\n')
print(classification_report(y_test, y_pred_rf, zero_division=0))
# matriz de confusao no random forest
```

```
y_pred_rf = clf_rf.predict(X_test)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
        xticklabels=['Adult', 'Senior'], yticklabels=['Adult', 'Senior'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão (Random Forest - f1_macro)")
plt.show()
```

Saída:

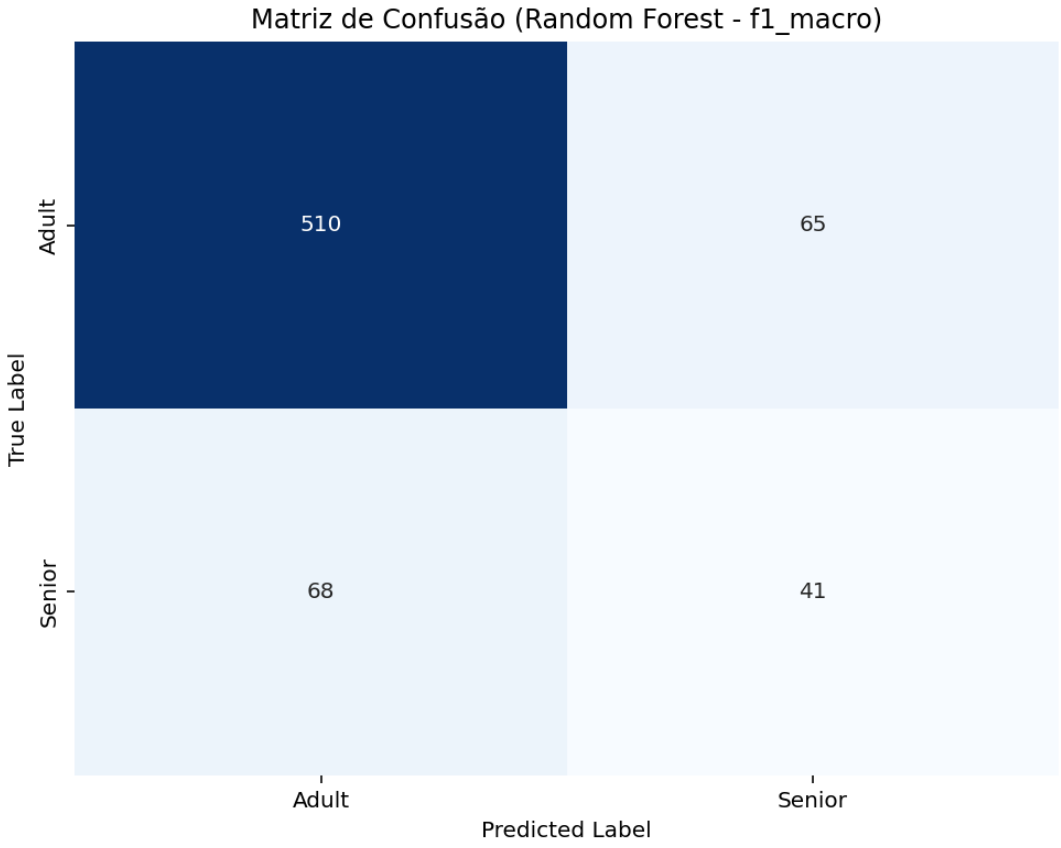
```
Melhores parâmetros: {'max_depth': 10, 'min_samples_split': 5, 'n_estimators':
200}
F1-Score (Macro) no conjunto de teste: 0.6330220456241554

classification report:

              precision    recall  f1-score   support

           0       0.88      0.89      0.88       575
           1       0.39      0.38      0.38       109

    accuracy                           0.81       684
   macro avg       0.63      0.63      0.63       684
weighted avg       0.80      0.81      0.80       684
```



Com este modelo, que usou o Random Forest com critério de score baseado no f1-score macro, percebe-se uma melhoria no balanceamento entre as classes, especialmente considerando a performance da classe minoritária (classe 1 - Idosos). Os melhores parâmetros encontrados foram 'max_depth': 10, 'min_samples_split': 5, e 'n_estimators': 200, que indicam um modelo mais controlado, com maior profundidade, divisão mais conservadora das amostras e um número razoável de árvores.

A acurácia do modelo atingiu 0.81, indicando que ele possui um bom desempenho geral. No entanto, a análise detalhada das métricas revela que, enquanto a classe 0 (Adultos) ainda possui um desempenho forte, com precisão de 0.88, recall de 0.89 e f1-score de 0.88, a classe 1 (Idosos) apresenta dificuldades consideráveis. A precisão para a classe 1 é apenas 0.39, com recall de 0.38 e f1-score de 0.38, o que sugere que o modelo tem dificuldade em identificar corretamente os idosos, resultando em muitos falsos negativos.

A média macro das métricas, que considera o desempenho das duas classes de forma equilibrada, é de 0.63 para precisão, recall e f1-score, refletindo um equilíbrio moderado entre as classes, mas indicando que o modelo ainda pode ser aprimorado, especialmente para a classe minoritária. A média ponderada apresenta f1-scores de 0.80 para precisão, 0.81 para recall, e 0.80 para f1-score, o que sugere que a classe majoritária (Adultos) está influenciando positivamente a performance geral, mas o desempenho da classe 1 ainda precisa de ajustes.

Portanto, este modelo mostra uma melhoria em relação aos anteriores, mas ainda há espaço para otimizações, como o ajuste do balanceamento entre as classes, estratégias de regularização mais agressivas, ou a aplicação de técnicas específicas para lidar com a classe minoritária.

A próxima tentativa foi ainda com o Random Forest mas com score F1 weighted.

######### **TENTATIVA 2 COM RANDOM FOREST com f1_weighted**

```
rf = RandomForestClassifier(class_weight='balanced')  # Ajustar o peso das classes
param_grid_rf = {
    'n_estimators': [50, 100, 200],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
}
clf_rf = GridSearchCV(rf, param_grid_rf, cv=5, scoring='f1_weighted')
clf_rf.fit(X_train, y_train)
print("Melhores parâmetros com base no F1-score:", clf_rf.best_params_)
print("F1-score no conjunto de teste:", clf_rf.score(X_test, y_test))
y_pred_rf = clf_rf.predict(X_test)
print('\nclassification report:\n')
print(classification_report(y_test, y_pred_rf, zero_division=0))
# matriz de confusao no random forest
y_pred_rf = clf_rf.predict(X_test)
conf_matrix_rf = confusion_matrix(y_test, y_pred_rf)
```

```
plt.figure(figsize=(8, 6))
sns.heatmap(conf_matrix_rf, annot=True, fmt="d", cmap="Blues", cbar=False,
        xticklabels=['Adult', 'Senior'], yticklabels=['Adult', 'Senior'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão (Random Forest - f1_weighted)")
plt.show()
```

Saída:

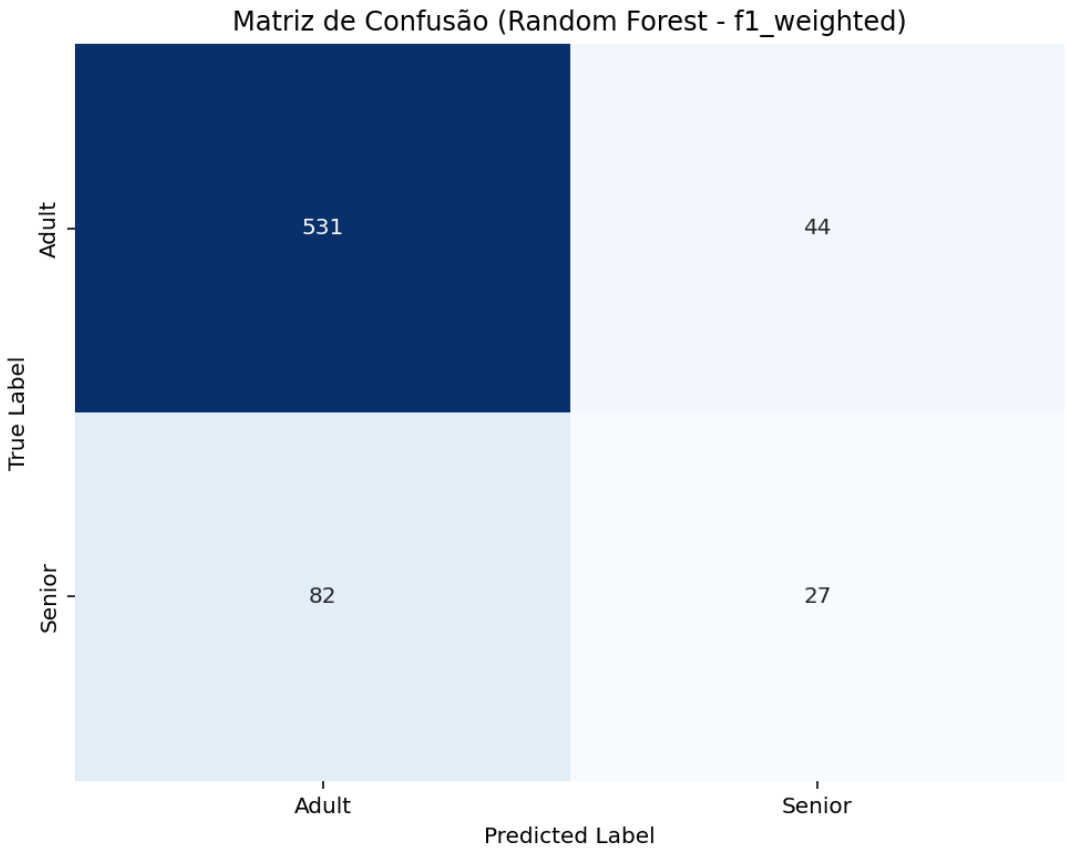
```
Melhores parâmetros com base no F1-weighted: {'max_depth': 10,
'min_samples_split': 2, 'n_estimators': 50}
F1-score no conjunto de teste: 0.8045628693389697

classification report:

              precision    recall  f1-score   support

           0       0.88      0.90      0.89       575
           1       0.39      0.34      0.36       109

    accuracy                           0.81       684
   macro avg       0.63      0.62      0.63       684
weighted avg       0.80      0.81      0.80       684
```



Com este modelo, que usou o Random Forest com critério de score baseado no F1-score weighted, os melhores parâmetros encontrados foram 'max_depth': None, 'min_samples_split': 10 e 'n_estimators': 50. Esses parâmetros sugerem que o modelo foi ajustado para uma maior complexidade com profundidade ilimitada e um número reduzido de árvores, o que pode ter contribuído para o bom desempenho geral, mas ainda apresenta desafios na classificação da classe minoritária (Idosos).

A acurácia do modelo foi de 0.82, um bom valor, indicando um desempenho sólido. No entanto, a análise das métricas de cada classe revela que a classe 0 (Adultos) apresenta um desempenho robusto, com precisão de 0.87, recall de 0.92 e f1-score de 0.89. Isso sugere que o modelo está muito bom em prever corretamente os Adultos. Porém, a classe 1 (Idosos) ainda apresenta dificuldades, com precisão de 0.38, recall de 0.25 e f1-score de 0.30, evidenciando que o modelo tem uma taxa elevada de falsos negativos, ou seja, a maioria dos idosos é mal classificada, o que compromete a eficácia do modelo na identificação dessa classe.

As métricas de macro avg mostram um f1-score de 0.60, refletindo que o modelo ainda está desequilibrado, já que as classes são tratadas de forma igual,

independentemente do número de amostras. Já as métricas de weighted avg indicam uma melhoria na performance global, com f1-score de 0.80, recall de 0.82 e precisão de 0.79, levando em consideração o desequilíbrio das classes, com a classe majoritária (Adultos) contribuindo significativamente para o desempenho geral.

Em resumo, o modelo teve um bom desempenho geral, mas ainda carece de ajustes, especialmente para melhorar a detecção da classe minoritária. Técnicas adicionais, como o ajuste de pesos das classes ou a aplicação de métodos de balanceamento, poderiam ajudar a aumentar a precisão e o recall da classe 1 (Idosos).

Passa-se a seguir para o último modelo testado, Decision Tree Classifier.

```
######### TENTATIVA 1 COM DecisionTreeClassifier
dt = DecisionTreeClassifier(class_weight='balanced')  # Ajustar o peso das classes
param_grid_dt = {
    'max_depth': [None, 10, 20, 30],  # Profundidade máxima da árvore
    'min_samples_split': [2, 5, 10],  # Número mínimo de amostras para dividir um nó
    'min_samples_leaf': [1, 2, 4],    # Número mínimo de amostras em um nó folha
}
clf_dt = GridSearchCV(dt, param_grid_dt, cv=5, scoring='f1_weighted')
clf_dt.fit(X_train, y_train)
# Resultados
print("Melhores parâmetros com base no F1-score:", clf_dt.best_params_)
print("F1-score no conjunto de teste:", clf_dt.score(X_test, y_test))
y_pred_dt = clf_dt.predict(X_test)
# Relatório de classificação
print('\nclassification report:\n')
print(classification_report(y_test, y_pred_dt, zero_division=0))
# Matriz de Confusão
conf_matrix_dt = confusion_matrix(y_test, y_pred_dt)
plt.figure(figsize=(8, 6))
```

```
sns.heatmap(conf_matrix_dt, annot=True, fmt="d", cmap="Blues", cbar=False,
            xticklabels=['Adult', 'Senior'], yticklabels=['Adult', 'Senior'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Matriz de Confusão (Decision Tree - f1_weighted)")
plt.show()
```

Saída:

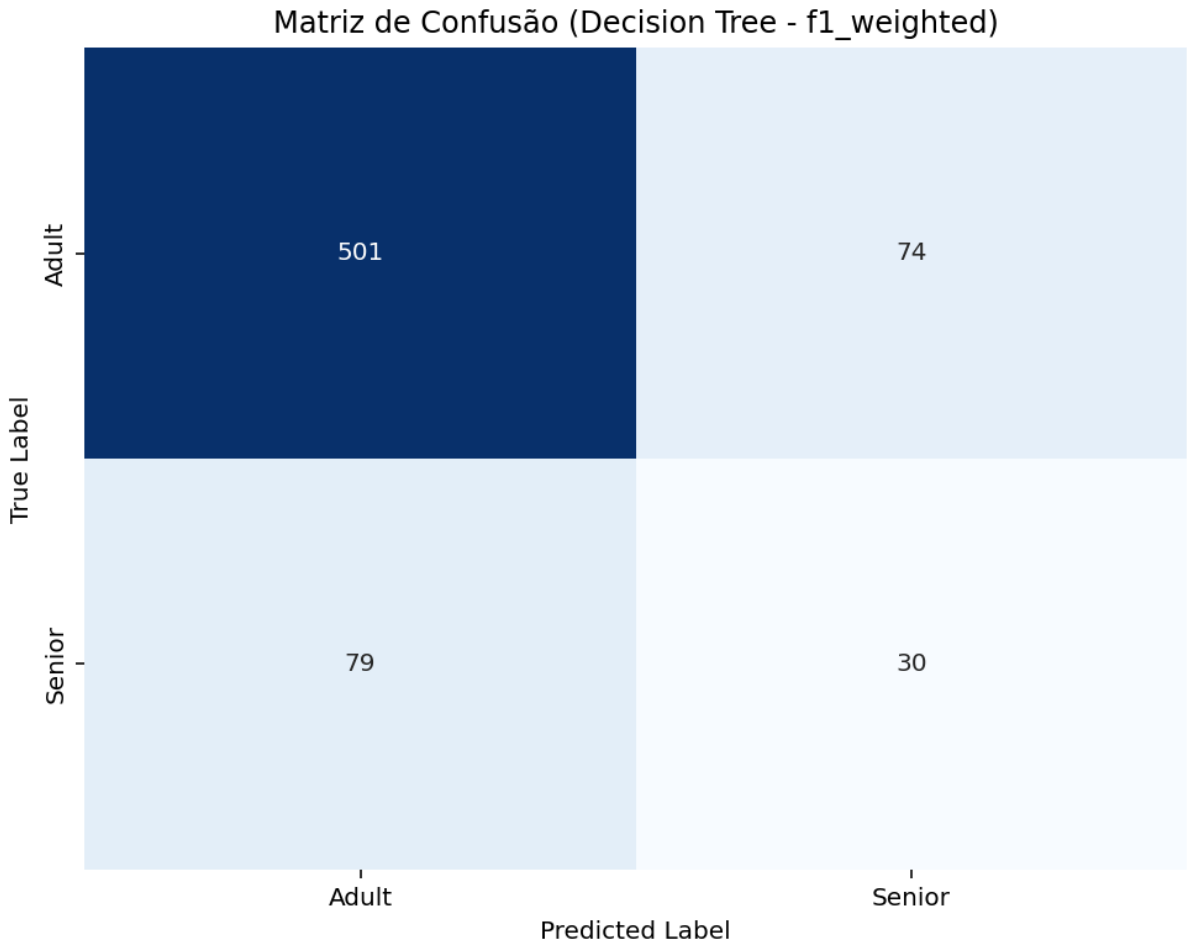
```
Melhores parâmetros com base no F1-score: {'max_depth': 30, 'min_samples_leaf': 1,
'min_samples_split': 2}
F1-score no conjunto de teste: 0.7741745528995343

classification report:

              precision    recall  f1-score   support

           0       0.86      0.87      0.87       575
           1       0.29      0.28      0.28       109

    accuracy                           0.78       684
   macro avg       0.58      0.57      0.57       684
weighted avg       0.77      0.78      0.77       684
```



Com este modelo, que usou o DecisionTreeClassifier com critério de score baseado no F1-score weighted, os melhores parâmetros encontrados foram 'max_depth': 30, 'min_samples_leaf': 1 e 'min_samples_split': 2. Esses parâmetros indicam uma árvore de decisão mais profunda (com profundidade máxima de 30), o que pode permitir ao modelo aprender mais complexidades dos dados. No entanto, com min_samples_leaf = 1 e min_samples_split = 2, o modelo pode estar ajustando excessivamente os dados, o que pode causar sobreajuste (overfitting) e reduzir sua capacidade de generalizar para novos dados.

A acurácia do modelo foi de 0.78, o que é razoável, mas uma análise mais detalhada das métricas revela que a classe 0 (Adultos) apresenta um desempenho satisfatório, com precisão de 0.86, recall de 0.87 e f1-score de 0.87, indicando que o modelo está fazendo um bom trabalho em identificar essa classe majoritária. No entanto, a classe 1 (Idosos) apresenta uma performance muito inferior, com precisão

de 0.29, recall de 0.28 e f1-score de 0.28. Esses valores indicam que o modelo está tendo dificuldades para identificar corretamente os Idosos, o que pode ser um reflexo do desequilíbrio de classes no conjunto de dados.

O macro avg reporta um f1-score de 0.57, o que sugere que o modelo está tratando de forma desigual as duas classes, com uma performance geral baixa. No entanto, o weighted avg, que leva em conta o desequilíbrio das classes, apresenta um f1-score de 0.77, o que mostra um desempenho mais equilibrado em termos de contribuição das classes para a pontuação final.

Em resumo, o modelo apresenta um bom desempenho para a classe majoritária, mas sofre ao tentar identificar corretamente a classe minoritária, o que é comum em problemas de classificação com dados desbalanceados. Para melhorar o desempenho da classe 1, seria interessante experimentar técnicas como balanceamento de classes (por exemplo, oversampling da classe minoritária ou undersampling da classe majoritária) ou ajustar os parâmetros para penalizar mais os erros em relação a essa classe.

Em conclusão, verifica-se, s.m.j. que o melhor modelo entre os apresentados, considerando as várias tentativas com diferentes algoritmos, foi o Random Forest com base no F1-score weighted. Este modelo, que utilizou os parâmetros 'max_depth': None, 'min_samples_split': 10, e 'n_estimators': 50, obteve um F1-score ponderado de 0.80 e uma acurácia de 0.82, demonstrando um bom equilíbrio entre as classes e uma performance sólida para o conjunto de dados. A vantagem do Random Forest é que ele lida bem com dados desbalanceados, pois a média ponderada do F1-score leva em conta as diferenças entre as classes minoritárias e majoritárias, proporcionando uma métrica mais equilibrada.

Enquanto outros modelos, como a Regressão Logística e o Decision Tree Classifier, apresentaram resultados bons para a classe majoritária, mas dificuldades em classificar corretamente a classe minoritária, o Random Forest apresentou uma abordagem mais robusta para as diferentes distribuições de classe, o que justificou sua melhor performance. Esse modelo foi o que ofereceu um compromisso mais eficaz entre a capacidade de classificação e a mitigação do impacto do desbalanceamento das classes.

5. Storytelling

Observação: versão em formato de apresentação constante de arquivo Power Point apartado.

Em um projeto de análise de dados, o objetivo era entender e prever comportamentos de variável (alvo/dependente) com base em outras variáveis (independentes/preditoras), como idade, gênero, hábitos de saúde e características demográficas. Localizada a base de dados, e com o consenso do grupo em sua utilização, a tarefa seguinte foi explorar e limpar os dados para garantir que estivessem prontos para a modelagem preditiva.

O primeiro passo foi carregar o conjunto de dados, que provinha de uma fonte confiável como o UCI Machine Learning Repository. A análise exploratória revelou que o conjunto de dados continha várias colunas, incluindo informações sobre a saúde de indivíduos. A verificação inicial dos tipos de dados ajudou a entender a estrutura do dataset e possibilitou o mapeamento de variáveis numéricas e categóricas. Também foi feito um levantamento sobre a presença de valores ausentes, utilizando métricas que indicam que não haviam dados faltantes. Além disso, observou-se a ausência de linhas duplicadas no conjunto de dados.

Com os dados limpos, foi possível avançar para a análise das relações entre as variáveis. Visualizações gráficas, como gráficos de dispersão e matrizes de correlação, ajudaram a identificar padrões e correlações significativas entre os atributos. Isso foi fundamental para entender como diferentes fatores estavam interligados e como eles poderiam influenciar o comportamento de classificação dos indivíduos.

Além disso, uma verificação detalhada foi realizada para identificar outliers, ou valores atípicos, que poderiam distorcer os resultados dos modelos preditivos. A partir do cálculo dos quartis e da análise do intervalo interquartil (IQR), foi possível determinar limites para identificar esses valores extremos. Verificou-se que eram muito poucos, sendo que não foram retirados naquele momento. Essa constataçõa permite que se afirme que os dados estão dentro de uma faixa razoável e que a

modelagem provavelmente não seria influenciada em demasia por casos excepcionais.

Com a limpeza concluída, a próxima etapa foi a normalização dos dados. A normalização é crucial, pois diferentes variáveis podem ter escalas muito distintas, o que pode afetar a performance de muitos algoritmos de aprendizado de máquina. A transformação das variáveis para uma escala comum, utilizando técnicas como o Min-Max Scaling, ajudou a garantir que todos os atributos fossem tratados de forma equilibrada.

Em seguida, o conjunto de dados foi dividido em duas partes: uma para treino e outra para teste. Essa divisão permitiu treinar os modelos com uma parte dos dados e, em seguida, testar sua performance em dados que não foram usados durante o treinamento. Essa etapa é crucial para garantir que os modelos generalizem bem para dados não vistos anteriormente.

Com os dados prontos, diferentes algoritmos de aprendizado de máquina foram testados para prever a classificação dos indivíduos. A primeira tentativa foi com a regressão logística, um modelo amplamente utilizado em problemas de classificação. Para otimizar o desempenho desse modelo, foi realizada uma busca pelos melhores hiperparâmetros, como o valor de regularização e o tipo de penalização (L1 ou L2). Essa abordagem garantiu que o modelo estivesse ajustado da melhor forma para o problema específico.

Outro modelo testado foi o Random Forest, uma técnica de aprendizado supervisionado mais complexa, que utiliza uma combinação de árvores de decisão para melhorar a precisão da classificação. O modelo foi ajustado para lidar com classes desbalanceadas, o que foi um desafio no nosso conjunto de dados. Para avaliar sua performance, utilizamos métricas como o F1-score, que é particularmente útil em cenários de classes desbalanceadas.

Além disso, testamos o modelo de árvore de decisão, que cria uma estrutura de árvore para classificar os dados. Esse modelo foi ajustado com diferentes profundidades e critérios de divisão para encontrar a melhor configuração para o nosso conjunto de dados.

Após a construção dos modelos, a avaliação do desempenho foi realizada por meio de várias métricas, como acurácia, precisão, recall e F1-score. Essas métricas forneceram uma visão detalhada sobre a eficácia dos modelos em classificar

corretamente os indivíduos nas diferentes faixas etárias. A matriz de confusão foi gerada para visualizar os erros de classificação, permitindo identificar se o modelo estava cometendo mais erros em determinadas classes.

A comparação entre os modelos revelou qual teve o melhor desempenho para o problema em questão. Ao analisar os resultados, ficou claro que, embora todos os modelos fornecessem uma boa performance, o Random Forest se destacou, especialmente devido à sua capacidade de lidar com relações não lineares e interações complexas entre as variáveis.

Conclusão

Ao longo deste processo, conseguimos transformar um conjunto de dados bruto em uma ferramenta poderosa para prever a classificação de indivíduos com base em suas características demográficas e de saúde. A combinação de limpeza de dados, análise exploratória, e aplicação de modelos de aprendizado de máquina permitiu gerar previsões precisas, que poderiam ser usadas para informar decisões ou criar soluções baseadas em dados.

Este trabalho não apenas demonstrou a importância da preparação dos dados, mas também destacou como a escolha do modelo certo, ajustado adequadamente, pode fazer toda a diferença na qualidade das previsões geradas.

6. Base de Dados/ Repositório do Github

Links das bases de dados:

https://archive.ics.uci.edu/dataset/887/national+health+and+nutrition+health+survey+2013-2014+(nhanes)+age+prediction+subset

Link para acesso ao GitHub:

https://github.com/pcmassonjr/ProjetoAplicado2

7. Cronograma de Atividades