

ENTENDIMENTO DO PROBLEMA - Ataque Cardíaco em Jovens e Adultos da África do Sul

O dataset abaixo foi retirado da plataforma Kaggle, pode ser encontrado no seguinte link: [Heart Attack in Youth vs Adult in South Africa Dataset](#).

Ele contém fatores de risco de ataque cardíaco relacionados à indivíduos da África do Sul. Inclui detalhes demográficos, histórico médico, hábitos de estilo de vida e medidas clínicas para avaliar resultados de ataque cardíaco.

O conjunto de dados é projetado para modelagem preditiva, análise estatística e aplicações de aprendizado de máquina em pesquisa de saúde.

```
In [2]: #importando pacotes:
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
import statsmodels.formula.api as smf
import statsmodels.api as sm

from sklearn.tree import DecisionTreeClassifier, plot_tree
from sklearn.model_selection import train_test_split
from sklearn import metrics
from scipy.stats import ks_2samp
```

Nome	Explicação
Patient_ID	Identificador único do paciente.
Age	Idade do paciente em anos.
Gender	Gênero do paciente: Masculino ou Feminino.
Cholesterol_Level	Nível de colesterol total no sangue, medido em miligramas por decilitro (mg/dL).
Blood_Pressure_Systolic	Pressão arterial sistólica do paciente, medida em milímetros de mercúrio (mmHg).
Blood_Pressure_Diastolic	Pressão arterial diastólica do paciente, medida em milímetros de mercúrio (mmHg).
Smoking_Status	Status de tabagismo do paciente: Fumante (Sim) ou Não Fumante (Não).
Alcohol_Intake	Nível de consumo de álcool: Baixo, Moderado ou Alto.
Physical_Activity	Nível de atividade física do paciente: Sedentário, Ativo ou Altamente Ativo.
Obesity_Index	Índice de Massa Corporal (IMC), que é uma medida de obesidade calculada a partir do peso e altura.
Diabetes_Status	Status de diabetes do paciente: Tem diabetes (Sim) ou Não tem diabetes (Não).

Nome	Explicação
Family_History_Heart_Disease	Histórico familiar de doenças cardíacas: Sim (se houver histórico) ou Não (se não houver).
Diet_Quality	Qualidade da dieta do paciente: Ruim, Média ou Boa.
Stress_Level	Nível de estresse do paciente: Baixo, Médio ou Alto.
Heart_Attack_History	Histórico de infarto do miocárdio: Já teve infarto (Sim) ou Nunca teve (Não).
Medication_Usage	Uso de medicação pelo paciente: Sim (usa medicação) ou Não (não usa medicação).
Triglycerides_Level	Nível de triglicerídeos no sangue, medido em miligramas por decilitro (mg/dL).
LDL_Level	Nível de LDL (lipoproteína de baixa densidade), conhecido como "colesterol ruim" (mg/dL).
HDL_Level	Nível de HDL (lipoproteína de alta densidade), conhecido como "colesterol bom" (mg/dL).
Heart_Attack_Outcome	Resultado de um ataque cardíaco: 0 (Não) ou 1 (Sim), indicando se o paciente sofreu um infarto.

```
In [4]: df=pd.read_csv('heart_attack_south_africa.csv')
df.head()
```

Out[4]:

	Patient_ID	Age	Gender	Cholesterol_Level	Blood_Pressure_Systolic	Blood_Pressure_I
0	1	76	Female	156	94	
1	2	39	Female	160	185	
2	3	85	Male	254	173	
3	4	45	Female	261	187	
4	5	48	Male	206	189	

Como temos uma variável categórica, vamos analisar o balanceamento:

```
In [6]: df['Heart_Attack_Outcome'].value_counts()
```


```
Out[6]: Heart_Attack_Outcome
1      58732
0      41268
Name: count, dtype: int64
```

Não temos aqui um problema de balanceamento, até que as categorias estão bem equilibradas.

```
In [8]: df.describe()
```

Out[8]:

	Patient_ID	Age	Cholesterol_Level	Blood_Pressure_Systolic	Blood_F
count	100000.000000	100000.000000	100000.000000	100000.000000	
mean	50000.500000	56.929210	224.578740	144.317750	
std	28867.657797	18.776713	43.316257	31.759636	
min	1.000000	25.000000	150.000000	90.000000	
25%	25000.750000	41.000000	187.000000	117.000000	
50%	50000.500000	57.000000	225.000000	144.000000	
75%	75000.250000	73.000000	262.000000	172.000000	
max	100000.000000	89.000000	299.000000	199.000000	



Através da análise descritiva acima, podemos observar que:

- **Ausência de dados faltantes:** Todas as variáveis têm a mesma quantidade de amostras (100.000), indicando que não há valores ausentes no dataset.
- **Idade:** A média de idade dos pacientes é **56,9 anos**, sendo que **25% têm 41 anos ou menos** e **25% têm 73 anos ou mais**, indicando uma amostra com predominância de adultos e idosos.
- **Pressão Arterial:**
  - **Sistólica:** Média de **144,3 mmHg**, acima do ideal (<120 mmHg).
  - **Diastólica:** Média de **89,6 mmHg**, superior ao recomendado (<80 mmHg).
  - **25% da amostra tem pressão sistólica maior que 172 mmHg**, sugerindo prevalência de hipertensão.
- **Colesterol:** O nível médio é **224,6 mg/dL**, acima do recomendado (<190 mg/dL), com **75% dos pacientes acima de 187 mg/dL**.
- **Índice de Massa Corporal (IMC):** A média é **29,0**, acima do ideal (18,5 - 24,9), sugerindo uma predominância de sobrepeso/obesidade na amostra.
- **Triglicerídeos:** Média de **174,6 mg/dL** (ideal: <150 mg/dL), com **25% dos pacientes acima de 237 mg/dL**, indicando um fator de risco significativo.
- **LDL (Colesterol Ruim):** Média de **124,3 mg/dL**, dentro do limite máximo (130 mg/dL), mas **25% da amostra ultrapassa 162 mg/dL**.
- **HDL (Colesterol Bom):** Média de **49,5 mg/dL**, acima do mínimo recomendado (>40 mg/dL), o que pode ser um fator protetor.
- **Ocorrência de Infarto:** **58,7% da amostra já sofreu um infarto**, sugerindo um grupo de alto risco cardiovascular.

Os dados indicam um perfil de pacientes com **fatores de risco significativos para doenças cardiovasculares**, como **hipertensão, colesterol elevado, obesidade e**

**triglicerídeos altos.**

## Analise Univariada:

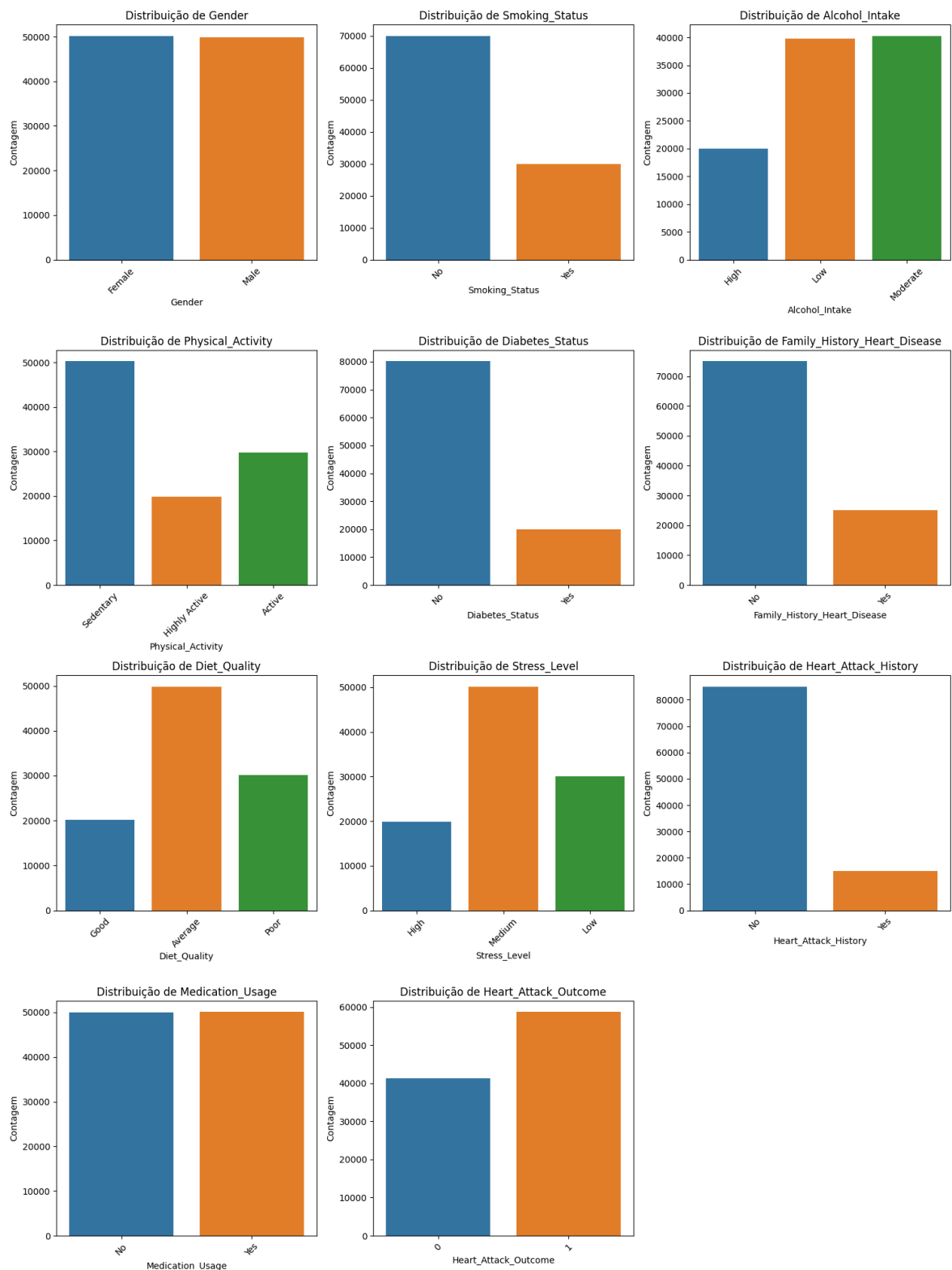
```
In [11]: def grafico_count_todas(df, limite_categorico=10):
colunas_categoricas = [col for col in df.columns if df[col].nunique() <= lim

total = len(colunas_categoricas)
linhas = (total // 3) + (total % 3 > 0)
plt.figure(figsize=(15, 5 * linhas))

for i, col in enumerate(colunas_categoricas, 1):
    plt.subplot(linhas, 3, i)
    sns.countplot(data=df, x=col, hue=col, legend=False)
    plt.title(f'Distribuição de {col}')
    plt.xlabel(col)
    plt.ylabel('Contagem')
    plt.xticks(rotation=45)

plt.tight_layout()
plt.show()

grafico_count_todas(df)
```



## Analise Bivariada:

```
In [13]: def analise_bivariada(df, target='Heart_Attack_Outcome', limite_categorico=10):
colunas_categoricas = [col for col in df.columns if df[col].nunique() <= lim
colunas_numericas = [col for col in df.select_dtypes(include=['int64', 'float64'])

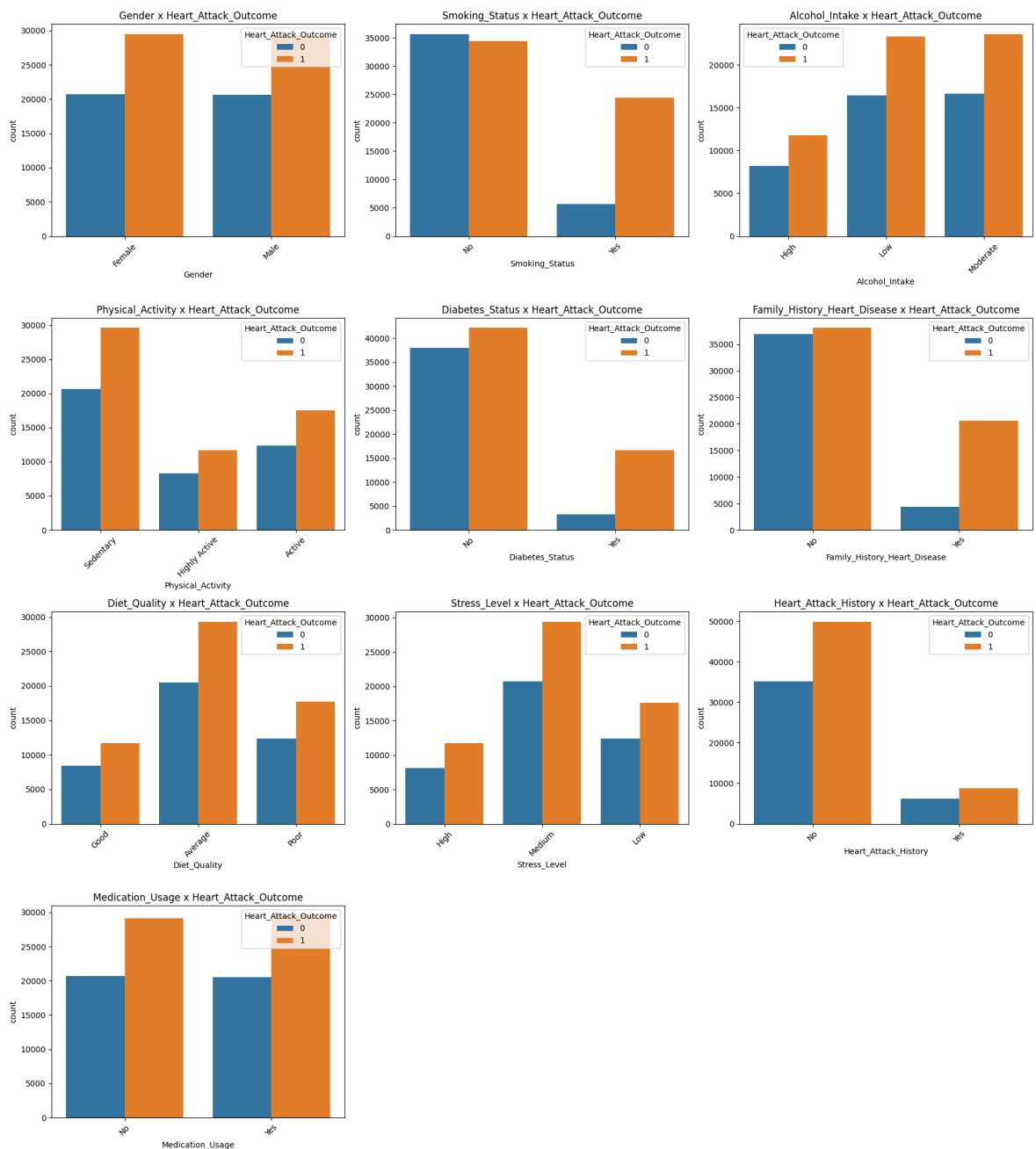
# Variáveis categóricas:
total = len(colunas_categoricas)
if total > 0:
    linhas = (total // 3) + (total % 3 > 0)
    plt.figure(figsize=(18, 5 * linhas))
    for i, col in enumerate(colunas_categoricas, 1):
```

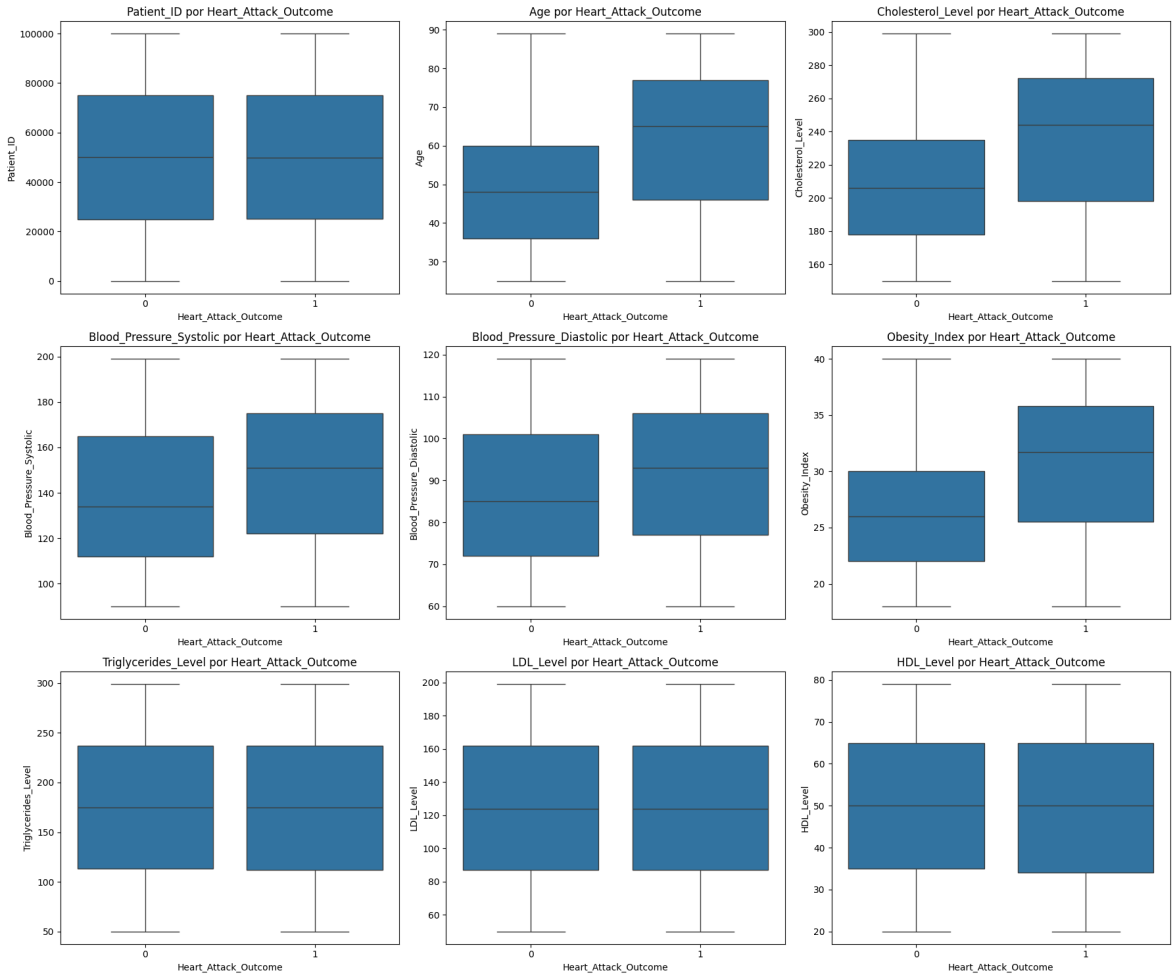
```
plt.subplot(linhas, 3, i)
sns.countplot(data=df, x=col, hue=target)
plt.title(f'{col} x {target}')
plt.xticks(rotation=45)
plt.tight_layout()
plt.show()
```

# Variáveis numéricas:

```
total = len(colunas_numericas)
if total > 0:
    linhas = (total // 3) + (total % 3 > 0)
    plt.figure(figsize=(18, 5 * linhas))
    for i, col in enumerate(colunas_numericas, 1):
        plt.subplot(linhas, 3, i)
        sns.boxplot(data=df, x=target, y=col)
        plt.title(f'{col} por {target}')
    plt.tight_layout()
    plt.show()
```

analise\_bivariada(df)





Veificação de nulos:

```
In [15]: df.isna().sum()
```

```
Out[15]: Patient_ID      0
         Age            0
         Gender         0
         Cholesterol_Level 0
         Blood_Pressure_Systolic 0
         Blood_Pressure_Diastolic 0
         Smoking_Status    0
         Alcohol_Intake    0
         Physical_Activity 0
         Obesity_Index     0
         Diabetes_Status   0
         Family_History_Heart_Disease 0
         Diet_Quality      0
         Stress_Level      0
         Heart_Attack_History 0
         Medication_Usage  0
         Triglycerides_Level 0
         LDL_Level         0
         HDL_Level         0
         Heart_Attack_Outcome 0
         dtype: int64
```

Verificação de duplicados:

```
In [17]: # Verificar duplicatas
duplicates = df[df.duplicated()]
print(duplicates)
```

Empty DataFrame

Columns: [Patient\_ID, Age, Gender, Cholesterol\_Level, Blood\_Pressure\_Systolic, Blood\_Pressure\_Diastolic, Smoking\_Status, Alcohol\_Intake, Physical\_Activity, Obesity\_Index, Diabetes\_Status, Family\_History\_Heart\_Disease, Diet\_Quality, Stress\_Level, Heart\_Attack\_History, Medication\_Usage, Triglycerides\_Level, LDL\_Level, HDL\_Level, Heart\_Attack\_Outcome]

Index: []

verificação dos tipos de dados:

```
In [19]: df.dtypes
```

```
Out[19]: Patient_ID          int64
Age          int64
Gender       object
Cholesterol_Level    int64
Blood_Pressure_Systolic  int64
Blood_Pressure_Diastolic int64
Smoking_Status    object
Alcohol_Intake    object
Physical_Activity  object
Obesity_Index     float64
Diabetes_Status    object
Family_History_Heart_Disease object
Diet_Quality       object
Stress_Level       object
Heart_Attack_History object
Medication_Usage   object
Triglycerides_Level int64
LDL_Level          int64
HDL_Level          int64
Heart_Attack_Outcome int64
dtype: object
```

Vamos verificar os valores unicos para cada variável do tipo object e substituir para numérico, pois assim facilitará a nossa análise.

```
In [21]: #célula utilizada para verificar todos os unique:
df['Family_History_Heart_Disease'].unique()
```

```
Out[21]: array(['No', 'Yes'], dtype=object)
```

Substituição dos unique para numéricos.

Aqui seguiremos um padrão:

- 0: homem, low e no.
- 1: famele, moderado/médio e yes.
- 2: high, good.

```
In [23]: # Conversão: Masculino -> 0, Feminino -> 1
df['Gender'] = df['Gender'].map({'Male': 0, 'Female': 1})
```



```

# Conversão: Fumante -> 1, Não Fumante -> 0
df['Smoking_Status'] = df['Smoking_Status'].map({'Yes': 1, 'No': 0})

# Conversão: Low -> 0, Moderate -> 1, high -> 2
df['Alcohol_Intake'] = df['Alcohol_Intake'].map({'Low': 0, 'Moderate': 1, 'High': 2})

# Conversão: Sedentary -> 0, Active -> 1, Highly Active -> 2
df['Physical_Activity'] = df['Physical_Activity'].map({'Sedentary': 0, 'Active': 1, 'Highly Active': 2})

# Conversão: no -> 0, yes -> 1
df['Diabetes_Status'] = df['Diabetes_Status'].map({'No': 0, 'Yes': 1})

# Conversão: no -> 0, yes -> 1
df['Family_History_Heart_Disease'] = df['Family_History_Heart_Disease'].map({'No': 0, 'Yes': 1})

# Conversão: Good -> 0, Average -> 1, Poor -> 2
df['Diet_Quality'] = df['Diet_Quality'].map({'Poor': 0, 'Average': 1, 'Good': 2})

# Conversão: Good -> 0, Average -> 1, Poor -> 2
df['Stress_Level'] = df['Stress_Level'].map({'Low': 0, 'Medium': 1, 'High': 2})

# Conversão: no -> 0, yes -> 1
df['Heart_Attack_History'] = df['Heart_Attack_History'].map({'No': 0, 'Yes': 1})

# Conversão: no -> 0, yes -> 1
df['Medication_Usage'] = df['Medication_Usage'].map({'No': 0, 'Yes': 1})

# Exibir o DataFrame
df.head()

```

Out[23]:

	Patient_ID	Age	Gender	Cholesterol_Level	Blood_Pressure_Systolic	Blood_Pressure_Diastolic
--	------------	-----	--------	-------------------	-------------------------	--------------------------

0	1	76	1	156	94	80
1	2	39	1	160	185	105
2	3	85	0	254	173	100
3	4	45	1	261	187	105
4	5	48	0	206	189	105



Verificação novamente dos tipos:

In [25]:

```
df.dtypes
```

```
Out[25]: Patient_ID      int64
         Age             int64
         Gender          int64
         Cholesterol_Level int64
         Blood_Pressure_Systolic int64
         Blood_Pressure_Diastolic int64
         Smoking_Status   int64
         Alcohol_Intake   int64
         Physical_Activity int64
         Obesity_Index     float64
         Diabetes_Status   int64
         Family_History_Heart_Disease int64
         Diet_Quality      int64
         Stress_Level      int64
         Heart_Attack_History int64
         Medication_Usage  int64
         Triglycerides_Level int64
         LDL_Level         int64
         HDL_Level         int64
         Heart_Attack_Outcome int64
         dtype: object
```

Podemos deletar a variável que traz o ID do paciente, pois essa não tem poder preditivo:

```
In [27]: df.drop('Patient_ID', axis=1, inplace=True)
```

Agora todas as nossas variáveis são numéricas. A seguir, exibiremos a matriz de correlação:

```
In [29]: # Calcular a correlação entre as variáveis numéricas
         correlation_matrix = df.corr()

         # Exibir a matriz de correlação
         correlation_matrix
```

Out[29]:

	Age	Gender	Cholesterol_Level	Blood_Pressure_Sys
Age	1.000000	-0.000265	-0.000364	3.672133
Gender	-0.000265	1.000000	0.001288	1.561311
Cholesterol_Level	-0.000364	0.001288	1.000000	-1.240133
Blood_Pressure_Systolic	0.003672	0.001561	-0.001240	1.000000
Blood_Pressure_Diastolic	-0.001093	-0.000438	0.005179	-1.568419
Smoking_Status	-0.001992	0.003381	-0.002368	-2.310456
Alcohol_Intake	0.001440	0.000312	-0.001211	3.378105
Physical_Activity	0.001244	-0.002084	0.001546	-2.231550
Obesity_Index	-0.001118	0.001526	-0.000109	-3.730381
Diabetes_Status	0.004342	0.001644	-0.002158	-5.901092
Family_History_Heart_Disease	0.001038	0.002607	0.002946	2.520226
Diet_Quality	-0.003163	-0.001768	0.001665	3.930109
Stress_Level	0.000031	-0.000681	0.000613	4.228029
Heart_Attack_History	0.001438	0.001375	0.000608	1.777125
Medication_Usage	-0.000674	0.003214	0.002553	-7.560601
Triglycerides_Level	0.001992	0.001929	0.001935	-2.962143
LDL_Level	0.001466	0.001228	-0.000127	6.897071
HDL_Level	-0.001979	-0.003194	-0.002934	2.398179
Heart_Attack_Outcome	0.298683	0.001068	0.283860	1.485734

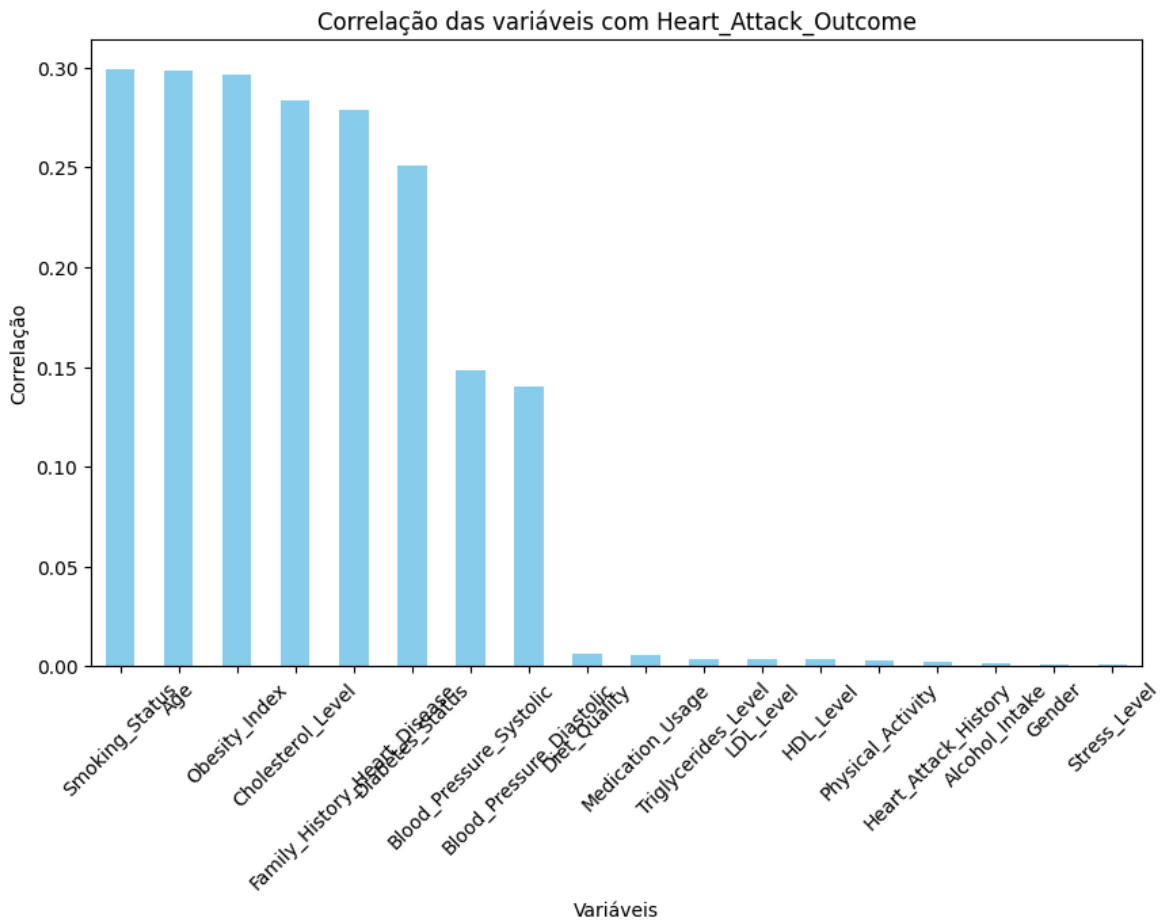
A tabela acima está um pouco extensa, por isso, iremos plotar um gráfico que também traz essa correlação e nos permite verificar melhor:

```
In [31]: # Calcular a correlação de todas as variáveis com 'Heart_Attack_Outcome'
correlation_with_heart_attack = df.corr()['Heart_Attack_Outcome'].drop('Heart_Attack_Outcome')

# Ordenar as correlações de forma decrescente
sorted_correlation = correlation_with_heart_attack.abs().sort_values(ascending=False)

# Plotar um gráfico de barras das variáveis com maior correlação
```

```
plt.figure(figsize=(10, 6))
sorted_correlation.plot(kind='bar', color='skyblue')
plt.title('Correlação das variáveis com Heart_Attack_Outcome')
plt.xlabel('Variáveis')
plt.ylabel('Correlação')
plt.xticks(rotation=45)
plt.show()
```

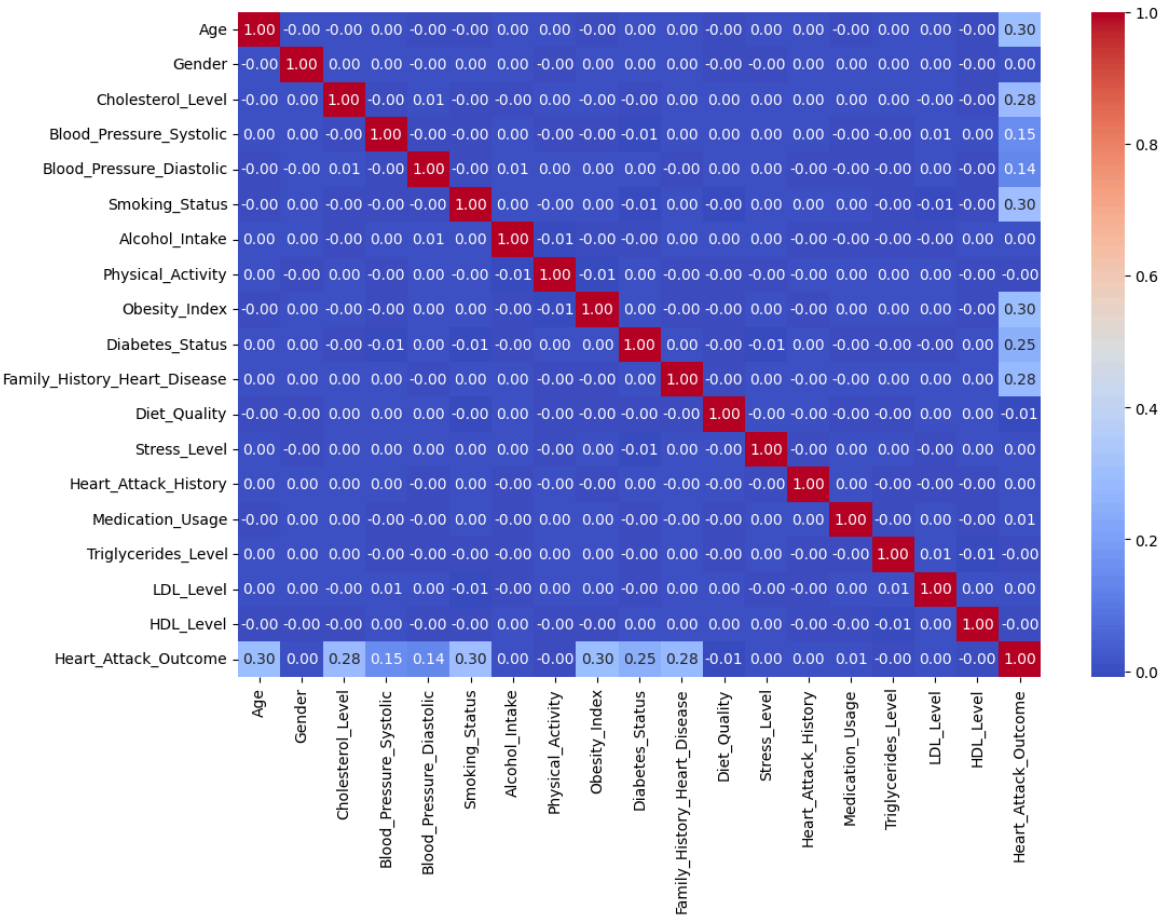


De acordo com o gráfico acima, temos como maiores correlação com a nossa target as seguintes variáveis:

- Age → 0.2987 (correlação positiva)
- Smoking\_Status → 0.2990 (correlação positiva)
- Cholesterol\_Level → 0.2839 (correlação positiva)
- Obesity\_Index → 0.2964 (correlação positiva)
- Diabetes\_Status → 0.2508 (correlação positiva)
- Family\_History\_Heart\_Disease → 0.2789 (correlação positiva)
- Blood\_Pressure\_Systolic → 0.1486 (correlação positiva)
- Blood\_Pressure\_Diastolic → 0.1402 (correlação positiva)

A seguir, vamos plotar o heatmap da nossa matriz de correlação para identificar possíveis casos de multicolineariedade:

```
In [34]: #matriz de correlação:
plt.figure(figsize=(12,8))
sns.heatmap(df.corr(), annot=True, cmap='coolwarm', fmt=".2f")
plt.show()
```



Através do plot acima podemos ver que não temos multicolineariedade no nosso df. A seguir criar um metadados para ver o IV das variáveis:

```
In [36]: metadados = pd.DataFrame(df.dtypes, columns=['dtype'])
metadados['valores_unicos'] = df.nunique()
metadados['papel'] = 'covariavel'
metadados.loc['Heart_Attack_Outcome', 'papel'] = 'resposta'
metadados
```

Out[36]:

	dtype	valores_unicos	papel
Age	int64	65	covariavel
Gender	int64	2	covariavel
Cholesterol_Level	int64	150	covariavel
Blood_Pressure_Systolic	int64	110	covariavel
Blood_Pressure_Diastolic	int64	60	covariavel
Smoking_Status	int64	2	covariavel
Alcohol_Intake	int64	3	covariavel
Physical_Activity	int64	3	covariavel
Obesity_Index	float64	221	covariavel
Diabetes_Status	int64	2	covariavel
Family_History_Heart_Disease	int64	2	covariavel
Diet_Quality	int64	3	covariavel
Stress_Level	int64	3	covariavel
Heart_Attack_History	int64	2	covariavel
Medication_Usage	int64	2	covariavel
Triglycerides_Level	int64	250	covariavel
LDL_Level	int64	150	covariavel
HDL_Level	int64	60	covariavel
Heart_Attack_Outcome	int64	2	resposta

```
In [37]: #função cálculo IV
def IV(variavel, resposta):
    tab = pd.crosstab(variavel, resposta, margins=True, margins_name='total')

    rótulo_evento = tab.columns[0]
    rótulo_nao_evento = tab.columns[1]

    tab['pct_evento'] = tab[rótulo_evento]/tab.loc['total',rótulo_evento]
    tab['ep'] = tab[rótulo_evento]/tab.loc['total',rótulo_evento]

    tab['pct_nao_evento'] = tab[rótulo_nao_evento]/tab.loc['total',rótulo_nao_ev
    tab['woe'] = np.log(tab.pct_evento/tab.pct_nao_evento)
    tab['iv_parcial'] = (tab.pct_evento - tab.pct_nao_evento)*tab.woe
    return tab['iv_parcial'].sum()
```

```
In [38]: #Loop:
for var in metadados[metadados.papel=='covariavel'].index:
    if (metadados.loc[var, 'valores_unicos']>6): #para mais que 6 valores unico
        metadados.loc[var, 'IV'] = IV(pd.qcut(df[var],5,duplicates='drop'), df.H
    else:
        metadados.loc[var, 'IV'] = IV(df[var], df.Heart_Attack_Outcome)
```

metadados

Out[38]:

	dtype	valores_unicos	papel	IV
Age	int64	65	covariavel	0.462877
Gender	int64	2	covariavel	0.000005
Cholesterol_Level	int64	150	covariavel	0.511672
Blood_Pressure_Systolic	int64	110	covariavel	0.101007
Blood_Pressure_Diastolic	int64	60	covariavel	0.087411
Smoking_Status	int64	2	covariavel	0.417627
Alcohol_Intake	int64	3	covariavel	0.000037
Physical_Activity	int64	3	covariavel	0.000037
Obesity_Index	float64	221	covariavel	0.439663
Diabetes_Status	int64	2	covariavel	0.308958
Family_History_Heart_Disease	int64	2	covariavel	0.372500
Diet_Quality	int64	3	covariavel	0.000218
Stress_Level	int64	3	covariavel	0.000025
Heart_Attack_History	int64	2	covariavel	0.000015
Medication_Usage	int64	2	covariavel	0.000123
Triglycerides_Level	int64	250	covariavel	0.000112
LDL_Level	int64	150	covariavel	0.000061
HDL_Level	int64	60	covariavel	0.000179
Heart_Attack_Outcome	int64	2	resposta	NaN

Geralmente, utiliza-se no modelo, as variáveis com IV maior que 2%, seriam elas:

- Age;
- Cholesterol\_Level;
- Blood\_Pressure\_Systolic;
- Blood\_Pressure\_Diastolic;
- Smoking\_Status;
- Obesity\_Index;
- Diabetes\_Status;
- Family\_History\_Heart\_Disease.

Treino e Teste:

```
In [41]: # Definir X (features) e y (target)
X = df.drop(columns=['Heart_Attack_Outcome']) # Remove a variável alvo
y = df['Heart_Attack_Outcome'] # Define a variável alvo
```

```
# Separar em treino (80%) e teste (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_

# Criar DataFrames completos de treino e teste
df_train = pd.concat([X_train, y_train], axis=1)
df_test = pd.concat([X_test, y_test], axis=1)

# Exibir tamanhos das amostras
print(f"Tamanho do df_train: {df_train.shape}")
print(f"Tamanho do df_test: {df_test.shape}")
```

Tamanho do df\_train: (80000, 19)

Tamanho do df\_test: (20000, 19)

## Equação de Regressão:

```
In [43]: # ajuda para definir a equação da regressão
' + '.join(list(df.columns))
```

```
Out[43]: 'Age + Gender + Cholesterol_Level + Blood_Pressure_Systolic + Blood_Pressure_Di
astolic + Smoking_Status + Alcohol_Intake + Physical_Activity + Obesity_Index +
Diabetes_Status + Family_History_Heart_Disease + Diet_Quality + Stress_Level +
Heart_Attack_History + Medication_Usage + Triglycerides_Level + LDL_Level + HDL
_Level + Heart_Attack_Outcome'
```

```
In [44]: formula= '''
Heart_Attack_Outcome ~ Age + Gender + Cholesterol_Level + Blood_Pressure_Systoli
'''
r1 = smf.glm(formula, data=df_train, family=sm.families.Binomial()).fit()

r1.summary()
```



Out[44]:

## Generalized Linear Model Regression Results

<b>Dep. Variable:</b>	Heart_Attack_Outcome	<b>No. Observations:</b>	80000
<b>Model:</b>	GLM	<b>Df Residuals:</b>	79981
<b>Model Family:</b>	Binomial	<b>Df Model:</b>	18
<b>Link Function:</b>	Logit	<b>Scale:</b>	1.0000
<b>Method:</b>	IRLS	<b>Log-Likelihood:</b>	-21677.
<b>Date:</b>	Thu, 10 Apr 2025	<b>Deviance:</b>	43354.
<b>Time:</b>	20:36:21	<b>Pearson chi2:</b>	5.38e+04
<b>No. Iterations:</b>	7	<b>Pseudo R-squ. (CS):</b>	0.5568
<b>Covariance Type:</b>	nonrobust		

	coef	std err	z	P> z	[0.025	0.975]
<b>Intercept</b>	-31.1586	0.258	-120.689	0.000	-31.665	-30.653
<b>Age</b>	0.0900	0.001	101.856	0.000	0.088	0.092
<b>Gender</b>	-0.0171	0.024	-0.703	0.482	-0.065	0.031
<b>Cholesterol_Level</b>	0.0375	0.000	99.947	0.000	0.037	0.038
<b>Blood_Pressure_Systolic</b>	0.0276	0.000	64.970	0.000	0.027	0.028
<b>Blood_Pressure_Diastolic</b>	0.0467	0.001	60.889	0.000	0.045	0.048
<b>Smoking_Status</b>	3.8858	0.038	102.713	0.000	3.812	3.960
<b>Alcohol_Intake</b>	0.0013	0.016	0.080	0.936	-0.031	0.033
<b>Physical_Activity</b>	-0.0056	0.016	-0.358	0.720	-0.036	0.025
<b>Obesity_Index</b>	0.2670	0.003	102.010	0.000	0.262	0.272
<b>Diabetes_Status</b>	3.8985	0.043	90.917	0.000	3.814	3.983
<b>Family_History_Heart_Disease</b>	3.8907	0.040	97.576	0.000	3.813	3.969
<b>Diet_Quality</b>	-0.0212	0.017	-1.221	0.222	-0.055	0.013
<b>Stress_Level</b>	-0.0088	0.017	-0.505	0.613	-0.043	0.025
<b>Heart_Attack_History</b>	0.0038	0.034	0.112	0.911	-0.063	0.070
<b>Medication_Usage</b>	0.0374	0.024	1.538	0.124	-0.010	0.085
<b>Triglycerides_Level</b>	-0.0003	0.000	-1.681	0.093	-0.001	4.69e-05
<b>LDL_Level</b>	3.278e-06	0.000	0.012	0.991	-0.001	0.001
<b>HDL_Level</b>	0.0003	0.001	0.359	0.720	-0.001	0.002

Obtivemos aqui, que as variáveis estatisticamente significativas para o modelo são as mesmas já citadas anteriormente ao analisar o Information Value, são elas:

- Age;
- Cholesterol\_Level;

- Blood\_Pressure\_Systolic;
- Blood\_Pressure\_Diastolic;
- Smoking\_Status;
- Obesity\_Index;
- Diabetes\_Status;
- Family\_History\_Heart\_Disease.

Portanto, criaremos o próximo modelo utilizando essas variáveis:

```
In [46]: formula= '''
Heart_Attack_Outcome ~ Age + Cholesterol_Level + Blood_Pressure_Systolic + Blood
'''
r2 = smf.glm(formula, data=df_train, family=sm.families.Binomial()).fit()
r2.summary()
```

Out[46]:

#### Generalized Linear Model Regression Results

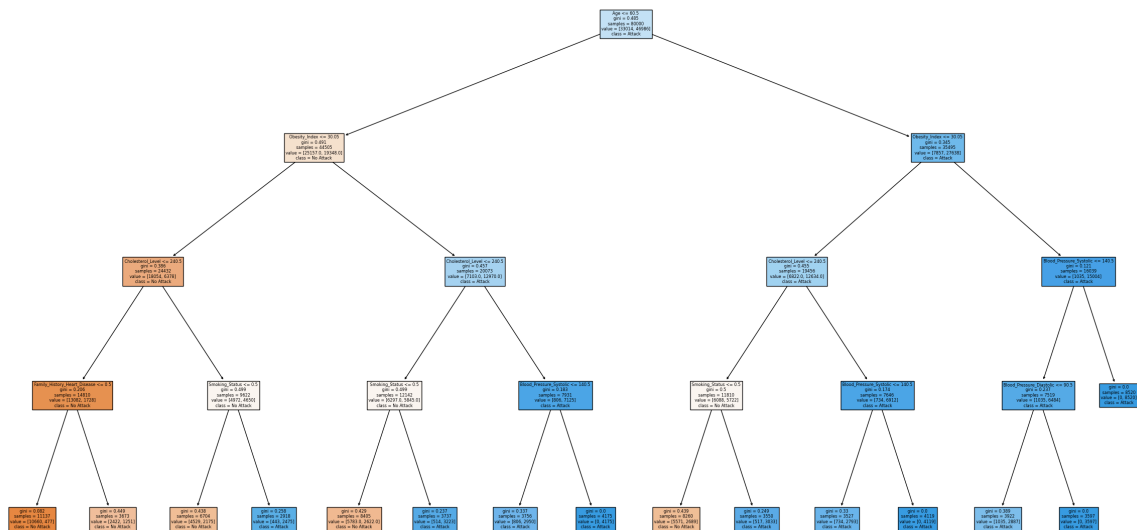
Dep. Variable:	Heart_Attack_Outcome	No. Observations:	80000
Model:	GLM	Df Residuals:	79991
Model Family:	Binomial	Df Model:	8
Link Function:	Logit	Scale:	1.0000
Method:	IRLS	Log-Likelihood:	-21681.
Date:	Thu, 10 Apr 2025	Deviance:	43362.
Time:	20:36:22	Pearson chi2:	5.37e+04
No. Iterations:	7	Pseudo R-squ. (CS):	0.5567
Covariance Type:	nonrobust		
	coef	std err	z P> z  [0.025 0.975]
Intercept	-31.2094	0.250	-124.981 0.000 -31.699 -30.720
Age	0.0900	0.001	101.861 0.000 0.088 0.092
Cholesterol_Level	0.0375	0.000	99.963 0.000 0.037 0.038
Blood_Pressure_Systolic	0.0276	0.000	64.971 0.000 0.027 0.028
Blood_Pressure_Diastolic	0.0467	0.001	60.886 0.000 0.045 0.048
Smoking_Status	3.8860	0.038	102.726 0.000 3.812 3.960
Obesity_Index	0.2670	0.003	102.023 0.000 0.262 0.272
Diabetes_Status	3.8974	0.043	90.919 0.000 3.813 3.981
Family_History_Heart_Disease	3.8903	0.040	97.587 0.000 3.812 3.968

A seguir, utilizaremos uma classificação em árvore com as variáveis acima para observarmos quais dessas variáveis ela traz como principais nós de decisão a fim de as utilizarmos com o objetivo de validar ou simplificar o nosso modelo.

```
In [48]: # Separando features e target
X_train = df_train[['Age', 'Cholesterol_Level', 'Blood_Pressure_Systolic', 'Blood_Pressure_Diastolic', 'Smoking_Status', 'Obesity_Index', 'Diabetes_Status', 'Family_History']]
y_train = df_train['Heart_Attack_Outcome']

# Criando e treinando a árvore
tree_model = DecisionTreeClassifier(max_depth=4, random_state=42) # Limitando a profundidade da árvore
tree_model.fit(X_train, y_train)

# Visualizando a árvore
plt.figure(figsize=(30, 16))
plot_tree(tree_model, feature_names=X_train.columns, class_names=['No Attack', 'Heart Attack'])
plt.show()
```



Nas três primeiras camadas da nossa árvore a gente encontra Idade (nó raiz), Obesidade, Colesterol e Pressão Sistólica, respectivamente nessa ordem de importância. Vamos tentar um novo modelo de regressão com as 3 variáveis principais.

```
In [50]: formula= ''
Heart_Attack_Outcome ~ Age + Cholesterol_Level + Obesity_Index
'''
r3 = smf.glm(formula, data=df_train, family=sm.families.Binomial()).fit()
r3.summary()
```

Out[50]:

Generalized Linear Model Regression Results							
Dep. Variable:		Heart_Attack_Outcome		No. Observations:		80000	
Model:		GLM		Df Residuals:		79996	
Model Family:		Binomial		Df Model:		3	
Link Function:		Logit		Scale:		1.0000	
Method:		IRLS		Log-Likelihood:		-42466.	
Date:		Thu, 10 Apr 2025		Deviance:		84932.	
Time:		20:36:24		Pearson chi2:		7.79e+04	
No. Iterations:		5		Pseudo R-squ. (CS):		0.2547	
Covariance Type:		nonrobust					
		coef	std err	z	P> z	[0.025	0.975]
Intercept		-9.5448	0.079	-120.190	0.000	-9.700	-9.389
Age		0.0426	0.000	88.166	0.000	0.042	0.044
Cholesterol_Level		0.0177	0.000	85.133	0.000	0.017	0.018
Obesity_Index		0.1251	0.001	87.801	0.000	0.122	0.128

Avaliando as métricas:

Modelo 1: Todas as variáveis:

Treino:

```
In [53]: # Fazer previsões no conjunto de treino
df_train['score'] = r1.predict(df_train)

# Acurácia (usando threshold de 0.5 para classificação)
acc = metrics.accuracy_score(df_train.Heart_Attack_Outcome, df_train.score > 0.5)

# AUC
fpr, tpr, thresholds = metrics.roc_curve(df_train.Heart_Attack_Outcome, df_train.score)
auc = metrics.auc(fpr, tpr)

# Gini (2*AUC -1)
gini = 2 * auc - 1

# KS (Kolmogorov-Smirnov)
ks = ks_2samp(
    df_train.loc[df_train.Heart_Attack_Outcome == 1, 'score'],
    df_train.loc[df_train.Heart_Attack_Outcome == 0, 'score']
).statistic

# Exibir métricas
print('Acurácia (Treino): {0:.1%} \nAUC (Treino): {1:.1%} \nGINI (Treino): {2:.1%} \nKS (Treino): {3:.1%}'
      .format(acc, auc, gini, ks))
```

Acurácia (Treino): 87.4%  
 AUC (Treino): 95.3%  
 GINI (Treino): 90.6%  
 KS (Treino): 74.8%

### Teste:

```
In [55]: # Obter as previsões no conjunto de teste
df_test.loc[:, 'score'] = r1.predict(df_test)

# Acurácia
acc_test = metrics.accuracy_score(df_test.Heart_Attack_Outcome, df_test.score > 0.5)

# AUC
fpr_test, tpr_test, thresholds_test = metrics.roc_curve(df_test.Heart_Attack_Outcome, df_test.score)
auc_test = metrics.auc(fpr_test, tpr_test)

# GINI
gini_test = 2 * auc_test - 1

# KS
ks_test = ks_2samp(
    df_test.loc[df_test.Heart_Attack_Outcome == 1, 'score'],
    df_test.loc[df_test.Heart_Attack_Outcome == 0, 'score']
).statistic

# Imprimir as métricas para o conjunto de teste
print('Acurácia (Teste): {0:.1%} \nAUC (Teste): {1:.1%} \nGINI (Teste): {2:.1%} \nKS (Teste): {3:.1%}'
      .format(acc_test, auc_test, gini_test, ks_test))
```

Acurácia (Teste): 87.7%  
 AUC (Teste): 95.5%  
 GINI (Teste): 91.1%  
 KS (Teste): 75.4%

## Modelo 2: Variáveis com IV (Information Value) > 2%

### Treino:

```
In [57]: # Fazer previsões no conjunto de treino
df_train['score'] = r2.predict(df_train)

# Acurácia (usando threshold de 0.5 para classificação)
acc = metrics.accuracy_score(df_train.Heart_Attack_Outcome, df_train.score > 0.5)

# AUC
fpr, tpr, thresholds = metrics.roc_curve(df_train.Heart_Attack_Outcome, df_train.score)
auc = metrics.auc(fpr, tpr)

# Gini (2*AUC - 1)
gini = 2 * auc - 1

# KS (Kolmogorov-Smirnov)
ks = ks_2samp(
    df_train.loc[df_train.Heart_Attack_Outcome == 1, 'score'],
    df_train.loc[df_train.Heart_Attack_Outcome == 0, 'score']
).statistic

# Exibir métricas
```

```
print('Acurácia (Treino): {0:.1%} \nAUC (Treino): {1:.1%} \nGINI (Treino): {2:.1%} \nKS (Treino): {3:.1%}'
      .format(acc, auc, gini, ks))
```

Acurácia (Treino): 87.4%

AUC (Treino): 95.3%

GINI (Treino): 90.6%

KS (Treino): 74.8%

### Teste:

```
In [59]: # Obter as previsões no conjunto de teste
df_test.loc[:, 'score'] = r2.predict(df_test)

# Acurácia
acc_test = metrics.accuracy_score(df_test.Heart_Attack_Outcome, df_test.score)

# AUC
fpr_test, tpr_test, thresholds_test = metrics.roc_curve(df_test.Heart_Attack_Outcome, df_test.score)
auc_test = metrics.auc(fpr_test, tpr_test)

# GINI
gini_test = 2 * auc_test - 1

# KS
ks_test = ks_2samp(
    df_test.loc[df_test.Heart_Attack_Outcome == 1, 'score'],
    df_test.loc[df_test.Heart_Attack_Outcome == 0, 'score'],
).statistic

# Imprimir as métricas para o conjunto de teste
print('Acurácia (Teste): {0:.1%} \nAUC (Teste): {1:.1%} \nGINI (Teste): {2:.1%} \nKS (Teste): {3:.1%}'
      .format(acc_test, auc_test, gini_test, ks_test))
```

Acurácia (Teste): 87.7%

AUC (Teste): 95.5%

GINI (Teste): 91.1%

KS (Teste): 75.3%

## Modelo 3: 3 maiores preditoras de acordo com a árvore de decisão:

### Treino:

```
In [61]: # Fazer previsões no conjunto de treino
df_train['score'] = r3.predict(df_train)

# Acurácia (usando threshold de 0.5 para classificação)
acc = metrics.accuracy_score(df_train.Heart_Attack_Outcome, df_train.score > 0.5)

# AUC
fpr, tpr, thresholds = metrics.roc_curve(df_train.Heart_Attack_Outcome, df_train.score)
auc = metrics.auc(fpr, tpr)

# Gini (2*AUC - 1)
gini = 2 * auc - 1

# KS (Kolmogorov-Smirnov)
ks = ks_2samp(
    df_train.loc[df_train.Heart_Attack_Outcome == 1, 'score'],
    df_train.loc[df_train.Heart_Attack_Outcome == 0, 'score'],
).statistic
```

```
# Exibir métricas
print('Acurácia (Treino): {0:.1%} \nAUC (Treino): {1:.1%} \nGINI (Treino): {2:.1%} \nKS (Treino): {3:.1%}'
      .format(acc, auc, gini, ks))
```

Acurácia (Treino): 72.5%  
AUC (Treino): 80.1%  
GINI (Treino): 60.1%  
KS (Treino): 44.8%

Teste:

```
In [63]: # Obter as previsões no conjunto de teste
df_test.loc[:, 'score'] = r2.predict(df_test)

# Acurácia
acc_test = metrics.accuracy_score(df_test.Heart_Attack_Outcome, df_test.score)

# AUC
fpr_test, tpr_test, thresholds_test = metrics.roc_curve(df_test.Heart_Attack_Outcome, df_test.score)
auc_test = metrics.auc(fpr_test, tpr_test)

# GINI
gini_test = 2 * auc_test - 1

# KS
ks_test = ks_2samp(
    df_test.loc[df_test.Heart_Attack_Outcome == 1, 'score'],
    df_test.loc[df_test.Heart_Attack_Outcome == 0, 'score'],
).statistic

# Imprimir as métricas para o conjunto de teste
print('Acurácia (Teste): {0:.1%} \nAUC (Teste): {1:.1%} \nGINI (Teste): {2:.1%} \nKS (Teste): {3:.1%}'
      .format(acc_test, auc_test, gini_test, ks_test))
```

Acurácia (Teste): 87.7%  
AUC (Teste): 95.5%  
GINI (Teste): 91.1%  
KS (Teste): 75.3%

Conclusão:

Métrica	Modelo 1 (18 variáveis)	Modelo 2 (8 variáveis)	Modelo 3 (3 variáveis)
Acurácia (Treino)	87.4%	87.4%	72.5%
AUC (Treino)	95.3%	95.3%	80.1%
GINI (Treino)	90.6%	90.6%	60.1%
KS (Treino)	74.8%	74.8%	44.8%
-----	-----	-----	-----
--	-	--	--
Acurácia (Teste)	87.7%	87.7%	87.7%
AUC (Teste)	95.5%	95.5%	95.5%
GINI (Teste)	91.1%	91.1%	91.1%

Métrica	Modelo 1 (18 variáveis)	Modelo 2 (8 variáveis)	Modelo 3 (3 variáveis)
KS (Teste)	75.4%	75.3%	75.3%

Modelo 1:

- 18 variáveis;
- Alto desempenho em todas as métricas;
- Difícil interpretar por conta do alto número de variáveis;

Modelo 2:

- 8 variáveis;
- Alto desempenho em todas as métricas;
- Mais simples e interpretável que o modelo 1 e mantém a mesma performance.

Modelo 3:

- 3 variáveis;
- Queda brusca no treino em relação aos modelos anteriores;
- Performance estranhamente boa para teste visto que esse desempenho é muito menor para treino;
- Arriscado para um problema sério como ataque cardíaco.

A nossa escolha final será o Modelo 2, uma vez que é interpretável, simples e tem uma boa performance com alta acurácia, excelente capacidade preditiva (AUC e GINI elevados) e boa separação das classes (KS alto).

In [ ]: