

Pós Tech Fiap - Tech Challenge Fase 1

Aplicando Modelo de aprendizagem de máquina para Previsão de Custos Médicos

Grupo 56

- Araguacy Bezerra Pereira RM362367
- Emerson Vitorio de Oliveira RM362731
- Jonas Lisboa Silva RM362907
- Robson Carvalho Calixto RM362870
- Vinicius Fernando M. Costa RM363007

Vamos a uma explicação rápida sobre as etapas de um projetos de Machine Learning

Podemos dividir o Processo de Ciência de Dados nas seguintes etapas:

- 1 - Definição do Problema de negócio
- 2 - Carregando e explorando os dados
- 3 - Análise Exploratória dos Dados / Pré-processamento
- 4 - Modelagem
- 5 - Comparação entre Random Forest e XGBoost
- 6 - Deploy do Modelo
- 7 - Prever o valor do prêmio de seguro médico

Vamos então para Hands on

1 - Definição do Problema de negócio

Quais fatores mais influenciam no custo médico

[Fonte dos dados](#)

- Age - Idade do cliente
- Diabetes - Se a pessoa tem níveis anormais de açúcar no sangue
- BloodPressureProblems - Se a pessoa tem níveis anormais de pressão arterial
- AnyTransplants - Qualquer transplante de órgão importante
- AnyChronicDiseases - Se o cliente sofre de doenças crônicas como asma
- Height - Altura do cliente
- Weight - Peso do Cliente
- KnownAllergies - Se o cliente tem alguma alergia conhecida
- HistoryOfCancerInFamily - Se algum parente consanguíneo do cliente teve algum tipo de câncer

- NumberOfMajorSurgeries - O número de cirurgias importantes que a pessoa passou
- PremiumPrice - Preço Premium Anual

In [655]:

```
# Importar as bibliotecas usadas no projeto

# Bibliotecas para manipulação dos dados e operações matemáticas
import pandas as pd
import numpy as np

# Analises estatísticas
from scipy import stats
from scipy.stats import shapiro, ttest_ind, mannwhitneyu, kruskal, f_oneway, norm, pearsonr

# Biblioteca para modelagem de machine learning
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split

# Bibliotecas para manipulação gráfica
import seaborn as sns
import matplotlib.pyplot as plt

# Definir uma semente para o gerador de números aleatórios
np.random.seed(42)

#Para ignorar os avisos
import warnings
warnings.filterwarnings('ignore')
sns.set_theme(style = "dark")
%matplotlib inline
```

In [656]:

```
# Versão da Linguagem Python
from platform import python_version

print('Versão da Linguagem Python Usada Neste Jupyter Notebook:', python_version())
print('')
# Versões dos pacotes usados neste jupyter notebook
```

Versão da Linguagem Python Usada Neste Jupyter Notebook: 3.12.2

2 - Carregando e explorando os dados

- Obter uma visão geral dos dados e se realmente podem ser usados para resolver o problema de negócio.

In [657]:

```
# Carregando os dados
df_original = pd.read_csv("data/Medicalpremium.csv")
```

In [658]:

```
# Shape
df_original.shape
```

```
Out[658]:  
(986, 11)
```

Tabela com 986 registros e 11 colunas

```
In [659]:
```

```
# Visualizando alguns registros  
df_original.head()
```

```
Out[659]:
```

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	Know
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	

```
In [660]:
```

```
df_original.info()
```

```
<class 'pandas.core.frame.DataFrame'>  
RangeIndex: 986 entries, 0 to 985  
Data columns (total 11 columns):  
#   Column                                Non-Null Count  Dtype  
---  -  
0   Age                                   986 non-null    int64  
1   Diabetes                             986 non-null    int64  
2   BloodPressureProblems                986 non-null    int64  
3   AnyTransplants                       986 non-null    int64  
4   AnyChronicDiseases                   986 non-null    int64  
5   Height                               986 non-null    int64  
6   Weight                               986 non-null    int64  
7   KnownAllergies                       986 non-null    int64  
8   HistoryOfCancerInFamily              986 non-null    int64  
9   NumberOfMajorSurgeries               986 non-null    int64  
10  PremiumPrice                         986 non-null    int64  
dtypes: int64(11)  
memory usage: 84.9 KB
```

Todas as variáveis com Dtype int64, tipagem correta será ajustada no análise exploratória / pré-processamento

```
In [661]:
```

```
# Verificando valores ausentes  
missing_values = df_original.isnull().sum().sort_values(ascending = False)  
missing_values
```

```
Out[661]:
```

```
Age                0  
Diabetes           0  
BloodPressureProblems  0  
AnyTransplants     0
```

```
AnyChronicDiseases      0
Height                  0
Weight                  0
KnownAllergies          0
HistoryOfCancerInFamily 0
NumberOfMajorSurgeries  0
PremiumPrice            0
dtype: int64
```

Dataset sem valores ausentes

In [662]:

```
# Resumo estatístico para variáveis quantitativas
df_original[['Age', 'Height', 'Weight', 'NumberOfMajorSurgeries', 'PremiumPrice']].describe
```

Out[662]:

	Age	Height	Weight	NumberOfMajorSurgeries	PremiumPrice
count	986.000000	986.000000	986.000000	986.000000	986.000000
mean	41.745436	168.182556	76.950304	0.667343	24336.713996
std	13.963371	10.098155	14.265096	0.749205	6248.184382
min	18.000000	145.000000	51.000000	0.000000	15000.000000
25%	30.000000	161.000000	67.000000	0.000000	21000.000000
50%	42.000000	168.000000	75.000000	1.000000	23000.000000
75%	53.000000	176.000000	87.000000	1.000000	28000.000000
max	66.000000	188.000000	132.000000	3.000000	40000.000000

Resumo estatístico aponta valores outliers em duas variáveis

Variável alvo PremiumPrice o valor máximo está longe da média e a variável Weight o valor máximo também distante de média.

In [663]:

```
# Definindo função para classificar a faixa etária e categoria de IMC
def classificar_faixa_etaria(idade):
    if idade >= 0 and idade <= 5:
        return 'Primeira infância',1      # 0 a 5 anos
    elif idade >= 6 and idade <= 12:
        return 'Infância',2                # 6 a 12 anos
    elif idade >= 13 and idade <= 17:
        return 'Adolescência',3            # 13 a 17 anos
    elif idade >= 18 and idade <= 24:
        return 'Jovem adulto',4            # 18 a 24 anos
    elif idade >= 25 and idade <= 39:
        return 'Adulto',5                  # 25 a 39 anos
    elif idade >= 40 and idade <= 59:
        return 'Meia-idade',6              # 40 a 59 anos
```

```

elif idade >= 60 and idade <= 74:
    return 'Idoso', 7          # 60 a 74 anos
elif idade >= 75:
    return 'Idoso longo', 8     # 75+ anos
else:
    return 'Idade inválida'

def calcular_imc(peso, altura):
    """
    Calcula o IMC com base no peso (kg) e altura (m).
    Retorna o valor do IMC e a classificação correspondente.
    """
    imc = peso / (altura ** 2)

    if imc < 18.5:
        classificacao = "Abaixo do peso"
    elif 18.5 <= imc < 25:
        classificacao = "Peso normal"
    elif 25 <= imc < 30:
        classificacao = "Sobrepeso"
    elif 30 <= imc < 35:
        classificacao = "Obesidade grau I"
    elif 35 <= imc < 40:
        classificacao = "Obesidade grau II"
    else:
        classificacao = "Obesidade grau III"

    return imc, classificacao

```

Vamos a uma explicação sobre o IQR e Outliers

O IQR é a diferença entre o terceiro quartil(Q3) e o primeiro quartil(Q1): $IQR = Q3 - Q1$ Q1 (1º quartil): é o valor abaixo do qual estão 25% dos dados Q3 (3º quartil): é o valor abaixo do qual estão 75% dos dados

Valores considerados outliers se estiverem fora desse limites (1.5 margem aceitável de variação), já multiplicar por 3 detecta outliers extremos: Abaixo de: $Q1 - 1.5 \times IQR$ Acima de: $Q3 + 1.5 \times IQR$

Esses limites são chamados de limites de Tukey. Tudo fora disso é potencial outlier. A multiplicação por 1.5 é um fator de tolerância, criado por John Tukey, o estatístico que inventou o boxplot. Uma maneira simples e automático de encontrar valores que estão fora da faixa normal dos dados, sem precisar de suposições sobre a distribuição (tipo normalidade).

In [664]:

```

def analisar_outliers(df, coluna, seed=42, exibir_plot=True):
    """
    Analisa outliers em uma coluna numérica de um DataFrame com base no IQR (1.5x e 3x)
    e plota o boxplot com os limites.

    Parâmetros:
    - df: DataFrame contendo os dados
    - coluna: nome da coluna a ser analisada (string)
    - seed: semente para reprodução de resultados (default=42)
    - exibir_plot: se True, exibe o boxplot com limites (default=True)

    Retorna:
    - dicionário com Q1, Q3, IQR, limites e DataFrames de outliers
    """

```

```

"""
np.random.seed(seed)

Q1 = df[coluna].quantile(0.25)
Q3 = df[coluna].quantile(0.75)
IQR = Q3 - Q1

limite_inf_1_5 = Q1 - 1.5 * IQR
limite_sup_1_5 = Q3 + 1.5 * IQR
limite_inf_3 = Q1 - 3.0 * IQR
limite_sup_3 = Q3 + 3.0 * IQR

if exibir_plot:
    plt.figure(figsize=(10, 5))
    sns.boxplot(x=df[coluna], color='skyblue')
    plt.axvline(limite_inf_1_5, color='orange', linestyle='--', label='1.5 × IQR lim
    plt.axvline(limite_sup_1_5, color='orange', linestyle='--')
    plt.axvline(limite_inf_3, color='red', linestyle=':', label='3.0 × IQR limites (
    plt.axvline(limite_sup_3, color='red', linestyle=':')
    plt.title(f'Boxplot com Limiares de 1.5×IQR e 3.0×IQR - {coluna}')
    plt.xlabel('Valores')
    plt.legend()
    plt.grid(True)
    plt.tight_layout()
    plt.show()

outliers_1_5 = df[(df[coluna] < limite_inf_1_5) | (df[coluna] > limite_sup_1_5)]
outliers_3 = df[(df[coluna] < limite_inf_3) | (df[coluna] > limite_sup_3)]

print("Q1:", Q1)
print("Q3:", Q3)
print("IQR (Interquartile Range):", IQR)
print("-----")
print("Limite inferior 1.5:", limite_inf_1_5)
print("Limite superior 1.5:", limite_sup_1_5)
print("-----")
print("Limite inferior 3:", limite_inf_3)
print("Limite superior 3:", limite_sup_3)
print("-----")
print(f"Número de outliers 1.5: {outliers_1_5.shape[0]}")
print(f"Número de outliers 3 (extremos): {outliers_3.shape[0]}")

return {
    'Q1': Q1,
    'Q3': Q3,
    'IQR': IQR,
    'limite_1_5': (limite_inf_1_5, limite_sup_1_5),
    'limite_3': (limite_inf_3, limite_sup_3),
    'outliers_1_5': outliers_1_5,
    'outliers_3': outliers_3
}

```

In [665]:

```

def comparar_grupos_categoricos(df, categorias, target):
    resultados = [] # Lista para armazenar os resultados finais

    #Loop por cada variável categórica
    for grupo in categorias:
        unique_vals = df[grupo].dropna().unique() # Identifica os valores únicos da var

```

```

group_data = []          # Lista para armazenar os dados do target por grupo
normalities = []         # Lista para registrar se cada grupo tem distribuição nor

#Separar os dados por grupo e testar normalidade
for val in unique_vals:
    sample = df[df[grupo] == val][target].dropna() # Seleciona os valores da var
    group_data.append(sample)

    # Teste de normalidade (Shapiro-Wilk), amostrando até 500 observações
    if len(sample) >= 3:
        stat, p = shapiro(sample.sample(min(len(sample), 500), random_state=42))
    else:
        p = 0 # Amostras muito pequenas são tratadas como não normais
    normalities.append(p > 0.05) # True se a amostra for normal

#Selecionar o teste estatístico com base nos grupos e na normalidade
if len(unique_vals) == 2:
    # 2 grupos: teste t ou Mann-Whitney
    if all(normalities):
        stat, pval = ttest_ind(group_data[0], group_data[1], equal_var=False)
        test_name = "t-test"
    else:
        stat, pval = mannwhitneyu(group_data[0], group_data[1], alternative="two")
        test_name = "Mann-Whitney U"
else:
    # Mais de 2 grupos: ANOVA ou Kruskal-Wallis
    if all(normalities):
        stat, pval = f_oneway(*group_data)
        test_name = "ANOVA"
    else:
        stat, pval = kruskal(*group_data)
        test_name = "Kruskal-Wallis"

# Registrar os resultados
resultados.append({
    "Variável": grupo,
    "Grupos": list(unique_vals),
    "Teste": test_name,
    "p-valor": round(pval, 4),
    "Resultado": "Significativo" if pval < 0.05 else "Não significativo"
})

return pd.DataFrame(resultados)

```

In [666]:

```

def plotar_boxplots_resultados(df, resultado_df, target):
    n = len(resultado_df)
    n_cols = 2
    n_rows = (n + 1) // n_cols

    fig, axes = plt.subplots(n_rows, n_cols, figsize=(18, n_rows * 5))
    axes = axes.flatten()

    for i, row in resultado_df.iterrows():
        var = row["Variável"]
        test_name = row["Teste"]
        pval = row["p-valor"]
        resultado = row["Resultado"]

```

```

sns.boxplot(data=df, x=var, y=target, palette="Set2", ax=axes[i])
axes[i].set_title(f"{var} - {test_name} (p = {pval})\n{resultado}")
axes[i].set_xlabel(var)
axes[i].set_ylabel(target)
axes[i].tick_params(axis='x', rotation=45)

# Adiciona valores de mediana como ponto com texto
#for tick, label in enumerate(df[var].dropna().unique()):
#    mediana = df[df[var] == label][target].median()
#    axes[i].text(tick, mediana, f"{mediana:.1f}",
#                horizontalalignment='center', color='white', fontsize=10)

# Remove gráficos extras
for j in range(i + 1, len(axes)):
    fig.delaxes(axes[j])

```

In [667]:

```

def analisar_correlacoes_com_target(df, variaveis, target, exibir_graficos=True):
    """
    Calcular correlações estatísticas entre um conjunto de variáveis independentes (variáveis
    numéricas (coeficientes de correlação) e, opcionalmente, gráficos de dispersão com l
    """

    resultados = []

    for var in variaveis:
        # Remove valores nulos
        dados_validos = df[[var, target]].dropna()

        # Calcula correlações
        r_pearson, p_pearson = pearsonr(dados_validos[var], dados_validos[target])
        r_spearman, p_spearman = spearmanr(dados_validos[var], dados_validos[target])

        # Armazena os resultados
        resultados.append({
            'variavel': var,
            'pearson_r': round(r_pearson, 4),
            'pearson_p': round(p_pearson, 4),
            'spearman_r': round(r_spearman, 4),
            'spearman_p': round(p_spearman, 4)
        })

        # Gráfico
        if exibir_graficos:
            plt.figure(figsize=(7, 5))
            sns.regplot(x=var, y=target, data=dados_validos, scatter_kws={'alpha':0.5},
            plt.title(f'Dispersão: {var} vs {target}')
            plt.xlabel(var)
            plt.ylabel(target)
            plt.grid(True)
            plt.tight_layout()
            plt.show()

    # Converte para DataFrame e retorna
    return pd.DataFrame(resultados)

```

In [668]:

```

def plotar_heatmap_correlacoes(df, variaveis, target='PremiumPrice'):
    # Filtra apenas as variáveis de interesse (incluindo o target)

```



```

colunas = variaveis + [target]
df_corr = df[colunas].dropna() # remove nulos

# Calcula matriz de correlação
matriz_corr = df_corr.corr(method='spearman') # pearson ou 'spearman'

# Plot do heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(matriz_corr, annot=True, fmt=".2f", cmap='coolwarm', linewidths=0.5)
plt.title('Heatmap de Correlação (Spearman)')
plt.tight_layout()
plt.show()

```

In [669]:

```

def transformar_variavel(serie, nome_variavel="Variável"):
    """
    Aplica transformações log e Box-Cox à série fornecida.

    Retorna um dicionário com:
    - Original
    - Log
    - Box-Cox
    - Lambda do Box-Cox
    - Deslocamento usado
    """
    # Garantir que é array NumPy
    serie = np.array(serie)

    # Verifica se há valores <= 0 (log e Box-Cox precisam de dados positivos)
    deslocamento = 0
    if (serie <= 0).any():
        deslocamento = abs(serie.min()) + 1
        serie_pos = serie + deslocamento
    else:
        serie_pos = serie.copy()

    # Aplicar transformações
    log_transf = np.log(serie_pos)
    boxcox_transf, lambda_bc = stats.boxcox(serie_pos)

    # Calcular skewness
    skew_original = skew(serie)
    skew_log = skew(log_transf)
    skew_boxcox = skew(boxcox_transf)

    # Plot
    plt.figure(figsize=(18,5))

    plt.subplot(1,3,1)
    sns.histplot(serie, kde=True)
    plt.title(f"Original ({nome_variavel})\nSkew = {skew_original:.2f}")

    plt.subplot(1,3,2)
    sns.histplot(log_transf, kde=True)
    plt.title(f"Log Transform\nSkew = {skew_log:.2f}")

    plt.subplot(1,3,3)
    sns.histplot(boxcox_transf, kde=True)
    plt.title(f"Box-Cox (λ = {lambda_bc:.2f})\nSkew = {skew_boxcox:.2f}")

```

```
plt.suptitle(f'Transformações para {nome_variavel}', fontsize=16)
plt.tight_layout()
plt.show()

# Retorno
return {
    "original": serie,
    "log": log_transf,
    "boxcox": boxcox_transf,
    "lambda_boxcox": lambda_bc,
    "deslocamento": deslocamento
}
```

3 - Análise Exploratória dos Dados / Pré-processamento

- Nesta etapa vamos explorar os dados e compreender como estão organizados com a ajuda da Estatística.
- Aplicação da técnica de Data Wrangling para limpar, transformar e preparar os dados brutos para análise, modelagem e visualização.

In [670]:

```
# Criando um dataframe para manipulação dos dados
df_dados = df_original

# Aplicando ao DataFrame novas colunas
df_dados[['faixa_etaria', 'ordem_faixa']] = df_original['Age'].apply(lambda x: pd.Series

# Calculo IMC
df_dados[['imc', 'categoria_imc']] = df_original.apply(
    lambda row: pd.Series(calcular_imc(row['Weight'], row['Height'] / 100)),
    axis=1
)

# Ordem lógica para as categorias de IMC, faixa etária
ordem_categorias = [
    'Abaixo do peso',
    'Peso normal',
    'Sobrepeso',
    'Obesidade grau I',
    'Obesidade grau II',
    'Obesidade grau III'
]

df_dados['categoria_imc'] = pd.Categorical(
    df_dados['categoria_imc'],
    categories=ordem_categorias,
    ordered=True
)

ordem_faixa = ['Primeira infância', 'Infância', 'Adolescência', 'Jovem adulto', 'Adulto']
df_dados['faixa_etaria'] = pd.Categorical(
    df_dados['faixa_etaria'],
    categories=ordem_faixa,
```

```

    ordered=True
)

cols_binarias = [
    'Diabetes',
    'BloodPressureProblems',
    'AnyTransplants',
    'AnyChronicDiseases',
    'KnownAllergies',
    'HistoryOfCancerInFamily'
]

# Transformar para categoria (tipo categórico)
for col in cols_binarias:
    df_dados[col] = df_dados[col].astype("category")

```

In [671]:

```

# Salvando o dataset para pre-processamento
df_dados.to_csv('./data/dados_projeto.csv', index = False)

```

In [672]:

```
df_dados.shape
```

Out[672]:

(986, 15)

Tabela com 986 registros e 15 colunas - Criação de 4 colunas para ajudar a entender os dados

In [673]:

```

# Informações do dataset
df_dados.info()

```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 986 entries, 0 to 985
Data columns (total 15 columns):
#   Column                                Non-Null Count  Dtype
---  -
0   Age                                   986 non-null    int64
1   Diabetes                             986 non-null    category
2   BloodPressureProblems                986 non-null    category
3   AnyTransplants                       986 non-null    category
4   AnyChronicDiseases                   986 non-null    category
5   Height                               986 non-null    int64
6   Weight                               986 non-null    int64
7   KnownAllergies                       986 non-null    category
8   HistoryOfCancerInFamily              986 non-null    category
9   NumberOfMajorSurgeries               986 non-null    int64
10  PremiumPrice                         986 non-null    int64
11  faixa_etaria                         986 non-null    category
12  ordem_faixa                          986 non-null    int64
13  imc                                   986 non-null    float64
14  categoria_imc                        986 non-null    category
dtypes: category(8), float64(1), int64(6)
memory usage: 63.1 KB

```

In [674]:

```
#visualizando os dados
df_dados.head()
```

Out[674]:

	Age	Diabetes	BloodPressureProblems	AnyTransplants	AnyChronicDiseases	Height	Weight	Know
0	45	0	0	0	0	155	57	
1	60	1	0	0	0	180	73	
2	36	1	1	0	0	158	59	
3	52	1	1	0	1	183	93	
4	38	0	0	0	1	166	88	

In [675]:

```
# Análise estatística descritiva da variável alvo

# Garantir que não haja NaNs
data = df_dados['PremiumPrice'].dropna()

# Estatísticas principais
mu = data.mean()
std = data.std()
med = data.median()
skewness = data.skew()
kurt = data.kurtosis()

# Histograma com densidade
plt.figure(figsize=(12, 6))
count, bins, ignored = plt.hist(data, bins=20, density=False, color='skyblue',
                                edgecolor='white', alpha=0.7, label='Histograma')

# Curva normal teórica
x = np.linspace(min(bins), max(bins), 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'r-', linewidth=2, label='Curva Normal')

# Linhas da média e mediana
plt.axvline(mu, color='green', linestyle='--', linewidth=2, label='Média')
plt.axvline(med, color='purple', linestyle='-.', linewidth=2, label='Mediana')

# Outliers visuais (> 3 desvios)
outliers = data[(data < mu - 3*std) | (data > mu + 3*std)]
plt.scatter(outliers, [0]*len(outliers), color='red', zorder=5, label='Outliers (>3σ)')

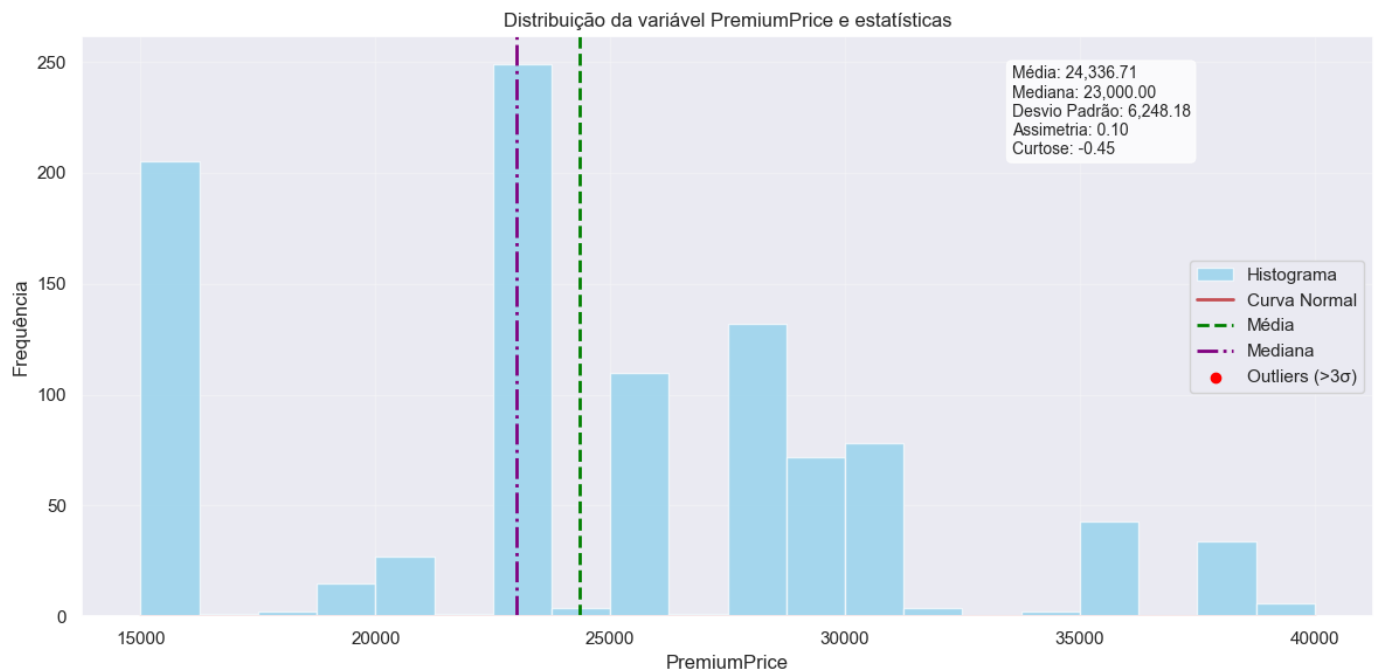
# Caixa de estatísticas
stats_text = '\n'.join((
    f'Média: {mu:,.2f}',
    f'Mediana: {med:,.2f}',
    f'Desvio Padrão: {std:,.2f}',
    f'Assimetria: {skewness:,.2f}',
    f'Curtose: {kurt:,.2f}'
))
props = dict(boxstyle='round', facecolor='white', alpha=0.8)
plt.text(0.72, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
```

```

verticalalignment='top', bbox=props)

# Ajustes finais
plt.title('Distribuição da variável PremiumPrice e estatísticas') #com curva normal
plt.xlabel('PremiumPrice')
plt.ylabel('Frequência')
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

```



Análise Descritiva da Variável PremiumPrice

A variável PremiumPrice apresenta uma média de aproximadamente 24.336,71, enquanto a mediana é 23.000,00, indicando uma distribuição levemente assimétrica à direita (positiva), com presença de alguns valores mais altos que puxam a média para cima.

O desvio padrão é de aproximadamente 6.248,18, o que revela uma dispersão moderada em torno da média — ou seja, os preços variam de forma relativamente ampla dentro do conjunto.

A assimetria (skewness) de 0,10 confirma essa leve cauda à direita, enquanto a curtose de -0,45 indica uma distribuição mais achatada que a normal (platicúrtica), com menos concentração de valores próximos à média e caudas menos acentuadas.

Em resumo, a distribuição de PremiumPrice é quase simétrica, com leve influência de valores mais altos, e mostra uma variabilidade moderada, sem concentração excessiva no centro ou nas extremidades.

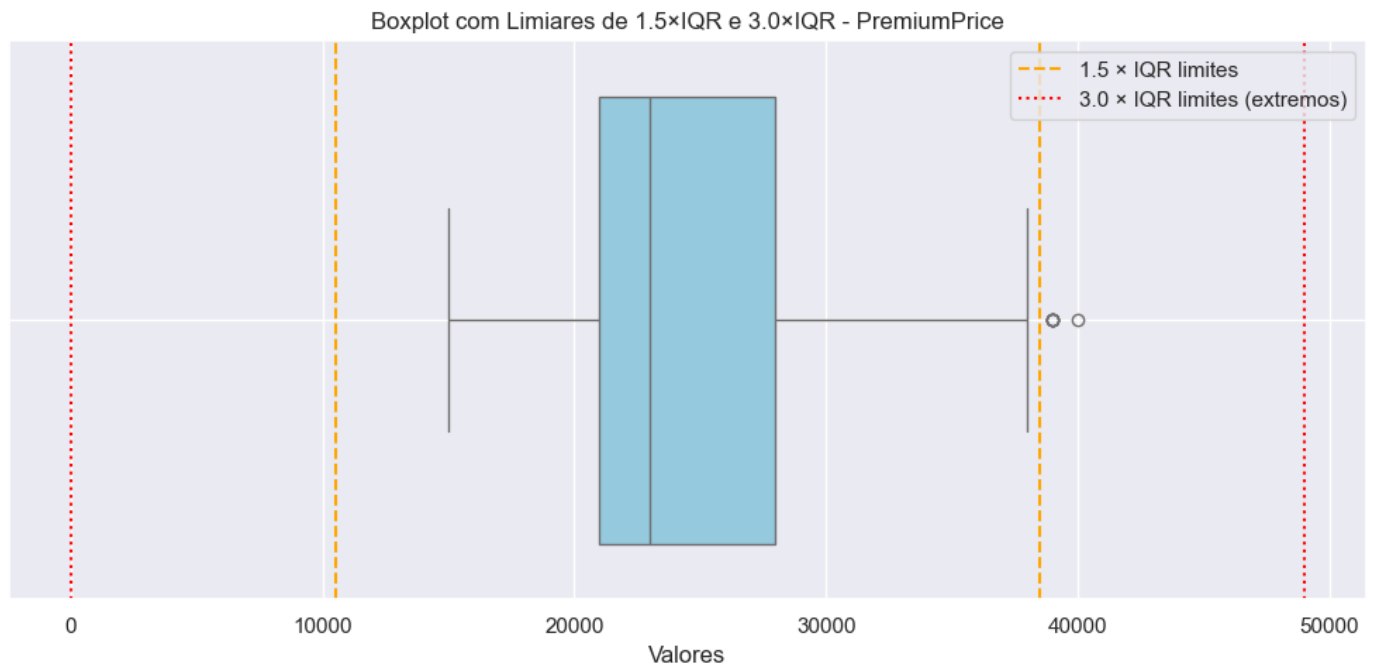
O gráfico indica possíveis outliers, esses valores devem ser analisados com cuidado pois pode influenciar negativamente modelos preditivos sensíveis a valores extremos.

In [676]:

```

# Verificar se a variável alvo possui outliers
resultado = analisar_outliers(df_dados, 'PremiumPrice')

```



Q1: 21000.0

Q3: 28000.0

IQR (Interquartile Range): 7000.0

Limite inferior 1.5: 10500.0


Limite superior 1.5: 38500.0

Limite inferior 3: 0.0

Limite superior 3: 49000.0

Número de outliers 1.5: 6

Número de outliers 3 (extremos): 0

 A visualização do Boxplot indica uma oportunidade de transformação no pré-processamento, 6 registros encontrado com outliers.

Para analisar:

- Aplicar transformação logarítmica na variável PremiumPrice para aproximar uma distribuição normal.
- Tratar ou remover outliers com base na análise de IQR (como a função que criamos anteriormente).

In [677]:

```
# Análise estatística descritiva da variável IMC
```

```
# Garantir que não haja NaNs
```

```
data = df_dados['imc'].dropna()
```

```
# Estatísticas principais
```

```
mu = data.mean()
```

```
std = data.std()
```

```
med = data.median()
```

```
skewness = data.skew()
```

```
kurt = data.kurtosis()
```

```
# Histograma com densidade
```

```
plt.figure(figsize=(12, 6))
```

```
count, bins, ignored = plt.hist(data, bins=20, density=False, color='skyblue',
```

```

edgecolor='white', alpha=0.7, label='Histograma')

# Curva normal teórica
x = np.linspace(min(bins), max(bins), 100)
p = norm.pdf(x, mu, std)
plt.plot(x, p, 'r-', linewidth=2, label='Curva Normal')

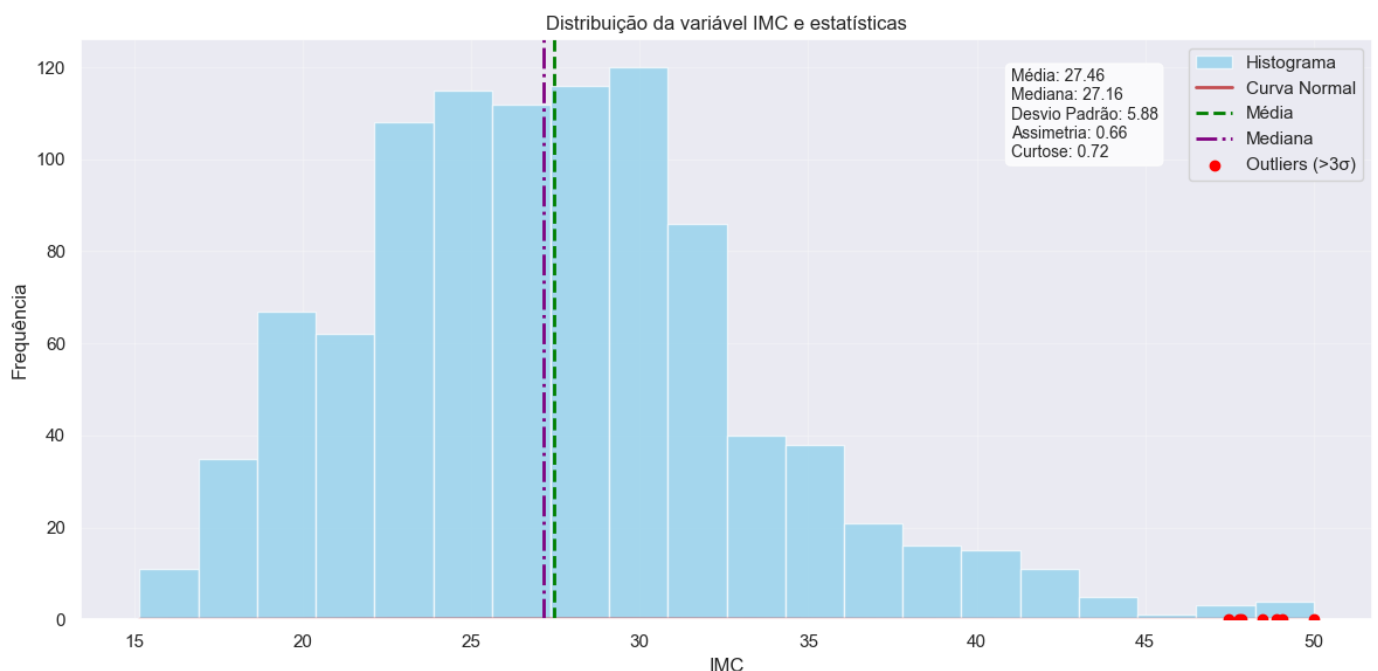
# Linhas da média e mediana
plt.axvline(mu, color='green', linestyle='--', linewidth=2, label='Média')
plt.axvline(med, color='purple', linestyle='-.', linewidth=2, label='Mediana')

# Outliers visuais (> 3 desvios)
outliers = data[(data < mu - 3*std) | (data > mu + 3*std)]
plt.scatter(outliers, [0]*len(outliers), color='red', zorder=5, label='Outliers (>3σ)')

# Caixa de estatísticas
stats_text = '\n'.join((
    f'Média: {mu:,.2f}',
    f'Mediana: {med:,.2f}',
    f'Desvio Padrão: {std:,.2f}',
    f'Assimetria: {skewness:,.2f}',
    f'Curtose: {kurt:,.2f}'
))
props = dict(boxstyle='round', facecolor='white', alpha=0.8)
plt.text(0.72, 0.95, stats_text, transform=plt.gca().transAxes, fontsize=10,
        verticalalignment='top', bbox=props)

# Ajustes finais
plt.title('Distribuição da variável IMC e estatísticas') #com curva normal
plt.xlabel('IMC')
plt.ylabel('Frequência')
plt.grid(alpha=0.3)
plt.legend()
plt.tight_layout()
plt.show()

```



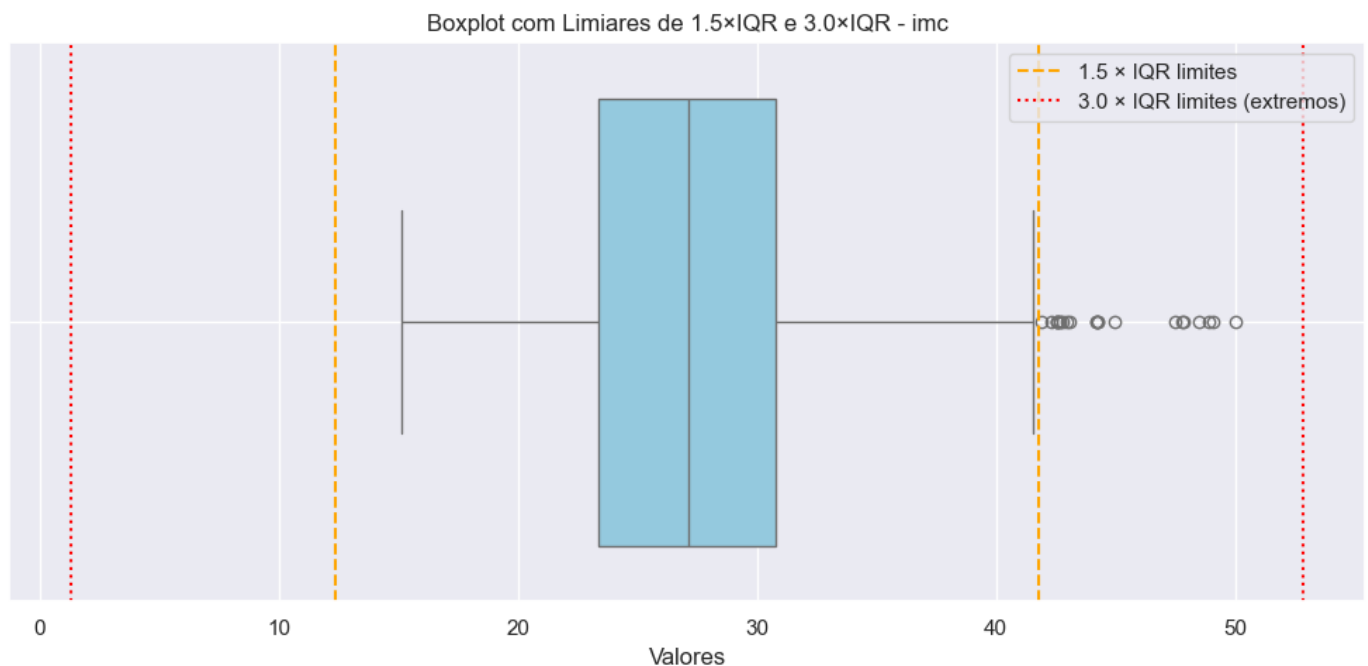
A distribuição é assimétrica à direita (cauda alongada para valores maiores), o que é comum em variáveis biométricas como IMC (Índice de Massa Corporal).

O pico de frequência está entre 25 e 30, com mais de 120 ocorrências no bin de maior densidade — o que indica que a maioria das pessoas da amostra está na faixa de sobrepeso (segundo a OMS, IMC entre 25 e 30).

📌 Possíveis outliers Valores acima de 40 (IMC ≥ 40 é considerado obesidade grau III ou obesidade mórbida) são bem menos frequentes e aparecem como potenciais outliers. Do lado oposto, há algumas ocorrências abaixo de 18.5 (limite inferior de IMC saudável), o que pode indicar baixo peso ou dados raros.

In [678]:

```
# Verificar se a variável possui outliers
resultado = analisar_outliers(df_dados, 'imc')
```



Q1: 23.393392200872924

Q3: 30.75987020010817

IQR (Interquartile Range): 7.366477999235247

Limite inferior 1.5: 12.343675202020053

Limite superior 1.5: 41.80958719896104

Limite inferior 3: 1.2939582031671826

Limite superior 3: 52.85930419781391

Número de outliers 1.5: 22

Número de outliers 3 (extremos): 0

📊 Oportunidade para transformações estatísticas como log ou Box-Cox podem ajudar caso essa variável vá alimentar modelos sensíveis à normalidade. Foram encontrados 22 registros com outliers.

Uma análise por categorias de IMC (baixo peso, saudável, sobrepeso, obesidade I/II/III) pode ser útil para enriquecer insights ou treinar modelos com variáveis categóricas derivadas.

In [679]:

```
# Análise de significância das variáveis categóricas x variável alvo
categorical_groups = [
    "Diabetes", "BloodPressureProblems", "AnyTransplants",
```



```

    "AnyChronicDiseases", "KnownAllergies", "HistoryOfCancerInFamily",
    "ordem_faixa", "categoria_imc"
]

resultado_testes = comparar_grupos_categoricos(df_dados, categorical_groups, "PremiumPri
display(resultado_testes)

```

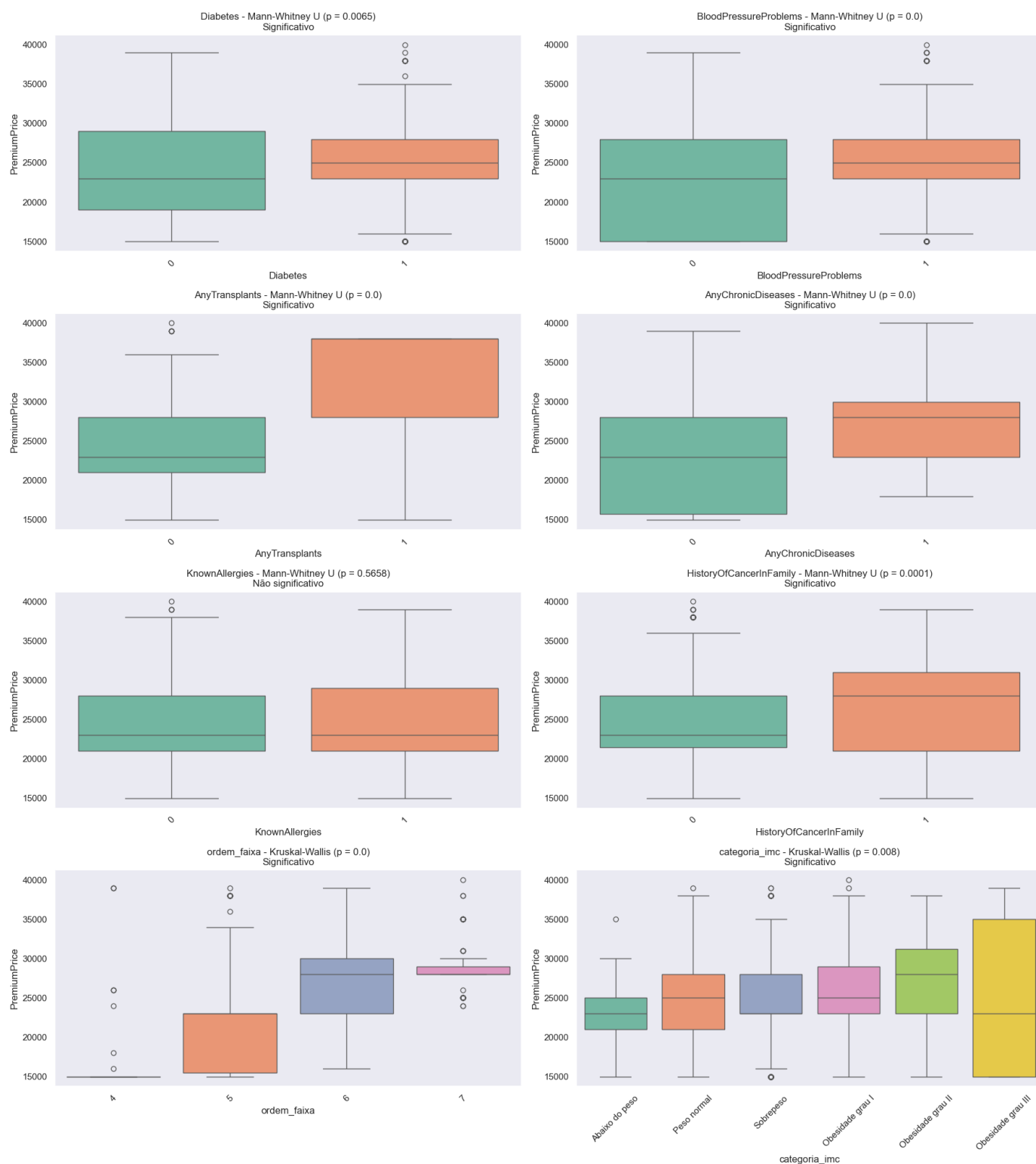
	Variável	Grupos	Teste	P-valor	Resultado
0	Diabetes	[0, 1]	Mann-Whitney U	0.0065	Significativo
1	BloodPressureProblems	[0, 1]	Mann-Whitney U	0.0000	Significativo
2	AnyTransplants	[0, 1]	Mann-Whitney U	0.0000	Significativo
3	AnyChronicDiseases	[0, 1]	Mann-Whitney U	0.0000	Significativo
4	KnownAllergies	[0, 1]	Mann-Whitney U	0.5658	Não significativo
5	HistoryOfCancerInFamily	[0, 1]	Mann-Whitney U	0.0001	Significativo
6	ordem_faixa	[6, 7, 5, 4]	Kruskal-Wallis	0.0000	Significativo
7	categoria_imc	[Peso normal, Sobrepeso, Obesidade grau I, Aba...	Kruskal-Wallis	0.0080	Significativo

In [680]:

```

# Plot do resultado
plotar_boxplots_resultados(df_dados, resultado_testes, "PremiumPrice")
plt.tight_layout()
plt.show()

```



📄 Análise variáveis categóricas x variável alvo

Esse resultado mostra uma análise estatística de diferença entre grupos de variáveis categóricas com relação a variável alvo contínua. Foram feitos testes não paramétricos (Mann-Whitney e Kruskal-Wallis), que são mais robustos para dados que não seguem distribuição normal.

Exceto a variável KnownAllergies com p-valor = 0.5658 maior que 0.05 se mostrou não significativa com a variável alvo, as demais variáveis apresenta diferença estatística entre os grupos, indicando que aspectos como doenças crônicas, IMC, idade/faixa e histórico de saúde influenciam fortemente o comportamento medido pela variável-alvo.

In [681]:

```
# Tabela de Frequência das variáveis categóricas
variaveis_significativas = [
    "Diabetes", "BloodPressureProblems",
    "AnyChronicDiseases", "KnownAllergies", "HistoryOfCancerInFamily",
    "faixa_etaria", "categoria_imc"
]
# Função do pandas que divide uma série numérica em quantis.
# O parâmetro q=4 quer dizer: dividir a variável PremiumPrice em 4 faixas com aproximada
df_dados['faixa_preco'] = pd.qcut(df_dados['PremiumPrice'], q=4, labels=['Baixo', 'Médio', 'Alto', 'Total'])
df_dados['range'] = pd.qcut(df_dados['PremiumPrice'], q=4)

print(f"\n📊 Faixa preço: {df_dados['range'].unique()}")
for var in variaveis_significativas:
    print(f"\n📊 Crosstab: {var} x faixa_preco")
    display(pd.crosstab(df_dados[var], df_dados['faixa_preco'], margins=True, margins_name='Total'))

# > 0 desbalanceamento de classe é um problema que terá que ser resolvido durante o pré-
```

📊 Faixa preço: [(23000.0, 28000.0], (28000.0, 40000.0], (21000.0, 23000.0], (14999.999, 21000.0]]

Categories (4, interval[float64, right]): [(14999.999, 21000.0] < (21000.0, 23000.0] < (23000.0, 28000.0] < (28000.0, 40000.0]]

📊 Crosstab: Diabetes x faixa_preco

faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
Diabetes					
0	171	149	103	149	572
1	79	101	144	90	414
Total	250	250	247	239	986

📊 Crosstab: BloodPressureProblems x faixa_preco


faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
BloodPressureProblems					
0	173	144	81	126	524
1	77	106	166	113	462
Total	250	250	247	239	986

📊 Crosstab: AnyChronicDiseases x faixa_preco

faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
AnyChronicDiseases					
0	224	220	194	170	808
1	26	30	53	69	178
Total	250	250	247	239	986

📊 Crosstab: KnownAllergies x faixa_preco


faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
KnownAllergies					
0	195	194	213	172	774
1	55	56	34	67	212
Total	250	250	247	239	986

 Crosstab: HistoryOfCancerInFamily x faixa_preco

faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
HistoryOfCancerInFamily					
0	218	249	213	190	870
1	32	1	34	49	116
Total	250	250	247	239	986

 Crosstab: faixa_etaria x faixa_preco

faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
faixa_etaria					
Jovem adulto	129	0	4	3	136
Adulto	117	143	13	26	299
Meia-idade	4	107	142	163	416
Idoso	0	0	88	47	135
Total	250	250	247	239	986

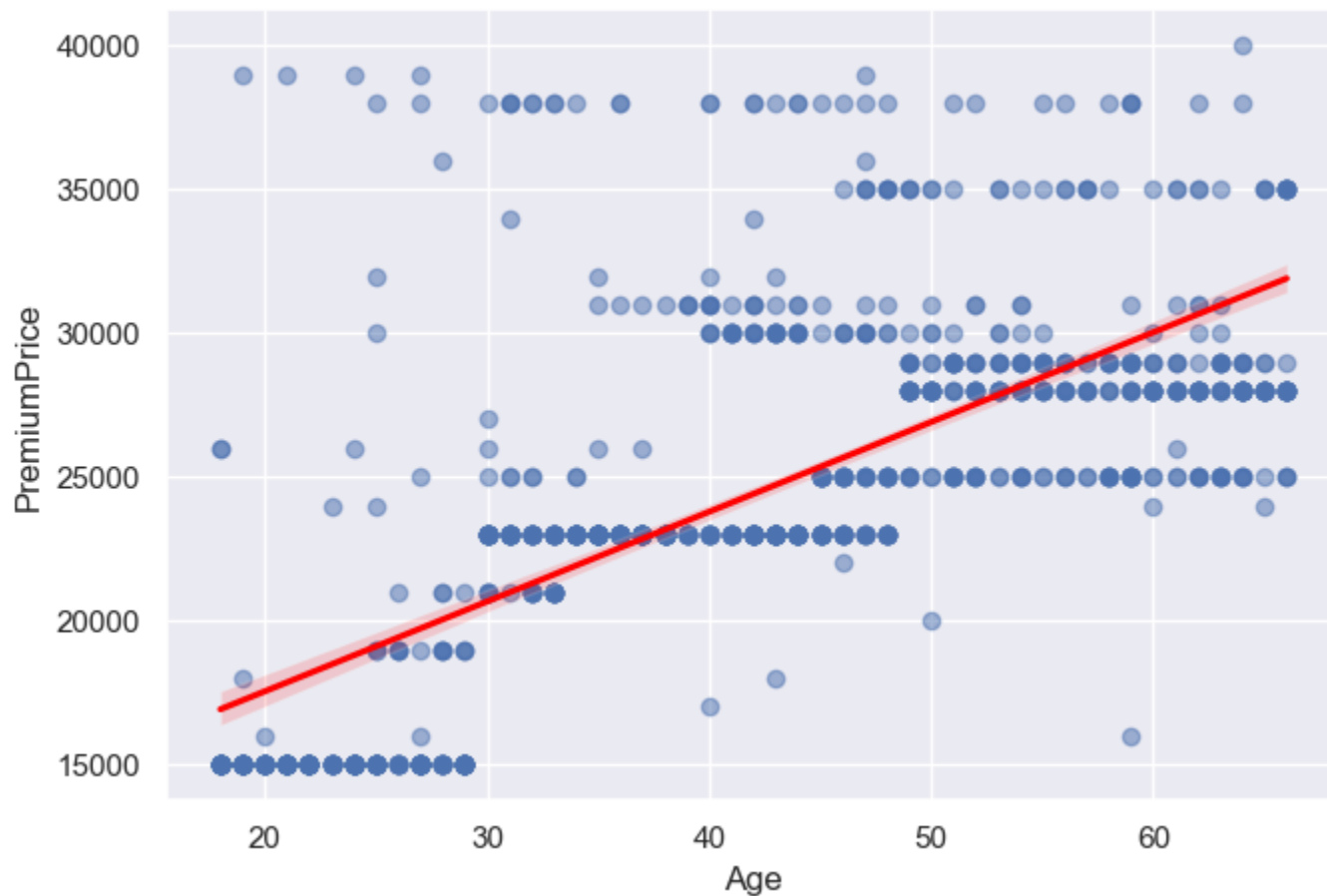
 Crosstab: categoria_imc x faixa_preco

faixa_preco	Baixo	Médio-Baixo	Médio-Alto	Alto	Total
categoria_imc					
Abaixo do peso	11	12	12	4	39
Peso normal	90	65	110	54	319
Sobrepeso	78	97	74	76	325
Obesidade grau I	49	50	36	68	203
Obesidade grau II	9	19	10	26	64
Obesidade grau III	13	7	5	11	36
Total	250	250	247	239	986

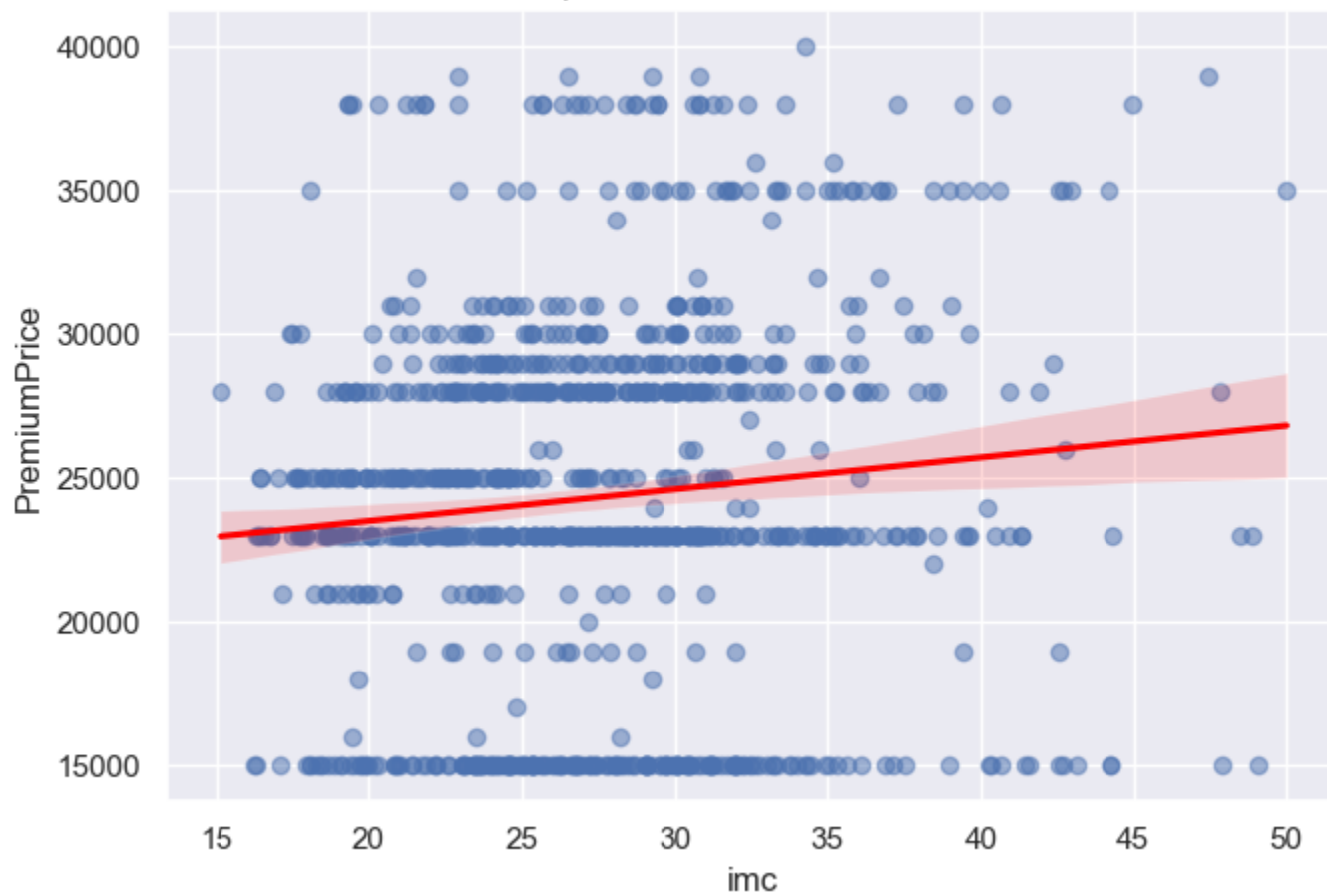
In [682]:

```
# Analisar as correlações com as variáveis quantitativas
variaveis = ['Age', 'imc', 'NumberOfMajorSurgeries', 'Height', 'Weight']
resultado_corr = analisar_correlacoes_com_target(df_dados, variaveis, 'PremiumPrice', ex
display(resultado_corr)
```

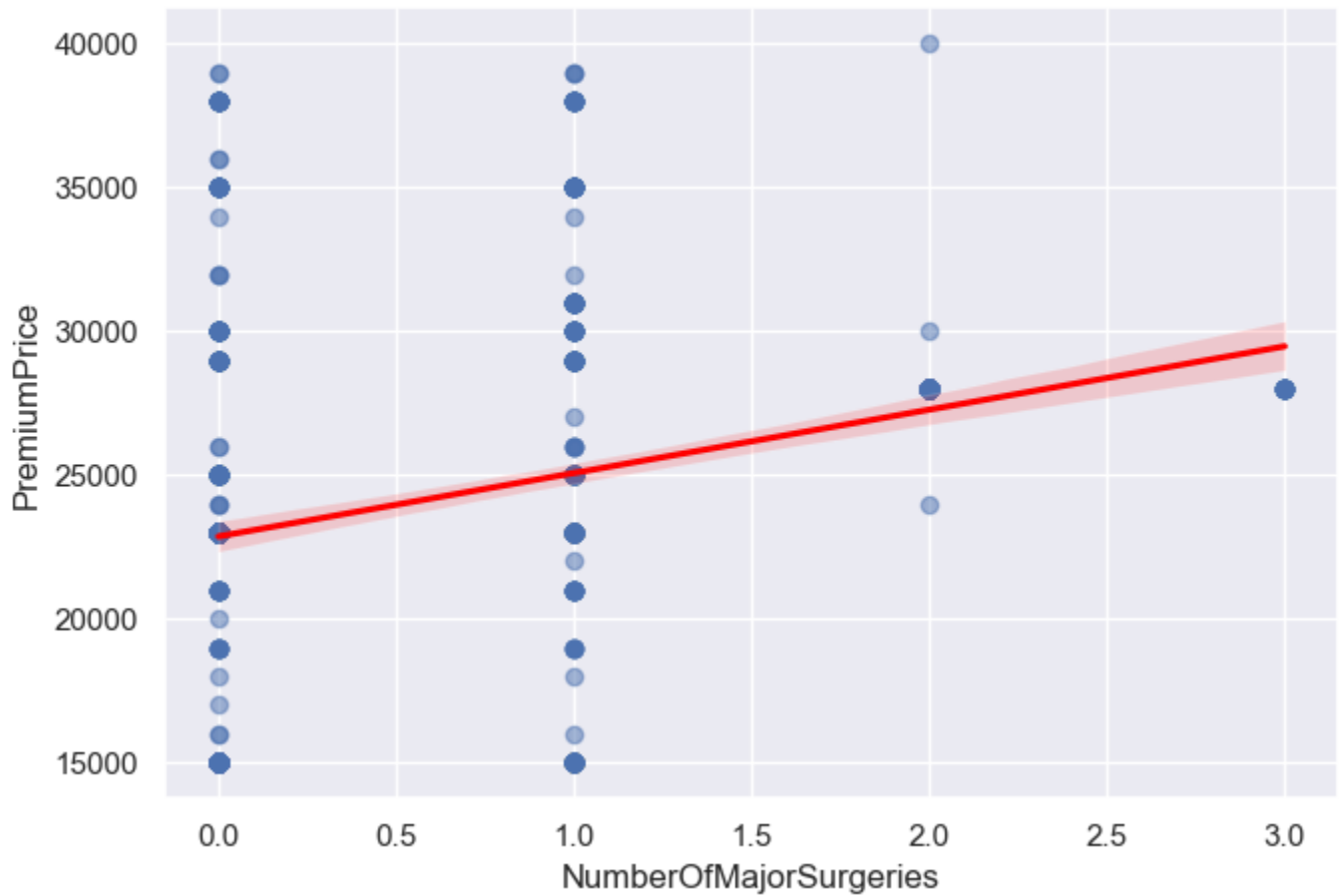
Dispersão: Age vs PremiumPrice



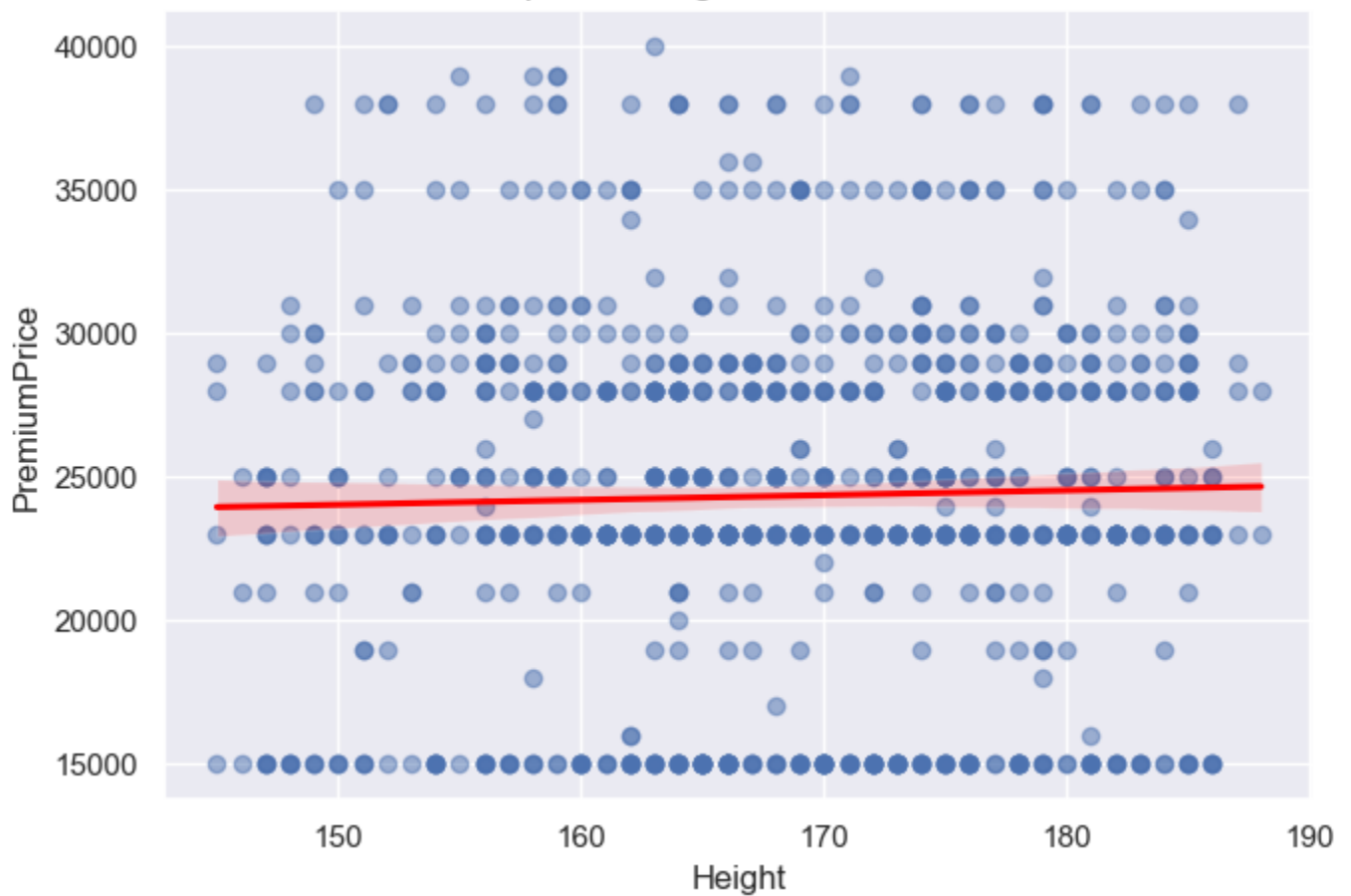
Dispersão: imc vs PremiumPrice



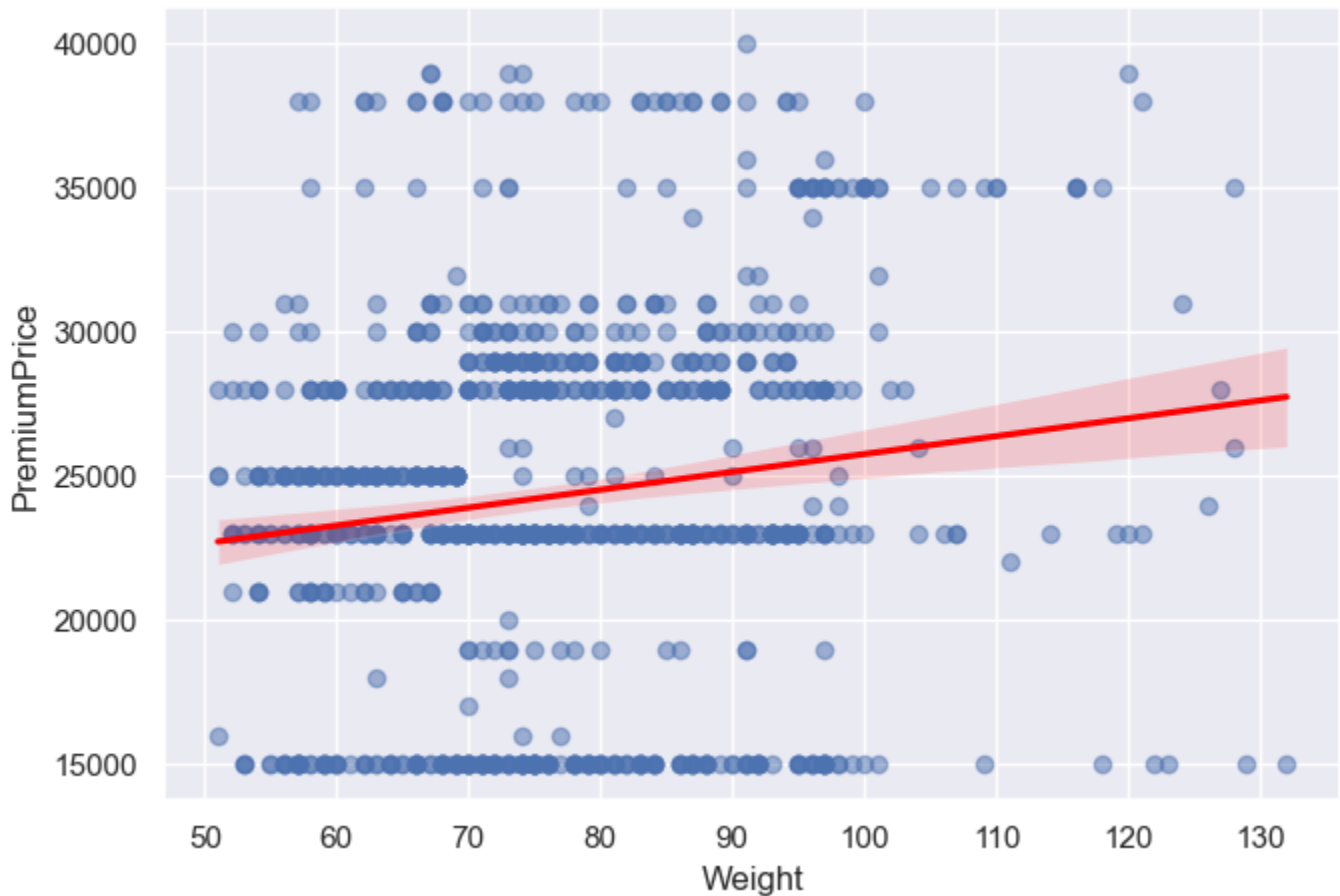
Dispersão: NumberOfMajorSurgeries vs PremiumPrice



Dispersão: Height vs PremiumPrice



Dispersão: Weight vs PremiumPrice



	variavel	pearson_r	pearson_p	spearman_r	spearman_p
0	Age	0.6975	0.0000	0.7391	0.0000
1	imc	0.1038	0.0011	0.0979	0.0021
2	NumberOfMajorSurgeries	0.2642	0.0000	0.2895	0.0000
3	Height	0.0269	0.3986	0.0231	0.4682
4	Weight	0.1415	0.0000	0.1293	0.0000

⚠ Conclusão:

- O coeficiente de correlação r diz o quão forte é a relação. Correlação positiva perfeita (+1), negativa perfeita (-1) e 0 nenhuma correlação.
- O p-valor significância estatística diz o quanto podemos confiar que a relação existe no geral. Um p-valor pequeno (geralmente < 0.05) indica que é muito improvável que aquela correlação tenha surgido por acaso.

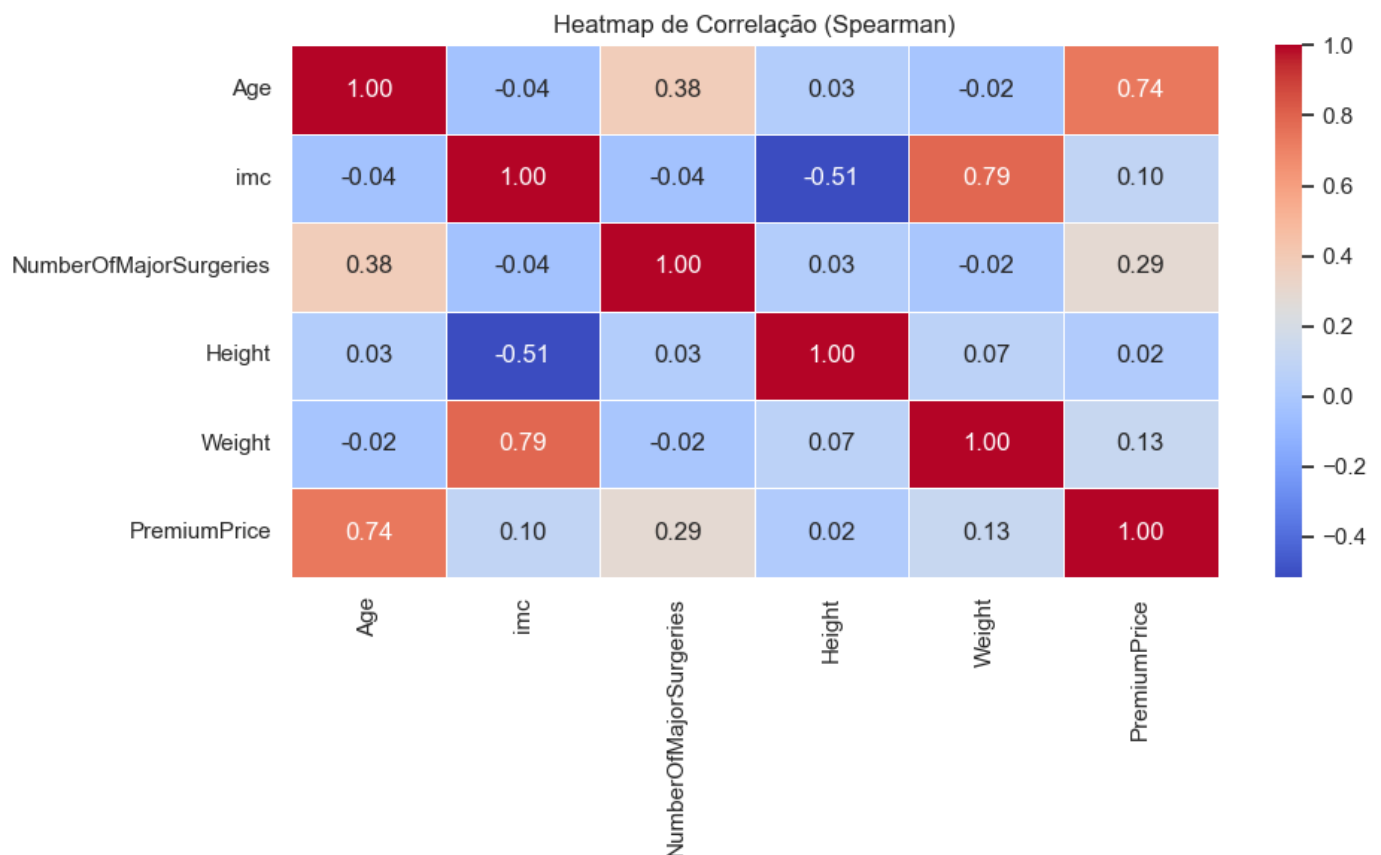
Uma análise robusta olha os dois juntos:

- Primeiro: Verifica se o r é alto o bastante para ser interessante.
- Depois: Confirma se o p-valor é baixo o suficiente para dizer que essa relação é estatisticamente confiável.

Variável	Correlação (r)	p-valor	Interpretação
Age	0.6975	0.0000	Correlação forte e significativa
imc	0.1038	0.0011	Correlação fraca, mas significativa
NumberOfMajorSurgeries	0.2642	0.0000	Correlação moderada e significativa
Height	0.0269	0.3986	Sem correlação significativa
Weight	0.1415	0.0000	Correlação fraca e significativa

In [683]:

```
# Plotar o heatmap
variaveis = ['Age', 'imc', 'NumberOfMajorSurgeries', 'Height', 'Weight']
plotar_heatmap_correlacoes(df_dados, variaveis, target='PremiumPrice')
```



In [684]:

```
# Salvando o dataset
df_dados.to_csv('./data/dados_projeto.csv', index = False)
```

4 - Modelagem

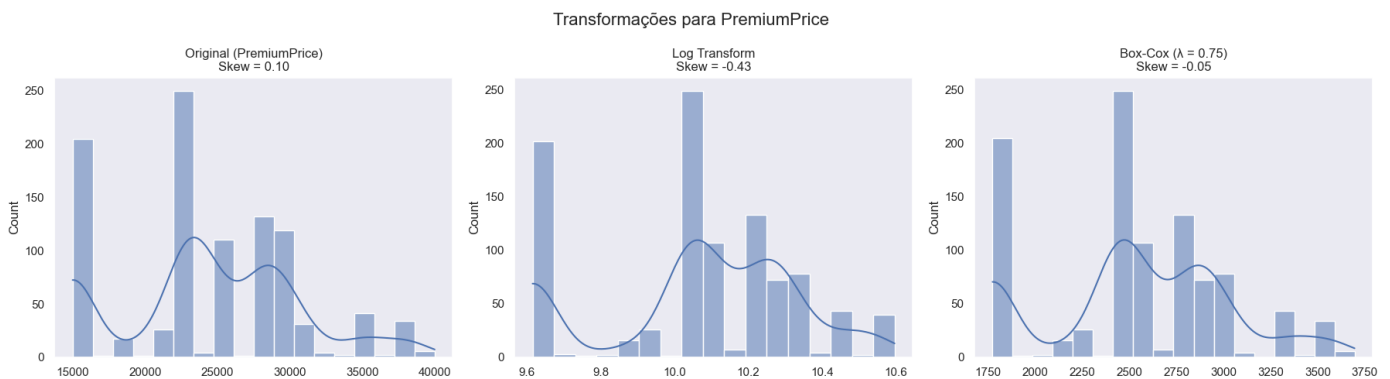
- Construção, teste e validação para diferentes modelos. A modelagem pode ser estatística (quando o interesse é em compreender o relacionamento entre os dados) ou preditiva (quando o interesse é em fazer previsões).
- Para este projeto vamos trabalhar na modelagem preditiva e criaremos diferentes modelos, sendo cada modelo com sua própria abordagem de cálculos e estatísticas.

In [685]:


```
# Função de transformação log natural e box_cox
resultado = transformar_variavel(df_dados['PremiumPrice'], nome_variavel="PremiumPrice")

# Acessar versões transformadas
df_transformacao = pd.DataFrame(
    {
        'original': np.array(resultado['original']),
        'boxcox': np.array(resultado['boxcox']),
        'log': np.array(resultado['log'])
    }
)

df_dados['boxcox'] = df_transformacao['boxcox']
df_dados['log'] = df_transformacao['log']
df_transformacao.head()
```



Out[685]:

	original	boxcox	log
0	25000	2600.102569	10.126631
1	29000	2905.489685	10.275051
2	23000	2442.837412	10.043249
3	28000	2830.197215	10.239960
4	23000	2442.837412	10.043249

5 - Comparação entre Random Forest e XGBoost

A escolha dos modelos Random Forest e XGBoost se justifica principalmente pela natureza do conjunto de dados, que combina variáveis numéricas e categóricas, presença de outliers e relações não-lineares. Ambos os modelos, por serem baseados em árvores, lidam bem com essa heterogeneidade e não exigem transformações extensas, além de serem robustos a distribuições assimétricas e outliers. Outro ponto relevante é a capacidade desses algoritmos de capturar interações complexas entre variáveis, o que é útil diante das relações não-lineares identificadas na análise exploratória.

Além disso, os dois modelos fornecem medidas claras de importância das variáveis, o que contribui para interpretar quais fatores mais influenciam o custo do seguro. O tamanho do dataset (986 registros) é adequado para modelos baseados em ensemble, como o Random Forest (bagging) e o XGBoost (boosting), ambos com histórico de bom desempenho em problemas de regressão. A comparação entre os

dois permite avaliar a abordagem mais eficaz para o problema, especialmente considerando o pré-processamento simples exigido por eles.

In [686]:

```
# Definir função para avaliar e comparar modelos
def avaliar_modelo(modelo, X_train, X_test, y_train, y_test, nome_modelo):
    # Treinar o modelo
    modelo.fit(X_train, y_train)
    y_pred = modelo.predict(X_test)

    # Calcular métricas
    mse = mean_squared_error(y_test, y_pred)
    rmse = np.sqrt(mse)
    r2 = r2_score(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)

    # Validação cruzada
    cv_scores = cross_val_score(modelo, X_train, y_train, cv=5, scoring='neg_mean_square
    cv_rmse = np.sqrt(-cv_scores.mean())

    # Imprimir resultados
    print(f"\nModelo: {nome_modelo}")
    print(f"Média quadrática (MSE): {mse:.2f}")
    print(f"Raiz média quadrática (RMSE): {rmse:.2f}")
    print(f"Média absoluta (MAE): {mae:.2f}")
    print(f"R² Score: {r2:.4f}")
    print(f"RMSE com validação cruzada (5-fold): {cv_rmse:.2f}")

    return {
        'modelo': modelo,
        'nome': nome_modelo,
        'mse': mse,
        'rmse': rmse,
        'mae': mae,
        'r2': r2,
        'cv_rmse': cv_rmse,
        'y_pred': y_pred
    }
```

In [687]:

```
# Comparação entre Random Forest e XGBoost
import xgboost as xgb
from sklearn.model_selection import cross_val_score
from sklearn.metrics import mean_squared_error, r2_score, mean_absolute_error
```

In [688]:

```
# Criar e avaliar modelo Random Forest
rf_model = RandomForestRegressor(n_estimators=100, random_state=42)
rf_results = avaliar_modelo(rf_model, X_train_scaled, X_test_scaled, y_train, y_test, "R
```

```
Modelo: Random Forest
Média quadrática (MSE): 166207.58
Raiz média quadrática (RMSE): 407.69
Média absoluta (MAE): 137.02
R² Score: 0.9961
RMSE com validação cruzada (5-fold): 794.44
```

In [689]:

```
# Criar e avaliar modelo XGBoost
xgb_model = xgb.XGBRegressor(
    objective='reg:squarederror',
    n_estimators=100,
    learning_rate=0.1,
    max_depth=5,
    random_state=42
)
xgb_results = avaliar_modelo(xgb_model, X_train_scaled, X_test_scaled, y_train, y_test,
```

Modelo: XGBoost
Média quadrática (MSE): 339360.97
Raiz média quadrática (RMSE): 582.55
Média absoluta (MAE): 242.45
R² Score: 0.9920
RMSE com validação cruzada (5-fold): 928.73

In [690]:

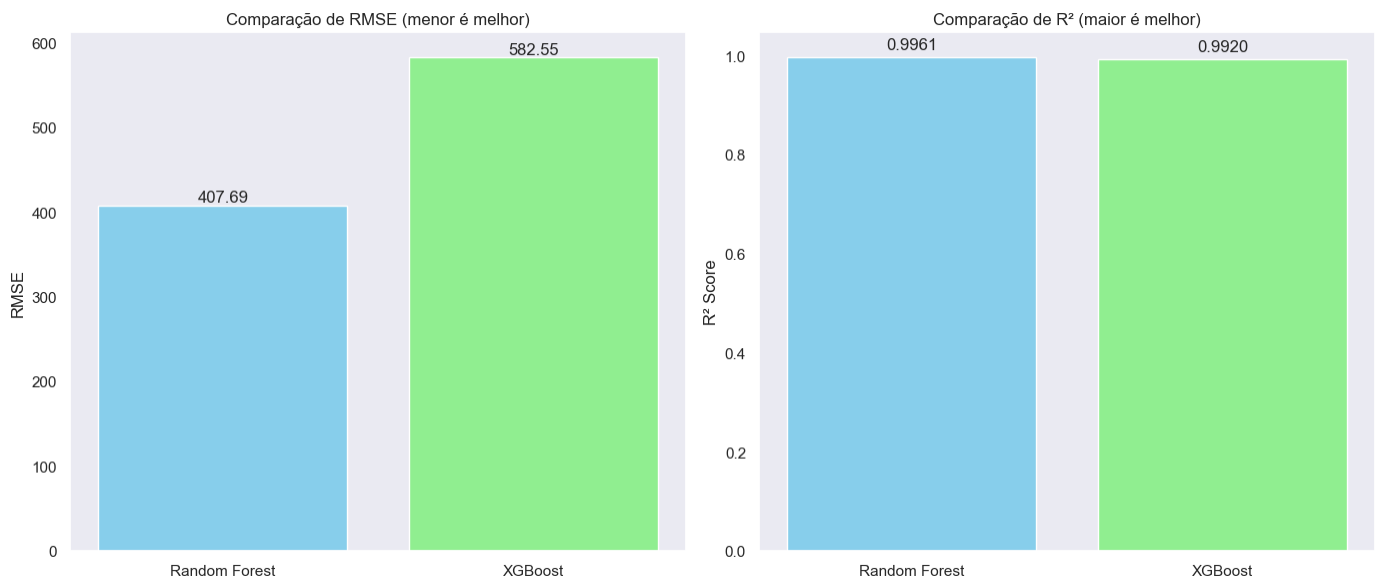
```
# Comparar visualmente os modelos
modelos = [rf_results, xgb_results]
nomes = [resultado['nome'] for resultado in modelos]
rmse_valores = [resultado['rmse'] for resultado in modelos]
r2_valores = [resultado['r2'] for resultado in modelos]

# Criar gráficos de comparação
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(14, 6))

# Gráfico RMSE (menor é melhor)
barras1 = ax1.bar(nomes, rmse_valores, color=['skyblue', 'lightgreen'])
ax1.set_ylabel('RMSE')
ax1.set_title('Comparação de RMSE (menor é melhor)')
for barra in barras1:
    altura = barra.get_height()
    ax1.text(barra.get_x() + barra.get_width()/2., altura + 0.1,
             f'{altura:.2f}', ha='center', va='bottom')

# Gráfico R² (maior é melhor)
barras2 = ax2.bar(nomes, r2_valores, color=['skyblue', 'lightgreen'])
ax2.set_ylabel('R² Score')
ax2.set_title('Comparação de R² (maior é melhor)')
for barra in barras2:
    altura = barra.get_height()
    ax2.text(barra.get_x() + barra.get_width()/2., altura + 0.01,
             f'{altura:.4f}', ha='center', va='bottom')

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



In [691]:

```
# Comparar importância de features
# Criar DataFrames para importância de features
rf_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': rf_model.feature_importances_
}).sort_values('Importance', ascending=False)

xgb_importance = pd.DataFrame({
    'Feature': X.columns,
    'Importance': xgb_model.feature_importances_
}).sort_values('Importance', ascending=False)
```

In [692]:

```
# Mostrar top 10 features para cada modelo
print("\nTop 10 features mais importantes - Random Forest:")
print(rf_importance.head(10))
```

Top 10 features mais importantes - Random Forest:

	Feature	Importance
20	faixa_preco_Baixo	0.613743
25	range_(28000.0, 40000.0]	0.228995
9	NumberOfMajorSurgeries	0.053265
3	AnyTransplants	0.031058
0	Age	0.028114
6	Weight	0.014346
4	AnyChronicDiseases	0.011192
22	faixa_preco_Médio-Baixo	0.002999
10	ordem_faixa	0.002832
24	range_(23000.0, 28000.0]	0.002228

In [693]:

```
print("\nTop 10 features mais importantes - XGBoost:")
print(xgb_importance.head(10))
```

Top 10 features mais importantes - XGBoost:

	Feature	Importance
20	faixa_preco_Baixo	0.750608
25	range_(28000.0, 40000.0]	0.164301
9	NumberOfMajorSurgeries	0.031958
3	AnyTransplants	0.022819

```

21         faixa_preco_Médio-Alto      0.009999
4         AnyChronicDiseases           0.009078
6         Weight                       0.003889
0         Age                         0.003315
16 categoria_imc_Obesidade grau II    0.000762
5         Height                      0.000687

```

In [694]:

```

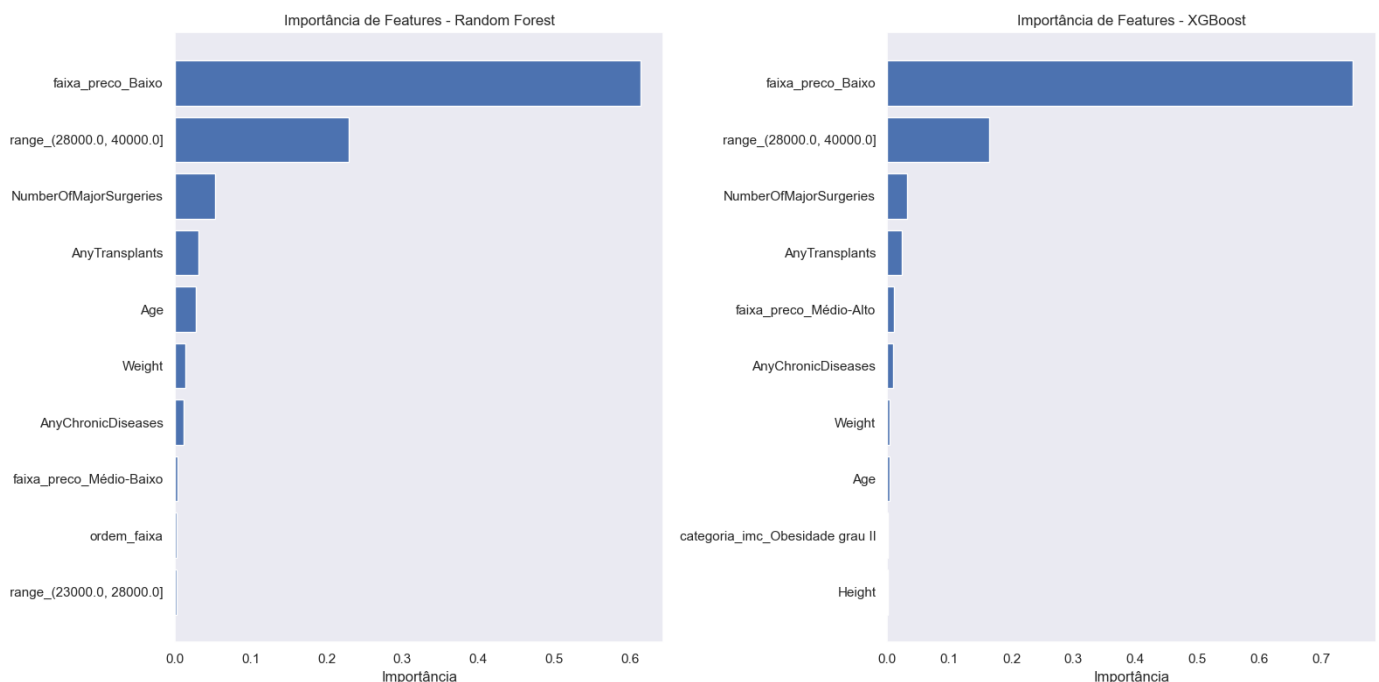
# Visualizar top 10 features para cada modelo
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(16, 8))

# Random Forest
ax1.barh(rf_importance.head(10)['Feature'][:-1], rf_importance.head(10)['Importance'][:])
ax1.set_title('Importância de Features - Random Forest')
ax1.set_xlabel('Importância')

# XGBoost
ax2.barh(xgb_importance.head(10)['Feature'][:-1], xgb_importance.head(10)['Importance'][:])
ax2.set_title('Importância de Features - XGBoost')
ax2.set_xlabel('Importância')

plt.tight_layout()
plt.show()

```



In [695]:

```

# Análise de resíduos
fig, axes = plt.subplots(2, 2, figsize=(16, 12))

# Random Forest - Resíduos vs Valores Previstos
residuos_rf = y_test - rf_results['y_pred']
axes[0, 0].scatter(rf_results['y_pred'], residuos_rf, alpha=0.5)
axes[0, 0].axhline(y=0, color='r', linestyle='-')
axes[0, 0].set_title('Resíduos vs Valores Previstos - Random Forest')
axes[0, 0].set_xlabel('Valores Previstos')
axes[0, 0].set_ylabel('Resíduos')

# XGBoost - Resíduos vs Valores Previstos
residuos_xgb = y_test - xgb_results['y_pred']

```

```

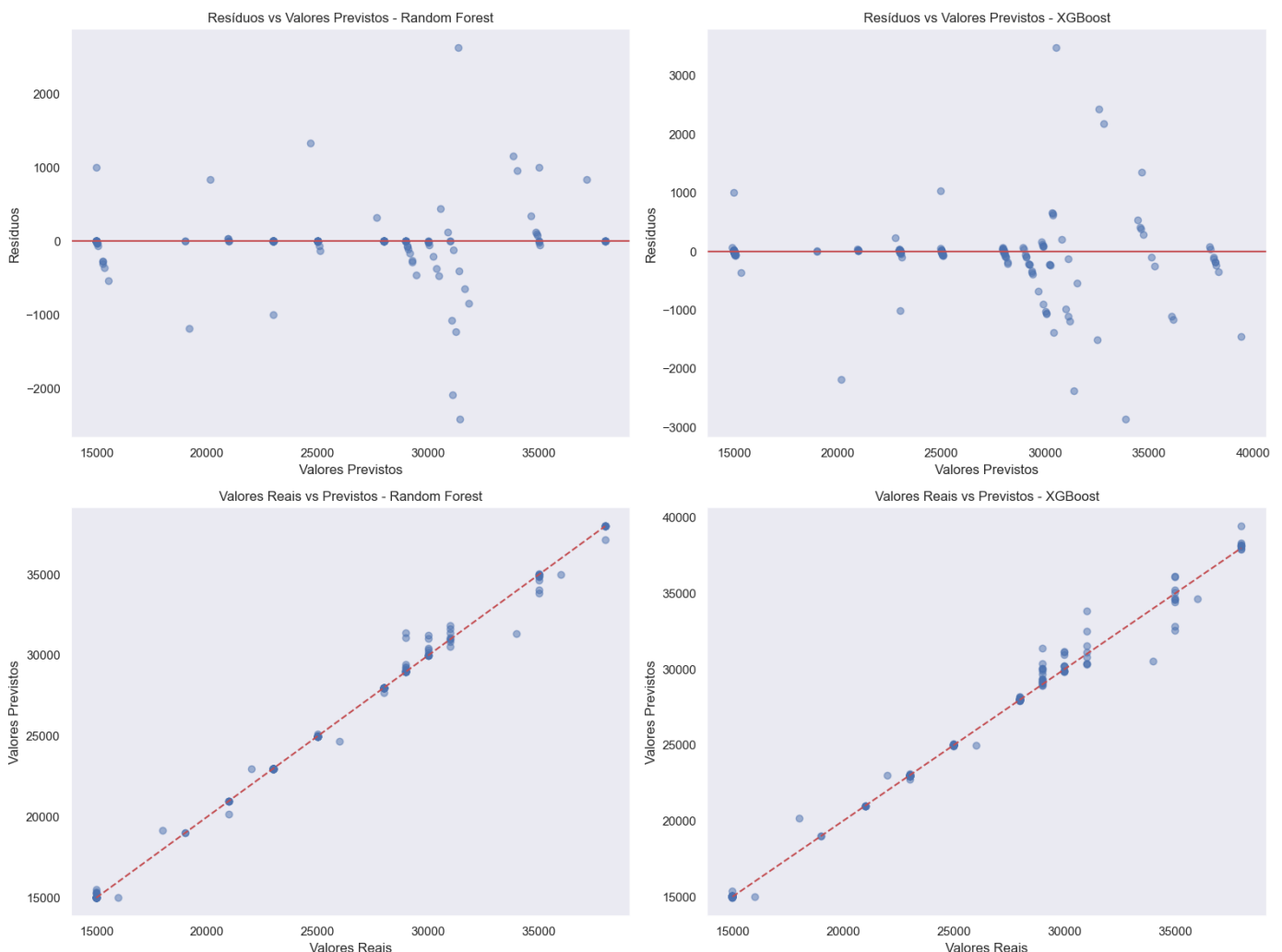
axes[0, 1].scatter(xgb_results['y_pred'], residuos_xgb, alpha=0.5)
axes[0, 1].axhline(y=0, color='r', linestyle='--')
axes[0, 1].set_title('Resíduos vs Valores Previstos - XGBoost')
axes[0, 1].set_xlabel('Valores Previstos')
axes[0, 1].set_ylabel('Resíduos')

# Random Forest - Valores Reais vs Previstos
axes[1, 0].scatter(y_test, rf_results['y_pred'], alpha=0.5)
axes[1, 0].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1, 0].set_title('Valores Reais vs Previstos - Random Forest')
axes[1, 0].set_xlabel('Valores Reais')
axes[1, 0].set_ylabel('Valores Previstos')

# XGBoost - Valores Reais vs Previstos
axes[1, 1].scatter(y_test, xgb_results['y_pred'], alpha=0.5)
axes[1, 1].plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')
axes[1, 1].set_title('Valores Reais vs Previstos - XGBoost')
axes[1, 1].set_xlabel('Valores Reais')
axes[1, 1].set_ylabel('Valores Previstos')

plt.tight_layout()
plt.show()

```



In [696]:

```

# Comparar distribuição dos erros
plt.figure(figsize=(12, 6))

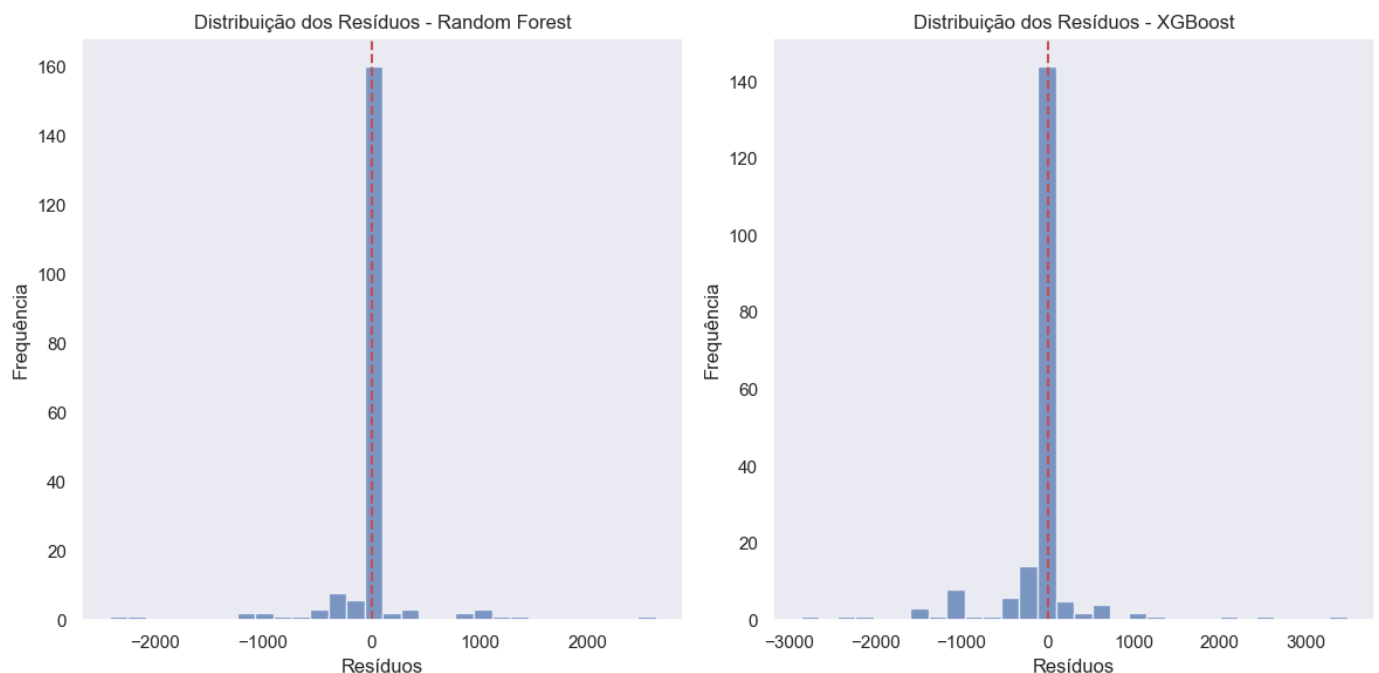
plt.subplot(1, 2, 1)

```

```
plt.hist(residuos_rf, bins=30, alpha=0.7, label='Random Forest')
plt.axvline(x=0, color='r', linestyle='--')
plt.title('Distribuição dos Resíduos - Random Forest')
plt.xlabel('Resíduos')
plt.ylabel('Frequência')

plt.subplot(1, 2, 2)
plt.hist(residuos_xgb, bins=30, alpha=0.7, label='XGBoost')
plt.axvline(x=0, color='r', linestyle='--')
plt.title('Distribuição dos Resíduos - XGBoost')
plt.xlabel('Resíduos')
plt.ylabel('Frequência')

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



In [697]:

```
# Determinar o melhor modelo
if xgb_results['rmse'] < rf_results['rmse']:
    print(f"\n0 XGBoost teve melhor desempenho com RMSE de {xgb_results['rmse']:.2f} vs
    melhor_modelo = xgb_model
    melhor_nome = "XGBoost"
else:
    print(f"\n0 Random Forest teve melhor desempenho com RMSE de {rf_results['rmse']:.2f}
    melhor_modelo = rf_model
    melhor_nome = "Random Forest"

print(f"Melhoria percentual: {abs(rf_results['rmse'] - xgb_results['rmse']) / max(rf_res
```

0 Random Forest teve melhor desempenho com RMSE de 407.69 vs 582.55 do XGBoost
Melhoria percentual: 30.02%

6 - Deploy do Modelo

- Aqui usamos o modelo final com novos dados para então resolver o problema para o qual ele foi criado.

In [698]:

```
# Carrega os dados do pré processamento
df = pd.read_csv('./data/dados_projeto.csv')

# Separa features e target
X = df.drop('PremiumPrice', axis=1)
Y = df['PremiumPrice']

# Exibir estatística básica da variável alvo
print("\nTarget variable statistics:")
print(f"Média: {Y.mean():.2f}")
print(f"Mediana: {Y.median():.2f}")
print(f"Min: {Y.min():.2f}")
print(f"Max: {Y.max():.2f}")
```

Target variable statistics:

Média: 24336.71

Mediana: 23000.00

Min: 15000.00

Max: 40000.00

Aplicar técnica one-hot para converter variáveis categóricas em formato que pode ser lido pelos os algoritmos

Exemplo: Para a coluna "categoria_imc", serão criadas colunas como:

- categoria_imc_Abaixo do peso (0 ou 1)
- categoria_imc_Peso normal (0 ou 1)
- categoria_imc_Sobrepeso (0 ou 1)

Com drop_first=True, a primeira categoria é omitida e representada quando todas as demais forem 0.

In [699]:

```
X = pd.get_dummies(X, drop_first=True)
print(f" Features depois do encoding: {X.shape}")
```

Features depois do encoding: (986, 26)

In [700]:

```
# Separar o dataset em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print(f"Dataset de treinamento: {X_train.shape}, Dataset de teste: {X_test.shape}")
```

Dataset de treinamento: (788, 26), Dataset de teste: (198, 26)

In [701]:

```
# Normalização das features usando StandardScaler
# Transforma os dados para que cada feature tenha média 0 e desvio padrão 1
# Ajusta o scaler apenas nos dados de treinamento
```



```

scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

model=RandomForestRegressor(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

```

Out[701]:

```

RandomForestRegressor
RandomForestRegressor(random_state=42)

```

7 - Prever o valor do prêmio de seguro médico

- O objetivo é usar um exemplo de dados de um cliente para prever o valor do prêmio de seguro médico

In [702]:

```

#Montagem do dataset do cliente

dados_cliente = pd.DataFrame({
    'Age': [45],
    'Diabetes': [0],
    'BloodPressureProblems': [0],
    'AnyTransplants': [0],
    'AnyChronicDiseases': [0],
    'Height': [170],
    'Weight': [70],
    'KnownAllergies': [0],
    'HistoryOfCancerInFamily': [0],
    'NumberOfMajorSurgeries': [0]
})

# Calculando características derivadas automaticamente
# Calculando IMC
altura_m = dados_cliente['Height'].values[0] / 100
dados_cliente['imc'] = dados_cliente['Weight'].values[0] / (altura_m ** 2)

# Definindo faixa etária
idade = dados_cliente['Age'].values[0]
if idade < 18:
    dados_cliente['faixa_etaria'] = 'Criança'
    dados_cliente['ordem_faixa'] = 1
elif idade < 25:
    dados_cliente['faixa_etaria'] = 'Jovem'
    dados_cliente['ordem_faixa'] = 2
elif idade < 35:
    dados_cliente['faixa_etaria'] = 'Jovem Adulto'
    dados_cliente['ordem_faixa'] = 3
elif idade < 45:
    dados_cliente['faixa_etaria'] = 'Adulto'
    dados_cliente['ordem_faixa'] = 4
elif idade < 55:
    dados_cliente['faixa_etaria'] = 'Adulto'
    dados_cliente['ordem_faixa'] = 5
elif idade < 65:

```

```

    dados_cliente['faixa_etaria'] = 'Meia-idade'
    dados_cliente['ordem_faixa'] = 6
else:
    dados_cliente['faixa_etaria'] = 'Idoso'
    dados_cliente['ordem_faixa'] = 7

# Definindo categoria de IMC
imc = dados_cliente['imc'].values[0]
if imc < 18.5:
    dados_cliente['categoria_imc'] = 'Abaixo do peso'
elif imc < 25:
    dados_cliente['categoria_imc'] = 'Peso normal'
elif imc < 30:
    dados_cliente['categoria_imc'] = 'Sobrepeso'
elif imc < 35:
    dados_cliente['categoria_imc'] = 'Obesidade Grau I'
elif imc < 40:
    dados_cliente['categoria_imc'] = 'Obesidade Grau II'
else:
    dados_cliente['categoria_imc'] = 'Obesidade Grau III'

# Adicionando faixa de preço (apenas para completar o dataset)
dados_cliente['faixa_preco'] = 'Médio'
dados_cliente['range'] = 2

```

In [703]:

```

# Convertendo variáveis categóricas para dummies
dados_codificados = pd.get_dummies(dados_cliente, drop_first=True)

for col in X.columns:
    if col not in dados_codificados.columns:
        dados_codificados[col] = 0

# Garantindo que as colunas estejam na mesma ordem do treinamento
dados_codificados = dados_codificados[X.columns]
dados_normalizado = scaler.transform(dados_codificados)

# Fazendo a previsão
preco_previsto = model.predict(dados_normalizado)

print(f"Características do cliente: {idade} anos, IMC {imc:.1f} ({dados_cliente['categoria_imc']})")
print(f"Faixa etária: {dados_cliente['faixa_etaria'].values[0]}")
print(f"Faixa de preço: {dados_cliente['faixa_preco'].values[0]}")
print(f"\nPreço previsto: {preco_previsto[0]:.2f}")

```

Características do cliente: 45 anos, IMC 24.2 (Peso normal)

Faixa etária: Adulto

Faixa de preço: Médio

Preço previsto: 23970.00

FIM