

File Edit View Run Kernel Tabs Settings Help

Terminal1 tech_challenge.ipynb + Python 3 (ipykernel) ⌂

Objetivo

Criar um modelo preditivo e comprovar sua eficácia com métricas estatísticas.

Dicionário dos dados

- Idade: idade do beneficiário principal.
- Gênero: gênero do contratante de seguros.
- IMC: índice de massa corporal, fornecendo uma compreensão do corpo, pesos relativamente altos ou baixos em relação à altura.
- Filhos: número de filhos cobertos por seguro saúde / Número de dependentes.
- Fumante: se a pessoa fuma (sim ou não).
- Região: a área residencial do beneficiário nos EUA, nordeste, sudeste, sudoeste ou noroeste.
- Encargos: custos médicos individuais cobrados pelo seguro de saúde.

```
[1]: # 01 ===[ Importando as bibliotecas necessárias ]=====
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import numpy as np

from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
```

```
[2]: # 02 ===[ Subindo os dados para um dataframe ]=====
df_pacientes = pd.read_csv('./insurance.csv')
```

```
[3]: # 03 ===[ Conhecendo os Dados ]=====
# 03.1 - As cinco primeiras linhas:
df_pacientes.head()
```

| | age | sex | bmi | children | smoker | region | charges |
|---|-----|--------|--------|----------|--------|-----------|-------------|
| 0 | 19 | female | 27.900 | 0 | yes | southwest | 16884.92400 |
| 1 | 18 | male | 33.770 | 1 | no | southeast | 1725.55230 |
| 2 | 28 | male | 33.000 | 3 | no | southeast | 4449.46200 |
| 3 | 33 | male | 22.705 | 0 | no | northwest | 21984.47061 |
| 4 | 32 | male | 28.880 | 0 | no | northwest | 3866.85520 |

```
[4]: # 03.2 - As cinco últimas linhas:
df_pacientes.tail()
```

| | age | sex | bmi | children | smoker | region | charges |
|------|-----|--------|-------|----------|--------|-----------|------------|
| 1333 | 50 | male | 30.97 | 3 | no | northwest | 10600.5483 |
| 1334 | 18 | female | 31.92 | 0 | no | northeast | 2205.9808 |
| 1335 | 18 | female | 36.85 | 0 | no | southeast | 1629.8335 |
| 1336 | 21 | female | 25.80 | 0 | no | southwest | 2007.9450 |
| 1337 | 61 | female | 29.07 | 0 | yes | northwest | 29141.3603 |

Alvo:

Predizer o custo hospitalar.
Mais especificamente o campo charges.

```
[5]: # 03.3 - Dimensões:
df_pacientes.shape
```

```
[5]: (1338, 7)
```

7 dimensões com 1338 registros.

```
[6]: # 03.4 - Metadados:
df_pacientes.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1338 entries, 0 to 1337
Data columns (total 7 columns):
 #   Column   Non-Null Count  Dtype  
 --- 
 0   age      1338 non-null   int64  
 1   sex      1338 non-null   object 
 2   bmi      1338 non-null   float64 
 3   children 1338 non-null   int64  
 4   smoker    1338 non-null   object 
 5   region    1338 non-null   object 
 6   charges   1338 non-null   float64 
dtypes: float64(2), int64(2), object(3)
memory usage: 73.3+ KB
```

Fatos:

- Não existem dados nulos.
- 3 são do tipo object: sex, smoker e region.
- 4 são numéricos.
- Base pequena: 73.3KB.

```
[7]: # 03.5 - Verificando valores únicos por coluna:
df_pacientes.nunique()
```

```
[7]: age      47
sex       2
bmi     548
children   6
smoker     2
region     4
charges   1337
dtype: int64
```

Categóricas Binárias:

Como a prática mais moderna e segura para o pré-processamento do modelo é usar OneHotEncoder (ou pd.get_dummies) para todas as variáveis categóricas, incluindo as binárias.
Mesmo tendo somente 2 tipos de valores:

- sex
- somker

Será utilizado OnHotEncoder.

```
[8]: # 03.6 - Verificando os tipos existentes em "sex":
set(df_pacientes['sex'])
```

```
[8]: {'female', 'male'}
```

```
[9]: # 03.7 - Verificando os tipos existentes em "smoker":
set(df_pacientes['smoker'])
```

```
[9]: {'no', 'yes'}
```

```
[10]: # 03.8 - Verificando os tipos existentes em "region":
set(df_pacientes['region'])
```

```
[10]: {'northeast', 'northwest', 'southeast', 'southwest'}
```

```
[11]: # 03.9 - Estatísticas descritivas:
df_pacientes.describe()
```

| | age | bmi | children | charges |
|-------|-------------|-------------|-------------|--------------|
| count | 1338.000000 | 1338.000000 | 1338.000000 | 1338.000000 |
| mean | 39.207025 | 30.663397 | 1.094918 | 13270.422265 |
| std | 14.049960 | 6.098187 | 1.205493 | 12110.011237 |
| min | 18.000000 | 15.960000 | 0.000000 | 1121.873900 |
| 25% | 27.000000 | 26.296250 | 0.000000 | 4740.287150 |
| 50% | 39.000000 | 30.400000 | 1.000000 | 9382.033000 |
| 75% | 51.000000 | 34.693750 | 2.000000 | 16639.912515 |
| max | 64.000000 | 53.130000 | 5.000000 | 63770.428010 |

Em média temos uma idade de 39 anos, com BMI (IMC) 30, 01 filho e custo hospitalar de 13270.

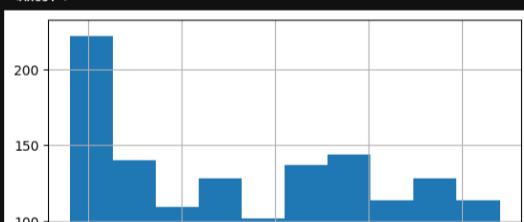
```
[12]: # 03.10 - Verificando valores duplicados:
df_pacientes.duplicated().sum()
```

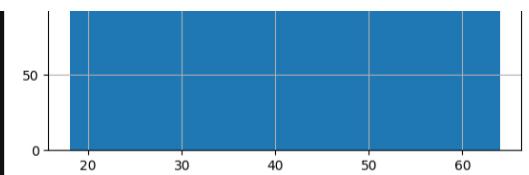
```
[12]: np.int64(1)
```

Não existem valores duplicados.

```
[13]: # 04 ===[ Análise de Cada Feature Individualmente ]=====
# 04.1 - Histograma da idade:
df_pacientes['age'].hist()
```

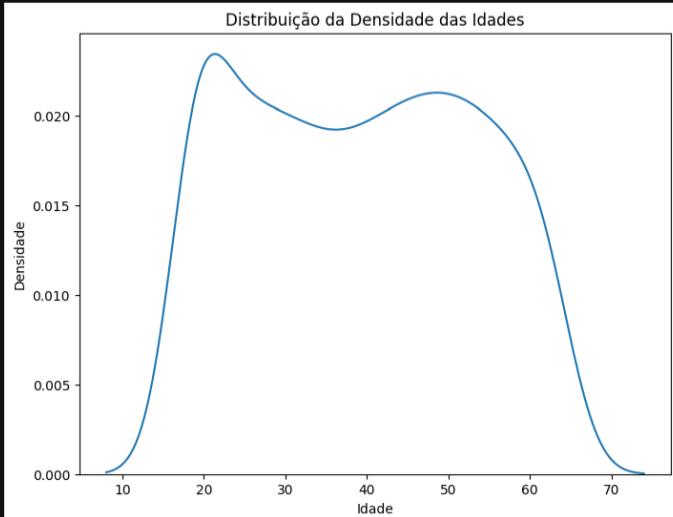
```
[13]: <Axes: >
```





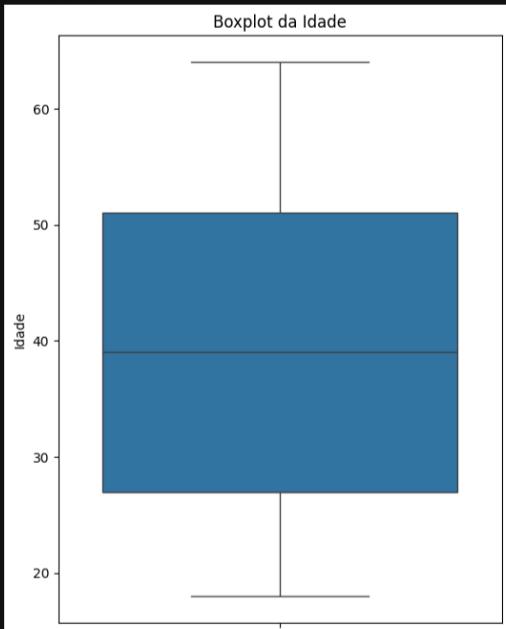
```
[14]: # 04_2 - Distribuição da densidade no campo "age":
plt.figure(figsize=(8, 6))
sns.kdeplot(data=df_pacientes, x='age')
plt.title('Distribuição da Densidade das Idades')
plt.xlabel('Idade')
plt.ylabel('Densidade')
```

```
[14]: Text(0, 0.5, 'Densidade')
```



```
[15]: # 04_3 - Boxplot do campo "age":
plt.figure(figsize=(6, 8))
sns.boxplot(y=df_pacientes['age'])
plt.title('Boxplot da Idade')
plt.ylabel('Idade')
```

```
[15]: Text(0, 0.5, 'Idade')
```

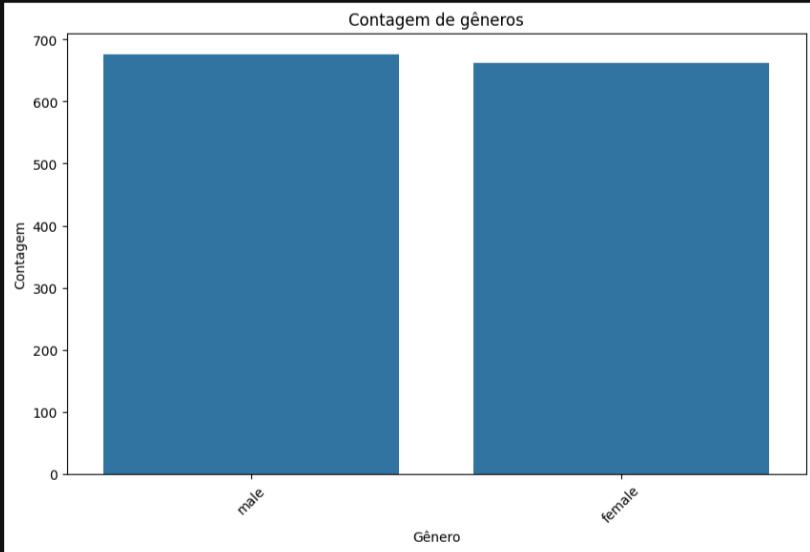


Sobre as idades:

A maioria tem 20 anos.
As demais idades ficam em uma quantidade registrada de 125 em média.
O histograma contém um gráfico assíncrono, não normatizado.
Essa feature numérica será padronizada com StandardScaler, para não interferir no treinamento da regressão linear.
Não existem outliers.

```
[16]: # 04_4 - Countplot do campo "sex":
plt.figure(figsize=(10, 6))
sns.countplot(x='sex', data=df_pacientes, order=df_pacientes['sex'].value_counts().index)
plt.title('Contagem de gêneros')
plt.xlabel('Gênero')
plt.ylabel('Contagem')
plt.xticks(rotation=45)
```

```
[16]: ([0, 1], [Text(0, 0, 'male'), Text(1, 0, 'female')])
```

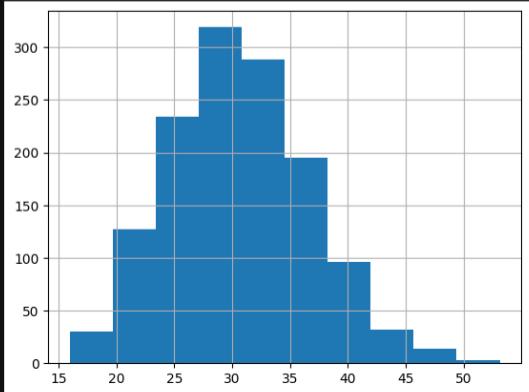


Sobre os gêneros - sex:

Quantidade entre os dois gêneros é quase igual.

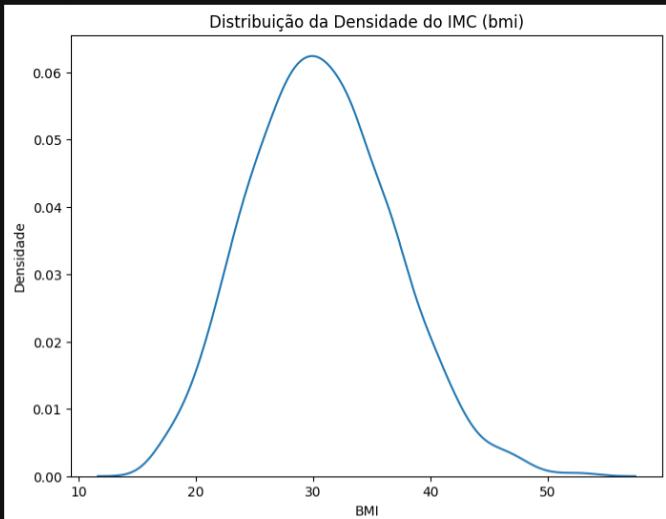
```
[17]: # 04.5 - Histograma do campo "bmi":  
df_pacientes['bmi'].hist()
```

```
[17]: <Axes: >
```



```
[18]: # 04.6 - Distribuição da densidade do campo "bmi":  
plt.figure(figsize=(8, 6))  
sns.kdeplot(data=df_pacientes, x='bmi')  
plt.title('Distribuição da Densidade do IMC (bmi)')  
plt.xlabel('BMI')  
plt.ylabel('Densidade')
```

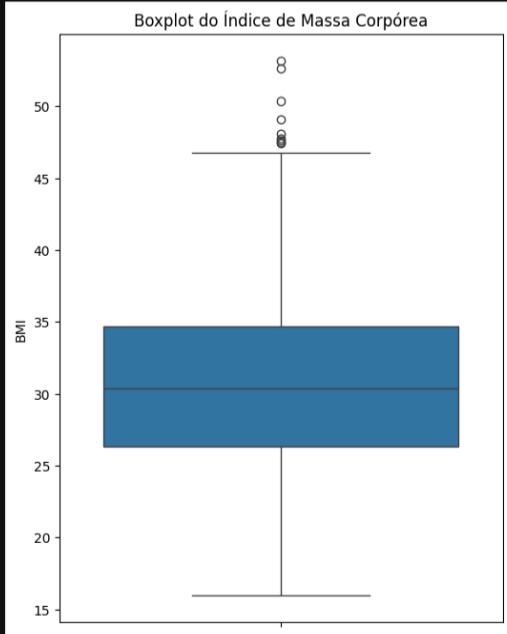
```
[18]: Text(0, 0.5, 'Densidade')
```



```
[19]: # 04.7 - Boxplot do campo "bmi":  
plt.figure(figsize=(6, 8))  
sns.boxplot(y=df_pacientes['bmi'])  
plt.title('Boxplot do Índice de Massa Corpórea')  
plt.xlabel('bmi')
```

```
[19]: plt.ylabel('BMI')
```

```
[19]: Text(0, 0.5, 'BMI')
```

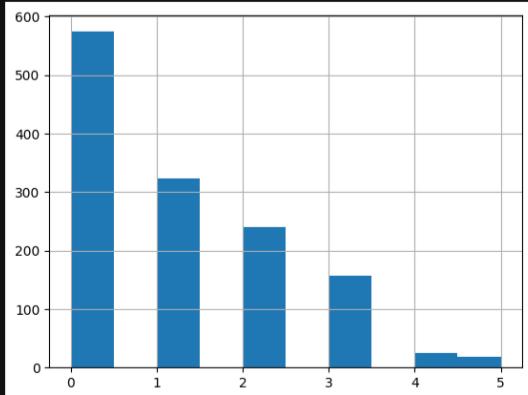


Sobre o BMI ou IMC:

Mais de 300 registros tem BMI 30.
O gráfico é quase Simétrico, distribuição quase normal.
Com tudo, será aplicado o StandardScaler por terem
valores altos e baixos que podem influenciar erronamente
o modelo.
Existem outliers superiores.
Estes outliers são reais e não devem ser removidos.
Será aplicado o StandardScaler.

```
[20]: # 04.8 - Histograma do campo "Children":  
df_pacientes['children'].hist()
```

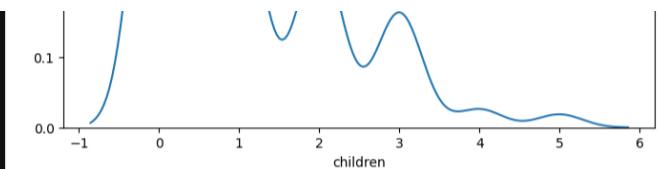
```
[20]: <Axes: >
```



```
[21]: # 04.9 - Distribuição da densidade de filhos, campo "Children":  
plt.figure(figsize=(8, 6))  
sns.kdeplot(data=df_pacientes, x='children')  
plt.title('Distribuição da Densidade da quantidade de filhos')  
plt.xlabel('children')  
plt.ylabel('Densidade')
```

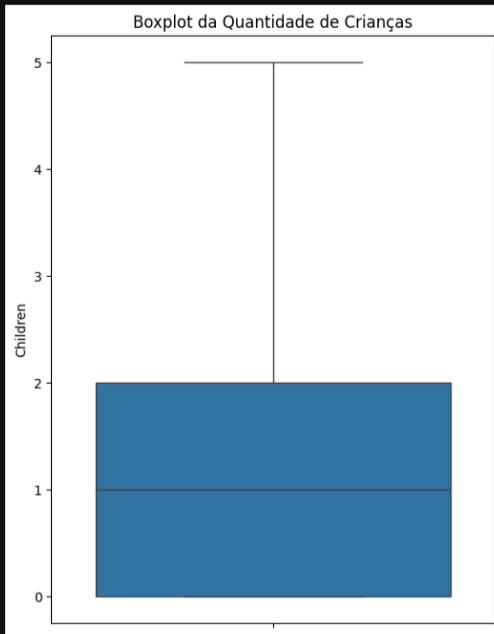
```
[21]: Text(0, 0.5, 'Densidade')
```





```
[22]: # 04.10 - Boxplot da quantidade de filhos, campo "Children":
plt.figure(figsize=(6, 8))
sns.boxplot(y=df_pacientes['children'])
plt.title('Boxplot da Quantidade de Crianças')
plt.ylabel('Children')
```

[22]: Text(0, 0.5, 'Children')

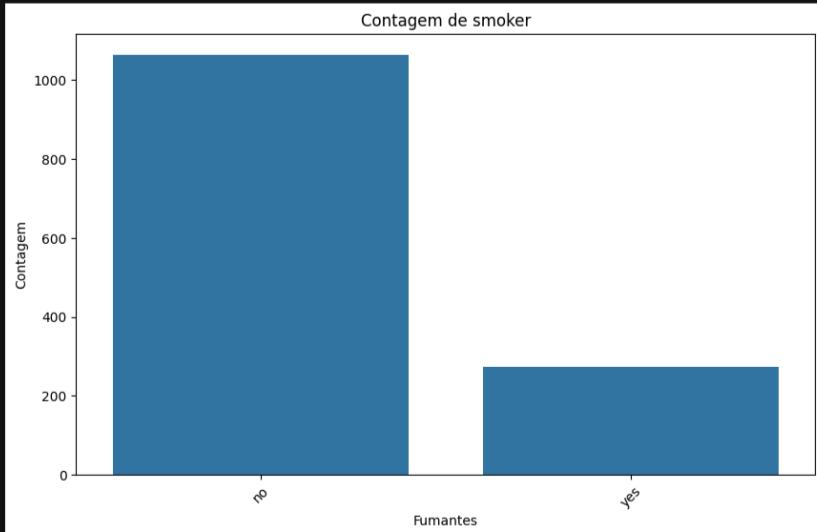


Sobre a quantidade de filhos:

Assimétrico: os dados não estão uniformemente distribuídos.
Será aplicado o Standardscaler.

```
[23]: # 04.11 - Countplot do campo "smoker":
plt.figure(figsize=(10, 6))
sns.countplot(x='smoker', data=df_pacientes, order=df_pacientes['smoker'].value_counts().index)
plt.title('Contagem de smoker')
plt.xlabel('Fumantes')
plt.ylabel('Contagem')
plt.xticks(rotation=45)
```

[23]: ([0, 1], [Text(0, 0, 'no'), Text(1, 0, 'yes')])

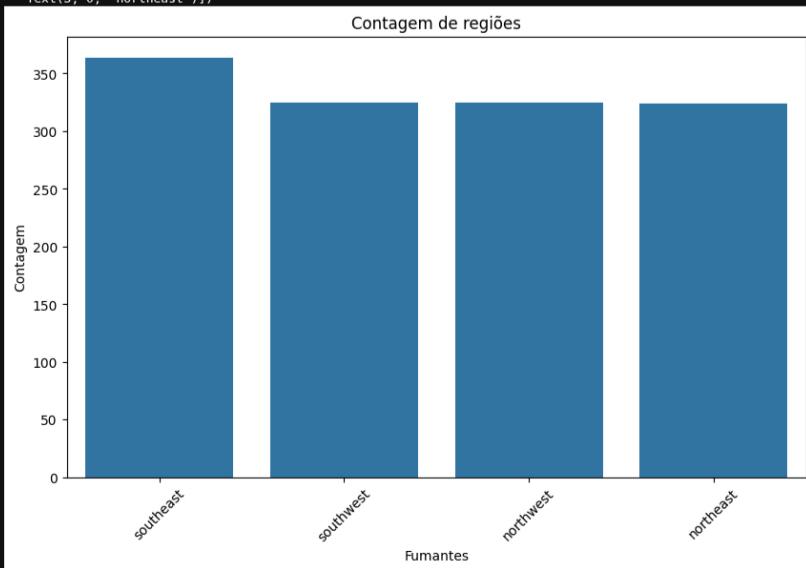


Sobre o campo **smoker**:

Os dados são muito distantes: fumantes e não fumantes;
isso em quantidades de um e outro.
Novamente, como a prática mais moderna e segura para o pré-processamento
do modelo é usar OneHotEncoder, incluindo colunas com apenas dois valores, será utilizado o
OneHotEncoder com este campo.

```
[24]: # 04.12 - Countplot do campo "region"
plt.figure(figsize=(10, 6))
sns.countplot(x='region', data=df_pacientes, order=df_pacientes['region'].value_counts().index)
plt.title('Contagem de regiões')
plt.xlabel('Fumantes')
plt.ylabel('Contagem')
plt.xticks(rotation=45)
```

```
[24]: ([0, 1, 2, 3],
[Text(0, 0, 'southeast'),
Text(1, 0, 'southwest'),
Text(2, 0, 'northwest'),
Text(3, 0, 'northeast')])
```

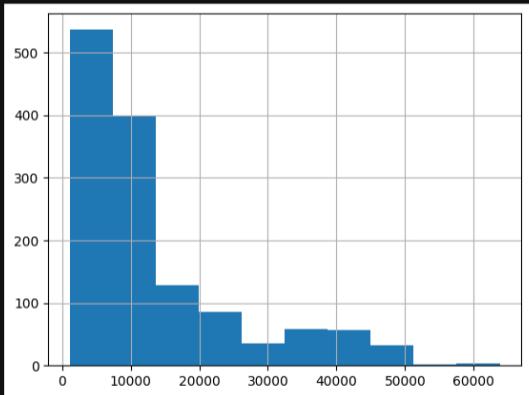


Sobre o campo region:

Os dados estão bem parecidos em termos das quantidade.
Aqui será aplicado um OneHotEncoder.

```
[25]: # 04.13 - Histograma dos custos hospitalares, campo "charges"
df_pacientes['charges'].hist()
```

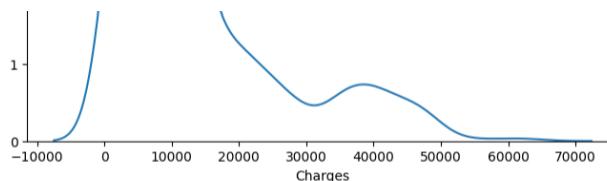
```
[25]: <Axes: >
```



```
[26]: # 04.14 - Distribuição da densidade do custo hospitalar, campo "charges"
plt.figure(figsize=(8, 6))
sns.kdeplot(data=df_pacientes, x='charges')
plt.title('Distribuição da Densidade do custo hospitalar')
plt.xlabel('Charges')
plt.ylabel('Densidade')
```

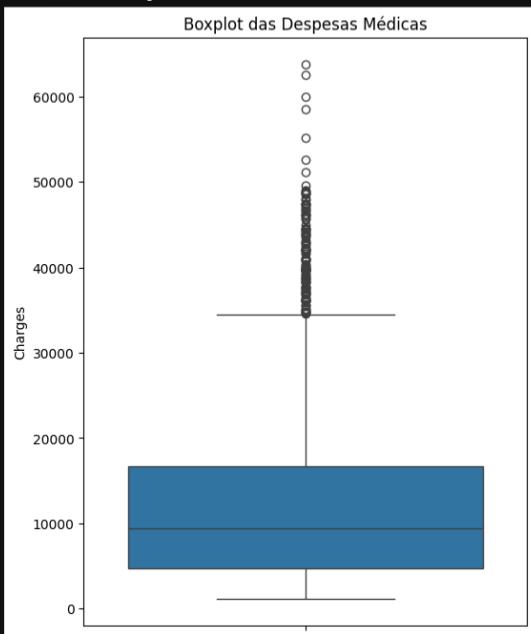
```
[26]: Text(0, 0.5, 'Densidade')
```





```
[27]: # 04.15 - Boxplot do campo "charges"
plt.figure(figsize=(6, 8))
sns.boxplot(y=df_pacientes['charges'])
plt.title('Boxplot das Despesas Médicas')
plt.ylabel('Charges')
```

```
[27]: Text(0, 0.5, 'Charges')
```



Sobre o charges, os custos médicos individuais:

A feature "charges" é o (alvo) da predição (o target).
Porém é assimétrico: os dados não estão uniformemente distribuídos.
Muitos outliers superiores, mas, reais; e por isso não serão removidos.
Será aplicado um cálculo logarítmico na coluna charges para evitar vícios
no modelo.
A função np.log1p do NumPy é ideal porque ela lida bem com
valores de zero (calcula $\log(1+x)$).

```
[28]: # 05 ===[ Padronizando e Normalizando ]=====
df_log = df_pacientes.copy() # copiando a base principal
df_log['charges'] = np.log1p(df_log['charges']) # normalizando charges com log1p.

X = df_log.drop('charges', axis=1) # X contém todas as colunas, exceto o alvo
y = df_log['charges'] # y contém apenas o alvo

numeric_features = ['age', 'bmi', 'children']
categorical_features = ['sex', 'smoker', 'region']

numeric_transformer = Pipeline(steps=[
    ('scaler', StandardScaler())
])

categorical_transformer = Pipeline(steps=[
    ('onehot', OneHotEncoder(handle_unknown='ignore', drop='first'))
])

preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ],
    remainder='passthrough' # Mantém as colunas não especificadas (se houver)
)

arr_npy_transformed = preprocessor.fit_transform(X)
arr_npy_new_categorical_columns = preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_features)
arr_npy_all_column_names = numeric_features + list(arr_npy_new_categorical_columns)

df_transformed = pd.DataFrame(arr_npy_transformed, columns=arr_npy_all_column_names)
df_transformed['charges'] = y.values
```

Agora temos um dataframe transformado e pronto para a continuação
da análise.

```
[29]: # 05.1 - Visualizando as cinco primeiros registros do datagrama já transformado:
df_transformed.head()
```

| | age | bmi | children | sex_male | smoker_yes | region_northwest | region_southeast | region_southwest | charges |
|---|-----------|-----------|-----------|----------|------------|------------------|------------------|------------------|----------|
| 0 | -1.438764 | -0.453320 | -0.908614 | 0.0 | 1.0 | 0.0 | 0.0 | 1.0 | 9.734236 |

| | | | | | | | | | |
|---|-----------|-----------|-----------|-----|-----|-----|-----|-----|----------|
| 1 | -1.509965 | 0.509621 | -0.078767 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 7.453882 |
| 2 | -0.797954 | 0.383307 | 1.580926 | 1.0 | 0.0 | 0.0 | 1.0 | 0.0 | 8.400763 |
| 3 | -0.441948 | -1.305531 | -0.908614 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 9.998137 |
| 4 | -0.513149 | -0.292556 | -0.908614 | 1.0 | 0.0 | 1.0 | 0.0 | 0.0 | 8.260455 |

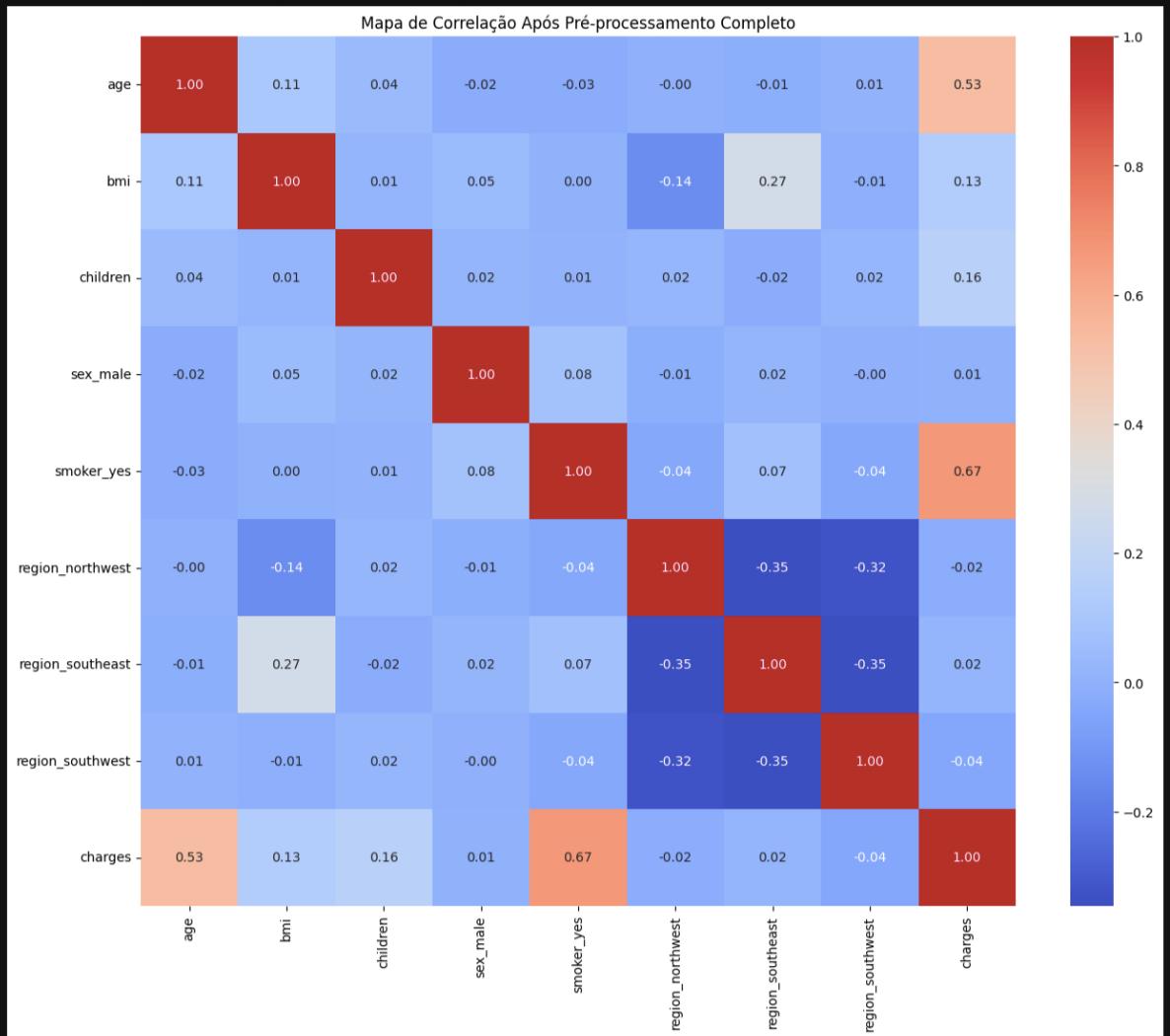
```
[30]: # 05.2 - Exibindo a correlação:  
df_transformed.corr()
```

| | age | bmi | children | sex_male | smoker_yes | region_northwest | region_southeast | region_southwest | charges |
|------------------|-----------|-----------|-----------|-----------|------------|------------------|------------------|------------------|-----------|
| age | 1.000000 | 0.109272 | 0.042469 | -0.020856 | -0.025019 | -0.000407 | -0.011642 | 0.010016 | 0.527807 |
| bmi | 0.109272 | 1.000000 | 0.012759 | 0.046371 | 0.003750 | -0.135996 | 0.270025 | -0.006205 | 0.132678 |
| children | 0.042469 | 0.012759 | 1.000000 | 0.017163 | 0.007673 | 0.024806 | -0.023066 | 0.021914 | 0.161317 |
| sex_male | -0.020856 | 0.046371 | 0.017163 | 1.000000 | 0.076185 | -0.011156 | 0.017117 | -0.004184 | 0.005644 |
| smoker_yes | -0.025019 | 0.003750 | 0.007673 | 0.076185 | 1.000000 | -0.036945 | 0.068498 | -0.036945 | 0.665539 |
| region_northwest | -0.000407 | -0.135996 | 0.024806 | -0.011156 | -0.036945 | 1.000000 | -0.346265 | -0.320829 | -0.017830 |
| region_southeast | -0.011642 | 0.270025 | -0.023066 | 0.017117 | 0.068498 | -0.346265 | 1.000000 | -0.346265 | 0.015803 |
| region_southwest | 0.010016 | -0.006205 | 0.021914 | -0.004184 | -0.036945 | -0.320829 | -0.346265 | 1.000000 | -0.041633 |
| charges | 0.527807 | 0.132678 | 0.161317 | 0.005644 | 0.665539 | -0.017830 | 0.015803 | -0.041633 | 1.000000 |

```
[31]: # 05.3 - Visualizado em um mapa de correlação:  
correlation_matrix = df_transformed.corr()
```

```
plt.figure(figsize=(15, 12))  
sns.heatmap(  
    correlation_matrix,  
    annot=True,  
    fmt='.2f',  
    cmap='coolwarm'  
)  
plt.title('Mapa de Correlação Após Pré-processamento Completo')
```

```
[31]: Text(0.5, 1.0, 'Mapa de Correlação Após Pré-processamento Completo')
```



Baixa correlação.

```
[32]: # 06 ===[ Criando o Pipeline Principal (único) ]=====  
# O primeiro passo é o pré-processador (que lida com todas as colunas),  
# criado anteriormente.  
# O segundo passo é o modelo de Regressão Linear.  
  
# 06.1 - Importando a lib do regreção linear:  
from sklearn.linear_model import LinearRegression  
  
# 06.2 - Pipeline principal do modelo de regressão linear:  
model_pipeline = Pipeline(steps=[
```

```

        ('preprocessor', preprocessor),
        ('regressor', LinearRegression())
    )

[33]: # 07 ===== Preparação dos Dados: Treino e Teste =====
# 07.1 - importando algumas lib necessárias:
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error, r2_score

[34]: # 07.2 - Aplicando a transformação log. no alvo (como planejado!).
# diretamente no dataframe [df_pacientes].
df_pacientes['charges'] = np.log1p(df_pacientes['charges'])

[35]: # 07.3 - Separando features (X) e alvo (y)
X = df_pacientes.drop('charges', axis=1)
y = df_pacientes['charges']

[36]: # 07.4 - Dividindo em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

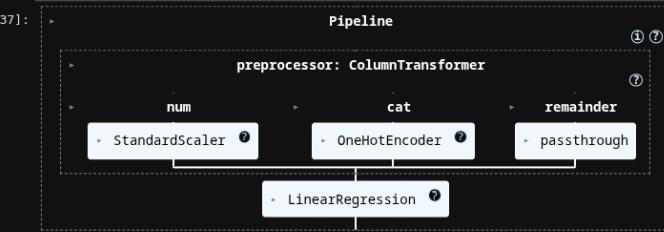
```

Vou testar um modelo de regressão linear.

```

[37]: # 08 ===== Treinamento e Predição =====
# Treinar o pipeline inteiro com uma única linha!
# Será aplicado todos os passos de pré-processamento e treinar o modelo.
model_pipeline.fit(X_train, y_train)

```



```

[38]: # 08.1 - Fazer previsões com uma única linha!
# Ele irá aplicar as transformações nos dados de teste e fazer a predição.
log_predictions = model_pipeline.predict(X_test)

```

```

[39]: # 08.2 - Reverter a transformação log para obter os valores reais
real_predictions = np.expm1(log_predictions)

```

```

[40]: # 08.3 - Revertendo o y_test também para comparar
y_test_real = np.expm1(y_test)

```

```

[41]: # 09 ===== Avaliação =====
rmse = np.sqrt(mean_squared_error(y_test_real, real_predictions))
print(f"RMSE do modelo no conjunto de teste é: {rmse:.2f}")

```

O RMSE do modelo no conjunto de teste é: \$7814.06

```

[42]: # 09.1 - Calculando o R² Score
# A ordem dos argumentos é a mesma: (valores verdadeiros, valores previstos)
r2 = r2_score(y_test_real, real_predictions)
print(f"R² Score do modelo é: {r2:.4f} (ou {r2:.2%})") # Imprime como decimal e porcentagem

```

O R² Score do modelo é: 0.6067 (ou 60.67%)

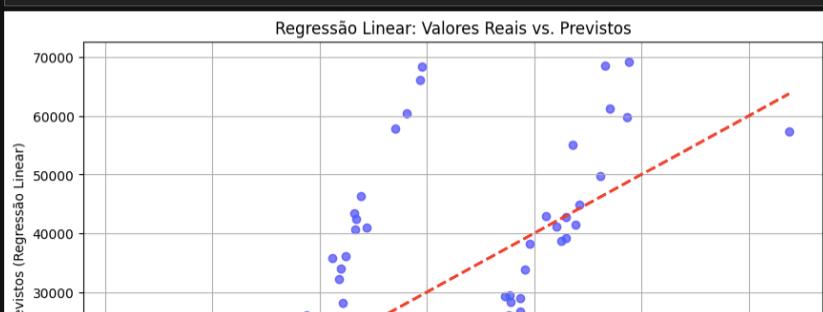
Analizando o Resultado

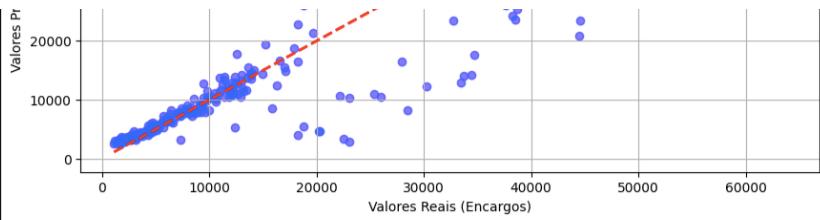
Apesar do RMSE mostrar um erro em dólar, ele é relativo aos valores totais, então, o R² mostrou um acerto de apenas 60% de forma arredondada. Como imaginado anteriormente, a baixa relação deixou esse modelo ineficaz. Logo, um novo modelo deve ser testado como próximo passo nesta análise. Será utilizado o RandomForestRegressor por ser mais robusto aos outliers e não depender tanto da correlação. Com tudo, ainda será feita uma análise com a lib Statmodels para identificar mais detalhes do modelo de regressão linear treinado.

```

[43]: # 10 ===== Gráfico: Previsões vs. Valores Reais (Regressão Linear) ====
plt.figure(figsize=(10, 6))
plt.scatter(y_test_real, real_predictions, alpha=0.5, color='blue')
plt.plot([y_test_real.min(), y_test_real.max()], [y_test_real.min(), y_test_real.max()], '--r', lw=2) # Linha diagonal de referência
plt.xlabel("Valores Reais (Encargos)")
plt.ylabel("Valores Previstos (Regressão Linear)")
plt.title("Regressão Linear: Valores Reais vs. Previstos")
plt.grid(True)
plt.show()

```





```
[44]: # 11 ===[ Validação Estatística com Statsmodels (Regressão Linear) ]===
import statsmodels.api as sm

# 11.1 - Pré-processar os dados de TREINO usando o 'preprocessador' do pipeline
# O 'preprocessador' já foi ajustado (fit) aos dados de treino
# quando treinou o model_pipeline.
# ColumnTransformer se chama 'preprocessador'.
X_train_processed = preprocessor.transform(X_train)

# 11.2 - Obter os nomes das features após o pré-processamento
# Isso ajudará a interpretar o sumário do statsmodels.
try:
    # Acessando o OneHotEncoder dentro do ColumnTransformer e da Pipeline do transformador categórico
    ohe_categories = preprocessor.named_transformers_['cat'].named_steps['onehot'].get_feature_names_out(categorical_features)
    processed_feature_names = numeric_features + list(ohe_categories)
except AttributeError:
    # Fallback se a estrutura de nomes for diferente (ex: se não houver pipeline no transformador)
    print("Aviso: Não foi possível obter os nomes das features processadas automaticamente. O sumário usará nomes genéricos se não forem fornecidos como DataFrame.")
    processed_feature_names = None

# 11.3 - Converter para DataFrame para usar os nomes das colunas com statsmodels, melhorando o sumário
if processed_feature_names:
    X_train_processed_df = pd.DataFrame(X_train_processed, columns=processed_feature_names, index=X_train.index)
else:
    X_train_processed_df = pd.DataFrame(X_train_processed, index=X_train.index)

# 11.4 - Adicionar uma constante (intercepto) ao modelo
X_train_sm = sm.add_constant(X_train_processed_df)

# 11.5 - Criar e treinar o modelo OLS (Ordinary Least Squares)
model_sm = sm.OLS(y_train, X_train_sm).fit()

# 11.6 - Exibir o sumário estatístico do modelo
print("\n--- Sumário Estatístico do Modelo de Regressão Linear (statsmodels) ---")
print(model_sm.summary())
```

```
--- Sumário Estatístico do Modelo de Regressão Linear (statsmodels) ---
OLS Regression Results
=====
Dep. Variable: charges R-squared:      0.757
Model:          OLS   Adj. R-squared:  0.755
Method:         Least Squares F-statistic:   413.7
Date:           Sat, 31 May 2025 Prob (F-statistic): 7.52e-320
Time:          20:41:31 Log-Likelihood:     -661.17
No. Observations: 1070 AIC:            1340.
Df Residuals:   1061 BIC:            1385.
Df Model:       8
Covariance Type: nonrobust
=====
            coef  std err      t  P>|t|  [0.025]  [0.975]
const      8.9123  0.032  279.105  0.000    8.850    8.975
age        0.4816  0.014  34.555  0.000    0.454    0.509
bmi        0.0821  0.014   5.697  0.000    0.054    0.110
children   0.1123  0.014   8.128  0.000    0.085    0.139
sex_male   -0.0745  0.028  -2.687  0.007   -0.129   -0.026
smoker_yes  1.5515  0.034  45.238  0.000    1.484    1.619
region_northwest -0.0564  0.039  -1.429  0.153   -0.134    0.021
region_southeast -0.1360  0.040  -3.428  0.001   -0.214   -0.058
region_southwest -0.1225  0.039  -3.114  0.002   -0.200   -0.045
=====
Omnibus:            353.753 Durbin-Watson:      2.034
Prob(Omnibus):      0.000 Jarque-Bera (JB): 1163.234
Skew:                1.619 Prob(JB):    2.55e-253
Kurtosis:             6.951 Cond. No.        5.59
=====
```

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.

Statmodels

Em [coef] (coeficiente), podemos observar que a idade [age] e ser fumante [smoker_yes] são as colunas que mais impactam no modelo, as que tem influência no alvo [charges] (custos hospitalares).

Na questão da significância estatística de cada coeficiente, segundo a regra geral de que $P>|t| < 0.05$, a variável é considerada estatisticamente significativa, temos que apenas a coluna [region northwest] não é significativa.

No Intervalo de Confiança, representado por [0.025 0.975], region_northwest contém zero no seu intervalo: -0.134 0.021; reforçando a conclusão de que essa coluna não é estatisticamente significativa neste modelo.

Apesar do modelo estar estatisticamente robusto, estava havendo muitos erros que puderam ser vistos no gráfico das variáveis reais vs as preditas.

Usando RandomForestRegressor

```
[45]: # 12 ===[ Importações de libs Necessárias ]=====
from sklearn.ensemble import RandomForestRegressor

[46]: # 13 ===[ Redefinição do Pipeline Principal ]=====
# O primeiro passo é o pré-processador (que lida com todas as colunas)
# O segundo passo é o modelo de Regressão RandomForestRegressor.
model_pipeline = Pipeline(steps=[
    ('preprocessor', preprocessor),
```

```
[47]: ('regressor', RandomForestRegressor(n_estimators=100, random_state=42))
])

[47]: # 14 ===[ Treinamento e Predição ]=====
# Treinar o pipeline inteiro com uma única linha!
# Aplicar todos os passos de pré-processamento e treinar o modelo.
model_pipeline.fit(X_train, y_train)

[47]: Pipeline
      preprocessor: ColumnTransformer
          num: StandardScaler
          cat: OneHotEncoder
          remainder: passthrough
      RandomForestRegressor
```

```
[48]: # 14.1 - Fazer previsões com uma única linha!
# Ele irá aplicar as transformações nos dados de teste e fazer a predição.
log_predictions = model_pipeline.predict(X_test)

[49]: # 14.2 - Revertendo a transformação log. para obter os valores reais
real_predictions = np.expm1(log_predictions)

[50]: # 14.3 - Revertendo o y test também para comparar
y_test_real = np.expm1(y_test)

[51]: # 15 ===[ Avaliação ]
rmse = np.sqrt(mean_squared_error(y_test_real, real_predictions))
print(f'O RMSE do modelo no conjunto de teste é: {rmse:.2f}')

O RMSE do modelo no conjunto de teste é: $4357.47
```

Houve uma diminuição aqui no erro em dólar.

```
[52]: # 15.1 - Calculando o R² Score
# A ordem dos argumentos é a mesma: (valores verdadeiros, valores previstos)
r2 = r2_score(y_test_real, real_predictions)
print(f'O R² Score do modelo é: {r2:.4f} (ou {r2:.2%})') # Imprime como decimal e porcentagem

O R² Score do modelo é: 0.8777 (ou 87.77%)
```

Agora um R2 com 87.77% de acerto, o RandomForestRegressor fica sendo definido assim como melhor que a regressão linear.

```
[53]: # 16 ===[ Verificando a Importância das Features ]=====
# Acessando os nomes das features após o pré-processamento
feature_names = numeric_features + list(preprocessor.named_transformers_['cat'].get_feature_names_out(categorical_features))
# Acessando as importâncias do modelo treinado
importances = model_pipeline.named_steps['regressor'].feature_importances_

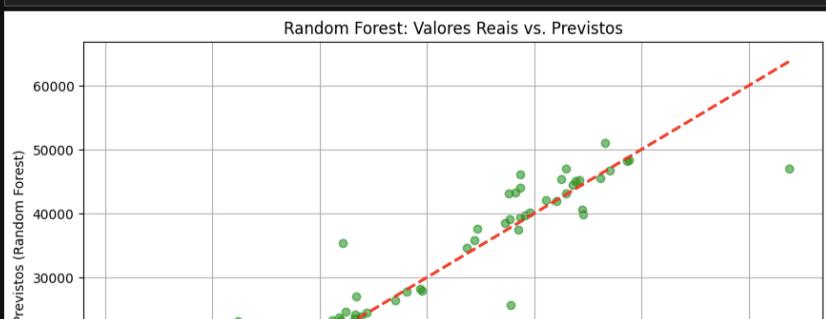
# Criando um DataFrame para melhor visualização
feature_importance_df = pd.DataFrame({'Feature': feature_names, 'Importance': importances})
feature_importance_df = feature_importance_df.sort_values(by='Importance', ascending=False)

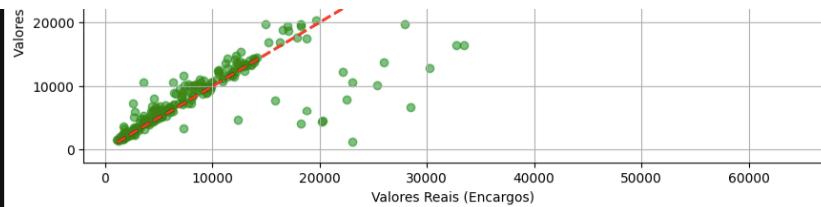
print("\nImportância das Features segundo o modelo:")
print(feature_importance_df)

Importância das Features segundo o modelo:
    Feature  Importance
4   smoker_yes  0.436858
0        age  0.380272
1       bmi  0.165749
2   children  0.041763
3   sex_male  0.011619
6  region_southeast  0.009092
5  region_northwest  0.007687
7  region_southwest  0.006960
```

O fato de ser fumante, somada à idade, são os fatores que mais influenciam nos custos hospitalares.

```
[54]: # 17 ===[ Gráfico: Previsões vs. Valores Reais (Random Forest) ]==
plt.figure(figsize=(10, 6))
plt.scatter(y_test_real, real_predictions, alpha=0.5, color='green')
plt.plot(y_test_real.min(), y_test_real.max(), [y_test_real.min(), y_test_real.max()], '--r', lw=2) # Linha diagonal de referência
plt.xlabel("Valores Reais (Encargos)")
plt.ylabel("Valores Previstos (Random Forest)")
plt.title("Random Forest: Valores Reais vs. Previstos")
plt.grid(True)
plt.show()
```





```
## ===[ Comparação dos Modelos ]=====
```

| Modelo | RMSE | R ² (Score) |
|-------------------------|-----------|------------------------|
| Regressão Linear | \$7814.06 | 60.67% |
| Random Forest Regressor | \$4357.47 | 87.77% |

O modelo com melhor RMSE foi o Random Forest Regressor por apresentar o menor erro.
O modelo com melhor R² foi o Random Forest Regressor por apresentar uma representação maior da variância da variável alvo (custos hospitalares - encargos <charges>).
Dessa forma o modelo que teve o melhor desempenho foi o segundo: Random Forest Regressor.

Conclusão Geral

O objetivo do projeto

A princípio era criar um modelo preditivo e comprovar sua eficácia com métricas estatísticas, mas, faltava um alvo e uma correlação que se mostraram na fase de análise dos dados, onde o alvo escolhido foram os custos hospitalares.

Os principais passos realizados

1. Conhecer os Dados
2. Padronização dos dados
3. Verificar a correlação
 - a. Mapa de Correlação (heatmap)
4. Um pipeline principal
5. Estatística com Statsmodels (Regressão Linear)
6. Comparativo do RMSE e R2 Score
7. Gráficos das Previsões vs Valores Reais

O modelo de melhor desempenho

Definitivamente o Random Forest Regressor apresentou melhores resultados tanto nos gráficos das previsões vs valores reais, quanto no comparativo das estatísticas métricas RMSE e R2 Score.
Se houvesse uma correlação melhor talvez o resultado fosse diferente.

Os principais insights

As features mais importantes no modelo são:

```
--> smoker_yes    0.436858
-->      age     0.380272
-->      bmi     0.105749
```

Limitações do Estudo

- Base de dados pequena.
- Uso de somente dois modelos de regressão.
- Não alteração dos valores padrões dos hiperparâmetros.

Sugestão para o Futuro

- Implementação de uma API.
- Construção de uma interface gráfica.

```
[ ]:
```

