

Introdução a Machine Learning

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br

Aula 6.1 – Valor de Shapley para Regressão.

Recapitulando o projeto criado.

Refazendo o passo a passo do Machine Learning para regressão.

Introduzindo o valor de Shapley na regressão.

Recapitulando a Jornada do Machine Learning

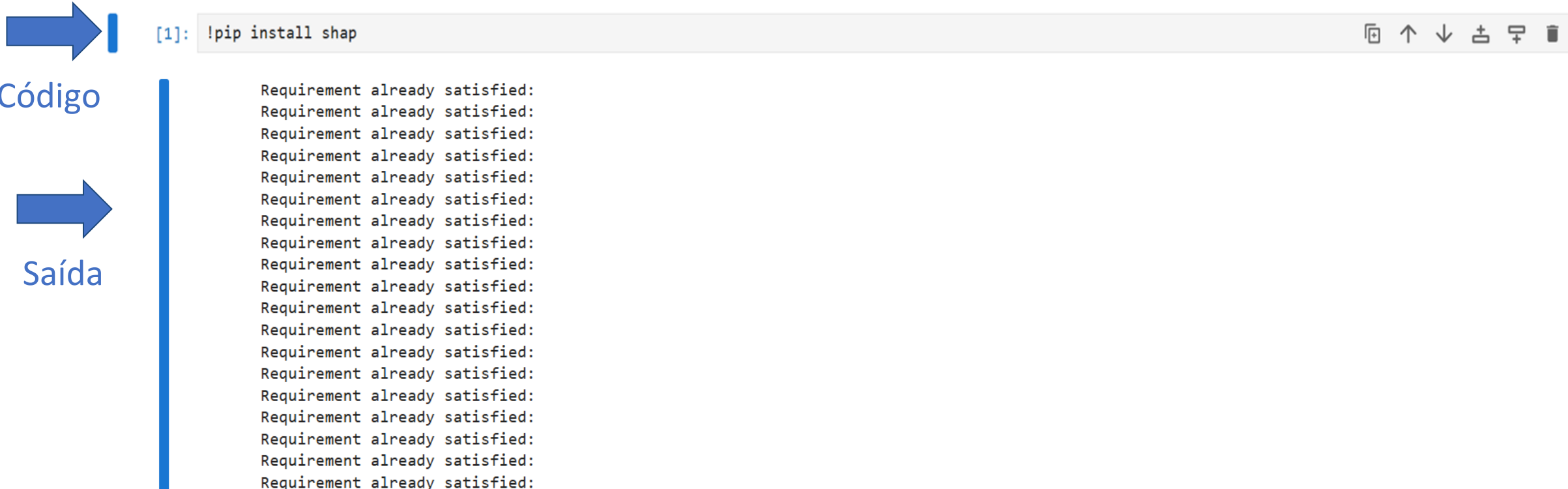
Recapitulando

- Na aula 06, vimos que conseguimos fazer com que os modelos de classificação explicassem como chegaram a conclusão na previsão.
- Calculamos o valor de Shapley e construímos um gráfico para nos mostrar as variáveis que mais impactam no modelo e de que forma é esse impacto.
- Mas como ficaria o gráfico de Shapley no contexto da regressão? Veremos isso nessa aula.

Vamos praticar?

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos organizar as bibliotecas que serão utilizadas nessa análise. Mas primeiro, vamos baixar a biblioteca para manipular os **Valores de Shapley**.



The image shows a terminal window with a light gray header bar. On the left, there are two blue arrows pointing right, one above the word 'Código' and one above the word 'Saída'. The terminal content shows a command prompt '[1]: !pip install shap' followed by 15 lines of output, each reading 'Requirement already satisfied:'. The terminal window has standard icons for copy, paste, and other functions in the top right corner.

```
[1]: !pip install shap

Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
Requirement already satisfied:
```

- Se esse código rodou e não obteve algum erro, a biblioteca já está instalada.

Resumindo: Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.



Código

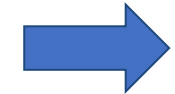
```
[2]: import pandas as pd # manipulação de tabelas
from sklearn.model_selection import train_test_split, GridSearchCV # divisão treino/teste + grid search
from sklearn.preprocessing import OneHotEncoder, StandardScaler # codificação categórica e padronização numérica
from sklearn.impute import SimpleImputer # imputação dos dados
from sklearn.compose import ColumnTransformer # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline # encadeia pré-processamento + modelo
from sklearn.tree import DecisionTreeRegressor # modelo baseado em várias árvores
from sklearn.metrics import mean_squared_error, r2_score # métricas de regressão
import numpy as np # biblioteca numérica
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # métricas de avaliação do modelo
from sklearn.neighbors import KNeighborsRegressor # modelo KNN
import shap # criar os gráficos de shapley value
import matplotlib.pyplot as plt
# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # Lê o arquivo aula_01_exemplo_01.csv em um DataFrame

df['tem_filhos'] = (df['children'] > 0).astype(int) # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
# (df['children'] > 0) cria uma Series booleana (True/False)
# .astype(int) converte True para 1 e False para 0
```

- Adicionamos a importação da biblioteca para o cálculo do Shapley.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Vamos criar um novo script para realizar todo o passo a passo de modelagem, mas primeiro vamos carregar nossos dados.



Código

```
[2]: import pandas as pd # manipulação de tabelas
from sklearn.model_selection import train_test_split, GridSearchCV # divisão treino/teste + grid search
from sklearn.preprocessing import OneHotEncoder, StandardScaler # codificação categórica e padronização numérica
from sklearn.impute import SimpleImputer # imputação dos dados
from sklearn.compose import ColumnTransformer # aplicar transformações diferentes em colunas
from sklearn.pipeline import Pipeline # encadeia pré-processamento + modelo
from sklearn.tree import DecisionTreeRegressor # modelo baseado em várias árvores
from sklearn.metrics import mean_squared_error, r2_score # métricas de regressão
import numpy as np # biblioteca numérica
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score # métricas de avaliação do modelo
from sklearn.neighbors import KNeighborsRegressor # modelo KNN
import shap # criar os gráficos de shapley value
import matplotlib.pyplot as plt
# Carregando dataset
df = pd.read_csv("aula_01_exemplo_01.csv") # Lê o arquivo aula_01_exemplo_01.csv em um DataFrame

df['tem_filhos'] = (df['children'] > 0).astype(int) # Cria uma nova coluna 'tem_filhos' que indica se a pessoa tem filhos (1) ou não (0)
# (df['children'] > 0) cria uma Series booleana (True/False)
# .astype(int) converte True para 1 e False para 0
```

O código acima não retorna nada pra gente!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Agora vamos deixar claro quais são as variáveis categóricas e quais são as numéricas.


Código

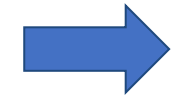
```
[5]: # Definição explícita das colunas por tipo
variaveis_categoricas = ["sex", "smoker", "region"] # listas com nomes das colunas categóricas (devem bater exatamente com os nomes do DataFrame)
variaveis_numericas = ["age", "bmi", "children", "tem_filhos"] # lista das colunas numéricas que serão escalonadas
```

- Perceba que não incluímos a variável target, já que fazemos manipulações de padronizar e transformar apenas nas covariáveis.

O código acima não retorna nada pra gente!

Recapitulando: Dividindo os Dados da aula_01_exemplo_01

- Novamente vamos separar nossos dados de forma que fique claro o **que é nossa variável target e nossas covariáveis**.



Código

```
[2]: # Separando as covariáveis (X) e target (y)
X = df.drop("charges", # drop("charges", axis=1) remove a coluna 'charges' (axis=1 indica coluna; axis=0 seria linhas);
          axis=1) # retorna um novo DataFrame sem 'charges'
y = df["charges"] # seleciona a coluna 'charges' como Series – esta é a variável alvo que queremos prever

X.head()
```



Saída

```
[2]:
```

	age	sex	bmi	children	smoker	region	tem_filhos
0	19	female	27.900	0	yes	southwest	0
1	18	male	33.770	1	no	southeast	1
2	28	male	33.000	3	no	southeast	1
3	33	male	22.705	0	no	northwest	0
4	32	male	28.880	0	no	northwest	0

- Perceba que agora temos a variável **X** que só possui as nossas covariáveis e a variável **y** que possui apenas a variável target.

Recapitulando: Dividindo os Dados da aula_01_exemplo_01


- Vamos dividir nossos dados em dados de treinamento e dados de teste. Usaremos o comando `train_test_split()`.


Código

```
[4]: # Divisão treino/teste

x_treino, x_teste, y_treino, y_teste = train_test_split( # função para divisão dos dados
    X,                                                    # covariáveis
    y,                                                    # variável target
    test_size=0.2,                                       # 20% para teste
    random_state=42                                     # Reprodutibilidade, como essa função aleatoriza os dados, vamos fixar essa aleatorização
)

print("número de linhas e colunas da base de treino:", x_treino.shape)
print("número de linhas e colunas da base de teste:", x_teste.shape)
```


Saída

```
número de linhas e colunas da base de treino: (1070, 7)
número de linhas e colunas da base de teste: (268, 7)
```

- Sempre definindo quais são nossas covariáveis e variável target.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos recriar cada processo de imputação e transformação de dados.



Código

```
[5]: # Pré-processamento

escalador = StandardScaler() # Escalonador, ele vai padronizar as variáveis numéricas

categorizador = OneHotEncoder(drop="first", # O categorizador vai Remover a primeira dummy para evitar redundância
                               handle_unknown="ignore") # Ignora categorias desconhecidas em dados de teste

imputador_numerico = SimpleImputer(strategy="median") # imputador numérico usando mediana

imputador_categorico = SimpleImputer(strategy="most_frequent") # imputador categórico usando a moda
```

O código acima não retorna nada pra gente!

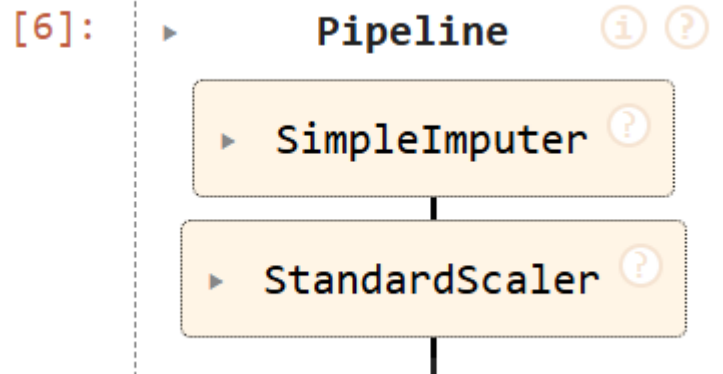
Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

→
Código

```
[6]: # Etapas de transformação das variáveis numéricas
etapas_numericas = Pipeline(
    [
        ("imputer", imputador_numerico),    # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela mediana)
        ("scaler", escalonador)             # "scaler" é o nome da etapa → aplica padronização
    ]
)
etapas_numericas
```

→
Saída



- Criando a etapa numérica.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

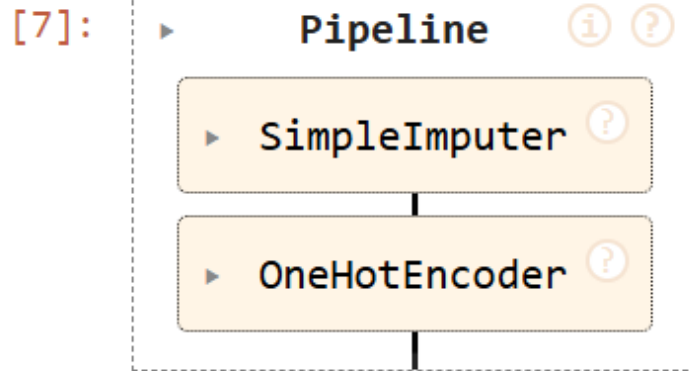
- Vamos agora recriar a criação das etapas numéricas e categóricas no python.

→
Código

```
[7]: # Etapas de transformação das variáveis categóricas
etapas_categoricas = Pipeline(
    [
        ("imputer", imputador_categorico), # "imputer" é o nome da etapa → aplica a imputação (substitui valores ausentes pela moda)
        ("encoder", categorizador)        # "encoder" é o nome da etapa → aplica OneHotEncoder
    ]
)

etapas_categoricas
```

→
Saída



- Criando a etapa categórica.

Recapitulando: Preparando os Dados da aula_01_exemplo_01

- Agora vamos juntar tudo novamente.


Código

```
[8]: # Juntando todo o processamento das variáveis categóricas e numéricas
preprocessador = ColumnTransformer(
    [
        ("num", etapas_numericas, variaveis_numericas), # Nome da etapa, Escalonador e a lista de variáveis numéricas
        ("cat", etapas_categoricas, variaveis_categoricas) # Nome da etapa, categorizador e a lista de variáveis categóricas
    ]
)

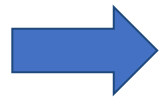
x_treino_transformado = preprocessador.fit_transform(x_treino) # aplicando a imputação e transformação nos dados de treino
x_teste_transformado = preprocessador.transform(x_teste) # aplicando a imputação e transformação nos dados de teste
```

- O **ColumnTransformer()** vai servir como um processo de união entre as etapas categóricas e contínuas.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar os novos comandos para inserir os modelos que aprendemos nessa aula. Os comandos anteriores são repetições da aula passada e já foram escritos previamente, basta rodá-los previamente.
- Vamos criar o modelos de árvore de decisão e o KNN. Começaremos pela árvore de decisão.



```
[9]: # criando o modelo  
modelo = DecisionTreeRegressor(random_state=42) # criando o modelo e fixando a aleatorização
```

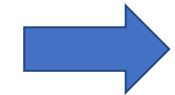


Código

- Criamos nosso modelo e armazenamos ele em uma variável chamada **modelo**.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.



Código

```
[10]: # Grade de hiperparâmetros

param_grid = {
    "max_depth": [2,3,5,7], # Profundidade máxima da árvore
    "min_samples_split": [2,5,10,15,20,25]# Nº mínimo de amostras para dividir um nó
}
```

- Note que são os mesmos hiperparâmetros que vimos: “**max_depth**” é a profundidade máxima da árvore e o “**min_samples_split**” é Número mínimo de amostras que um nó precisa ter para ser dividido.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.


Código

```
[11]: # Configurando Grid Search
      grid_search = GridSearchCV(
          estimator= modelo,          # modelo a ser otimizado
          param_grid=param_grid,      # Grade de hiperparâmetros
          cv=5,                       # 5-fold cross-validation
          scoring="neg_root_mean_squared_error" # Métrica de comparação: RMSE
      )
```

- Como o RMSE tem a característica de ser interpretável, vamos seguir utilizando ele como critério para selecionar os melhores hiperparâmetros.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search nos nossos dados de treino usando o comando `.fit()`.


Código

```
[12]: # Treinando com Grid Search
      grid_search.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

      melhor_modelo_arvore = grid_search.best_estimator_ # Melhor modelo encontrado
```



- Agora temos o melhor modelo para o caso da árvore de decisão.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Vamos calcular o Feature Importance da árvore de decisão.


Código

```
[14]: nomes_das_features = preprocessor.get_feature_names_out() # pegando o nome das colunas preprocessadas

# O atributo 'feature_importances_' armazena uma lista com a importância de cada feature.
importancias = melhor_modelo_arvore.feature_importances_

# Crie um DataFrame para organizar e classificar os resultados
df_importancias = pd.DataFrame({
    'Variaveis': nomes_das_features, # coluna com as variáveis
    'Importancia': importancias # coluna com as importancias
})

# 4. Classifique as features em ordem decrescente de importância
df_importancias = df_importancias.sort_values(by='Importancia', ascending=False)
df_importancias
```


Saída

```
[14]:
```

	Variaveis	Importancia
5	cat_smoker_yes	0.712802
1	num_bmi	0.176411
0	num_age	0.110787
2	num_children	0.000000
3	num_tem_filhos	0.000000
4	cat_sex_male	0.000000
6	cat_region_northwest	0.000000
7	cat_region_southeast	0.000000
8	cat_region_southwest	0.000000

- Note que utilizando o comando `.feature_importances_`, conseguimos extrair as importâncias e cria um DataFrame para nos auxiliar na visualização.

Modelando os Dados da aula_01_exemplo_01

- Vamos calcular o Feature Importance da árvore de decisão.


Código

```
[14]: nomes_das_features = preprocessor.get_feature_names_out() # pegando o nome das colunas preprocessadas

# O atributo 'feature_importances_' armazena uma lista com a importância de cada feature.
importancias = melhor_modelo_arvore.feature_importances_

# Crie um DataFrame para organizar e classificar os resultados
df_importancias = pd.DataFrame({
    'Variaveis': nomes_das_features, # coluna com as variáveis
    'Importancia': importancias # coluna com as importancias
})

# 4. Classifique as features em ordem decrescente de importância
df_importancias = df_importancias.sort_values(by='Importancia', ascending=False)
df_importancias
```


Saída

```
[14]:
```

	Variaveis	Importancia
5	cat_smoker_yes	0.712802
1	num_bmi	0.176411
0	num_age	0.110787
2	num_children	0.000000
3	num_tem_filhos	0.000000
4	cat_sex_male	0.000000
6	cat_region_northwest	0.000000
7	cat_region_southeast	0.000000
8	cat_region_southwest	0.000000

- Perceba que se a pessoa é fumante é o fator mais influente para a predição dos dados, seguido pelo IMC.

Modelando os Dados da aula_01_exemplo_01

- Vamos calcular o Feature Importance da árvore de decisão.

→
Código

```
[14]: nomes_das_features = preprocessor.get_feature_names_out() # pegando o nome das colunas preprocessadas

# O atributo 'feature_importances_' armazena uma lista com a importância de cada feature.
importancias = melhor_modelo_arvore.feature_importances_

# Crie um DataFrame para organizar e classificar os resultados
df_importancias = pd.DataFrame({
    'Variaveis': nomes_das_features, # coluna com as variáveis
    'Importancia': importancias # coluna com as importancias
})

# 4. Classifique as features em ordem decrescente de importância
df_importancias = df_importancias.sort_values(by='Importancia', ascending=False)
df_importancias
```

→
Saída


```
[14]:
```

	Variaveis	Importancia
5	cat_smoker_yes	0.712802
1	num_bmi	0.176411
0	num_age	0.110787
2	num_children	0.000000
3	num_tem_filhos	0.000000
4	cat_sex_male	0.000000
6	cat_region_northwest	0.000000
7	cat_region_southeast	0.000000
8	cat_region_southwest	0.000000

- Perceba que depois da variável de idade (age), as demais variáveis não foram importantes para o modelo.
- **O que acontece se retirarmos essas variáveis e criarmos um novo modelo sem ela?**

Modelando os Dados da aula_01_exemplo_01

- Vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.


Código

```
[13]: # Predição no treino
      y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

      # Métricas de avaliação
      mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
      rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
      mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
      r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
      mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- O comando **mean_squared_error()** calcula o MSE. Já o comando **np.sqrt()** calcula a raiz do MSE o que consequentemente faz com que tenhamos o RMSE. O MAE é calculado pelo **mean_absolute_error()** e o r^2 pelo **r2_score()**.

Modelando os Dados da aula_01_exemplo_01

- Vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[13]: # Predição no treino
y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

# Métricas de avaliação
mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- como o MAPE não possui uma função própria, temos que criar do zero! primeiro começando de dentro, calculamos o **valor absoluto dos valores reais com os preditos divididos pelos valores reais com o comando `np.abs()`**.

Modelando os Dados da aula_01_exemplo_01

- Vamos analisar o desempenho do modelo nos dados de treinamento com as novas métricas.



Código

```
[13]: # Predição no treino
      y_pred_treino_arvore = melhor_modelo_arvore.predict(x_treino_transformado) # covariáveis de treino

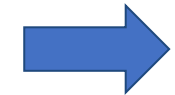
      # Métricas de avaliação
      mse_treino_arvore = mean_squared_error(y_treino, y_pred_treino_arvore) # Mean Squared Error
      rmse_treino_arvore = np.sqrt(mse_treino_arvore) # Root Mean Squared Error
      mae_treino_arvore = mean_absolute_error(y_treino, y_pred_treino_arvore) # Mean Absolute Error
      r2_treino_arvore = r2_score(y_treino, y_pred_treino_arvore) # R²: proporção da variância explicada pelo modelo
      mape_treino_arvore = np.mean(np.abs((y_treino - y_pred_treino_arvore) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- Agora pegamos tudo isso e utilizamos o comando **np.mean()** para calcular a média e **por fim multiplicamos tudo por 100**.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos mostrar as métricas calculadas.



Código

```
[14]: # Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("R² no treino:", np.round(r2_treino_arvore, 3))
print("MSE no treino:", np.round(mse_treino_arvore, 3))
print("RMSE no treino:", np.round(rmse_treino_arvore, 3))
print("MAE no treino:", np.round(mae_treino_arvore, 3))
print("MAPE no treino (%)ate:", np.round(mape_treino_arvore, 2))
```



Saída

```
Melhores parâmetros encontrados: {'max_depth': 3, 'min_samples_split': 2}
R² no treino: 0.854
MSE no treino: 21120357.016
RMSE no treino: 4595.689
MAE no treino: 2785.253
MAPE no treino (%): 34.69
```

- O modelo de árvore de decisão explica cerca de 85% da variabilidade dos custos médicos nos EUA ($R^2 = 0,854$). Em média, a previsão difere do valor real em 2.785 dólares (MAE), com RMSE = 4.595 dólares e MAPE = 34,7%, indicando que o modelo captura bem as tendências gerais.

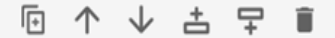
Modelando os Dados da aula_01_exemplo_01

- Agora vamos implementar o modelo KNN.



Código

```
[15]: # criando o modelo  
modelo_knn = KNeighborsRegressor() # criando o modelo KNN; aqui não há aleatoriedade fixa como na árvore
```



- Note que o KNN não tem o **random_state**.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos criar nossa grade de hiperparâmetros.


Código

```
[16]:  
# Grade de hiperparâmetros  
param_grid_knn = {  
    "n_neighbors": [3, 5, 7, 9, 11], # Número de vizinhos a considerar na média para prever  
    "p": [1, 2] # Tipo de distância: 1 = Manhattan, 2 = Euclidiana  
}
```

- Perceba que são os mesmos hiperparâmetros que discutimos antes!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos usar o Grid Search para encontrar os melhores hiperparâmetros.


Código

[17]:

```
# Configurando Grid Search
grid_search_knn = GridSearchCV(
    estimator=modelo_knn,          # modelo KNN a ser otimizado
    param_grid=param_grid_knn,    # Grade de hiperparâmetros
    cv=5,                          # 5-fold cross-validation: o dataset será dividido em 5 partes
    scoring="neg_root_mean_squared_error" # Métrica de comparação: RMSE (quanto menor, melhor)
)

# Treinando com Grid Search
grid_search_knn.fit(x_treino_transformado, y_treino) # usando as covariáveis já preprocessadas e nossa variável target

melhor_modelo_knn = grid_search_knn.best_estimator_ # Melhor modelo KNN encontrado
```

- Vamos continuar utilizando o RMSE.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Da mesma forma que a árvore de decisão, vamos calcular as métricas.


Código

```
[18]: # Predição no treino
y_pred_treino_knn = melhor_modelo_knn.predict(x_treino_transformado) # covariáveis de treino

# Avaliação do modelo no treino
mse_treino_knn = mean_squared_error(y_treino, y_pred_treino_knn) # Valores reais, Valores preditos

# Métricas de avaliação
mse_treino_knn = mean_squared_error(y_treino, y_pred_treino_knn) # Mean Squared Error
rmse_treino_knn = np.sqrt(mse_treino_knn) # Root Mean Squared Error
mae_treino_knn = mean_absolute_error(y_treino, y_pred_treino_knn) # Mean Absolute Error
r2_treino_knn = r2_score(y_treino, y_pred_treino_knn) # R²: proporção da variância explicada pelo modelo
mape_treino_knn = np.mean(np.abs((y_treino - y_pred_treino_knn) / y_treino)) * 100 # Mean Absolute Percentage Error (%)
```

- Note que todos os passos se repetem, apenas mudando o modelo!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Agora vamos mostrar as métricas calculadas.

→
Código

```
[19]: # Resultados
print("Melhores parâmetros encontrados:", grid_search.best_params_)
print("R² no treino:", np.round(r2_treino_knn, 3))
print("MSE no treino:", np.round(mse_treino_knn, 3))
print("RMSE no treino:", np.round(rmse_treino_knn, 3))
print("MAE no treino:", np.round(mae_treino_knn, 3))
print("MAPE no treino (%):", np.round(mape_treino_knn, 2))
```

→
Saída

```
Melhores parâmetros encontrados: {'n_neighbors': 3, 'p': 1}
R² no treino: 0.861
MSE no treino: 20116578.002
RMSE no treino: 4485.151
MAE no treino: 2464.023
MAPE no treino (%): 25.87
```

- O modelo KNN explica cerca de 86% da variabilidade dos custos médicos nos EUA ($R^2 = 0,861$). Em média, a previsão difere do valor real em 2.464 dólares (MAE), com RMSE = 4.485 dólares e MAPE = 25,9%, indicando que o modelo captura bem as tendências gerais, embora possa divergir em casos extremos.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos ver como cada modelo se comporta no teste. Começaremos com a árvore de decisão.


Código

```
[20]: # Predição no teste
y_pred_teste_arvore = melhor_modelo_arvore.predict(x_teste_transformado) # covariáveis de teste

# Métricas de avaliação
mse_teste_arvore = mean_squared_error(y_teste, y_pred_teste_arvore) # Mean Squared Error
rmse_teste_arvore = np.sqrt(mse_teste_arvore) # Root Mean Squared Error
mae_teste_arvore = mean_absolute_error(y_teste, y_pred_teste_arvore) # Mean Absolute Error
r2_teste_arvore = r2_score(y_teste, y_pred_teste_arvore) # R²: proporção da variância explicada pelo modelo
mape_teste_arvore = np.mean(np.abs((y_teste - y_pred_teste_arvore) / y_teste)) * 100 # Mean Absolute Percentage Error (%)
```

- Da mesma forma que os anteriores, o que muda é a base de dados!

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Vamos observar o desempenho.

Código

```
[21]: # Resultados
print("Avaliação do modelo de Árvore de Decisão no teste:")
print("R²:", np.round(r2_teste_arvore, 3))
print("MSE:", np.round(mse_teste_arvore, 3))
print("RMSE:", np.round(rmse_teste_arvore, 3))
print("MAE:", np.round(mae_teste_arvore, 3))
print("MAPE (%):", np.round(mape_teste_arvore, 2))
```

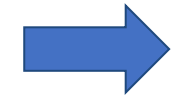
Saída

```
Avaliação do modelo de Árvore de Decisão no teste:
R²: 0.853
MSE: 22812669.852
RMSE: 4776.261
MAE: 2865.638
MAPE (%): 37.68
```

- o modelo de árvore de decisão apresentou $R^2 = 0,853$, indicando que cerca de 85% da variabilidade dos custos médicos nos EUA foi explicada pelo modelo. As previsões apresentam MAE de 2.866 dólares, RMSE de 4.776 dólares e MAPE de 37,7%.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

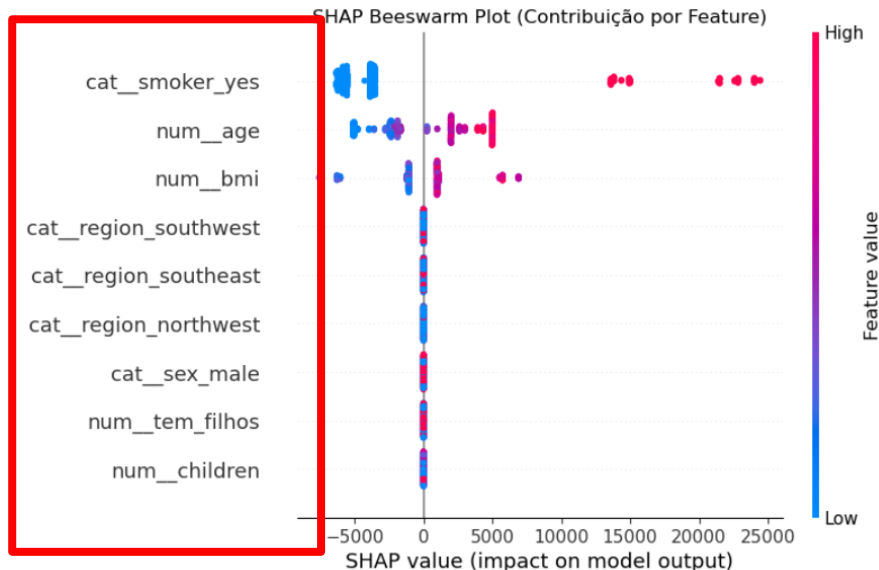
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



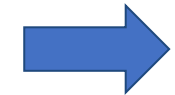
Saída



- De cima pra baixo, temos quais variáveis contribuem mais para a previsão.
- Note se é fumante e a idade foram as que mais impactam o modelo de árvore.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

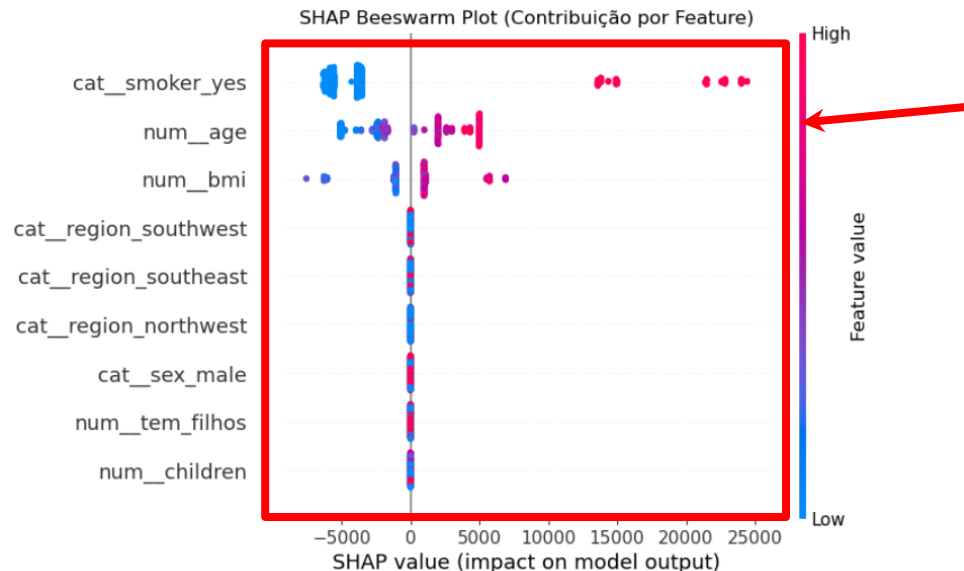
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



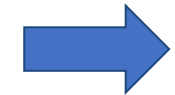
Saída



- Cada bolinha representa um indivíduo na base de dados de teste.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

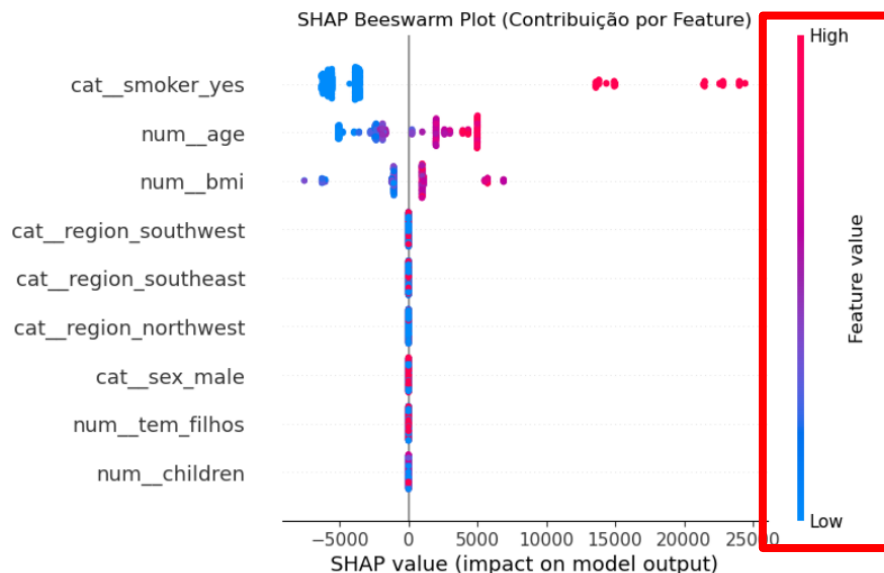
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



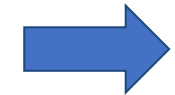
Saída



- A **cor mais avermelhada** indica valores altos, por exemplo idades mais avançadas de 40,60 até 90 terão essa cor **avermelhada**.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

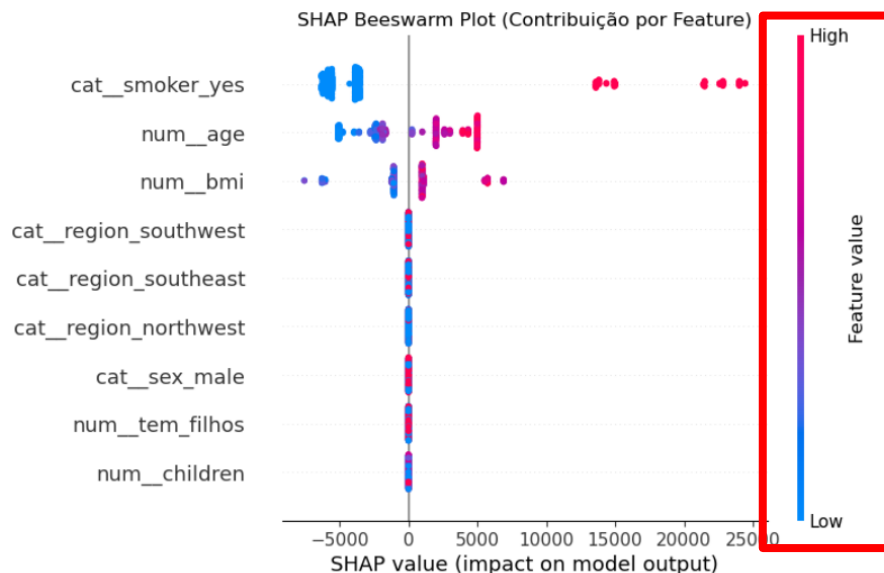
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



Saída

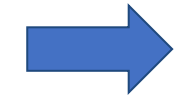


- **cores mais azuis** indicam valores mais baixos da variável, por exemplo ainda na variável de idade, valores como 7, 10, 15 20 ou 30 anos terão cores mais **azuladas**.



Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

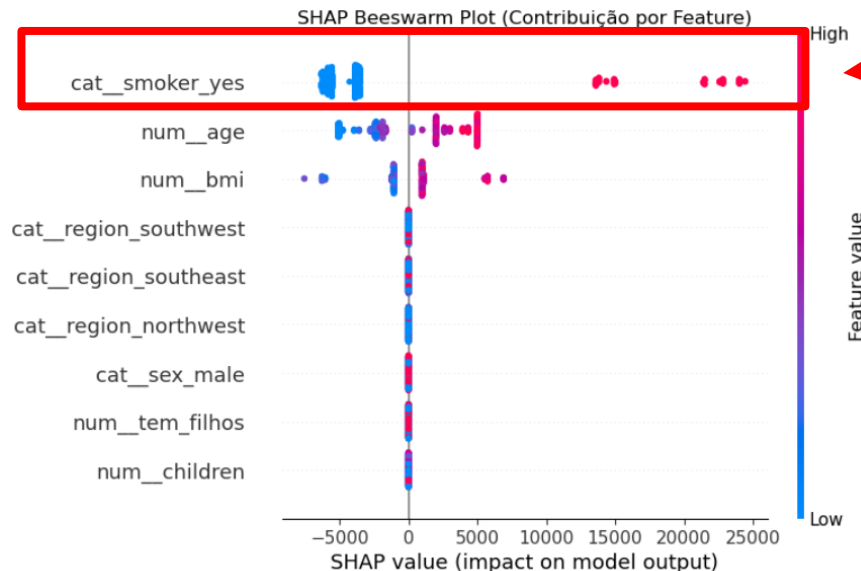
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



Saída



- Note que o valor de smoker_yes (se o individuo fuma) é 0 ou 1.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

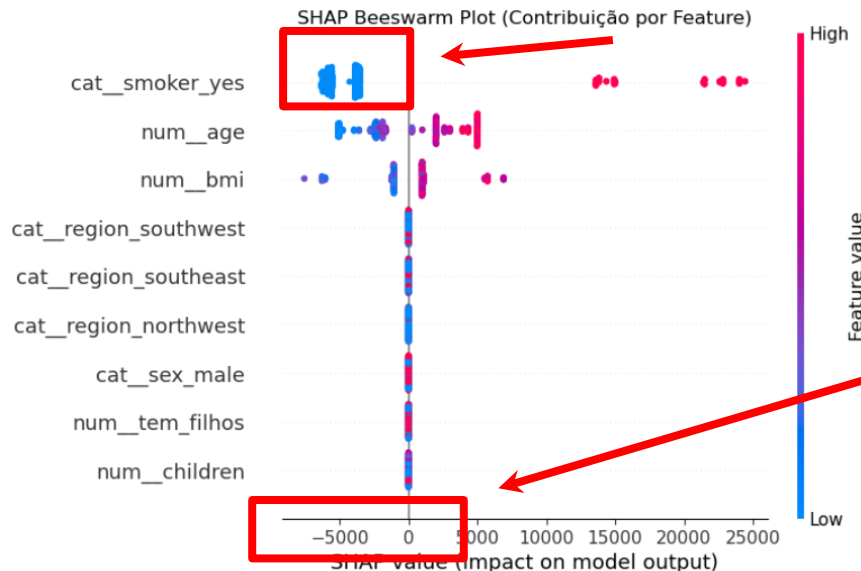
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



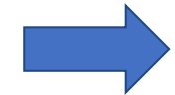
Saída



- o Valor 0 (Não Fumante) que é o menor possível **está em azul.**
- Note que o valor de shapley está no quadrante à esquerda, onde possui valores negativos.
- Isso significa que não ser fumante está associado a custos médicos mais baixos

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

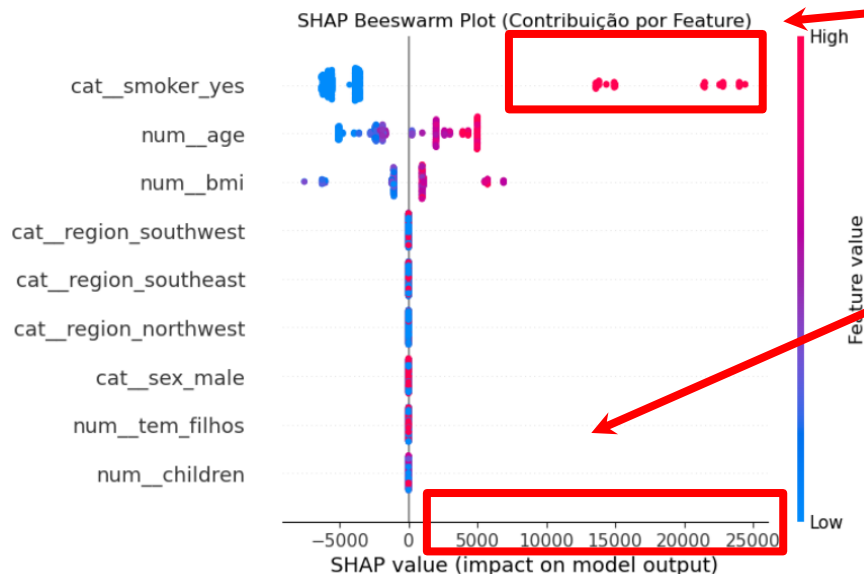
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



Saída



- O valor 1 (Fumante) que é o maior possível está **avermelhado**.
- Note que os valores de Shapley estão no quadrante positivo a direita.
- Isso significa que ser fumante (sim) está fortemente associado a custos médicos muito mais alto

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

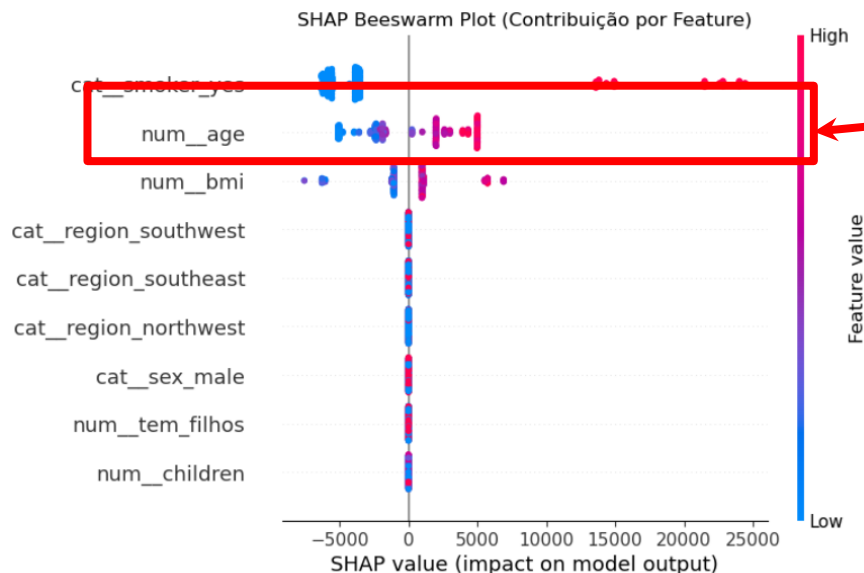
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



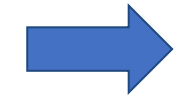
Saída



- Para a idade, temos que pessoas mais velhas tendem a ter custos médicos mais altos.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.



Código

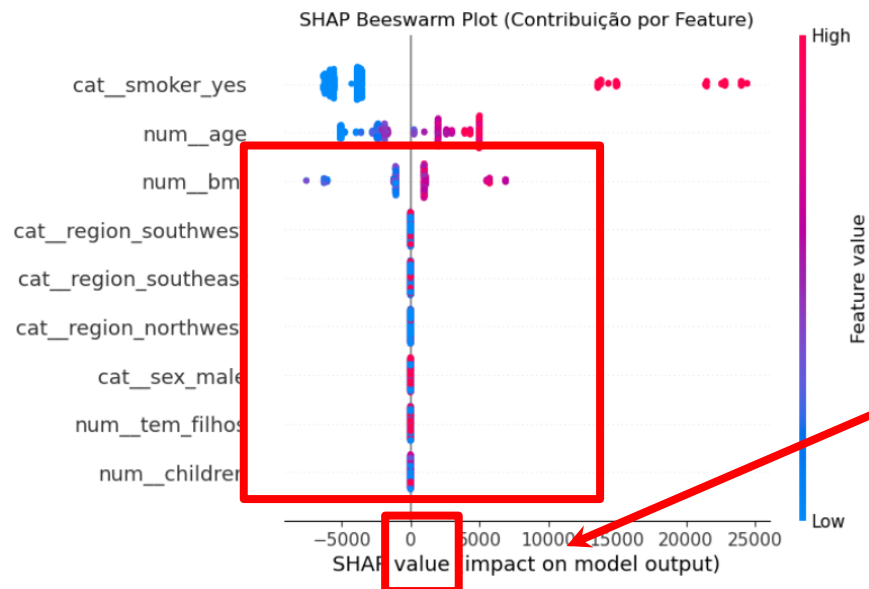
```
[24]: explainer_arvore = shap.TreeExplainer(melhor_modelo_arvore) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_arvore = explainer_arvore.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot( # cria o gráfico
    shap_values_arvore,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```



Saída



- Para as demais variáveis, os valores estão mais próximos de Zero no eixo X (valor de Shapley).
- Essas variáveis possuem contribuição nula para a previsão de custos , ou seja não são tão relevantes para a modelagem.

Modelando os Dados da aula_01_exemplo_01

- Agora vamos analisar pro KNN.



Código

```
[22]: # Predição no teste usando o melhor modelo KNN
      y_pred_teste_knn = melhor_modelo_knn.predict(x_teste_transformado) # covariáveis de teste

      # Métricas de avaliação
      mse_teste_knn = mean_squared_error(y_teste, y_pred_teste_knn)           # Mean Squared Error
      rmse_teste_knn = np.sqrt(mse_teste_knn)                               # Root Mean Squared Error
      mae_teste_knn = mean_absolute_error(y_teste, y_pred_teste_knn)         # Mean Absolute Error
      r2_teste_knn = r2_score(y_teste, y_pred_teste_knn)                   # R²: proporção da variância explicada pelo modelo
      mape_teste_knn = np.mean(np.abs((y_teste - y_pred_teste_knn) / y_teste)) * 100 # Mean Absolute Percentage Error (%)
```

- Agora conseguiremos comparar o desempenho dos dois modelos.

O código acima não retorna nada pra gente!

Modelando os Dados da aula_01_exemplo_01

- Vamos observar o desempenho.


Código

```
[23]: # Resultados
print("Avaliação do modelo KNN no teste:")
print("R²:", np.round(r2_teste_knn, 3))
print("MSE:", np.round(mse_teste_knn, 3))
print("RMSE:", np.round(rmse_teste_knn, 3))
print("MAE:", np.round(mae_teste_knn, 3))
print("MAPE (%):", np.round(mape_teste_knn, 2))
```


Saída

```
Avaliação do modelo KNN no teste:
R²: 0.708
MSE: 45388801.136
RMSE: 6737.121
MAE: 3996.595
MAPE (%): 42.63
```

- O modelo KNN apresentou $R^2 = 0,708$, indicando que explica cerca de 71% da variabilidade dos custos médicos nos EUA. As previsões têm MAE de 3.997 dólares, RMSE de 6.737 dólares e MAPE de 42,6%,

Explicando as previsões para o KNN

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.

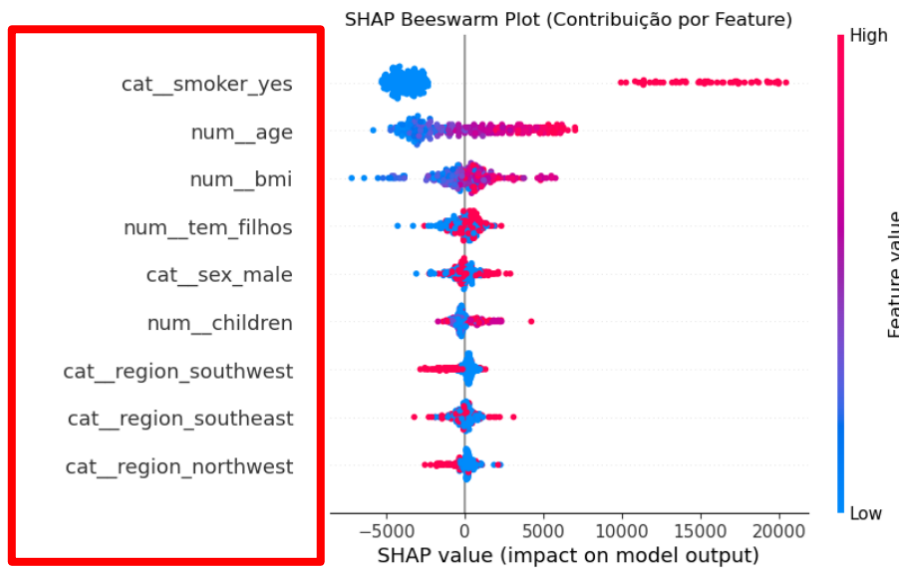
➔
Código

```
[27]: explainer_knn = shap.KernelExplainer(melhor_modelo_knn.predict, x_teste_transformado) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_knn = explainer_knn.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot(
    shap_values_knn,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```

➔
Saída



- De cima pra baixo, temos quais variáveis contribuem mais para a previsão.
- ser fumante causa o maior aumento nos custos médicos.

Explicando as previsões para a Árvore de Decisão

- Agora podemos criar um gráfico para entender melhor como cada variável se comportou na previsão.

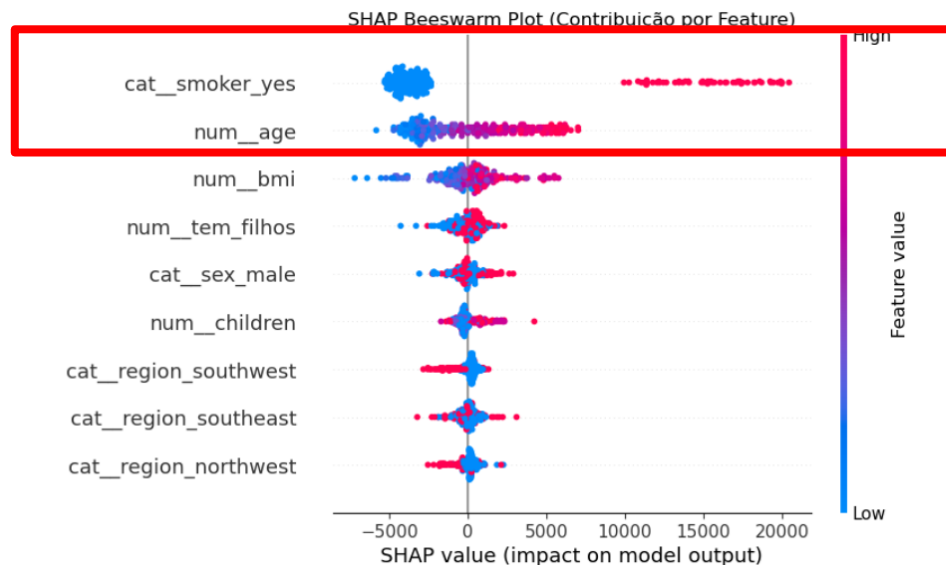
→
Código

```
[27]: explainer_knn = shap.KernelExplainer(melhor_modelo_knn.predict, x_teste_transformado) # comando necessário para calcular o valor de shapley

# Calcule os valores SHAP (ATENÇÃO: Este passo é demorado)
shap_values_knn = explainer_knn.shap_values(x_teste_transformado) # armazenando o valor de shapley em uma variável

shap.summary_plot(
    shap_values_knn,
    x_teste_transformado,
    feature_names=nomes_das_features,
    plot_type="dot", # O tipo "dot" é o padrão beeswarm
    show=False
)
plt.title("SHAP Beeswarm Plot (Contribuição por Feature)")
plt.show()
```

→
Saída



- ser fumante causa o maior aumento nos custos médicos.
- Idades mais avançadas também causa o maior aumento nos custos médicos.

Considerações finais

Concluindo

- Agora temos noção de que forma as variáveis impactam na predição dos custos médicos na forma contínua.
- Vimos que as variáveis de idade e se é fumante ou não são sempre as preditoras chave que o modelo utiliza para a previsão.
- Mesmo do ponto de vista da classificação, quanto da regressão, essas variáveis ainda impactam na previsão do modelo.

Disclaimer: propriedade intelectual

Este material foi criado pela professora Roberta Moreira Wichmann e é de sua propriedade intelectual.

É destinado exclusivamente ao uso dos alunos para fins educacionais no contexto das aulas.

Qualquer reprodução, distribuição ou utilização deste material, no todo ou em parte, sem a expressa autorização prévia da autora, é estritamente proibida.

O não cumprimento destas condições poderá resultar em medidas legais.

Referências Bibliográficas

- ABU-MOSTAFA, Yaser S.; MAGDON-ISMAIL, Malik; LIN, Hsuan-Tien. Learning from Data: A Short Course. Pasadena: California Institute of Technology (AMLBook), 2012.
- DEMÉTRIO, Clarice Garcia Borges; ZOCCHI, Sílvia Sandoval. Modelos de regressão. Piracicaba: Departamento de Ciências Exatas, ESALQ/USP, 2011. Disponível em: https://www.researchgate.net/publication/266233241_Modelos_de_Regressao. Acesso em: 23 set. 202
- GERON, Aurélien. Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems. 2. ed. Sebastopol, CA: O'Reilly Media, 2019.
- HARRIS, C. R. et al. Array programming with NumPy. Nature, v. 585, p. 357–362, 2020. DOI: 10.1038/s41586-020-2649-2.
- HASTIE, Trevor; TIBSHIRANI, Robert; FRIEDMAN, Jerome. The elements of statistical learning: data mining, inference, and prediction. 2. ed. New York: Springer, 2009.
- KAPOOR, Sayash; NARAYANAN, Arvind. Leakage and the reproducibility crisis in machine-learning-based science. Patterns, v. 4, n. 9, 2023.

Referências Bibliográficas

- IZBICKI, Rafael; DOS SANTOS, Tiago Mendonça. Aprendizado de máquina: uma abordagem estatística. Rafael Izbicki, 2020.
- MORETTIN, Pedro Alberto; SINGER, Júlio da Motta. Estatística e ciência de dados. 2. ed. Rio de Janeiro: LTC, 2022.
- PEDREGOSA, F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, v. 12, p. 2825–2830, 2011.
- PONCE-BOBADILLA, Ana Victoria et al. Practical guide to SHAP analysis: Explaining supervised machine learning model predictions in drug development. Clinical and Translational Science, v. 17, n. 11, p. e70056, nov. 2024. DOI: 10.1111/cts.70056.
- PYTHON SOFTWARE FOUNDATION. Python Language Reference. Disponível em: <https://docs.python.org/3/reference/index.html>. Acesso em: 10 set. 2025.
- THE PANDAS DEVELOPMENT TEAM. pandas-dev/pandas: Pandas. Zenodo, 2024. Disponível em: <https://doi.org/10.5281/zenodo.10537285>. Acesso em: 10 set. 2025.

Referências Bibliográficas

- VON LUXBURG, Ulrike; SCHÖLKOPF, Bernhard. Statistical Learning Theory: Models, Concepts, and Results. In: GABBAY, D. M.; HARTMANN, S.; WOODS, J. H. (eds.). Handbook of the History of Logic, vol. 10: Inductive Logic. Amsterdam: Elsevier North Holland, 2011. p. 651–706. DOI: 10.1016/B978-0-444-52936-7.50016-1.

Introdução à Interpretabilidade em modelos de Machine Learning

Obrigada!

Profa. Dra. Roberta Wichmann

roberta.wichmann@idp.edu.br



INSTITUTO BRASILEIRO DE ENSINO,
DESENVOLVIMENTO E PESQUISA