


*German Association of Actuaries (DAV) — Working Group "Explainable Artificial Intelligence"*

# Model-Agnostic Explainability Methods for Regression Problems: A Case Study on Medical Costs Data

Dr. Simon Hatzesberger (simon.hatzesberger@gmail.com)

---

*In this Jupyter notebook, we offer a comprehensive walkthrough for actuaries and data scientists on applying model-agnostic explainability methods to regression tasks, using a medical costs dataset as our case study. With the growing prevalence of modern black box machine learning models, which often lack the interpretability of classical statistical models, these explainability methods become increasingly important to ensure transparency and trust in predictive modeling.*

*We illuminate both global methods – such as global surrogate models, PDPs, ALE plots, and permutation feature importances – for a thorough understanding of model behavior, and local methods – like SHAP, LIME, and ICE plots – for detailed insights into individual predictions.*

*In addition to concise overviews of these methods, the notebook provides practical code examples that readers can easily adopt, offering a user-friendly introduction to explainable artificial intelligence.*

---

## Table of Contents

## 1. Introduction

The primary objective of this notebook is to showcase a variety of techniques for interpreting the internal workings and predictive behavior of machine learning models in regression problems. Our goal is to deliver insights that span two critical levels of model understanding: the general workings of a machine learning model, which we refer to as global explainability, and the specific factors driving individual predictions, known as local explainability. Implementing these forms of explainable artificial intelligence (XAI) is crucial, as it empowers stakeholders – including model developers, regulatory bodies, and executive management – to make well-informed decisions rather than relying on black box models. Readers are encouraged to apply – or at least find inspiration in – the approaches and code snippets that will be shared in this work.

Throughout this case study, we analyze a slightly modified version of the medical costs dataset found in [1]. Our regression problem is to predict individual medical costs billed by health insurance based on personal factors such as age, sex, body mass index (BMI), the number of children covered by the health insurance, smoking status, and regional location within the US. We aim to shed light on the model's decision-making process with model-agnostic explainability methods, rather than solely maximizing the predictive accuracy. Therefore, we apply these methods to a CatBoost model for practical illustration. Note that we focus on model-agnostic methods rather than model-specific methods that are tailored to particular models. This choice ensures the techniques we discuss can be broadly applied across different machine learning models, offering greater flexibility and wider applicability.

The remainder of this notebook is as follows. Section 2 contains a brief exploratory data analysis (EDA) of the medical cost dataset, acquainting us with the data and its predictive features. Section 3 covers the training and evaluation process of the CatBoost machine learning model as applied to our regression problem. Section 4 introduces global model-agnostic explainability techniques, specifically examining global surrogate models (Subsection 4.1), Partial Dependence Plots (PDP, Subsection 4.2), Accumulated Local Effects (ALE, Subsection 4.3), and Permutation Feature Importance (PFI, Subsection 4.4). Section 5 contrasts this global perspective with local explainability methods such as Shapley Additive Explanations (SHAP, Subsection 5.1), Local Interpretable Model-Agnostic Explanations (LIME, Subsection 5.2), and Individual Conditional Expectations (ICE, Subsection 5.3). Section 6 concludes the notebook by discussing the limitations of our case study and presenting perspectives on future work. Finally, the Appendix provides a technical guide on applying XAI methods to machine learning models that do not inherently handle categorical features (Appendix A.1). It also includes deep dives into selected topics: a comparison between PDPs and ALE plots (Appendix A.2), diverse feature importance methods (Appendix A.3), and variants of SHAP (Appendix A.4).

## 2. Brief Exploratory Data Analysis

Before diving into a brief exploratory data analysis (EDA), we import all libraries that are used throughout this notebook, define global constants, and set a random seed to ensure consistency and reproducibility.

In [ ]:
```
# Libraries for data processing
import numpy as np
import pandas as pd

# Libraries for model development, evaluation, and preprocessing
from catboost import CatBoostRegressor
from sklearn.neural_network import MLPRegressor
from sklearn.tree import DecisionTreeRegressor, plot_tree
from sklearn.metrics import root_mean_squared_error, r2_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import OneHotEncoder, LabelEncoder, StandardScaler
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
import statsmodels.api as sm

# Libraries for model explainability
from sklearn.inspection import PartialDependenceDisplay, permutation_importance
import shap
shap.initjs()
from PyALE import ale
from lime.lime_tabular import LimeTabularExplainer

# Libraries for visualization
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns

# Configurations for pandas display options and matplotlib inline backend
pd.set_option('display.max_rows', 50)
pd.set_option('display.max_columns', None)

# Color palette for visualizations
COLOR_LIGHT, COLOR_DARK, COLOR_GREY = '#4BB9E6', '#0050A0', '#7f7f7f'
```

```
# Global constants and settings
TRAIN_RATIO = 0.70   # Ratio for splitting data into training and test sets
RANDOM_SEED = 12345  # Seed for reproducibility of random operations

# Configuration for warnings
import warnings
warnings.filterwarnings('ignore', category=DeprecationWarning)
```

For initial insights into the medical costs dataset, we conduct a basic EDA. We begin by loading the dataset, inspecting the first ten entries, and reviewing the overall structure, including the number of rows, columns, and any missing values, to provide a concise overview of the data.

In [ ]:
```
# Load the medical costs dataset
df_raw = pd.read_csv('MedicalCosts.csv')

# Output the dimensions of the dataset and the total number of missing values
num_entries = df_raw.shape[0]
num_features = df_raw.shape[1]
total_missing = df_raw.isnull().sum().sum()

print(f"The dataset contains {num_entries} entries and {num_features} features.")
print(f"There are {total_missing} missing values in the dataset.")

# Display the first ten entries to preview the data
df_raw.head(10)
```

```
The dataset contains 1338 entries and 7 features.
There are 0 missing values in the dataset.
```

Out[ ]:

| | AGE | SEX | BMI | NUMBERCHILDREN | SMOKER | REGION | MEDICALCHARGES |
|---|---|---|---|---|---|---|---|
| **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 |
| **5** | 31 | female | 25.740 | 0 | no | southeast | 3756.62160 |
| **6** | 46 | female | 33.440 | 1 | no | southeast | 8240.58960 |
| **7** | 37 | female | 27.740 | 3 | no | northwest | 7281.50560 |
| **8** | 37 | male | 29.830 | 2 | no | northeast | 6406.41070 |
| **9** | 60 | female | 25.840 | 0 | no | northwest | 28923.13692 |

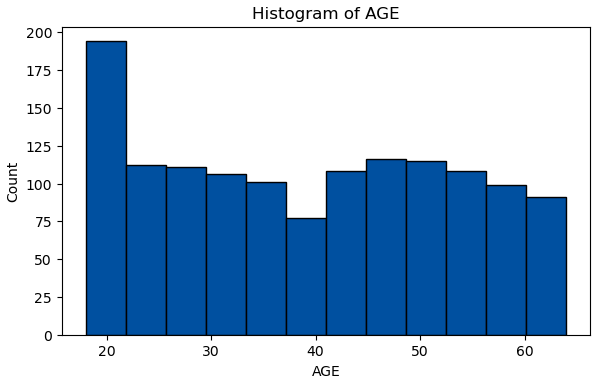
The dataset consists of several personal features, and our objective is to predict the medical charges of an individual based on these attributes. Below are concise descriptions of these features and the target variable for a clear understanding of the dataset's contents:

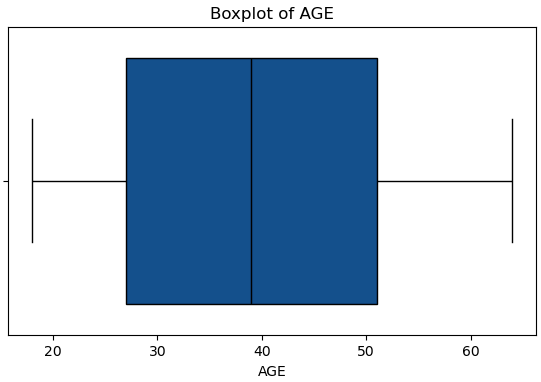
- `AGE` : The age of the policyholder.
- `SEX` : The policyholder's gender, with categories 'male' or 'female'.
- `BMI` : Body Mass Index, which uses weight and height to categorize the individual's body weight status.
- `NUMBERCHILDREN` : The count of children or dependents included under the policyholder's coverage.
- `SMOKER` : A binary indicator of whether the policyholder smokes, denoted by 'yes' or 'no'.
- `REGION` : The policyholder's residential region within the US, classified into 'northeast', 'southeast', 'southwest', or 'northwest' areas.
- `MEDICALCHARGES` : The target variable, representing the total amount billed for the policyholder's medical services, which is to be reimbursed by the insurer.

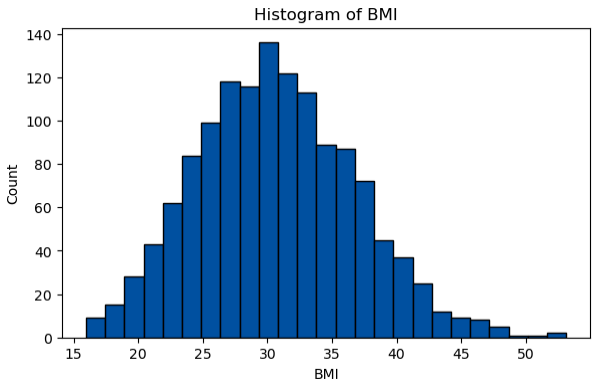
In the following, histograms and boxplots are used to illustrate the distribution of each numerical variable, namely `AGE` , `BMI` , `NUMBERCHILDREN` , and `MEDICALCHARGES` .

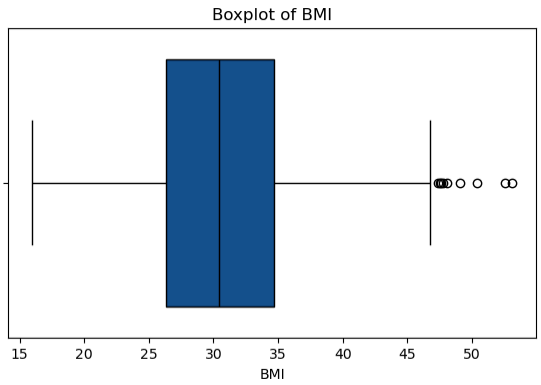
In [ ]:
```
# List of numerical columns in the dataset
numerical_features = ['AGE', 'BMI', 'NUMBERCHILDREN', 'MEDICALCHARGES']

# Plot histograms and boxplots for each numerical feature
for col in numerical_features:
    # Create a new figure with a specific size
    fig, ax = plt.subplots(1, 2, figsize=(15, 4))

    # Histogram: distribution of the feature
    sns.histplot(df_raw[col], ax=ax[0], color=COLOR_DARK,
```

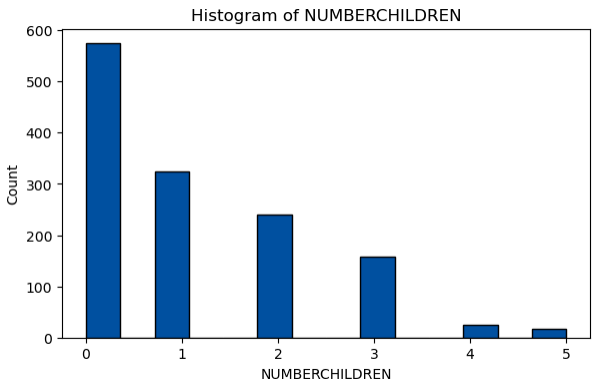
```
                edgecolor='black', alpha=1)
    ax[0].set_title(f'Histogram of {col}')
    ax[0].set_xlabel(col)
    ax[0].set_ylabel('Count')

    # Boxplot: spread of the feature
    sns.boxplot(x=df_raw[col], ax=ax[1], color=COLOR_DARK,
                linecolor='black')
    ax[1].set_title(f'Boxplot of {col}')
    ax[1].set_xlabel(col)

    # Display all plots
    plt.show()
```

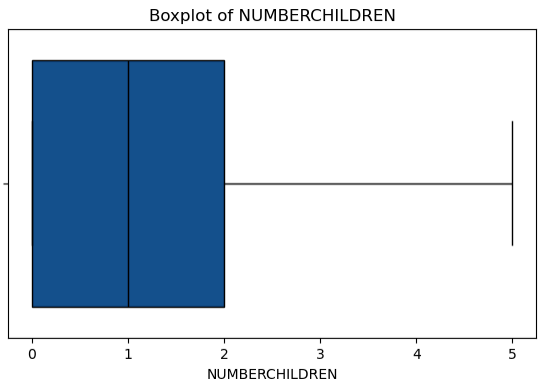


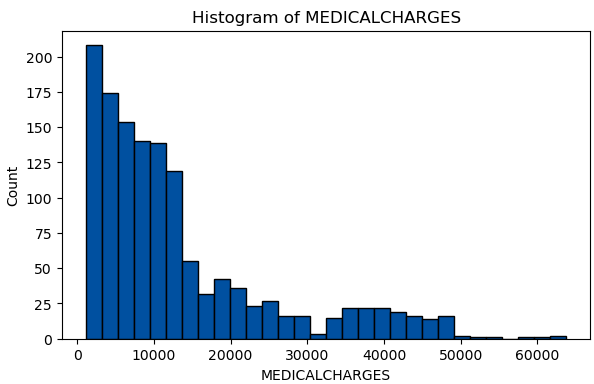



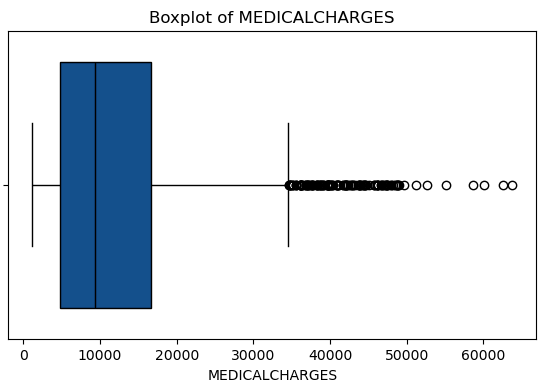



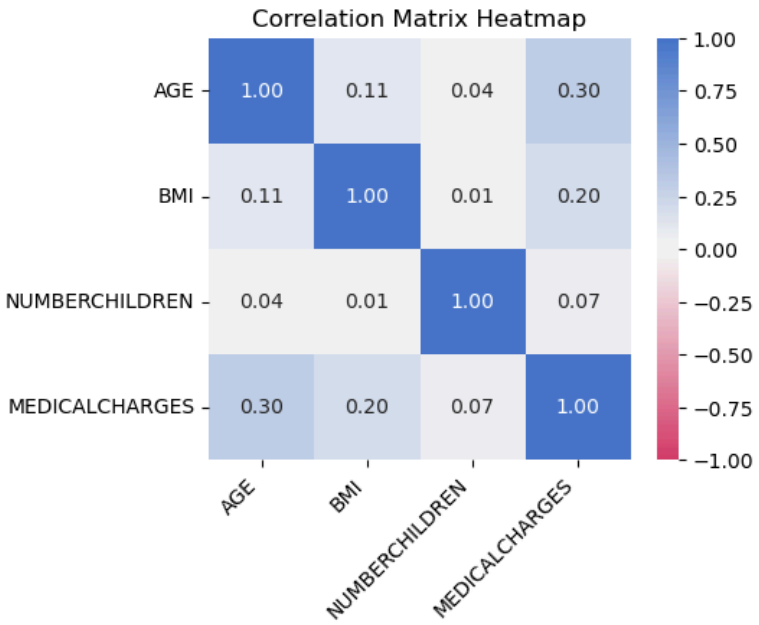




Having visualized the individual distributions and variations of the numerical features through histograms and boxplots, we will next explore the relationships between these variables. To achieve this, we build a correlation matrix to illustrate the degree to which the

features are linearly related to one another. This aims to uncover any strong correlations between features that could influence the performance of our forthcoming machine learning model.

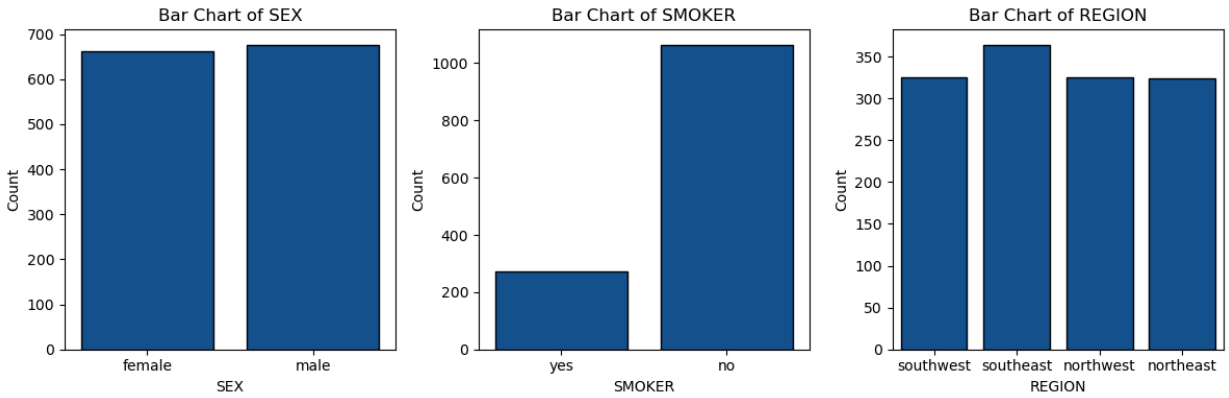
In [ ]:
```
# Compute the correlation matrix for the numerical features
corr_matrix = df_raw[numerical_features].corr()

# Set the size of the heatmap
plt.figure(figsize=(5.5, 4.5))

# Generate a heatmap with annotation
sns.heatmap(corr_matrix, annot=True, square=True, fmt=".2f", vmin=-1, vmax=1,
            center=0, cmap=sns.diverging_palette(0, 255, sep=10, n=256))

# Configure tick labels and set the title for the correlation matrix heatmap
plt.xticks(rotation=45, ha='right')
plt.yticks(rotation=0)
plt.title('Correlation Matrix Heatmap')

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




The correlation matrix shows that both `AGE` and `BMI` have a moderate positive correlation with the target variable `MEDICALCHARGES`, suggesting that as age and BMI increase, medical expenses tend to rise accordingly. In contrast, `NUMBERCHILDREN` appears to have a minimal influence on `MEDICALCHARGES`. Furthermore, the three non-target features `AGE`, `BMI`, and `NUMBERCHILDREN` exhibit almost no correlation with one another. These correlations provide preliminary insights for the upcoming regression analysis.

After examining the numerical features, we turn our attention to the categorical variables, `SEX`, `SMOKER`, and `REGION`, and visualize their distributions using bar charts.

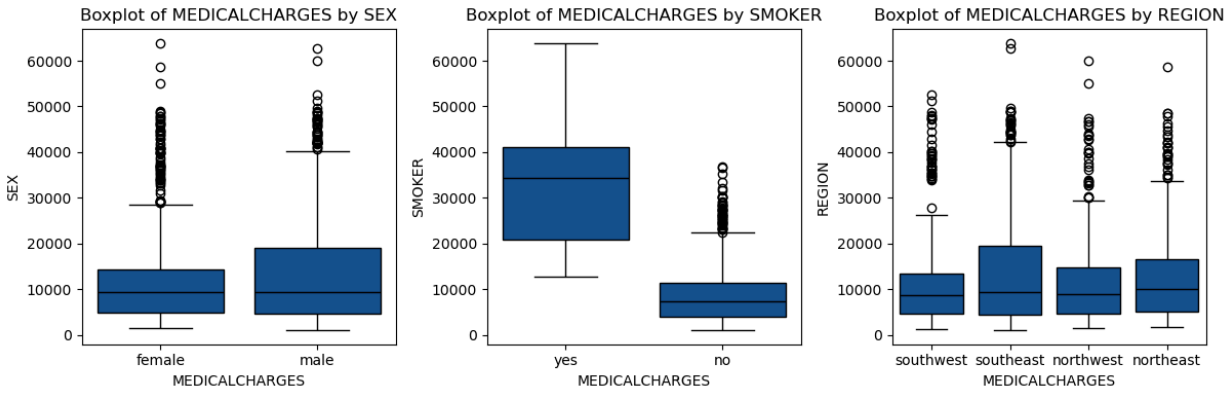
In [ ]:
```
# List of categorical columns in the dataset
categorical_features = ['SEX', 'SMOKER', 'REGION']

# Create a single figure and a grid of subplots
fig, axes = plt.subplots(nrows=1, ncols=len(categorical_features), figsize=(12, 4))

# Create bar charts for each categorical column
for i, col in enumerate(categorical_features):
    # Count plot on the appropriate subplot
    sns.countplot(x=col, data=df_raw, ax=axes[i], color=COLOR_DARK,
                  edgecolor='black')

    # Set title and labels
    axes[i].set_title(f'Bar Chart of {col}')
    axes[i].set_xlabel(col)
    axes[i].set_ylabel('Count')

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




The counts for the categorical variables show a fairly balanced distribution of gender ( `SEX` ), with nearly equal numbers of male and female policyholders. There is a notable imbalance in smoking status ( `SMOKER` ), with significantly more non-smokers than smokers. The regional distribution ( `REGION` ) is relatively uniform across the southeast, southwest, northwest, and northeast regions.

Expanding our exploration of categorical variables, we employ boxplots to examine their impact on `MEDICALCHARGES` , providing a clearer visual representation of how these factors may drive healthcare costs.

In [ ]:
```python
# Create a single figure and a grid of subplots
fig, axes = plt.subplots(nrows=1, ncols=len(categorical_features), figsize=(12, 4))

# Create boxplots for each categorical column
for i, col in enumerate(categorical_features):
    # Boxplot on the appropriate subplot
    sns.boxplot(x=col, y='MEDICALCHARGES', data=df_raw, ax=axes[i],
                color=COLOR_DARK, linecolor='black')

    # Set title and labels
    axes[i].set_title(f'Boxplot of MEDICALCHARGES by {col}')
    axes[i].set_xlabel('MEDICALCHARGES')
    axes[i].set_ylabel(col)

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




The boxplots clearly demonstrate that being a `SMOKER` is generally associated with substantially higher `MEDICALCHARGES` compared to non-smokers, suggesting that smoking status is a pivotal determinant in predicting medical costs. In contrast, the features `SEX` and `REGION` do not show a significant effect on `MEDICALCHARGES` , indicating that these characteristics may not be critical in forecasting healthcare expenditures in this dataset.

## 3. Developing and Evaluating a Black Box Machine Learning Model

Building on our analysis from Section 2, we now proceed to develop a machine learning model to forecast medical costs. We choose to implement a CatBoost model, which has been widely recognized for its ability to handle categorical data natively and deliver strong predictive performance with minimal data preprocessing. This is well-documented in [2], as well as evidenced by its frequent use in winning solutions in Kaggle competitions. However, since CatBoost is considered a black box model, it necessitates the use of XAI methods to decode its decisions, offering clarity into the mechanics that drive its predictions.

Before applying the CatBoost algorithm, we divide our dataset into two components: the set of predictor variables `X_raw` and the target variable `y` , consistent with conventional practices in supervised learning. Additionally, we further split the dataset into training and test sets to evaluate the model's performance on unseen data.

In [ ]:
```
# Define the target variable
target = 'MEDICALCHARGES'

# Update the numerical_features list to exclude the target variable
numerical_features.remove(target)

# Set data type for categorical features to category for efficient processing
for cat_feature in categorical_features:
    df_raw[cat_feature] = df_raw[cat_feature].astype('category')

# Separate the dataset into explanatory features and the target variable
X_raw = df_raw.drop(target, axis=1)
y = df_raw[target]
features_X_raw = X_raw.columns

# Display data types of features for verification
print("Data types of features in X_raw:")
print(X_raw.dtypes)

# Split the data into training and testing sets based on predefined TRAIN_RATIO
# and use a consistent RANDOM_SEED for reproducibility
X_raw_train, X_raw_test, y_train, y_test = train_test_split(
    X_raw, y, train_size=TRAIN_RATIO, random_state=RANDOM_SEED
)

# Output the shape of the training and testing sets to confirm the split
print(f"\nTraining set dimensions: {X_raw_train.shape}")
print(f"Test set dimensions:     {X_raw_test.shape}")
```

```
Data types of features in X_raw:
AGE                 int64
SEX              category
BMI               float64
NUMBERCHILDREN      int64
SMOKER           category
REGION           category
dtype: object

Training set dimensions: (936, 6)
Test set dimensions:     (402, 6)
```

With data preparations complete, we move on to train the CatBoost model using our training dataset and measure its performance on the test data with the root mean squared error (RMSE) as our primary evaluation metric and the $R^2$ score for further validation.

In [ ]:
```
# Define the CatBoost regressor with specified parameters and a random seed
model_CB_raw = CatBoostRegressor(
    iterations=100,
    learning_rate=0.05,
    random_seed=RANDOM_SEED,
    allow_writing_files=False,
    logging_level='Silent'
)

# Fit the CatBoost model
model_CB_raw.fit(X_raw_train, y_train, cat_features=categorical_features)

# Predict on the test set
predictions_CB_raw = model_CB_raw.predict(X_raw_test)

# Evaluate the model on the test set using RMSE and R² score
rmse_CB_raw = root_mean_squared_error(y_test, predictions_CB_raw)
r2_CB_raw = r2_score(y_test, predictions_CB_raw)

# Print the RMSE and R² score
print(f"RMSE for the CatBoost model on the test data:     {rmse_CB_raw:.2f}")
print(f"R² score for the CatBoost model on the test data:    {r2_CB_raw:.2f}")
```

```
RMSE for the CatBoost model on the test data:     4360.91
R² score for the CatBoost model on the test data:    0.87
```

The CatBoost model demonstrates strong predictive capabilities, evidenced by its low RMSE and high $R^2$ values, confirming its effectiveness in making forecasts. This machine learning model will serve as the foundation for the XAI techniques discussed in the upcoming sections.

As we progress through this notebook, it is important to note that certain XAI methods we will explore later require the categorical features to be one-hot encoded for technical reasons (see, e.g., Subsection 4.1 or Subsection 5.2). Therefore, we will now apply one-hot encoding to our categorical features. Subsequently, we will partition the encoded dataset into training and test sets, ensuring that the same rows are used for the split as before. This consistency will allow for a fair comparison between models trained on the raw and encoded data.

In [ ]:
```
# One-hot encode categorical features and display the first five rows
X_enc = pd.get_dummies(X_raw)
display(pd.concat([X_enc, y], axis=1).head(5))
features_X_enc = X_enc.columns

# Display data types of features for verification
print("Data types of features in X_enc:")
print(X_enc.dtypes)

# Split the data into training and testing sets based on predefined TRAIN_RATIO
# and use a consistent RANDOM_SEED for reproducibility
X_enc_train, X_enc_test, _, _ = train_test_split(
    X_enc, y, train_size=TRAIN_RATIO, random_state=RANDOM_SEED
)

# Output the shape of the training and testing sets to confirm the split
print(f"\nTraining set dimensions: {X_enc_train.shape}")
print(f"Test set dimensions:     {X_enc_test.shape}")
```

| | AGE | BMI | NUMBERCHILDREN | SEX_female | SEX_male | SMOKER_no | SMOKER_yes | REGION_northeast | REGION_northwest | REG |
|---|---|---|---|---|---|---|---|---|---|---|
| **0** | 19 | 27.900 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | |
| **1** | 18 | 33.770 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | |
| **2** | 28 | 33.000 | 3 | 0 | 1 | 1 | 0 | 0 | 0 | |
| **3** | 33 | 22.705 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | |
| **4** | 32 | 28.880 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | |

```
Data types of features in X_enc:
AGE                 int64
BMI               float64
NUMBERCHILDREN      int64
SEX_female          uint8
SEX_male            uint8
SMOKER_no           uint8
SMOKER_yes          uint8
REGION_northeast    uint8
REGION_northwest    uint8
REGION_southeast    uint8
REGION_southwest    uint8
dtype: object

Training set dimensions: (936, 11)
Test set dimensions:     (402, 11)
```

Equipped with our trained machine learning model, the next section will focus on global model-agnostic explainability methods to elucidate the model's general functionality. In the subsequent section, we will then address local model-agnostic techniques to analyze the decision-making process for individual predictions.

## 4. Global Model-Agnostic Explainability Methods

In this section, we explore *global* model-agnostic explainability methods, which provide insights into the overall behavior of the machine learning model developed in the previous section. These techniques are designed to offer a holistic understanding of the model's decisions across the entire data space, allowing to identify general trends and patterns that the model uses to make predictions. In our study, we concentrate on some of the most prominent global model-agnostic methods, namely global surrogate models, partial dependence plots, accumulated local effects, and permutation feature importances. Additional techniques are cited for interested readers in Section 6.

### 4.1 Global Surrogate Model

*Main Idea:*

The core concept of global surrogate models is straightforward: rather than attempting to interpret a complex black box model (e.g., a gradient boosting model) directly, one substitutes it with a more interpretable white box model (e.g., a decision tree) and extracts insights from this substitute. While the white box model may not match the predictive performance of the black box model, it provides the necessary interpretability to understand and mimic the model's behavior.

*Operational Details:*

To implement this approach, we undertake the following steps:

1. Train the original (black box) model on the data `X` and labels `y` (or subsets thereof).
2. Generate the model's predictions `y_hat_original` using the data `X`.
3. Select an interpretable surrogate model and train it on `X` and `y_hat_original`.
4. Evaluate how accurately the surrogate model approximates the original model, and if the approximation is satisfactory, interpret the surrogate to gain insights into the workings of the original model.

Steps 1 and 2 are typically straightforward processes. In Step 3, one has to ensure that the surrogate model is indeed easy to interpret (i.e., a white box model). Step 4 is critical: the evaluation measure selected must accurately reflect the fidelity of the surrogate model to the original. The $R^2$ score is commonly used for this purpose, calculated as

$$R^2 := 1 - \frac{\sum_{i=1}^{n} \left( \hat{y}_i^{\text{original}} - \hat{y}_i^{\text{surrogate}} \right)^2}{\sum_{i=1}^{n} \left( \hat{y}_i^{\text{original}} - \bar{y}^{\text{original}} \right)^2}$$

where

- $\hat{y}_i^{\text{original}}$ is the prediction from the original (black box) model for the $i$-th instance,
- $\hat{y}_i^{\text{surrogate}}$ is the prediction from the surrogate (interpretable) model for the $i$-th instance, and
- $\bar{y}^{\text{original}}$ is the mean over all the predictions from the original (black box) model.

Note that an $R^2$ score of 1 indicates perfect alignment, meaning the surrogate model's predictions are identical to those of the original. Conversely, an $R^2$ score near 0 indicates that the surrogate model provides little to no reliable insight into the original model's behavior.

*Application to the Medical Costs Dataset:*

To illustrate the XAI method of global surrogate models, we will attempt to replicate the CatBoost model described in Section 3 using both a decision tree and a linear regression model, which are recognized for their inherent interpretability and status as white box models.

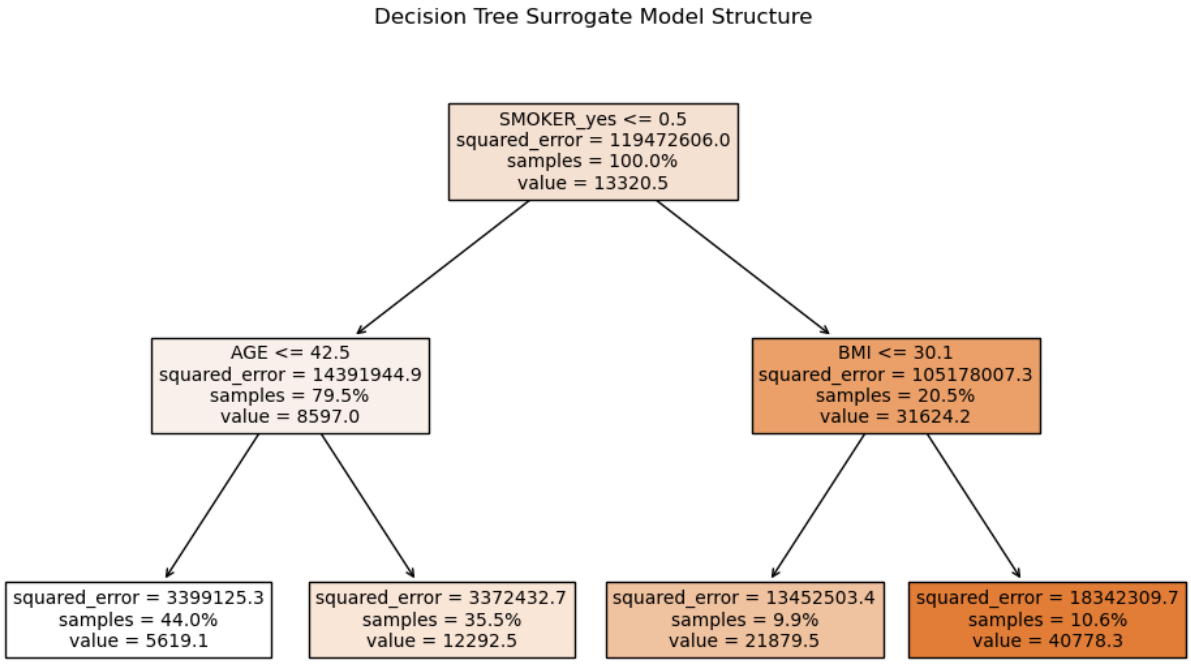
We will begin by employing a flat decision tree as the surrogate model.

In [ ]:
```
# Step 1: Training the CatBoost model has already been completed in Section 3
# via the following code
# model_CB_raw.fit(X_raw_train, y_train, cat_features=categorical_features)

# Step 2: Generate the predictions from the original (black box) model
pred_model_CB_train = model_CB_raw.predict(X_raw_train)
pred_model_CB_test  = model_CB_raw.predict(X_raw_test)

# Step 3: Train a decision tree (white box) model as the surrogate
# We must use the encoded feature set `X_enc_train` since DecisionTreeRegressor
# cannot handle raw categorical data
surrogate_model_DT = DecisionTreeRegressor(max_depth=2,
                                           random_state=RANDOM_SEED)
surrogate_model_DT.fit(X_enc_train, pred_model_CB_train)

# Step 4: Evaluate the surrogate model's performance by comparing its
# predictions to those of the black box model
pred_surrogate_model_DT = surrogate_model_DT.predict(X_enc_test)
r2_value_DT = r2_score(pred_model_CB_test, pred_surrogate_model_DT)
print(f'R² score (decision tree surrogate): {r2_value_DT:.2f}')
```

```
R² score (decision tree surrogate): 0.96
```

The $R^2$ score is close to 1, which suggests that the decision tree provides a reliable approximation of the original CatBoost model. Consequently, we can reasonably expect that insights gained from analyzing the decision tree will reflect the workings of the more complex model. The most straightforward method to interpret a decision tree involves visual inspection of its internal branching structure. Note that in the presented visualization, the right branch of a node indicates that the condition at that node is true.

In [ ]:
```
# Plot the fitted decision tree to gain insights into its prediction rules
plt.figure(figsize=(12, 7))
plot_tree(surrogate_model_DT, feature_names=features_X_enc, fontsize=10,
          filled=True, proportion=True, precision=1)
plt.title('Decision Tree Surrogate Model Structure')
plt.show()
```




The decision tree identifies `SMOKER` as the most influential feature, with subsequent splits based on `AGE` and `BMI`. Note that the features `SEX`, `NUMBERCHILDREN`, and `REGION` do not appear in this flat tree's structure, echoing the insights from the exploratory data analysis which suggested that these variables possess limited predictive influence. The decision tree can also be further analyzed, e.g., by examining its internal feature importances.

For our second example, we utilize a linear regression model to serve as an alternate global surrogate model. We will adhere to the same methodology previously demonstrated with the decision tree surrogate model.

In [ ]:
```
# Step 1: Training the CatBoost model has already been completed in Section 3
# via the following code
# model_CB_raw.fit(X_raw_train, y_train, cat_features=categorical_features)

# Step 2: Generate the predictions from the original (black box) model has
# already been completed before
# pred_model_CB_train = model_CB_raw.predict(X_raw_train)
# pred_model_CB_test  = model_CB_raw.predict(X_raw_test)

# Step 3: Train a linear regression (white box) model as the surrogate
# We must use the encoded feature set `X_enc_train` since linear regression
# cannot handle raw categorical data
X_enc_train_with_const = sm.add_constant(
    X_enc_train.drop(columns=['SEX_male', 'SMOKER_no', 'REGION_southwest'])
)
X_enc_test_with_const = sm.add_constant(
    X_enc_test.drop(columns=['SEX_male', 'SMOKER_no', 'REGION_southwest'])
)
surrogate_model_LR = sm.OLS(pred_model_CB_train, X_enc_train_with_const).fit()

# Step 4: Evaluate the surrogate model's performance by comparing its
# predictions to those of the black box model
pred_surrogate_model_LR = surrogate_model_LR.predict(X_enc_test_with_const)
r2_value_LR = r2_score(pred_model_CB_test, pred_surrogate_model_LR)
print(f'R² score (linear regression surrogate): {r2_value_LR:.2f}')
```

```
R² score (linear regression surrogate): 0.87
```

The $R^2$ score for the linear regression model, while somewhat lower than that of the decision tree, is still considered acceptable. However, whether this value is definitively satisfactory can be subjective. For the purposes of this example, we consider the score adequate and proceed to examine the internals of the linear regression model by analyzing its coefficients.

In [ ]:
```
# Print the summary of the fitted linear regression model
# This includes statistics such as coefficients, R-squared, etc.
print(surrogate_model_LR.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.870
Model:                            OLS   Adj. R-squared:                  0.869
Method:                 Least Squares   F-statistic:                     778.2
Date:                Sun, 28 Jul 2024   Prob (F-statistic):               0.00
Time:                        14:20:27   Log-Likelihood:                -9076.0
No. Observations:                 936   AIC:                         1.817e+04
Df Residuals:                     927   BIC:                         1.821e+04
Df Model:                           8
Covariance Type:            nonrobust
===================================================================================
                      coef    std err          t      P>|t|      [0.025      0.975]
-----------------------------------------------------------------------------------
const           -1.117e+04    808.867    -13.805      0.000   -1.28e+04   -9578.812
AGE                257.2564      9.398     27.374      0.000     238.813     275.700
BMI                298.1334     22.243     13.403      0.000     254.481     341.786
NUMBERCHILDREN     307.4062    106.058      2.898      0.004      99.265     515.548
SEX_female         112.5564    260.240      0.433      0.665    -398.172     623.285
SMOKER_yes        2.339e+04    322.253     72.580      0.000    2.28e+04     2.4e+04
REGION_northeast   332.3621    374.447      0.888      0.375    -402.500    1067.224
REGION_northwest    62.7697    371.442      0.169      0.866    -666.196     791.735
REGION_southeast  -247.5136    370.283     -0.668      0.504    -974.203     479.176
==============================================================================
Omnibus:                       21.028   Durbin-Watson:                   2.029
Prob(Omnibus):                  0.000   Jarque-Bera (JB):               32.360
Skew:                          -0.196   Prob(JB):                     9.40e-08
Kurtosis:                       3.822   Cond. No.                         332.
==============================================================================

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

Upon examining the linear regression coefficients, we see that `SMOKER` status is the most influential factor, with a large positive coefficient highlighting its importance. Both `AGE` and `BMI` also show positive and significant contributions to the model's predictions. In contrast, `NUMBERCHILDREN` has a noticeable but only minor impact. The coefficients for `SEX` and different `REGION` categories suggest that these variables do not significantly affect the predictions, given their smaller magnitudes and high $p$-values.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of global surrogate models:

**Advantages**

- Simplicity and Interpretability:
  Global surrogate models, such as linear regression or decision trees, are inherently simpler and more interpretable than complex machine learning models. They approximate the behavior of the black box model, making it easier to understand the overall trends and patterns that the model captures.
- Insight into Feature Importance:
  Global surrogate models can often offer quantitative measures of feature importance, helping to identify which variables have the most significant impact on the model's predictions.
- Regulatory Compliance:
  Global surrogate models can be particularly beneficial in the insurance industry, where actuaries are often required to provide clear explanations for their predictive models under various regulations. These models can bridge the gap between complex algorithmic decisions and the need for transparent, interpretable explanations that satisfy regulatory mandates.

**Disadvantages**

- Approximation Error:
  Global surrogate models are simplifications that aim to mimic the complex model's decisions. The precision of a surrogate model may not always be high, leading to approximation errors where the surrogate's explanations might not fully capture the behavior of the original model.
- Subjectivity in Thresholds for Similarity:
  There is inherent subjectivity in determining the level at which a surrogate model is considered adequately accurate. Metrics like $R^2$ lack universal benchmarks for sufficiency, with acceptability varying by application and the need for precision in explanations.
- Loss of Granularity:
  While global surrogate models are good for providing an overall understanding, they might not capture local behaviors or

interactions specific to individual predictions. This loss of granularity can be critical for use cases where individual decisions are as important as the overall trend.

- Risk of Misinterpretation:
  Users might incorrectly assume that the explanations provided by the surrogate model fully represent the inner workings of the complex model, which can lead to overconfidence in the simplicity of the explanations and misunderstandings when the model behaves unexpectedly.
- Complexity Trade-off:
  The more interpretable a surrogate model is, typically, the less accurate it becomes. This creates a trade-off between how well the surrogate model represents the complex model and how easily its decisions can be interpreted. Choosing the right balance is a subjective process and can impact the effectiveness of the explanations.

## 4.2 Partial Dependence Plot (PDP)

*Main Idea:*

Partial Dependence Plots (PDPs) are a key visualization technique in XAI that show how one or two features affect a machine learning model's predictions when all other features are held constant. Introduced in [3], the main goal of a PDP is to reveal the kind of relationship – e.g., linear, monotonic, or more complex – between a feature and the target outcome. They are particularly useful for understanding how a single feature influences predictions across the entire dataset, independent of other variable values.

*Operational Details:*

Calculating one-dimensional PDPs for a numerical feature involves several steps. Initially, we choose a range of distinct values to explore for the feature we are interested in. For each value in this range, we take our dataset and create copies where the feature of interest is set to this chosen fixed value for all instances. The model then makes predictions for each of these modified datasets. After computing these predictions, we average them to get a single prediction value for our feature of interest at the fixed value. Repeating this for all the selected values from our range, we obtain a series of average prediction values. Plotting these values against the feature's distinct values, we generate the Partial Dependence Plot, which visualizes the relationship between the feature and the averaged predicted outcomes of the model. For categorical variables, a similar process is used, but the feature is varied across its unique categories rather than a continuous range. Two-dimensional PDPs are calculated by selecting a pair of features, creating a grid of their values, and then averaging model predictions with both features set to each grid point. This process can reveal the interaction between the two features and their combined impact on the predictions when visualized as a surface plot.

> **Actuarial Diligence Note**
>
> When applying PDPs, it is crucial to note that the mathematical foundations of this XAI technique assume feature independence – a condition seldomly met in real-world data, where features may be correlated, leading to potentially misleading interpretations. A more in-depth (technical) discussion on this topic is available in [4]. In contrast, ALE plots, which will be introduced in the upcoming Subsection 4.3, do not require independent features and adjust for feature correlations, offering a more nuanced view of the data's structure. Although in practice PDPs and ALE plots can produce similar visualizations (for instance, observe the similarities between PDPs and ALE plots within this notebook), ALE plots are generally more reliable when feature dependence is present. It is important to be aware of this shortcoming when using PDP and to validate its findings with additional XAI techniques.
>
> In the case of the medical costs dataset featured in this notebook, the independence of the features is not present, as can be inferred from the correlations shown in Section 2. Furthermore, we will identify noteworthy differences between the PDP and ALE plot for the variable `BMI` in the Appendix. These differences hint at minor but significant dependencies, such as indicated by the weak correlation between `AGE` and `BMI`, suggesting that the PDP does not fully account for the subtle interdependencies between these features. For a more detailed deep dive into these insights, we refer to Appendix A.2.

*Relation to Other XAI Methods:*

PDPs are related to other XAI methods like Individual Conditional Expectation (ICE) (see Subsection 5.3) and Accumulated Local Effects (ALE) (see Subsection 4.3). ICE plots are similar to PDPs but display one line per instance, showing the model's prediction for an individual instance across a range of feature values. In fact, PDPs can be thought of as the average of all the ICE plot lines, smoothing out individual variances to show a more general trend. ICE plots can reveal heterogeneous effects and feature interactions that PDPs might average out. ALE plots, on the other hand, address the assumption of feature independence by considering the local dependency structure of the dataset. ALE measures the accumulated local effect of a feature on the prediction and is, therefore, less influenced by correlations between features compared to PDP and ICE.

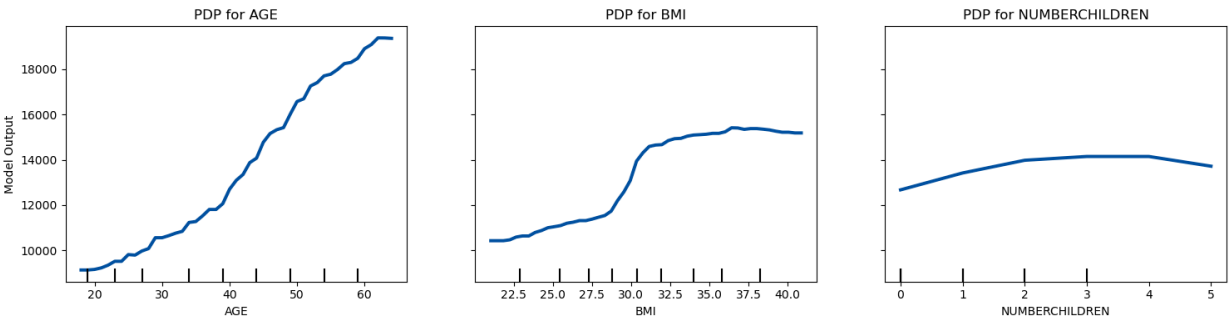
*Interpretation:*

Interpreting one-dimensional PDPs is generally straightforward: the $y$-axis displays the expected prediction (the average predicted outcome), and the $x$-axis displays the values of the feature of interest. A flat line would suggest that the model output is not sensitive to changes in the feature, while a slope indicates a dependency. The steeper the slope, the greater the influence of the feature on the predicted outcome. The interpretation for two-dimensional PDPs is similar, where the interaction between two features and their joint impact on the predicted outcomes are visualized.

*Application to the Medical Costs Dataset:*

We begin the demonstration of Partial Dependence Plots by constructing one-dimensional PDPs for the numerical features within our medical costs dataset.

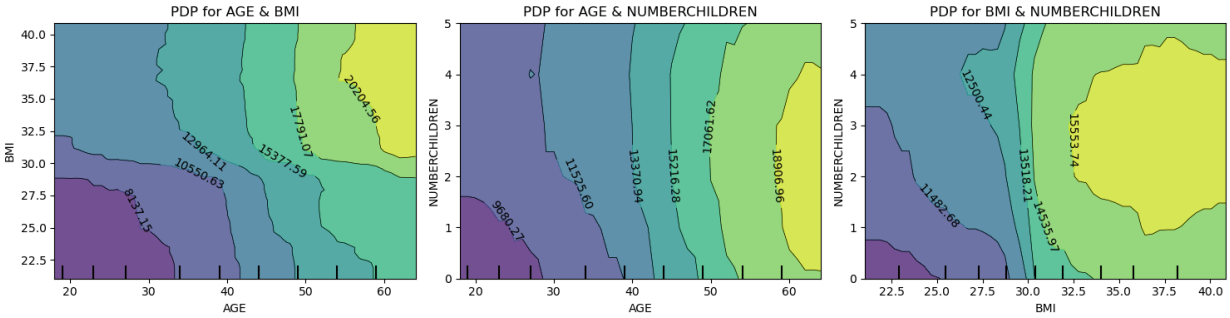
In [ ]:
```
# Determine the positions (indices) of the numerical features in X_raw_test
numerical_feature_positions = [X_raw_test.columns.get_loc(feature)
                               for feature in numerical_features]

# Create the Partial Dependence Plot (PDP)
display = PartialDependenceDisplay.from_estimator(
    model_CB_raw, X_raw_test, features=numerical_feature_positions,
    kind='average', n_cols=3, grid_resolution=50,
    pd_line_kw={'color': COLOR_DARK, 'linewidth': 2.8, 'linestyle': '-'}
)

# Set the figure size
display.figure_.set_size_inches(15, 4)

# Customize the display
for i, axi in enumerate(display.axes_.ravel()):
    axi.set_title(f'PDP for {numerical_features[i]}')
    axi.set_ylabel("Model Output" if i == 0 else "")
    axi.set_xlabel(numerical_features[i])

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




The PDPs for both `AGE` and `BMI` demonstrate a clear monotonic relationship with the model's output, indicating that as these variables increase, so does the predicted cost, albeit at different rates. Additionally, note that the PDP for `BMI` exhibits a steep increase around the value 30. On the other hand, the PDP for `NUMBERCHILDREN` suggests a negligible impact on the model's predictions, as the plot reveals little to no change in the output across different values of this feature.

Next, we generate two-dimensional PDPs for the numerical variables to reveal potential interactions between feature pairs affecting the model's predictions.

In [ ]:
```
# Define the list of pairs of features for which to create two-dimensional PDPs
features_pairs = [('AGE', 'BMI'), ('AGE', 'NUMBERCHILDREN'), ('BMI', 'NUMBERCHILDREN')]

# Create a figure with subplots
fig, axs = plt.subplots(1, 3, figsize=(15, 4))  # 3 plots side by side

# Compute and plot the two-dimensional PDPs on the respective subplots
for i, feature_pair in enumerate(features_pairs):
    display = PartialDependenceDisplay.from_estimator(
        model_CB_raw, X_raw_test, features=[feature_pair],
        kind='average', n_cols=1, grid_resolution=50, ax=axs[i]
    )
    display.axes_[0, 0].set_title(f'PDP for {feature_pair[0]} & {feature_pair[1]}')

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```

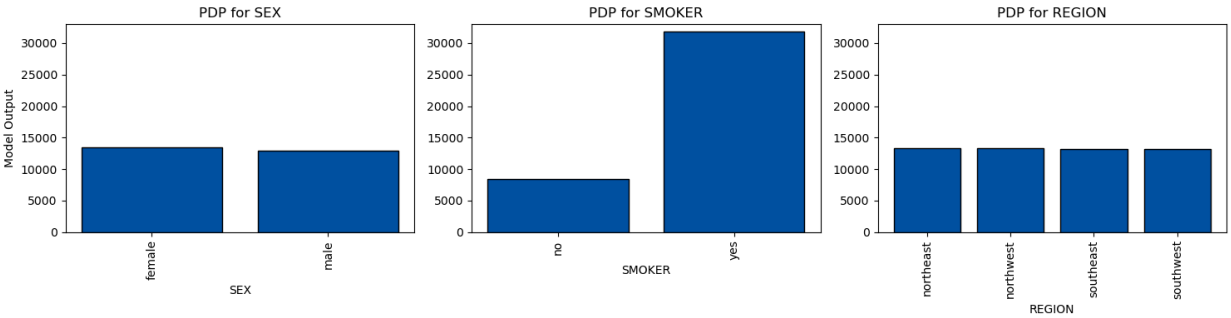



The two-dimensional PDP of `AGE` and `BMI` shows some interactions, with the strong increase around a BMI value of 30 being less pronounced at lower `AGE` values than at higher `AGE` values. The two-dimensional PDPs also reveal that there are almost no interactions between `NUMBERCHILDREN` and both `AGE` and `BMI`.

Having constructed both one-dimensional and two-dimensional Partial Dependence Plots for our numerical features, we now shift our attention to the categorical variables. To gain a similar depth of insight, we will proceed to generate one-dimensional and two-dimensional PDPs for these categorical attributes as well.

We will start with the one-dimensional PDPs for the categorical features.

In [ ]:
```
# Determine the positions (indices) of the categorical features in X_raw_test
categorical_feature_positions = [X_raw_test.columns.get_loc(feature)
                                 for feature in categorical_features]

# Set up the figure for multiple subplots
fig, axs = plt.subplots(ncols=len(categorical_features), figsize=(15, 4))

# Plot partial dependence for each categorical feature
for i, (feature, ax) in enumerate(zip(categorical_features, axs.flatten())):
    display = PartialDependenceDisplay.from_estimator(
        model_CB_raw, X_raw_test, kind='average',
        features=[categorical_feature_positions[i]],
        categorical_features=[categorical_feature_positions[i]],
        feature_names=X_raw_test.columns, ax=ax
    )
    # Adjust colors
    for container in display.axes_[0][0].containers:
        for bar in container:
            bar.set_color(COLOR_DARK)
            bar.set_edgecolor('black')
    # Customize the display
    for j, axi in enumerate(display.axes_.ravel()):
        axi.set_title(f'PDP for {feature}')
        axi.set_ylim([0, 33000])
        axi.set_ylabel("Model Output" if i == 0 else "")
        axi.set_xlabel(feature)

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```

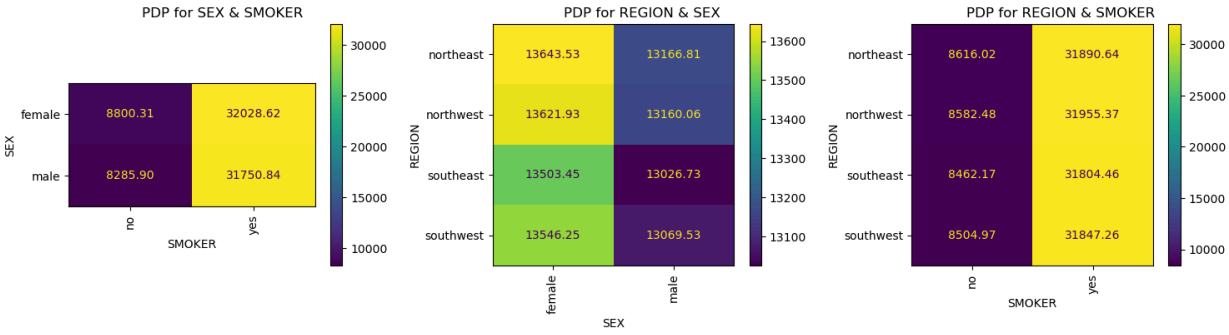



From the Partial Dependence Plots depicted above, we infer that the features `SEX` and `REGION` exhibit minimal impact on the model's predictions. Conversely, the PDP for `SMOKER` indicates that smoking status has a substantial effect on the model's output, highlighting it as an influential factor in the prediction process.

Next, we will implement the two-dimensional PDPs for the categorical features.

In [ ]:
```
# Define pairs of features for which to plot 2D PDPs
feature_pairs = [('SEX', 'SMOKER'), ('REGION', 'SEX'), ('REGION', 'SMOKER')]

# Determine the column indices for categorical features
category_indices = {
```

```
    'SEX':    X_raw_train.columns.get_loc('SEX'),
    'SMOKER': X_raw_train.columns.get_loc('SMOKER'),
    'REGION': X_raw_train.columns.get_loc('REGION')
}

# Set up the figure for multiple subplots
fig, axs = plt.subplots(ncols=len(feature_pairs), figsize=(15, 4))

# Plot partial dependence for each feature pair
for i, feature_pair in enumerate(feature_pairs):
    feature_indices = [category_indices[feat] for feat in feature_pair]
    display = PartialDependenceDisplay.from_estimator(
        model_CB_raw, X_raw_train, features=[feature_indices],
        categorical_features=feature_indices, kind='average', ax=axs[i],
        feature_names=X_raw_train.columns
    )
    display.axes_[0][0].set_aspect(0.5)
    # Set the title for each subplot
    axs[i].set_title(f'PDP for {feature_pair[0]} & {feature_pair[1]}')

# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




The two-dimensional Partial Dependence Plots clearly demonstrate that the `SMOKER` variable is the dominant factor affecting the model's predictions, while variables such as `SEX` and `REGION` appear to have negligible effects. A subtle interaction is noticeable in the PDP for `SEX` and `SMOKER`, showing that the higher costs for females compared to males are more pronounced for non-smokers than for smokers.

Note that while it is possible to generate a two-dimensional PDP combining one numerical and one categorical feature, we have chosen to exclude this for the sake of brevity.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of Partial Dependence Plots:

**Advantages**

- Intuitive Visualization:
  Partial Dependence Plots offer a clear and accessible way to depict the relationship between a feature and the predicted outcomes, facilitating easy interpretation of complex model behaviors.
- Non-Technical Stakeholder Engagement:
  PDPs provide visual explanations that can be readily presented to stakeholders without a data science background, aiding in transparent decision-making processes.
- Feature Interaction Insights:
  Two-dimensional PDPs can reveal the interaction effects between pairs of features on the predicted outcome, offering a deeper understanding of how feature combinations influence model predictions.

**Disadvantages**

- Assumption of Feature Independence:
  PDPs assume that the features are independent which is rarely the case in real-world data. This can lead to incorrect interpretations if features are correlated.
- Creation of Unfeasible Instances:
  By design, PDPs can suggest interpretations based on combinations of feature values that do not occur naturally, leading to conclusions about non-existent or unfeasible instances.

- Limited by Dimensionality:
  Designed for one or two features at a time, PDPs struggle in high-dimensional space where feature interactions are complex.
- Oversimplified Representation:
  By averaging out the effects of all other features, PDPs can oversimplify the model behavior and may not accurately reflect the local nuances that could be captured by Individual Conditional Expectation (ICE) plots.

## 4.3 Accumulated Local Effects (ALE)

*Main Idea:*

Accumulated Local Effects (ALE) plots, introduced in [5], are used to understand how individual features influence model predictions by focusing on their local effects rather than global ones. This means that ALE plots capture subtle variations that other methods might miss. Unlike Partial Dependence Plots (PDPs, see Subsection 4.2), which show the average effect of a feature over the entire dataset and assume that features are independent, ALE plots examine the impact of small changes in feature values within local regions of the data. This approach addresses PDP's limitations by providing a more accurate reflection of a feature's effect in the presence of interactions with other features.

*Operational Details:*

For numerical features, ALE is calculated by partitioning the range of a feature into intervals and accumulating the differences in predictions that result from perturbing the feature within these intervals. This is done by evaluating the model with feature values slightly above and below the observed value and taking the average of these prediction changes. The local effect is then the accumulated average across these intervals for a feature. In the case of categorical features, ALE determines the predictive effects by comparing how the model's outputs vary when the feature value changes between categories. It then integrates these variations to capture the overall influence of category changes throughout the dataset.

*Relation to Other XAI Methods:*

ALE plots provide a valuable connection between PDPs (see Subsection 4.2) and Individual Conditional Expectation (ICE) plots (see Subsection 5.3). Contrasting with PDPs that examine average effects and ICE plots that detail paths for individual instances, ALE strikes a balance by aggregating localized changes – circumventing assumptions of feature independence and thus furnishing a more faithful representation of a feature's influence.

*Interpretation:*

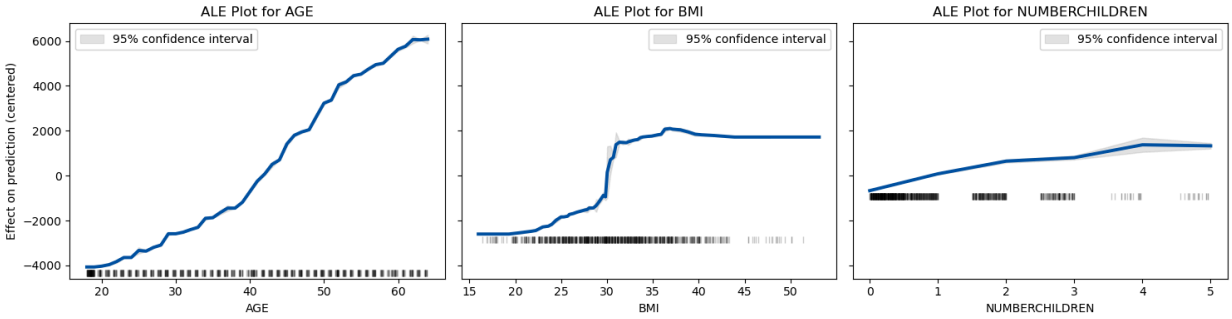
Interpreting ALE plots involves examining the shape and slope of the curve. A flatter curve suggests that the feature has a minimal effect on the model's predictions within that range, while a curve with a steep slope or significant fluctuations indicates that the feature has a substantial effect that may vary across different values.

*Application to the Medical Costs Dataset:*

First, we examine the one-dimensional ALE plots corresponding to the numerical features in our dataset.

In [ ]:
```
# Set up a figure with subplots (one for each feature)
fig, axes = plt.subplots(1, len(numerical_features), figsize=(15, 4), sharey=True)

# Generate ALE plot for each feature
for i, feature in enumerate(numerical_features):
    ale_plot = ale(
        X=X_raw_train,        # DataFrame containing the training feature data
        model=model_CB_raw,   # Trained model
        feature=[feature],    # Feature to analyze (as a list)
        feature_type='continuous',
        grid_size=50,         # Grid size, can be set according to needs
        include_CI=True,
        fig=fig,
        ax=axes[i]            # Use the i-th axis for the plot
    )

    # Adjust line style and color
    axes[i].lines[0].set_color(COLOR_DARK)
    axes[i].lines[0].set_linewidth(2.8)

    # Set title for each subplot
    axes[i].set_title(f"ALE Plot for {feature}")

    # Clean up x and y labels if necessary
    if i > 0:
        axes[i].set_ylabel('')
    axes[i].set_xlabel(feature)
```

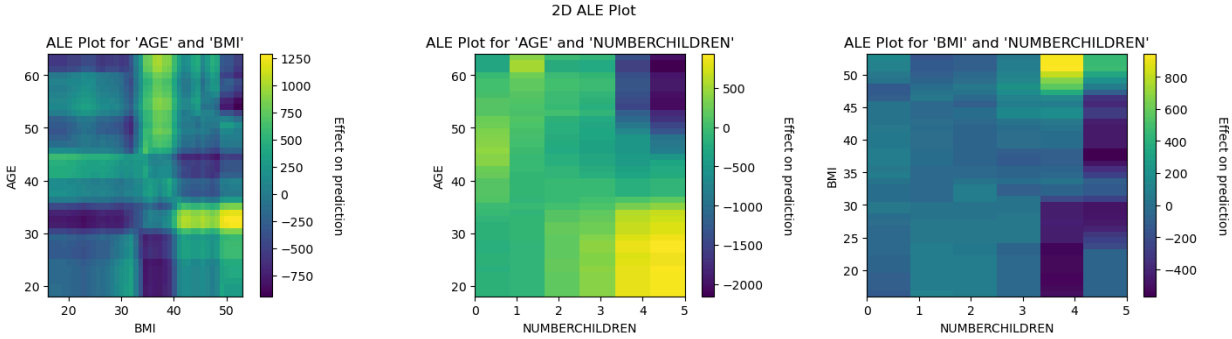
```
# Adjust the layout and show the plot
plt.tight_layout()
plt.show()
```




Observe that the scale of the ALE plots is now centered, distinguishing them from the previously discussed Partial Dependence Plots. Although the overall contours of the ALE plots resemble their PDP counterparts, there are notable variances: In the case of `BMI`, the ALE plot reveals a more pronounced incline around a BMI of 30 and displays a downward trend beginning at 37, contrasting with the nearly monotonically increasing curve of the PDP. Furthermore, the ALE plot for `NUMBERCHILDREN` differs as it does not exhibit the reduction between 4 and 5 children that is apparent in the PDP.

Lastly, we visualize the interactions between selected feature pairs using two-dimensional ALE plots.

In [ ]:
```
# Define feature pairs to plot
feature_pairs = [
    ('AGE', 'BMI'),
    ('AGE', 'NUMBERCHILDREN'),
    ('BMI', 'NUMBERCHILDREN')
]

# Set up a figure with three subplots (1 row, 3 columns)
fig, axes = plt.subplots(nrows=1, ncols=3, figsize=(15, 4))

# Iterate over feature pairs and axes to create ALE plots
for i, (feature1, feature2) in enumerate(feature_pairs):
    # Use the ale function, providing the corresponding ax
    ale_plot = ale(
        X=X_raw_train,               # DataFrame containing the training data
        model=model_CB_raw,          # Trained model
        feature=[feature1, feature2], # Feature pair to analyze
        grid_size=40,                # Grid size, can be set according to needs
        include_CI=False,
        fig=fig,
        ax=axes[i]                   # Use the i-th axis for the plot
    )
    axes[i].set_aspect(0.125 if i != 0 else 1)
    # Set the title for each subplot
    axes[i].set_title(f"ALE Plot for '{feature1}' and '{feature2}'")

# Adjust the layout so that plots do not overlap
plt.tight_layout()
plt.show()
```




The two-dimensional ALE plot for the `AGE` and `BMI` interaction highlights two specific negative impact zones on predicted costs: individuals around thirty with BMIs under 30, and younger individuals with BMIs in the mid-thirties. The ALE plots for interactions with `NUMBERCHILDREN` show less influence on predictions, with notable effects only in sparse data regions associated with high child counts.

Please note that for brevity, we have omitted the inclusion of both one-dimensional and two-dimensional ALE plots for categorical features, as well as combined two-dimensional ALE plots that involve one numerical and one categorical feature.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of Accumulated Local Effects:

**Advantages**

- Handles Feature Dependency:
  Unlike Partial Dependence Plots (PDPs), ALE plots take into account the potential interactions among features by focusing on local effects of features, thus providing more accurate representations when features are correlated.
- Reduced Computational Cost:
  ALE plots require fewer computations than PDPs when dealing with high-cardinality categorical features or continuous features, because they do not have to average predictions across all possible values of other features.
- Localized Interpretations:
  ALE plots provide insights into the effects of features on predictions in different regions of the feature space, which can be more informative about the actual operation of the model in specific localized areas, compared to the more global perspective of PDPs.

**Disadvantages**

- More Complex Calculation:
  The calculation and explanation of ALE plots are more complex than those of PDPs, making them particularly challenging to present to non-technical stakeholders.
- Potentially Misleading with Sparse Data:
  In regions of the feature space where data is sparse, ALE estimates might be unreliable due to the lack of sufficient data to calculate accurate local accumulated effects.
- Dimensionality Constraints:
  Similar to PDPs, ALE plots are primarily useful for visualizing one or two features at a time, and they may not effectively communicate the presence of complex multidimensional interactions.
- Lack of Individual Variation Insights:
  Unlike PDPs, ALE plots are not accompanied by ICE curves and therefore it is hard to dectect heterogeneity in the feature effect.

## 4.4 Permutation Feature Importance (PFI)

*Main Idea:*

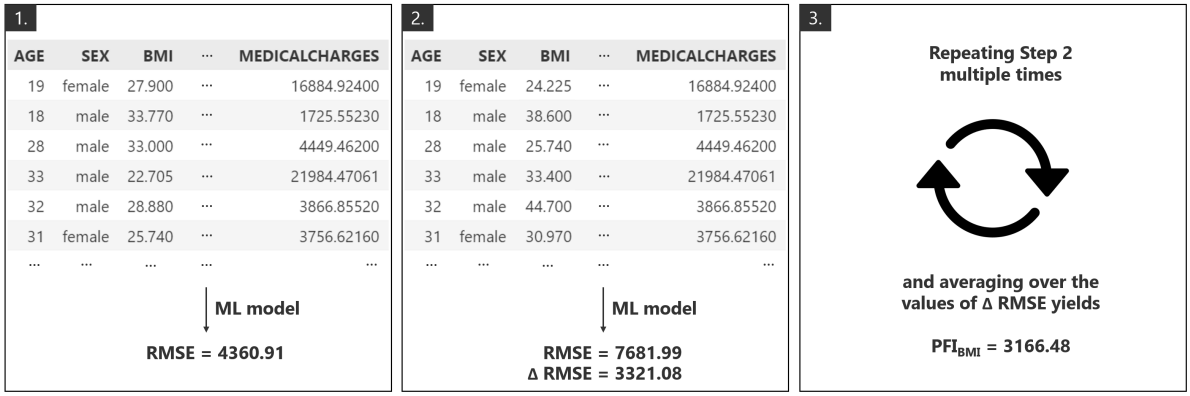
Feature importances play a crucial role in understanding the decision-making process of machine learning models by identifying how much each feature contributes to the model's predictions. This information is valuable not only for interpreting the model but also for feature engineering and feature selection for future model iterations. Permutation Feature Importance (PFI) is a model-agnostic technique used to measure the importance of a feature by calculating the decrease in the model's performance when that feature's values are shuffled, thereby breaking the relationship between the feature and the outcome. Initially, PFI was used in the context of random forests, see [6].

*Operational Details:*

PFI is calculated using the following method for both numerical and categorical variables. The process involves the following steps, repeated for each feature separately:

1. The model is initially evaluated on a dataset to establish a baseline performance score, using a metric such as RMSE or $R^2$ score.
2. For the feature under consideration, the values are permuted in the dataset – randomly shuffled to disrupt the correlation between that feature and the target – and the model is re-evaluated on this perturbed dataset. The performance decrease, as compared to the baseline, is recorded as the importance score for the feature.
3. The permutation and evaluation process is typically repeated multiple times to obtain a reliable estimate of feature importance, and the average drop in performance is calculated.

The following picture provides a visual illustration of the three steps outlined above for calculating the PFI of the feature `BMI`.




*Interpretation:*

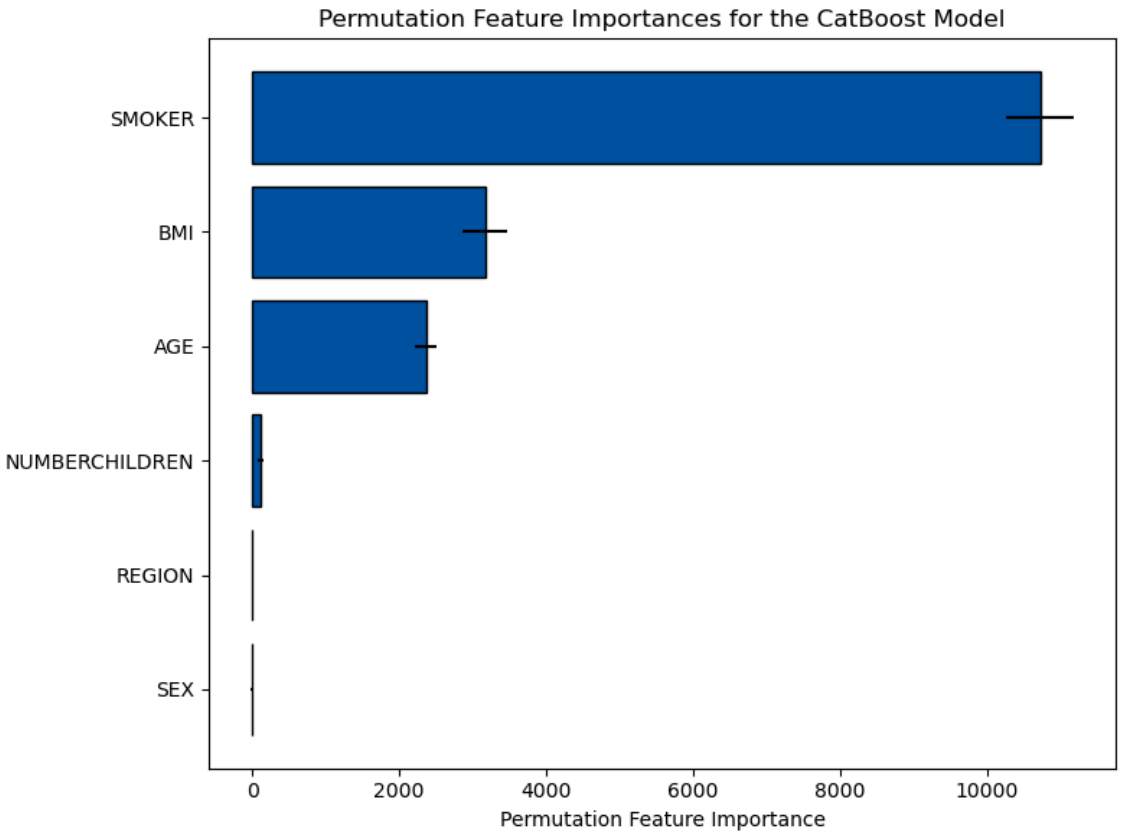
The heuristic interpretation of PFI results is as follows: A notable decrease in the model's performance indicates that the feature is significant for the model's predictive accuracy. Conversely, a minor or insubstantial decrease points to a feature being non-essential. An advantage of PFI is its simplicity and applicability to any model, regardless of its complexity. However, it is essential to note that PFI can be sensitive to data leakage or features that are artificially over-represented. It can also fail to detect interactions between features since it evaluates them in isolation.

> **Actuarial Diligence Note**
>
> When implementing permutation feature importance (PFI), one has to be aware of its fundamental assumption that shuffling a feature's values does not impact the distribution of other features. This assumption can be problematic in real-world datasets where features are often correlated, potentially leading to misleading interpretations. To ensure robustness in the conclusions drawn from PFI, it is advisable to compare its outcomes with other feature importance measures. This could include models' internal feature importances (if existent), like those within CatBoost, feature importances derived from SHAP (see Subsection 5.1), or other relevant methods. These additional methods, which typically take into account the model's structure, can provide alternative importance rankings and thus a more comprehensive understanding of feature significance. For a detailed comparison using the medical costs dataset, refer to Appendix A.3.

*Application to the Medical Costs Dataset:*

The code snippet below demonstrates the application of PFI on our medical costs dataset.

In [ ]:
```
# Perform permutation feature importance
results = permutation_importance(
    model_CB_raw, X_raw_test, y_test,
    scoring='neg_root_mean_squared_error',  # Metric for scoring permutations
    n_repeats=1000,                         # Number of times to permute a feature
    random_state=RANDOM_SEED                # The random seed for reproducibility
)

# Sort feature importances in descending order and get their indices
sorted_idx = results.importances_mean.argsort()

# Get the standard deviations for the sorted features
sorted_std = results.importances_std[sorted_idx]

# Plot the feature importances
plt.figure(figsize=(8, 6))
plt.barh(
    range(len(sorted_idx)),
    results.importances_mean[sorted_idx],
    color=COLOR_DARK,
    edgecolor='black',
    xerr=sorted_std  # Include empirical standard errors
)
plt.yticks(range(len(sorted_idx)),
           [X_raw_test.columns[i] for i in sorted_idx])
plt.xlabel("Permutation Feature Importance")
plt.title("Permutation Feature Importances for the CatBoost Model")
plt.tight_layout()
plt.show()
```




Confirming findings from the initial exploratory data analysis, the permutation feature importance underscores that `SMOKER` status is the dominant feature, wielding the highest predictive influence. The subsequent features, `BMI` and `AGE`, are also critical to the model's predictions. In contrast, the remaining attributes – `NUMBERCHILDREN`, `REGION`, and `SEX` – are shown to have little to no effect on the predictive outcome.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of Permutation Feature Importance:

**Advantages**

- Intuitive Interpretation:
  The method is straightforward to understand – features that impact the model performance more significantly when permuted are considered more important.
- Useful for Feature Selection:
  Permutation feature importance can be used as a tool for feature selection by identifying the features that do not contribute significantly to the model's predictive power.

**Disadvantages**

- Impact of Correlated Features:
  When features are correlated, permuting one feature can also indirectly affect the importance of another, which can lead to misinterpreting feature importance.
- Generation of Unrealistic Data:
  The shuffling of values to assess importance can result in the generation of improbable or impossible data combinations, which can in turn affect the reliability of the importance measurements.
- Computationally Expensive:
  For models that are slow to predict or datasets that are large, computing PFI can be computationally expensive because it involves re-evaluating the model multiple times.
- Does Not Account for Model Internals:
  Permutation feature importance is purely based on changes in model performance without considering the internal structure or coefficients of the model that might give more insight into how features are used by the model.
- Randomness in Importance Scores:
  The randomness introduced by permuting features can sometimes lead to variability in importance scores across multiple

runs, especially when the dataset is small or the model is very complex.

## 5. Local Model-Agnostic Explainability Methods

In this section, we examine *local* model-agnostic explainability methods, where the objective is to understand the decision-making process of machine learning models at an individual prediction level. Such methods can, for instance, provide insights into how each feature affects a single prediction, or they may illuminate the localized behavior of the model in the area surrounding a specific instance. We focus on the key techniques Shapley Additive Explanations (SHAP), Local Interpretable Model-Agnostic Explanations (LIME), and Individual Conditional Expectation (ICE) plots.

Throughout this section, our goal is to detail the model's decision-making process for the following particular entry within our medical cost dataset:

In [ ]:
```
# Instance of interest
observation_index = 4

# Combine the independent features and the prediction into one DataFrame
observation_features = X_raw_test.iloc[[observation_index]]
observation_prediction = model_CB_raw.predict(observation_features).item()
observation_combined = observation_features.assign(
    MEDICALCHARGES_predicted = observation_prediction
)

# Display the combined information as one line
observation_combined
```

Out[ ]:

| | AGE | SEX | BMI | NUMBERCHILDREN | SMOKER | REGION | MEDICALCHARGES_predicted |
|---|---|---|---|---|---|---|---|
| **484** | 48 | male | 34.3 | 3 | no | southwest | 11443.273135 |

### 5.1 Shapley Additive Explanations (SHAP)

*Main Idea:*

Shapley Additive Explanations (SHAP) is a powerful XAI method based on cooperative game theory that provides a way to explain the output of machine learning models. Introduced in [7], it assigns each feature an importance value for a particular prediction by simulating a "game" where each feature value of the instance is considered a "player" and the prediction is the "payout". SHAP values are calculated by averaging the marginal contributions of a feature across all possible combinations of features (see below). This framework ensures that SHAP values are – according to Shapley's fairness criteria – fairly distributed among the features and the sum of SHAP values explains why the model's output deviates from the baseline prediction.

*Operational Details:*

The Shapley value $\phi_i$ for the $i$-th feature value of instance $x$ is computed as follows:

$$\phi_i(f,x) = \sum_{S \subseteq N \setminus \{i\}} \underbrace{\frac{|S|!(|N|-|S|-1)!}{|N|!}}_{\text{Shapley weight}} \cdot \underbrace{\Big[f_x(S \cup \{i\}) - f_x(S)\Big]}_{\text{marginal contribution of } i\text{-th feature}}$$

In the formula above, we have:

- $f$ is our CatBoost model.
- $x$ is the specific instance for which we want to compute the SHAP values.
- $N$ is the set of all features.
- $S$ is a subset of features excluding the $i$-th feature.
- $|S|$ is the number of features in the subset $S$.
- $|N|$ is the total number of features.
- $f_x(S)$ is the prediction made by the model using the features in $S$ and the baseline values for all other features.

For an in-depth understanding of the technical aspects and methodology behind computing SHAP values, particularly the calculation of a feature's marginal contribution, readers are referred to [4].

Note that the SHAP values for each feature sum up to the difference between the prediction for the input $x$ and the average prediction across all data points:

$$f(x) = \text{mean prediction} + \sum_{i \in N} \phi_i(f,x)$$

> **Actuarial Diligence Note**
>
> The computation of Shapley values is inherently computationally expensive due to the necessity to evaluate the model's prediction for all possible feature subset combinations. This complexity grows exponentially with the number of features, making exact calculations impractical for models with a substantial number of features. To address this challenge, various approximation methods have been developed. *Kernel SHAP* employs a weighted linear regression to approximate SHAP values, offering a trade-off between accuracy and computational efficiency and allowing for application to any model. *Tree SHAP*, on the other hand, is designed specifically for tree-based models and leverages the internal structure of decision trees to compute exact SHAP values much more efficiently. These specialized approaches reduce computational load and make the application of SHAP values feasible in practical scenarios, while still adhering to the original Shapley value properties. For a detailed examination of the nuanced differences between various SHAP value approximation techniques as applied to our medical costs dataset, please see Appendix A.3, which offers a critical analysis of how these approximations impact our results.

*Interpretation:*

Interpreting SHAP values provides both global and local insights. Locally, one can look at the SHAP values for each feature of a single prediction to understand how much each feature contributed to the prediction and in what direction. A positive SHAP value for a feature implies that this feature pushed the model's prediction higher, while a negative value would imply the opposite. Globally, aggregating SHAP values over a dataset can provide insights into feature importance, showing which features have more impact on the model's predictions across all data points. Furthermore, SHAP values can also be utilized highlight interactions between features. SHAP interaction values quantify not only the individual impact of each feature but also the combined effects of feature pairs on the prediction.

*Application to the Medical Costs Dataset:*

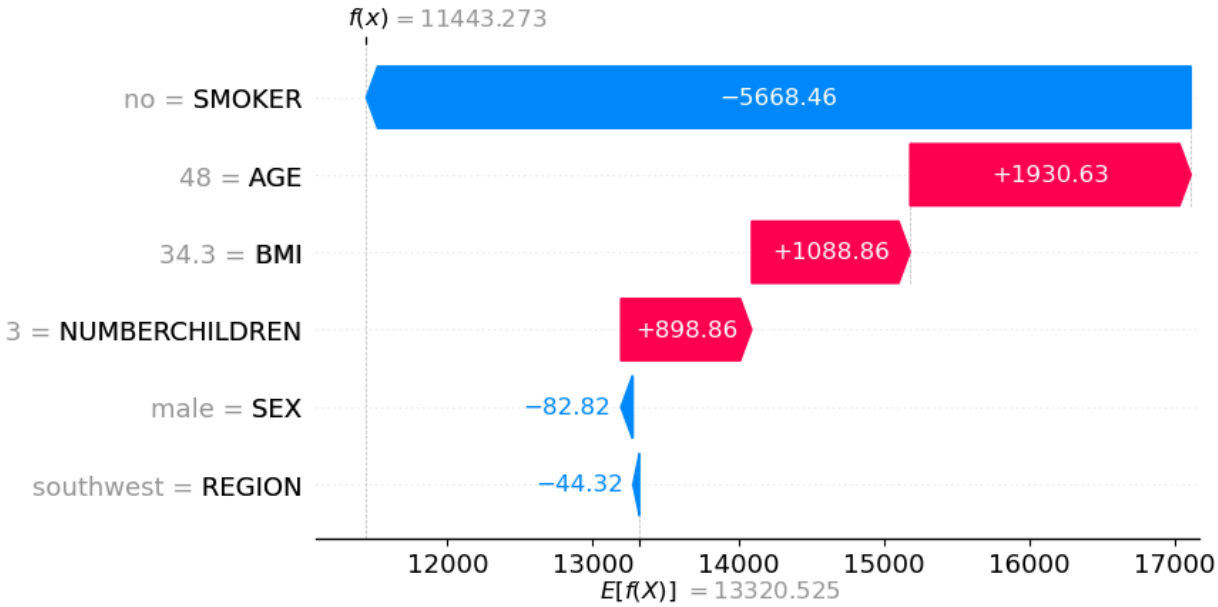
Below, we will compute (an approximation of) the Shapley values utilizing the `KernelExplainer` from the `shap` library. We chose this method because it is model-agnostic and hence can be applied to any machine learning model. While this method is thorough, it is computationally intensive. For tree-based models such as CatBoost, for instance, `TreeExplainer` is a faster alternative (see the Actuarial Diligence Note above).

In [ ]:
```
# Create a SHAP KernelExplainer using the training set
explainer = shap.KernelExplainer(model_CB_raw.predict, X_raw_train)

# Calculate SHAP values for the test set
shap_values = explainer.shap_values(X_raw_test)
```

```
shap:WARNING: Using 936 background data samples could cause slower run times. Consider using shap.sample(data, K) or sh
ap.kmeans(data, K) to summarize the background as K samples.
  0%|          | 0/402 [00:00<?, ?it/s]
```

The subsequent code will generate a SHAP waterfall plot, which visually represents the contribution of each feature to a specific prediction. Beginning with the base value – the model's average prediction – the plot stacks each feature's effect, with positive impacts extending to the right (in red) and negative ones to the left (in blue). To interpret the plot, start from the base value and follow the features' contributions upwards to arrive at the final model prediction at the top.

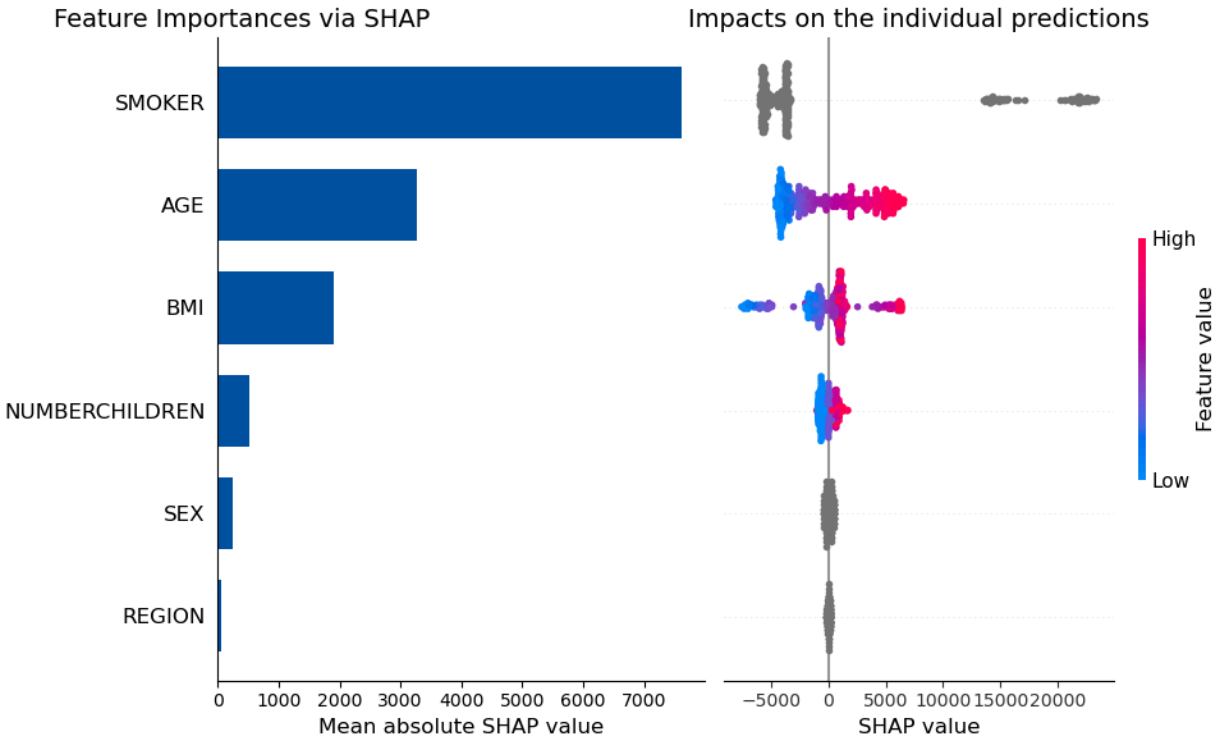
In [ ]:
```
# Create an Explanation object for the particular instance of interest
explanation = shap.Explanation(values=shap_values[observation_index],
                               base_values=explainer.expected_value,
                               data=X_raw_test.iloc[observation_index])

# Show the SHAP waterfall plot
shap.waterfall_plot(explanation)
```




From the SHAP waterfall plot above, we can see that our specific prediction of 11443.273 breaks down into the model's average guess of 13320.525, plus or minus the effects of different features. For the observed individual, not being a `SMOKER` greatly lowers the predicted costs. On the other hand, being older, having a higher `BMI`, and having more children increase the costs. Features like `SEX` and `REGION`, however, seem to have almost no impact on this particular prediction.

SHAP's local interpretations can be aggregated to provide a global perspective on explainability. The upcoming figure features a SHAP summary plot on the right, wherein the SHAP values of all instances are plotted together. By taking the absolute values of these SHAP values and computing their average, we obtain a measure of feature importance for each feature. On the left side of the figure, we present a plot that displays these aggregated feature importances across all features. This approach facilitates a comprehensive understanding of the most influential factors in the model.

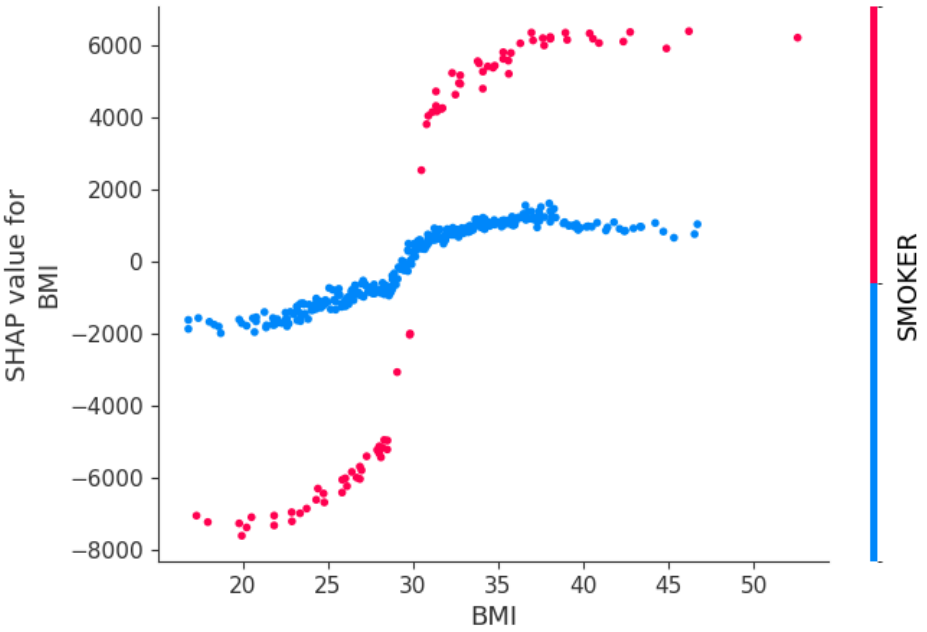
In [ ]:
```
# Calculate the mean absolute SHAP values for each feature
mean_abs_shap_values = np.abs(shap_values).mean(0)

fig = plt.figure()
ax0 = fig.add_subplot(121)
shap.bar_plot(mean_abs_shap_values, feature_names=X_raw_test.columns,
              show=False)

# Change the colormap of the artists
for fc in plt.gcf().get_children():
    # Ignore last Rectangle
    for fcc in fc.get_children()[:-1]:
        if isinstance(fcc, matplotlib.patches.Rectangle):
            if matplotlib.colors.to_hex(fcc.get_facecolor()) == "#ff0051":
                fcc.set_facecolor(COLOR_DARK)
        elif isinstance(fcc, plt.Text):
            if matplotlib.colors.to_hex(fcc.get_color()) == "#ff0051":
                fcc.set_color(COLOR_DARK)

# Create the plot
title_obj = plt.title('Feature Importances via SHAP', fontsize=14)
pos = title_obj.get_position()
title_obj.set_position([0.0495, pos[1]])
ax0.tick_params(axis='x', labelsize=10)
ax0.tick_params(axis='y', labelsize=12)
plt.xlabel('Mean absolute SHAP value', fontsize=12)
ax1 = fig.add_subplot(122)
shap.summary_plot(shap_values, X_raw_test, show=False)
plt.gcf().axes[-1].set_aspect(30)
plt.title('Impacts on the individual predictions', fontsize=14)
plt.xlabel('SHAP value', fontsize=12)
ax1.tick_params(axis='x', labelsize=10)
ax1.set_yticklabels([])
ax1.set_ylim(-0.61, 5.6)
plt.gcf().set_size_inches(10, 6)
plt.tight_layout()
plt.show()
```




Echoing the findings from Permutation Feature Importance examined in Subsection 4.4, the feature `SMOKER` emerges as the most influential in affecting our model's predictions. It is followed in significance by `AGE` and `BMI`, which likewise contribute notably to the model's decision-making process. The remaining features, `NUMBERCHILDREN`, `SEX`, and `REGION`, have only minimal importance.

Next, we turn to SHAP dependence plots, a valuable tool for examining the effect of a single feature on model predictions and pinpointing potential interactions with other features. We use this plot type to analyze how the `BMI` feature impacts medical cost predictions from our CatBoost model and investigate if this effect varies with the `SMOKER` status. The code below produces a dependence plot for `BMI`, with data points colored based on `SMOKER` values, offering insight into the interaction between these two features.

In [ ]:
```
# Create and display the SHAP dependence plot for 'BMI' and 'SMOKER'
shap.dependence_plot('BMI', shap_values, X_raw_test, interaction_index='SMOKER')
```




The SHAP dependence plot clearly illustrates that for individuals identified as smokers (`SMOKER` status 'yes', colored in red), the SHAP values for `BMI` exhibit a considerably sharper upward trend once BMI exceeds 30, in comparison to non-smoking individuals. This suggests that being a smoker amplifies the influence of higher BMI on the predicted medical costs.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of SHAP:

**Advantages**

- Strong Theoretical Fundament:
  SHAP values adhere to a consistent attribution of contributions across features, drawing from cooperative game theory to provide a fair and stable distribution of predictive 'credits'.
- Detailed Insights for Individual Predictions:
  SHAP allows for a deep dive into the decision-making process for individual data points, offering specific, nuanced explanations rather than broad generalizations.
- Intuitive Visualizations:
  SHAP's visual tools, like the waterfall plot, provide clear and intuitive visual representations of feature contributions, facilitating easier communication and understanding of model behavior for various audiences.
- Local to Global Explainability:
  While SHAP provides explanations for individual predictions, these local explanations can be aggregated to quantify global feature importance, offering insights into the model's overall behavior.

**Disadvantages**

- Computational Intensity:
  Computing SHAP values, especially with methods like `KernelExplainer`, can be computationally expensive, posing challenges with large datasets or complex models.
- Complexity for End-Users:
  While visualizations are intuitive, the underlying concept of SHAP and the interpretation of individual contributions can be complex for end-users without a technical background.

## 5.2 Local Interpretable Model-Agnostic Explanations (LIME)

*Main Idea:*

Local Interpretable Model-Agnostic Explanations (LIME), introduced in [8], is designed to approximate a given black box model locally by an interpretable model, such as a linear model or decision tree, that is easier to understand. By doing so, LIME provides explanations for individual predictions (local explanations), revealing which features are most important for the model's decision in a neighborhood of the particular instance of interest.

*Operational Details:*

The calculation process for a LIME explanation involves several steps. Given an instance for which we seek an explanation, LIME generates a set of new samples around the instance by perturbing its feature values. It then applies the complex model to these new samples to calculate the corresponding predictions. Next, it weighs these samples according to their proximity to the original instance – samples that are more similar to the original instance have a larger influence. LIME then fits an interpretable model, such as a linear regression or a decision tree, to the new samples using the calculated predictions and weights. The interpretable model aims to mimic the complex model's behavior in a neighborhood of the instance being explained. The coefficients or structure of this simple model offer insights into which features have the greatest impact on the prediction.

*Interpretation:*

Interpreting the output of LIME on basis of a linear regression typically involves looking at the coefficients of the underlying linear model. In such a case, each coefficient represents the contribution of the corresponding feature to the prediction for the instance at hand. A positive coefficient indicates a feature that contributes to increasing the prediction value, while a negative coefficient indicates a feature that contributes to decreasing it. The magnitude of the coefficient reflects the strength of the feature's influence. While LIME provides valuable insights into individual predictions, it is important to remember that the explanations are local; different models or even slightly different instances can lead to different explanations. Therefore, LIME is best used to explore the model behavior around specific predictions, rather than to derive global conclusions about the model's behavior.

*Application to the Medical Costs Dataset:*

The `lime` packages used below for the application of LIME requires the input data to be label encoded. As a first step, we hence employ label encoding on our dataset.

In [ ]:
```
# Encode categorical features using LabelEncoder
label_encoders = {
    feature: LabelEncoder().fit(df_raw[feature])
    for feature in categorical_features
}

# Function to transform the dataset using the encoders and inverse transform it
def transform_data(df, label_encoders, inverse=False):
    df_transformed = df.copy()
    for feature, le in label_encoders.items():
        if inverse:
            # Manually reverse the transformation to handle unseen labels
            mapping = dict(zip(le.transform(le.classes_), le.classes_))
            df_transformed[feature] = df_transformed[feature].apply(
                lambda x: mapping.get(x, x)
            )
        else:
            try:
                df_transformed[feature] = le.transform(df_transformed[feature])
            except ValueError as e:  # Catch the exception for unseen labels
                # Handle unseen labels - to be determined based on requirements
                pass
    return df_transformed

# Transform categorical features of the raw dataframe
df_labelenc = transform_data(df_raw, label_encoders)

# Drop the target column to obtain the features DataFrame
X_labelenc = df_labelenc.drop(target, axis=1)

# Create the categorical_names dict with the proper format
categorical_names = {}
categorical_features_indices = []
for feature in categorical_features:
    le = label_encoders[feature]  # Get the LabelEncoder for the feature
    feature_index = X_labelenc.columns.get_loc(feature)  # Get feature index
    categorical_features_indices.append(feature_index)
    categorical_names[feature_index] = dict(
        zip(le.transform(le.classes_), le.classes_)
    )

# Use the same train-test split as before
X_labelenc_train, X_labelenc_test, _, _ = train_test_split(
    X_labelenc, y, train_size=TRAIN_RATIO, random_state=RANDOM_SEED
)
```

Next, we apply LIME to obain an explanation of the considered intance of interest.

In [ ]:
```
# Initialize the LIME explainer
explainer = LimeTabularExplainer(
    X_labelenc_train.values,
    feature_names=X_labelenc.columns.tolist(),
    mode='regression',
    categorical_features=categorical_features_indices,
    categorical_names=categorical_names,
    kernel_width=3
)

# Create a function to predict with CatBoost on the LIME format data
def predict_fn(data_array):
    data_df = pd.DataFrame(data_array, columns=X_labelenc_test.columns)
    # Inverse transform the categorical features before prediction
    retransformed_df = transform_data(data_df, label_encoders, inverse=True)
    predictions = model_CB_raw.predict(retransformed_df).astype(float)
    return predictions

# Apply LIME on a particular instance from the test set
exp = explainer.explain_instance(
    X_labelenc_test.iloc[observation_index].values, predict_fn
)
exp.show_in_notebook(show_all=True)
```


The LIME analysis for this particular instance indicates that being a non-`SMOKER` notably reduces the predicted medical costs, whereas a higher `BMI`, being of middle `AGE`, and having more children are factors contributing to an increase in the model's cost predictions. `SEX` and the `REGION`, on the other hand, show negligible effects on the predicted outcome for this individual.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of LIME:

**Advantages:**

- Intuitive Explanations Using White Box Models Locally:
  LIME explains individual predictions by using an interpretable white box model. This makes the complex underlying model's behavior transparent and easy to understand, helping users see the factors influencing a specific prediction.
- Flexibility for Different Types of Data:
  LIME can be applied to a variety of data types, including tabular data, text, and images, making it useful for a wide range of applications.

**Disadvantages:**

- Local Approximations May Not Capture Global Behavior:
  LIME focuses on local explanations and thus might not accurately represent the model's behavior globally or on other instances beyond the one being explained.
- Dependency on Perturbation Strategy:
  The quality of LIME explanations can greatly depend on how the data is perturbed, which can introduce variability in the explanations.
- Computational Complexity:
  Generating explanations for individual predictions can be computationally expensive, especially for very large datasets or in cases where explanations are required in real time.

## 5.3 Individual Conditional Expectation (ICE)

*Main Idea:*

Individual Conditional Expectation (ICE) plots, proposed by [9], offer a detailed view on the relationship between a feature of interest and model predictions for individual instances. While Partial Dependence Plots (PDPs) provide an averaged view across the dataset, ICE plots drill down to how the model's prediction changes for an individual observation when the feature value changes, keeping all other features for that instance constant. ICE plots are particularly useful for uncovering interactions and heterogeneity of feature effects that PDPs might obscure due to averaging.

*Operational Details:*

The calculation of ICE plots involves the following process: For a specific data instance, we vary the feature of interest over its range while keeping all other features fixed at their original values. We then compute the prediction for each of these modified instances. Repeating this for the entire range of the feature's values, we trace out a line that shows how the prediction changes with the feature values for that instance. An ICE plot consists of one such line for each instance in the dataset – or for a representative sample if the dataset is very large – resulting in multiple lines overlaying each other in the plot.

*Relation to Other XAI Methods:*

ICE and PDP are interconnected techniques. While PDP represents the averaged effect of a feature on model predictions across the dataset, ICE provides a granular view by showing this relationship for each instance separately. Specifically, a PDP is essentially the average of the individual lines found in an ICE plot. This averaging process in PDPs can obscure individual variations, which ICE plots can highlight. Thus, PDPs are ideal for understanding the general, average influence of a feature, whereas ICE plots are excellent for uncovering the variability and interactions of feature effects at the instance level. Combining both methods offers a more comprehensive understanding of model behavior: PDPs revealing the global average effects and ICE plots detailing local, instance-specific effects.

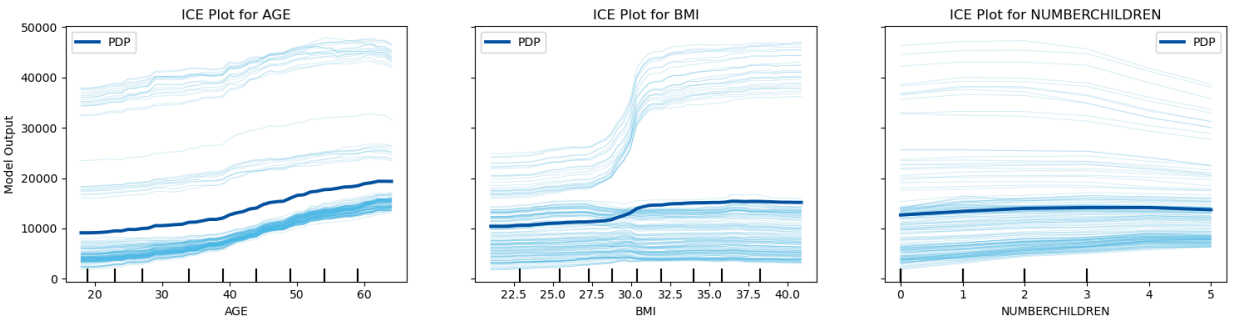
*Interpretation:*

When interpreting ICE plots, each line represents the predicted outcome for a particular instance as the value of the feature of interest varies. On the $y$-axis, we have the predicted outcome, and the $x$-axis represents the values of the feature of interest. A steep slope in an individual line suggests a strong sensitivity of the prediction to the feature for that specific instance. Diverging lines across the plot can indicate potential interactions between the variable of interest and other variables, or they might reveal that the model behaves differently for different subsets of the data. A bundle of lines moving together in a consistent fashion suggests a relatively uniform effect of the feature across the instances.

*Application to the Medical Costs Dataset:*

We demonstrate ICE plots for the numerical features in our dataset. Additionally, the respective pointwise average of the ICE plots is also displayed, which coincides with the PDP.

In [ ]:
```
# Create the ICE plot
display = PartialDependenceDisplay.from_estimator(
    model_CB_raw, X_raw_test, features=numerical_feature_positions,
    kind='both', n_cols=3, grid_resolution=50, subsample=0.4,
    pd_line_kw={'color': COLOR_DARK, 'linewidth': 2.8,
                'linestyle': '-', 'label': 'PDP'},
    ice_lines_kw={'color': COLOR_LIGHT}
)

# Set the figure size
display.figure_.set_size_inches(15, 4)

# Customize the display
for i, axi in enumerate(display.axes_.ravel()):
    axi.set_title(f'ICE Plot for {numerical_features[i]}')
    axi.set_ylabel("Model Output" if i == 0 else "")
    axi.set_xlabel(numerical_features[i])

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




The ICE plot for `AGE` displays a consistent monotonic relationship, indicating that the impact of age on the model's predictions is stable across different instances. For `BMI`, the ICE plot shows two behaviors: some instances have negligible changes in predictions with varying BMI, whereas for others, predictions increase significantly when BMI exceeds 30. This suggests an interaction effect, potentially with the `SMOKER` feature. The `NUMBERCHILDREN` plot shows no clear pattern, with the effect of the number of children on predictions varying between instances. This variability indicates complex interactions with other variables, leading to mixed effects on the model's output.

ICE plots for categorical features are omitted due to the lack of support in `PartialDependenceDisplay` and to maintain focus on key concepts.

*Advantages and Disadvantages:*

At the end of this section, we list several advantages and disadvantages of ICE:

**Advantages**

- Visualizes Individual Predictive Paths:
  Each line on an ICE plot corresponds to the predictive path of an individual instance across a range of feature values, enabling the analysis of predictions at the individual level rather than at an aggregated level.
- Detects Heterogeneity of Effects:
  ICE plots illustrate how the model's predictions change for individual observations when a feature's value is altered, thus revealing the variability of the effects of a feature across different data points.
- Reveals Feature Interactions:
  With ICE plots, one can potentially discover interactions between features since they can show how the model responds to changes in a feature for each instance, including when other features are affecting the outcome.
- Complements Aggregated Methods:
  When used alongside PDPs, ICE plots offer a more comprehensive understanding of the model by providing insight into both the average prediction and the distribution of predictions across individuals.

**Disadvantages**

- Can Be Overwhelming:
  An ICE plot with a large number of lines can be overwhelming and difficult to interpret, especially with large datasets where it becomes challenging to visually distinguish between the effects of different instances.
- Potential for Misinterpretation:
  The presence of widely divergent lines can make it difficult to draw clear conclusions about feature effects, and observers may misinterpret the variance across lines as noise rather than true heterogeneity.
- Sensitivity to Outliers:
  ICE plots can be sensitive to outliers or extreme values, which may skew the visualization and potentially lead to misinterpretation if not addressed properly.

## 6. Limitations and Outlook

As we conclude our study on applying model-agnostic explainability methods to a medical costs dataset, it is important to acknowledge the limitations of our current approach and identify directions for future research. These limitations and potential research directions are organized into three main topics: the specifics of the task and dataset, the process of machine learning model development and evaluation, and the selection of the explainability techniques we implemented.

**Limitations and Outlook regarding the Underlying Task and Dataset:**

- This notebook is dedicated to regression analysis, presenting a medical costs dataset as a representative use case. Additionally, we offer a separate notebook that addresses binary classification, and future research on multiclass classification could effectively build upon this binary classification groundwork.
- The dataset showcased in this notebook is composed of structured, tabular data. Prospective studies could extend our work by investigating XAI methods with different data formats, such as text, images, or time series. Expanding the range of data types would both challenge and potentially validate the versatility of the XAI techniques discussed.

**Limitations and Outlook regarding Machine Learning Model Development and Evaluation:**

- While this notebook did not require complex data preprocessing steps (e.g., handling of missing values, scaling, feature engineering), such operations are typically essential in modeling and can influence both results and interpretations. Further research should examine the interplay between preprocessing techniques and model understanding, especially when applied to datasets with missing values or datasets that have undergone extensive feature engineering.
- The use of machine learning pipelines was not explored here, but incorporating them in future work could standardize and expedite the model development cycle. Pipelines facilitate a seamless transition from data preprocessing to model training,

ensuring consistency and reproducibility across experiments.
- Although not addressed in this notebook for the sake of simplicity, hyperparameter tuning is a crucial aspect of model development. It would be valuable to explore the impact of optimized parameters on model performance and the subsequent effects on model explanations in subsequent work.

**Limitations and Outlook regarding Employed Explainability Methods:**

- While the current emphasis on model-agnostic methods ensures broad applicability, subsequent research could benefit from investigating model-specific explainability techniques. A combined approach integrating both types could yield a more nuanced understanding and offer richer insights into the inner workings of predictive models.
- This study focuses on some of the most established model-agnostic methods, yet a multitude of other techniques remain that could further be investigated. Additionally, the rapidly evolving domain of XAI consistently introduces new methods, presenting a continuous opportunity to enhance our understanding of complex models.
- The robustness of XAI methods' results has not been extensively tested in this notebook. Replicating the analyses with varying random seeds, multiple train-test splits, and a range of hyperparameter settings for XAI techniques would be instrumental in assessing the stability and reliability of the explanatory outcomes derived.

# A. Appendix

## A.1 Adapting this Notebook's Code to Other Machine Learning Models

We chose CatBoost as our primary machine learning model due to its strong predictive capabilities and native support for categorical features, which makes it an ideal choice for datasets with mixed data types. However, when applying different machine learning models that do not natively handle categorical features, adjustments to the preprocessing steps are necessary.

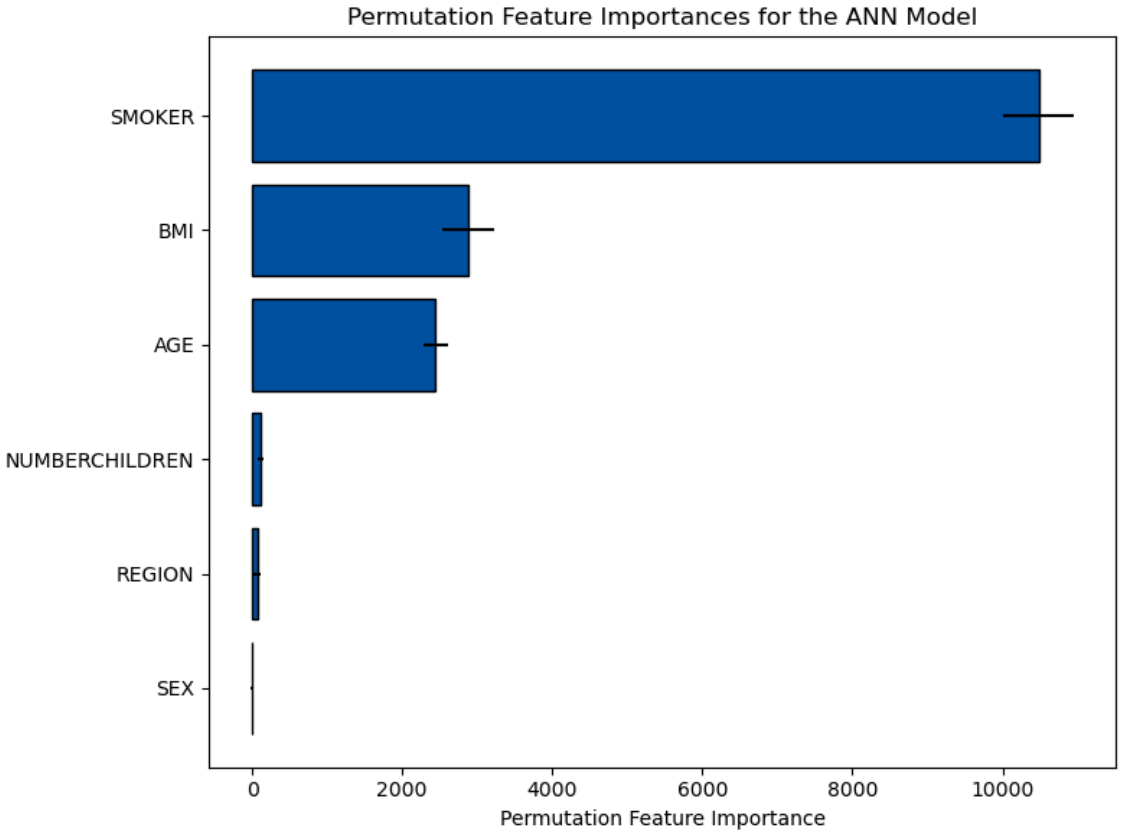
Below, we illustrate how to adapt the code for an artificial neural network using the `MLPRegressor` implementation from the scikit-learn framework. Specifically, we use a suitable pipeline that includes a preprocessing step to scale the numerical features and handle the categorical data before fitting the machine learning model. The preprocessing step involves using a `ColumnTransformer` to apply one-hot encoding to the categorical features while passing through the numerical features unchanged.

In [ ]:
```
# Create the preprocessing steps for both numeric and categorical data
preprocessor = ColumnTransformer(
    transformers=[
        ('num', StandardScaler(), numerical_features),
        ('cat', OneHotEncoder(handle_unknown='ignore'), categorical_features)
    ]
)

# Create an artificial neural network via MLPRegressor
model_ANN = MLPRegressor(
    hidden_layer_sizes=(100), learning_rate_init=0.05,
    random_state=RANDOM_SEED, max_iter=1000
)

# Create the pipeline
pipeline = Pipeline(
    steps=[('preprocessor', preprocessor),
           ('model', model_ANN)]
)

# Fit the pipeline
pipeline.fit(X_raw_train, y_train)

# Evaluate the model on the test set using RMSE and R² score
rmse_ANN_raw = root_mean_squared_error(y_test, pipeline.predict(X_raw_test))
r2_ANN_raw = r2_score(y_test, pipeline.predict(X_raw_test))

# Print the RMSE and R² score
print(f"RMSE for the ANN model on the test data:     {rmse_ANN_raw:.2f}")
print(f"R² score for the ANN model on the test data:    {r2_ANN_raw:.2f}")
```

```
RMSE for the ANN model on the test data:     4769.36
R² score for the ANN model on the test data:    0.84
```

The pipeline approach described above can be used for many of the XAI methods discussed in this notebook. However, it's important to note that certain methods may require additional modifications or may not be directly applicable. To demonstrate the exemplary use of the pipeline with a different machine learning model, we reimplemented the Permutation Feature Importance method, using the artificial neural network pipeline instead of the CatBoost model.

In [ ]:
```
# Perform permutation importance
results = permutation_importance(
    pipeline, X_raw_test, y_test,
    scoring='neg_root_mean_squared_error',
    n_repeats=1000,
    random_state=RANDOM_SEED
)

# Sort feature importances in descending order and get their indices
sorted_idx = results.importances_mean.argsort()

# Get the standard deviations for the sorted features
sorted_std = results.importances_std[sorted_idx]

# Plot the feature importances
plt.figure(figsize=(8, 6))
plt.barh(
    range(len(sorted_idx)),
    results.importances_mean[sorted_idx],
    color=COLOR_DARK,
    edgecolor='black',
    xerr=sorted_std
)
plt.yticks(
    range(len(sorted_idx)),
    [X_raw_test.columns[i] for i in sorted_idx]
)
plt.xlabel("Permutation Feature Importance")
plt.title("Permutation Feature Importances for the ANN Model")
plt.tight_layout()
plt.show()
```




## A.2 Deep Dive: PDP vs. ALE

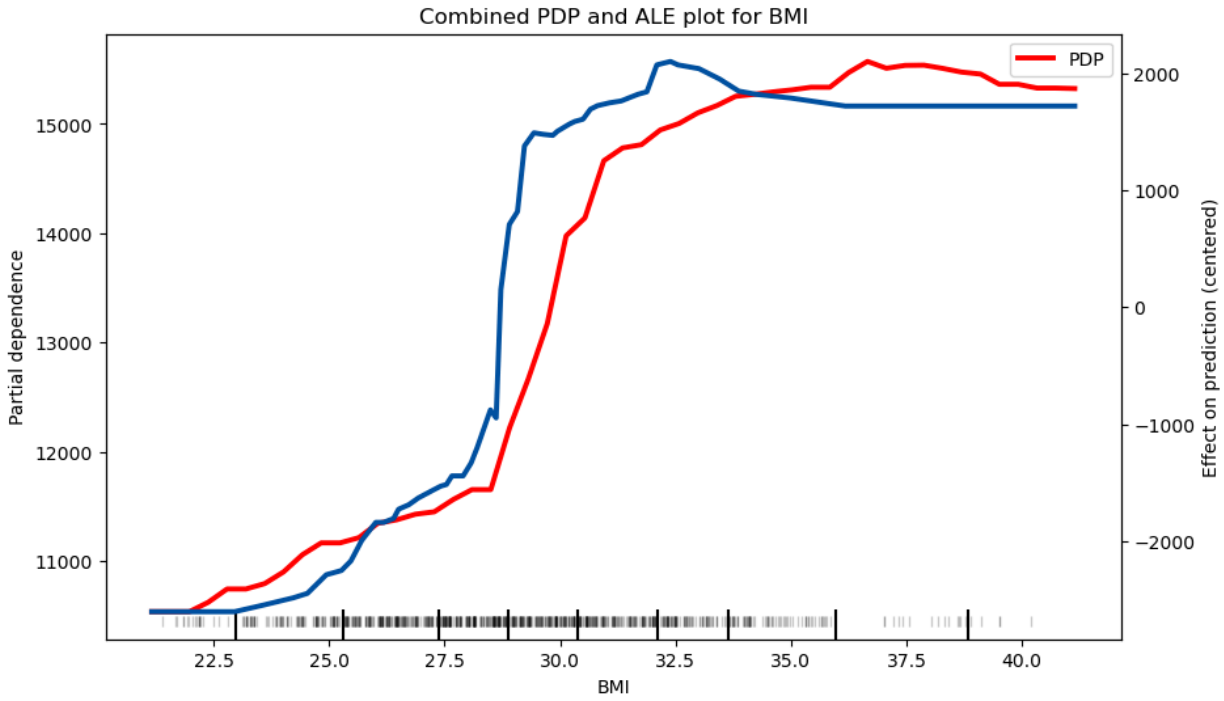
In the actuarial diligence note in Subsection 4.2, we discussed the importance of considering feature independence when applying Partial Dependence Plots and the advantages of using Accumulated Local Effects plots when dealing with correlated features. PDPs assume feature independence, which is often not the case in real-world datasets, potentially leading to misleading interpretations. ALE plots, on the other hand, do not rely on this assumption and adjust for feature correlations, providing a more accurate view of the data's structure.

Although PDPs and ALE plots can sometimes produce similar visualizations, ALE plots are generally more reliable when feature dependencies are present. This is particularly relevant for our medical costs dataset, where feature independence is not present, as

indicated by the correlations discussed in Section 2. For example, the weak correlation between `AGE` and `BMI` suggests that these features are dependent.

In the following, we present a detailed comparison of PDP and ALE plots, exemplarily using the feature `BMI`. This comparison highlights the differences between the results of these two techniques and underscores the importance of considering feature dependencies in model interpretation. Below, we provide the code and resulting plots to visualize these differences.

In [ ]:
```
# Set the feature to be considered
feature_name = 'BMI'

# Create the figure for the combined plots
fig, ax1 = plt.subplots(figsize=(10, 6))

# Generate PDP with PartialDependenceDisplay.from_estimator
pdp_display = PartialDependenceDisplay.from_estimator(
    model_CB_raw,
    X_raw_train,
    features=[X_raw_test.columns.get_loc(name) for name in [feature_name]],
    kind='average',
    grid_resolution=50,
    feature_names=numerical_features,
    pd_line_kw={'color': 'red', 'linewidth': 2.8, 'linestyle': '-', 'label': 'PDP'},
    ax=ax1
)

pdp_display.axes_[0][0].set_xlabel(feature_name)

ax2 = ax1.twinx()

ale_plot = ale(
    X=X_raw_train,
    model=model_CB_raw,
    feature=[feature_name],
    feature_type='continuous',
    grid_size=50,
    include_CI=False,
    fig=fig,
    ax=ax2
)

# Adjust line style and color
ax2.lines[0].set_color(COLOR_DARK)
ax2.lines[0].set_linewidth(2.8)
ax2.lines[0].set_label('ALE')

# Set title for each subplot
ax2.set_title("")

# Add labels and legend to the original figure
ax1.set_title(f'Combined PDP and ALE plot for {feature_name}')
ax1.set_ylabel('Effect on model output')

plt.show()
```




In the combined PDP (red line) and ALE (blue line) plot, we observe that ALE shows a steeper increase at around a `BMI` value of 30 and a decline after `BMI` value 32, while the PDP grows quasi-monotonically. This occurs because ALE adjusts for feature dependencies, capturing more nuanced interactions, while PDP assumes feature independence, smoothing out such variations.

In summary, while ALE plots cope better with feature dependencies and provide a more accurate representation of feature effects, it is advisable to examine both PDP and ALE plots to gain a comprehensive understanding of model behavior.

## A.3 Deep Dive: Diverse Feature Importance Methods

In this notebook, we have explored two model-agnostic methods for calculating feature importances: Permutation Feature Importance, which we discussed in Subsection 4.4, and feature importance based on mean absolute SHAP values derived in Subsection 5.1. Comparing the outcomes of these methods is crucial, as it can lend credibility to the results by demonstrating their robustness. This section is dedicated to the comparison of the two mentioned model-agnostic feature importance methods as well as a model-specific technique, namely CatBoost's internal feature importance. More details on how CatBoost calculates its internal feature importance can be found in [10].

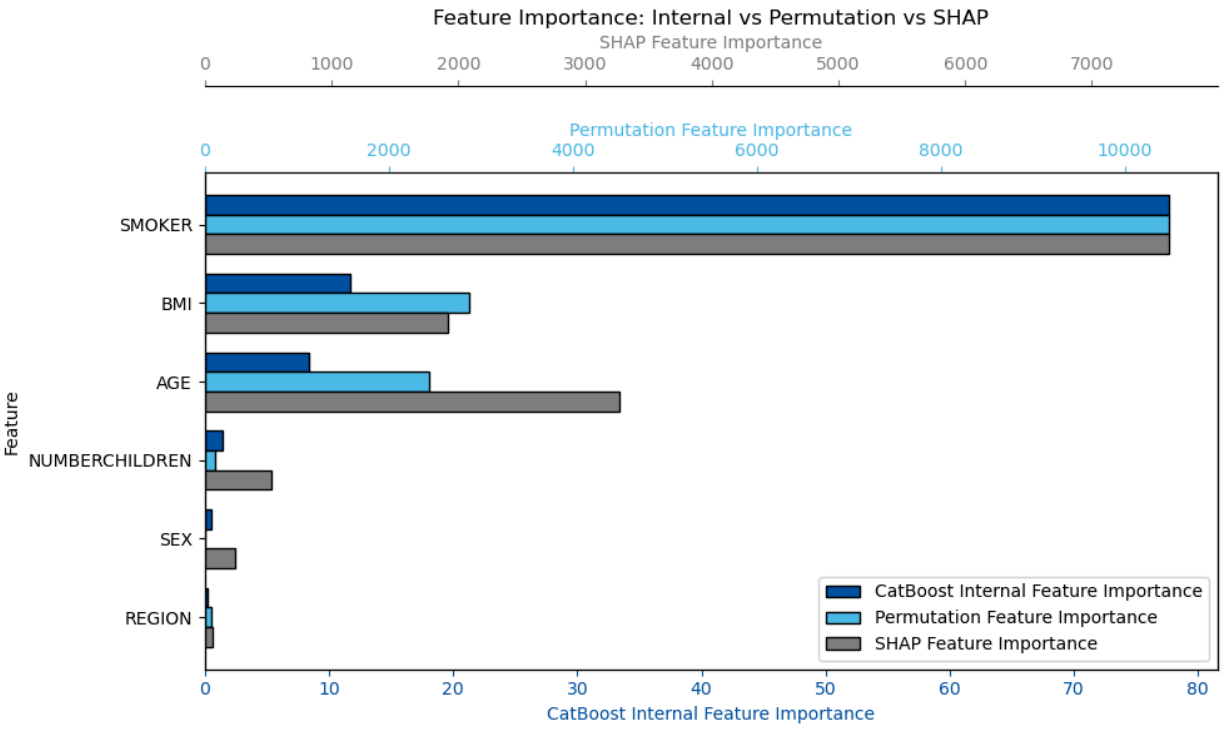
There is no universally perfect method for calculating and interpreting feature importances. However, we will present two visualizations to help assess these values. Prior to this, we calculate the CatBoost internal feature importances and store them, along with the feature importances obtained from PFI and SHAP, in a single data frame.

In [ ]:
```
# Calculate feature importances
internal_feature_importances = model_CB_raw.get_feature_importance()

# Create a DataFrame for all feature importances
feature_importance_df = pd.DataFrame({
    'Feature': features_X_raw,
    'Internal_Importance': internal_feature_importances,
    'PFI_Mean': results.importances_mean,
    'SHAP_Importance': mean_abs_shap_values
}).sort_values(by='Internal_Importance', ascending=False)
```

First, we will use a grouped bar chart to display the feature importance values of the different methods. Note that these values often have different scales, so we have adjusted them to ensure comparability.

In [ ]:
```
# Plot the feature importances
fig, ax1 = plt.subplots(figsize=(10, 6))

# Calculate bar width and positions
bar_width = 0.25
r1 = np.arange(len(feature_importance_df))
r2 = [x + bar_width for x in r1]
r3 = [x + bar_width for x in r2]

# Plot internal CatBoost feature importances
bars1 = ax1.barh(r1, feature_importance_df['Internal_Importance'],
```

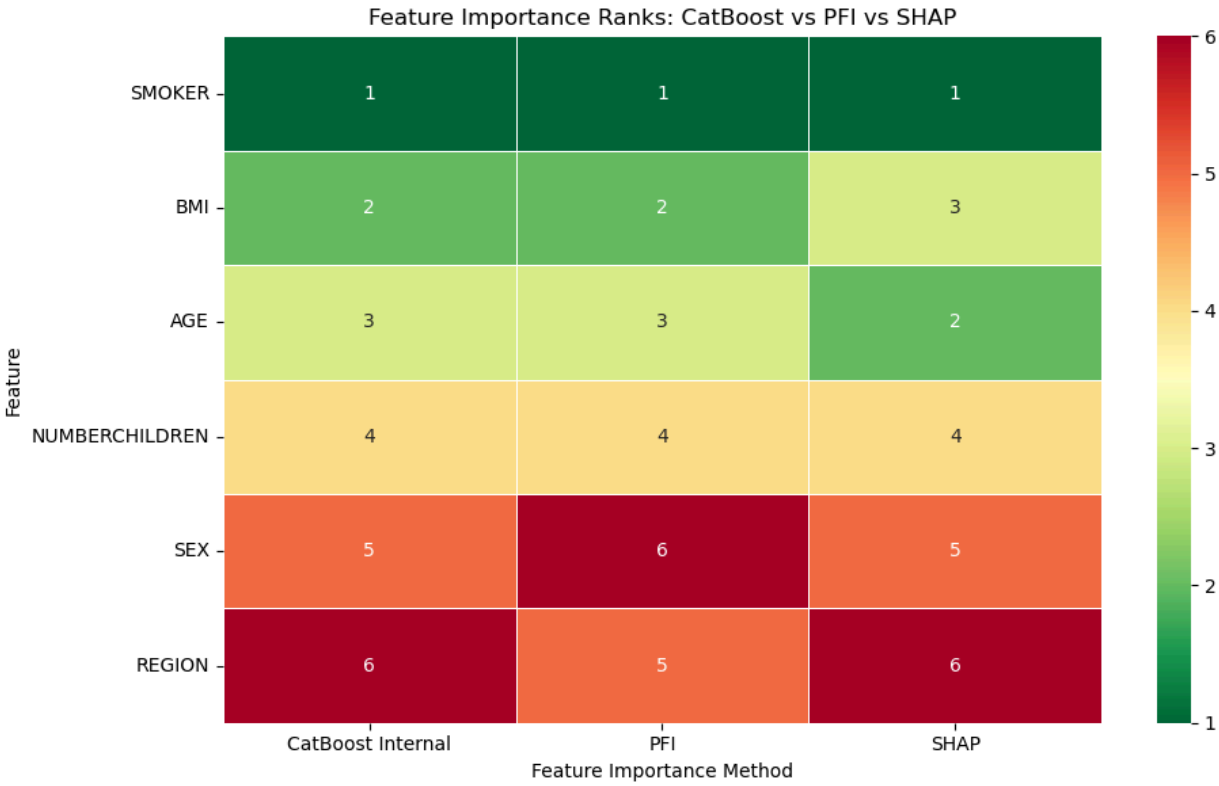
```
                height=bar_width, color=COLOR_DARK, edgecolor='black',
                label='CatBoost Internal Feature Importance')

# Create a second x-axis for permutation feature importances
ax2 = ax1.twiny()
bars2 = ax2.barh(r2, feature_importance_df['PFI_Mean'],
                height=bar_width, color=COLOR_LIGHT, edgecolor='black',
                label='Permutation Feature Importance')

# Create a third x-axis for SHAP feature importances
ax3 = ax1.twiny()
bars3 = ax3.barh(r3, feature_importance_df['SHAP_Importance'],
                height=bar_width, color=COLOR_GREY, edgecolor='black',
                label='SHAP Feature Importance')

# Set labels for the first axis
ax1.set_yticks([r + bar_width for r in range(len(feature_importance_df))])
ax1.set_yticklabels(feature_importance_df['Feature'])
ax1.set_xlabel('CatBoost Internal Feature Importance', color=COLOR_DARK)
ax1.tick_params(axis='x', colors=COLOR_DARK)
ax1.set_ylabel('Feature')
ax1.invert_yaxis()

# Set labels for the second axis
ax2.set_xlabel('Permutation Feature Importance', color=COLOR_LIGHT)
ax2.tick_params(axis='x', colors=COLOR_LIGHT)

# Set labels for the third axis
ax3.spines['top'].set_position(('outward', 50))  # Move the third axis down
ax3.set_xlabel('SHAP Feature Importance', color=COLOR_GREY)
ax3.tick_params(axis='x', colors=COLOR_GREY)

# Align the x-axis at 0 for all axes
ax1.set_xlim(left=0)
ax2.set_xlim(left=0)
ax3.set_xlim(left=0)

# Title and legend
plt.title('Feature Importance: Internal vs Permutation vs SHAP')
ax1.legend(handles=[bars1, bars2, bars3], loc='lower right')

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




The benefit of this plot is that it allows comparison of feature ranks across methods and highlights variations in feature importance strengths. Additionally, these differences can be analyzed across all methods. However, a drawback of this visualization is the inherent scale differences among methods; for instance, the CatBoost internal feature importances sum up to one, while permutation feature importances are linked to RMSE reduction, making direct comparisons challenging. In our grouped bar chart, `SMOKER` emerges as consistently the most relevant feature across all methods, followed by `BMI` and `AGE`. Conversely, `NUMBERCHILDREN`, `SEX`, and `REGION` exhibit negligible to low feature importances.

Next, we will rank the feature importances for each method and visualize them using a heatmap. This will enable us to identify both similarities and differences in the rankings of the methods.

In [ ]:
```
# Calculate ranks for each method
feature_importance_df['CatBoost Internal'] = \
    feature_importance_df['Internal_Importance'].rank(ascending=False)
feature_importance_df['PFI'] = \
    feature_importance_df['PFI_Mean'].rank(ascending=False)
feature_importance_df['SHAP'] = \
    feature_importance_df['SHAP_Importance'].rank(ascending=False)

# Plotting the heatmap
plt.figure(figsize=(10, 6))
sns.heatmap(feature_importance_df.set_index('Feature')[['CatBoost Internal',
                                                        'PFI', 'SHAP']],
            annot=True, cmap='RdYlGn_r', linewidths=.5)
plt.title('Feature Importance Ranks: CatBoost vs PFI vs SHAP')
plt.xlabel('Feature Importance Method')
plt.ylabel('Feature')
plt.yticks(rotation=0)
plt.tight_layout()
plt.show()
```




Examining the heatmap of feature ranks across different methods reveals a consistent pattern in feature importance values, with minor discrepancies noticeable between methods. Across all methods, `SMOKER` consistently ranks highest, followed by `BMI` and `AGE` in CatBoost's internal feature importance and PFI. For the SHAP-based feature importance, the order of `AGE` and `BMI` is reversed. Additionally, the ranks for `SEX` and `REGION` exhibit slight variations across methods.

In summary, it is wise not to rely solely on one method for determining feature importance values, but instead to consider multiple methods and possibly reassess them to ensure robustness.

## A.4 Deep Dive: Variants of SHAP

In Subsection 5.1, we introduced and discussed the XAI method SHAP, noting different methods for calculating Shapley values in an actuarial diligence note. Various SHAP variants address different scenarios and model types, each with unique benefits and computational considerations. Here, we focus on two common variants: *Kernel SHAP* and *Tree SHAP*, while acknowledging that many other variants are available for specific needs. For more detailed analyses and comparisons, we refer the interested reader to [11].

- *Kernel SHAP:* This model-agnostic variant can be applied to any machine learning model. It uses a weighted linear regression approach to approximate the Shapley values. *Kernel SHAP* is particularly useful for models without a specific SHAP implementation.

- *Tree SHAP:* Designed specifically for tree-based models, *Tree SHAP* leverages the structure of decision trees to compute Shapley values efficiently. It is significantly faster for tree-based models compared to *Kernel SHAP*.

Differences in SHAP value calculations arise from the underlying algorithms and approximations used by each variant. *Kernel SHAP* treats the model as a black-box, offering flexibility but at a high computational cost. In contrast, *Tree SHAP* exploits the internal structure of tree-based models, resulting in more efficient and precise calculations.

Other variants, such as Sampling SHAP and Permutation SHAP, offer increased flexibility for different use-cases and computational constraints, though they are not the focus of this section.

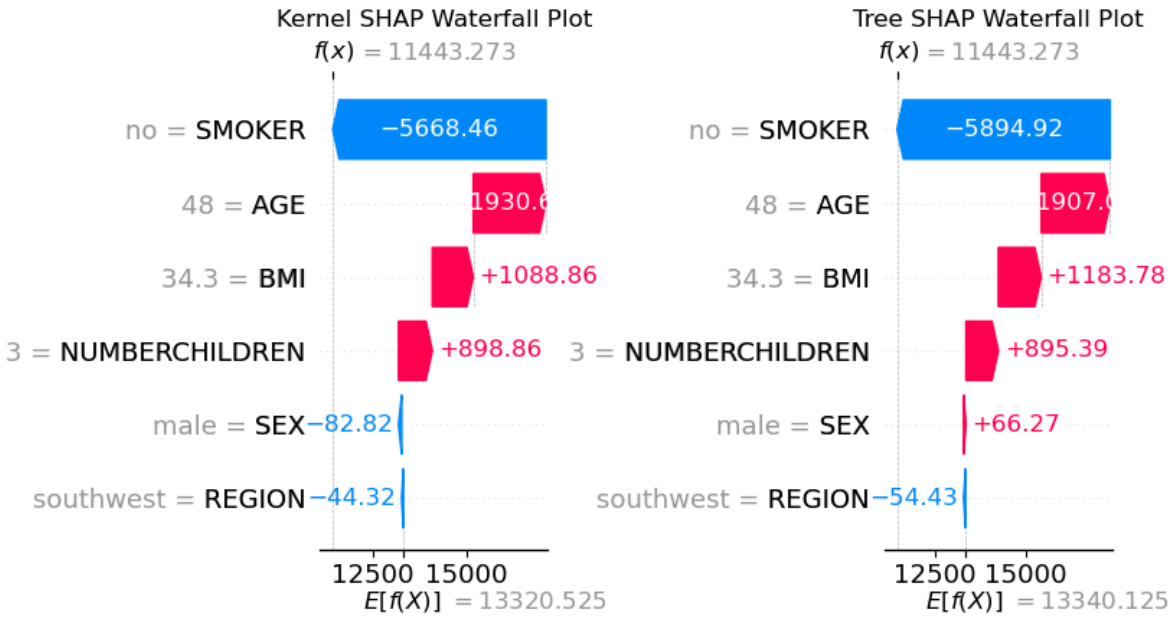
Below, we demonstrate how to create SHAP explainers using *Kernel SHAP* and *Tree SHAP*, and compute SHAP values for our CatBoost model.

In [ ]:
```
# Create various SHAP explainers using the training set
expl_kernel = shap.KernelExplainer(model_CB_raw.predict, X_raw_train)
expl_tree   = shap.TreeExplainer(model_CB_raw)

# Calculate SHAP values for the test set
shap_values_kernel = expl_kernel.shap_values(X_raw_test)
shap_values_tree   = expl_tree.shap_values(X_raw_test)
```

```
shap:WARNING: Using 936 background data samples could cause slower run times. Consider using shap.sample(data, K) or sh
ap.kmeans(data, K) to summarize the background as K samples.
  0%|          | 0/402 [00:00<?, ?it/s]
```

To visualize the SHAP values for a specific instance, we create SHAP explanation objects and generate waterfall plots, similar to those discussed in Subsection 5.1. The visualizations reveal slight differences in SHAP values produced by each variant.

In [ ]:
```
# Create an Explanation object for the instance of interest using KernelExplainer
explan_kernel = shap.Explanation(
    values=shap_values_kernel[observation_index],
    base_values=expl_kernel.expected_value,
    data=X_raw_test.iloc[observation_index]
)

# Create an Explanation object for the instance of interest using TreeExplainer
explan_tree = shap.Explanation(
    values=shap_values_tree[observation_index],
    base_values=expl_tree.expected_value,
    data=X_raw_test.iloc[observation_index]
)

# Plot side-by-side waterfall plots
fig, axes = plt.subplots(ncols=2)

# Temporarily plot to get artists
plt.sca(axes[0])
shap.waterfall_plot(explan_kernel, show=False)
axes[0].set_title('Kernel SHAP Waterfall Plot')

plt.sca(axes[1])
shap.waterfall_plot(explan_tree, show=False)
axes[1].set_title('Tree SHAP Waterfall Plot')

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




Note that while the direction and strength of the SHAP values are quite similar, there are slight variations in the individual values and the base value.

## References


[1] https://www.kaggle.com/datasets/mirichoi0218/insurance

[2] Prokhorenkova, L., Gusev, G., Vorobev, A., Dorogush, A. V., & Gulin, A. (2018). CatBoost: unbiased boosting with categorical features. *Advances in Neural Information Processing Systems*, 31, 6639–6649.

[3] Friedman, J. H. (2001). Greedy Function Approximation: A Gradient Boosting Machine. *Annals of Statistics*, 29(5), 1189–1232.

[4] Molnar, C. (2022). Interpretable Machine Learning: *A Guide for Making Black Box Models Explainable* (2nd ed.). Retrieved from https://christophm.github.io/interpretable-ml-book/

[5] Apley, D. W., & Zhu, J. (2020). Visualizing the Effects of Predictor Variables in Black Box Supervised Learning Models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 82(4), 1059–1086.

[6] Breiman, L. (2001). Random Forests. *Machine Learning*, 45(1), 5–32.

[7] Lundberg, S. M., & Lee, S.-I. (2017). A Unified Approach to Interpreting Model Predictions. *Advances in Neural Information Processing Systems 30 (NeurIPS 2017)*, 4765–4774.

[8] Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16)*, 1135–1144.

[9] Goldstein, A., Kapelner, A., Bleich, J., & Pitkin, E. (2015). Peeking Inside the Black Box: Visualizing Statistical Learning With Plots of Individual Conditional Expectation. *Journal of Computational and Graphical Statistics*, 24(1), 44–65.

[10] https://catboost.ai/en/docs/concepts/fstr#internal-feature-importance

[11] Chen, H., Covert, I. C., Lundberg, S. M., & Lee, S.-I. (2022). Algorithms to estimate Shapley value feature attributions. *arXiv preprint arXiv:2207.07605*.