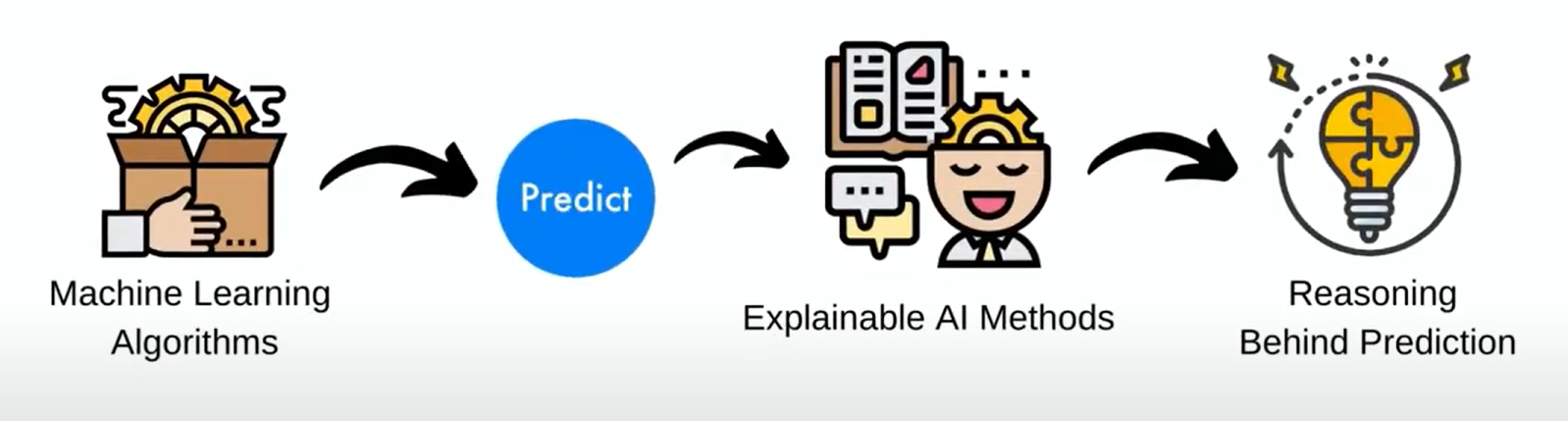
**Tools and techniques for explainability**

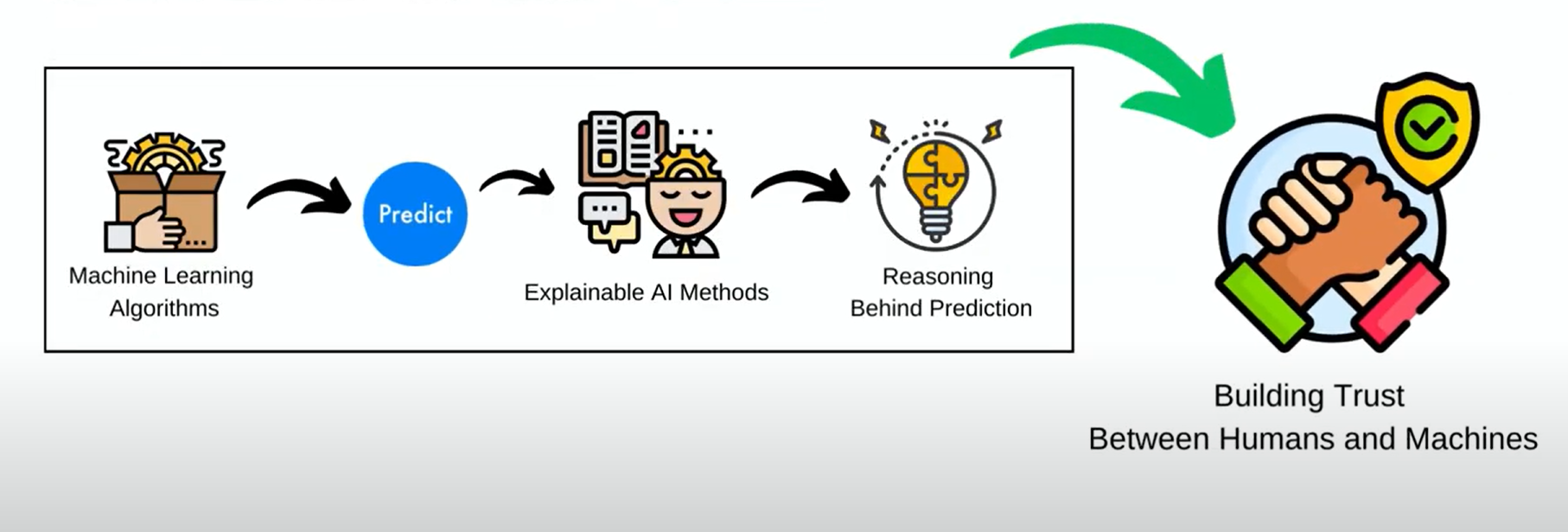
**Explainability**

Explainability in machine learning is the ability to understand and explain how a model makes its decisions or predictions. It helps users and developers see why and how a model arrived at a particular outcome, making the model’s behavior more transparent and trustworthy.

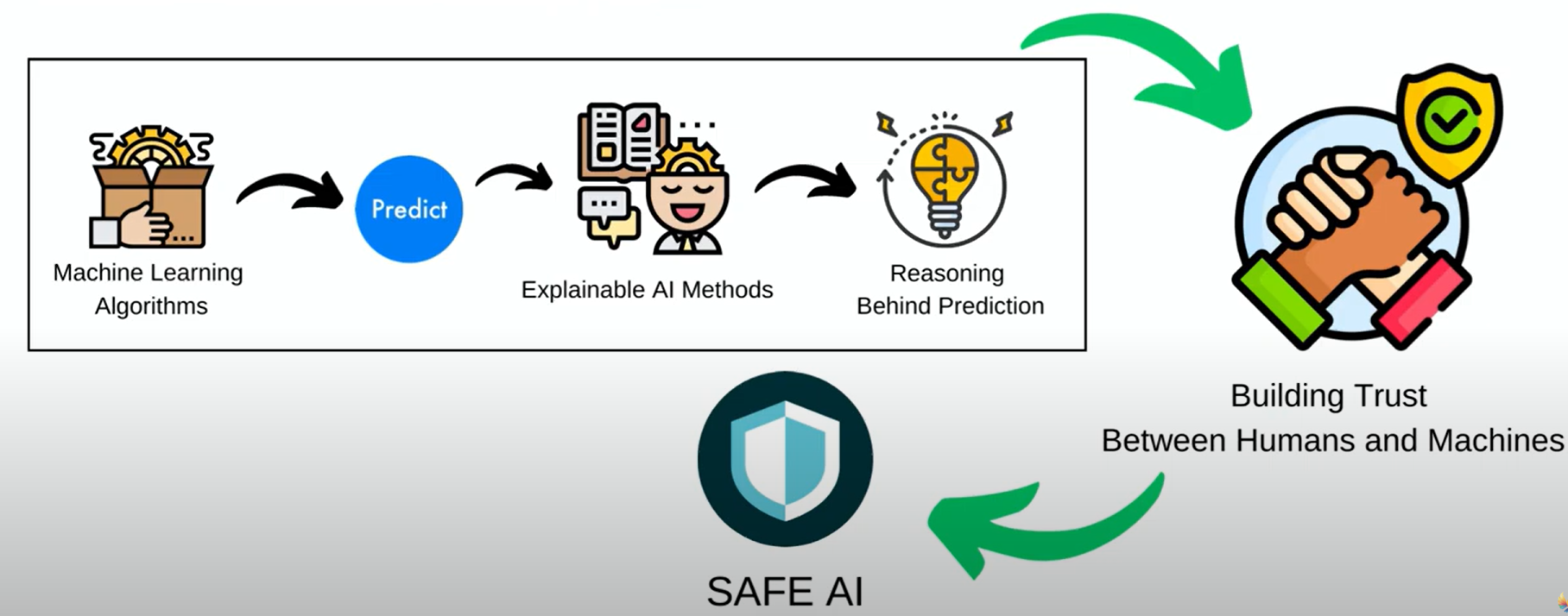


**Why Explainability is Important**

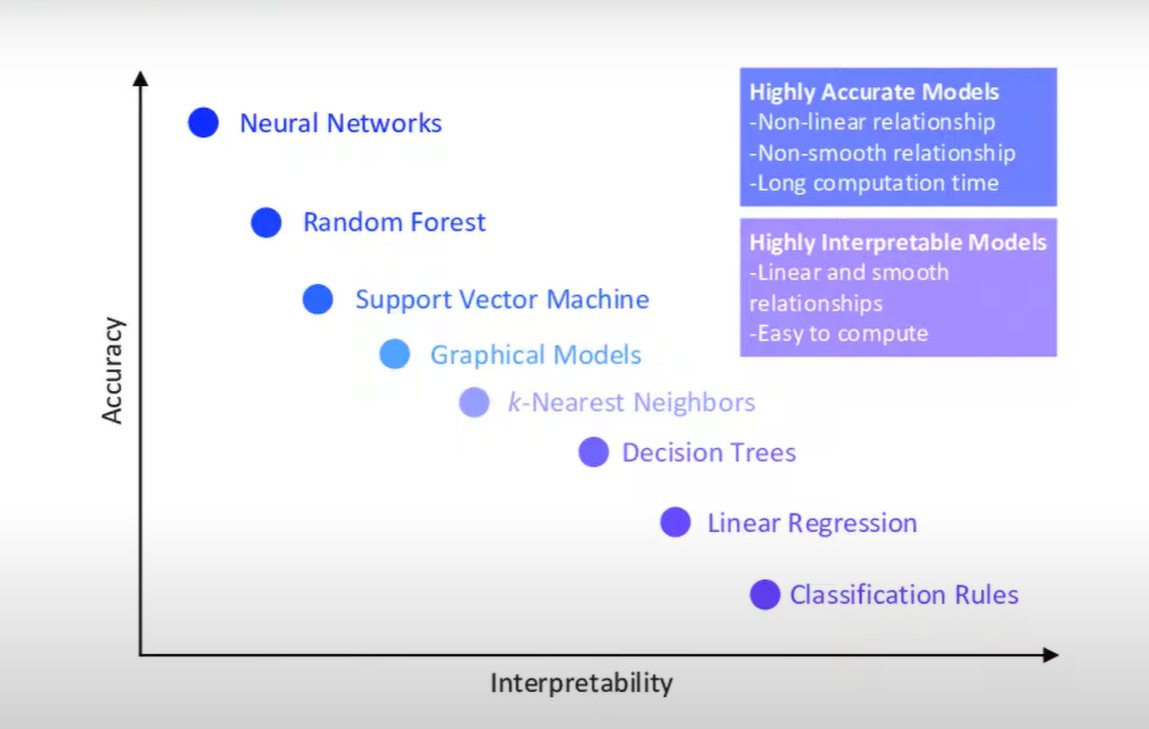
1. **Trust**: Users are more likely to trust a machine learning model if they can understand how and why it makes certain decisions. This is especially true when the model is used for critical decisions like loan approvals, medical diagnoses, or criminal sentencing.

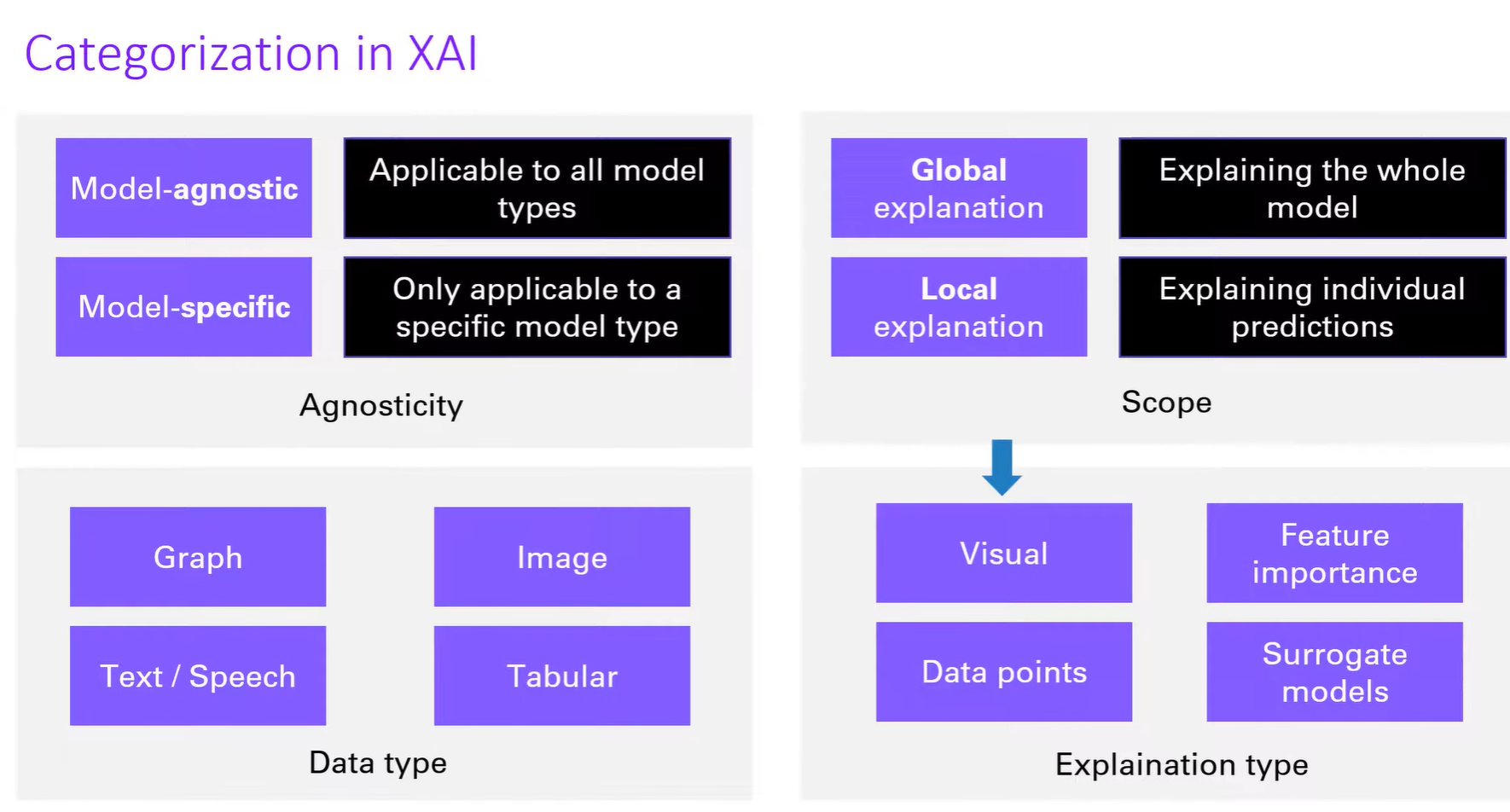


1. **Debugging and Model Improvement**: Explainability helps data scientists and engineers understand why a model behaves the way it does. This can highlight issues like overfitting, bias, or model errors that need to be corrected.



1. **Ethical Concerns**: The ability to explain decisions is a key part of creating ethical AI. Without explainability, machine learning models could perpetuate or exacerbate existing biases, leading to unfair treatment of certain groups of people.



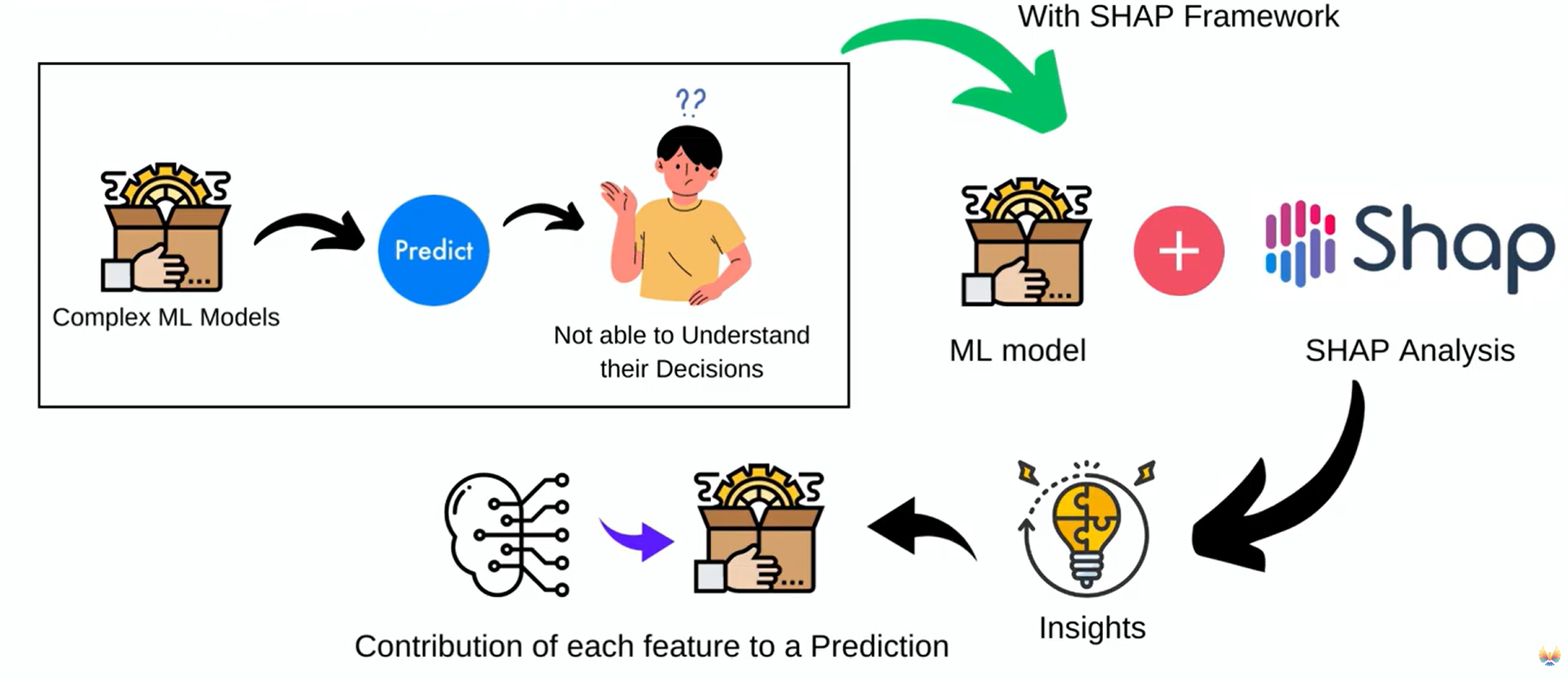


**Tools and techniques used to achieve explainability in machine learning:**

SHAP[Shapley Additive exPlanations]

SHAP (SHapley Additive exPlanations) is a popular method in machine learning for explainability. It provides a way to understand the contribution of each feature to a model's predictions, offering insights into why a model made a specific decision.

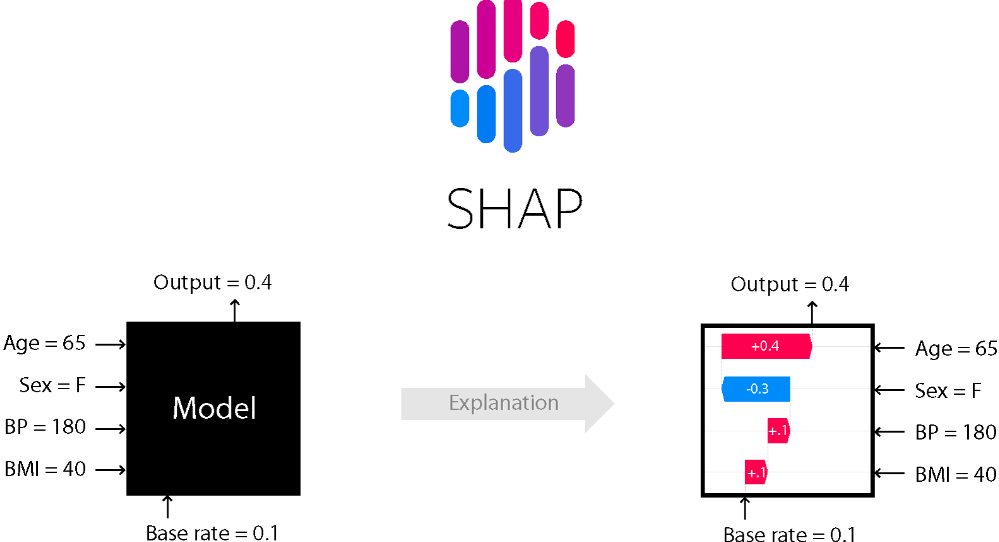
It is a powerful python package used in explainability.



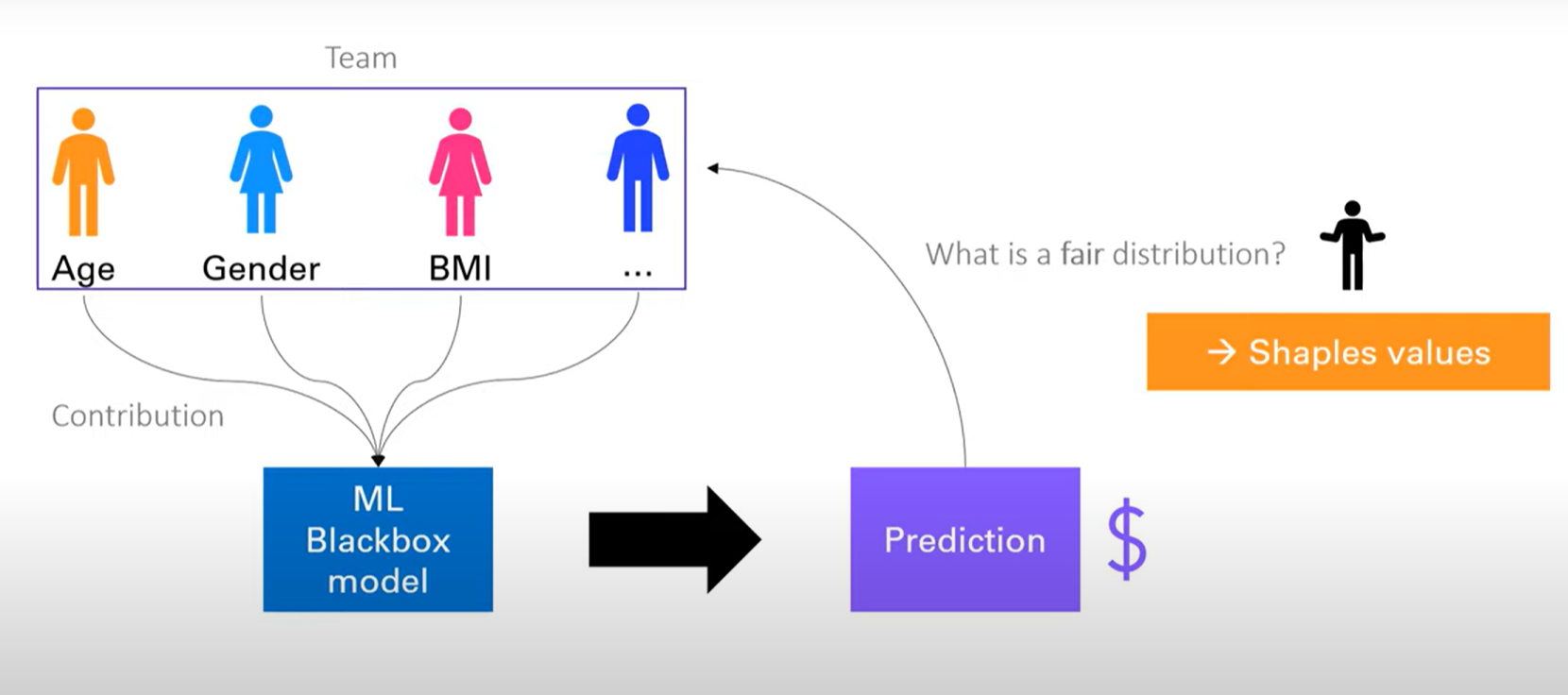
**Scenario: Predicting Diabetes Based on Clinical Data**

**Context**: A healthcare clinic is trying to identify patients at high risk of developing diabetes based on common clinical factors. The clinic has gathered data on several key variables for each patient, including:

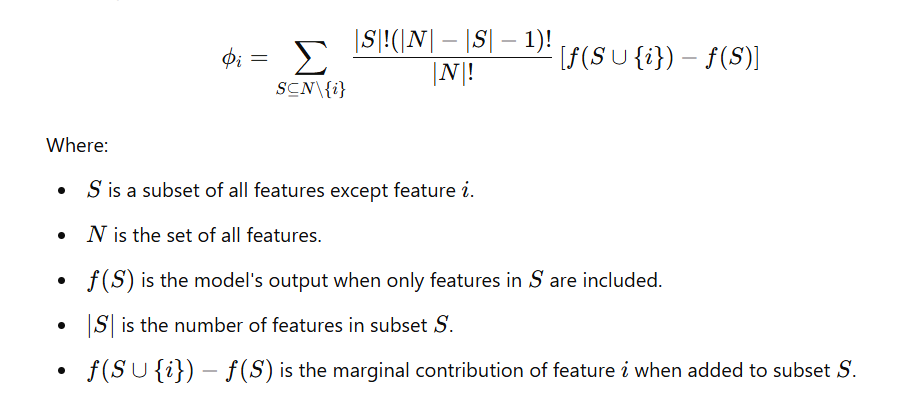
* **Age**: Older patients are more prone to developing diabetes due to changes in metabolism and lifestyle.
* **Sex**: Males and females may exhibit different diabetes risk factors due to biological and lifestyle differences.
* **Blood Pressure (BP)**: High blood pressure is often associated with insulin resistance, a precursor to diabetes.
* **BMI**: A higher BMI often indicates overweight or obesity, both of which are strong predictors of diabetes.



**Shap value is calculated**

****

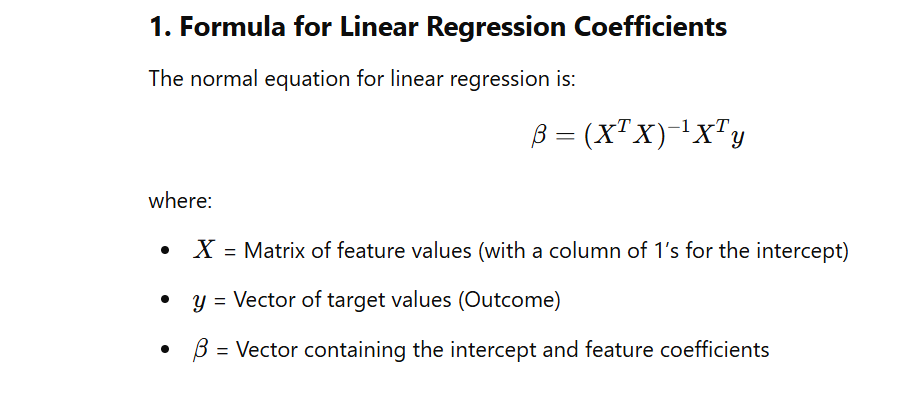
**Formula for shap value:**

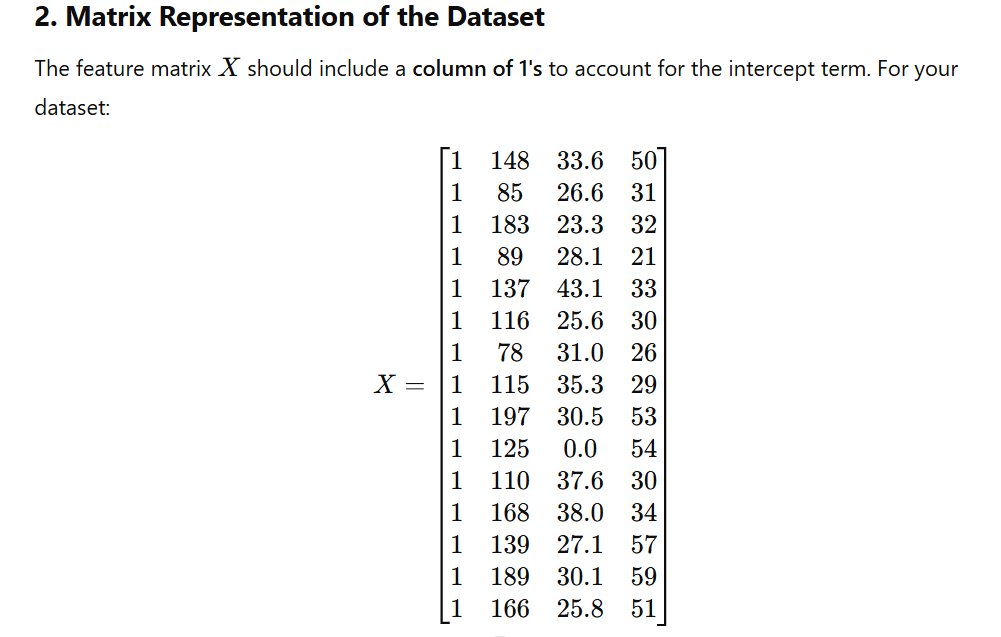
****

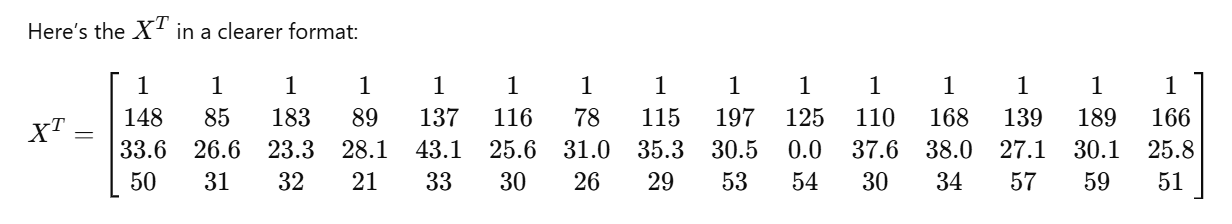
**Example :**

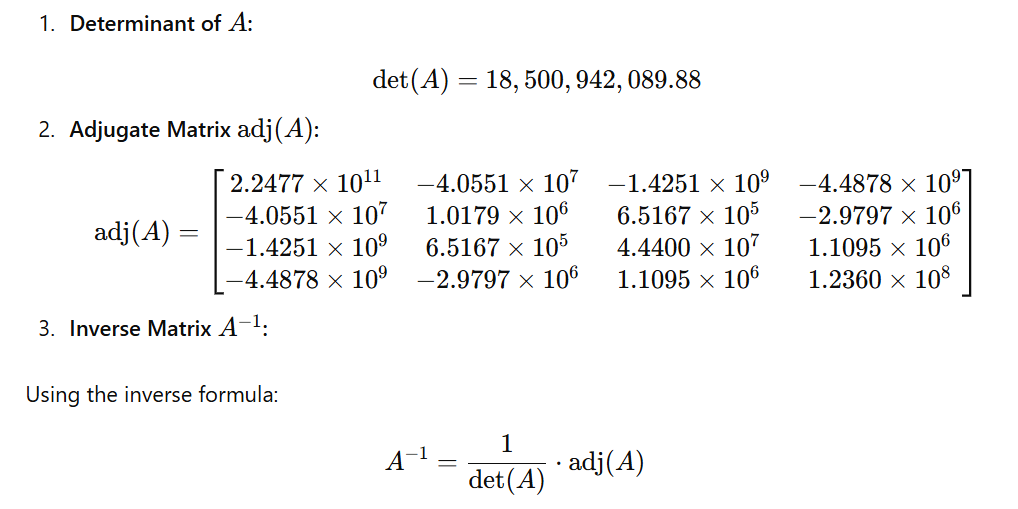
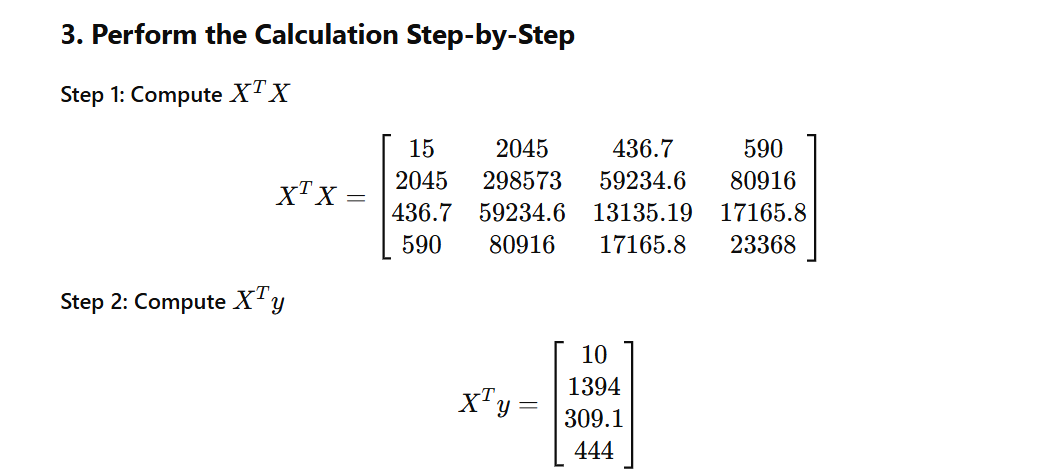
Let us take simple dataset to understand the shap value :

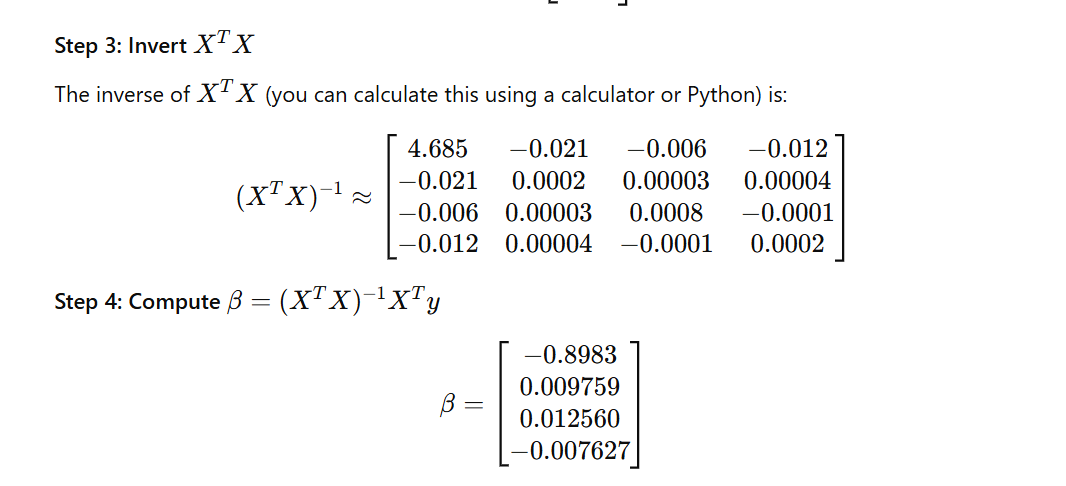
|  |  |  |  |
| --- | --- | --- | --- |
| Glucose | BMI | Age | Outcome |
| 148 | 33.6 | 50 | 1 |
| 85 | 26.6 | 31 | 0 |
| 183 | 23.3 | 32 | 1 |
| 89 | 28.1 | 21 | 0 |
| 137 | 43.1 | 33 | 1 |
| 116 | 25.6 | 30 | 0 |
| 78 | 31 | 26 | 1 |
| 115 | 35.3 | 29 | 0 |
| 197 | 30.5 | 53 | 1 |
| 125 | 0 | 54 | 1 |
| 110 | 37.6 | 30 | 0 |
| 168 | 38 | 34 | 1 |
| 139 | 27.1 | 57 | 0 |
| 189 | 30.1 | 59 | 1 |
| 166 | 25.8 | 51 | 1 |
|  |  |  |  |











**Model Information**

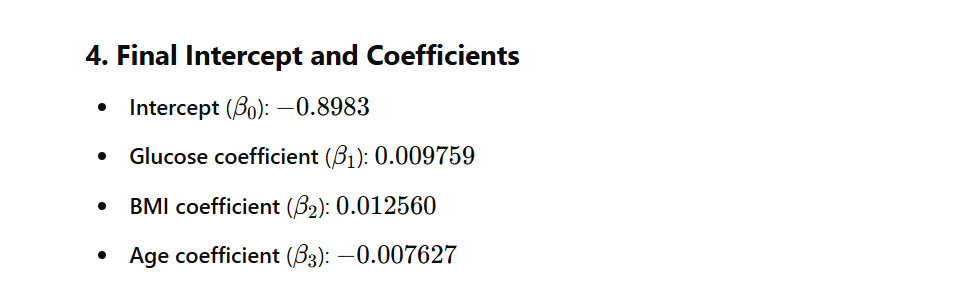
* Intercept: -0.8982820516360828
* Coefficients:
  + Glucose: 0.009759
  + BMI: 0.012560
  + Age: -0.007627

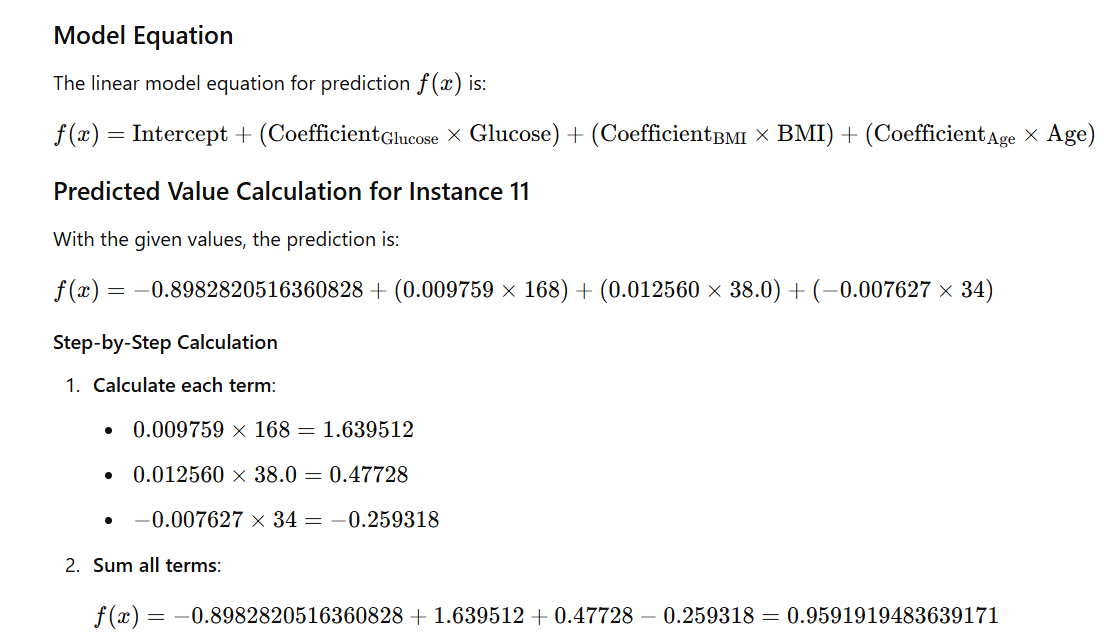
Input Features for Instance 11

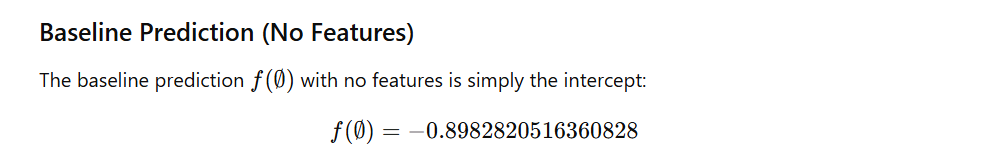
* Glucose = 168
* BMI = 38.0
* Age = 34

Model Equation

The linear model equation for prediction f(x)is:



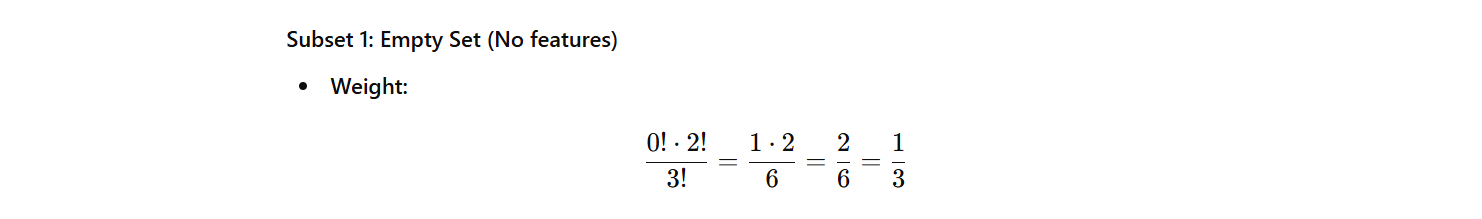
****

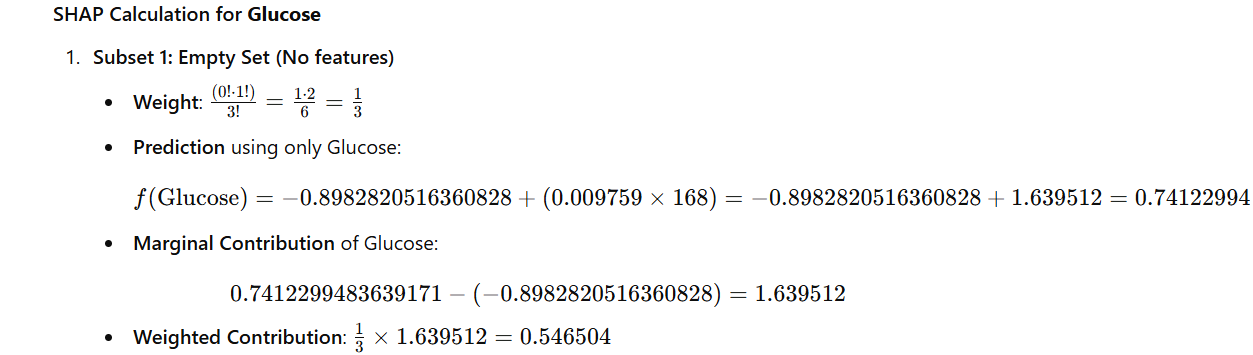


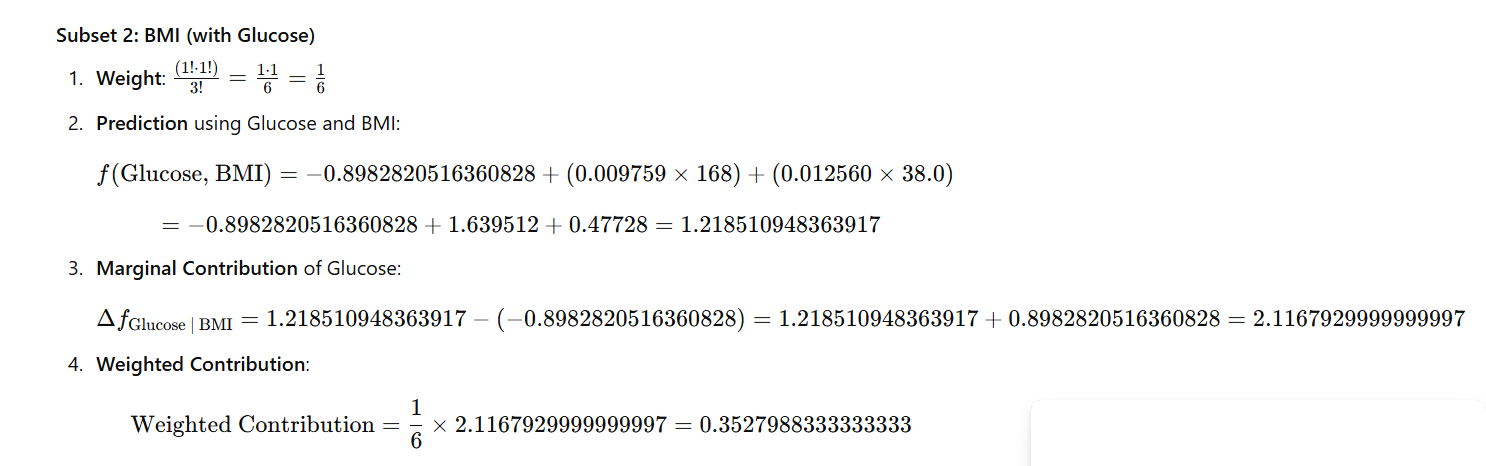
**SHAP Calculation for Each Feature using Kernal explainer.**

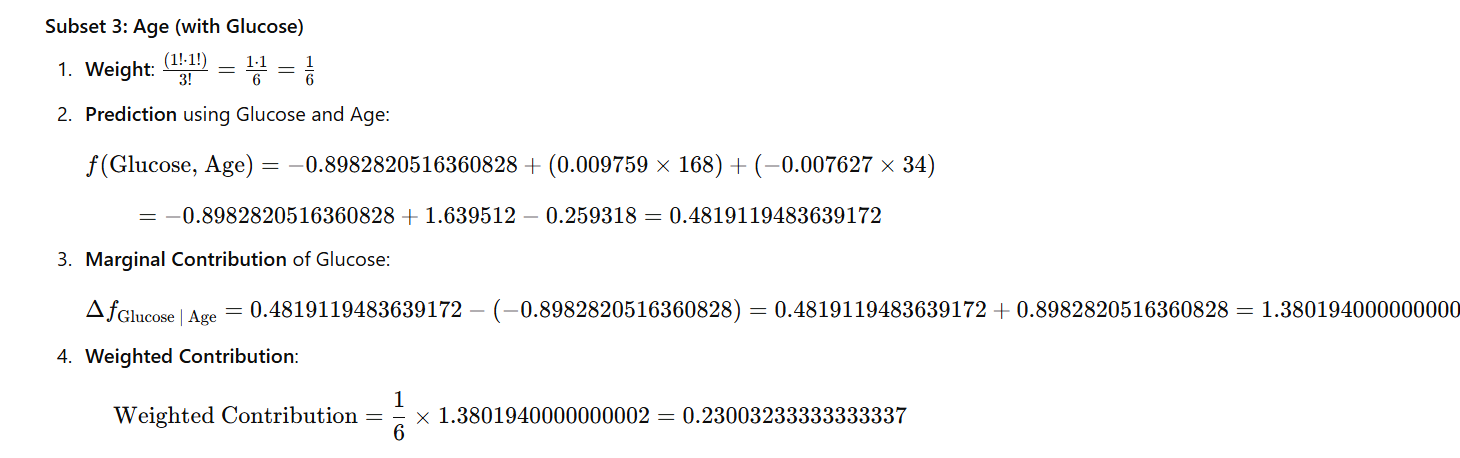
To calculate SHAP values for each feature (Glucose, BMI, and Age), we consider every subset of the other features, compute the marginal contribution of each subset, and use the weights derived from the SHAP formula. Let's examine each feature in detail.

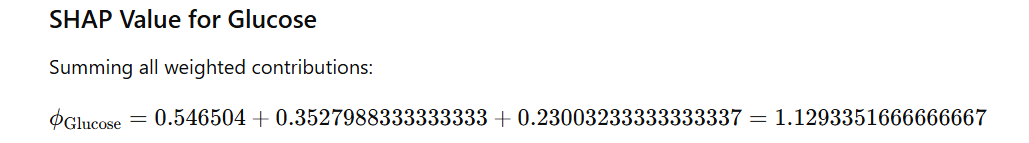
Shap value for glucose:



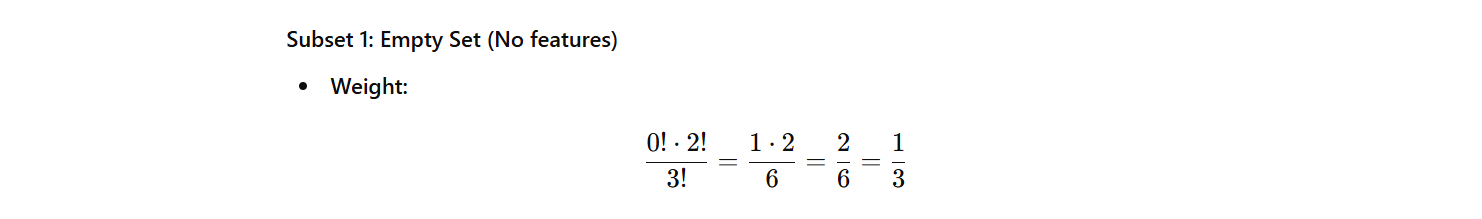


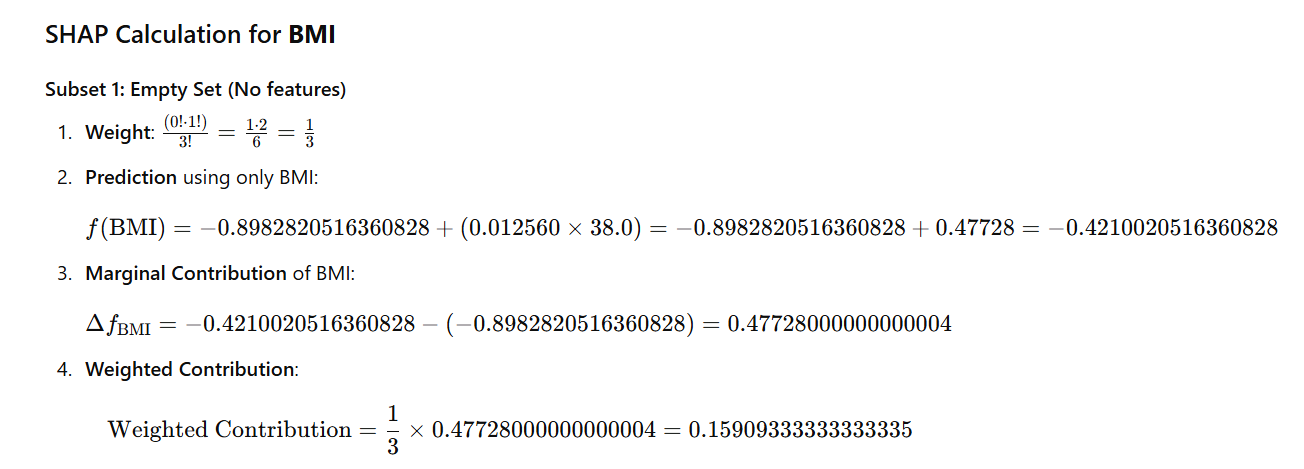


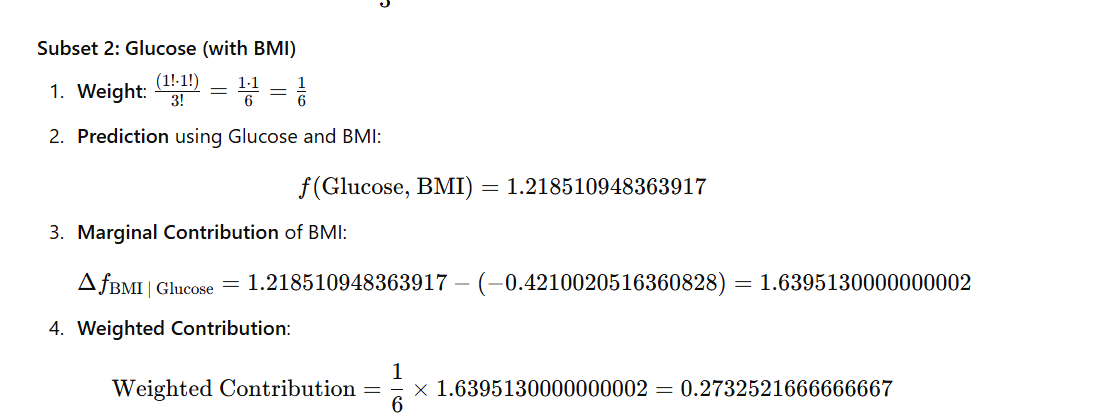




Shap values for BMI

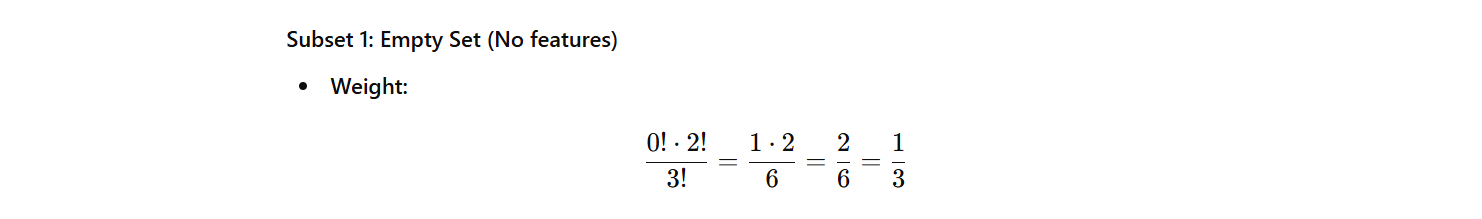


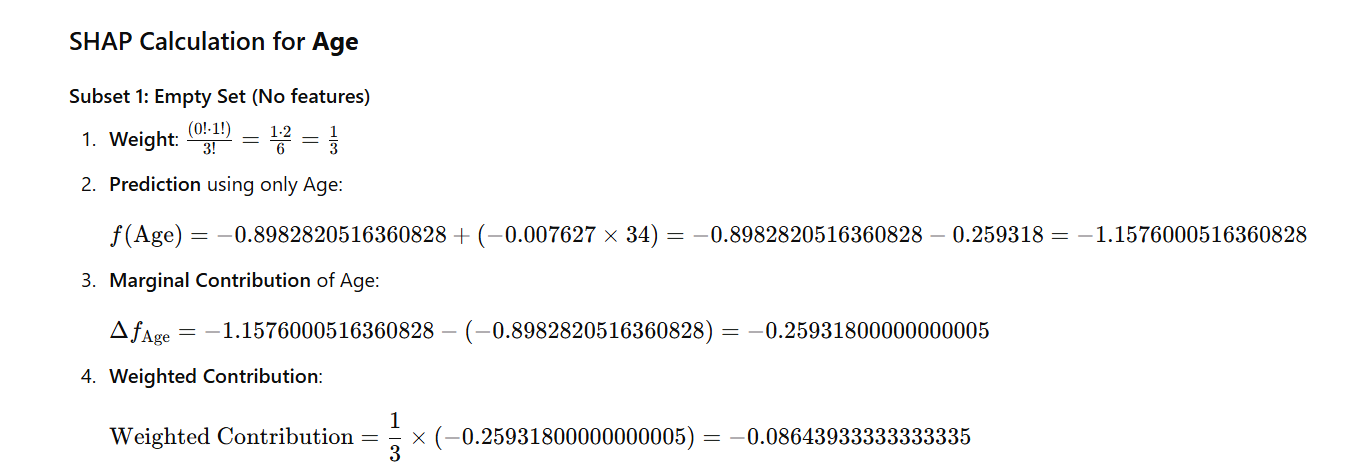


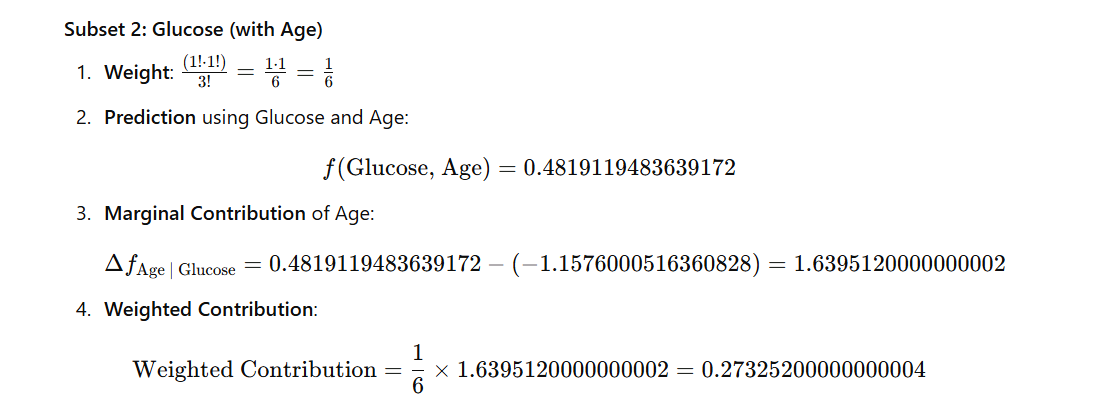




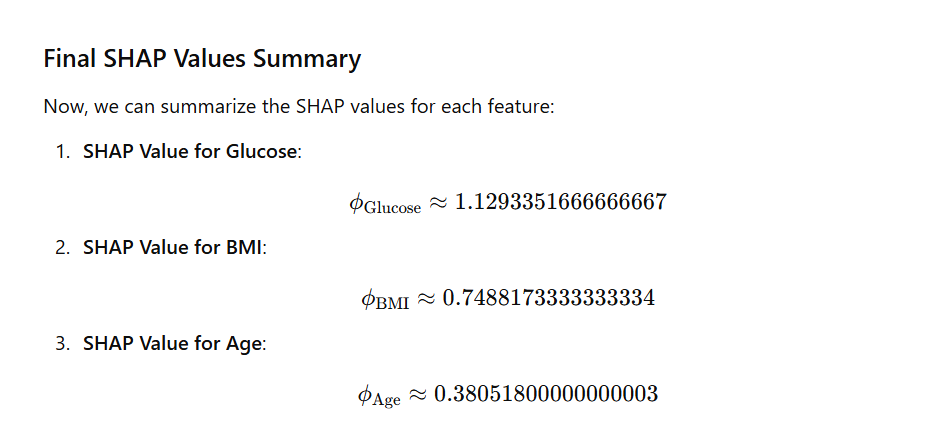
Shap value for AGE:



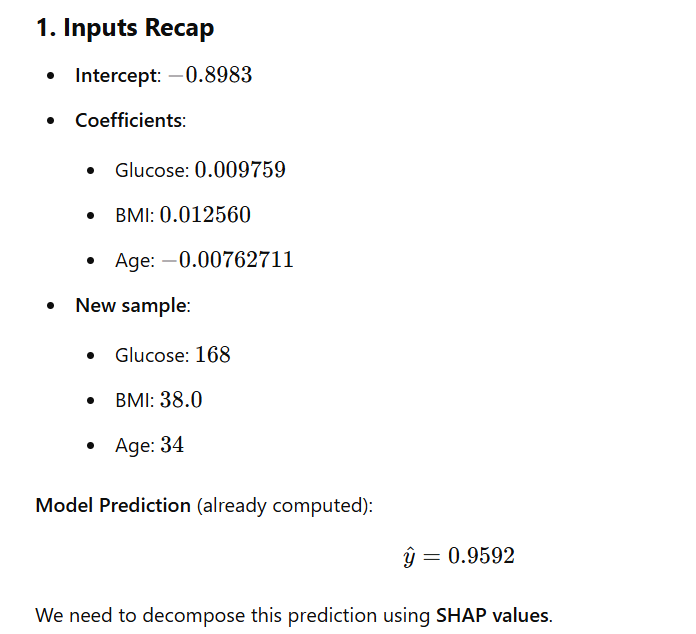






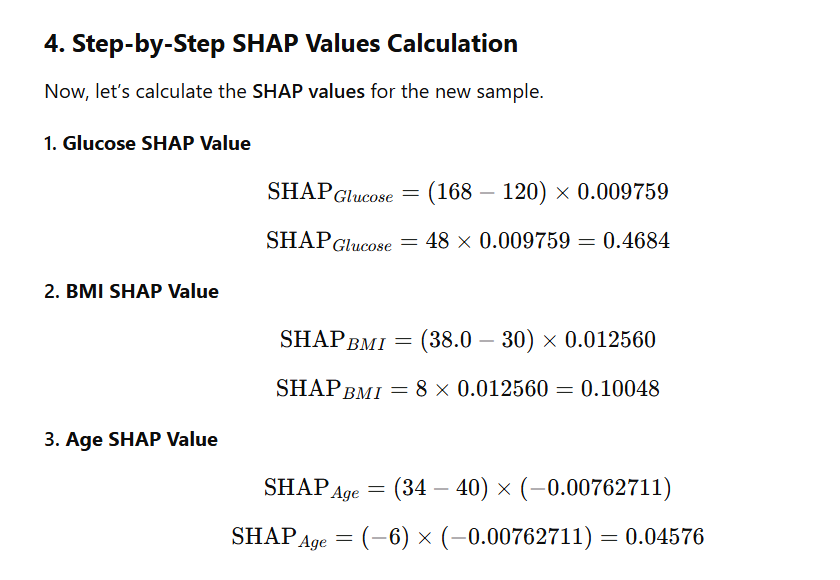


**SHAP Calculation for Each Feature using linear explainer.**









**Code for predicting diabetes using shap**

# Install SHAP (if not already installed)

!pip install shap

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

import shap

import matplotlib.pyplot as plt

# Load dataset from CSV file

data = pd.read\_csv('shap1.csv')  # Replace 'your\_dataset.csv' with your actual CSV file name

# Display the first few rows of the dataset

print("Loaded dataset:")

print(data.head())

# Assume the target variable is named 'Outcome' and the rest are features

X = data.drop("Outcome", axis=1)  # Features

y = data["Outcome"]  # Target variable

# Split into training and testing data (80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train linear regression model

model = LinearRegression()

model.fit(X\_train, y\_train)

# Print the intercept and coefficients

print("Intercept:", model.intercept\_)

coefficients = pd.Series(model.coef\_, index=X.columns)

print("Coefficients:")

print(coefficients

# Create a SHAP explainer

explainer = shap.Explainer(model, X\_train)

# Calculate SHAP values for the entire test set

shap\_values = explainer(X\_test)

# Convert SHAP values to DataFrame (for inspection)

shap\_values\_df = pd.DataFrame(shap\_values.values, columns=X.columns)

# Plot SHAP summary plot for the entire test set

shap.summary\_plot(shap\_values.values, X\_test, feature\_names=X.columns)

# Select a single sample from the test set

new\_sample = X\_test.iloc[[-1]]

print("New sample for prediction:")

print(new\_sample)

# Predict the value for the new sample

predicted\_value = model.predict(new\_sample)

print(f'Predicted value for new sample: {predicted\_value[0]}')

# Calculate SHAP values for the new sample

new\_sample\_shap\_values = explainer(new\_sample)

print(new\_sample\_shap\_values

# Plot SHAP values for the new sample (force plot)

shap.force\_plot(explainer.expected\_value, new\_sample\_shap\_values.values[0], new\_sample, feature\_names=X.columns, matplotlib=True)

# Waterfall plot for single instance

shap.waterfall\_plot(new\_sample\_shap\_values[0])

OUTPUT:

Glucose BMI Age Outcome

0 148 33.6 50 1

1 85 26.6 31 0

2 183 23.3 32 1

3 89 28.1 21 0

4 137 43.1 33 1

Intercept: -0.8982820516360828

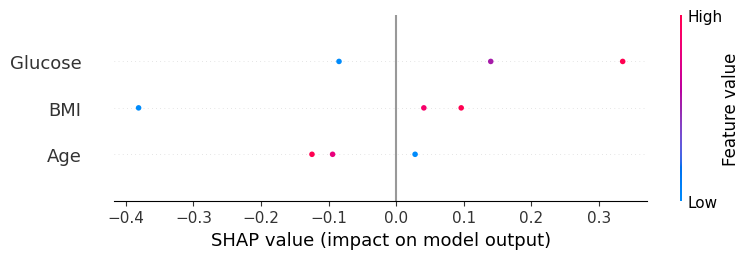
Coefficients:

Glucose 0.009759

BMI 0.012560

Age -0.007627

dtype: float64



New sample for prediction:

Glucose BMI Age

0 148 33.6 50

Predicted value for new sample: 0.586736206126522

.values =

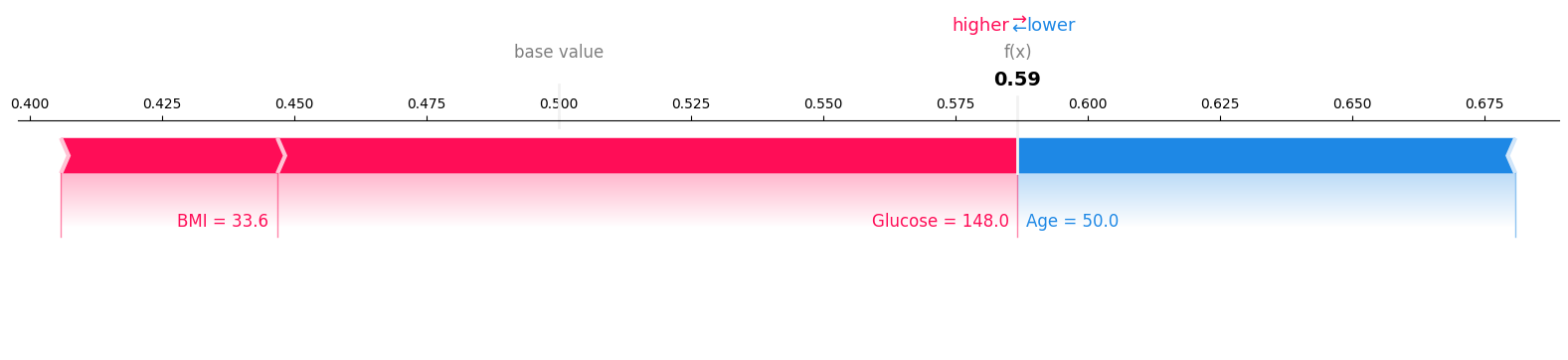
array([[ 0.13988387, 0.04092376, -0.09407142]])

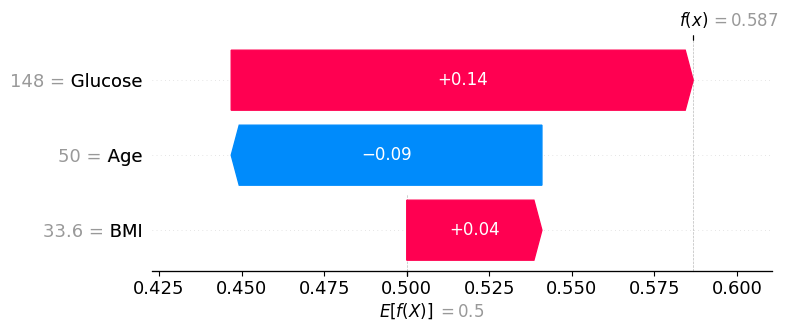
.base\_values =

array([0.5])

.data =

array([[148. , 33.6, 50. ]])





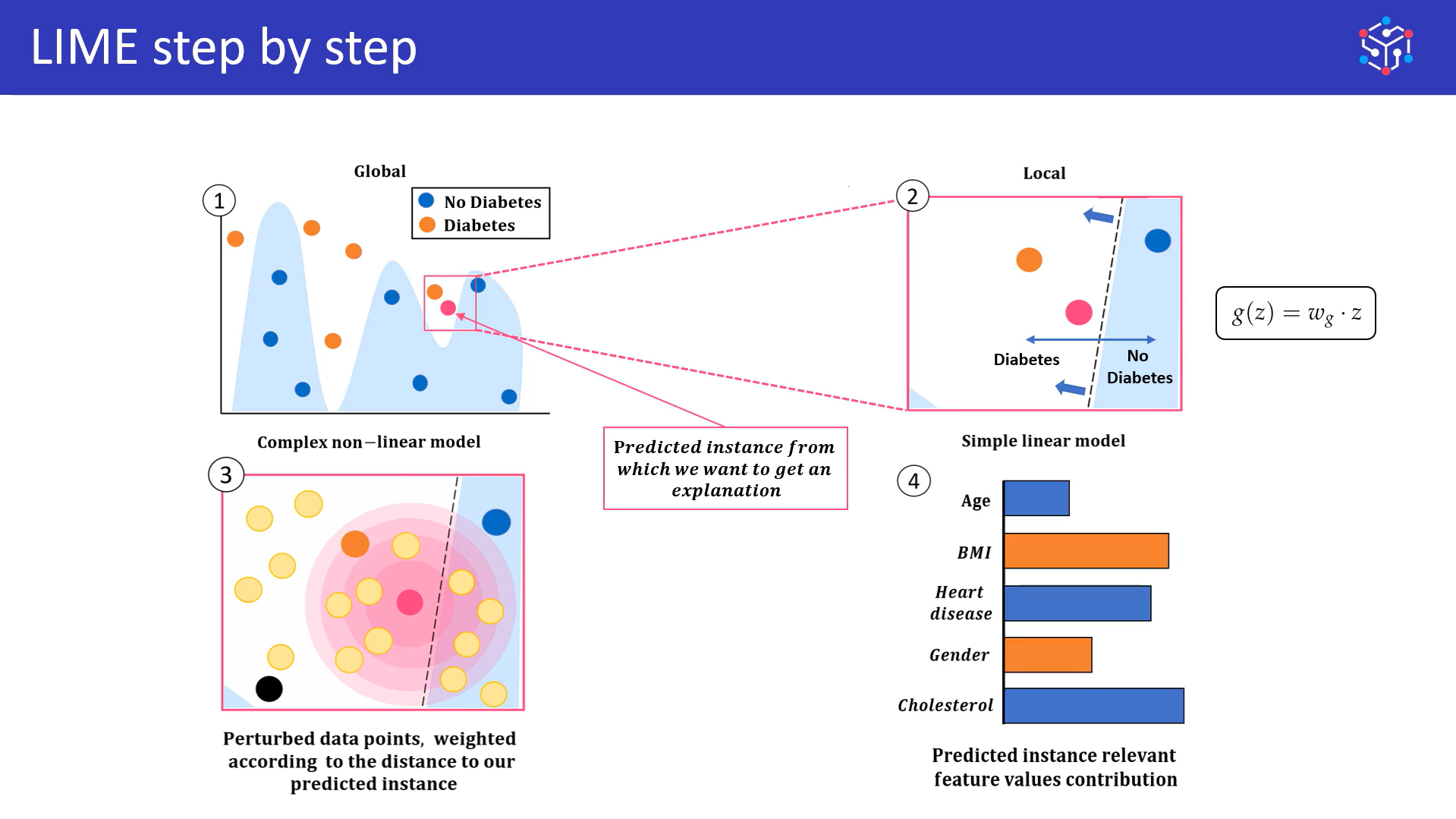
**LIME [Local Interpretable Model-agnostic Explanations]**

**LIME** is an algorithm that generates local explanations for individual predictions of any machine learning model by perturbing the input data and fitting a simple interpretable model (like a linear regression) in the vicinity of the prediction.

**Key Features**

* **Local**: Focuses on understanding individual predictions rather than the entire model.
* **Model-agnostic**: Can be applied to any machine learning model, regardless of its architecture.
* **Interpretable**: Provides explanations that are easily understandable by humans, often through feature importance scores

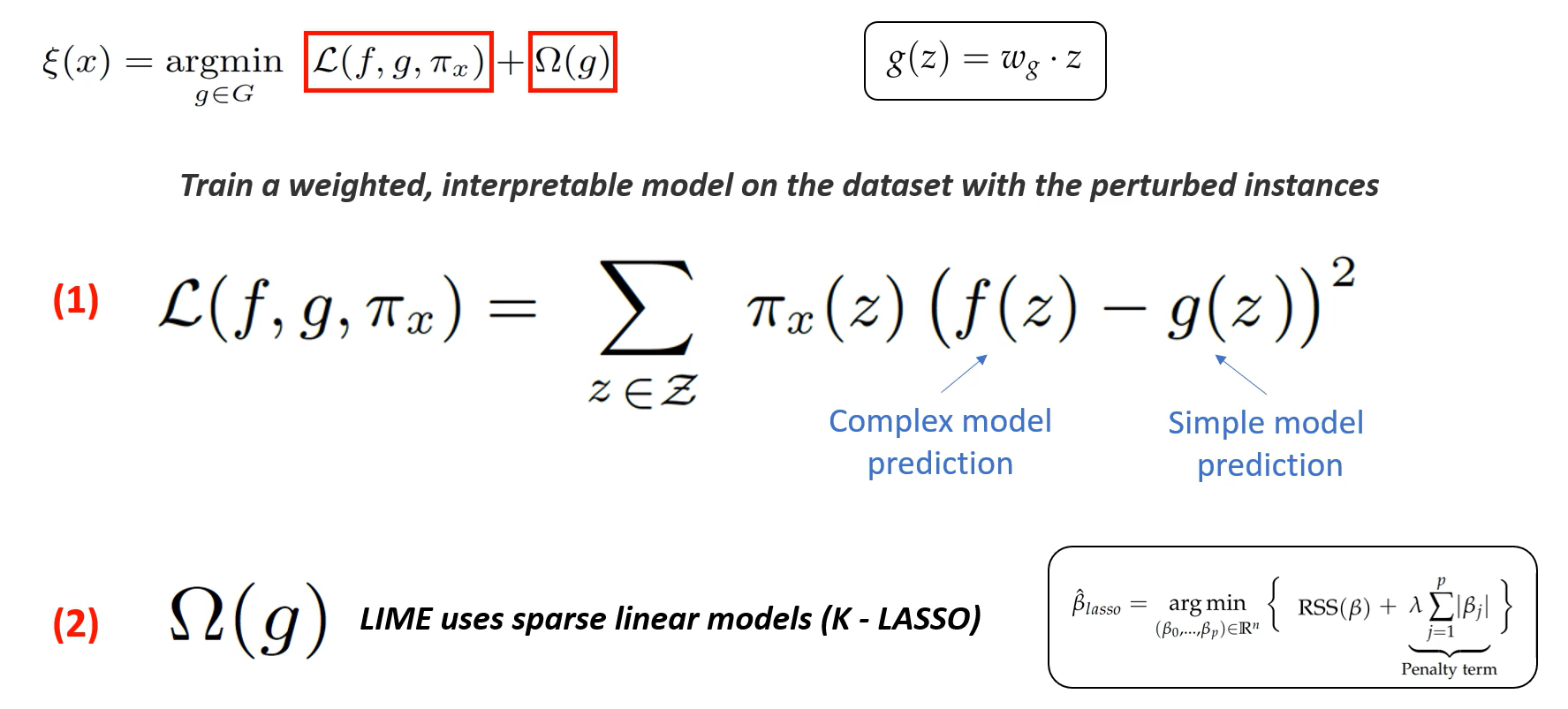
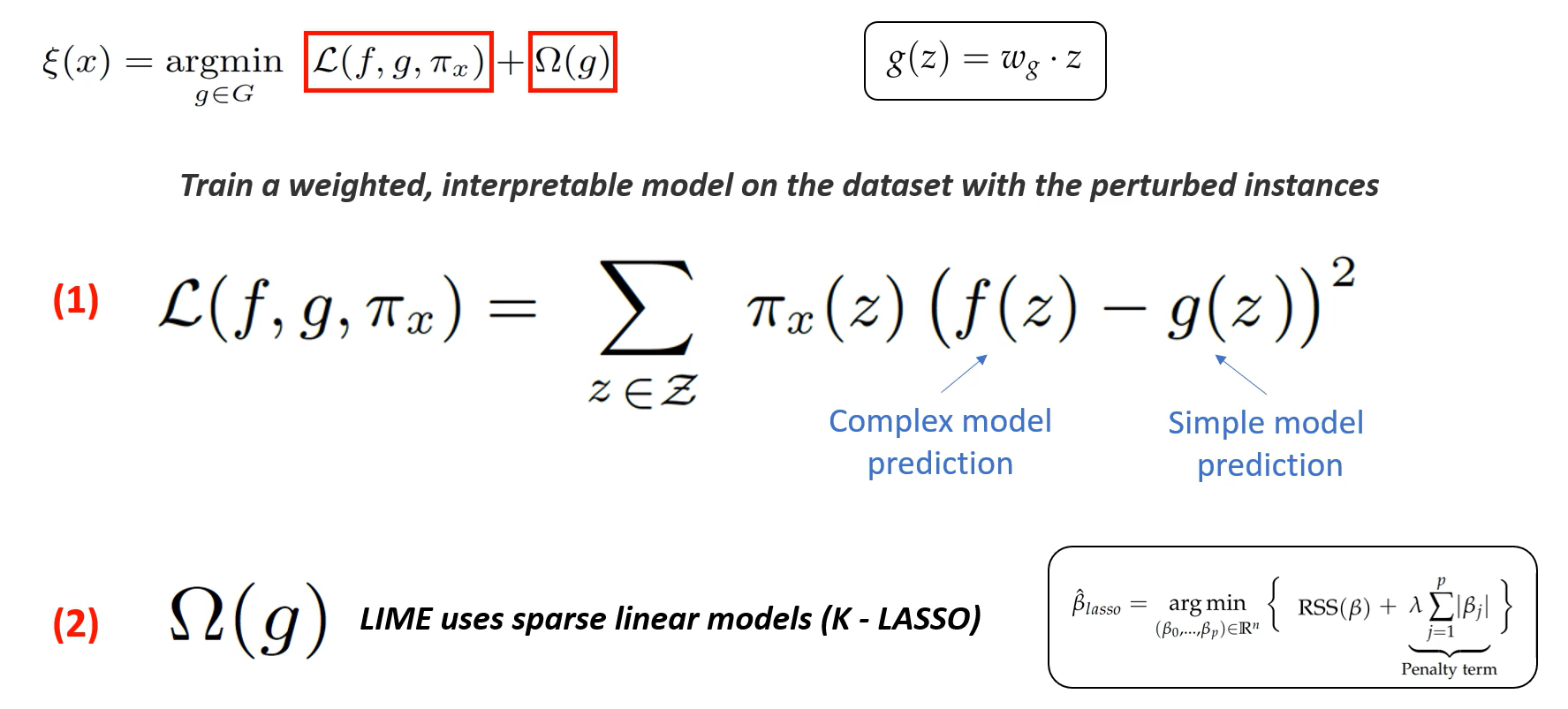
**LIME step by step**



**Lime calculation**

****

**Expansion of the formula**

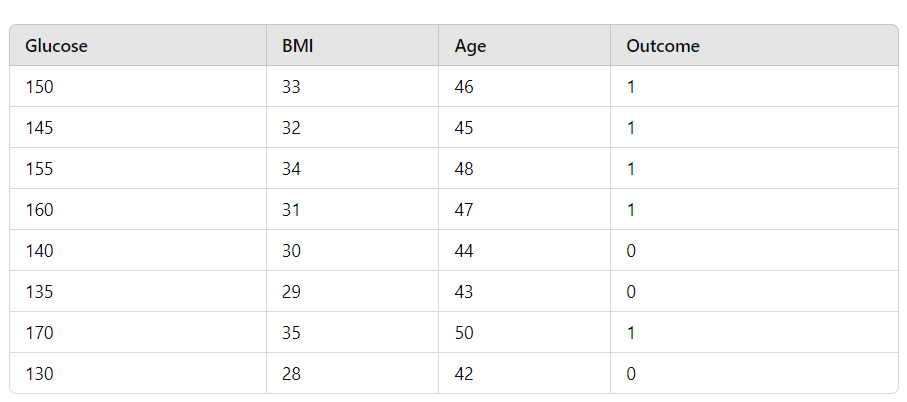
****

**Example to understand the lime calculation:**

**Sample Dataset**

Let's create a sample diabetes dataset with features relevant for prediction. The dataset includes:

* **Glucose**
* **BMI**
* **Age**
* **Outcome** (1 = Diabetes, 0 = No Diabetes)



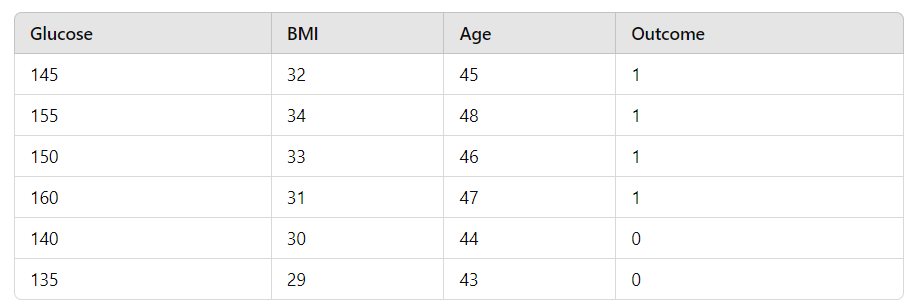
**Step 1: Define the Original Instance**

For this example, let’s take the original instance (the one we want to explain) with the following values:

* **Glucose**: 150
* **BMI**: 33
* **Age**: 46
* **Outcome**: 1 (Predicted diabetes)

**Step 2: Generate Perturbed Instances**

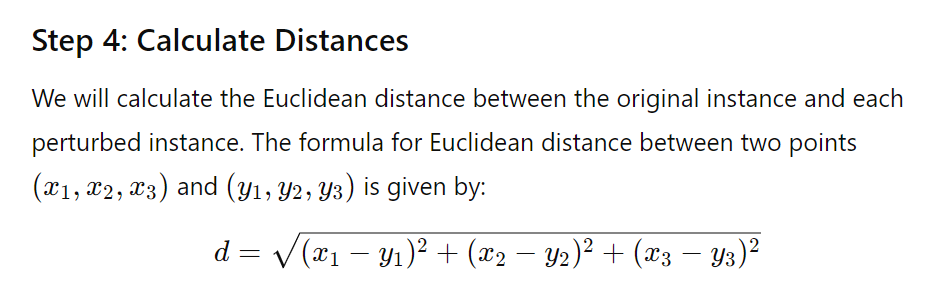
We’ll generate perturbed instances by slightly altering the values of the original instance. Here’s a set of perturbed instances based on small random changes:

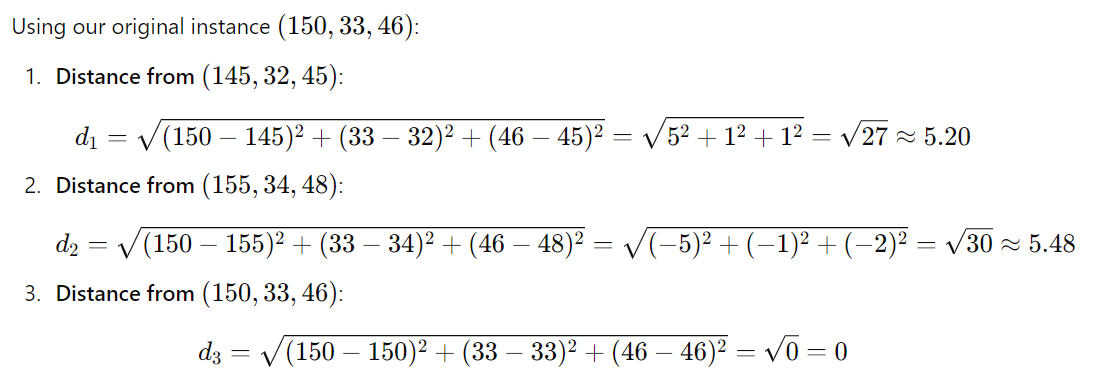


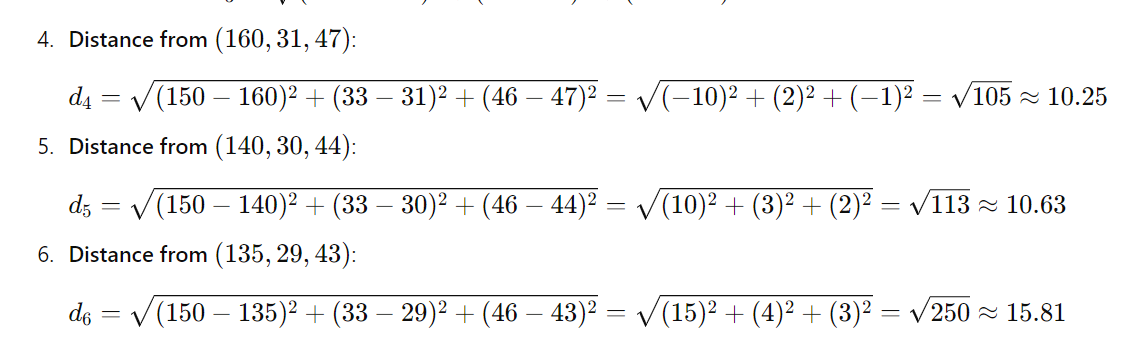
**Step 3: Calculate Complex Model Predictions f(z)**

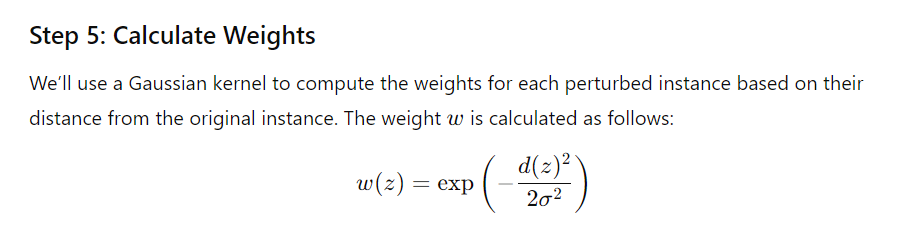
Assuming a complex model (e.g., a Random Forest) predicts the following probabilities of having diabetes for the perturbed instances:

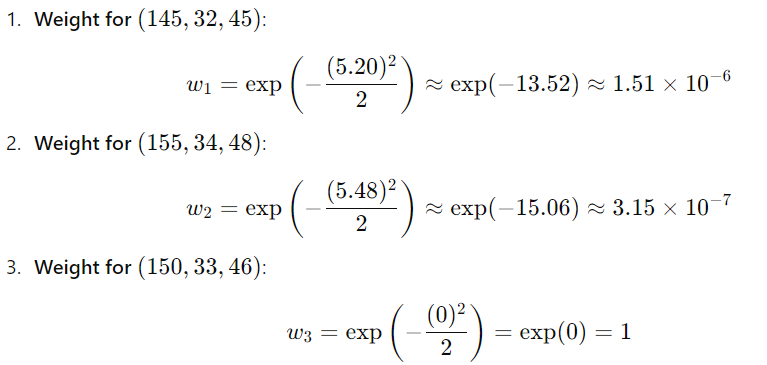
* **Complex Predictions**:
  + f(145,32,45)=0.9
  + f(155,34,48)=0.8
  + f(150,33,46)=0.85
  + f(160,31,47)=0.75
  + f(140,30,44)=0.6
  + f(135,29,43)=0.4

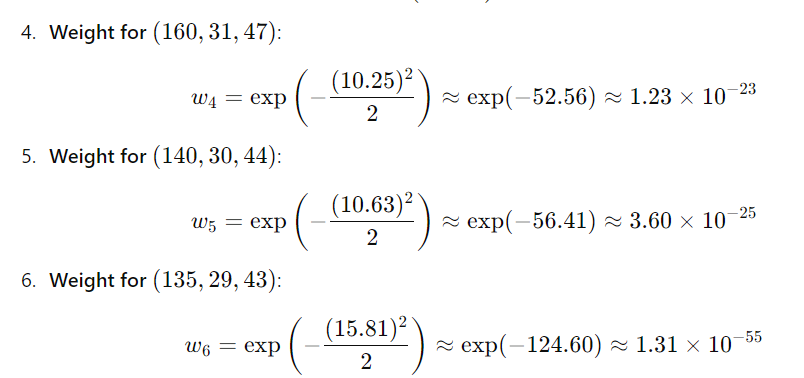


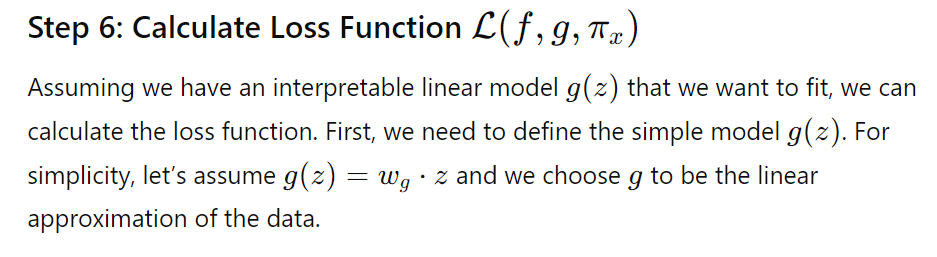


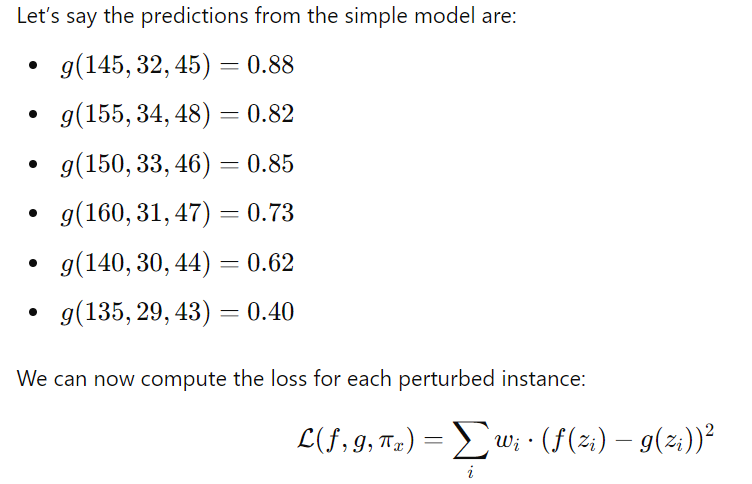


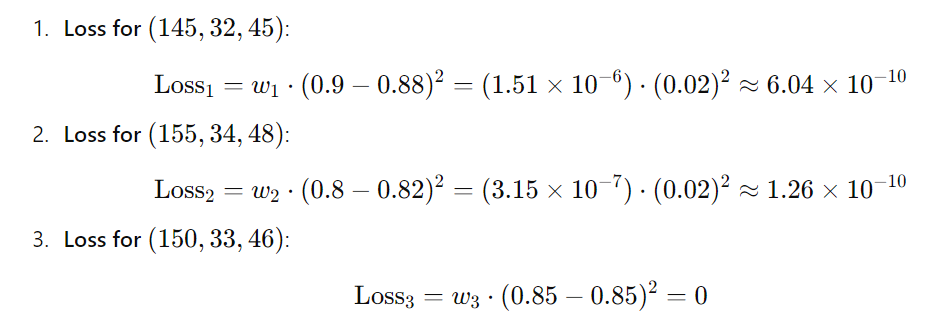


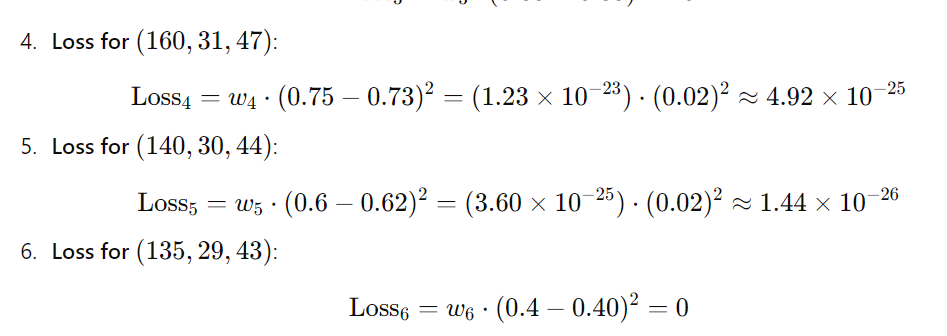




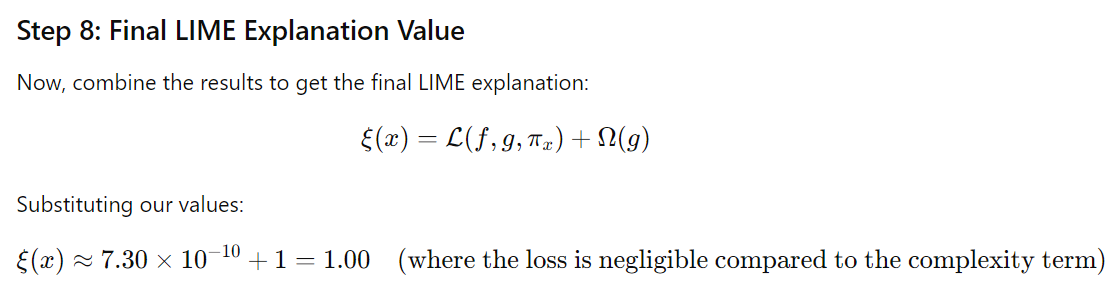












**Code for predicting diabetes using lime**

import numpy as np

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.ensemble import RandomForestClassifier

import lime

import lime.lime\_tabular

import matplotlib.pyplot as plt

# Load the dataset

data = pd.read\_csv('LIME.CSV') # Ensure the file path is correct

# Separate features and target

X = data.drop("Outcome", axis=1)

y = data["Outcome"]

# Split into training and testing data

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Train the Random Forest classifier

rf\_model = RandomForestClassifier(n\_estimators=100, random\_state=42)

rf\_model.fit(X\_train, y\_train)

# Define feature and class names for LIME

feature\_names = X.columns

class\_names = ['Non-Diabetic', 'Diabetic']

# Create a LIME explainer

explainer = lime.lime\_tabular.LimeTabularExplainer(

X\_train.values,

feature\_names=feature\_names,

class\_names=class\_names,

mode='classification'

)

# Use specific input values for a new test sample

new\_sample = pd.DataFrame({

'Glucose': [150],

'BMI': [33],

'Age': [46]

}, columns=feature\_names) # Ensure the column names match

# Predict the class for the new sample

predicted\_class = rf\_model.predict(new\_sample)[0]

predicted\_class\_name = class\_names[predicted\_class]

print(f'Predicted class: {predicted\_class\_name}')

# Get LIME explanation for the new sample

exp = explainer.explain\_instance(new\_sample.values[0], rf\_model.predict\_proba, num\_features=3)

# Print the LIME explanation

print("\nLIME Explanation (Feature Contributions):")

for feature, contribution in exp.as\_list():

print(f"{feature}: {contribution:.4f}")

# Visualize the explanation

exp.show\_in\_notebook(show\_table=True)

fig = exp.as\_pyplot\_figure()

plt.show()

output:

Predicted class: Diabetic

LIME Explanation (Feature Contributions):

32.00 < BMI <= 33.75: 0.1332

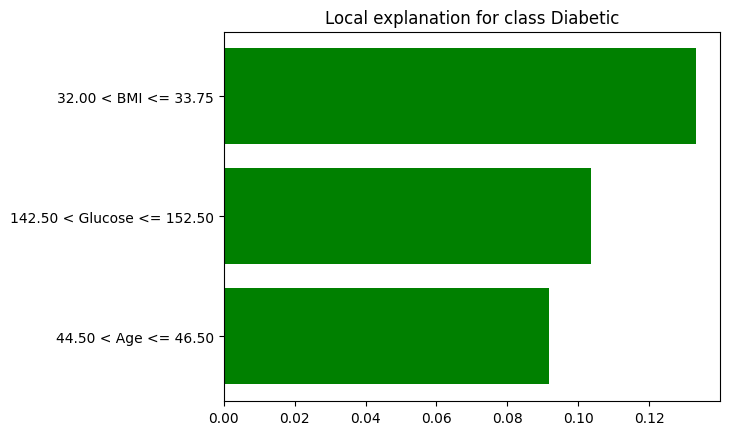
142.50 < Glucose <= 152.50: 0.1035

44.50 < Age <= 46.50: 0.0919

Prediction probabilities0.05Non-Diabetic0.95Diabetic

Non-DiabeticDiabetic32.00 < BMI <= 33.750.13142.50 < Glucose <= 1...0.1044.50 < Age <= 46.500.09

|  |  |
| --- | --- |
| Feature | Value |
| BMI | 33.00 |
| Glucose | 150.00 |
| Age | 46.00 |



m

Difference between Lime and Shap:

