Model Interpretability & Explainability Using SHAP

Understanding How XGBoost Algorithm Make Decisions

by : Rola and Abdelrahman

Agenda

01

Abstract

02

Introduction

03

Methodology

04

XGBoost Library

05

Dataset overview

06

Interpretability using SHAP AI

07

Limitation

08

Application

Abstract

This presentation discusses a powerful technique for improving weaker models, known as the Gradient Boosting algorithm.

We will explore how it works, why it is used, the methodology behind it, and its effectiveness in achieving higher accuracy.

Introduction

- **The Gradient Boosting algorithm is one of the most powerful techniques for improving predictive models and achieving higher accuracy.**
- **Gradient Boosting can be applied to different types of problems, including both classification and regression.**
- **Introduced by Jerome H. Friedman in his paper “Greedy Function Approximation: A Gradient Boosting Machine” in 1999. It became commonly used in practice during the 2010s**

Methodology

- Gradient boosting works by building sequential decision trees to predict residuals
- The algorithm takes the results of a weaker model and then builds decision trees sequentially where each tree learns from the previous tree

- New Prediction = Old Prediction + η x New Tree Prediction

Methodology

- Let’s take this data set for example and try to find a model that predict the Weight (Target)
- Let’s assume initially that the model is a line representing
- the mean

$$\frac{88 + 76 + 56 + 73 + 77 + 57}{6} = 71.2$$

- The line representing the mean (71.167) can be seen as a very simple weak model, since predicting the same average value for all inputs is too basic and fails to capture important differences in the data.

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88
1.6	Green	Female	76
1.5	Blue	Female	56
1.8	Red	Male	73
1.5	Green	Male	77
1.4	Blue	Female	57

Methodology

- **Now we will calculate the residual (Error) for each row**

- **Next step is to build a decision tree to predict the residuals using the current features (Height, Favorite Color, Gender)**

Height (m)	Favorite Color	Gender	Weight (kg)	Residua
1.6	Blue	Male	88	16.8
1.6	Green	Female	76	4.8
1.5	Blue	Female	56	-15.2
1.8	Red	Male	73	1.8
1.5	Green	Male	77	5.8
1.4	Blue	Female	57	-14.2

Methodology

We average the value at each branch

Methodology

- If we tried to apply the decision tree now we would encounter a problem
- Let’s take the first row as example and try to calculate it’s predicted value using the residual that the decision tree outputs

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

Methodology

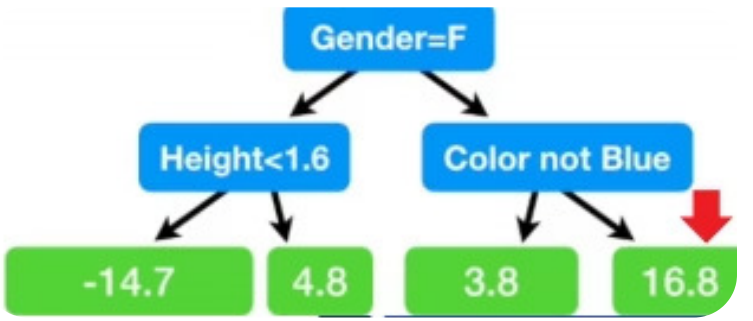
- If we tried to apply the decision tree now we would encounter a problem
- Let's take the first row as example and try to calculate it's predicted value using the residual that the decision tree outputs

$$71.2 + 16.8 = 88$$

This is an Overfitting problem

- This is why we introduced η (The learning rate) at the first equation

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88

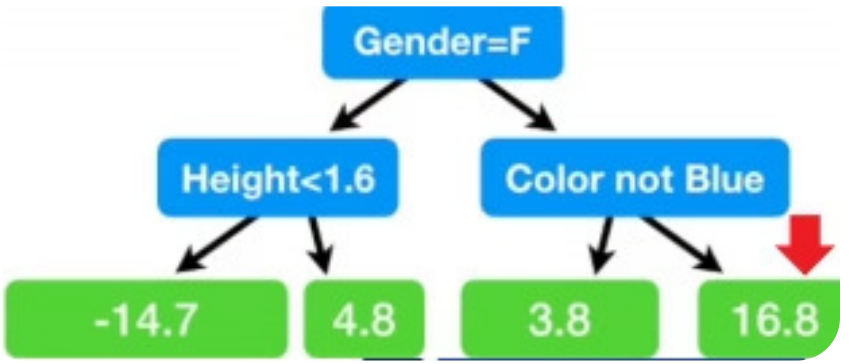


Methodology

- Now if we plugin η for 0.1 in the equation we get 72.88

$$71.2 + 16.8 = 88$$

Height (m)	Favorite Color	Gender	Weight (kg)
1.6	Blue	Male	88



Methodology

- Now we recalculate the residuals using the decision tree.
- We can see that the residuals (errors) are reduced, which leads to higher accuracy.
- As more decision trees are added, the error continues to decrease and the accuracy improves, all while keeping the model from overfitting

Methodology

- We repeat the process using the updated residuals.
- This produces a new decision tree that sequentially corrects the errors of the previous one.

$$71.2 + 0.1 \times 16.8 + 0.1 \times 15.1 = 74.39$$

- Now if we calculate the residuals for the first row again we get 13.6

Height (m)	Favorite Color	Gender	Weight (kg)	Residual
1.6	Blue	Male	88	15.1
1.6	Green	Female	76	4.3
1.5	Blue	Female	56	-13.7
1.8	Red	Male	73	1.4
1.5	Green	Male	77	5.4
1.4	Blue	Female	57	-12.7

Methodology

- Once again, the residuals decrease after adding another decision tree.

- We can think of decision trees as learning steps, where each one builds on the mistakes of the previous and produces a slightly better result

XGBoost Lib

Now that we understand how Gradient Boosting algorithm works let's take a look at how XGBoost library in python implements it

- **XGBoost implements the algorithm a little differently to improve the results of the algorithm**

1. Regularization: Prevents overfitting and keeps trees simpler
2. Shrinkage: Helps prevent overfitting by scaling down the impact of each new tree, ensuring the model learns more gradually and generalizes better.
3. Second-order approximation: XGBoost uses both the gradient and the Hessian (second derivative) when building trees, which helps it make more accurate splits.

Limitations

- **Just like any other machine learning algorithm Gradient Boosting has it’s limitations**

1. Overfitting: Gradient Boosting has a high risk of overfitting.
2. Outliers: Gradient Boosting is extremely sensitive to outliers since each tree is obliged to fix the error in the previous one.
3. Scalability & Computational Cost: Because each tree depends on the previous one, gradient boosting can’t train trees in parallel. On large datasets, this sequential process becomes slow and memory-intensive, making it hard to scale.

Applications

- **Gradient Boosting is widely used for its high accuracy. Some common applications include:**

1. Retail and e-commerce: personalized recommendations, inventory management, fraud detection.
2. Finance and insurance: credit risk assessment, business prediction, algorithmic trading.
3. Healthcare and medicine: disease diagnosis, drug discovery, personalized medicine.
4. Search and online advertising: search ranking, ad targeting, click-through rate prediction.

Dataset Overview: FlavorSense

The chosen dataset is designed to predict people's taste preferences (spicy, sweet, sour, salty) based on their lifestyle habits, making it both **engaging and relatable**.

- **Type:** Multi Classification
- **Samples:** 10,000
- **Features:** 6 features

Go To Dataset

	age	sleep_cycle	exercise_habits	climate_zone	historical_cuisine_exposure	preferred_taste
0	56.0	Irregular	Heavy	Temperate	NaN	Salty
1	NaN	Night Owl	Heavy	Temperate	Asian	Sweet
2	46.0	Night Owl	Heavy	Cold	Mixed	Sour
3	32.0	Early Bird	Heavy	Cold	Mediterranean	Salty
4	60.0	Night Owl	Moderate	Dry	Mediterranean	Sour

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10000 entries, 0 to 9999
Data columns (total 6 columns):
 #   Column                      Non-Null Count  Dtype
---  ------                      --------------  -----
 0   age                         9032 non-null   float64
 1   sleep_cycle                 9316 non-null   object
 2   exercise_habits             9196 non-null   object
 3   climate_zone                9467 non-null   object
 4   historical_cuisine_exposure 9346 non-null   object
 5   preferred_taste             10000 non-null  object
dtypes: float64(1), object(5)
memory usage: 468.9+ KB
```

Dataset Overview: FlavorSense

Why?

- Exploration of multi class classification problem which is perfect to test XGBoost ability to handle complex patterns
- The data is big enough to show complex patterns, and small enough to train quickly for light explaination
- One of the advantages of XGBoost is that it handels missing values efficienlty we will explore that
- Explore how the algorithm does with duplicates and without.

XGBoost Classifier

Since XGBoost allows us to skip many preprocessing steps

Preprocessing achieved best results:

- ONEHOTENCODER for categorical data
- Label encoder for label
- Fill Nan values => numeric(mean), categorical ("missing") (Not necessary but improved Performance)

```
ohe = OneHotEncoder(handle_unknown="ignore", sparse_output=False)
xtrainenc = ohe.fit_transform(xtrain[cat_f[:-1]])
xtestenc = ohe.transform(xtest[cat_f[:-1]])
```

```
le = LabelEncoder()
ytrain = le.fit_transform(ytrain)
ytest = le.transform(ytest)
```

```
agemean = round(xtrain['age'].mean())
xtrain['age'] = xtrain['age'].fillna(agemean)
xtest['age'] = xtest['age'].fillna(agemean)
```

XGBoost Classifier

- **objective="multi:softmax"** → Predicts class labels directly for multiclass problems.
- **num_class=4** → Specifies that there are 4 classes in the target.
- **eval_metric="mlogloss"** → Uses multiclass log loss to evaluate model performance.
- **colsample_bytree=0.6** → Uses 60% of features per tree to reduce overfitting.
- **gamma=0.3** → Minimum loss reduction required to make a split, controls tree complexity.
- **learning_rate=0.05** → Step size for each boosting iteration, lower value improves stability.
- **max_depth=3** → Limits tree depth to prevent overfitting.
- **n_estimators=300** → Number of boosting rounds (trees) to train.
- **subsample=1.0** → Uses all rows for each tree, can reduce overfitting if <1.0.

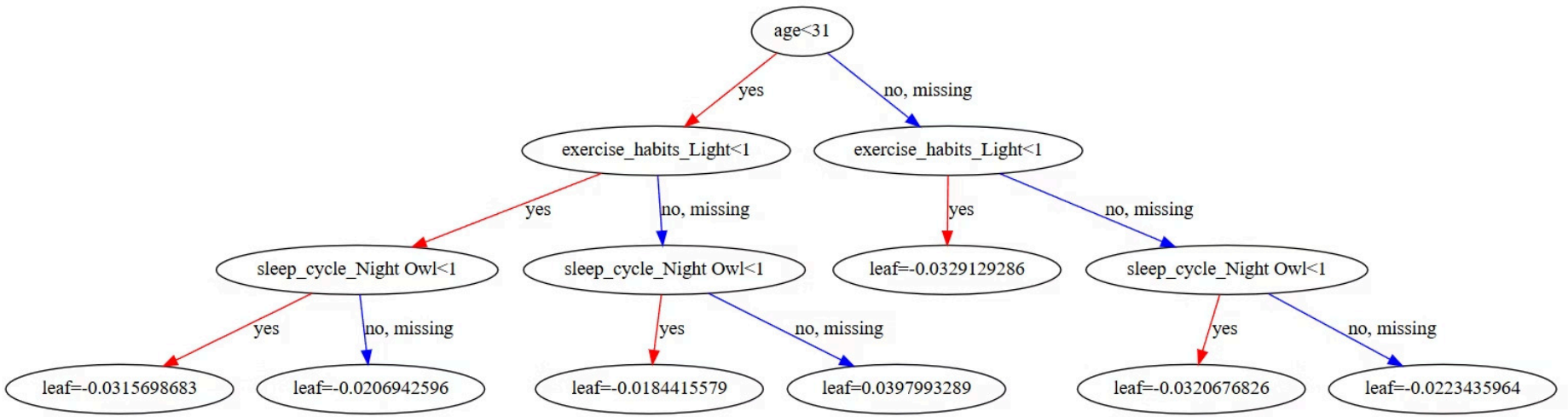
```
model = XGBClassifier(
    objective="multi:softmax",
    num_class = 4,
    eval_metric="mlogloss",
    random_state=42,
    colsample_bytree=0.6,
    gamma=0.3,
    learning_rate=0.05,
    max_depth=3,
    n_estimators=300,
    subsample=1.0,
    n_jobs=-1
)
model.fit(xtrain, ytrain)
```

Reference 1 Reference 2 Reference 3

SHAP Explainable AI Intro

In tree models, I can identify which features contributed the most to a prediction, but I can't tell whether that contribution was positive or negative.

For example, I might know that **AGE** was important in predicting an outcome, but I don't know if being older increased or decreased the prediction.



XGBoost plotting

That's where SHAP (SHapley Additive exPlanations) comes in.

Global Feature Importance: The SHAP Summary Plot

This SHAP summary plot indicates the avarage impact of each feature on the model output

What it Shows:

x - axis → mean SHAP value indicating how much each feature contributes to prediction on avarage.

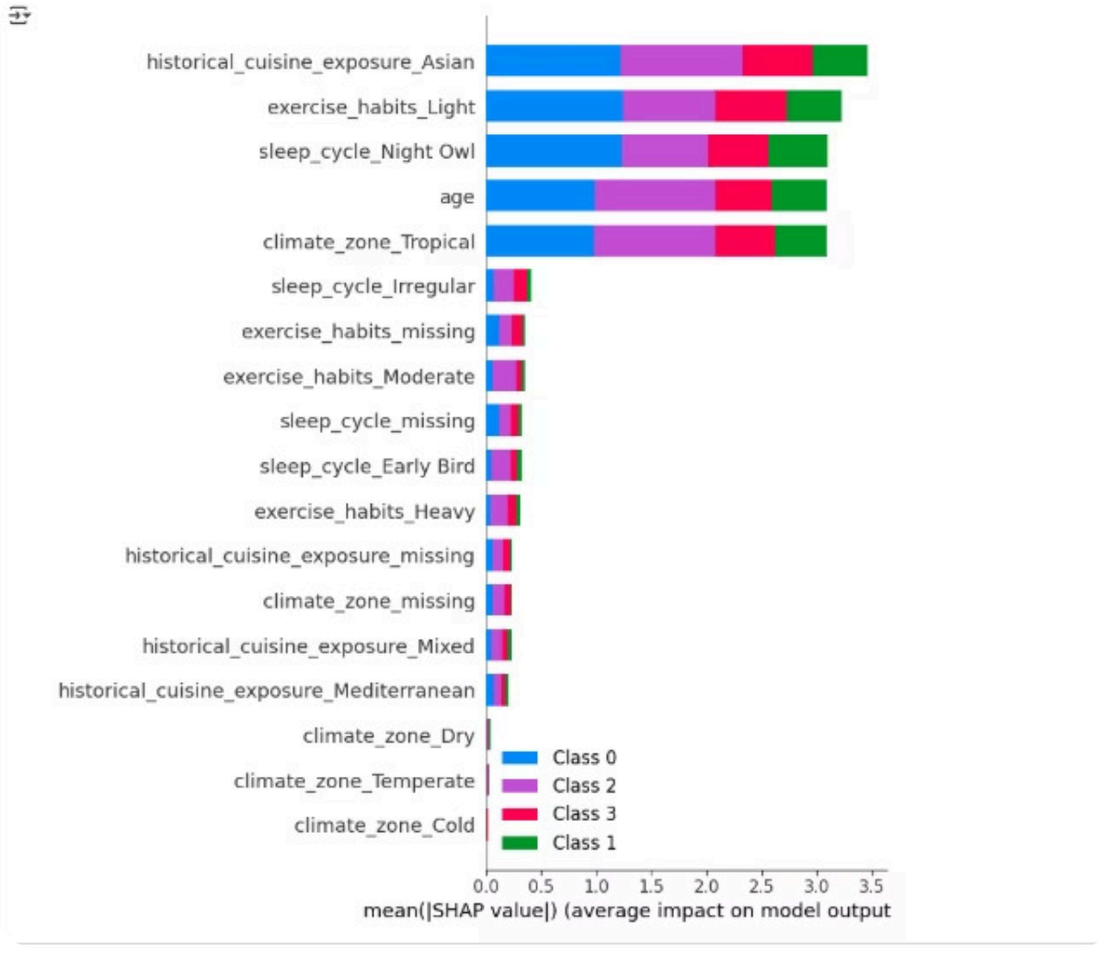
y- axis → features ranked by average SHAP values (importance)

Colors → represents classes so you can see which feature matters more for each class.

WHICH FEATURES ARE MOST IMPORTANT OVERALL ACROSS THE ENTIRE DATASET?

Key takeaways:

- historical_cuisine_exposure_Asian, exercise_habits_Light, sleep_cycle_Night Owl, age, and climate_zone_Tropical are the **top 5 most influential features** globally.
- Features like exercise_habits_Heavy or climate_zone_Dry have almost **no impact**.



SHAP Multiclass

SHAP Intro

Global Feature Importance: The SHAP Dependence Plot

So now we know that there are top influential features , but what if i want to know what values exactly lead to the change in prediction

Lets take age

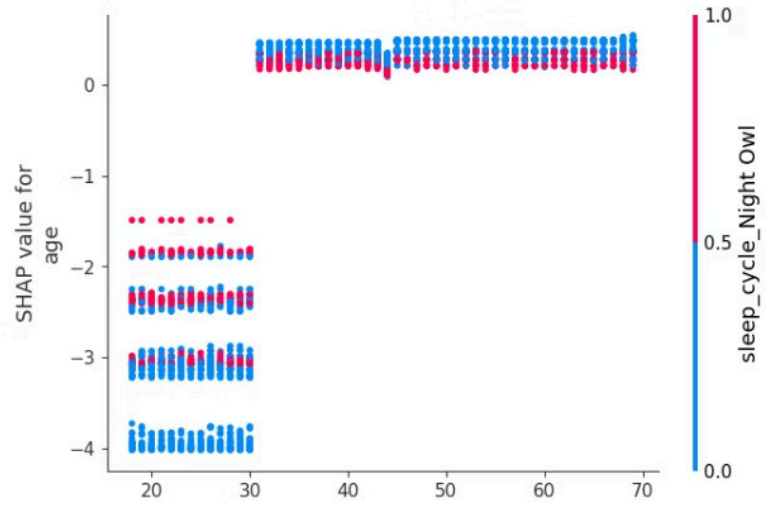
```
shap.dependence_plot("age", shap_values.values[:, :,0], xtrain)
shap.dependence_plot("age", shap_values.values[:, :,1], xtrain)
shap.dependence_plot("age", shap_values.values[:, :,2], xtrain)
shap.dependence_plot("age", shap_values.values[:, :,3], xtrain)
```

How to interpretate

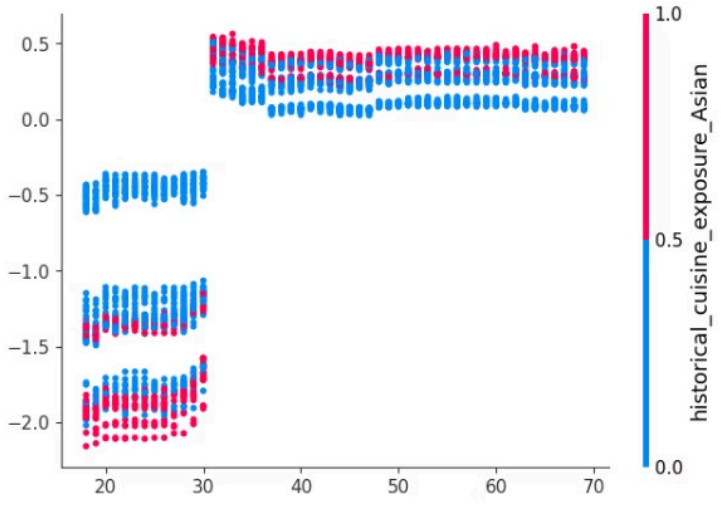
Y-axis → the shape values for age (<0) decreases the baseline prediction, (=0) no affect, (>0) increases prediction.

X- axis → Age values

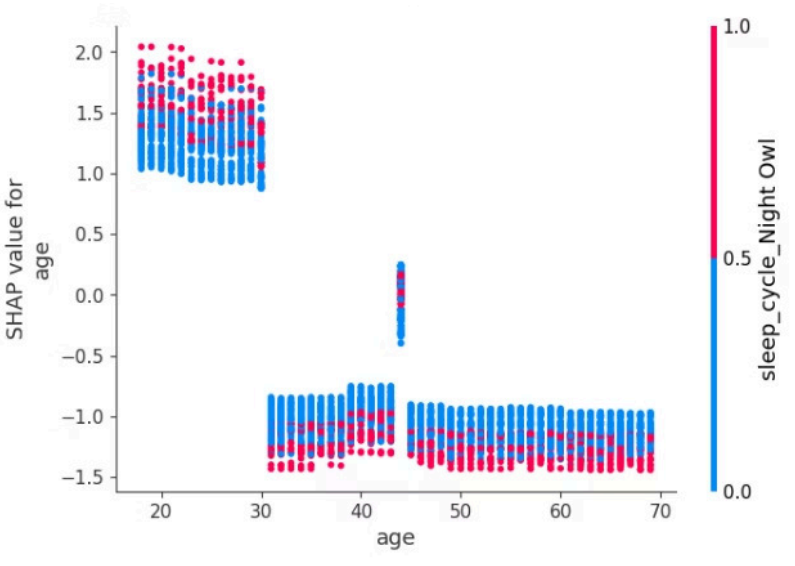
right axis : the feature that greatly interacts with age for each class



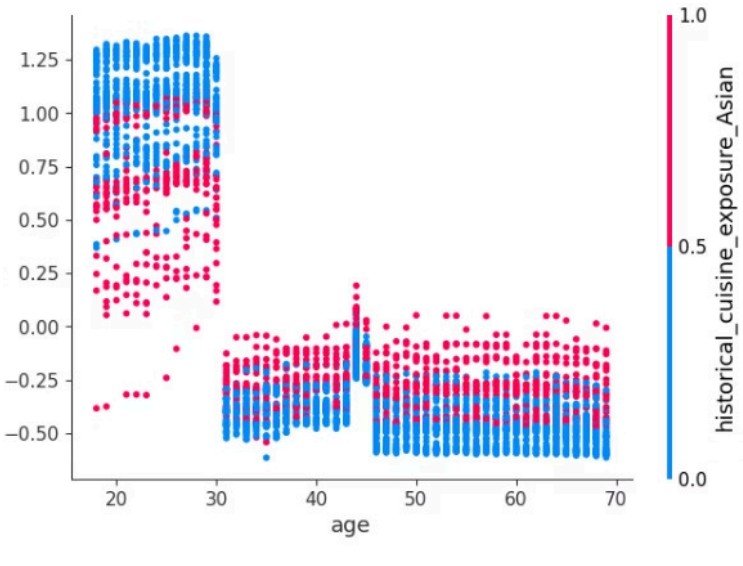
Class 0



Class 1



Class 2



Class 3

Local Feature Explanation: Let's Take a closer look at a prediction

Now i know that the class imbalance for class 1 & 3 is affecting the performance , lets zoom on the predictions that were classified as 1 & 3 to see what features contributes most to it (can be taken as a feature engineering indicator)

```
np.where((ytrain_pred==1))[0][0]
```

```
np.int64(4)
```

```
model.predict_proba(xtrain)[4]
```

```
array([0.01664915, 0.84711885, 0.00127481, 0.13495718], dtype=float32)
```

As you can see

The probabilities of record 4 indicates that class 1 was chosen due to its high weight

---

```
explainer = shap.Explainer(model)
shap_values = explainer(xtrain)

print(np.shape(shap_values))
```

```
(5059, 18, 4)
```

This step tells to SHAP which model to explain

XGBoost → TreeExplainer

Black-box → KernelExplainer

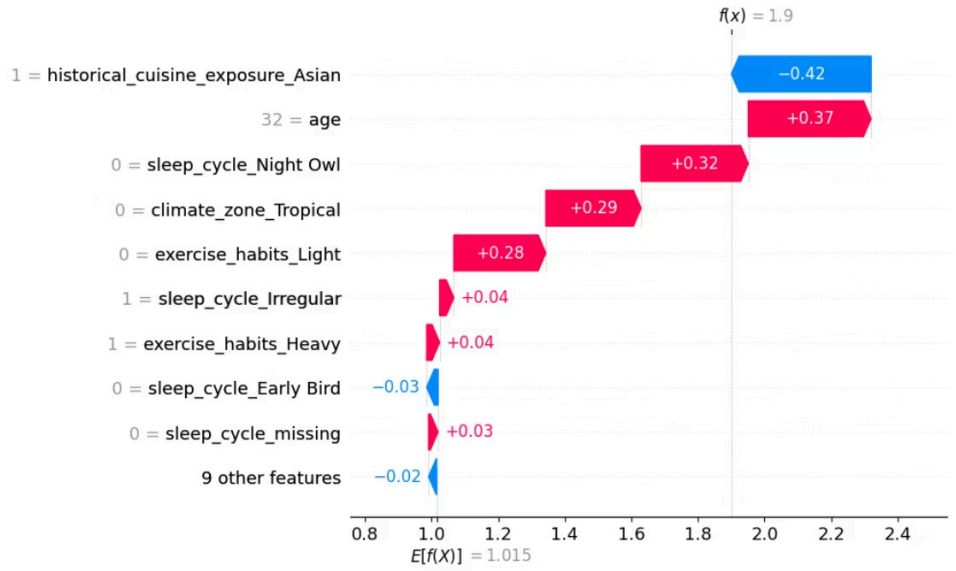
now that it knows it computes feature contributions for each value

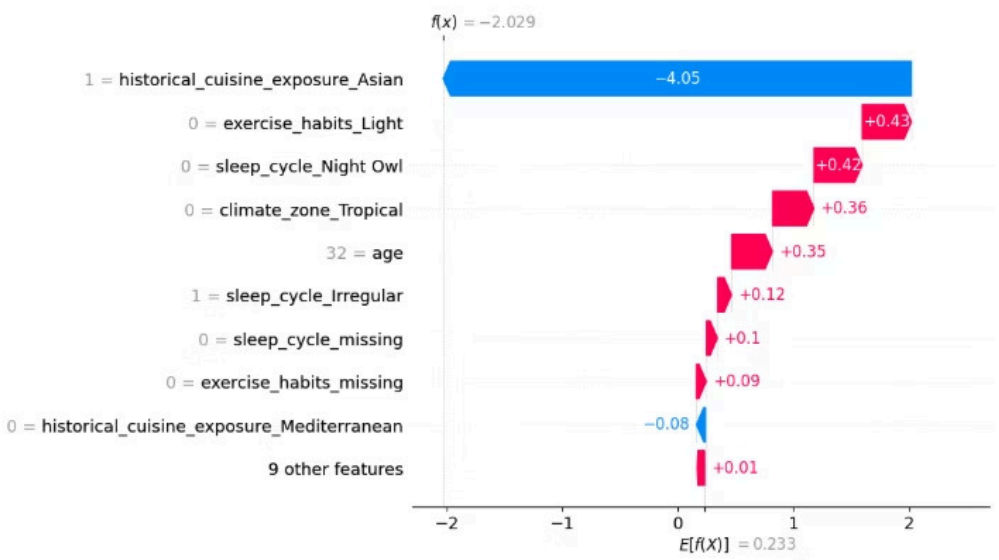
Local Feature Explanation: Waterfall plot

```
shap.plots.waterfall(shap_values[4, :, 0])
shap.plots.waterfall(shap_values[4, :, 1])
shap.plots.waterfall(shap_values[4, :, 2])
shap.plots.waterfall(shap_values[4, :, 3])
```

Right now, I want to focus on class 1 on record 4 but as you can see for prediction 4 can look at all classes to see how the features contribute to each probability I showed you before

keep in mind → [0.01664915, **0.84711885**, 0.00127481, 0.13495718]





What it shows

E(F(x)) → Baseline prediction average log-odds prediction of model before looking at features

Blue bars→ features that push the prediction lower towards 0

Red bars→ features that push predictions higher towards 1

length of bar→ Strength of Contribution (SHAP Value)

F(x) → Final Model Output after starting at baseline and adding all feature contributions.

Key Takeaways

- Asian→ **decreases** the model output.
- age, Night_owl, Tropical, Light → **increases** output to counter the decrease this was how Class 1 was reached.

AS for Class 0 :

- Asian Cuisine strongly decreased the output even with all the other features increasing the contribution it didn't counter the reduction
- This was how class 0 probability was not big enough

Comparison

To decide what do you want to use to interpretate your model

	Regular Tree	SHAP Local	SHAP Global
Explains?	One Tree only	All Trees Combined	All Trees Combined
Shows Path Rules?	Yes	No Just Contributions	No Just Contributions
Feature importance?	Yes overall	Yes, per Prediction	Yes
Magnitude & Direction of Impact	No	Direction and strength	Average strength

Importance & Limitations of SHAP

Importance

Models

Works with any ML model, tree based, linear, neural networks, black box models

Local & Global Explanations

- **Local:** Explains why a specific instance was classified as x
- **Global:** Aggregates contributions to rank features by overall importance.

Direction + Magnitude of Effect

Unlike plain feature importance, SHAP shows **whether a feature increases or decreases** the prediction and by how much.

Enhances Feature Engineering

Guides towards more impactful and less redundant features.

Limitations

Computational Cost

SHAP values require computing contributions across **all feature combinations**, which grows exponentially with number of features.

Assumption of Feature Independence

SHAP assumes features are independent when calculating contributions, if features dependant it may **over- or under-assign importance** to them.

Interpretability Challenge

While powerful, explaining SHAP plots still requires expertise for non-technical audiences.