6CCS3PRJ Final Year
SHAP Values: Explainable Methods for Predictive Analytics

Final Project Report

Author: David Gutierrez Moreno

Supervisor: Dr. Maria Polukarov

Student ID: 19032572

April, 2022

Abstract

There is a trade-off between accuracy and interpretability when it comes to machine learning models. As models grow in complexity, they tend to become highly accurate, such as deep learning models. However, that often leads to low interpretability as black-box models do not tend to have the transparency required to provide trust between their users. A variety of model explainability methods have been proposed to respond to this problem.

This project focuses on the importance of model interpretability in machine learning, specifically, how SHAP (Shapley Additive Explanations) can further demystify feature importance in model predictions. The project presents a series of problems to showcase this explainability method. The problems are addressed by training machine learning models that are best fitted. The models used are then interpreted by computing SHAP values, demonstrating how convenient their use can be applied in real-world problems. These problems use a range of non-complex models such as linear models and complex models such as gradient boosting algorithms.

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary. I grant the right to King's College London to make paper and electronic copies of the submitted work for purposes of marking, plagiarism detection and archival, and to upload a copy of the work to Turnitin or another trusted plagiarism detection service. I confirm this report does not exceed 25,000 words.

David Gutierrez Moreno

April, 2022

Acknowledgements

I would like to thank my project supervisor, Dr. Maria Polukarov, for her guidance and support throughout the length of this project.

Contents

Chapter 1

Introduction

Interpretability in machine learning is the problem of explaining machine learning reasoning to humans. Interpretability can provide insights that can benefit users, such as detecting bias in predictions, finding bugs, and gaining user trust. Although there are many white-box models, such as linear regression, that can be fairly explained, various black-box models fail to get the necessary level of transparency. Higher model complexity often leads to high accuracy and, usually, less complex models are less accurate in comparison (Figure 1.1). Model explainability methods address this problem and give high accuracy and high interpretability.

SHAP (SHapley Additive exPlanations) is a framework that can interpret any machine learning model, including black-box model predictions. SHAP assigns an importance value to the dataset used for every feature and instance. This way, one can interpret and see how much a feature affects a particular prediction. Furthermore, SHAP values can explain a model globally, explaining the entire model behaviour instead of only one individual prediction.

1.1 Project aims

Businesses are pretty careful in adopting some Machine Learning solutions due to their non-transparent nature. That is where model explainability comes into place. Concerns and suspicion can be gradually lowered with explainable AI solutions. For example, a piece of software that administers the exact drug dose for patients in healthcare by analyzing their clinical data is a situation in which there has to be a form to interpret how or why the software does the way it performs. The main objective of this project is to demonstrate the benefits of SHAP values and their use across business sectors, research, and more.

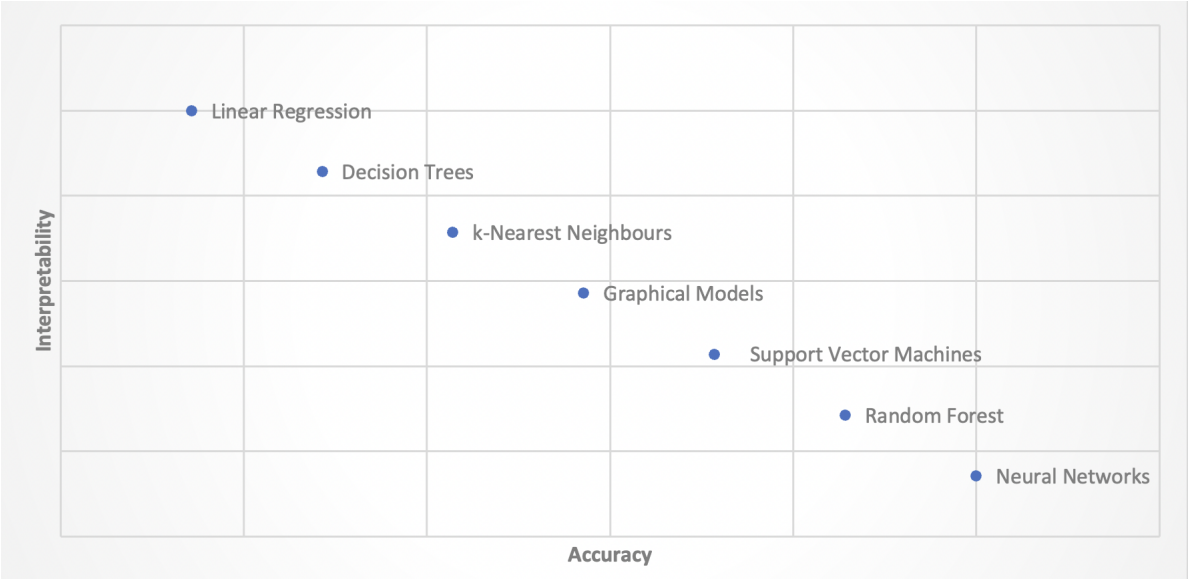


Figure 1.1: Interpretability/Accuracy trade-off between ML models

For the main section of the project, three different practical cases are conveyed, and their Shapley values are demonstrated throughout each of the case sections. We compare SHAP value explanation to other methods to interpret white-box models (transparent) and demonstrate results with black-box models (non-transparent). The first two cases will work with the development of white box machine learning models, while the third will train a black-box model. While these are some example applications in which these SHAP values can be used, their use is unlimited in the field of data analysis.

1.2 Report Structure

The report starts with a background section showing the theory behind SHAP and Shapley values (Shapley et al., 1988) and how it started as Cooperative Game Theory. Recently, in 2017 the theory was adopted into the field of machine learning by Scott M. Lundberg and Su-In Lee. Furthermore, It is shown how the idea of Shapley values moved to the field of Machine Learning for feature importance, and its convenient properties are exhibited.

The first case scenario involves a dataset containing red and white wine variants with its physio-chemical features (Cortez et al., 2009). After training the dataset with a linear regression model, the secondary objective is to predict the fixed acidity in the wine samples. We will use the SHAP library to explain feature importance locally and globally. Additionally, it will be compared with other commonly used methods to interpret the linear model.

For the second scenario, we have a dataset containing eating habits and physical condition

features from individuals from Mexico, Peru, and Colombia (Palechor & de la Hoz Manotas, 2019). Based on these features, the secondary objective is to estimate obesity levels by training a decision tree algorithm. Afterwards, it will be explained how those estimates reached a particular conclusion using SHAP and compared with other ways to acquire feature importance.

For the third scenario, we have a dataset that contains home sales from Washington state in the United States (Anselin et al., 2003). We are trying to predict the price of homes with this dataset after being trained with a gradient boosting model (catboost). Afterwards, we give feature importance to analyze its interpretability.

After all the scenarios are analyzed, a general evaluation of the project is taken.

Chapter 2

Background

2.1 Machine Learning

Humans have been using Machine Learning that has vastly outperformed humans in certain tasks, such as predicting the weather or beating us in games (AlphaGo, Deep Blue). Machine Learning has been used as algorithms to make predictions based on given data. These algorithms recognize certain patterns in the data that we feed it to bring out certain insights. Supervised learning is one of the main branches of Machine learning, this branch in particular talks about learning from past data to predict new data that a machine learning model has not seen before. We usually get two types of problems in these supervised learning algorithms: regression and classification problems. If we predict a numerical value like an amount of money, it is called a regression problem. A classification problem involves predicting a particular category for unseen data. This data comes in what are called features. A given sample in the dataset contain features ($X = [x_0, x_1, ..., x_n]$) in which are mapped to the output of the model (y) (Figure 2.1). The algorithm is a function that wishes to approximate the real output of the trained data to predict new unseen data correctly ($f(x) \approx y$). The features in the data are the main focus of this project, particularly its importance in both local and global scope.

2.2 Origin of Shapley values

The origin of SHAP begins with economist and mathematician Lloyd Stowell Shapley, who contributed significantly to the field of cooperative game theory. He proposed what are called Shapley Values (Shapley et al., 1988). There is a cooperative game among some players, where

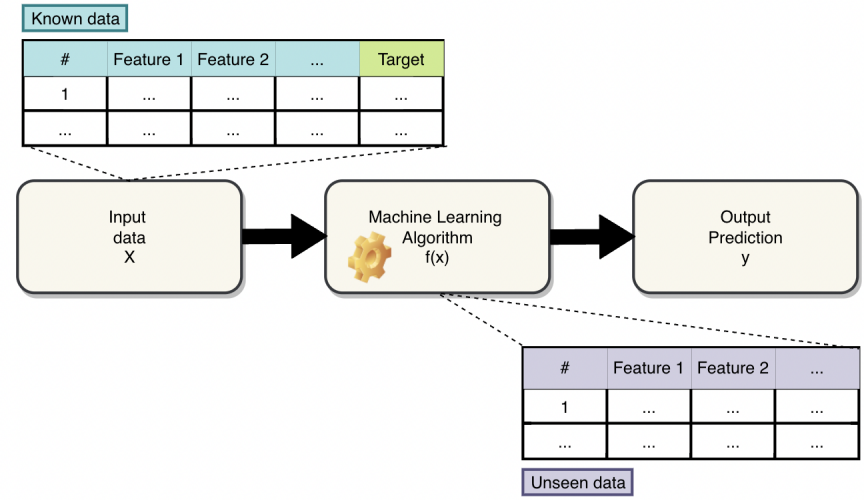


Figure 2.1: Supervised Machine Learning

each player contributes to the game outcome, also called the payout. The game's outcome is compared to each of the player's contributions. To calculate the fair distribution of importance values of the game, Shapley values are calculated by comparing all outcomes and all possible arrangements of players in a game.

Shapley values can be easily represented as a game with M players. We start with the full set of players F (Equation 2.1). Assume p_M is the M^{th} player in the game. Note that a coalition S is defined as a set in this paper. The subset of a coalition F is S such that $S \subset F$.

$$F = \{p_1, p_2, \ldots, p_{M-1}, p_M\} \tag{2.1}$$

All possible coalitions S of $F = \{p_1, p_2, p_3\}$ can be shown as (Equation 2.2). Note that the subset contains also an empty set $\emptyset$:

$$S = \{\emptyset, \{p_1\}, \{p_2\}, \{p_3\}, \{p_1, p_2\}, \{p_1, p_3\}, \{p_2, p_3\}, \{p_1, p_2, p_3\}\} \tag{2.2}$$

To find the contribution of a game, we define a function val; this function returns the collective payoff of the players that participated in that game, with the function's input being the set of players present. We can assume that $val(\emptyset) = 0$ at all times during the game.

However, we do not want the collective payoff of the players. What we want is each player's contribution to the collective payoff. We begin with an empty set S, and sequentially, we begin adding new players to the coalition, which must increase the payoff ($val(S)$) as we add new players. The calculation continues until there are no more players to add to the game.

To find the contribution of player p_x in a given game, we subtract the payoff of the previous coalition before including p_x to the payoff of the current coalition, which includes p_x (Equation 2.3):

$$val(S \cup p_x) - val(S) \tag{2.3}$$

A simple example of this could be using three players in the form of $F = \{p_1, p_2, p_3\}$. We receive a coalition of $S = \{p_2, p_3, p_1\}$. Assuming we want the contribution of $\{p_3\}$ of that coalition, we get the following Equation 2.4:

$$val(\{p_2, p_3\}) - val(\{p_2\}) \tag{2.4}$$

We add the final player p_1 at the end; however, it makes sense that it does not affect the player's individual contribution p_3 after already being added into the coalition. The main idea of this is that a player's contribution is calculated by subtracting an instance in the game in which the player is not in the game and an instance in which the player is present in that same game. The result in Equation 2.4 is only of the coalition shown for the given player p_3. In order to have a fair contribution value, the calculation used in 2.4 should be used for all possible permutations of the super-set F, as well as for each player, that is, all possible order arrangements for the players in the game, wherein this example permutation $[p_1, p_2, p_3]$, p_1 is the first player to be added to the coalition, and sequentially, p_3 is the last player to be added to the coalition.

The Shapley value of player p_i, denoted as ϕ_i, is the difference between the payoff of the coalition before adding p_i ($val(S)$) and the coalition including p_i ($val(S \cup \{p_i\})$), added along with all other possible permutations of the super-set F. Finally, the contribution value is averaged by dividing the number of possible permutations, ($|F|!$), as shown on Equation 2.5. Note that $|F|$ returns the number of players inside the super-set and is not an absolute operation.

$$\phi_i = \frac{1}{|F|!} \sum_{Permutations} (val(S \cup \{p_i\}) - val(S)) \tag{2.5}$$

It is shown in Table 2.1 an example for calculating the Shapley value of player p_3 with three players in the game. The greyed coloured rows of Table 2.1 show a waste of repeated calculations that increase as the number of players in the game increases, increasing the number of permutations of the super-set. Given M players, in the super-set$F$, the number of possible permutations is $M!$. Therefore in practice, calculating the Shapley values using Equation 2.5

Permutations of F	$val(S \cup \{i\})$	$val(S)$	$val(S \cup \{i\}) - val(S)$
$[p_1, p_2, p_3]$	$val(p_1, p_2, p_3)$	$val(p_1, p_2)$	$val(p_1, p_2, p_3) - val(p_1, p_2)$
$[p_2, p_1, p_3]$	$val(p_1, p_2, p_3)$	$val(p_1, p_2)$	$val(p_1, p_2, p_3) - val(p_1, p_2)$
$[p_1, p_3, p_2]$	$val(p_1, p_3)$	$val(p_1)$	$val(p_1, p_3) - val(p_1)$
$[p_2, p_3, p_1]$	$val(p_2, p_3)$	$val(p_2)$	$val(p_2, p_3) - val(p_1)$
$[p_3, p_1, p_2]$	$val(p_3)$	$val(\emptyset)$	$val(p_3)$
$[p_3, p_2, p_1]$	$val(p_3)$	$val(\emptyset)$	$val(p_3)$

Table 2.1: Showing permutations for calculating the Shapley value of p_3 given players $F = [p_1, p_2, p_3]$.

is inefficient.

However, we can further improve on Equation 2.5 by multiplying the distinct payoffs with its similar occurrences, such as permutations $[p_3, p_1, p_2]$ and $[p_3, p_2, p_1]$ shown in Table 2.1. By slightly modifying Equation 2.5, we need to find the number of permutations that can be formed with each coalition. Using again the example from Table 2.1, we take our super-set $F = \{p_1, p_2, p_3\}$ excluding player p_3, denoted as $(F \backslash \{p_3\})$. We now see all possible subsets $S \subseteq F \backslash \{p_3\}$ in Equation 2.6:

$$S \subseteq \{\emptyset, \{p_1\}, \{p_2\}, \{p_1, p_2\}, \{p_2, p_1\}\} \tag{2.6}$$

Lets take for example $S = \{p_2\}$, we know that $|S|! = 1! = 1$ returns us the number of permutations of the subset mentioned ($[p_2]$). Furthermore, we add our excluded player, say, so, p_3, $S \cup \{p_3\}$. The rest of the set F has $|F| - |S| - 1 = 3 - 1 - 1 = 1$ remaining elements (which is, p_1). This means there are $(|F| - |S| - 1)!$ ways to add the rest of the players after having $S \cup \{p_i\}$. Finally, to get all the possible permutations of S, we need to calculate $|S|!(|F| - |S| - 1)!$. The example is shown below on Equation 2.7, $|S|!$ represents all possible combinations before adding the player p_i we are calculating the contribution for. In 2.7, the addition of that 1 represents p_i. Finally, $|F| - |S| - 1$ represents all possible combinations that can be computed after already having $S \cup \{p_i\}$ in the game:

$$Coalition \rightarrow S + \{p_i\} + F - S - \{p_i\} = F$$

$$Coalition \rightarrow p_2 + p_3 + p_1 = \{p_1, p_2, p_3\}$$

$$Permutation \rightarrow [p_2] + [p_3] + [p_1] = [p_2, p_3, p_1]$$

$$Num.Permutations_{p_i} \rightarrow (|S|! = 1) + 1 + ((|F| - |S| - 1)! = 1) = (|S|!(|F| - |S| - 1)! = 1) \quad (2.7)$$

After having calculated all possible combinations of a subset $S \subseteq F \backslash \{p_i\}$, these result is multiplied by the contribution of p_i given subset S, that is $val(S \cup \{p_i\}) - val(S)$. This process then is used for all possible subsets of F, and finally averaged by the total number of permuations $|F|!$. The improved equation of 2.5 can be shown in Equation 2.8, with a more efficient solution:

$$\phi_i = \frac{1}{|F|!} \sum_{S \subseteq F \backslash \{i\}} |S|!(|F| - |S| - 1)!(val(S \cup \{i\}) - val(S)) \quad (2.8)$$

The Equation 2.8 can be further rearranged with Equation 2.9

$$\phi_i = \sum_{S \subset F \backslash \{i\}} \frac{|S|!(|F| - |S| - 1)!}{|F|!} (val(S \cup \{i\}) - val(S)) \quad (2.9)$$

This shows a more efficient way to calculate the contribution of a single player p_i; however, it is still an expensive computation for a single player as the number of players in a game grow, even more, expensive by calculating for all M players. However expensive it may be, it can indeed represent a reliable contribution of every player who participates in a cooperative game.

2.3 Properties of Shapley values

Out of the feature attribution methods such as Layerwise Propagation (Bach et al., 2015), and Local Interpretable Model-agnostic Explanations (LIME) (Ribeiro et al., 2016), Shapley values contain many valuable properties worth looking at. These properties are Symmetry, Additivity, Dummy, and Efficiency (Molnar, 2020).

2.3.1 Additivity property

$$\phi_j + \phi_j^+ \quad (2.10)$$

Given a game with combined payouts, the total contribution of a feature j is demonstrated in Equation 2.10. If, for example, there is an ensemble model, then the prediction is based on component models that make up the prediction. In this case, the Shapley values of the overall model should be the addition of the Shapley values of all component models. More specifically,

with a random forest model, the sum of the Shapley values of all the component decision trees, then averaging it over all models, will give off the total contribution of the entire model.

2.3.2 Efficiency property

When each of the M players contributions are added together ($\sum_{i=1}^{M} \phi_i$), this amount is the same as the difference of the game outcome of that instance ($\hat{f}(x)$, of Equation 2.11, and the average of all game outcomes ($E(\hat{f}(x))$).

This means that Shapley values can be interpreted as the same units as the units of the target feature. For example, we have a model that predicts the medical insurance cost in dollars, given a dataset with features such as "age", "sex", "body mass index", and so forth. The Shapley values of any given feature would represent the amount in dollars in which it affects the prediction, negatively or positively.

For classification problems, the model's output usually has two or more classes depending on if it is a multi-class target feature. Shapley values can represent why a class was true or false in a given sample. Therefore, in these classification cases, Shapley values tend to represent values between -1 and 1, tilting the model's prediction towards whether a particular class is true or false.

$$\sum_{i=1}^{M} \phi_i = \hat{f}(x) - E(\hat{f}(X)) \tag{2.11}$$

2.3.3 Symmetry property

As the property name stated, Shapley values are symmetric in terms that, in a case where two features contribute equally in any given coalition. In other words, it gives proper fairness to each feature contribution. As Equation 2.12 demonstrates, if feature i and j provide the same contribution for a given coalition S that does not contain feature i and j, the Shapley values should be equal in any given prediction or payoff.

This can be further explained by using a small example using the SHAP python library (S. Lundberg, 2018). We have generated some regression samples using scikit-learn to test this property. As shown in Figure 2.2, the 0th and 1st features are copies of each other, therefore, giving the same contribution to each prediction. Now, calculating its Shapley values, we can see that indeed the contribution of features 0 and 1 are equal, given that they have the same values that contributed to the prediction.

$$\phi_i = \begin{cases} \phi_j, & \text{if } val(S \cup \{x_i\}) = val(S \cup \{x_j\}) \\ \not{\phi}_j, & \text{otherwise} \end{cases} \quad \text{for all } S \subseteq \{1, ..., k\} \backslash \{i, j\} \tag{2.12}$$

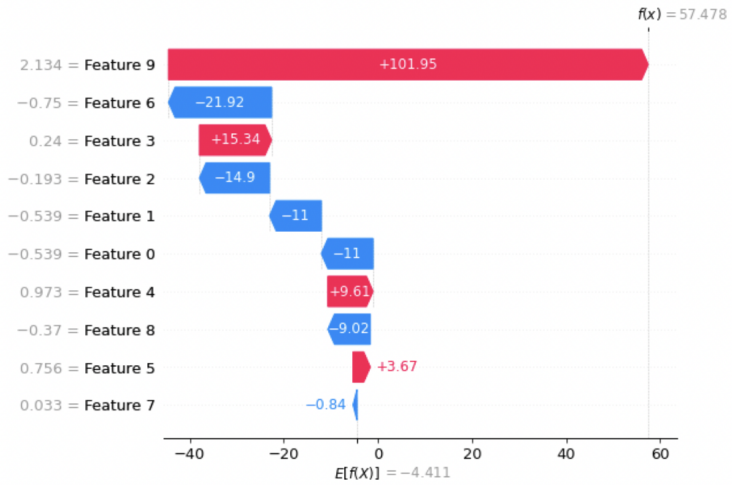


Figure 2.2: Forceplot with symmetric features 0 and 1

2.3.4 Dummy property

If a feature does not add any contribution to the prediction, then its Shapley value must be 0. That is, for every coalition S such that $val(S) = val(S \cup \{i\})$, that means that Shapley value provides no contribution whatsoever, $(\phi_i = 0)$.

$$\phi_i = \begin{cases} 0, & \text{if } val(S \cup \{x_i\}) = val(S) \\ \not{0}, & \text{otherwise} \end{cases} \tag{2.13}$$

All these favourable properties provide fairness in terms of each player's contribution that takes part in a game. In a way, these properties give us insight into how players are affecting the game payoff. This idea can give us use in the area of Machine Learning, as discussed further on.

2.4 From Game Theory to Machine Learning

Using the theory of Shapley values from cooperative game theory, Lundberg and Lee (S. M. Lundberg & Lee, 2017) introduced SHAP (SHapley Additive exPlanation) explanation methods.

A machine learning model's "learning" function $f(x)$ contains these parameters usually defined as feature inputs. From Shapley values, players in games can be thought of as features, and the machine learning model prediction can be thought of as the game's payoff.

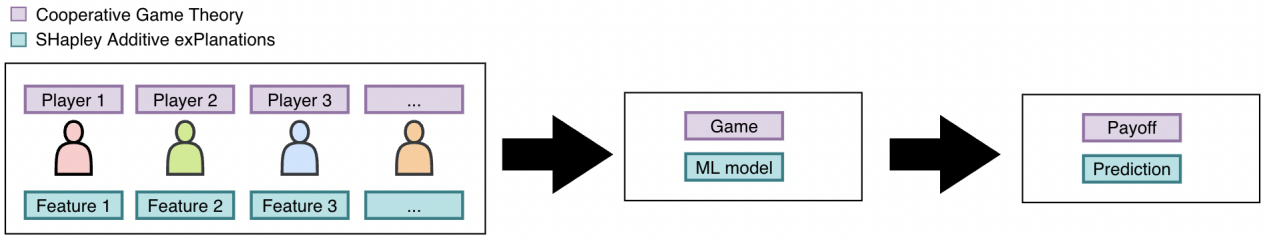


Figure 2.3: Comparing Shapley values used in Game theory and Machine Learning

So looking at Figure 2.3 we can think of a predicting model as a coalition game. In Cooperative Game Theory, we assume $val(\emptyset) = 0$, that is, when no players are present for the game, the payout is zero. Now, in machine learning, when no features are present, it is assumed we get the average estimate prediction over our n training samples, which we call the expected value ($E[f(x)]$). This transition from game theory to machine learning can be further shown in Equations 2.14 and 2.15.

$$val(\emptyset) = E[f(x)] - E[f(x)] = 0 \tag{2.14}$$

$$f(\emptyset) = E[f(x)] = \frac{1}{n}\sum_{i=0}^{n} f(x_i) \tag{2.15}$$

Furthermore, we need to calculate the worth of any coalition $S \subseteq F \backslash \{i\}$ to use the Shapley value Equation 2.9. We can assume that we can substitute their place with a $NULL$ value for any absent features in the input. Note that $NULL$ cannot be used in the actual input of the model, which is for demonstration purposes only. We have a super-set $F = \{x_1, x_2, x_3\}$ and our subset is $S = \{x_1, x_3\}$. We can display our coalition as $f(x_1, NULL, x_3)$; however, one needs to think about how we would apply f to a subset in practice. Here we can assume that the contribution of a coalition S is shown in Equation 2.16.

$$val(S) = f(x_1, NULL, x_3) - E[f(x)] \tag{2.16}$$

$$val(F) = f(x_1, x_2, x_3) - E[f(x)] \tag{2.17}$$

The way used to represent whether a feature is present or not is through the use of what is called simplified input vector x' (S. M. Lundberg & Lee, 2017). These vector values are binary, with 1 representing that the feature is present and 0 otherwise.

Lets take for example,

$$x = [x_1, NULL, x_3]$$

can be shown as simplified input vector

$$x' = [1, 0, 1]$$

Both these vectors values are mapped by a function h_x such that:

$$h_x(x') = x$$

One needs to think about how to exclude features from a Machine Learning model since a model requires all of its parameters to be in the input. In practice, the way "absent" features are represented is by replacing the absent value with a random feature value from the sampling data. The 1s in x' are mapped by h_x to its corresponding values from the instance it is on. For 0s, these are then retrieved from a random value in the overall sampling data.

The explainer methods used in the SHAP python documentation are represented as model $g(z')$ (S. M. Lundberg & Lee, 2017). g takes in as input a simplified input vector z' where $z'_i \in \{0, 1\}$. In difference with x', z' refers to any coalition vector $S \subseteq F$, whereas x' refers to the the super-set.

The explainer model g tries to approximate the predictions made by the learning function f. To do this, we make use of mapping function h_x. $h_x(z')$ should return the actual features values and the values that are missing as $NULL$. We can say that $g(z') \approx f(h_x(z'))$ whenever there is a z' that is close to x'.

Having all these functions, we can compare to the Shapley value Equation 2.9 and further adapt it to model predictions. Similarly to the Shapley value Equation 2.9, we get the Shapley value ϕ_i calculating the marginal contribution of z' and $z' \backslash \{i\}$ for all subsets of simplified feature input x' over all M features of the dataset in use. A minimal difference in the adapted Equation 2.18 (from (S. M. Lundberg & Lee, 2017)) is that $|z'|$ returns the amount of 1s in the vector. Instead of using $S \cup i$, z' is used assuming feature $\{i\}$ is present in the simplified input vector. Similarly, $z' \backslash \{i\}$ is used instead of S, representing the coalition that does not contain

feature $\{i\}$. Moreover, f_x is represented as a function to calculate the marginal contribution of the features present in z':

$$f_x(z') = f(h_x(z'))$$

An example of the representation of f_x is as follows:

$$f_x(z') = f_x([1,1,0]) = f(x_1, x_2, NULL)$$

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!}[f_x(z') - f_x(z' \backslash \{i\})] \tag{2.18}$$

To better explain this process a complete example of one coalition is shown next:

$$x = \{x_1, x_2, x_3, x_4\}$$

$$i = x_3$$

$$x' = [1,1,1,1]$$

$$z' = [1,1,1,0]$$

$$z' \backslash i = [1,1,0,0]$$

$$\phi_i(f,x) = \sum_{z' \subseteq x'} \frac{3!(4-3-1)!}{4!}[f_x([1,1,1,0]) - f_x([1,1,0,0])]$$

This final equation (2.18) can calculate the exact Shapley values for every feature value, therefore explaining the prediction of a given machine learning problem. SHAP values are part of a class of methods called Additive Feature Attribution methods, some of which were previously mentioned, such as LIME.

2.5 Additive feature attribution properties

It is believed that SHAP values are the only set of values that satisfy all desirable properties mentioned here: Local Accuracy, Missingness, and Consistency.

2.5.1 Local Accuracy

One of the properties of SHAP values is that the prediction of g given simplified input vector x' matches the original prediction made by $f(x)$. This can be proven by adding all of the available (hence the x' in the summation part of Equation 2.19) Shapley values and the expected value $E[f(x)]$.

$$f(x) = g(x') = E[f(x)] + \sum_{i=0}^{M} \phi_i x'_i \tag{2.19}$$

This Equation 2.19 is similar to the Efficiency property from Shapley values. The only difference is that the Efficiency property has all the simplified input vector x' as all present (all 1s), namely representing the whole coalition.

2.5.2 Missingness

Any missing feature, that is, not "present" in the input, should have a Shapley value ϕ_i of zero. For the feature's Shapley value calculated, if that same feature is not present in x', then ϕ_i is zero. Furthermore, if $x'_i = 0$, then z'_i would be zero as well, making the whole calculation be zero as shown below:

$$\phi_i = \sum_{z' \subseteq x'} \frac{|z'|!(M - |z'| - 1)!}{M!} f_x(z') - f_x(z') = 0$$

$$x'_i = 0 \rightarrow z'_i = 0 \rightarrow \phi_i = 0 \tag{2.20}$$

2.5.3 Consistency

Having a different model f' from the original f with plans to increase the contribution of a feature will never decrease the Shapley value of that feature.

$$\phi_i(f', x) \geq \phi_i(f, x) \tag{2.21}$$

2.6 The Importance of Explainability

Accuracy is not the only factor worth considering when developing ML models. Many projects (particularly in analytics) rely on Explainable AI as a critical component to support model transparency. Various approaches are being proposed and adopted by Data Scientists to explain

Machine Learning models. Considering how many companies have a high demand to adopt these methods to make optimal and trustworthy business decisions. These companies vary from the finance, healthcare, investment sectors, and many others. From cancer detection in patients from diagnosis data to managing investments, the use cases for these methods are unbounded. Understanding the "how" of predictions in black-box machine learning models is necessary for better insight and to establish trust with all actors involved in the development and deployment of the product. Bias detection is a big area to be concerned about, depending on the problem. SHAP is a tool that can provide general insight into this issue, as it may serve as a debugger that can help us detect unwanted bias in the model being developed.

Chapter 3

Materials & Methods

Since this is primarily a data science and machine learning project, the programming language used for this project is the python language. Python is a popular programming language used widely in data science as it contains popular libraries that will be used, such as NumPy, pandas, scikit-learn, and matplotlib. It also happens to be that the shap library S. Lundberg (2018) is implemented as a python library, so it is convenient already to use python.

Jupyter notebooks, provided by Anaconda distribution, were used to implement and visualise this project's problem cases. Jupyter notebooks is a great software for visualising the code written during the project implementation stage.

There are many interpretations of the process stages when conducting a data analysis project, even though most of them end up in a similar state. These systematic processes are used to analyse, visualise and model any given data, depending on the project. The processes used in this project are shown in Figure 3.1. Having processing stages tends to help yield better results, having a more organised and productive architecture of the tasks that need to be taken into account.

The first stage is defining the problem, that is, planting what the problem entails, the goals, expectations, tools, and what can and cannot be done given a time frame. In this project, the primary source of inspiration for building our analysis cases came from the paper that introduced SHAP values (S. M. Lundberg & Lee, 2017). After extracting the information and seeing the method's potential in black-box model explainability, three dataset scenarios were planned to test the properties of SHAP values and see their value to help understand the model better for debugging or finding features contributions.

Secondly, after coming out with the requirements described in the data collection stage,

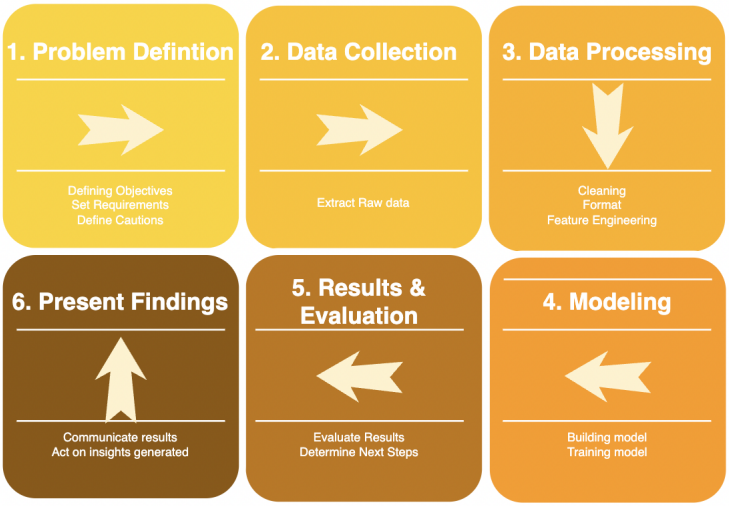


Figure 3.1: Data Analysis process stages

data that fits the problem must be extracted. Data sets were collected from two sources; the first two cases were collected from the UCI machine learning repository and the third one from GeoDa (Anselin et al., 2003) which is an open software tool where spatial data can be extracted.

The subsequent processing stage was data processing. This process involves cleaning the data, configuring the format, and getting rid of irregularities in the data. Feature engineering, which is the process of creating or deleting new features to improve the model's performance, is usually done in this stage. However, this stage was not thoroughly done for the test cases, as it felt out of scope from the project's main objective, which was feature importance after training the model on any given data. This process is mainly concerned with the model's performance and accuracy and should be taken seriously in practice. More of this is mentioned in the Results/Evaluation section.

For the modelling process, a particular machine learning model was chosen depending on the problem facing. The models were trained and assessed; after being assessed, depending on the performance, it could be taken to data processing to improve the model's performance further. If the model has acceptable performance, the model is taken to the following processing stage to be evaluated.

Results are then analyzed and evaluated to figure out how to proceed. The results could show some faults that could start from the beginning of the process stages. This stage is the most important from the project, as it analyzes the Shapley values computed from the trained model and data set. Feature importance is observed along with any other form of interpretability used in this project.

Finally, the last stage is communicating one's findings and results from the analysis project. In these circumstances, the findings presented are written in this project and jupyter notebooks from the test cases in the source code provided along with the project.

Chapter 4

Data Analysis & Findings

We can see by the SHAP Equation 2.18 that this is computationally expensive were there to be a considerable amount of features. The total number of subsets is 2^n, where n is the number of features in the set. That means that for every single feature value, there would be 2^n calculations being made to calculate the Shapley value of a single feature value. Now thanks to the SHAP python implementation by Lundberg and Lee (S. Lundberg, 2018), we can approximate Shapley values without calculating all possible combinations.

4.1 Case 1: Predicting Acidity in Wine samples (Regression)

A regression problem using a Linear Regression model is presented in this Test Case. It is helpful to explain how SHAP values work with simple models and compare them to this highly interpretable model.

The dataset used for this test case originates from UCI machine learning repository (Cortez et al., 2009). The goal was set to predict the fixed acidity of the wine samples in the dataset. Acidity contributes significantly to wine's taste. As seen in Figure 4.1, the data shows its features along with some descriptive statistics about its distribution. These are 12 features of red and white wine variants of the Portuguese "Vinho Verde" wine, with the fixed acidity being the target feature (values ranging from 4.6 to 15.9 ml/L).

Linear models can be interpreted by examining their model coefficients (β_p, also known as weights) for each feature, which tells us how the prediction can change by modifying each input feature. For reference, the learning function is shown below, where β_0 is the intercept, β_p are

	mean	std	min	25%	50%	75%	max
fixed acidity	8.319637	1.741096	4.60000	7.1000	7.90000	9.200000	15.90000
volatile acidity	0.527821	0.179060	0.12000	0.3900	0.52000	0.640000	1.58000
citric acid	0.270976	0.194801	0.00000	0.0900	0.26000	0.420000	1.00000
residual sugar	2.538806	1.409928	0.90000	1.9000	2.20000	2.600000	15.50000
chlorides	0.087467	0.047065	0.01200	0.0700	0.07900	0.090000	0.61100
free sulfur dioxide	15.874922	10.460157	1.00000	7.0000	14.00000	21.000000	72.00000
total sulfur dioxide	46.467792	32.895324	6.00000	22.0000	38.00000	62.000000	289.00000
density	0.996747	0.001887	0.99007	0.9956	0.99675	0.997835	1.00369
pH	3.311113	0.154386	2.74000	3.2100	3.31000	3.400000	4.01000
sulphates	0.658149	0.169507	0.33000	0.5500	0.62000	0.730000	2.00000
alcohol	10.422983	1.065668	8.40000	9.5000	10.20000	11.100000	14.90000
quality	5.636023	0.807569	3.00000	5.0000	6.00000	6.000000	8.00000

Figure 4.1: Feature Statistics of Wine dataset

the coefficients, ϵ is the error (difference between prediction and actual value), and x_p is the feature input (Equation 4.1). The coefficient values of the trained model are shown in Table 4.1.

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon \tag{4.1}$$

Feature	**Coefficient**
Volatile Acidity	0.2319
Citric Acid	1.9631
Residual Sugar	−0.2458
Chlorides	−3.6441
Free Sulfur Dioxide	0.0094
Total Sulfur Dioxide	−0.0067
Density	647.8952
pH	−5.2405
Sulphates	−0.7793
Alcohol	0.5345
Quality	0.04

Table 4.1: Linear Model's Coefficients

These coefficient values provide good insight into how each input feature may affect the prediction; however, it does not mean it measures the importance of a feature (Molnar, 2020). The more non-linear the problem is, the less accurate the model will become, as well as the coefficients calculated per feature. The coefficient dependency is related to the input scale, not its importance. If a measure were to be changed to different metrics(millilitres to litters), its coefficient value would dramatically change in the linear model. Looking at these statements, Shapley values are worth looking over as an alternative for model interpretability. The linear model coefficients only scale the feature values and do not provide a meaningful value regarding the target unit of value. Thanks to the Efficiency property, Shapley values are demonstrated in the target's scale; for example, if one were predicting the dollar price of houses, Shapley value would represent the feature's contribution in dollars.

We can visualize a partial dependency plot (PDP) to show feature importance. A partial dependence plot shows the relationship between the target and a feature in particular. As shown in Figure 4.2, the target value given the feature is shown on the y-axis, while the feature value is shown on the x-axis. In this case, the feature 'density' is chosen for demonstration purposes. Since this is a linear model, it is expected to show a clear linear relationship. The grey horizontal line ($E[f(x)]$) represents the expected value of the model, while the vertical line (E[density]) represents the feature's average value. Additionally, a histogram of the feature value distribution is shown in the lower area of the x-axis.

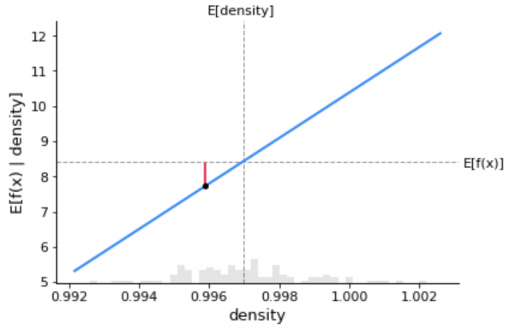


Figure 4.2: Partial Dependency Plot of "Density" feature

SHAP values can be easily observed in partial dependence plots thanks to the model's linearity. The SHAP value of a feature instance is the difference between the expected model prediction (E[f(x)]) and the feature value (density in this case). The SHAP value is shown in the black sample dot, with the red line being the difference between the expected overall prediction and the feature's value. Similarly, plotting all the SHAP values of that particular feature correspond with the partial dependence plotline as shown in Figure 4.3, which can clearly show the linear relation of the model.

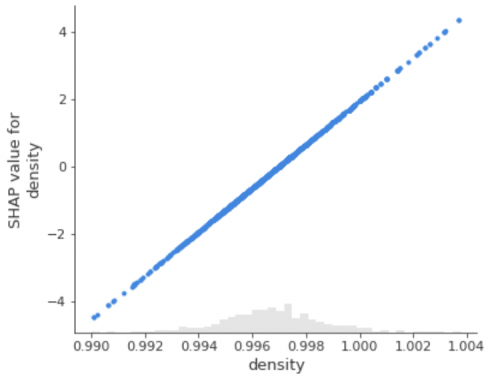


Figure 4.3: SHAP values of all density feature values

4.1.1 Local Scope

SHAP has an additive nature; that is, SHAP values for a particular instance can be summed up to be the difference between the expected model prediction ($E[f(x)]$) and the prediction being made ($f(x)$). One of the most transparent ways to visualize the explanation of a local prediction is through the use of the waterfall plot. As shown in Figure 4.4, this plot shows the feature importance of a particular instance (10th observation) in the dataset. The blue label means the feature SHAP value negatively affects the model's prediction, and the red labels do the opposite. In this particular prediction, we can see that the feature "density" and "alcohol" has the most contribution to the observation's prediction. This prediction was calculated using the '$LinearExplainer()$' in the SHAP library (S. Lundberg, 2018).

Remember that the calculated prediction is the sum of the j Shapley values of the given instance and the average over all predicted results ($E[f(x)]$):

$$f(x) = E[f(x)] + \sum_{i=1}^{j} \phi_i$$

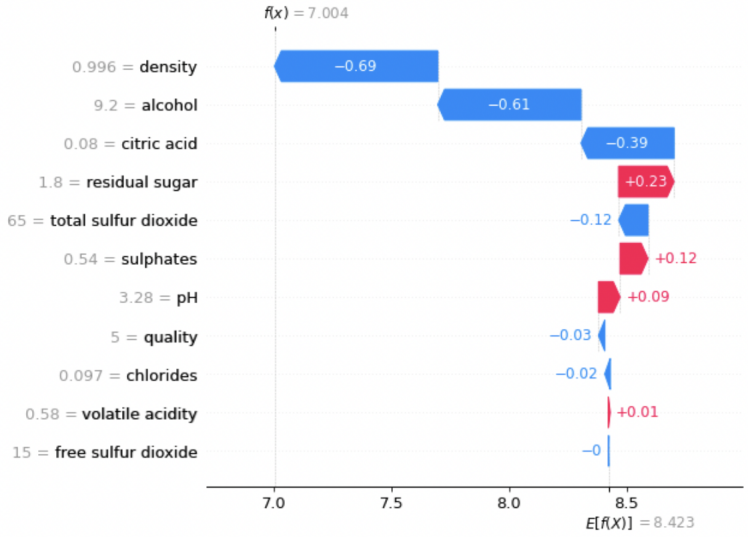


Figure 4.4: Waterfall plot of instance number 10

4.1.2 Linear Explainer

Since this is using a Linear model, Shapley values can be easily calculated since all we need to know is the difference between the feature's average value and the feature's value in the linear equation, k being the number of the data set samples. However, note that Equation 4.2 represents the calculation of Shapley values only for Linear models. To calculate the SHAP

values for model-agnostic explainers, we have to use the SHAP equation 2.18 and calculate for all possible coalitions, which was previously explained in the background section.

$$\phi_i = \beta_i x_i - \beta_i \frac{1}{k} \sum_{j=1}^{k} x_i^{(j)} \tag{4.2}$$

4.1.3 Global Scope

The "beeswarm" plot can be used to summarize all the SHAP values of all instances in the dataset. In this "beeswarm" Figure 4.5, we plot the distribution of SHAP values for every single feature. If feature value is high, it will be displayed in the colours red and blue otherwise. An example of how this can be interpreted is as follows: as the density increases, the fixed acidity is more likely to be higher, and as the pH values increase, it is likely to have lower fixed acidity compared to the average fixed acidity over all training samples.

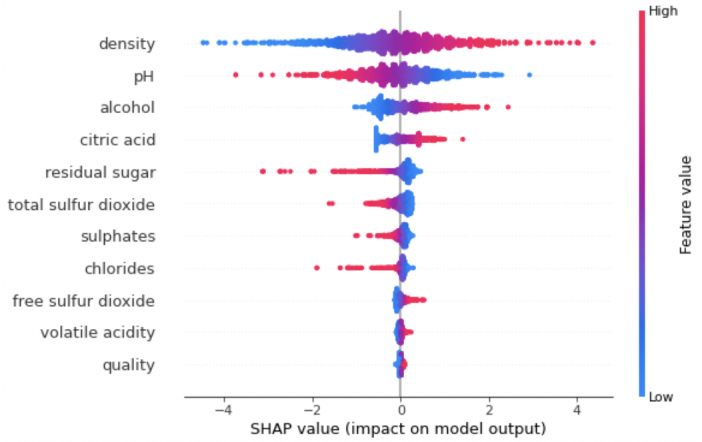


Figure 4.5: Beeswarm plot of all feature values

We can also visualize the data by a simple plot bar containing the absolute mean of the SHAP values per feature (Figure 4.6). This plot shows the overall absolute impact (negative or positive) on the model's predictions. This plot highlights the features that affect the most in the overall model, whether they affect negatively or positively. This plot is preferred if the objective at hand is to spot the most or least impactful features.

4.2 Case 2: Predicting Obesity Class (Classification)

For this case, we will use a decision tree algorithm along with a classification problem. The dataset used in this case comes from the UCI Machine Learning Repository (Palechor & de la Hoz Manotas, 2019), containing 2,111 samples. This data contains the eating habits and physi-

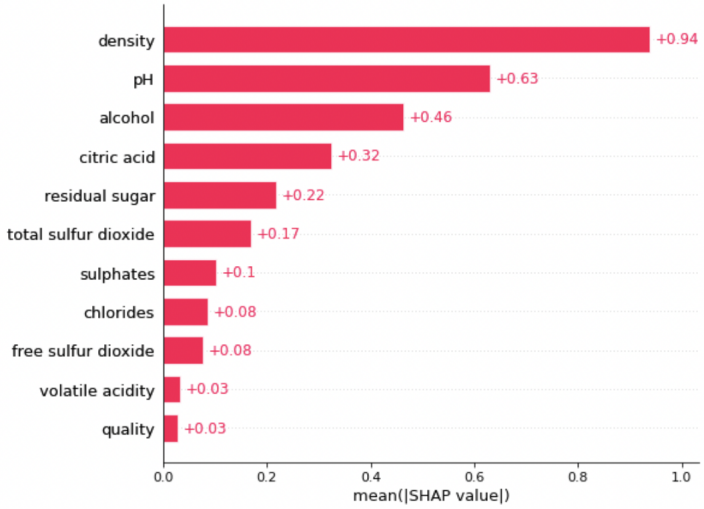


Figure 4.6: Bar plot of all feature values (SHAP absolute values)

cal conditions of individuals from Mexico, Peru, and Colombia. The target of this classification problem is to predict which Obesity class an individual belongs to. The target contains 7 classes (Table 4.2). There are various feature values of different types in the dataset (Table 4.3), categorical, continuous and discrete.

No.	Label encoding	Category
1	0	Insufficient_Weight
2	1	Normal_Weight
3	2	Obesity_Type_I
4	3	Obesity_Type_II
5	4	Obesity_Type_III
6	5	Overweight_Level_I
7	6	Overweight_Level_II

Table 4.2: "NObesity" Target Classes

After Extracting the dataset from the UCI repository, each feature is analyzed for its data type and relevance in terms of the target (Obesity class). The height and weight are eliminated entirely due to their high correlation to the BMI feature (body mass index).

For the next stage, we separate our numerical features from our categorical features. For the categorical features, encoders are used as some machine learning models do not work well with categorical features. The encoders map a feature class to a discrete number to represent the feature class. Since we have binary class features (i.e. yes/no) and ordinal class features (i.e. never/sometimes/always), encoders provided from the scikit-learn python library are used.

The data is now ready to be used, so we split the data set into a training set and a test set. After fitting the training set with a Decision Tree Classifier, it shows an accuracy score of

#	Feature Name	Variable Type	Description	Data Type
1	Gender	Categorical	Male or Female	string
2	Age	Numerical	Age in years	float
3	Height	Numerical	Height in meters	float
4	Weight	Numerical	Weight in kilograms	float
5	Family history with overweight	Categorical	Yes or No	string
6	FAVC	Categorical	Yes or No, frequent consumption of high caloric food	string
7	FCVC	Numerical	Frequency of consumption of vegetables	float
8	NCP	Numerical	Number of main meals	float
9	CAEC	Categorical	Consumption of food between meals	string
10	SMOKE	Categorical	Yes or No	string
11	CH20	Numerical	Consumption of water daily	float
12	SCC	Categorical	Yes or no, calories consumption monitoring	string
13	FAF	Numerical	Frequency of physical activity	float
14	TUE	Numerical	Time using technology deviced	float
15	CALC	Categorical	Consumption of alcohol	string
16	MTRANS	Categorical	Type of transportation used	string
17	NOeyesdad	Categorical	Obesity class based on body mass index (BMI)	string

Table 4.3: Dataset Features before processing

74.5%. Furthermore, the data was tested using a confusion matrix (Figure 4.7), which allows visualizing its performance. The confusion matrix seems to have the data well distributed and with not many outlier errors. Additionally, from Figure 4.7, it seems to have good accuracy as highest heat values correspond with both the same true values and predicted values. This accuracy score could be further improved by continuing to modify the dataset and maybe even change the machine learning model. However, the main focus of this project is not on the model accuracy aspect of the project but on its feature explainability, which could gain insight into further improving the model's performance and transparency.

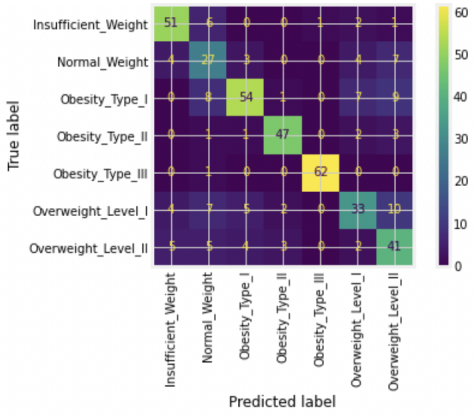


Figure 4.7: Confusion matrix of Obesity dataset

4.2.1 Local Scope

Shapley values work slightly different with classification problems. For clarity and simplicity, in this section, we use sample number 5 from the dataset as an example, and its Shapley values will be computed. As shown in Figure 4.8, the first row shows the original values along with their categorical values. The second row shows the instance after encoders have transformed it in order to make the categorical values into numerical. To explain the sample, the Tree Explainer function was used to calculate the Shapley values of the Decision Tree Classifier. The Tree Explainer returns the Shapley values of the dataset for every single target class (shown in Figure 4.8, starting from the third row), that is $shap\_values.shape(2111, 14, 7)$, $2,111$ being the number of samples in the dataset, 14 being the number of features and 7 being the number of classes. In Figure 4.8, the target value for the given observation is "Normal_Weight", which is why it is the only row with a value of 1 in the Predicted column. We have the Shapley values that show their contribution to getting or not getting the outcome of a specific class.

	Gender	Age	family_hist...	FAVC	FCVC	NCP	CAEC	SMOKE	CH2O	SCC	FAF	TUE	CALC	MTRANS	Predicted
No Encoding	Male	29.000000	no	yes	2.000000	3.000000	Sometimes	no	2.000000	no	0.000000	0.000000	Sometimes	Automobile	NaN
Encoded	1	29.000000	0	1	2.000000	3.000000	2	0	2.000000	0	0.000000	0.000000	2	0	NaN
Insufficient_Weight	0.0195	-0.111833	0.049333	0.002	-0.044667	-0.017667	-0.005833	0.0025	0.002833	0.002833	0.001000	-0.022000	-0.001	0.023	0.0
Normal_Weight	0.123619	-0.026869	0.357821	0.023369	0.225607	0.012536	0.002619	-0.000833	0.024262	0.008429	-0.005988	0.159821	-0.000405	-0.013988	1.0
Obesity_Type_I	-0.010341	-0.015306	-0.04471	0.003726	-0.015417	-0.006103	0.002516	-0.001667	-0.027234	-0.0025	0.007373	0.007468	-0.005972	-0.001833	0.0
Obesity_Type_II	0.092369	0.042464	-0.036536	0.0	-0.187036	0.016298	-0.0005	0.0	-0.022536	0.0	0.004798	-0.012833	-0.002726	-0.023762	0.0
Obesity_Type_III	-0.074167	0.004167	0.0	0.000833	-0.076667	0.000000	0.0025	0.0	0.000000	0.0	0.000000	-0.076667	0.0	0.0	0.0
Overweight_Level_I	-0.179123	0.076246	-0.009921	0.000905	-0.024964	0.013353	-0.003206	0.0	-0.009861	-0.008667	-0.007575	-0.048718	0.014925	0.016607	0.0
Overweight_Level_II	0.028143	0.031131	-0.315988	-0.030833	0.123143	-0.018417	0.001905	0.0	0.032536	-0.000095	0.000393	-0.007071	-0.004821	-0.000024	0.0

Figure 4.8: Feature and Shapley values of the 5th sample

On the waterfall plots below (Figure 4.9), we are showing the Shapley values for two target classes, "Normal_Weight" being the predicted value, and "Insufficient_Weight" being one of the false target values. The predicted class of sample number 5 is "Normal_Weight", which is why we can see in 4.9 (a) that $f(x) = 1$ and 4.9 (b) is $f(x) = 0$. So we can see in 4.9 (a) that not having a family history with overweight and having a high frequency of vegetable consumption are one of the most important factors which classified this individual to be "Normal_Weight".

4.2.2 Global Scope

A way to interpret a decision tree algorithm could be using the Gini importance. Gini importance is a way to calculate each feature's importance by calculating the total decrease in node impurity (a measure of homogeneity of the labels at a node in the decision tree). There are some drawbacks to Ginis importance: features can be selected as more critical if they have high cardinality when splitting through branches in the decision trees.

(a) Normal_Weight target class (b) Insufficient_Weight target class

Figure 4.9: Waterfall plots of 5th sample

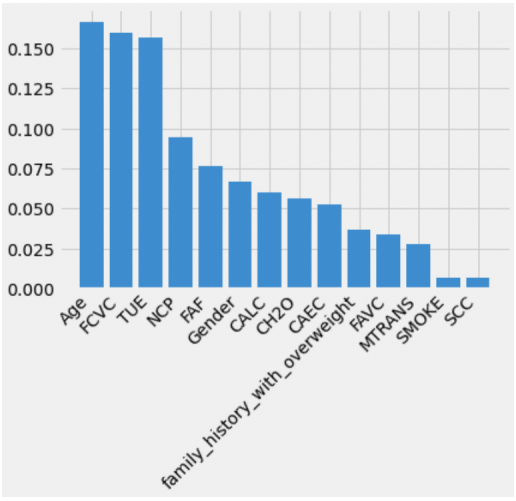


Figure 4.10: Gini importance of decision tree

It is shown in Figure 4.11 in a summary plot that represents all the mean absolute Shapley values for all target classes. We can see that the frequency of vegetable consumption was the most important feature when deciding the Obesity class on a given sample. Meanwhile, whether an individual smoked or not, did not matter much when classifying an individual for obesity.

In Figure 4.12, there is a beeswarm plot of the shap values given the highest obesity class: "Overweight_Level_II". We can observe that, for example, if vegetable frequency consumption is high (FCVC), it is improbable that the individual is this class of overweight. Similarly, looking at a low family history of overweight tends to make it less likely for the individual to be overweight.

4.3 Case 3: Predicting House Prices (Regression)

In this analysis case, we are looking at a house price data set from King County, located in the state of Washington, United States (Anselin et al., 2003). The target is to predict the price of a house given its feature values (Table 4.4). We are going to be using a Gradient Boosting

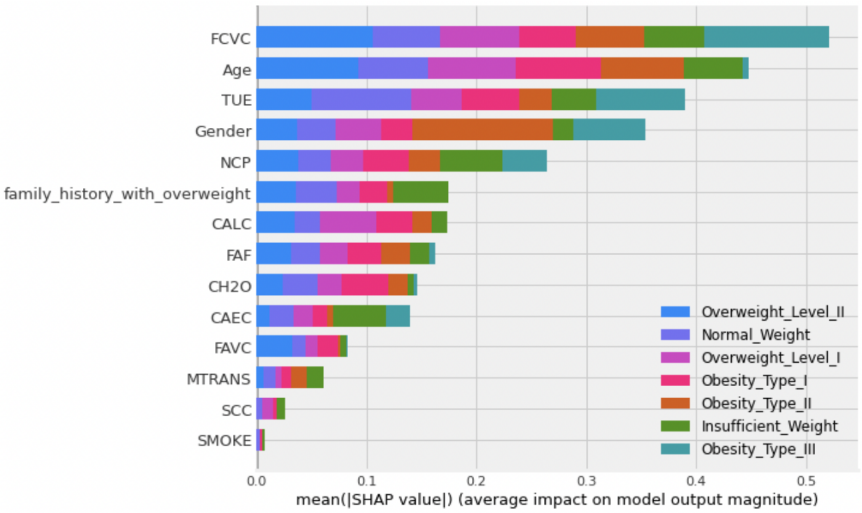


Figure 4.11: Shap values of all target classes

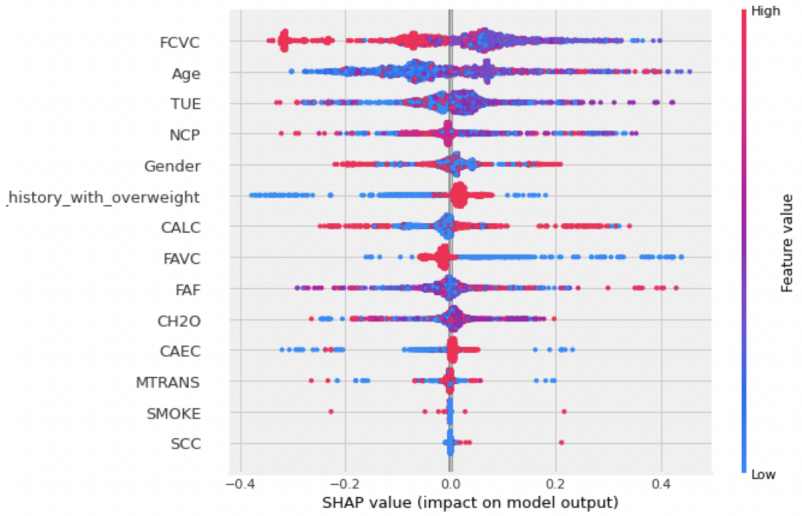


Figure 4.12: Beeswarm plot for target class "Overweight_Level_II"

Regression model, particularly a Catboost Regressor model (Prokhorenkova et al., 2018). The features are all numerical, so there will not be any need to encode the data as with the last case scenario. The dataset contains 21,613 samples for our model for testing and training.

Since Catboost is an algorithm for gradient boosting on decision trees, we use the Tree-Explainer() method from the SHAP library. The TreeExplainer() may not be model-agnostic; however, it has a more efficient algorithm for decision tree models. The SHAP model-agnostic Explainer() could still calculate the Shap values; however, it would be computationally expensive compared to the Tree Explainer. The TreeExplainer() method can be used with other models such as XGBoost, LightGBM, and most of the sci-kit learn algorithms that are supported by decision trees.

#	Feature Name	Variable Type	Description	Data Type
1	price	Numerical	Sale price (Target)	float
2	sqft_living	Numerical	Size of living area in square feet	float
3	sqft_above	Numerical	Square feet above ground	float
4	sqft_basement	Numerical	Square feet below ground	float
5	sqft_living15	Numerical	Avg size of interior housing living space for the closest 15 houses, in square feet	float
6	sqft_lot	Numerical	Size of the lot in square feet	float
7	sqft_lot15	Numerical	Average size of land lots for the closest 15 houses, in square feet	float
8	yr_built	Numerical	Year built	float
9	yr_renovated	Numerical	Year renovated. '0' if never renovated	integer
10	lat	Numerical	Latitude	float
11	long	Numerical	Longitude	float
12	grade	Categorical	Classification by construction quality. Better quality mean "high grade"	integer
13	bathrooms	Numerical	Number of bathrooms	integer
14	bedrooms	Numerical	Number of bedrooms	integer
15	view	Categorical	0 to 4 of how good the view of the property is	integer
16	floors	Numerical	Number of floors	integer
17	waterfront	Categorical	property has a waterfront. 1 yes, 0 no	integer
18	condition	Categorical	Condition of the house, ranked from 1 to 5	integer
19	zipcode	Numerical	5 digit zipcode	integer

Table 4.4: Feature descriptions of house price dataset

4.3.1 Model Evaluation

Getting rid of outlier occurrences in the dataset to ensure that the rest of the dataset is well represented. Not taking care of outliers could lead to severe anomalies in the model's performance. Irregularities were analyzed by computing the histogram of each variable to compare their value distribution. A few samples were eliminated from the dataset, such as a house worth 7.7 million dollars, whereas most of the houses in the dataset contained prices in the range of 75 thousand dollars to 2.5 million. Given a large number of occurrences, it does not have a negative effect on getting rid of those outlier samples.

After training our CatBoost Regressor with the data, evaluation was carried using some of the main evaluation methods for regression models, which include: Mean Absolute Error (MAE), Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and R^2 score (Table 4.5). MAE give us an indication of how far off our predictions are on average ($MAE = \frac{1}{k}\sum_{i=1}^{k}|y_i - y_i\_pred|$). Similar variations follow with MSE and RMSE, amplifying the larger error differences in the test set. The values for these metrics are quite large given how big our price range is with such large amount of samples of houses. The R^2 score is somewhat similar to accuracy testing, the closer the R^2 score is to 1, the better the model (represents the

proportion of variance across test samples) ($R^2 = 1 - \frac{\sum_{i=1}^{k}(y_i - y_i\_pred)^2}{\sum_{i=1}^{k}(y_i - avg(y))^2)}$).

Metrics	Score
MAE	67,378.75
MSE	12,493,686,073.66
RMSE	111,775.16
R^2	0.897

Table 4.5: Evaluation Metrics for CatBoost model using the test data

Table 4.5 shows a good R^2 score, which can be further improved by dealing with feature engineering (further feature selection, transformations, and more). However, taking a closer look into feature engineering is out of this project's scope.

4.3.2 Local Scope

The sample #10 from the dataset is illustrated for demonstration purposes. The sample contains a house in which the true price is $y_{10} = \$662,500$. The model explainer predicted the house price to be at $f(x_{10}) = \$691,137$. We can see that the size of the living area is the biggest attribute contribution to the local prediction since it is substantially greater than the average size of the living area, which is $E[\text{sqft\_living}] = 2076.36$ square feet (Figure 4.13). The rest of the features not shown were mostly insignificant in contributing to the sample and were not needed for the visualization.

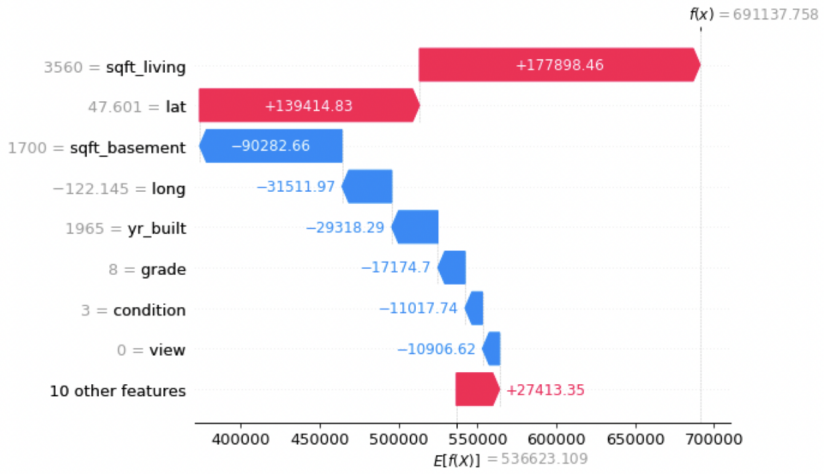


Figure 4.13: Shapley values for observation x_{10}

For the square feet below ground (sqft_living), having such high square footage, it would be thought that the contribution would be positive towards the model's prediction. However, computing a histogram of the feature values to analyze its value distribution, it was found

that there are not enough samples like the one in the demonstration where they have a large basement, therefore, possibly affecting the model's performance. More than half of the dataset's samples do not even have a basement. Furthermore, this sample is one of the top three highest in terms of basement square footage. The insight taken from this is that there is not enough data similar to this outlier feature value, so it is worth considering re-visiting the processing stage to clean or remove this outlier sample (see Figure 4.14). This is a perfect example of how Shapley values can give powerful insights that can help interpret the model and debug the model to improve its performance further.

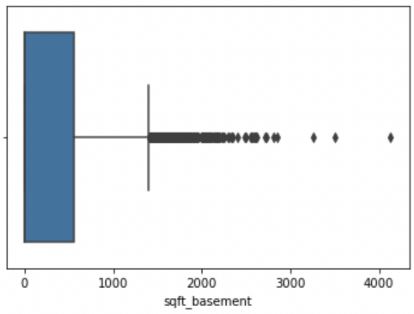


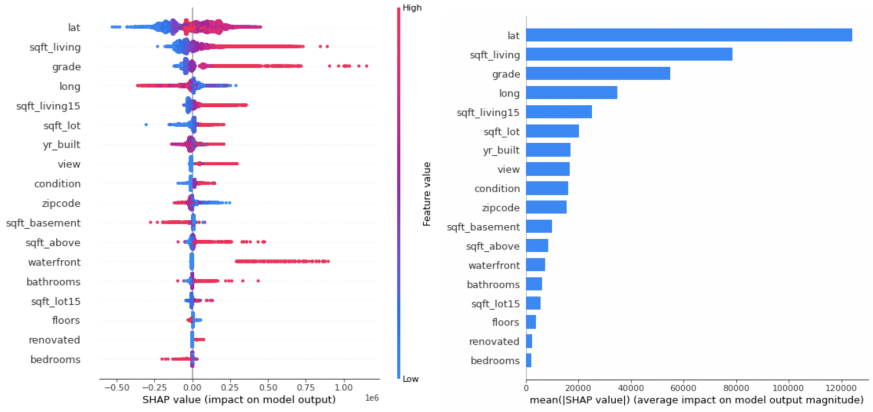
Figure 4.14: Boxplot for feature "sqft_living"

4.3.3 Global Scope

The overall mean Shapley values and its impact to the The overall mean Shapley values and their impact on the model output are shown below (Figure 4.15). We can see that features like latitude, size of living area, and construction quality ("grade" feature) are the most impactful features in the overall CatBoost model.

One observation looking at (Figure 4.15) can be by looking at the longitude and latitude features, as the house location tends to be in the north-west (higher latitude goes north from the globe, and lower longitude goes towards the left) of King County, there tend to be more houses with a high price, so more expensive areas. Note that this observation may be not entirely accurate, as there may be many factors for having that relation, one being that the King County area is surrounded by two or more lakes, being another factor in why specific coordinates have not enough houses.

Some features are more "obvious" to draw conclusions from, such as the "grade" feature (classification by construction quality, high grade = high quality). As the house has higher construction quality, its price will likely be higher than the average price.



(a) SHAP impact on model (b) Mean Absolute SHAP values

Figure 4.15: Global explanation of CatBoost model

Chapter 5

Results/Evaluation

The problem cases in the project were approached by using the Linear Explainer and the Tree Explainer from the SHAP python library. However, more Explainers in the library can combat expensive computation from some models, even though they are not all model-agnostic. One of them is the Deep Explainer, a model suited for Deep Learning models, which is highly convenient, with Deep Learning models being one of the least interpretable models in machine learning but have proven to be entirely accurate.

An alternative way of computing SHAP values in the problem cases would have been using some of the model-agnostic explainers in the SHAP library. Some of them are the Kernel-Explainer and the base Explainer. The Kernel Explainer has a similar algorithm to LIME (Ribeiro et al., 2016) but uses different weights for its calculation. One must keep in mind that this Kernel-based method is an approximation and does not compute the exact Shap Values. Furthermore, the Kernel Explainer algorithm is very slow and currently impractical to use in practice in comparison to other Explainers. The base Explainer can take any type of machine learning model and decide the best algorithm to compute the Shapley values, where the algorithm is not specified.

The project addressed real-world case tabular data; however, that does not mean SHAP values can only be used in tabular data. There are many examples online demonstrating how SHAP values can interpret models that are used in fields like Computer Vision or Natural Language Processing (NLP). The project could have further improved by adding or maybe replacing some of the tabular cases and adding some of the Computer Vision problems or NLPs to expand the scope of the project further and provide a more broad view of the impact these Shapley values can provide.

One needs a deep understanding of the data to conduct a meaningful analysis. Even though Shapley values are calculated correctly, there is a probability that the user may misinterpret these values and therefore create misleading explanations. Insights were drawn from the project, although there is a probability of being misinterpreted, so one has to understand the data in place as much as possible and understand the way to extract insights from these contribution values. On the other side, Shapley values may heavily rely on how the model acts on unrealistic input data. Having noisy data full of anomalies will heavily impact the machine learning model and the model impacting the calculated Shapley values.

Undergoing data extraction and feature exploration, there could have been more elegant methods to process the data to uncover its statistical power and further improve model efficiency. Feature engineering is an essential task in Data Science to arrive at satisfactory conclusions; the project's scope could have been extended to take more time into data cleaning to make more optimal predictions. However, it was also intended to show in some instances how SHAP values can help in uncovering anomalies within the feature data and the model globally.

Towards the end of these analysis cases, interpreting these machine learning models gave insights into the inside mechanics of how well they predicted, finding anomalies within the data, and feature importances that hold some common truth—for example, having the frequency of eating vegetables increasing the probability that one is doing a healthy activity, therefore having a likelihood of having average weight. It is common sense that eating vegetables is good for human health, leading to a balanced weight. On the other side, the "smoke" feature had the lowest Shapley values in predicting an individual's weight class. It is believed that smoking often leads to losing weight by increasing the metabolic rate and decreasing caloric consumption, so having an individual who regularly smokes should have a more substantial impact on the prediction than, say, the "Gender" feature.

For the Wine sample case, the model's performance was somewhat good enough. The model had an R^2 score of 87.5%, which could be further improved by model tuning, feature engineering, cleaning, and more processes. This case was a fine example of introducing the linearity of Shapley values and comparing it with other feature importance variables such as the coefficients used in the learning function of the linear model.

Thirdly, the house price scenario had a gradient boosting algorithm with some desirable inbuilt feature importance, such as the Gini importance. While these inbuilt feature importances are not as computationally expensive as the Shapley values, they provided quite different

results. Furthermore, more processing in this scenario happened than in the other cases due to its categorical features. However, these were encoded to fit the CatBoosting model better and provide better predictions.

Shapley values can be a potent tool; however, they must be used with caution. Misreading the interpretation results may lead to inconvenient misinterpretations. There also has to be a high knowledge of the area of what is the field of Data Science in general. One has to fully understand the data at hand in order to draw relevant conclusions. As Shapley values can also help spot anomalies within the model, it is also necessary to know how to take the necessary actions to fix them.

Chapter 6

Legal, Social, Ethical and Professional Issues

6.1 Legal & Professional Issues

Generally, in many countries, there has been an increase in regulatory compliance around the right to explanation in the field of Artificial Intelligence (AI). There is a strong need to take action and develop regulations around the accountability towards AI systems in many sectors such as healthcare and finance (Harper, 2019). These regulations enforce companies into the adoption of explainability into machine learning systems in order to become regulatory-compliant. As regulations expand across sectors, more and more organisations will be compelled to adapt Explainable AI models like SHAP, avoiding the risk of being liable for penalties concerning explainability.

6.2 British Computing Society Code of Conduct

The professional standards and protocols from the British Computer Society were followed throughout the entire length of this project. The third-party data extraction was conducted legitimately as these sources are publicly available for research purposes. The analysis and investigation of this project did in no way partake in unethical manners and in no way breach the code of conduct.

Chapter 7

Conclusion and Future Work

SHAP values are a powerful tool for model interpretability. The current downside is how computationally expensive it is to be calculated. However, as this is a recent novel idea, there is room for further development and new algorithms to calculate exact Shapley values more efficiently.

The projects' experiment cases were implemented using relatively fast Explainers (Linear SHAP and Tree SHAP); however, model-agnostic methods are prefered as; ideally, one would want to test the same data on different models to be able to interpret data across models. Shapley values have been widely used in model interpretability and continue to be improved upon with newer, faster algorithms to calculate these Shapley values. Particularly, model-agnostic explainers will unlock tremendous potential as they can work well with as many models as possible and, in that way, tune the best model for a given problem (Molnar, 2020). As automation is bound to grow continually, the training and interpretation of a machine learning model could be easily made by individuals who are not strictly Machine Learning experts. This technology will become widely accessible and continually improve prediction problems, not to mention its compliance with the regulation in specific business areas.

Since black-box models are still non-transparent, further research in model explainability will demystify these models' trust. As new research on Explainable methods surfaces in the near future, we will have more machine learning models that can explain themselves, clearing a path toward building more intelligent models with more predictive power.

Bibliography

Anselin, D. L., et al. (2003). Geoda. *geodacenter.github.io*.

Bach, S., Binder, A., Montavon, G., Klauschen, F., Müller, K.-R., & Samek, W. (2015). On pixel-wise explanations for non-linear classifier decisions by layer-wise relevance propagation. *PloS one*, *10*(7), e0130140.

Cortez, P., Cerdeira, A., Almeida, F., Matos, T., & Reis, J. (2009). Modeling wine preferences by data mining from physicochemical properties. *Decision support systems*, *47*(4), 547–553.

Harper, J. (2019). Proceed with caution: Explainability, ai, and the risks of regulatory compliance. *AI Business*.

Lundberg, S. (2018). Shap documentation. *https://shap.readthedocs.io/en/latest/index.html*.

Lundberg, S. M., & Lee, S.-I. (2017). A unified approach to interpreting model predictions. *Advances in neural information processing systems*, *30*.

Molnar, C. (2020). *Interpretable machine learning*.

Palechor, F. M., & de la Hoz Manotas, A. (2019). Dataset for estimation of obesity levels based on eating habits and physical condition in individuals from colombia, peru and mexico. *Data in brief*, *25*, 104344.

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*.

Ribeiro, M. T., Singh, S., & Guestrin, C. (2016). " why should i trust you?" explaining the predictions of any classifier. In *Proceedings of the 22nd acm sigkdd international conference on knowledge discovery and data mining* (pp. 1135–1144).

Shapley, L. S., Roth, A. E., et al. (1988). *The shapley value: essays in honor of lloyd s. shapley*. Cambridge University Press.

Appendix A

Source Code

A.1 Case 1 Code

Listing A.1: Wine fixed acidity source code

```
#!/usr/bin/env python
# coding: utf-8


# <div class="alert alert-block alert-info"><strong>Content</strong
    ></div>
# <div class="list-group">
#     <a class="list-group-item list-group-item-action" href="#
    imports">Imports</a>
#     <a class="list-group-item list-group-item-action" href="#
    collection">Data Collection</a>
#     <a class="list-group-item list-group-item-action" href="#
    processing">Data Processing</a>
#     <a class="list-group-item list-group-item-action" href="#
    modeling">Modeling</a>
#     <a class="list-group-item list-group-item-action" href="#
    evaluation">Model Evaluation</a>
#     <a class="list-group-item list-group-item-action" href="#shap
    ">SHAP</a>
#     <a class="list-group-item list-group-item-action" href="#
```

```
    shaplocal">SHAP Local</a>
#       <a class="list-group-item list-group-item-action" href="#
    shapglobal">SHAP Global</a>
#
# </div>

# <div class="alert alert-block alert-success" id='imports'><strong>
    Imports</strong></div>

# In[1]:

import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.linear_model import LinearRegression # model used in
    case 1
from sklearn.model_selection import train_test_split # splitting
    data for validation
from sklearn.metrics import mean_squared_error, mean_absolute_error
    # model evaluation

import shap # feature importance

get_ipython().run_line_magic('matplotlib', 'inline')

# <div class="alert alert-block alert-success" id='collection'><
    strong>Data collection</strong></div>

# Dataset citation
# P. Cortez, A. Cerdeira, F. Almeida, T. Matos and J. Reis. Modeling
     wine preferences by data mining from physicochemical properties.
```

In Decision Support Systems, Elsevier, 47(4):547-553, 2009.

```
# In[2]:

# importing wine quality dataset
data = pd.read_csv('src/winequality-red.csv')
data.info()

# In[3]:

## Features
# 0 fixed acidity
# 1 volatile acidity
# 2 citric acid
# 3 residual sugar
# 4 cholrides
# 5 free sulfur dioxide
# 6 total sulfur dioxide
# 7 density
# 8 ph
# 9 sulphates
# 10 alcohol
# 11 quality
data.head()

#<div class="alert alert-block alert-success" id='processing'><
    strong>Data Processing</strong></div>

# In[4]:

sns.pairplot(data)

# In[5]:
```

```
data.describe().iloc[1:,:].T

# In[6]:

# Choosing target
target = 'fixed acidity'
X = data.loc[:, data.columns != target]
y = data.loc[:,target]

# In[7]:

# Split data for testing
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                test_size=0.15, random_state=0)

print('Train instances:', np.shape(X_train), np.shape(y_train))
print('Test instances:', np.shape(X_test), np.shape(y_test))

#<div class="alert alert-block alert-success" id='modeling'><strong
    >Modeling</strong></div>

# In[8]:

# Modeling
model = LinearRegression()
model.fit(X_train, y_train)

#<div class="alert alert-block alert-success" id='evaluation'><
    strong>Model Evaluation</strong></div>

# In[9]:
```

```
# Model evaluation
y_pred = model.predict(X_test)

print('MAE: ',mean_absolute_error(y_test,y_pred)) # sum(abs(y_test -
    y_pred) / len(y_test))
print('MSE: ',mean_squared_error(y_test,y_pred)) # sum((y_test -
    y_pred)**2 / len(y_test))
print('RMSE: ',np.sqrt(mean_squared_error(y_test,y_pred)))
print('R^2 score', model.score(X_test,y_test)) # Calculates r^2


# In[10]:


# Model coefficients
print('Linear model coefficients:\n')
for i in range(X.shape[1]):
    print(X.columns[i], "=", model.coef_[i].round(4))


# <div class="alert alert-block alert-success" id='shap'><strong>
    SHAP</strong></div>

# In[11]:


# Compute Linear SHAP values on whole set
explainer = shap.LinearExplainer(model, X)
shap_values = explainer(X)


# In[12]:
```

```
# Compute base SHAP values on whole set (model-agnostic)
explainer2 = shap.Explainer(model.predict, X)
shap_values2 = explainer2(X)


# <div class="alert alert-block alert-success" id='shaplocal'><
    strong>SHAP Local</strong></div>

# In[13]:

sample = 10
shap_values[sample]


# In[14]:

print('Data sample', sample)
data.iloc[sample]


# In[15]:


sample_pred = model.predict(X.iloc[sample:sample+1,:])
print('Predicted sample', sample)
print(y.name, sample_pred)


# In[16]:


X100 = shap.sample(X, 100)
```

```
plot = shap.plots.partial_dependence(
    "density", model.predict, X100, ice=False,
    model_expected_value=True, feature_expected_value=True,
    shap_values=shap_values[sample:sample+1,:]
)

# In[17]:


shap.plots.scatter(shap_values[:,"density"])


# In[18]:


# the waterfall_plot
shap.plots.waterfall(shap_values[sample], max_display=len(X.columns)
    )


# In[19]:


# Generate force plot - Single
shap.initjs()
shap.force_plot(base_value=shap_values.base_values[0],
                shap_values=shap_values.values[sample,:], features=X
                    .loc[sample,:])


# <div class="alert alert-block alert-success" id='shapglobal'><
    strong>SHAP Global</strong></div>
```

```
# In[20]:


# Generate summary bar plot (Global)
shap.summary_plot(shap_values, X)


# In[21]:


shap.force_plot(shap_values.base_values[0], shap_values.values, X)


# In[22]:


shap.plots.bar(shap_values)#, max_display=11)
```

A.2 Case 2 Code

Listing A.2: Obesity classification source code

```
#!/usr/bin/env python
# coding: utf-8

#<div class="alert alert-block alert-info"><strong>Content</strong
    ></div>
#<div class="list-group">
#     <a class="list-group-item list-group-item-action" href="#
    imports">Imports</a>
#     <a class="list-group-item list-group-item-action" href="#
    collection">Data Collection</a>
#     <a class="list-group-item list-group-item-action" href="#
    processing">Data Processing</a>
```

```
#      <a class="list-group-item list-group-item-action" href="#
    modeling">Modeling</a>
#      <a class="list-group-item list-group-item-action" href="#
    evaluation">Model Evaluation</a>
#      <a class="list-group-item list-group-item-action" href="#shap
    ">SHAP</a>
#      <a class="list-group-item list-group-item-action" href="#
    shaplocal">SHAP Local</a>
#      <a class="list-group-item list-group-item-action" href="#
    shapglobal">SHAP Global</a>
#
# </div>

# <div class="alert alert-block alert-success" id='imports'><strong>
    Imports</strong></div>

# In[1]:


#imports
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.preprocessing import LabelEncoder # transform
    categorical data
from sklearn.tree import DecisionTreeClassifier # Model for case 2
from sklearn.metrics import classification_report, accuracy_score #
    metrics
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import train_test_split # model
    validation
```

```
from sklearn.tree import plot_tree

import shap

get_ipython().run_line_magic('matplotlib', 'inline')


# <div class="alert alert-block alert-success" id='collection'><
    strong>Data Collection</strong></div>

# In[2]:


data = pd.read_csv('src/ObesityDataSet_raw_and_data_sinthetic.csv')
data.info()


# <div class="alert alert-block alert-success" id='processing'><
    strong>Data Processing</strong></div>

# In[3]:


# height and weight highly correlated to BMI (target based)
data = data.drop(columns=['Height', 'Weight'])


# In[4]:


data.head()
```

```
# In[5]:


# group categorical data and create encoded data version
columns = ["Gender", "family_history_with_overweight", "FAVC",
           "CAEC", "SMOKE", "SCC", "CALC", "MTRANS", "NObeyesdad"]
for col in columns:
    data[col] = data[col].astype('category')

data_enc = data.copy()


# In[6]:


# transform categorical data to numberical
category = ['Gender','family_history_with_overweight', 'FAVC', 'CAEC
    ',
           'SMOKE', 'SCC', 'CALC', 'MTRANS', 'NObeyesdad']
encoder_list = []
for c in category:
    encoder = LabelEncoder()
    data_enc[c] = encoder.fit_transform(data_enc[c])
    encoder_list.append(encoder)

data_enc.head()


# In[7]:


# split data
X, y = data_enc.iloc[:,:-1], data_enc.iloc[:,-1]
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size
    =0.2,
                                                    random_state=1)


# <div class="alert alert-block alert-success" id='modeling'><strong
    >Modeling</strong></div>

# In[8]:


model = DecisionTreeClassifier()
model.fit(X_train, y_train)


# <div class="alert alert-block alert-success" id='evaluation'><
    strong>Model Evaluation</strong></div>

# In[9]:


y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print('Accuracy_score', accuracy)


# In[10]:


y_test_inv = encoder_list[-1].inverse_transform(y_test)
y_pred_inv = encoder_list[-1].inverse_transform(y_pred)
print(classification_report(y_test_inv,y_pred_inv))
```

```
# In[11]:


# print 5 samples to compare prediction and true value
for i in range(5):
    result = False
    if y_test.iloc[i] ==  y_pred[i]:
        result = True
    print(i, result, y_test.iloc[i], y_pred[i], y_pred_inv[i])


# In[12]:


# Gets the gini importance
feature_imp = pd.Series(model.feature_importances_,
                        index=X.columns).sort_values(ascending=False
                            )
plt.bar(feature_imp.index, feature_imp)
plt.xticks(rotation=45, horizontalalignment='right')
plt.show()


# In[13]:


print('Gini importance:\n')
feature_imp


## Use confusion matrix
# Used to analyze target predictions
```

```
# In[14]:


class_names = ['Insufficient_Weight', 'Normal_Weight', '
    Obesity_Type_I', 'Obesity_Type_II',
                'Obesity_Type_III', 'Overweight_Level_I', '
                   Overweight_Level_II']
disp = plot_confusion_matrix(model, X_test, y_test, display_labels=
    class_names,
                                 xticks_rotation='vertical')


# In[15]:


# visualize top of the decision tree and its splits
class_names = data.iloc[:,-1].dtypes.categories.to_list()
plt.figure(figsize=(16,12))
plot_tree(model, feature_names=X.columns, max_depth=2, class_names=
    class_names, fontsize=10, filled=True)
plt.show()


#<div class="alert alert-block alert-success" id='shap'><strong>
    SHAP</strong></div>

# In[16]:


explainer = shap.TreeExplainer(model, X)
shap_values = explainer(X)
```

```
# In[17]:


print('shap shape', np.shape(shap_values))


#<div class="alert alert-block alert-success" id='shaplocal'><
    strong>SHAP Local</strong></div>

# In[18]:


# print data sample to be explained
sample = 5
print('data sample', sample)
data.iloc[sample,:]


# In[19]:


# show shap values for the predicted target (1)
shap_values[sample,:,1]


# In[20]:


print('Actual value = ', y[sample], class_names[y[sample]])
sample_pred = model.predict(X.iloc[sample:sample+1,:])[0]
print('Predicted value = ', sample_pred, class_names[sample_pred])
```

```
# In[21]:


# note that the third index on the shap values paramter
# indicates the index of the class predicted in the explainer
shap.initjs()
shap.plots.waterfall(shap_values[sample,:,1], max_display=9)


# In[22]:


# this is an explanation for a class that was not predicted
shap.plots.waterfall(shap_values[sample,:,0], max_display=9)


#<div class="alert alert-block alert-success" id='shapglobal'><
    strong>SHAP Global</strong></div>

# In[23]:


np.shape(shap_values)


# In[24]:


# bar plot to explain global scope
target_class = ['Insufficient_Weight',
                'Normal_Weight',
                'Obesity_Type_I',
```

```
                    'Obesity_Type_II',
                    'Obesity_Type_III',
                    'Overweight_Level_I',
                    'Overweight_Level_II']
shap.summary_plot(shap_values.values[:,:,1], X.values, plot_type="
    bar", class_names=target_class,
                    feature_names=X.columns)


# In[25]:


# beesawarm plot
shap.summary_plot(shap_values[:,:,1], X.values, class_names=
    target_class,
                    feature_names=X.columns)


# In[26]:


#global force plot
shap.initjs()
shap.force_plot(base_value=explainer.expected_value[6],
                    shap_values=np.array(shap_values).reshape(2111,14,7)
                        [:,:,6],
                    feature_names=X.columns)
```

A.3 Case 3 Code

Listing A.3: House price source code

```
#!/usr/bin/env python
# coding: utf-8
```

```
# <div class="alert alert-block alert-info"><strong>Content</strong
    ></div>
# <div class="list-group">
#     <a class="list-group-item list-group-item-action" href="#
    imports">Imports</a>
#     <a class="list-group-item list-group-item-action" href="#
    collection">Data Collection</a>
#     <a class="list-group-item list-group-item-action" href="#
    processing">Data Processing</a>
#     <a class="list-group-item list-group-item-action" href="#
    modeling">Modeling</a>
#     <a class="list-group-item list-group-item-action" href="#
    evaluation">Model Evaluation</a>
#     <a class="list-group-item list-group-item-action" href="#shap
    ">SHAP</a>
#     <a class="list-group-item list-group-item-action" href="#
    shaplocal">SHAP Local</a>
#     <a class="list-group-item list-group-item-action" href="#
    shapglobal">SHAP Global</a>
#
# </div>

# <div class="alert alert-block alert-success" id='imports'><strong>
    Imports</strong></div>

# In[1]:


#imports
import numpy as np
import pandas as pd
import seaborn as sns
```

```
import matplotlib.pyplot as plt

from catboost import CatBoostRegressor
from sklearn.model_selection import train_test_split

from sklearn.metrics import mean_squared_error,mean_absolute_error
from sklearn.metrics import classification_report,confusion_matrix,
    accuracy_score

import shap

get_ipython().run_line_magic('matplotlib', 'inline')


#<div class="alert alert-block alert-success" id='collection'><
    strong>Data Collection</strong></div>

# In[2]:


data = pd.read_csv('src/kc_house_data.csv')


# In[3]:


data.head()
data.info()


#<div class="alert alert-block alert-success" id='processing'><
    strong>Data Processing</strong></div>
```

```
# In[4]:


# drop date because it is irrelevant for our findings
data = data.drop(['date', 'id'],axis=1)


# In[5]:


# Getting rid of price outliers
data = data.drop(data[data.price > 3887500.0].index) # rid of 12
    incredibly high outliers


# In[6]:


# Getting rid of bathroom outliers
data = data.drop(data[data.bedrooms > 15].index)


# In[7]:


# Since most of the yr_renovated is has 0 values,
# the data will be transformed to boolean (yr_renovated to renovated
    )
data['yr_renovated'] = pd.Series(np.where(data.yr_renovated.values >
     0, 1, 0),
            data.index)
data.rename(columns = {'yr_renovated':'renovated'}, inplace = True)
```

```
# In[8]:


# Check out for outliers by printing histograms of all features
for feature in data.columns:
    print(feature)
    sns.histplot(data[feature])
    plt.show()
    print(data[feature].value_counts(bins=10, sort=False))
    print('Min', min(data[feature]))
    print('Max', max(data[feature]))
    print('\n\n')


# In[9]:


data.info()


# Pearson correlation matrix
#
# We use the Pearson correlation coefficient to examine the strength
    and direction of the linear relationship between two continuous
    variables.
#
# The correlation coefficient can range in value from  1  to +1.
    The larger the absolute value of the coefficient, the stronger
    the relationship between the variables. For the Pearson
    correlation, an absolute value of 1 indicates a perfect linear
    relationship. A correlation close to 0 indicates no linear
    relationship between the variables.
```

```
#
# The sign of the coefficient indicates the direction of the
    relationship. If both variables tend to increase or decrease
    together, the coefficient is positive, and the line that
    represents the correlation slopes upward. If one variable tends
    to increase as the other decreases, the coefficient is negative,
    and the line that represents the correlation slopes downward.

# In[10]:


sns.set(style="whitegrid", font_scale=1)
plt.figure(figsize=(13,13))
plt.title('Pearson Correlation Matrix',fontsize=25)
sns.heatmap(data.corr(),linewidths=0.25,vmax=0.7,square=True,cmap="
    GnBu",
                linecolor='w',annot=True, annot_kws={"size":7},
                cbar_kws={"shrink": .7})


# Which features are more correlated to the price?
#

# In[11]:


price_corr = data.corr()['price'].sort_values(ascending=False)
print(price_corr)


# Discovered various outlier samples that have incredibly large
    square foot basements, while more than half the dataset tends to
    not even have a basement. Feature should be cleaned
```

```
# In[12]:


plot = sns.boxplot(data['sqft_basement'])


# In[13]:


# check distribution grouped by bins
data['sqft_basement'].value_counts(bins=20, sort=False)


# In[14]:


sns.histplot(data['sqft_basement'])


# In[15]:


# print an overview of the data relation with every pair of features
sns.pairplot(data)


# as longitud tends to decrease, that is go west, in King county,
    there appears to be more houses, and particularly houses with
    higher price. Similarly with the latitud as it increases, that
    means, as the house is more to the north, there are more houses
    with high price in the King County area in washington state.
```

```
# In[16]:


sns.scatterplot(data['long'], data['price'])


# In[17]:


sns.scatterplot(data['lat'], data['price'])


#<div class="alert alert-block alert-success" id='modeling'><strong
    >Modeling</strong></div>

## Splitting the data

# In[18]:


X = data.drop('price', axis=1)
y = data['price']


# In[19]:


X_train, X_test, y_train, y_test = train_test_split(X,y,test_size
    =0.3,
                                                    random_state=0)


# In[20]:
```

```
print(X_train.shape)
print(X_test.shape)
print(y_train.shape)
print(y_test.shape)


## Creating model

# In[21]:


params = {'iterations':10000,
          'learning_rate':0.01,
          'depth':3,
          'loss_function':'RMSE',
          'eval_metric':'RMSE',
          'random_seed':55,
          #'cat_features':boston_categories,
          'metric_period':200,
          'od_type':"Iter",
          'od_wait':20,
          'verbose':True,
          'use_best_model':True}
model = CatBoostRegressor(**params)


## Training model

# In[22]:
```

```
model.fit(X_train, y_train,
          eval_set=(X_test, y_test),
          use_best_model=True,
          plot=True,
          verbose=True)


# <div class="alert alert-block alert-success" id='evaluation'><
    strong>Model Evaluation</strong></div>

## Metrics

# R^2: compares models prediction to the mean of the targets. that is
     the expected value

# In[23]:


# predictions on the test set
y_pred = model.predict(X_test)

print('MAE: ',mean_absolute_error(y_test,y_pred)) # sum(abs(y_test -
     y_pred) / len(y_test))
print('MSE: ',mean_squared_error(y_test,y_pred)) # sum((y_test -
    y_pred)**2 / len(y_test))
print('RMSE: ',np.sqrt(mean_squared_error(y_test,y_pred)))
print('R^2 score', model.score(X_test,y_test)) # Calculates r^2


# <div class="alert alert-block alert-success" id='shap'><strong>
    SHAP</strong></div>

# In[24]:
```

```
explainer = shap.TreeExplainer(model)
shap_values = explainer.shap_values(X) # returns an array of shapley
    values
shap_values2 = explainer(X) # this returns a dictionary containing
# shapley values, data values, base values. Convenient for plotting


# In[25]:


print('shap shape', np.shape(shap_values))


# <div class="alert alert-block alert-success" id='shaplocal'><
    strong>SHAP Local</strong></div>

# In[26]:


# print data sample to be explained
sample = 10
print('data sample', sample)
data.iloc[sample,:]


# In[27]:


# show shap values for the predicted target (1)
shap_values2[sample]
```

```
# In[28]:


print('Actual value = ', y[sample])
sample_pred = model.predict(X.iloc[sample:sample+1,:])
print('Predicted value = ', sample_pred)


# In[29]:


shap.initjs()
shap.force_plot(explainer.expected_value, shap_values[sample,:],
                X_test.iloc[sample,:])


# In[30]:


shap.plots.waterfall(shap_values2[sample], max_display=9)


#<div class="alert alert-block alert-success" id='shapglobal'><
    strong>SHAP Global</strong></div>

# In[31]:


shap.summary_plot(shap_values, X, max_display=22)


# In[32]:
```

```
shap.summary_plot(shap_values2, X, plot_type="bar", max_display=22)
```