Tunisian Republic
Ministry of Higher Education
and Scientific Research
University of Tunis El Manar
Higher Institute of Computer Science

Graduation Project Report

Presented in partial fulfillment of the requirements for the obtention of

National Bachelor in Computer Engineering

Speciality : Software Engineering and Information Systems

By:

Abdelghani Mohamed

InsurePrice: AI-Powered Car Market Valuation System

Company supervisor: **Mr.Nasri Med Mehdi**

Academic supervisor: **Mr.Marzouki Kirmene**

Realized within Star Assurances

Academic year : 2023-2024

Dedication

All the progress i was able to accomplish throughout my life has to be mainly due to the presence of certain individuals . This project is no different .

I dedicate this work to:
My dear parents, Sihem and Abdennacer for their unconditional love and unwavering support throughout my studies. No matter the circumstances , you were always there, showing me the way and shedding a light on the darkness.

All my friends who always helped me navigate through stressful moments and surpass many challenges with their encouragement and camaraderie.

And finally, to my biggest source of inspiration , my sister Salma, whose relentless aid motivated me to always push forward.

Acknowledgments

I first and foremost would like to express my deep appreciation to the Star team and in particular to Mr. Mehdi Nasri along with the IT department for their warm welcome and their constant support along the way. The trust and the numerous opportunities he has given me have been vital to my personal and professional development in the field of AI and machine learning.

I also owe many thanks to my academic advisor, Mr. Kirmene Marzouki for his dedication, availability, and his continuous help. He has been a great source of motivation throughout the journey of my end-of-year project.

I would also like to thank all of my professors at the Higher Institute of Computer Science. Their expertise in the field has been a great asset for my education and my future career. Lastly, I would like to thank the members of the jury for having accepted to evaluate my work. Both their feedback and their ideas are highly appreciated.

Contents

List of Figures

List of Tables

List of Acronyms

- AI: Artificial Intelligence
- API: Application Programming Interface
- CRISP-DM: Cross-Industry Standard Process for Data Mining
- CSV: Comma-Separated Values
- DL: Deep Learning
- EDA: Exploratory Data Analysis
- HTTP: HyperText Transfer Protocol
- HTML: HyperText Markup Language
- ML: Machine Learning
- MSE: Mean Squared Error
- NN: Neural Network
- PCA: Principal Component Analysis
- R2: Coefficient of Determination
- RMSE: Root Mean Squared Error
- SEMMA: Sample, Explore, Modify, Model, Assess
- SQL: Structured Query Language
- URL: Uniform Resource Locator
- XGB: XGBoost

General Introduction

In the realm of insurance, accurately assessing the value of used cars is a critical task and a fundamental component of the car insurance sector. As a matter of fact , in cases of car accidents , the insured drivers rely on "collision coverage" , where insurers assess vehicle damages . This means that after filing a claim detailing the accident, the insurance company is responsible for assessing the damages done to the vehicle and determining the amount they will cover after the deductible. If the cost of repairs exceeds a well-determining limit, of course , the insured must cover all expenses accordingly without supplementary help . Therefore, the determination of a car's "market value" , is an invaluable process for both insurer and insured,influencing the overall economic expenses in the car insurance sector.

The risks associated with failing this process are substantial. Outdated methods and potential human error can result in incorrect valuations, leading to financial discrepancies and fraud. There have been numerous cases of fraudulent attempts aiming to scam the insurers and greedly increase the covered fees.
As good as things stand in the present moment, the latter indicates that the insurance sector in Tunisia could greatly benefit from implementing a modern approach to automate these processes.
In this context , this project, conducted at Star Assurances, aims to address this challenge by developing a machine learning system to predict car market values based on various features. Our solution encompasses data collection and machine learning model development.
To create a comprehensive dataset, we will amass valuable information about used cars along with their second-hand prices from the real-time market, forming the basis for training our predictive models.

By evaluating different models and fine-tuning them, we aim to achieve the highest possible accuracy in price prediction. This not only streamlines the insurance reporting process but also provides a reliable tool for assessing car values.

The report is structured as follows: We'll start by introducing the problem and motivation behind this project in the first chapter, where we outline the challenges faced in traditional car value assessments and the potential benefits of a data science-driven approach. In the second chapter, we'll delve into a review of existing methods and technologies used for car price prediction and related fields, providing a comprehensive background for our work.

Next, in the third chapter, we'll move on to the data collection and preparation process, detailing the techniques used for web scraping and the steps taken to clean and organize the data into a usable format. Following this, the fourth chapter will focus on the development and evaluation of various machine learning models, discussing the algorithms tested, the training process, and the criteria used for model evaluation.

In the fifth chapter, we'll present the implementation of our chosen model, showcasing its results and demonstrating its accuracy in predicting car prices. Finally, we'll conclude in the sixth chapter by summarizing our findings, discussing the implications of our results, and suggesting potential improvements and future directions for this project. Finally, we will propose further extensions to the approaches adopted in this project.

By developing this machine learning system, we aim to enhance the accuracy and efficiency of car value assessment for insurance purposes, ultimately benefiting both the insurance company and its clients.

1

General Context

Introduction

In this chapter, we will outline the problem of vehicle value assessment in the automotive insurance industry and the motivation for automating this process using machine learning. We will describe the project's objectives and the expected outcomes. Additionally, we will provide an overview of the methodology, including data collection, preprocessing, and model development. Finally, we will summarize the structure of the report to set the stage for the subsequent chapters.

1.1 Host Organisation

1.1.1 Presentation

The Tunisian Insurance and Reinsurance Company STAR was established on December 6, 1958. It is a semi-public company that is composed of a head office, more than 200 agencies and 700 employees, which allows it to be closer to its customers.

Figure 1.1.1: STAR Logo

1.1.2 Company Objectives

The purpose of the company is to insure and re-insure against all risks that may result in material or immaterial damage, as well as against all civil liability risks,professional or otherwise. Indeed, the STAR distinguishes between different types of insurance:

- Property and casualty insurance, which manages "non-life" insurance and P&C insurance (Fire,miscellaneous, special, transport). They include property insurance, insurance insurance, insurance responsibility, etc.
- Personal insurance, which includes health insurance and insurance related to life, death, savings, retirement, ... For each of these insurances, a set of guarantees and services are offered.

1.2 Problem Statement :

In this subsection, we will discuss the current challenges and limitations in assessing the venal values of cars within the insurance industry. We will also outline the traditional methods used by insurance companies and identify the deficiencies that our proposed solution aims to address.

1.2.1 The world of insurance :

The insurance industry serves as a cornerstone of contemporary society, playing a pivotal role in managing risk and safeguarding financial stability for all kinds of clients.
At its essence, insurance functions as a mechanism for individuals and businesses to mitigate potential losses by transferring them to an insurance company in exchange for the payment of premiums .

This arrangement, encapsulated within the framework of insurance contracts, establishes a symbiotic relationship between insurer and insured.

As we delve into the intricacies of the insurance sector, we bear witness to the transformative influence of technological advancements and innovative practices. These developments are reshaping traditional paradigms, fostering greater efficiency, and enhancing the overall efficacy of insurance operations. Consequently, the insurance industry stands poised at the nexus of tradition and innovation, continually evolving to meet the dynamic demands of an ever-changing world.

1.2.2 Assessing cars market values :

Car insurance is a critical component of the insurance domain, providing financial protection against physical damage or bodily injury resulting from traffic collisions and against liability that could also arise from incidents in a vehicle. A key aspect of car insurance is accurately assessing the venal value of cars, which is essential for determining insurance premiums, payouts, and risk assessments. This value differs greatly from the car purchase price , despite the confusions made by most people.

Traditionally, insurance companies assess a car's venal value through a combination of manual inspections and historical data analysis. This process often involves evaluating the car's technical characteristics, maintenance history, accident data, and market trends. However, this method is time-consuming, subject to human error, and may not always reflect the current market conditions accurately.

As a matter of fact, this means that inaccurate assessments are a possible outcome in the scope of this traditional method, which poses a real issue for the auto insurance sector in Tunisia.

Insurers are trapped in outdated valuation methods based on thin, often unrevised, and imprecise data that undermines pricing insurance premiums correctly, claims settlement, and managing their exposure to risk. The insured suffers from a similar issue of unfair assessment, resulting in either undervaluation or overvaluation of their vehicle. This can translate into paying more than the fair price when the premium is concerned or being compensated less than the real value of the vehicle when a claim is involved. This in turn paves the way for financial via mediation and inefficiencies for both the insurers and the

insured, which showcases the importance of a fair and objective way of valuing cars.

1.2.3 The limitations of current methods :

The traditional method of assessing cars' venal values faces several challenges :

- **Time-Consuming:** Manual inspections and data analysis require significant time, delaying the insurance workflow.
- **Inaccuracy:** Human error and outdated data can lead to inaccurate valuations, affecting premiums and payouts.
- **Lack of Scalability:** The manual approach cannot efficiently handle a large volume of assessments, limiting the scalability of the insurance operations.
- **Inconsistent Results:** Different assessors may produce varying valuations for the same car, leading to inconsistencies and disputes.

These deficiencies highlight the need for an improved, automated solution to enhance the accuracy, efficiency, and consistency of car venal value assessments.

1.3 Proposed Solution :

The lately mentioned analysis of the existing system leads us to the realize the lack of an authenticated , agreed-upon method for the car valuation process .

To address these issues , our project proposes a data-driven solution leveraging advanced technologies such as web scraping and machine learning. By systematically collecting and analyzing data from online car listings, we can build predictive models that accurately estimate the market values of used cars. This approach offers several advantages, including real-time data updates, objective assessments, and improved accuracy.

The project is set within the context of improving the operational efficiency and accuracy of car insurance assessments. By employing a predictive approach, we aim to streamline the valuation process, reduce human error, and provide timely and reliable estimates of car values.

However, it is important to note that this approach is not enough to resolve the entirety of the problem, as the market value assessment process needs deep insight into the specified details of each accident and each insured car, which we won't have access to when trying to amass data online , including technical characteristics, maintenance history, accident history, etc..

Most importantly, this project will be a first-hand effort to start the automation process by determining the current second-hand value of said car in the Tunisian car market.

1.4 Project requirements :

Functional requirements

- **Data collection:** The system must be able to collect relevant vehicle data.
- **Data Preprocessing:** The system must be able to clean, standardize, and preprocess the collected data so that it can be used for analysis and modeling.
- **Data analysis:** The system must be able to conduct exploratory data analysis to identify trends, correlations, and important variables that influence the market value of vehicles.
- **Predictive modeling:** The system must be able to build predictive models based on the data collected to estimate the market value of vehicles.
- **Model Assessment:** The system must be able to assess the performance of predictive models using appropriate measurements.

Non-Functional requirements

- **Performance:** The system must be able to quickly process user requests in order to provide timely estimates of the vehicle's market value.
- **Accuracy:** The system must be accurate in its predictions, minimizing prediction errors, in order to provide reliable estimates of the market value of vehicles.

- **Scalability:** The system must be scalable to handle large volumes of data and support a growing number of users and queries.

1.5 Work Methodology

Every group of software developers is trying to meet the deadline. We all know that routines or document reformulations are two of the most frequent problems during a project. Therefore, in order to finish the project on time and deliver it successfully, we need to follow some kind of reasonable plan and a project management approach. We will therefore consider two of the most renowned methodologies in the data science world. Let's explore both of them in this section and find out what works best.

1.5.1 CRISP-DM :

CRISP-DM stands for cross-industry process for data mining. It is a methodology created by IBM in 1996 to meet the needs of data mining projects. It has become the most widely used process for developing data science projects and has successfully found its application in various fields: engineering, medicine, and marketing. This method is described as a hierarchical process and consists of six phases: understanding the business, understanding the data, data preparation, modeling, evaluation and deployment.

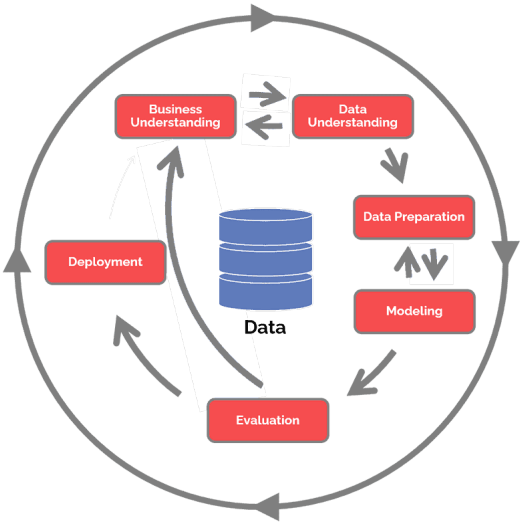


Figure 1.5.1: CRISP-DM Process

- **I. Business understanding:** This step is to give context to the data and objectives in order to obtain a notion of data relevance in a specific business model.
- **I. Data Understanding:** This stage is usually the longest. It consists of studying and exploring the data collected in order to verify its quality and find hidden information. It aims to build a representative sample that allows us to avoid unexpected problems in the following steps
- **III. Data Preparation :** This is the step that contains all the tasks needed to build the final data set. It consists of cleaning, integrating and transforming the data to make it more compatible with the algorithms to be used.
- **IV. Modeling:** This is the central stage of the methodology because it is responsible for the results that must meet the objectives of the project. In general, it is necessary to return to the previous step in order to improve the performance of the selected algorithms.
- **V. Evaluation:** At this level, the above-mentioned patterns and steps are evaluated to ensure that the project meets the enterprise objectives.
- **VI. Deployment:** This step is to organize and present the results to the end customer. It also involves deployment on production servers, if necessary.

1.5.2 SEMMA :

SEMMA was developed by the Statistical Analysis Institute (SAS) as a process for managing data exploration projects. SEMMA stands for Sample, Explore, Modify, Model, and Assess, which are the main stages of this process.
It facilitates the application of data exploration and visualization techniques, the selection and transformation of predictive variables, and variable modelling to ensure good model accuracy.

- **I.Sample :** In this step, a sample of data collection that is both large enough to include important information and small enough to be handled rapidly is extracted. It is noted that this step is optional.

- **II.Explore :** In order to obtain insight and ideas, this stage involves examining the data and looking for unexpected trends and abnormalities.
- **III. Modify:** In order to concentrate the model selection process, this step involves modifying the data by generating, choosing, and altering the variables.
- **IV. Model:** In this stage, the data is modeled by enabling the software to look for data combinations that consistently forecast the intended result.
- **V. Assess** This stage consists on assessing the data by evaluating the usefulness and reliability of the findings from the data mining process and estimate how well it performs.

1.5.2.1 CHOSEN METHODOLOGY:

To begin with, We can observe many similarities between SEMMA and CRISP-DM approaches: on one hand, they follow the same series of steps and require significant pretreatment activities. We also note that both techniques share the iterative property and require loops between specific phases.

SEMMA	CRISP-DM
(not supported)	Business Comprehension
Sample	Data Comprehension
Explore	
Modify	Data preparation
Model	Modeling
Evaluate	Evaluation
(not supported)	Deployment

Table 1.5.1: SEMMA vs CRISP-DM

We chose the CRISP-DM method after comparing each approach's sequential steps. This choice is reinforced by the weight this method places on the comprehension phase of the transaction as well as the ease and adaptability that set this procedure apart from others.

Conclusion

In this first chapter , We described the background and objectives of our study , with an emphasis on the current methods of car valuation within the insurance sector. We also outlined the shortcomings and irregularities of the existing manual assessment procedures and suggested a data-driven solution for automating it. The objective of this novel approach is to augment the precision, efficacy, and scalability of automobile appraisal procedures. Moving forward, we will delve into the specifics of our methodology, beginning with a thorough analysis of the current state of the art in relevant technologies.

2

Preliminary Study

Introduction

throughout this chapter , we'll delve into an exploration of the intersection between the world of insurance and the realm of Information Technology (I.T) , specifically the Data Science domain . We'll investigate an analysis of the fundamental principles underlying our project and the most noticeable fields aiding its conduct.

2.1 Insurtech :

Insurtech[2] refers to the use of technology innovations designed to find cost savings and efficiency from the current insurance industry model. Insurtech is a combination of the words "insurance" and "technology," inspired by the term fintech.

Insurtech is disrupting the insurance landscape by infusing I.T. into all insurer functions and operations to promote innovation, efficiency, and customer-centricity. The array of technological tools adopted under Insurtech includes Artificial Intelligence (AI), Big Data Analytics, the Internet of Things (IoT), and blockchain. These are all giving applicants a superior experience while revolutionizing risk analysis and fraud detection.

Machine Learning, in particular,emerges as a game changer in insurance. Machine learning

has the unique power to predict consumer behavior that can provide crucial insights to insurers.

By harnessing vast amounts of data and leveraging sophisticated algorithms, machine learning algorithms enable insurers to optimize underwriting processes, detect fraud, tailor pricing strategies, and automate claims processing. In a business that deals with trillions of dollars worth of premiums and claims, and with millions of variables and intricacies involved, an integral role of machine learning is central in optimizing the underwriting process, detecting frauds, pricing the insured at an optimal rate, and automating claims processing. In the context of our project, machine learning serves as a transformative tool for predicting used car prices accurately, thereby enabling insurance companies to make informed decisions and enhance their competitive edge in the market.Through this combination of insurtech and data science, we aim to further contribute to the ongoing digital transformation of the insurance landscape, in addition to, creating significant value and innovation for both the insured and the insurer.

2.2 Machine Learning :

2.2.1 introduction :

Machine learning is a groundbreaking field within artificial intelligence that has led to advancements in various industries. The field encompasses a wide range of algorithms, from traditional statistical methods to cutting-edge deep learning approaches. Machine learning has revolutionized decision-making procedures by allowing systems to predict outcomes, identify patterns, and adapt to changing contexts. Its practical applications are vast, including natural language comprehension, computer vision, self-driving cars, and medical diagnostics. As a constantly evolving discipline, machine learning offers ample opportunities for research, innovation, and the development of intelligent systems that can enhance our daily lives. and drive progress in the realm of artificial intelligence.

2.2.2 Types of Machine Learning :

There are four major categories: supervised learning, unsupervised learning, semisupervised learning, and Reinforcement Learning.

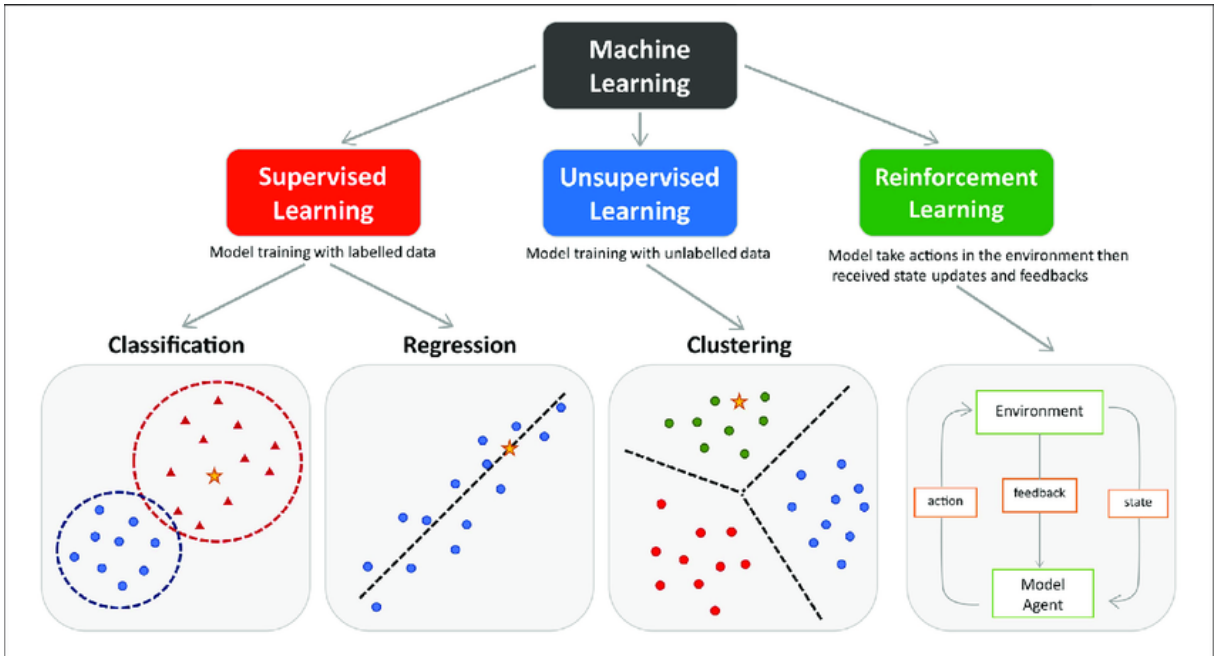


Figure 2.2.1: sections of Machine learning

Unsupervised learning :

In unsupervised learning, as you might guess, the training data is unlabeled. The system tries to learn without a teacher. It involves tasks like clustering, which groups similar data points for purposes such as customer segmentation and recommendation systems. Moreover, techniques for reducing dimensionality are utilized to simplify intricate data, representing it in a lower-dimensional space while preserving essential information. Unsupervised learning plays a vital role in exploratory data analysis and unearthing concealed insights within datasets.

Reinforcement learning :

Reinforcement Learning is a very different beast. The learning system, called an agent in this context, can observe the environment, select and perform actions, and get rewards in return (or penalties in the form of negative rewards, as in the next figure).

It must then learn by itself what is the best strategy, called a policy, to get the most reward over time. A policy defines what action the agent should choose when it is in a given situation.

Supervised learning : In supervised learning, the algorithm is trained on a labeled dataset, where each data point is paired with the correct output. The objective is for the algorithm to learn how to link inputs to outputs, making it capable of making accurate predictions on unseen data. Common applications include classification, where the algorithm categorizes data into predefined classes, and regression, where it predicts numerical values. Supervised learning is widely used in tasks like image recognition, spam email detection, and medical diagnosis

Here are some of the most important supervised learning algorithms:

- k-Nearest Neighbors
- Linear Regression
- Logistic Regression
- Support Vector Machines (SVMs)
- Ensemble learning and Random Forests
- Neural networks

2.2.3 Ensemble Learning

Suppose we ask a complex question to thousands of random people, then aggregate their answers. In many cases we will find that this aggregated answer is better than an expert's answer. This is called the wisdom of the crowd.

Similarly, if we aggregate the predictions of a group of predictors (such as classifiers or regressors), we will often get better predictions than with the best individual predictor. A group of predictors is called an ensemble; thus, this technique is called Ensemble Learning, and an Ensemble Learning algorithm is called an Ensemble method. For example, we can train a group of Decision Tree classifiers, each on a different random subset of the training set. To make predictions, we just obtain the predictions of all individual trees, then predict

the class that gets the most votes.
In fact, the winning solutions in Machine Learning competitions often involve several Ensemble methods. In this chapter, we will discuss the most popular Ensemble methods, including bagging, boosting and stacking.

2.2.3.1 Boosting :

boosting is a very practical supervised learning algorithm which refers to any Ensemble method that can combine several weak learners into a strong learner. The general idea of most boosting methods is to train predictors sequentially, each trying to correct its predecessor. There are many boosting methods available, but by far the most popular are Adaboost and Gradient boosting , next we'll talk extensively about gradient boosting as it is chosen as a key model to use and implement in this project (we'll dive more into that in the next chapter)

Gradient Boosting :

Gradient Boosting is a very popular Boosting algorithm.
It works by sequentially adding predictors to an ensemble, each one correcting its predecessor. However, instead of tweaking the instance weights at every iteration , this method tries to fit the new predictor to the residual errors made by the previous predictor.
Let's go through a simple regression example using Decision Trees as the base predictors (of course, Gradient Boosting also works great with regression tasks). This is called Gradient Tree Boosting, or Gradient Boosted Regression Trees (GBRT). First, let's fit a DecisionTreeRegressor to the training set (for example, a noisy quadratic training set):

The next figure represents the predictions of our three trained trees in the left column, and the ensemble's predictions in the right column. In the first row, the ensemble has just one tree, so its predictions are exactly the same as the first tree's predictions. In the second row, a new tree is trained on the residual errors of the first tree. On the right you can see that the ensemble's predictions are equal to the sum of the predictions of the first two trees.

Similarly, in the third row another tree is trained on the residual errors of the second tree. You can see that the ensemble's predictions gradually get better as trees are added to the ensemble.

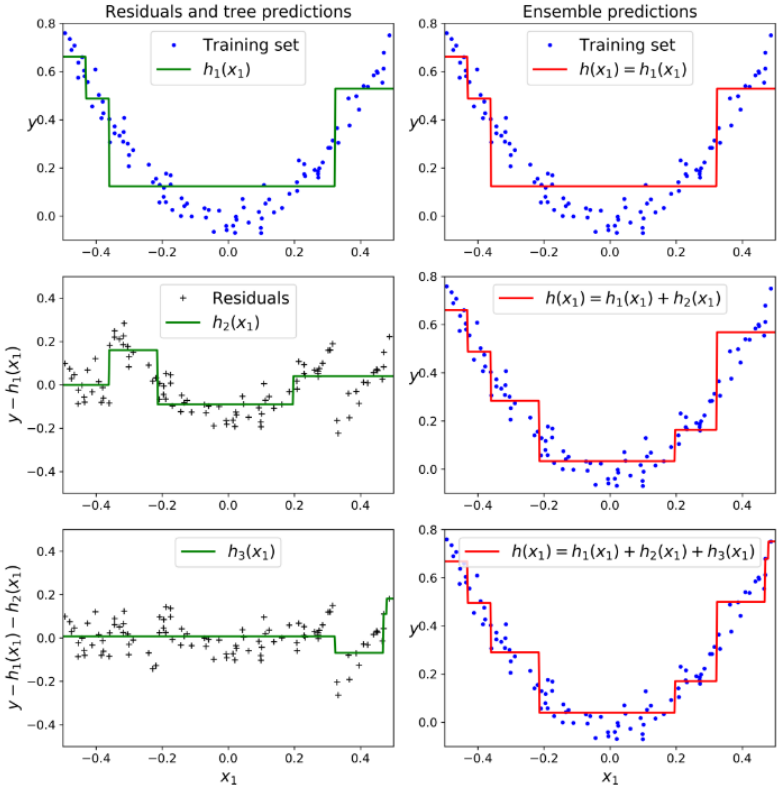


Figure 2.2.2: Gradient Boosting .

It is worth noting that an optimized implementation of Gradient Boosting is available in the popular python library XGBoost, which stands for Extreme Gradient Boosting.

2.3 Deep Learning :

Deep learning[3], a facet of AI, seeks to replicate the human brain using artificial neural networks (ANNs). However, it falls short of matching the brain's capabilities. The history of deep learning traces back to the 1960s when mathematician Alexey Ivakhnenko introduced the first multilayer perceptron (MLP) learning algorithm in 1967. In the late 20th century, deep learning faced the AI winter due to computational inefficiencies, causing a decline in

research interest. Interest was reignited with the advancement of hardware, especially GPUs, known for their efficiency in linear algebra computations due to parallelism. This renewed interest transformed deep learning into a cornerstone of modern AI research.

2.3.1 Feedforward Neural Networks

Neural networks[3] can be viewed as algorithms for approximating functions. Specifically, feedforward neural networks have been demonstrated to possess universal approximation capabilities. In simpler terms, for any continuous function, it is possible to find a feedforward neural network that can effectively approximate that function to any desired degree. The Universal Approximation Theorem has also been confirmed for other network architectures like Convolutional Neural Networks (CNNs). This theorem underscores the capacity of neural networks to approximate a wide range of functions.

2.3.2 Structure

Feedforward neural networks are computational structures that generally consist of multiple layers falling into one of three types:

- **Input Layer:** Where the data is introduced into the network.
- **Hidden Layer(s):** Intermediate layers designed to compute features within the input data. These features become more abstract as the layer approaches the output.
- **Output Layer:** This final layer produces the network's output, which can take various forms such as a scalar prediction, a probability distribution, or a lower-dimensional representation, depending on the specific application and network objectives.

The following figure shows two neurons in the output layer, four neurons in the input layer, and two hidden layers, each having six neurons, making up a multilayer perceptron. Each layer within the neural network serves as a representation of the input data. Specifically, the input layer corresponds to the original data itself. The underlying idea is that through the application of a learning algorithm, which involves the systematic adjustment of neuron values, the network can acquire meaningful data representations that contribute to the final

output. We will explore the mechanics of this learning process shortly, but first, let's examine how information flows between these layers.

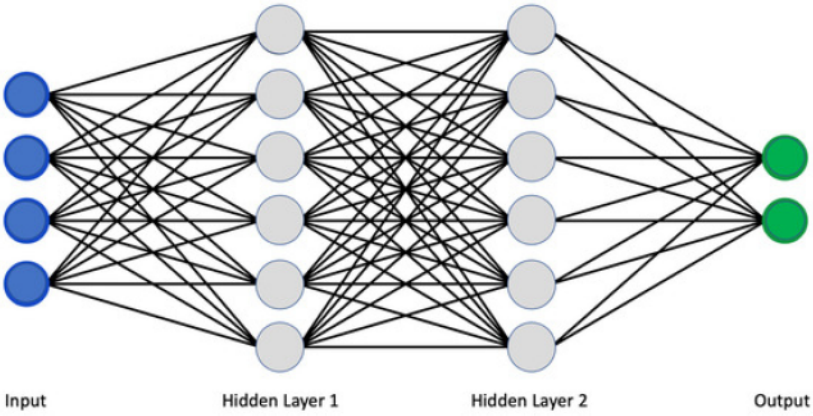


Figure 2.3.1: Feedforward Neural Netwrok[1]

2.3.3 Information passing between layers (Forward Propagation)

The neuron's value is determined by first applying a linear transformation with a bias to the values of neurons in the previous layer and then introducing a non-linear activation function. This process is crucial to enable the network to perform complex, non-linear transformations; without it, the network would only be capable of performing a simple linear mapping function.

To establish a rigorous mathematical framework, let's define important parameters:

- l: the layer's index.
- nl: how many neurons are present at layer l.
- $a^{(l)}$: the values of neurons in layer l, $a^{(l)} = ((a_1^{(l)}, ..., a_{nl}^{(l)}))^{(T)}$.
- $w^{(l)}$: The weight matrix encapsulates the relationships between neurons in two successive layers, denoted as l and $l-1$.
- $b^{(l)}$: bias term, $b^{(l)} = ((b_1^{(l)}, ..., b_{nl}^{(l)}))^{(T)}$

- $f^{(l)}$: The activation function at layer l, also known as non-linearity, is a continuous non-linear function.

The layers' relationship, expressed in this notation, is as follows:

$$a^{(l)} = f^{(l)}(w^{(l)} \cdot a(l-1) + b(l))$$

The choice of the activation function f plays a pivotal role as it empowers the network to capture complex patterns. Typically, these functions are continuous and monotonic. The selection of the activation function for the output layer depends on the nature of the problem. For instance, in classification tasks, the Softmax activation function is often utilized because it produces a probability distribution (alternatively, the sigmoid function may be employed).
Regarding the hidden layers, there are no rigid guidelines for selecting activation functions, but let's explore some commonly used examples:

- Rectified Linear Unit (ReLU): $f_{relu}(x) = Max(0, x)$
- Hyperbolic Tangent (tanh): $f_{tanh}(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$
- Sigmoid (σ): $f_\sigma(x) = \frac{1}{1+e^{-x}}$
- Softmax (softmax): $f_{softmax}(x) = \left(\frac{e^{x_i}}{\sum_{i=1}^{n} e^{x_i}}\right)$
- Leaky ReLU (lrelu): $f_{lrelu}(x) = max(\varepsilon x, x)$ where $0 \leq \epsilon \leq 1$
- Exponential Linear Unit (ELU): $f_{elu}(x) = \begin{cases} \alpha(e^x - 1) & \text{if} \quad x \leq 0 \\ x & \text{else} \end{cases}$

2.3.4 Learning

Loss Function

Essentially, the loss function quantifies the expense borne by the model, often a neural network, resulting from discrepancies in its predictions. To formalize this concept, let's examine a labeled dataset that comprises pairs (xi, yi), where xi denotes inputs and yi represents corresponding outputs. These pairs are indexed from 1 to m, where m signifies

the total number of training examples. The primary goal is to approximate the function that relates inputs (xi) to their corresponding outputs (yi) by employing a neural network.

The idea is that when provided with an input, xi, the network produces a prediction, ŷi, for the output.
Through a continuous process of comparing this prediction to the actual value, yi, the network incrementally improves its predictions while preserving its ability to generalize well. The evaluation of prediction accuracy is determined by a loss (or cost) function, which depends on the weights of the neural network.. The ultimate goal is to minimize this loss. For instance, in a regression problem, the mean squared error (MSE) is a commonly used loss function, defined as $L(w,b) = \frac{1}{m}\sum_{i=1}^{m}(y^{(i)} - \widehat{y}^{(i)})^2$.
In the case of a classification problem, the loss is often the cross-entropy loss, defined as $L(w,b) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)} log(\widehat{y}^{(i)})$, where n represents the number of possible classes.

Gradient Descent

Given that we can measure how good or bad predictions made by a model are, the question is how we can update the model's weights accordingly. This can be accomplished through an algorithm known as gradient descent. The intuition behind this method arises from the fact that a differentiable function experiences the steepest increase when its variables are adjusted in the direction of its gradient, as the gradient directly represents the function's slope. Since we aim to minimize the loss function, we seek to find the model's parameters that minimize the loss. The process begins with random weights and iteratively updates them in the opposite direction of the loss function's gradient (as this is a minimization problem).

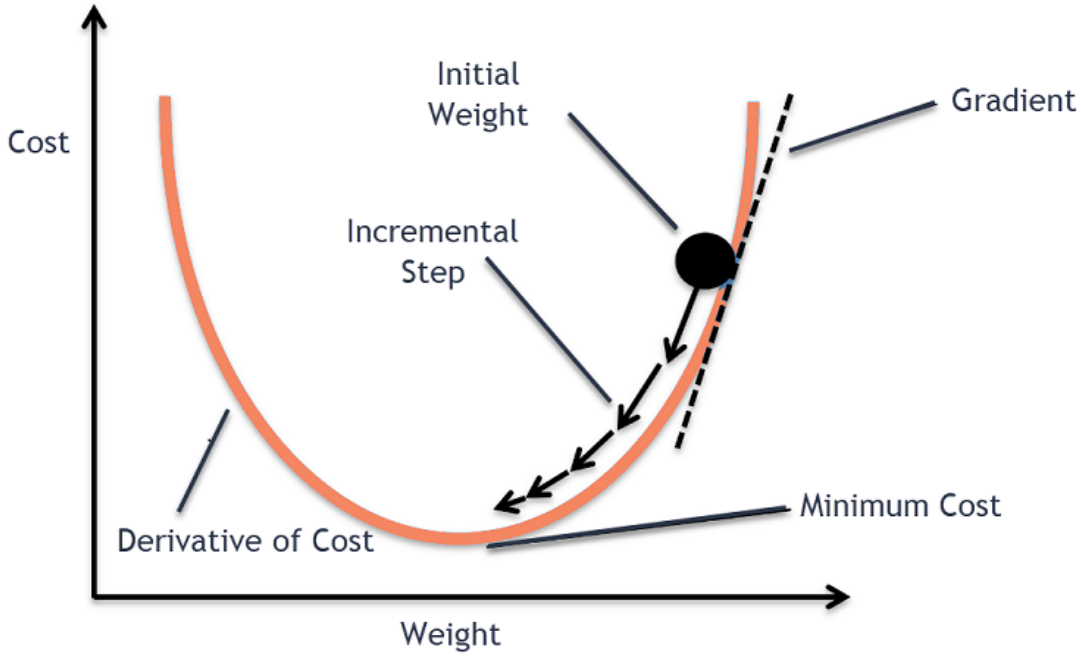


Figure 2.3.2: Gradient Descent Algorithm

If the weights are denoted as Wn, this sequence can be defined as:

$$W_{n+1} = W_n - \eta \cdot \Delta L\left(W_n\right)$$

Where :

- W_{n+1} represents the updated weights .
- W_n represents the current weights .
- η is the learning rate , a hyperparameter that determines the step size .
- $\Delta L\left(W_n\right)$ is the gradient of the loss function with respect to the weights at the current step .

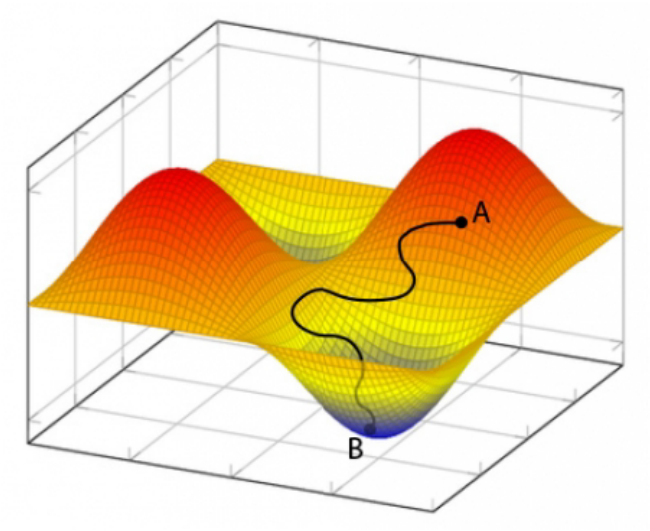


Figure 2.3.3: Visualization of the Progression of Gradient Descent Optimization

Back Prpagation

Backpropagation serves as a systematic and efficient approach to calculating gradient components, particularly the partial derivatives concerning each parameter. During the forward propagation phase, we compute the loss function based on the input data. Once the loss is ascertained,it is propagated backward through the neural network to fine-tune the weights with the goal of minimizing the loss. To simplify this concept, we can represent the interplay between parameters as follows:

$$z^{(l)} = w^{(l)} \cdot a^{l-1} + b^{(l)}$$

Leveraging the notations introduced earlier, we can articulate the backpropagation process between two layers using the following equations:

$$\frac{\partial L}{\partial z^{l-1}} = (w^{(l)})^T \cdot \frac{\partial L}{\partial z^l} \cdot f^{(l-1)'}(z^{(l-1)})$$

$$\frac{\partial L}{\partial w^l} = \frac{\partial L}{\partial z^l} \cdot (a^{(l-1)})^T$$

$$\frac{\partial L}{\partial b^l} = \frac{\partial L}{\partial z^l}$$

It's worth noting that these derivatives are not computed numerically, as this approach is inefficient and prone to uncontrolled error propagation. Instead, automatic differentiation is employed, which involves constructing computational graphs to represent the sequence of operations. For instance, even a simple computation like e = (a + b) * (b + 1) can be effectively represented by a computational graph.

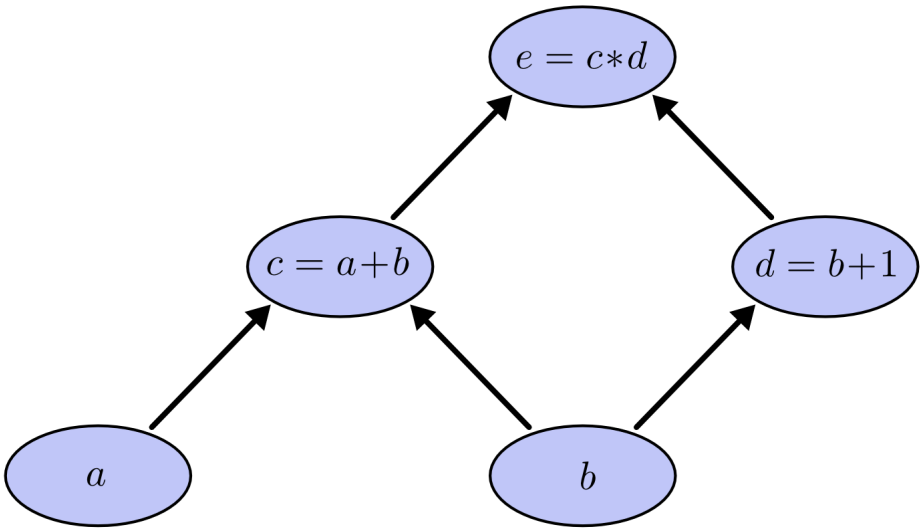


Figure 2.3.4: illustration of a computational graph

if the operations within the nodes are sufficiently straightforward to calculate their individual gradients concerning their inputs, we can systematically compute the gradient of any node concerning another. Consequently, we can methodically determine the gradient throughout the network.

Conclusion

This chapter provided an overview of the current technologies and methods relevant to our study, including deep learning and machine learning, specifically ensemble learning techniques. We covered a range of algorithms as well as their applications, setting the stage

for our methodology choice. The insights acquired here inform the design and implementation of our predictive models, which we will discuss in detail within the next chapters.

By understanding these foundational technologies, we can better tailor our approach to address the specific needs of our car valuation system.

3

Web Scraping and Data :

Introduction

3.1 Introduction to Web Scraping :

Web scraping[4] (or data scraping) is a technique used to collect content and data from the internet. This data is usually saved in a local file so that it can be manipulated and analyzed as needed. If you've ever copied and pasted content from a website into an Excel spreadsheet, this is essentially what web scraping is, but on a very small scale.
However, when people refer to 'web scrapers,' they're usually talking about software applications. Web scraping applications (or bots) are programmed to visit websites, grab the relevant pages and extract useful information. By automating this process, these bots can extract huge amounts of data in a very short time. This has obvious benefits in the digital age, when big data , which is constantly updating and changing , plays such a prominent role.

Its application in this project : In this project, we will leverage web scraping as a pivotal technique during our data acquisition phase. By systematically extracting data from second-hand car seller websites, we aim to curate an up-to-date and reliable dataset enriched with valuable insights regarding the market values of various cars. This

meticulously gathered dataset will serve as the foundation for our subsequent endeavors, particularly in the realm of machine learning.
With access to a comprehensive dataset encompassing diverse car models, specifications, and pricing trends, we will proceed to deploy machine learning algorithms to develop a predictive solution for estimating the market values of cars. This predictive model holds significant promise in revolutionizing the reporting process in the event of traffic accidents, a prevalent issue within the car insurance domain in Tunisia.

3.1.1 How Web scraping functions :

All web scraping bots follow three basic principles:

- Step 1: Making an HTTP request to a server
- Step 2: Extracting and parsing (or breaking down) the website's code
- Step 3: Saving the relevant data locally

Now let's take a look at each of these in a little more detail :
1. Making an HTTP request to a server : When an individual visits a website via a browser, an HTTP request is sent. This is basically the digital equivalent of knocking on the door, asking to come in. Once the request is approved, we can then access that site and all the information on it. Just like a person, a web scraper needs permission to access a site. Therefore, the first thing a web scraper does is send an HTTP request to the site they're targeting.
2. Extracting and parsing the website's code : Once a website gives a scraper access, the bot can read and extract the site's HTML or XML code. This code determines the website's content structure. The scraper will then parse the code (which basically means breaking it down into its constituent parts) so that it can identify and extract elements or objects that have been predefined by whoever set the bot .
3. Saving the relevant data locally: Once the HTML or XML has been accessed, scraped, and parsed, the web scraper will then store the relevant data locally. As mentioned, the data extracted is predefined by the user (having told the bot what to

collect). Data is usually stored as structured data, often in an Excel file, such as a ".csv" or ".xls" format.

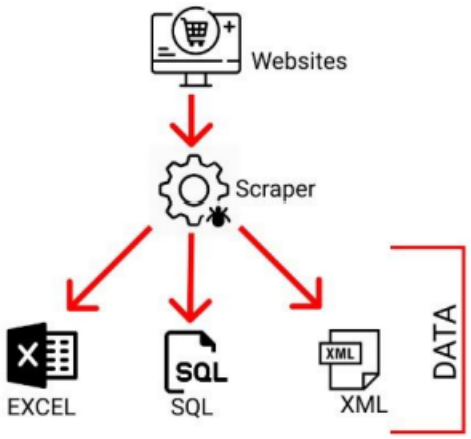


Figure 3.1.1: The procedure of web scraping

3.1.2 Important Tools :

Selenium : Selenium is a popular open-source umbrella project consisting of several tools, servers and libraries that help to automate different browsers. The main remote control interface, “Selenium WebDriver," enables the control of user agents and making connections with browsers to mimic the different actions humans can perform on them. The tool is quite popular in the web application testing world. But it can go beyond the testing realm and help people scrape websites. Since selenium supports multiple languages, people can easily use it to scrape websites by writing code of their choice.

Figure 3.1.2: selenium logo

BeautifulSoup :

Beautiful Soup is a Python library designed for parsing HTML and XML documents, making it an excellent choice for web scraping tasks. It provides developers with a simple and intuitive interface for navigating and extracting data from web pages. At its core, Beautiful Soup acts as a parser that takes raw HTML or XML markup and transforms it

into a structured tree-like data structure, known as the parse tree. This parse tree allows developers to easily navigate through the document's elements, access specific tags or attributes, and extract desired information.
In the context of web scraping, Beautiful Soup excels at extracting data from static web pages where the content is already present in the HTML markup. Developers typically use Beautiful Soup in conjunction with other libraries, such as requests, which fetches the HTML content of web pages, to create robust scraping workflows.
To scrape data using Beautiful Soup, developers first fetch the HTML content of the target web page using the requests library. They then pass this HTML content to Beautiful Soup, which parses it and creates a parse tree. Developers can then use Beautiful Soup's intuitive methods and functions to navigate the parse tree, search for specific elements based on tags or attributes, and extract the desired data.

Figure 3.1.3: BeautifulSoup logo

Visual Studio : In the domain of software development environments, Visual Studio Code (VSCode) emerges as a premier tool cherished for its versatility and user-centric design. Developed by Microsoft, VSCode serves as an open-source integrated development environment (IDE) offering a rich array of features and extensions tailored to accommodate a multitude of programming languages and project types. With its intuitive interface and seamless integration with various plugins, VSCode facilitates streamlined development workflows, enabling developers to write, debug, and deploy code with ease.
In the context of our project, VSCode will play a pivotal role in orchestrating the web scraping endeavor, as we harness its robust Python support to craft intricate scripts for harvesting data from prominent second-hand car sales websites such as "tayara.tn" and "automobile.tn". Leveraging the power of VSCode, we aim to optimize the efficiency and accuracy of our scraping process, laying a solid foundation for subsequent data analysis and model development.

[5]

Figure 3.1.4: Visual studio logo

3.2 Data acquisition :

The first and most important step in this project will have to be acquiring the necessary data , which will be analyzed then serve as the backbone of the solution . The key to creating a healthy and robust Machine Learning model is undoubtedly the quality as well as the integrity of the training data . In this section , we delve into the process of obtaining and refining our dataset to ensure its suitability for training purposes. Firstly , we'll detail the methodologies used in sourcing and wrangling the data. Furthemore , we explain the steps taken to reprocess and refine the acquired data. By meticulously documenting our data acquisition procedures, we establish a solid framework for the subsequent stages of our project and put emphasis on our commitment to excellence and precision in every facet of our analysis.

3.2.1 Implementing Web Scraping :

3.2.2 Data source :

After thorough research, we have identified two prominent websites from which to extract data: "Tayara.tn" and "Automobile.tn." These platforms are renowned online marketplaces, each catering to distinct niches within the automotive sector. "Tayara.tn" stands out as a versatile platform offering a wide array of products and services, including a dedicated section for showcasing second-hand cars available for purchase. On the other

hand, "Automobile.tn" specializes in providing a comprehensive platform specifically tailored for the selling and exhibition of automobiles. These websites serve as invaluable sources of data, offering a wealth of information on various car models, specifications, and pricing trends. By harnessing the data available from these reputable platforms, we aim to cultivate a comprehensive dataset that accurately reflects the dynamic landscape of the automotive market in Tunisia.

3.2.3 data Extraction :

Let's dive into the example of "Automobile.tn" by looking into snippets of our scraper code and explaining their content .

Inspecting the webpage

The main key to succesfully extract data from the world wide web is to understand the html structure of the web page in question . Let's check the source code by right-clicking the web interface then hiting the "inspect" button .This allows us to see all the images, links, and CSS codes that form the site.

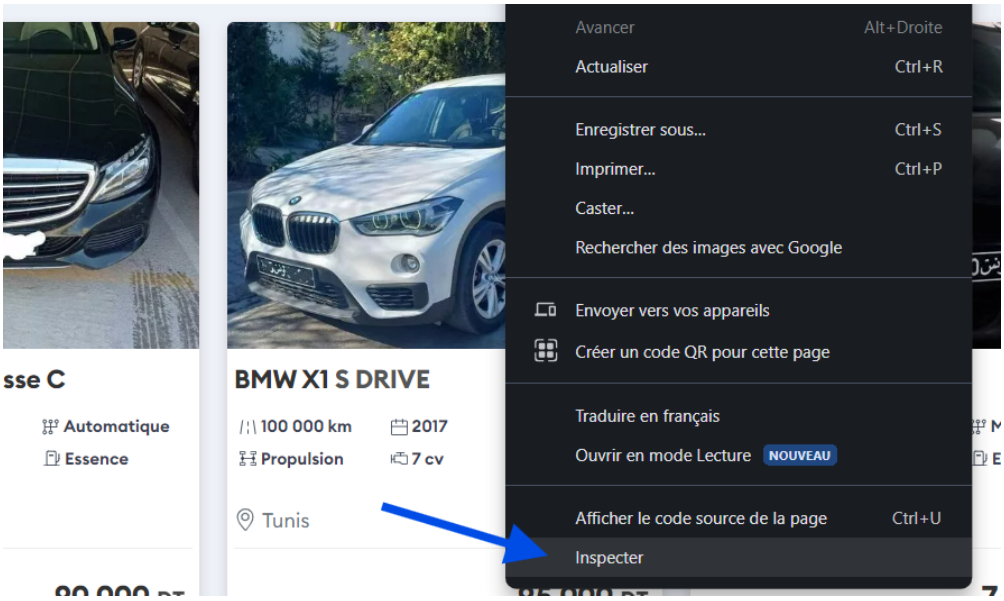


Figure 3.2.1: inspecting the webpage

We should have the browser's elements console pop up :

By navigating this console , we can decypher the underlying structure used to contain each displayed item on the page .

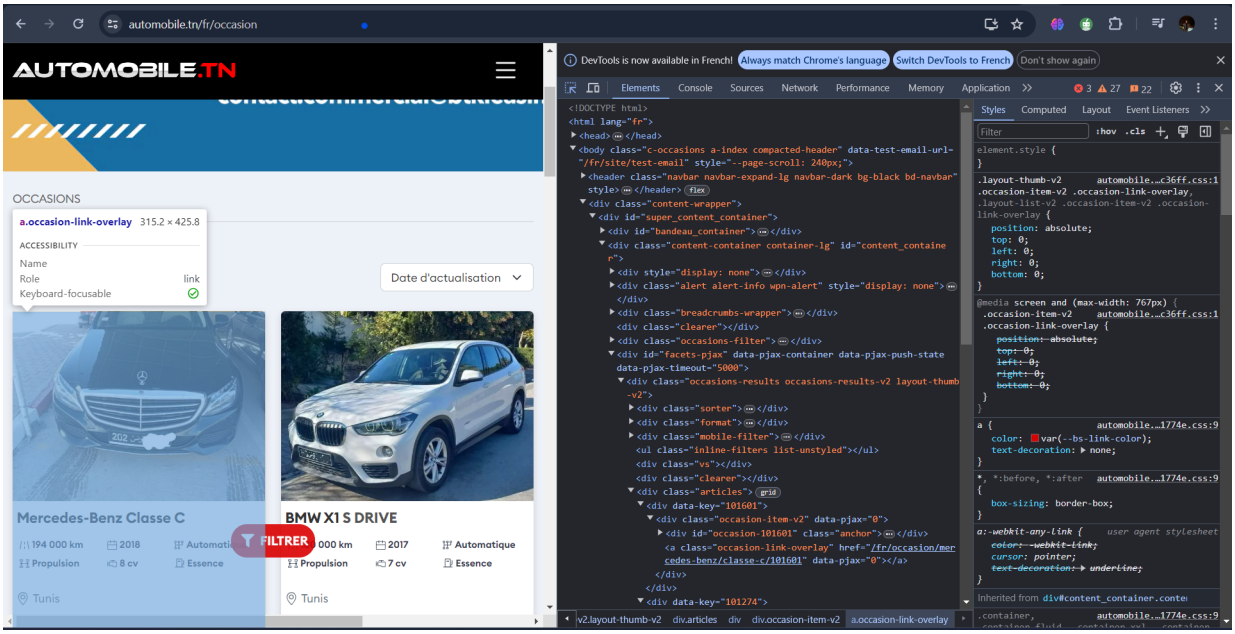


Figure 3.2.2: image of the console

The importance of the tools

Beautiful Soup is a Python library for getting data out of HTML, XML, and other markup languages. Beautiful Soup helps us pull particular content from a webpage, remove the HTML markup, and save the information. It is a tool for web scraping that helps clean up and parse the documents we have pulled down from the web by selecting specific HTML tags.

3.2.3.1 Dynamic Website Challenge

We sadly hit a brick wall at this stage since not all elements were successfully fetched with beautifulsoup's methods , as a matter of fact , the parsed html isn't the same as the html we inspect on the webpage console.

Upon further investigation, we saw that this was a dynamic web page generating most elements with JavaScript.

This means that when the page initially loads, not all content may be present in the HTML source retrieved by BS. This poses a challenge when trying to scrape data using

BeautifulSoup alone because the data we're interested in might not be present in the initial HTML.

After some research of other Python based web scraping packages, we came across Selenium. The Selenium package is used to automate web browser interaction from Python. With Selenium, programming a Python script to automate a web browser is possible. Using Selenium, parsing the JavaScript was easily done . Selenium starts a browser session , by creating a new instance of a web driver, in our case , it was a chrome driver , which allows interacting with web pages using Chrome .

3.2.3.2 Solution

BeautifulSoup and Selenium Integration:

Fetching and parsing the HTML content

Selenium is great for interacting with dynamic content by simulating user actions such as clicking buttons or scrolling. However, it's not as efficient for parsing HTML as BeautifulSoup.

BeautifulSoup, on the other hand, is excellent at parsing HTML and navigating the HTML tree to extract specific data. By combining these two , we leverage the strengths of both tools: Selenium can be used to navigate to the webpage, interact with dynamic elements, and wait for the content to load.

Once the dynamic content has loaded, we can retrieve the updated HTML source using "driver.page_source". Then, we pass this HTML source to BeautifulSoup for parsing and extraction of the desired data.

```python
from bs4 import BeautifulSoup
from selenium import webdriver
import urllib.parse
from selenium.webdriver.common.by import By

driver = webdriver.Chrome()

base_url = 'https://www.automobile.tn'

driver.get('https://www.automobile.tn/fr/occasion')
html = driver.page_source

soup = BeautifulSoup(html, 'lxml')
```

Figure 3.2.3: bs + selenium code implementation

• As shown in the last figure , we initialize a chrome webdriver using selenium then specify the URL of the page we're scraping ('base_url = 'https://www.automobile.tn').

• Using the "driver.get('')" method , selenium navigates to the target page and retrieves its html content (html = driver.page_source) and stores it in the variable 'html'.

• soup = BeautifulSoup(html, 'lxml') allows us to parse the content and srore it in the variable "soup" using BeautifulSoup.

```python
elements = driver.find_elements(By.CLASS_NAME, "page-item next disabled")
check = 0
while check == 0 :
    occasions = soup.find_all('div', class_='occasion-item-v2')



    for occasion in occasions:
        link = occasion.find('a', class_='occasion-link-overlay')['href']
        full_link = urllib.parse.urljoin(base_url, link)
        thumb_cap = occasion.find('div', class_='thumb-caption')
        price_div = occasion.find('div', class_='item-foot')
        price = price_div.find('div', class_='price').text
```

Figure 3.2.4: first segment of the scraping code

In This code snippet , we'll be :

Iterating through pages

we firstly define a while loop , to iterate through all the pages of our website by checking the presence of a "next" button (stored in the "elements" variable) to determine whether there are any additional pages to scrape . if the button is found , we extract the link to the next page and navigate to it using "driver.get()" once we're done with the present page.

fetching the car listings in the current page

Figure 3.2.5: html structure of each car listing

- we set to the "occasions" variable the list of cars displayed in the page
- we iterate through each listing and fetch the link for the car , as well as the displayed price

---

```
driver.get(full_link)
link_html = driver.page_source
link_soup = BeautifulSoup(link_html, 'lxml')
main_specs = link_soup.find('div', class_='main-specs').ul
cylindre_li = link_soup.find('span', class_='spec-name', text=re.compile(r'\bCylindrée\b'))
specifications = link_soup.find('div', class_='col-md-6 mb-3 mb-md-0').find('div', class_='box').find(
    'div', class_='divided-specs').ul
li_contents = []
cyl_content = ''

# Iterate over each li item within the ul
for spec in main_specs.find_all('li'):
    content = spec.find('span', class_='spec-value').get_text(strip=True)
    li_contents.append(content)
```

Figure 3.2.6: Second code snippet

```
for specific in specifications.find_all('li'):
    spec_name = specific.find('span', class_='spec-name').get_text(strip=True)
    spec_value = specific.find('span', class_='spec-value').get_text(strip=True)
    if spec_name == "Couleur extérieure":
        couleur_ext = spec_value
    if spec_name == "Couleur intérieure":
        couleur_int = spec_value
    if spec_name == "Sellerie":
        sellerie = spec_value
    if spec_name == "Nombre de places":
        nbre_places = spec_value
    if spec_name == "Nombre de portes":
        nbre_portes = spec_value
    if spec_name == "Modèle":
        modele = spec_value
    if spec_name == "Marque":
        marque = spec_value


nature ="occasion"
Kilométrage = li_contents[0]
Mise_en_circulation = li_contents[1]
Énergie = li_contents[2]
Boite_vitesse = li_contents[3]
Puissance_fiscale = li_contents[4]
Transmission = li_contents[5]
Carrosserie = li_contents[6]
Date_de_l_annonce = li_contents[7]
cylindrée = cyl_content
```

Figure 3.2.7: third code snippet

scraping every individual car on the page

• we use the scraped link and navigate to it using selenium then proceed to scrape its content using :

driver.get(full_link)

link_html = driver.page_source

link_soup = BeautifulSoup(link_html, 'lxml')

• we analyse the new html tree of the individual car pages , and look for the tags containing the desired specifications

Figure 3.2.8: the page of a specific car

Fetching the cars' specs :

• we find the "div" tag containing our target data and and store the unordered list of descriptions in our "specifications" variable .

• we do the same for the other spec containers ("cylindre_li" and "main_specs")

• we iterate through the "specifications" list , identify the name of each spec (spec_name = specific.find('span', class_='spec-name').get_text and fetch its value (spec_value = specific.find('span', class_='spec-value').get_text)

• we then assign each spec to its designated variable (in text form) and do the same process to the other containers .

```
    data.append([nature , Kilométrage.strip(), Mise_en_circulation.strip(), Énergie.strip(), Boite_vitesse.strip(),
                 Puissance_fiscale.strip(), Transmission.strip(), Carrosserie.strip(), Date_de_l_annonce.strip(),
                 cylindrée.strip(), couleur_ext.strip(), couleur_int.strip(), sellerie.strip(), nbre_places.strip(),
                 nbre_portes.strip(), marque.strip(), modele.strip(), price.strip()])
print('page ',i,' succesfully scraped')
i+=1
pages_div= soup.find('div', class_='pages')
if pages_div.find('li', class_='page-item next disabled') :
    print('last page')
    check=1
else :
    print('mezelna mouch fel last page ')
    nextBtn_link = pages_div.find('li', class_='page-item next').find('a')['href']
    next_link = urllib.parse.urljoin(base_url,nextBtn_link)
    driver.get(next_link)
    next_html = driver.page_source
    soup = BeautifulSoup(next_html, 'lxml')
```

Figure 3.2.9: fourth code snippet

Data Structuring and Storage

• the extracted data is structured into a list of lists (the 'data' variable) , where each entry represents the data for a single car. The 'data' is therefore appended in each car page iteration .

Redirecting towards the next page

• once all the listings are scraped and their extracted descriptors are stored , the code checks the presence of the "next" button in the footer of the webpage , fetches its url and re-navigates the selenium web driver to it .

nextBtn_link = pages_div.find('li', class_='page-item next').find('a')['href']

next_link = urllib.parse.urljoin(base_url,nextBtn_link)

driver.get(next_link)

• If the scraper is in the last page , the button won't be clickable and we'll find a "<li>" tag in the html assigned to a class "page-item next disabled" , in that case we'll assign 1 to the " check" variable which terminates the while loop defined on top.

```
# Close the Selenium WebDriver
driver.quit()
```

Figure 3.2.10: Finishing the data scraping process

• we finally use the "driver.quit()" function to simply close the web browser instance and terminate the web driver session.

3.3 Data comprehension :

After scraping the target websites , we concatenated the results in a unified database . we will now leave vscode and transition into our new workstation , which is the kaggle jupyter notebook . this workspace will allow better data visualization and facilitate our understanding of the data.

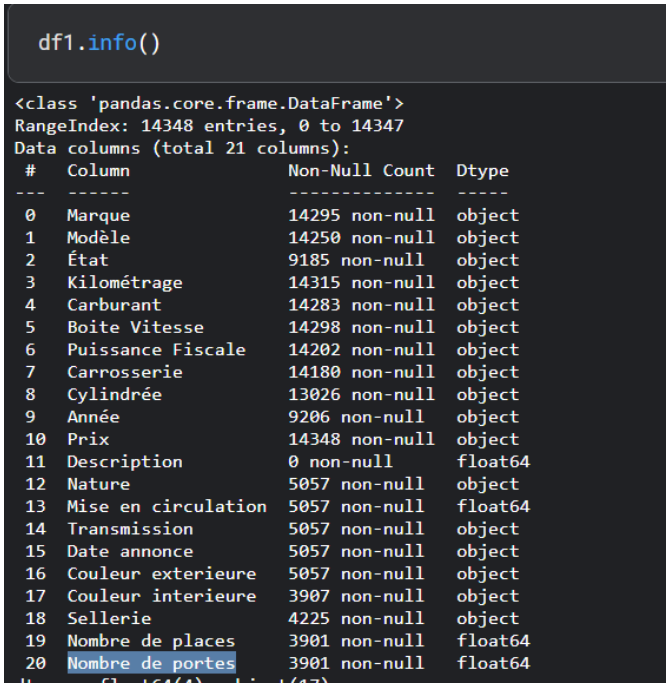
3.3.1 introduction

In this stage of the project , it is most valuable to comprehend the underlying structure and characteristics of the dataset .
Understanding the composition of the data is essential for informed decision making throughout the analysis process .
To assure this , we will , in this section, explore various aspects of our accumulated data , including the types of variables present , their distributions , and any potential patterns or anomalies . Therefore , through comprehensive data comprehension, we can unlock the full potential of the dataset and make informed decisions to drive the success of our project.

3.3.2 initial dataset overview :

For the moment , our dataset is an excel file containing 20 columns , each row represents a car sample with different descriptors and finally the displayed price .
We have 14348 data entries (rows) and our columns currently are as follow :

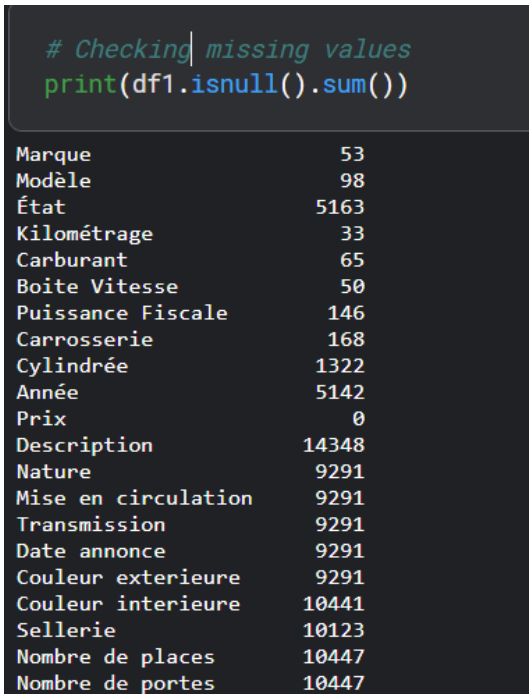
- Marque
- Modèle
- État
- Kilométrage
- Carburant
- Boite Vitesse
- Puissance Fiscale
- Carrosserie
- Cylindrée
- Année
- Description
- Nature
- Mise en circulation
- Transmission
- Date annonce
- Couleur exterieure
- Couleur interieure
- Sellerie
- Nombre de places
- Nombre de portes



```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14348 entries, 0 to 14347
Data columns (total 21 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Marque               14295 non-null  object
 1   Modèle               14250 non-null  object
 2   État                 9185 non-null   object
 3   Kilométrage          14315 non-null  object
 4   Carburant            14283 non-null  object
 5   Boite Vitesse        14298 non-null  object
 6   Puissance Fiscale    14202 non-null  object
 7   Carrosserie          14180 non-null  object
 8   Cylindrée            13026 non-null  object
 9   Année                9206 non-null   object
 10  Prix                 14348 non-null  object
 11  Description          0 non-null      float64
 12  Nature               5057 non-null   object
 13  Mise en circulation  5057 non-null   float64
 14  Transmission         5057 non-null   object
 15  Date annonce         5057 non-null   object
 16  Couleur exterieure   5057 non-null   object
 17  Couleur interieure   3907 non-null   object
 18  Sellerie             4225 non-null   object
 19  Nombre de places     3901 non-null   float64
 20  Nombre de portes     3901 non-null   float64
```

(a) results of ".info()"



```
# Checking missing values
print(df1.isnull().sum())
```

```
Marque                53
Modèle                98
État                5163
Kilométrage           33
Carburant             65
Boite Vitesse         50
Puissance Fiscale    146
Carrosserie          168
Cylindrée           1322
Année               5142
Prix                   0
Description        14348
Nature              9291
Mise en circulation 9291
Transmission        9291
Date annonce        9291
Couleur exterieure  9291
Couleur interieure 10441
Sellerie           10123
Nombre de places   10447
Nombre de portes   10447
```

(b) null values count

Figure 3.3.1: general description of the initial dataset

Throughout the scraping process , we extracted all available informations about the cars in both websites .

this inevitably resulted in our dataset having way too many columns , but it's important to only keep the ones we actually need and the eleminate those that will hinder the machine learning process ..

Irrelevant Columns :

Ultimately , our two target websites had different information about the displayed cars . The most important descriptors were found in both , but a lot of uncommun features were scraped in the process.

This phenomenon results in an unbalanced dataset , with columns containing a huge number missing values as shown in the last figure. Therefore , we'll start by dropping the largely sparse columns in order to :

• **reduce negative impact on our models performance :**

These incomplete columns can negatively impact the performance of machine learning algorithms by introducing noise or irrelevant information, making it harder for the model to learn meaningful patterns from the data.

• **Assure simplicity and efficiency :**

Dropping sparse columns simplifies the dataset and reduces its dimensionality, which can lead to more efficient analysis as fewer features need to be processed, stored, or manipulated.

3.3.3 Adjusted dataset :

After our cleaning session, the dataset now looks like this :

df

	Marque	Modèle	Kilométrage	Carburant	Boite Vitesse	Puissance Fiscale	Carrosserie	Année	Prix	Ann_Obtention
0	Volkswagen	Golf	162000	Essence	Manuelle	6.0	Compacte	2014	47000DT	NaN
1	Renault	Clio	150000	Essence	Manuelle	5.0	Utilitaire	2021	38000DT	NaN
2	Chevrolet	Cruze	121000	Essence	Manuelle	7.0	Berline	2017	32500DT	NaN
3	Renault	Laguna Estate	5868559	Diesel	Manuelle	6.0	Autres	2001	13000DT	NaN
4	Hyundai	Grand i10	90000	Essence	Manuelle	5.0	Berline	2021	44500DT	NaN
...	...	...	...	...	...	...	...	...	...	...
14343	Haval	H6	80 000km	Essence	Automatique	9cv	SUV/4x4	NaN	68 000 DT	11.2017
14344	Seat	Ibiza	80 000km	Essence	Manuelle	5cv	Citadine	NaN	53 000 DT	9.2020
14345	Volkswagen	Golf 7	145 000km	Essence	Manuelle	5cv	Berline	NaN	46 000 DT	6.2013
14346	Chery	Tiggo 2	45 000km	Essence	Manuelle	6cv	SUV/4x4	NaN	Sous leasing 49 400 DT	5.2020
14347	Peugeot	2008	175 000km	Essence	Manuelle	5cv	SUV/4x4	NaN	35 000 DT	5.2016

14348 rows × 10 columns

Figure 3.3.2: adjusted dataset overview

our determinde set of columns includes :

- Marque (brand)

- Modèle (model)
- Kilométrage (mileage)
- carburant (fuel type)
- Boite vitesse (gearbox type)
- Puissance fiscale (fiscal power)
- Carosserie (body type)
- Année (year of manufacture) : for the "tayara.tn" samples
- Prix (price)
- Ann_obtention (year of purchase) : for the "automobile.tn" samples

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 14348 entries, 0 to 14347
Data columns (total 10 columns):
 #   Column             Non-Null Count  Dtype
---  ------             --------------  -----
 0   Marque             14295 non-null  object
 1   Modèle             14250 non-null  object
 2   Kilométrage        14315 non-null  object
 3   Carburant          14283 non-null  object
 4   Boite Vitesse      14298 non-null  object
 5   Puissance Fiscale  14202 non-null  object
 6   Carrosserie        14180 non-null  object
 7   Année              9206 non-null   object
 8   Prix               14348 non-null  object
 9   Ann_Obtention      5057 non-null   float64
dtypes: float64(1), object(9)
```

(a) output for ".info()"

```
print(df.isnull().sum())
```

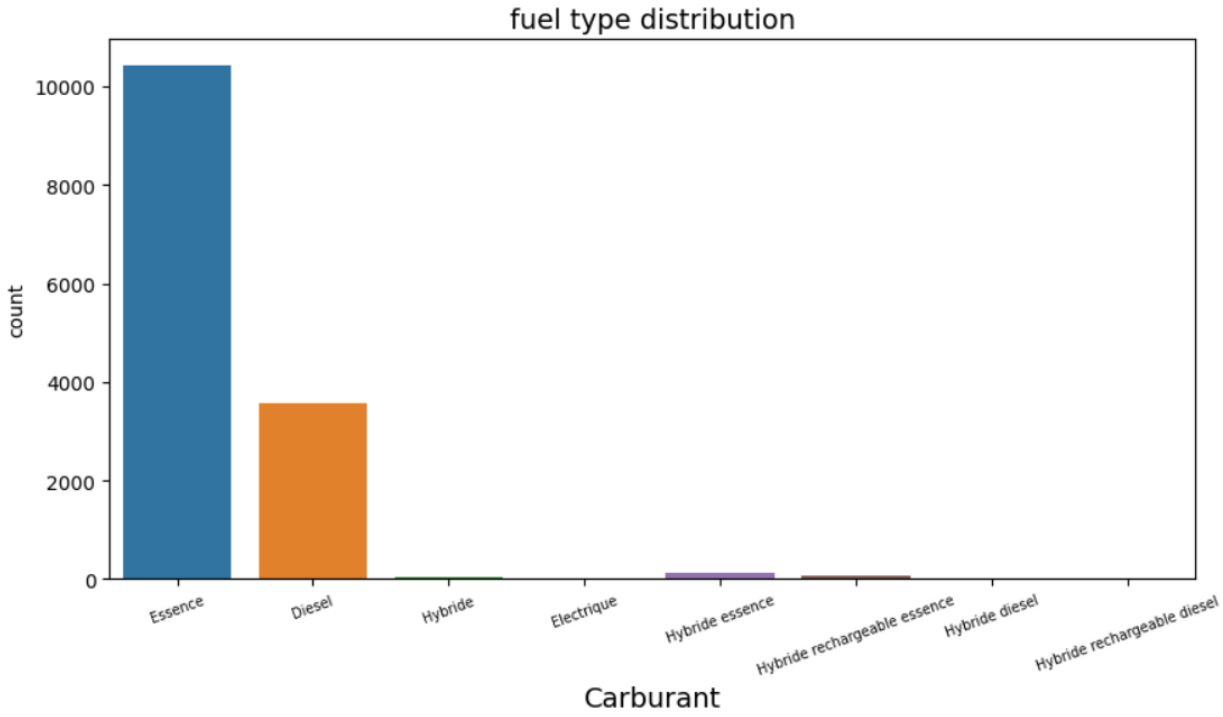
```
Marque               53
Modèle               98
Kilométrage          33
Carburant            65
Boite Vitesse        50
Puissance Fiscale   146
Carrosserie         168
Année              5142
Prix                  0
Ann_Obtention      9291
```

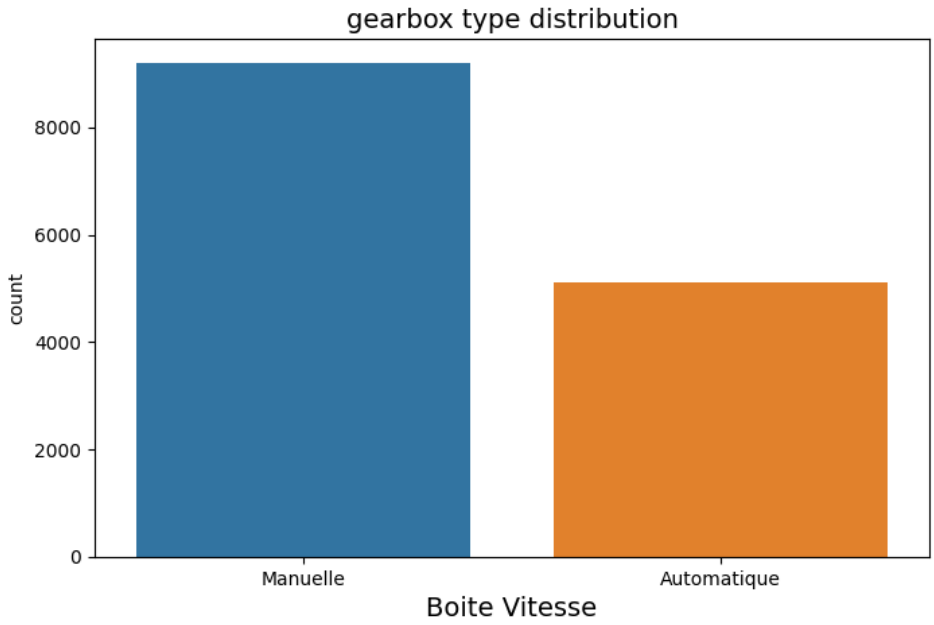
(b) null values count

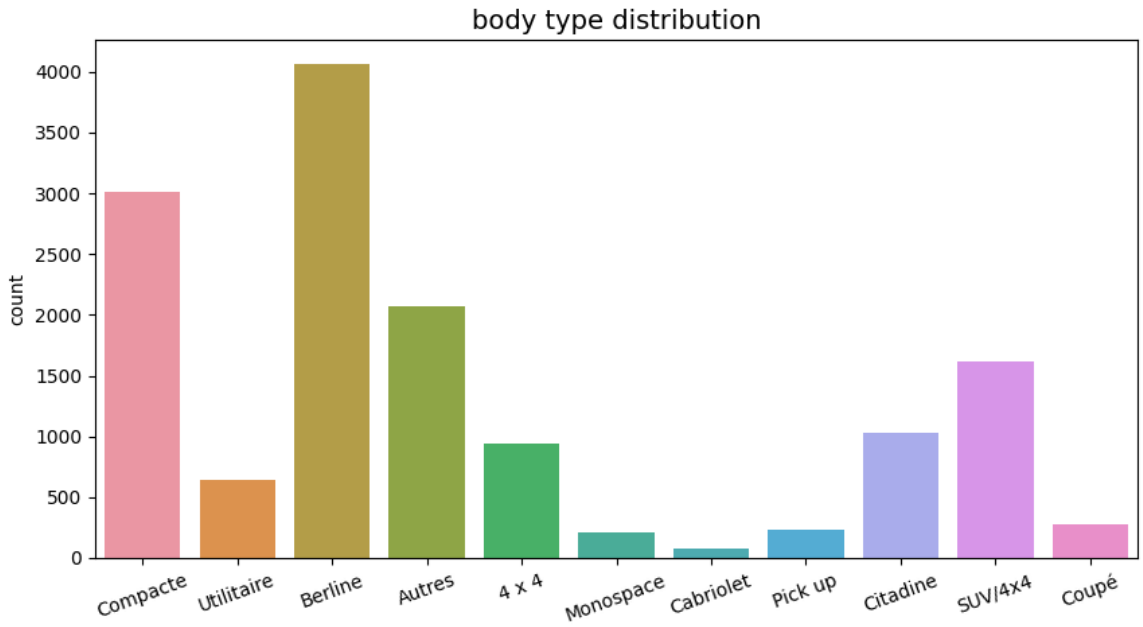
Figure 3.3.3: general description of the adjusted dataset

3.3.3.1 data distributions :

- **Some categorical variables distribution :**


fuel type distribution
count
10000
8000
6000
4000
2000
0
Essence
Diesel
Hybride
Electrique
Hybride essence
Hybride rechargeable essence
Hybride diesel
Hybride rechargeable diesel
Carburant


gearbox type distribution
count
8000
6000
4000
2000
0
Manuelle
Automatique
Boite Vitesse



3.4 Data preparation :

Data preparation [6](also referred to as "data preprocessing") is the process of transforming raw data so that data scientists and analysts can run it through machine learning algorithms to uncover insights or make predictions.

why is it important ?

Most machine learning algorithms require data to be formatted in a very specific way, so datasets generally require some amount of preparation before they can yield useful insights. Some datasets have values that are missing, invalid, or otherwise difficult for an algorithm to process. If data is missing, the algorithm can't use it. If data is invalid, the algorithm produces less accurate or even misleading outcomes. Some datasets are relatively clean but need to be shaped (e.g., aggregated or pivoted) and many datasets are just lacking useful business context (e.g., poorly defined ID values), hence the need for feature enrichment. Good data preparation produces clean and well-curated data which leads to more practical, accurate model outcomes.

theser are the steps we took in order to reprocess our data and make it compatibe for

training our machine learning and deep learning models :

3.4.0.1 Data cleaning

The first step we take when preparing data for machine learning is to clean it. Cleaning data involves finding and correcting errors, inconsistencies, and missing values. There are several approaches to doing that:

• **Handling irrelevant data** Irrelevant data refers to data that is not useful or applicable to solving the problem. Handling irrelevant data can help reduce noise and improve prediction accuracy. Once identified, such data points are removed from the dataset. In our case , the initial scraped data contained various inconsistent and irrelevant data , such as the columns "État", "Description", and "Nature", which contained no relevant information to the assessment of the used cars' prices. We , therefore , handled this issue in the last chapter by dropping irrelevant columns from our dataset

• **Handling missing data**

Missing values is a common issue in machine learning. It can be handled by imputation (filling in missing values with predicted or estimated data), interpolation (deriving missing values from the surrounding data points), or deletion (simply removing rows or columns with missing values from a dataset.)

Since the data in our project are from real-life instances , we shy away from filling in missing values ourselves in order to assure the fiability of the data . That's why we simply opted for dropping rows (car listings) containing null values.

• **Handling outliers** Outliers are data points that significantly differ from the rest of the dataset. Outliers can occur due to measurement errors, data entry errors, or simply because they represent unusual or extreme observations. Outliers can be handled by removing, transforming them to reduce their impact, winsorizing (replacing extreme values with the nearest values that are within the normal range of distribution), or treating them as a separate class of data.

Since the listed cars in the scraped websites were manually posted by everyday users , many of our cars had abnormal values such as a mileage of 0 or a price of 9999999 DT.

To fix this problem, we handled the outliers by defining normal margins of values in each numerical column , these margins are determined after common logic as well as an analysis

of the values' distribution. for example :

```
# Drop rows with 'Kilometrage' values larger than 10,000,000 and less than 1,000
data = data[(data['Kilométrage'] <= 10000000) & (data['Kilométrage'] >= 1000)]
```

similar normalization of values' range is done for the other numerical columns ("year" , "price" and "fiscal power") .

• **Removing duplicates** Another step in the process of preparing data for machine learning is removing duplicates. Duplicates don't only skew ML predictions, but also waste storage space and increase processing time, especially in large datasets.
we can simply remove all duplicates from the dataset with a simple line of code :


```
# Drop rows with 'Kilometrage' values larger than 10,000,000 and less than 1,000
data = data[(data['Kilométrage'] <= 10000000) & (data['Kilométrage'] >= 1000)]
```


• **Handling incorrect data** Data preparation for machine learning must also include handling incorrect and erroneous data. Common techniques of dealing with such data include data transformation (changing the data so that it meets the set criteria) or removing incorrect data points altogether.
we implemented both of these techniques by creating a transformation pipeline for the columns that needed correction.

```python
data['Prix'] = data['Prix'].apply(lambda x: extract_numerical_value(x))

# delete spacings
data['Prix'] = data['Prix'].str.replace(' ', '')

# Replace 'DT' by ''
data['Prix'] = data['Prix'].str.replace('DT', '')

# Convert it into numerical
data['Prix'] = pd.to_numeric(data['Prix'], errors='coerce')

# drop non-numerical values
data.dropna(subset=['Prix'], inplace=True)

# Convert into int
data['Prix'] = data['Prix'].astype(int)
```

Figure 3.4.1: transformation pipeline for "price" column

these are the results of the data transformation step : Let's compare the "mileage" column and the 'price' column before and after undergoing the transformation pipeline.

```python
display(data['Prix'].sample(n=100))
```

```
2967       68000DT
3205       61000DT
2269       39000DT
9039     64 000 DT
5993       32000DT
           ...
7827     78 000 DT
13174   111 000 DT
4107        9300DT
5113       41500DT
3094       25000DT
```

(a) sample before

```python
display(data['Prix'].sample(n=100))
```

```
6680     46900
11018    85000
1880      7500
10884    56000
401      35000
         ...
9587     87000
188      61000
5488    175000
5853     55000
9829     65000
```

(b) sample after

Figure 3.4.2: price before and after handling incorrect data

```python
display(data['Kilométrage'].sample(n=100))
```

```
7789     67 000km
10311       49900
10888      197000
9296    114 000km
8043    200 000km
          ...
9477     73 000km
1774        34000
11823      269000
10520       51000
4835          180
```

(a) sample before

```python
display(data['Kilométrage'].sample(n=100))
```

```
3541    120000
1595    248000
2932    218000
6805     69000
5133     90000
         ...
7123     70000
3589    120000
2254    120000
5145     62000
1384     94000
```

(b) sample after

Figure 3.4.3: mileage before and after handling incorrect data

These activities help ensure that the training data is accurate, complete, and consistent. Though a big achievement, it is not enough to produce a reliable ML model just yet. So, the next step on the journey of preparing data for machine learning involves making sure the data points in the training data set conform to specific rules and standards. And that stage in the data management process is referred to as data transformation.

3.4.0.2 Data transformation

During the data transformation stage, we convert raw data into a format suitable for machine learning algorithms. That, in turn, ensures higher algorithmic performance and accuracy.

- **Encoding**

Categorical data has a limited number of values, for example, colors, car models, or animal species. Because machine learning algorithms typically work with numerical data, categorical data must be encoded in order to be used as an input. So, encoding stands for converting categorical data into a numerical format. There are several encoding techniques to choose from, but we opted for our dataset to use label encoding, which is a method where categorical variables are converted into integer values. Each unique category value is assigned a unique integer. We then applied label-encoding on the columns "brand" , "model" , "fuel type" , gearbox type" and "body type"

```python
# Encode categorical variables

categorical_cols = ['Marque', 'Modèle', 'Carrosserie', 'Carburant', 'Boite Vitesse']
encoded_value_mappings = {}

for col in categorical_cols:
    label_encoder = LabelEncoder()
    clean_data.loc[:, col] = label_encoder.fit_transform(clean_data[col])

    encoded_value_mappings[col] = {label: category for label, category in enumerate(label_encoder.classes_)}

def save_mappings_to_csv(encoded_value_mappings, file_path):
    data = []
    for col, mapping in encoded_value_mappings.items():
        for encoded_value, categorical_label in mapping.items():
            data.append([col, encoded_value, categorical_label])

    df = pd.DataFrame(data, columns=['Column', 'Encoded_Value', 'Categorical_Label'])
    df.to_csv(file_path, index=False)
```

Figure 3.4.4: labelencoding

• **Normalization** The process of transforming the columns in a dataset to the same scale is referred to as normalization. Every dataset does not need to be normalized for machine learning.

In our project, feature normalization was implemented solely for the neural network model, while the XGBRegressor model was trained using unnormalized values. This decision was based on the inherent properties and requirements of the two different types of models. Neural networks are highly sensitive to the scale of input features; thus, normalization was essential to ensure stable and efficient training. Normalizing the features helped prevent issues such as vanishing or exploding gradients and facilitated faster convergence during gradient descent optimization.

•**the normalization equation :**

$$normalized - value = \frac{(value - mean)}{std}$$

On the other hand, the XGBRegressor, being a tree-based model, is robust to feature scaling. Tree-based models split data based on feature values rather than their magnitudes, making them unaffected by varying feature scales. Therefore, the decision to normalize features only for the neural network model and not for the XGBRegressor was made to leverage the strengths and address the specific needs of each model type.

3.4.1 Preparation Results

in the last section , our dataset underwent all the necessary data preparation steps , from cleaning to transformation . this Pipeline has made our scraped data ready for subsequent machine learning processes.

Now we can confidently say that our data is prepared. Let's take a look at the state of the dataset before and after undergoing this rigorous preparation process to highlight the improvements and readiness of our dataset .

```
display(data)
```

	Marque	Modèle	Kilométrage	Carburant	Boite Vitesse	Puissance Fiscale	Carrosserie	Année	Prix	Ann_Obtention
0	Renault	Capture	121000	Essence	Manuelle	5.0	Autres	2019	43000DT	NaN
1	Seat	Ibiza	97000	Essence	Manuelle	4.0	Compacte	2018	43500DT	NaN
2	Mercedes-Benz	Classe A	135000	Diesel	Automatique	6.0	Compacte	2019	93000DT	NaN
3	Peugeot	Expert	154000	Diesel	Manuelle	5.0	Utilitaire	2019	57000DT	NaN
4	Mercedes-Benz	207D	0	Diesel	Manuelle	0.0	Autres	1981	10000DT	NaN
...	...	...	...	...	...	...	...	...	...	...
14343	Renault	Kadjar	120 000km	Essence	Manuelle	7cv	Citadine	NaN	65 000 DT	11.2017
14344	Toyota	RAV 4	150 000km	Essence	Automatique	9cv	SUV/4x4	NaN	125 000 DT	8.2020
14345	Volkswagen	Polo Sedan	58 000km	Essence	Manuelle	5cv	Berline	NaN	42 000 DT	4.2016
14346	Peugeot	Expert	190 000km	Diesel	Manuelle	9cv	Utilitaire	NaN	Sous leasing 64 800 DT	3.2022
14347	Audi	A5 Sportback	70 000km	Essence	Automatique	8cv	Berline	NaN	122 000 DT	11.2019

14348 rows × 10 columns

Figure 3.4.5: Initial Dataset

```
display(data_shuffled)
```

	Marque	Modèle	Kilométrage	Carburant	Boite Vitesse	Puissance Fiscale	Carrosserie	Année	Prix
8078	47	160	170000	2	0	19	2	2008	78000
8696	47	143	119000	2	0	8	2	2015	115000
7228	47	241	83000	7	0	9	9	2021	193000
13052	20	414	91000	2	1	6	2	2019	44000
8001	47	151	138000	2	0	10	2	2016	145000
...	...	...	...	...	...	...	...	...	...
4230	24	232	120000	2	1	5	10	2019	25000
13783	23	404	80000	0	0	7	8	2018	120000
120	47	143	183000	0	1	7	2	2009	47498
9035	5	458	150000	2	0	7	2	2014	70000
519	54	163	220000	2	0	6	5	2000	19500

8134 rows × 9 columns

Figure 3.4.6: Prepared Dataset

As we can clearly see : the number of entries in the datasets (rows) dropped significantly from 14348 to 8134 , this reduction is the direct consequence of our data-cleaning process. in essence , by reducing the volume of data , we interchangeably enhance its quality . This trade-off is essential to produce the best results in our predicting system and ensure its reliability.

3.4.2 Data splitting

3.4.2.1 XGBRegressor model

For the XGBRegressor model, we divided our dataset into two subsets: a training set (80%) and a test set (the other 20 %) . The training set was utilized to fit the model, allowing it to learn the patterns and relationships within the data by adjusting its parameters. Once the model was trained, its performance was evaluated using the test set, providing an unbiased assessment of its ability to generalize to new, unseen data.

3.4.2.2 Neural network model :

On the other hand , the neural network model required a three-way split of the data: training (60%) , validation (20%) , and test sets (20%) . The training set was used to train the model, where it learned complex patterns and relationships through iterative optimization. The validation set played a crucial role in tuning hyperparameters and monitoring the model's performance during training, helping to prevent overfitting. By periodically evaluating the model on the validation set, we ensured that the model configuration provided the best generalization to unseen data. Finally, the test set was used for the final evaluation of the trained model, providing an unbiased estimate of its predictive accuracy and effectiveness in real-world applications."

Conclusion

With the help of online auto listings, we were able to scrape valuable insights and create a large dataset that we then cleaned, processed, and made suitable for machine learning. Our meticulous data preparation process reduced the dataset from 14,348 to 8,134 entries, empowering its quality and reliability.
This prepared dataset is now ready for model-training and testing. In the upcoming chapter, We will talk about how to put our machine learning models into practice which will set the foundation for predictive analysis.

4

Implementation :

Introduction

4.1 Work environment

In order to properly implement the above-defined comprehensive approach ,we must use the right and appropriate tools, so let us pay particular attention to the technical working environment in this chapter.

4.1.1 Software Development Environment :

4.1.1.1 Kaggle :

Kaggle emerges as a central hub for data enthusiasts and practitioners alike. Renowned for its vast repository of datasets, competitions, and collaborative platforms, Kaggle offers an invaluable resource for researchers and professionals seeking to explore, analyze, and share data-driven insights.

As we embark on our project journey, Kaggle will serve as a strategic repository for storing and accessing the datasets pivotal to our analysis and model training endeavors.

Additionally, Kaggle Notebooks will emerge as a key component of our machine learning

pipeline, providing a versatile environment for prototyping, experimentation, and iterative refinement of our predictive models.
By harnessing the collaborative capabilities of Kaggle, we aim to foster a dynamic ecosystem where insights are shared, knowledge is cultivated, and innovative solutions are forged, propelling our project towards its ultimate objectives with unwavering momentum.

Figure 4.1.1: Kaggle logo

4.1.2 Technical choices :

Robust programming languages, frameworks, and libraries are necessary to manage the benchmarking of numerous solutions. We selected Python as the programming language to train and test our models in order to solve this issue, in addition to a variety of other technologies. Here are the primary packages utilized for this project:
Pandas :
The pandas library is mainly used for data handling, i.e. to edit, modify and adjust certain components of a DataFrame object. However, pandas is very flexible and can also be used for other tasks such as visualizing data sets in graphs and storing time series values. Like other Python libraries, pandas is open source, meaning it can be used, modified and redistributed freely.

Figure 4.1.2: pandas logo

Sickit-Learn : Scikit-learn (also known as sklearn) is a free machine learning library for Python. It offers various classification, regression and grouping algorithms, including k-means, CAHs and DBSCANs. It is designed to interact with the NumPy and SciPy digital and scientific Python libraries.

Figure 4.1.3: sickit-learn logo

NumPy : Numpy is a key library for Python in the field of data science, thanks to its support for large-scale multidimensional tables and matrices, as well as a large collection of high-level mathematical functions to operate on these tables. In addition to its obvious scientific application, NumPy can be used as an effective multidimensional container for generic data. It is possible to specify arbitrary data types. This allows NumPy to connect to a wide range of databases smoothly and quickly.

Figure 4.1.4: Numpy logo

Matplotlib : The utility of NumPy and matplotlib is related to numbers - the utility for matplodlib is specifically related to graphic representation tools. In a sense, these resources are therefore more analytical than generative. However, all of this infrastructure works together to enable machine learning programs to produce results that are useful to human manipulators.

Figure 4.1.5: Matplot logo

Seaborn : Based on matplotlib, Seaborn is a Python data visualization package. It offers a sophisticated drawing tool for creating eye-catching and educational statistical visuals. With its extensive collection of built-in themes and color schemes, Seaborn, in contrast to Matplotlib, excels in producing eye-catching and educational statistical visuals. As it generally streamlines the process of creating intricate visuals, it is a priceless tool for presenting and exploring data.
Furthermore, Seaborn seamlessly interfaces with other Python libraries, such as Pandas and NumPy, making data processing and analysis workflows simple. Because of its

emphasis on statistical visualization and user-friendliness, Seaborn is therefore an essential utensil for data scientists and analysts to have in their toolkit. This helps them identify trends in their data and successfully share findings.

Figure 4.1.6: seaborn logo

4.2 Model Development Process

nwari l code li definit fih l models ezzouz ama maghir manehki aal hyperparameters w kol edheka nkhalih lel chap lekhreni

4.2.1 LazyPredict For Training multiple regression models :

4.2.1.1 The regression model choice problem :

Before choosing which machine learning algorithms to train , we have to take in consideration two important points :

- **Point 1: Limited time**

Data scientists have to prioritize. This may mean spending more time on understanding the business problem and identifying the most appropriate approach rather than focusing solely on developing machine learning algorithms.

- **point 2: Machine learning modeling can be time-consuming**

Fine-tuning a machine learning algorithm involves finding the optimal values for these hyperparameters, which can be a trial-and-error process. This takes a long time.

4.2.1.2 AutoML saves the day

AutoML can address these. Lazypredict is one such emerging library, and in this section , we'll address its greatness as well as implement it in our project .

4.2.1.3 What is Lazypredict ?

Lazypredict[7] is a Python package that aims to automate the machine learning modeling process. It works on both regression and classification tasks.
Its key feature is its ability to automate the training and evaluation of machine learning models. It provides a simple interface for defining a range of hyperparameters and then trains and evaluates a model using a variety of different combinations of these hyperparameters.

Lazypredict for Regression : The process we undergo to implement the lazypredict functionalities in our project is very simple , let's go go through it :
After installing the lazy predict package , we'll first import the necessary libraries.

```
from lazypredict.Supervised import LazyRegressor
```

Next, we read the dataset (containing the clean shuffled data as seen in the previous section) so that we can split it into train-test sets.

```
data= pd.read_csv("/kaggle/input/cars-final-version/Final_Shuffled_Clean_Data.csv")
data.head()
```

	Marque	Modèle	Kilométrage	Carburant	Boite Vitesse	Puissance Fiscale	Carrosserie	Année	Prix
0	47	160	170000	2	0	19	2	2008	78000
1	47	143	119000	2	0	8	2	2015	115000
2	47	241	83000	7	0	9	9	2021	193000
3	20	414	91000	2	1	6	2	2019	44000
4	47	151	138000	2	0	10	2	2016	145000

+ Code + Markdown

```
# Separate features and target variable
X = data.drop(columns=["Prix"])  # Adjust target_column_name
y = data["Prix"]  # Adjust target_column_name
# Convert X to float32
X = X.astype(np.float32)

X



xtrain, xtest, ytrain, ytest=train_test_split(X, y, test_size=0.15)
```

Then , we initialize the LazyRegressor object.

```
# Initialize LazyRegressor
reg = LazyRegressor(predictions=True, custom_metric = mean_absolute_error)
```

Now, we will fit multiple regression algorithms with the lazypredict library. This step took 3 seconds in total.
Under the hood, the fit method does the following:

• **Split all features into three categories:** numerical (features which are numbers) or categorical (features which are text)

• **Further split categorical features into two:** 'High' categorical features (which have more unique values than the total number of features) and 'low' categorical features (which have less unique values than the total number of features)

• **Each feature is then preprocessed in this manner:**
Numerical features: Impute missing values with mean, then standardize the feature (removing the mean and dividing by the variance)
'High' categorical features: Impute missing values with the value 'missing', then perform one-hot encoding.
'Low' categorical features: Impute missing values with the value 'missing', then perform ordinal encoding (convert each unique string value into an integer. In the example of a Gender column— 'Male' is encoded as 0 and 'Female' 1.)

• **Fit the training dataset on each algorithm.**

• **Test each algorithm on the testing set :** By default, the metrics are adjusted R-squared, R-squared, root-mean-squared error, and the time taken.

```
# Fit LazyRegressor
models, predictions = reg.fit(xtrain, xtest, ytrain, ytest)
model_dictionary = reg.provide_models(xtrain, xtest, ytrain, ytest)
models
```

the results are :

Model	Adjusted R-Squared	R-Squared	RMSE	Time Taken	$mean_absolute_error$
ExtraTreesRegressor	0.942240	0.942618	14711.351383	1.204173	4193.154313
XGBRegressor	0.917342	0.917884	17598.673509	0.236855	8880.518537
BaggingRegressor	0.910888	0.911472	18272.821546	0.207211	7762.445730
ExtraTreeRegressor	0.907175	0.907783	18649.659003	0.096163	5193.565384
RandomForestRegressor	0.902047	0.902689	19157.821000	2.052629	7357.891415
DecisionTreeRegressor	0.891614	0.892324	20152.313879	0.121914	5625.026481
GradientBoostingRegressor	0.815253	0.816464	26310.356761	0.778664	15580.003389
HistGradientBoostingRegressor	0.801047	0.802351	27303.166201	0.539734	12166.692111
LGBMRegressor	0.793600	0.794954	27809.430752	0.159855	11985.856343
KNeighborsRegressor	0.679852	0.681951	34634.851506	0.139712	14846.912039
PoissonRegressor	0.318441	0.322910	50534.662375	0.079249	26866.882693
LassoCV	0.312808	0.317314	50743.060771	0.436665	28838.224448
BayesianRidge	0.312620	0.317128	50749.988372	0.101614	28845.592278
RidgeCV	0.312549	0.317057	50752.610311	0.121084	28847.130718
Ridge	0.312460	0.316969	50755.895261	0.047749	28849.076903
Lasso	0.312455	0.316963	50756.108816	0.085750	28849.167569
LassoLars	0.312455	0.316963	50756.112561	0.087852	28849.171843
TransformedTargetRegressor	0.312450	0.316959	50756.260605	0.059366	28849.290716
LinearRegression	0.312450	0.316959	50756.260605	0.069948	28849.290716
LarsCV	0.312450	0.316959	50756.262341	0.116939	28849.293420
LassoLarsCV	0.312450	0.316959	50756.262341	0.125003	28849.293420
LassoLarsIC	0.312450	0.316959	50756.262341	0.083001	28849.293420
Lars	0.312450	0.316959	50756.262341	0.049743	28849.293420
SGDRegressor	0.311034	0.315552	50808.504562	0.130602	28973.348936
OrthogonalMatchingPursuitCV	0.309824	0.314350	50853.104735	0.092116	28883.624965
ElasticNet	0.308469	0.313003	50903.024719	0.071129	29523.600580
HuberRegressor	0.293787	0.298418	51440.547313	0.174969	26574.446804
PassiveAggressiveRegressor	0.289925	0.294581	51581.003349	0.185325	26438.696858
TweedieRegressor	0.281200	0.285914	51896.919358	0.097482	30757.127534
GammaRegressor	0.264077	0.268902	52511.440417	0.068753	29164.455985
AdaBoostRegressor	0.206066	0.211272	54541.836610	0.285437	46143.779162
OrthogonalMatchingPursuit	0.195247	0.200524	54912.216127	0.045227	33654.469499
ElasticNetCV	0.025170	0.031563	60436.879739	0.451493	39284.352753
DummyRegressor	-0.007023	-0.000420	61426.728423	0.010760	40238.909369
NuSVR	-0.031524	-0.024760	62169.500481	2.178353	37663.456117
RANSACRegressor	-0.033583	-0.026806	62231.514026	0.662449	29519.146265
SVR	-0.084658	-0.077545	63750.561472	2.971732	37338.730756
MLPRegressor	-0.823201	-0.811245	82652.282872	18.972173	61663.790692
KernelRidge	-1.057079	-1.043590	87793.643762	4.933402	72663.339297
LinearSVR	-1.172160	-1.157916	90215.994688	0.067487	66147.669568
GaussianProcessRegressor	-820.063427	-814.679404	1753985.439861	13.630195	451637.859809

4.2.1.4 Interpreting the results :

This output suggests that ExtraTreesRegressor , XGBRegressor , BaggingRegressor and ExtraTreeRegressor can be potentially explored as machine learning models to predict the prices of our car dataset and are a great fit for our data .
These models have high R-squared values (0.90 - 0.94), indicating good fit to the data. Additionally, they have relatively low mean squared error (MSE) values, suggesting accurate predictions.
Considering the results from LazyPredict evaluations and the promising scope for further fine-tuning, XGBRegressor emerges as our preferred choice for linear regression model training , as discussed in past chapters.
Additionally, XGBoost's flexibility, scalability, and ability to handle complex datasets make it well-suited for our regression tasks, further supporting this decision.

4.2.2 Creating the models :

4.2.3 Model 1 : XGBRegressor

the XGBregressor training process is exceptionally easy and simple , it can be done few code lines as follow :

4.2.3.1 OVERVIEW : Extreme Gradient Boosting : XGBoost

This package was initially developed by Tianqi Chen as part of the Distributed (Deep) Machine Learning Community (DMLC), and it aims at being extremely fast, scalable and portable.

In fact, XGBoost[8] is often an important component of the winning entries in ML competitions, which explains why we opted for XGBoost in our project as a main prediction model , we'll see its implementation in the next chapters , but why don't we learn more about what makes it so special ??

The term "gradient boosting" refers to the optimization strategy used during the training process. It involves sequentially adding weak learners to the model, with each new learner

focusing on correcting the errors made by the existing ensemble.
The “Extreme” in XGBoost emphasizes its ability to handle large datasets efficiently and effectively. It incorporates a number of enhancements over traditional gradient boosting methods, making it a robust and high-performance algorithm.

Key Features of XGBoost :

1. Regularization

XGBoost is equipped with built-in regularization techniques to control the complexity of the model and prevent overfitting. It includes both L1 (Lasso) and L2 (Ridge) regularization terms in its objective function, allowing for better generalization to unseen data.

2. Parallelization

Efficient parallelization is a hallmark of XGBoost. The algorithm is designed to utilize all available CPU cores, making it remarkably faster than many other gradient boosting implementations. This feature is particularly beneficial when dealing with large datasets, where speed is a crucial factor.

3. Handling Missing Values

XGBoost has a robust mechanism for handling missing values, which is a common challenge in real-world datasets. During the training process, it automatically learns the best imputation strategy for missing values, relieving the user from the burden of pre-processing.

4. Tree Pruning

To further enhance efficiency, XGBoost employs a technique called tree pruning. This involves cutting off branches of the tree that do not contribute significantly to the overall model performance. Pruning helps prevent overfitting and leads to a more compact and interpretable model.

5. Cross-Validation

XGBoost supports built-in cross-validation, allowing the user to assess the model’s

performance at each iteration during training. This facilitates the identification of the optimal number of boosting rounds and helps prevent overfitting.

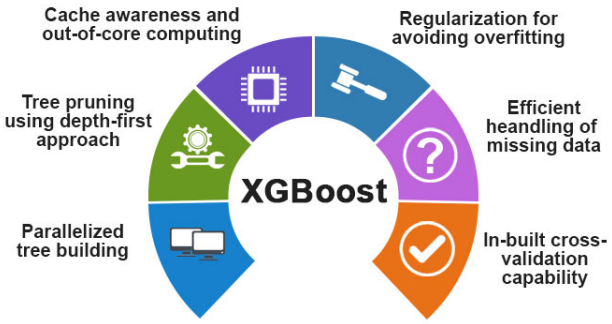


Figure 4.2.1: XGBoost features

How XGBoost's algorithm works :

1. Boosting Iterations

XGBoost builds a predictive model through an iterative process of adding weak learners, typically decision trees, to the ensemble. Each tree corrects the errors made by the existing ensemble, with more emphasis on the instances that were previously misclassified.

2. Objective Function

The optimization objective in XGBoost consists of two parts: a loss function that measures the model's performance, and a regularization term that penalizes overly complex models. The algorithm seeks to find the model parameters that minimize this objective function.

3. Tree Construction

Decision trees in XGBoost are constructed in a depth-first manner. At each step, the algorithm evaluates possible splits based on features and selects the one that maximizes the reduction in the loss function. This process continues until the specified maximum depth of the tree is reached.

4. Gradient-Based Optimization

XGBoost uses a gradient-based optimization technique to iteratively update the model parameters. It calculates the gradient of the objective function with respect to the model predictions and adjusts the parameters in the direction that minimizes the gradient.

4.2.3.2 Creating and fitting

```
import xgboost as xgb
```

```
xgbr = xgb.XGBRegressor(verbosity=0)
xgbr.fit(normed_xtrain, ytrain)
```

Figure 4.2.2: XGBRegressor object .

4.2.4 Model 2 : FNN

4.2.4.1 Deep Learning Tools :

Tensorflow

TensorFlow[9] is a free and open-source software library for machine learning and artificial intelligence. It can be used across a range of tasks but has a particular focus on training and inference of deep neural networks.

It was developed by the Google Brain team for Google's internal use in research and production. The initial version was released under the Apache License 2.0 in 2015. Google released an updated version, TensorFlow 2.0, in September 2019.

Keras

Keras[10] is the high-level API of the TensorFlow platform. It provides an approachable, highly-productive interface for solving machine learning (ML) problems, with a focus on modern deep learning. Keras covers every step of the machine learning workflow, from data processing to hyperparameter tuning to deployment. It was developed with a focus on enabling fast experimentation.

Figure 4.2.3: Tensorflow's Keras

With Keras, we have full access to the scalability and cross-platform capabilities of TensorFlow , these are some of its key components[11] :

Keras API components

Layers

The tf.keras.layers.Layer class is the fundamental abstraction in Keras. A Layer encapsulates a state (weights) and some computation .

Weights created by layers can be trainable or non-trainable. Layers are recursively composable: If you assign a layer instance as an attribute of another layer, the outer layer will start tracking the weights created by the inner layer.

Models

A model is an object that groups layers together and that can be trained on data.

The simplest type of model is the Sequential model, which is a linear stack of layers.

The tf.keras.Model class features built-in training and evaluation methods:

•tf.keras.Model.fit: Trains the model for a fixed number of epochs.

•tf.keras.Model.predict: Generates output predictions for the input samples.

• tf.keras.Model.evaluate: Returns the loss and metrics values for the model; configured via the tf.keras.Model.compile method.

These methods give the user access to other built-in training features.

4.2.4.2 Implementing Tenserflow.Keras

We traditionally start by importing the needed packages on our notebook .

```
# Tensorflow and Keras
import tensorflow as tf
from tensorflow import keras

from sklearn.model_selection import train_test_split

# import NN layers and other componenets.
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense
from tensorflow.keras import optimizers

import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling
```

Figure 4.2.4: Imports

Note :

1. **tensorflow_docs:** This package provides utilities and tools to help with TensorFlow documentation. It includes various submodules and functions that assist in documenting TensorFlow code and workflows.

2. **tensorflow_docs.plots:** This submodule contains functions for generating plots commonly used in TensorFlow documentation. It provides easy-to-use plotting functions tailored for visualizing various aspects of machine learning models.

3. **tensorflow_docs.modeling:** This submodule includes utilities and tools specifically aimed at helping with model development and documentation.

4.2.4.3 Creating the Models Using the Sequential API

Now let's build the neural network! In deep learning , when creating the model , there's no rules or determined structure that assures great results . Therefore , the only way to finding out the optimal structure of our model would be simply tweaking and playing around with the model's architecture . We'll dive deeper into that in the next chapter to finetune our model , but for now we can firstly experiment with different basic structures , that includes trying out different numbers of layers and neurons .

we'll set out the rest of our experiments on three models altogether , one having two hidden layers , the second having three and the last one having 5 hidden layers and we are setting random numbers of neaurons for now .

MODEL 1 : two hidden layers

```
def build_model1():

    model = Sequential()
    model.add(Dense(10, input_shape = (normed_train_data.shape[1],)))
    model.add(Dense(50,Activation('relu')))
    model.add(Dense(50,Activation('relu')))
    model.add(Dense(1))

    return model
```

Figure 4.2.5: Model 1

let's go through this code line by line :

• The first line creates a Sequential model. This is the simplest kind of Keras model, for neural networks that are just composed of a single stack of layers, connected sequentially. This is called the sequential API.

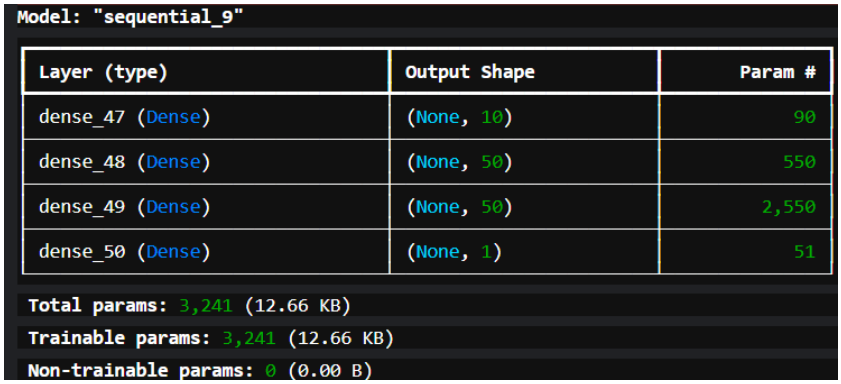
• Next, we build the first layer (input layer) and add it to the model. It is a dense layer , which means , it simply computes a weighted sum of the input features along with bias terms. Its role is to perform a linear transformation on the input data, followed by applying an activation function to introduce non-linearity to the model. Since it is the first layer in the model,we don't specify any activation function for it ,its role is purely linear. It performs a linear transformation on the input data by computing a weighted sum of the input features along with bias terms. we should also specify the input_shape: this includes the shape of the instances. In our case, the input shape is determined by the number of features in the training data. (input_shape = (normed_train_data.shape[1],))

• Next we add a Dense hidden layer with 300 neurons. It will use the ReLU activa- tion function. Each Dense layer manages its own weight matrix, containing all the connection weights between the neurons and their inputs. It also manages a vector of bias terms (one per neuron).

• Next we add a second Dense hidden layer with 100 neurons, also using the ReLU activation function.

• Finally, we add a Dense output layer with a single neuron. This represents the number of predictions our model is going to make.

Since it's a regression task (predicting a continuous value), our model's output will be one prediction (the price) , there's no activation function either. The output layer produces the final output of the model.



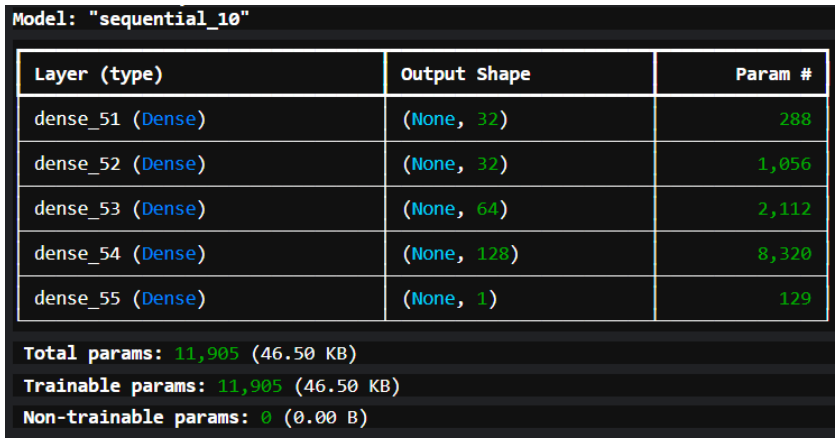
Model: "sequential_9"

Layer (type)	Output Shape	Param #
dense_47 (Dense)	(None, 10)	90
dense_48 (Dense)	(None, 50)	550
dense_49 (Dense)	(None, 50)	2,550
dense_50 (Dense)	(None, 1)	51

Total params: 3,241 (12.66 KB)
Trainable params: 3,241 (12.66 KB)
Non-trainable params: 0 (0.00 B)

Figure 4.2.6: Model 1 summary

MODEL 2 : three hidden layers we added one extra hidden layer in this model .



Model: "sequential_10"

Layer (type)	Output Shape	Param #
dense_51 (Dense)	(None, 32)	288
dense_52 (Dense)	(None, 32)	1,056
dense_53 (Dense)	(None, 64)	2,112
dense_54 (Dense)	(None, 128)	8,320
dense_55 (Dense)	(None, 1)	129

Total params: 11,905 (46.50 KB)
Trainable params: 11,905 (46.50 KB)
Non-trainable params: 0 (0.00 B)

Figure 4.2.7: Model 2 summary

MODEL 3 : five hidden layers

The third model has 5 five hidden layers in total .

Model: "sequential_11"

Layer (type)	Output Shape	Param #
dense_56 (Dense)	(None, 64)	576
dense_57 (Dense)	(None, 256)	16,640
dense_58 (Dense)	(None, 256)	65,792
dense_59 (Dense)	(None, 256)	65,792
dense_60 (Dense)	(None, 256)	65,792
dense_61 (Dense)	(None, 256)	65,792
dense_62 (Dense)	(None, 1)	257

Total params: 280,641 (1.07 MB)
Trainable params: 280,641 (1.07 MB)
Non-trainable params: 0 (0.00 B)

Figure 4.2.8: Model 3 summary

4.2.5 Fine-Tuning Neural Network Hyperparameters

The flexibility of neural networks is also one of their main drawbacks: there are many hyperparameters to tweak. Speaking of which , let's introduce the hyperparameters we worked with in this project :

- **Activation Functions** they determine the output of a neural network's neurons. They introduce non-linearity, enabling the network to learn complex patterns. Common activation functions include ReLU (Rectified Linear Unit), which is used in our models due to its simplicity and effectiveness in handling vanishing gradient problems and is renowned for being the best fit for regression problems .

- **Optimizers :** Optimizers are algorithms that adjust the weights of the network to minimize the loss function. We use the Adam optimizer (Adaptive Moment Estimation), which combines the benefits of both the AdaGrad and RMSProp algorithms, making it suitable for large datasets and high-dimensional parameter spaces. Adam uses adaptive learning rates and momentum to converge faster.

- **Loss Function :** The loss function measures how well the model's predictions match the actual data. For regression tasks, we use the Mean Squared Error (MSE) loss function, which calculates the average squared difference between predicted and actual values, penalizing larger errors more heavily.

- **Metrics :** they are used to evaluate the performance of the model. For our regression task, we use Mean Absolute Error (MAE) and MSE to assess the accuracy of the model's predictions. MAE measures the average absolute difference between predictions and actual values, providing a straightforward interpretation of the model's accuracy.

- **Epochs :** Epochs define the number of times the entire training dataset passes through the neural network. In our experiments, we typically train the model for a specified number of epochs to ensure it learns sufficiently from the data. Too few epochs can lead to underfitting, while too many can cause overfitting.

- **Batch Size :** Batch size is the number of training examples processed before the model's internal parameters are updated. Smaller batch sizes provide more accurate updates but are computationally expensive, while larger batch sizes are less accurate but more efficient. We balance these trade-offs to optimize training speed and model performance.

Early Stopping :

Early stopping is a regularization technique used to prevent overfitting. It monitors the model's performance on a validation set during training, and if the performance stops improving for a specified number of epochs (patience), it stops the training process early. This helps to ensure that the model generalizes well to unseen data and avoids overfitting. In our experiments, we use early stopping to halt training once the validation loss stops decreasing, indicating that the model has reached its optimal performance.

Callbacks :

Callbacks are functions executed during training at specific stages, such as at the end of an epoch. They are useful for implementing additional functionality such as learning rate schedules, logging, or early stopping. In our experiments we use a custom callback to :

- Monitor the Validation Loss: By setting monitor='val_loss', the callback tracks the validation loss throughout the training process

- Save the Best Weights: The 'save_best_only=True' parameter ensures that only the weights of the model that achieve the lowest validation loss are saved. This is an

indirect implementation of early stopping that prevents overfitting by storing the best-performing model rather than the final state of the training process.

This helps in retaining the optimal model configuration, which can be reloaded later for further evaluation or deployment. The callback ensures that our model generalizes well and prevents unnecessary storage of multiple model states.

```
ckpt_callback = tf.keras.callbacks.ModelCheckpoint(filepath=checkpoint_path,
                                                   monitor='val_loss',
                                                   save_best_only=True,
                                                   verbose=0,
                                                    )
```

Figure 4.2.9: the custom callback

Conclusion

Throughout the fourth chapter, We covered the development environment, model topologies, and basic configuration for our machine learning models. We introduced two key models: the XGBRegressor and the feedforward neural network (FNN). By means of this implementation, a strong framework for predictive modeling was developed. The next chapter will focus on experiments and results, where we will test, evaluate, and fine-tune these models to achieve optimal performance and bring out their full potential.

5

Experiments and Results

Introduction

In this chapter, we present the experimental setup, methodology, and results of the developed machine learning models .
The experiments were conducted to evaluate the performance of different models and to fine-tune them for optimal accuracy.

5.1 evaluation metrics :

To evaluate the performance of our models, we utilized the following metrics:

- R^2 Score: This measures the proportion of variance in the dependent variable that is predictable from the independent variables. An R^2 score closer to 1 indicates a better fit.
- Mean Cross-Validation Score: This metric represents the average performance score of the model across multiple folds in cross-validation.
- K-fold Cross-Validation (CV) Average Score: Similar to the mean cross-validation score, the K-fold CV average score is the average performance metric obtained by

dividing the dataset into K equal parts (folds). The model is trained on K-1 folds and tested on the remaining fold, and this process is repeated K times. The average of these scores provides a comprehensive measure of model performance.

- Mean Absolute Percentage Error (MAPE): This metric measures the accuracy of the model as a percentage.It calculates the average absolute percentage difference between the predicted values and the actual values, providing an intuitive measure of prediction accuracy.

- Mean Squared Error (MSE), defined as

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(\hat{y}_i - y_i)^2$$

  represents the average of the squares of the differences between predicted and ground truth values.

- Mean Absolute Error (MAE), defined as

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|\hat{y}_i - y_i|$$

  represents the average of the absolute differences between predicted and and ground truth values.

5.2 training the models

5.2.1 Training Extreme Gradient Boosting Regressor (XGBR)

Eeach simulation will be running on one of the following configurations :

- $configuration_1$: Dataset without any normalization.
- $configuration_2$: Dataset with normalization across all of the features.
- $configuration_3$: Dataset with only the "mileage" feature normalized since its the only one with high variance in its values.

5.2.1.1 simulation 1

This simulation runs on $configuration_1$ 5.2.1.

we can now evaluate our training by analyzing the calculated metrics :

- Training score (r2_score): 0.9833084204699034
- Mean cross-validation score: 0.90
- K-fold CV average score: 0.86
- MAPE: 9.76
- MAE: 5033.25

5.2.1.2 simulation 2

This simulation runs on $configuration_2$ 5.2.1.

- Training score (r2_score): 0.9818493743828094
- Mean cross-validation score: 0.83
- K-fold CV average score: 0.84
- MAPE: 9.80
- 5130.08

5.2.1.3 simulation 3

This simulation runs on $configuration_3$ 5.2.1.

- Training score (r2_score): 0.9838424858425898
- Mean cross-validation score: 0.89
- K-fold CV average score: 0.89
- MAPE: 9.44
- MAE : 4931.70

5.2.1.4 observation

let's take a look and compare the results :

model	r2_score	mean cross validation score	cross validation score	MAPE	MAE
experience 1	**0.983**	**0.90**	**0.86**	**9.76%**	**5033**
experience 2	**0.981**	**0.83**	**0.84**	**9.80%**	**5130**
experience 3	**0.984**	**0.89**	**0.89**	**9.44%**	**4931**

Table 5.2.1: observation 1

5.2.2 Testing XGBR

In this instance , We compute the loss over the predicted values for the test dataset and their ground truth counter parts.

5.2.2.1 simulation 4

This simulation evaluates on $configuration_1$ 5.2.1. The results are as follows:

- MAE: 7845.40
- MAPE: 15.99

here we can see the predictions scatterplot :

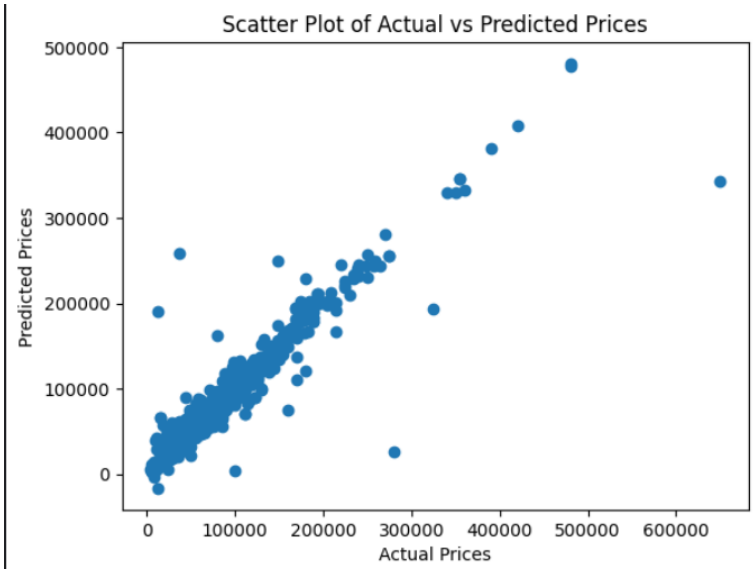


Figure 5.2.1: EXP 1 :Scatterplot

5.2.2.2 simulation 5

This simulation evaluates on $configuration_2$ 5.2.1. The results are as follows:

- MAE: 8278.70
- MAPE: 17.12

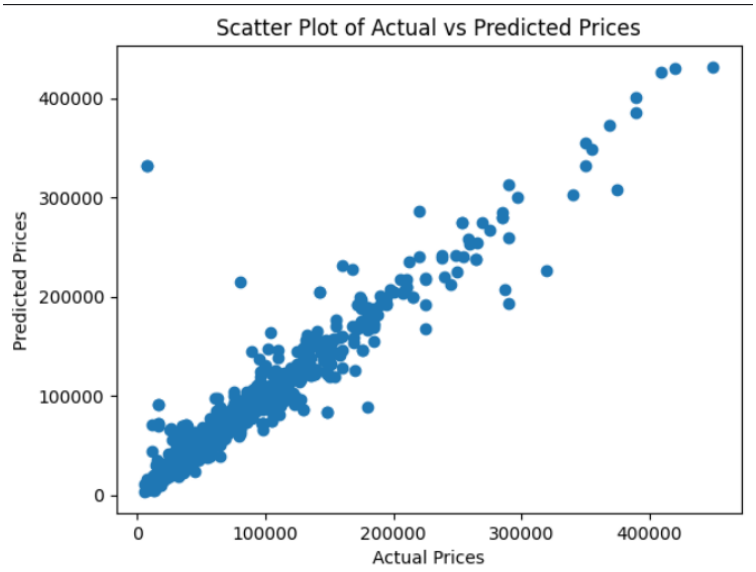


Figure 5.2.2: EXP 2 :Scatterplot

5.2.2.3 simulation 6

This simulation evaluates on $configuration_3$ 5.2.1. The results are as follows:

- MAE: 8536.96
- MAPE: 22.05

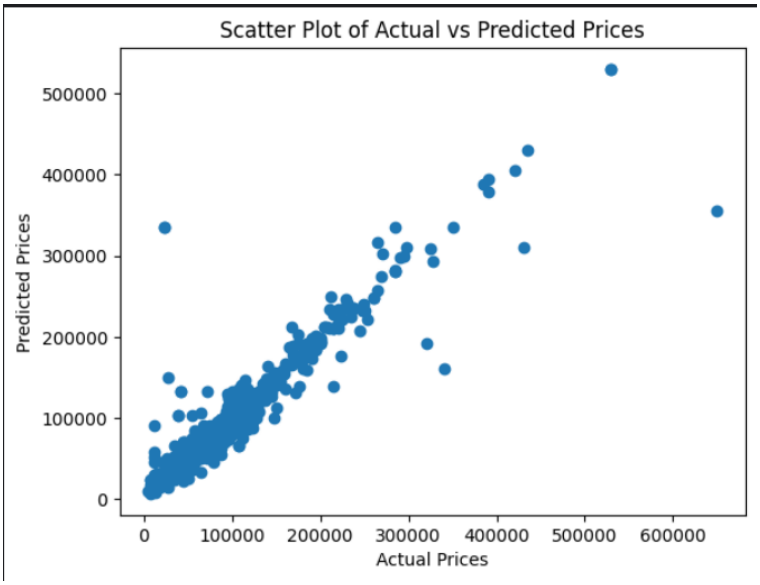


Figure 5.2.3: EXP 3 :Scatterplot

5.2.2.4 Discussion 2

model	MAE	MAPE
experiment 1	**7845**	**16%**
experiment 2	**8278**	**17.12%**
experiment 3	**8536**	**22%**

Table 5.2.2: 2nd observation

Upon evaluating the performance of the XGBR model, it is evident that the first experiment, which utilized unnormalized features, yielded the best results. This can be attributed to the inherent ability of the XGBoost algorithm to handle diverse feature scales and its robustness to the variations in the dataset. The model effectively captured the intricate patterns within the data without the need for feature normalization, which often simplifies the relationships among variables.

5.2.3 Training FNN

For training [12] our neural networks , we chose a well-defined structure where we fixed the values of some hyperparameters and experimented on the others . the fixed parameters were chosen because of their optimal use in regression problems , this includes

- Optimizer : adam
- loss : Mean Absolute Error (can be changed to MSE)
- number of epochs : 500 epochs plus the use of early stopping in our callback.

As for the rest of the parameters, we'll be experimenting in this section with different values to deduct the optimal shape of our model :
First of all , to choose the best model layout (number of hidden layers and neurons) , let's start by comparing our 3 FNN model structures in how well they perform in the training phase This will be evaluated based on spectating the learning curves as well as the error rates calculated by the "model.evaluate()" function provided by Keras . Throughout these experiments , we will be evaluating our neural network configurations based on the error rates produced in the training , validation and test subsets. To make decisions , the most important outputs to analyse are those of the test dataset , since it contains unseen data for the model meaning a simulation of its performance on the real-life data we'll use as input after implementing this project.

Note : these weren't the only experiments done throughout the project , we just selected a few examples to include in this documentation in order to justify our final model selection.

5.2.3.1 simulation 1

let's start with the first defined model layout, containing 2 hidden layers : we can here see the train/validation losses throughout the epochs iteration.

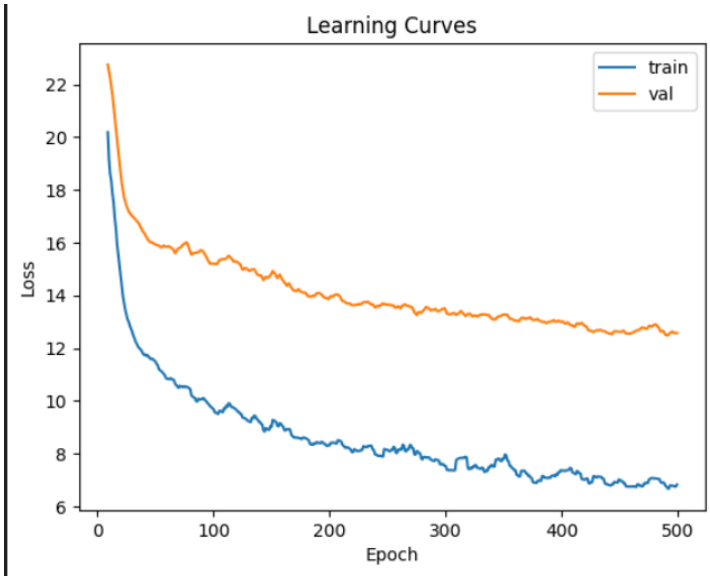


Figure 5.2.4: EXP 1 :Learning curves

evaluated dataset	Mean Absolute Error	Mean Squared Error	MAPE
Train set	**8.44**	**605**	**13.2%**
validation set	**12.4**	**799**	**21.4%**
Test set	**11.5**	**1064**	**19.3%**

Table 5.2.3: 1st simulation results

5.2.3.2 simulation 2

let's see how the second layout performs with three hidden layers

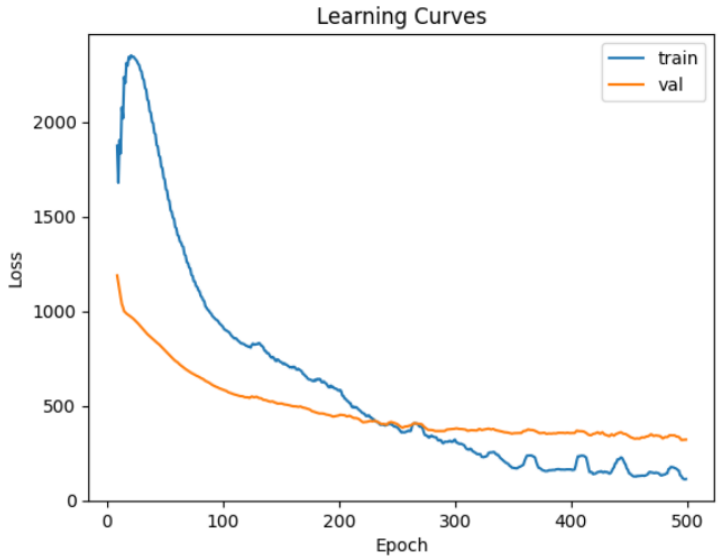


Figure 5.2.5: EXP 2 :Learning curves

evaluated dataset	Mean Absolute Error	Mean Squared Error	MAPE
Train set	**7.86**	**168.44**	**14.2%**
validation set	**11**	**328**	**20.8%**
Test set	**12.3**	**481**	**20.9%**

Table 5.2.4: 2nd simulation results

5.2.3.3 simulation 3

Finally , with five hidden layers , these are the results given by the third layout :

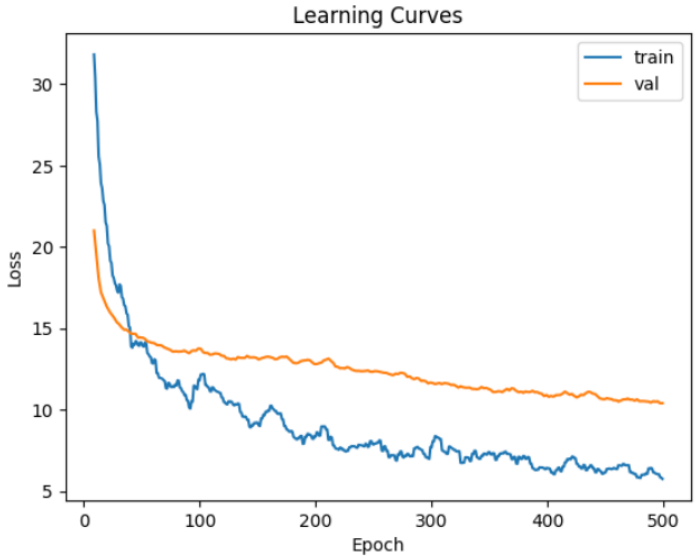


Figure 5.2.6: EXP 3 :Learning curves

model	Mean Absolute Error	Mean Squared Error	MAPE
Train set	**5.8**	**370**	**8.7%**
validation set	**10.25**	**580**	**15.5%**
Test set	**10.8**	**606**	**18%**

Table 5.2.5: 3rd simulation results

5.2.3.4 Discussion 1

By looking at the provided results , we can tell that the third experiment produced the best results after training , not only
Nevertheless , even the best performance from the previous experiments still lacks optimization . So , we'll stick with our five hidden layers model and let's experiment with adjusting other parameters starting with the loss determination

5.2.3.5 simulation 4

In this experience we set our loss function to "mse" instead of "mae"

evaluated dataset	Mean Absolute Error	Mean Squared Error	MAPE
Train set	**6.41**	**189**	**10.5%**
validation set	**10.7**	**442.9**	**18.8%**
Test set	**9.7**	**274.6**	**16.7%**

Table 5.2.6: 4th simulation results

5.2.3.6 discussion 2

Adjusting the loss to monitor the mean squared error increased the model's sensitivity to outliers, enhanced its learning dynamics and contributed to faster convergence. As a result, our model became more adept. we'll adopt the 'mse' as our loss function moving forward . To further optimize our model, one last important parameter to tweak is the learning rate. Fine-tuning the learning rate is crucial for achieving a balance between training efficiency and model accuracy.

5.2.3.7 simulation 5

learning rate : 0.01

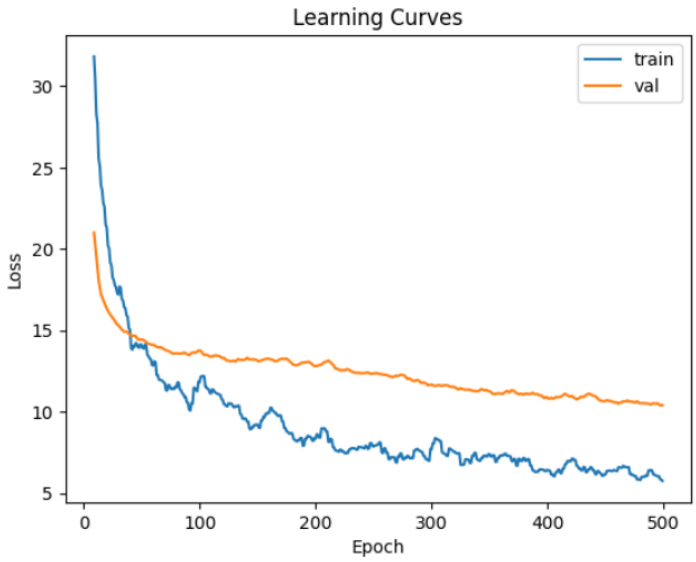


Figure 5.2.7: EXP 5 :Learning curves

evaluated dataset	Mean Absolute Error	Mean Squared Error	MAPE
train	**17.7**	**731**	**39.5%**
validation	**19.34**	**1300**	**38.4%**
Test	**20.5**	**1236.2**	**40.67%**

Table 5.2.7: 5th simulation results

5.2.3.8 simulation 6

learning rate : 0.0001

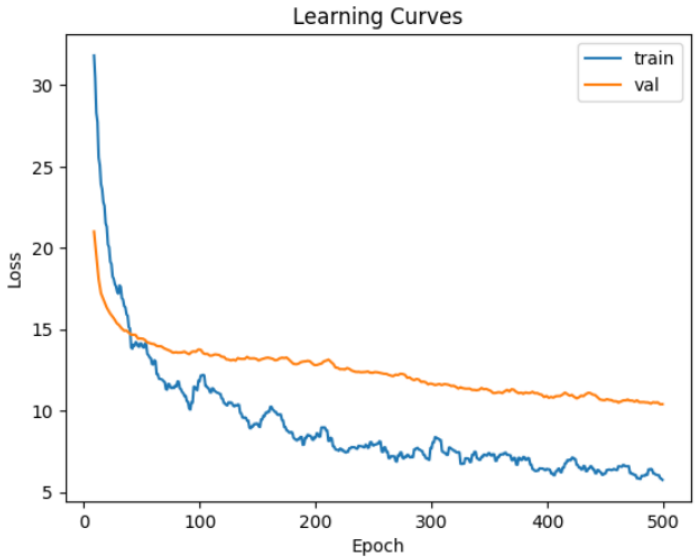


Figure 5.2.8: EXP 6 :Learning curves

evaluated dataset	Mean Absolute Error	Mean Squared Error	MAPE
train	**9.4**	**224**	**16.7%**
validation	**12.9**	**486**	**20.5%**
Test	**12.9**	**1111**	**22.3%**

Table 5.2.8: simulation 6 results

5.2.3.9 Discussion 3

Let's take a look at the different performance metrics seen on the testing dataset for our learning rate experiments :

learning rate	Mean Absolute Error	Mean Squared Error	MAPE
0.01	**20.5**	**1236.2**	**40.67%**
0.001	**9.7**	**274.6**	**16.7%**
0.0001	**12.9**	**1111**	**22.3%**

Table 5.2.9: 3rd discussion results

The learning rate with the lowest MAE of 9.7 is 0.001, which is coincidentally the learning rate used in the previous experiments.

5.3 Overall results :

The comparison between the neural network and the XGBoost model reveals that the XGBoost model is more accurate in predicting car prices. The neural network's average prediction error is 9700 DT with a Mean Absolute Percentage Error (MAPE) of 16.7%, indicating a 16.7% deviation from actual prices. This suggests the model captures complex patterns but requires further tuning for better accuracy.
In contrast, the XGBoost model has a lower average prediction error of 7845 DT and a slightly better MAPE of 16%, making it more reliable for predicting car market values in the given dataset. This demonstrates that the XGBoost Regressor provides more accurate and dependable predictions compared to the neural network model.

Real-Life Implications :

Translating these findings to the real-world context of assessing car market values, the models show considerable promise. We can safely say that our system produces accurate estimations on the cars fed to it in the training phase .

Future Directions :

The main performance enhancer for our system would most certainly be amassing a huge volume of car listings , by widening our data extraction sources and maintaining the quality of the data , the system efficiency goes up , generalizing its knowledge across a wider array of vehicle types and market conditions.
This also includes collecting data on newly released car models, changes in market trends, and all other relevant factors that might influence car valuations.

Regularly training the model on larger volumes of timely, high-quality data will not only enhance its predictive accuracy but also help in maintaining its robustness over time. This ongoing process of data collection and model retraining is essential for keeping the model's knowledge base updated and for refining its performance.

In conclusion, while our current system provides a solid foundation for car market value predictions, its long-term success hinges on a sustained effort to keep the model updated with the latest data. By doing so, we can ensure that it remains a valuable tool for the insurance industry, capable of delivering precise and reliable valuations for a wide range of vehicles.

Conclusions and prespectives

throughout this research study , we came to understand the reliability of the car insurance domain on the accurate assessment of automobile market values as it is the gateway for successfully executing various procedures in the case of car collisions , one of the main study cases in the insurance world. Current methods of car valuation are often time-consuming and lack precision, posing significant challenges for both insurers and insured parties. Throughout this internship , we aimed to address these challenges by developing a machine learning system to predict car market values based on various descriptive features.
During the project, we encountered several difficulties. One of the main challenges was dealing with the noise and missing values in the dataset, which affected model performance. To cope with this, we employed data preprocessing techniques.
Additionally, finding the optimal hyperparameters for our models required extensive experimentation and fine-tuning, but we managed to achieve significant improvements in performance.
Looking forward, there are several ways to enhance our work. Further hyperparameter tuning using grid search or randomized search can explore a wider range of configurations. Incorporating ensemble methods can leverage the strengths of multiple models, and advanced regularization techniques can improve model generalization. As for the data , the system will benefit exponentially from augmenting both the quality and the volume of the second hand current car information .
Furthermore , the next step towards automating the reporting process for car collision cases would implementing optical character recognition for analysing accident claims and extracting the vehicle's damage information as well as its history .
From a personal perspective, this project has been highly educational and enriching. I gained practical experience in the fields of data science and Machine learning in which i hope to pursue my future education.
I also learned the importance of perseverance and adaptability when facing challenges in a project. This internship has not only enhanced my technical skills but also provided

valuable insights into the insurance industry and the potential of machine learning to revolutionize traditional procedures.

In conclusion, while this project has laid a strong foundation for automating car market value assessments in the Tunisian insurance sector, there is still much room for improvement and expansion. As the industry continues to evolve, integrating more advanced machine learning techniques and continuously refining our models will be crucial in maintaining accuracy and efficiency. This project marks a significant step towards modernizing the insurance sector by providing more precise and timely valuations.

Bibliography

[1] https://medium.com/@b.terryjack/introduction-to-deep-learning-feed-forward-neural-networks-ffnns-a-k-a-c688d83a3(
2024. Accessed: 2024-05-01.

[2] https://www.investopedia.com/terms/i/insurtech.asp#:~:text=Insurtech%20refers%20to%20the%20use,the%20current%20insurance%20industry%20model., 2024. Accessed: 2024-04-13.

[3] https://drive.google.com/file/d/1LRHGe3g9n4TOiutI9sGUUu7zwJFOkxaS/view, 2024. Accessed: 2024-05-01.

[4] https://www.researchgate.net/publication/317177787_Web_Scraping, 2024. Accessed: 2024-05-01.

[5] https://code.visualstudio.com/docs, 2024. Accessed: 2024-04-13.

[6] https://www.pecan.ai/blog/data-preparation-for-machine-learning/, 2024. Accessed: 2024-05-01.

[7] https://pypi.org/project/lazypredict/, 2024. Accessed: 2024-05-01.

[8] https://xgboost.readthedocs.io/en/stable/python/python_api.html, 2024. Accessed: 2024-05-01.

[9] https://en.wikipedia.org/wiki/TensorFlow, 2024. Accessed: 2024-05-01.

[10] https://keras.io/getting$_s$*tarted*/.

[11] https://keras.io/api/, 2024. Accessed: 2024-05-01.

[12] https://machinelearningmastery.com/regression-tutorial-keras-deep-learning-library-python/, 2024. Accessed: 2024-04-13.