**VIETNAM NATIONAL UNIVERSITY HO CHI MINH CITY  
UNIVERSITY OF INFORMATION TECHNOLOGY  
FACULTY OF INFORMATION SYSTEMS**

**Logo

Description automatically generated**

**FINAL PROJECT**

**ROSSMANN STORE SALES: A DATA-DRIVEN APPROACH**

**INSTRUCTOR: TS. Nguyen Quoc Khanh**

**CLASS: CS331.N21.KHCL**

**GROUP IMPLEMENTATION**

|  |  |  |
| --- | --- | --- |
| **No** | **Full Name** | **Student Code** |
| **1** | **Nguyen Van A** | **2152xxxx** |
| **2** | **Nguyen Van B** | **2152xxxx** |
| **3** | **Nguyen Van C** | **2152xxxx** |

**Ho Chi Minh, 06/2023**

TABLE OF CONTENTS

[TABLE OF CONTENTS 2](#_Toc137864626)

[INTRODUCTION 3](#_Toc137864627)

[CHAPTER 1: IDENTIFICATION OF PROBLEMS 4](#_Toc137864628)

[1.1. Reason for choosing the topic 4](#_Toc137864629)

[1.2. About the dataset 5](#_Toc137864630)

[1.3. Tools and Source 6](#_Toc137864631)

[CHAPTER 2: DATA PREPROCESSING PROCESS 7](#_Toc137864632)

[2.1. Data Cleaning 7](#_Toc137864633)

[2.1.1. Dataset Description 7](#_Toc137864634)

[2.1.2. Data fields 8](#_Toc137864635)

[2.1.3. Merge data train.csv to store.csv by left join 9](#_Toc137864636)

[2.1.4. Narrowing dataset 9](#_Toc137864637)

[2.1.5. Adding DayofWeekName for better graph-labeling 9](#_Toc137864638)

[2.1.6. Adding MonthName 9](#_Toc137864639)

[2.1.7. Break up dataset into 4 quarters for visualization 9](#_Toc137864640)

[2.1.8. Ordering Week Day & Month Labels 10](#_Toc137864641)

[2.2. Exploratory Data Analysis 10](#_Toc137864642)

[2.2.1. StoreType vs Sales 11](#_Toc137864643)

[2.2.2. WeekDay vs Sales 14](#_Toc137864644)

[2.2.3. MonthName vs Sales 17](#_Toc137864645)

[2.2.4. StorType vs Customers 18](#_Toc137864646)

[CHAPTER 3: MODELING 19](#_Toc137864647)

[3.1. XGB Regressor 22](#_Toc137864648)

[3.2. Deep Learning (Keras) 24](#_Toc137864649)

[3.2.1. Deep Neural Network Architecture (DNN) 24](#_Toc137864650)

[3.2.2. DNN and Regressor 27](#_Toc137864651)

[3.2.3. Designing the DNN 29](#_Toc137864652)

[3.2.4. Configure the model 30](#_Toc137864653)

[3.2.5. Train the model 31](#_Toc137864654)

[3.2.6. Testing the Model Performance 32](#_Toc137864655)

[3.2.7. Improving the Model 32](#_Toc137864656)

[Improving the Model 35](#_Toc137864657)

[3.2.8. Plotting the Loss Metric Across Epochs 36](#_Toc137864658)

[3.2.9. Testing the Model Manually 37](#_Toc137864659)

[3.2.10. Calculate and print the MSE and MAE 38](#_Toc137864660)

[3.2.11. Conclusion 38](#_Toc137864661)

[CHAPTER 4: DEPLOYMENT 39](#_Toc137864662)

[4.1. Streamlit 39](#_Toc137864663)

[4.2. Graphical User Interface (GUI) 40](#_Toc137864664)

[Conclusion 43](#_Toc137864665)

[CHAPTER 4: REFERENCES 44](#_Toc137864666)

# INTRODUCTION

The retail industry plays a significant role in the global economy, and understanding sales patterns and trends is crucial for businesses to thrive and make informed decisions. With the increasing availability of data and advancements in machine learning techniques, there is an opportunity to leverage these tools to gain valuable insights and improve sales predictions.

In this project, we focus on analyzing and predicting sales using the Rossman dataset. The primary objective is to develop models that can accurately forecast sales based on various factors such as store attributes, time-related variables, and promotional activities. By accurately predicting sales, retailers can optimize their inventory management, staffing, and marketing strategies, ultimately leading to improved profitability and customer satisfaction.

The motivation behind this study lies in the potential benefits that accurate sales predictions can offer to the retail industry. With the ability to anticipate consumer demand, retailers can effectively allocate resources, plan promotions, and optimize their overall operations. This can lead to reduced costs, minimized stockouts, and increased customer loyalty.

To accomplish our goals, we will follow a structured approach. We will begin by exploring the dataset, gaining an understanding of its contents and characteristics. Next, we will preprocess the data, addressing any missing values, outliers, or inconsistencies that may affect the quality of our models. We will then perform exploratory data analysis to uncover insights and relationships between variables.

For modeling, we will employ both traditional machine learning algorithms, such as XGB Regressor, and deep learning techniques using the Keras framework. These models will be trained and evaluated based on their predictive performance, with metrics such as mean squared error (MSE) and mean absolute error (MAE) guiding our assessment.

To make our models accessible and user-friendly, we will deploy them using the Streamlit framework. This will enable stakeholders to interact with the models through a graphical user interface, providing them with visualizations, sales predictions, and actionable insights.

Overall, this project aims to demonstrate the power of machine learning in the retail industry and showcase how accurate sales predictions can drive informed decision-making and lead to improved business outcomes. By leveraging the Rossman dataset and implementing advanced modeling techniques, we can provide valuable tools and insights for retailers to navigate the complexities of the market and optimize their operations.

# CHAPTER 1: IDENTIFICATION OF PROBLEMS

## Reason for choosing the topic

The purpose of this project is to develop a predictive model that can forecast sales for Rossmann stores using store, promotion, and competitor data. By leveraging the available dataset, this study aims to provide insights and support decision-making processes for optimizing sales strategies in the retail industry.

**Rationale**: The choice of this topic stems from several compelling reasons. Firstly, accurate sales forecasting plays a crucial role in strategic planning and resource allocation for businesses. By understanding future sales trends, companies like Rossmann can make informed decisions regarding inventory management, staffing, and promotional activities.

Secondly, the availability of rich and diverse data in the Rossmann Store Sales dataset presents a valuable opportunity for analysis and prediction. With information about individual store characteristics, promotional campaigns, and competitor factors, we can explore the potential relationships and patterns that contribute to store sales.

Additionally, the retail industry is highly competitive and dynamic, making it essential for businesses to continuously improve their forecasting capabilities. By developing an accurate sales prediction model, Rossmann can gain a competitive edge by optimizing operations, maximizing profits, and enhancing customer satisfaction.

Overall, this project aims to harness the power of data analysis and predictive modeling techniques to provide valuable insights and practical solutions for optimizing sales strategies in the retail industry, specifically for Rossmann stores.

### About the dataset

**Rossmann** operates over 3,000 drug stores in 7 European countries. Currently, Rossmann store managers are tasked with predicting their daily sales for up to six weeks in advance. Store sales are influenced by many factors, including promotions, competition, school and state holidays, seasonality, and locality. With thousands of individual managers predicting sales based on their unique circumstances, the accuracy of results can be quite varied.

**The Rossmann Store Sales dataset** is a comprehensive collection of data that provides valuable insights into the sales performance of Rossmann stores. It includes a wide range of variables related to stores, promotions, and competitors, allowing for a thorough analysis of factors influencing sales.

The dataset encompasses historical sales data of Rossmann stores across multiple regions and time periods. It provides information about each store's attributes, such as store ID, store type, assortment type, competition distance, and competition open since.

In addition to store-specific details, the dataset also includes promotional information. This includes variables like the presence of promotions, the type of promotion (e.g., markdowns, holidays, school holidays), and the duration of promotions. These features enable the exploration of the impact of various promotional strategies on store sales.

Furthermore, the dataset incorporates competitor data, which plays a significant role in understanding the market dynamics. Variables such as the number of competing stores in the area and the distance to the nearest competitor store offer insights into the competitive landscape and its influence on sales.

The dataset covers a substantial timeframe, allowing for the analysis of trends and seasonality in sales patterns. This temporal dimension provides the opportunity to develop time-series forecasting models that can capture recurring patterns and make accurate predictions.

By utilizing the Rossmann Store Sales dataset, researchers and data analysts can delve into the intricacies of store sales dynamics and uncover key drivers of success in the retail industry. The richness and depth of the dataset make it a valuable resource for developing predictive models and deriving actionable insights for optimizing sales strategies.

## Tools and Source

* Programming Languages: Python
* IDE: Visual Studio Code 2022, Jupyter Notebook
* Data Visualization: Numpy, Matplotlib
* Deployment: Streamlit
* Source code: <https://github.com/nqkhanh2002/Sales-Predict-Streamlit>
* Usage
  + Clone the Repository: Start by cloning the project repository from GitHub. Open your terminal or command prompt and run the following command:



* + Navigate to the Project Directory: Change your working directory to the project folder using the following command:



* + Navigate to the Project Directory: Change your working directory to the project folder using the following command:



* Navigate to the Project Directory: Change your working directory to the project folder using the following command: **cd streamlit\_app**
* Prepare the Dataset: Place your Rossmann Store Sales dataset file (e.g., "train.csv") inside the "data" folder of the project.
* Access the App: After the Streamlit app is successfully launched, you should see a URL displayed in the terminal (e.g., http://localhost:8501). Open a web browser and enter the provided URL to access the app.
* Interact with the App: The Streamlit app should now be running in your browser. Use the interactive controls and features provided by the app to explore the dataset, perform data analysis, and make sales predictions based on the provided features.

# CHAPTER 2: DATA PREPROCESSING PROCESS

## 2.1. Data Cleaning

### 2.1.1. Dataset Description

You are provided with historical sales data for 1,115 Rossmann stores. The task is to forecast the "Sales" column for the test set. Note that some stores in the dataset were temporarily closed for refurbishment.

**Files**

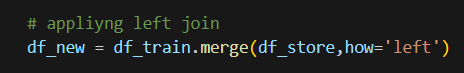
* **train.csv** - historical data including Sales
* **test.csv** - historical data excluding Sales
* **store.csv** - supplemental information about the stores

### 2.1.2. Data fields

Most of the fields are self-explanatory. The following are descriptions for those that aren't.

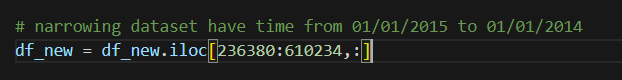
* **Id** - an Id that represents a (Store, Date) duple within the test set
* **Store** - a unique Id for each store
* **Sales** - the turnover for any given day (this is what you are predicting)
* **Customers** - the number of customers on a given day
* **Open** - an indicator for whether the store was open: 0 = closed, 1 = open
* **StateHoliday** - indicates a state holiday. Normally all stores, with few exceptions, are closed on state holidays. Note that all schools are closed on public holidays and weekends. a = public holiday, b = Easter holiday, c = Christmas, 0 = None
* **SchoolHoliday** - indicates if the (Store, Date) was affected by the closure of public schools
* **StoreType** - differentiates between 4 different store models: a, b, c, d
* **Assortment** - describes an assortment level: a = basic, b = extra, c = extended
* **CompetitionDistance** - distance in meters to the nearest competitor store
* **CompetitionOpenSince[Month/Year]** - gives the approximate year and month of the time the nearest competitor was opened
* **Promo** - indicates whether a store is running a promo on that day
* **Promo2** - Promo2 is a continuing and consecutive promotion for some stores: 0 = store is not participating, 1 = store is participating
* **Promo2Since[Year/Week]** - describes the year and calendar week when the store started participating in Promo2
* **PromoInterval** - describes the consecutive intervals Promo2 is started, naming the months the promotion is started anew. E.g. "Feb,May,Aug,Nov" means each round starts in February, May, August, November of any given year for that store

### 2.1.3. Merge data train.csv to store.csv by left join

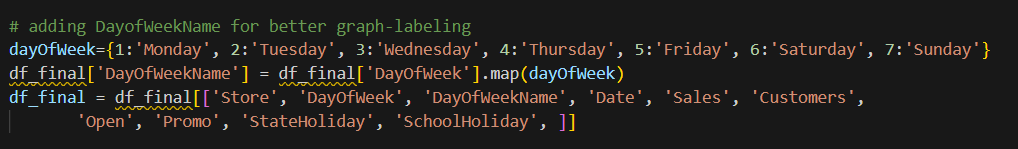
****

### 2.1.4. Narrowing dataset

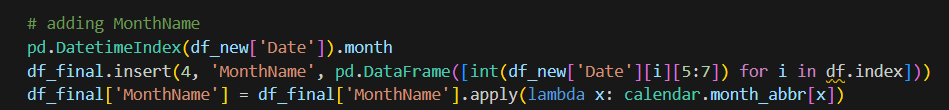
**-We will use a subset of dataset by narrowing dataset have time from 01/01/2015 to 01/01/2014**



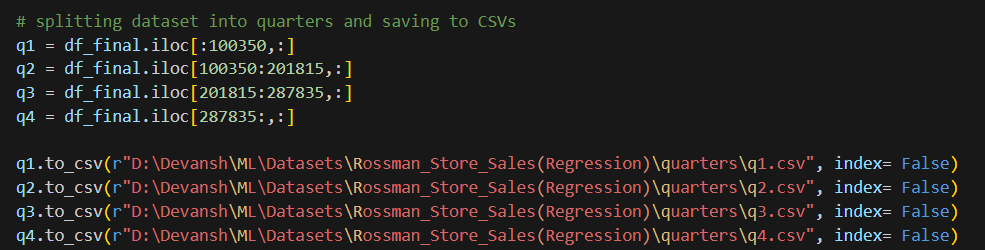
### 2.1.5. Adding DayofWeekName for better graph-labeling



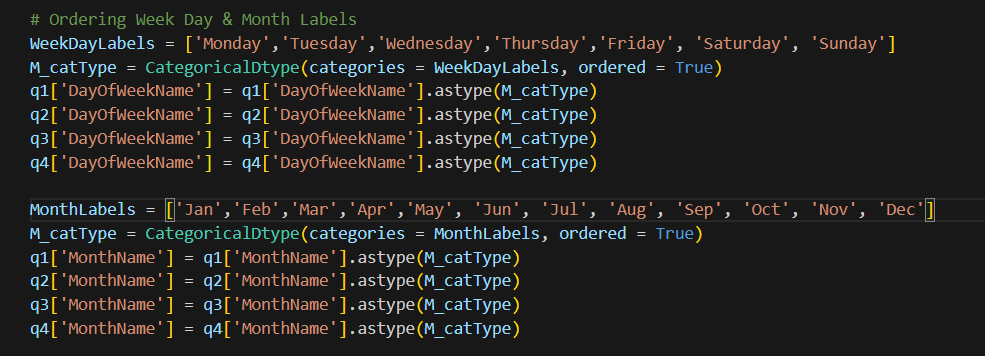
### 2.1.6. Adding MonthName



### 2.1.7. Break up dataset into 4 quarters for visualization



### 2.1.8. Ordering Week Day & Month Labels



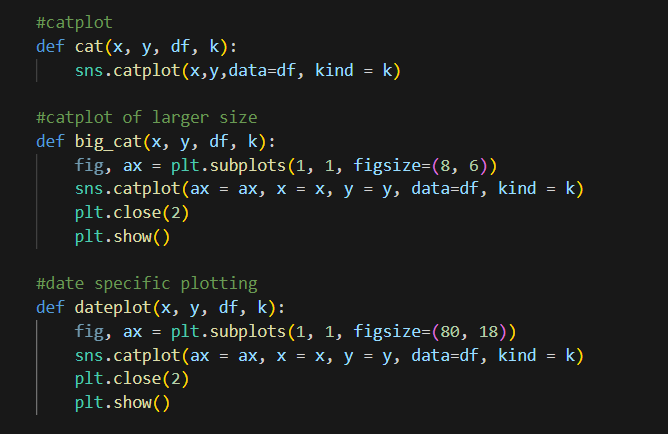
Finally, we have a dataset include 372944 rows × 20 columns



## 2.2. Exploratory Data Analysis

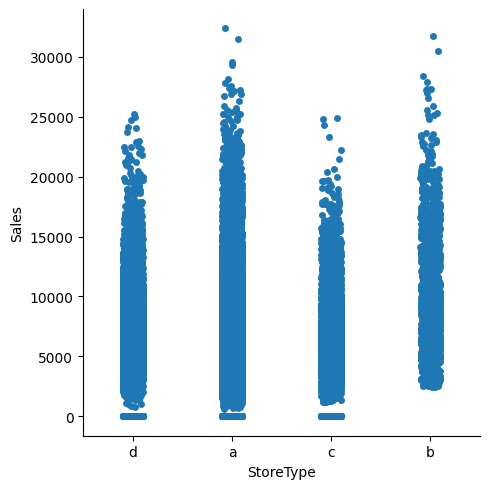
Now we will build some plotting function for visualize dataset

These function provides access to several axes-level functions that show the relationship between a numerical and one or more categorical variables using one of several visual representations.

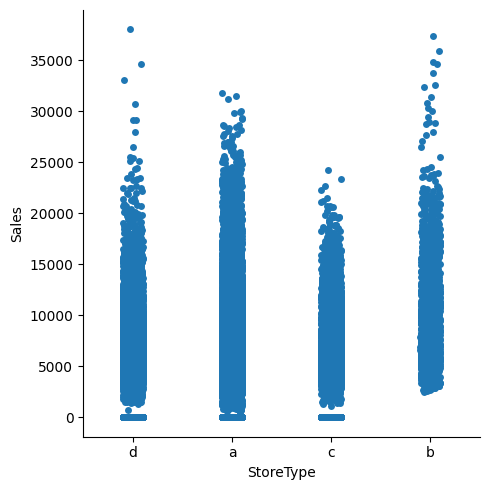


### StoreType vs Sales

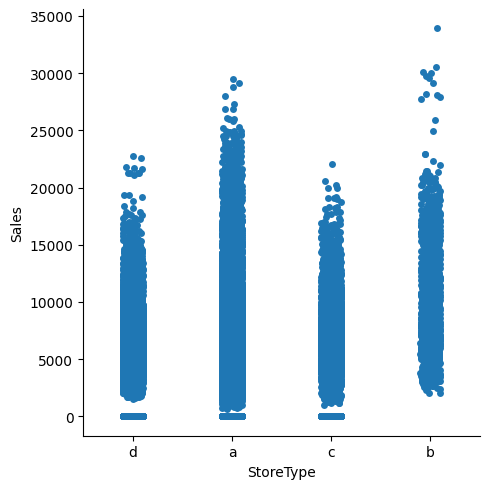
Quarter 1, 2, 3 and 4



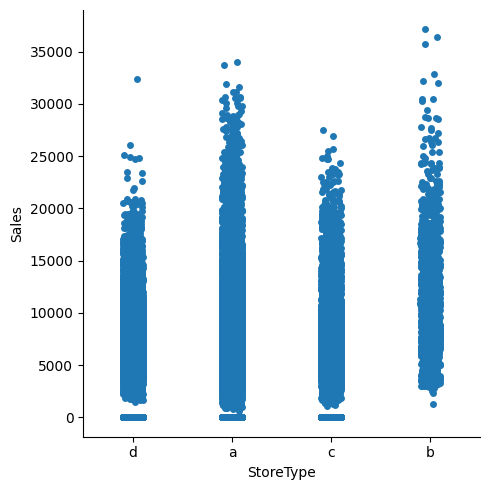
Quarter 1



Quarter 2



Quarter 3

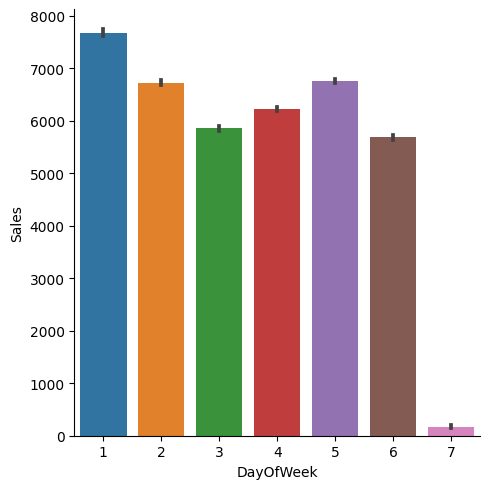


Quarter 4

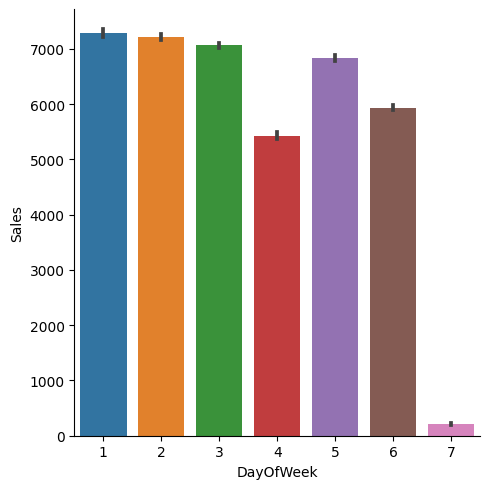
**We can't see a big difference between sales and StorType(a,b,c,d) in 4 quarters.**

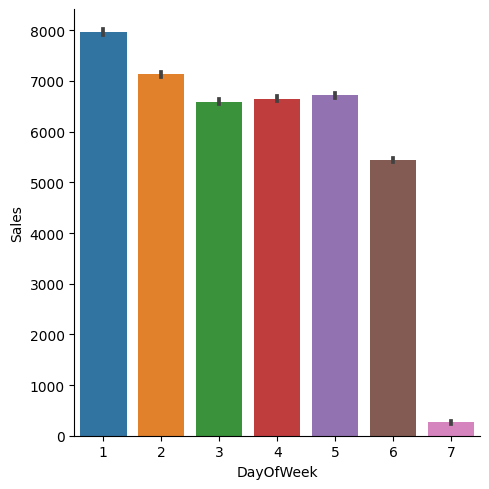
### WeekDay vs Sales

We will use bar chart instead chart in this section

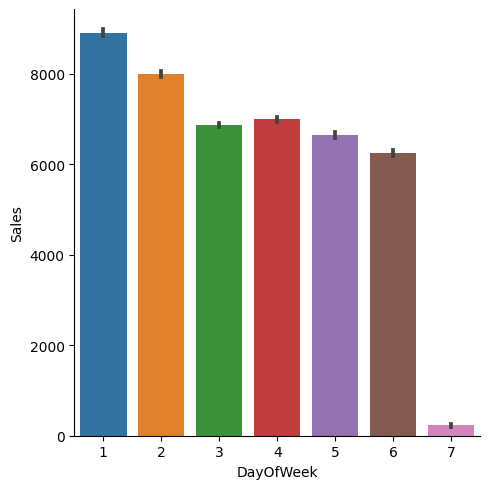


Quarter 1





Quarter 3



According to the observation, sales tend to be highest at the beginning of the week. This suggests that customers are more likely to make purchases early in the week, potentially due to factors such as the start of the workweek or specific shopping patterns.

However, as the week progresses, the sales figures begin to decline slightly in the middle of the week. This could be attributed to factors such as reduced customer footfall or a shift in consumer behavior during the mid-week period.

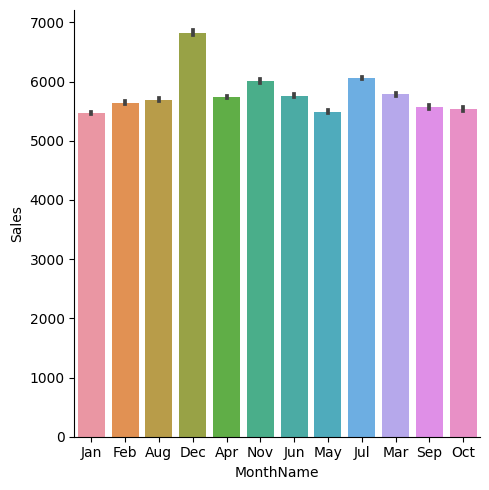
The analysis also mentions that there is either no significant increase or a slight decrease in sales towards the end of the week. This indicates that the sales figures remain relatively stable or show a minor dip during this time.

Furthermore, there is a sharp drop in sales on Sundays, with the note that this drop is almost negligible. This implies that Sunday is not a prominent sales day in the dataset, and the amount of sales generated on Sundays is considerably lower compared to other days of the week.

Overall, this analysis highlights the sales trends based on the day of the week in the Rossman dataset, with the highest sales occurring at the beginning of the week and a decline or stability in sales towards the end of the week, with Sundays experiencing a sharp drop in sales.

### MonthName vs Sales

We will observe all the counties



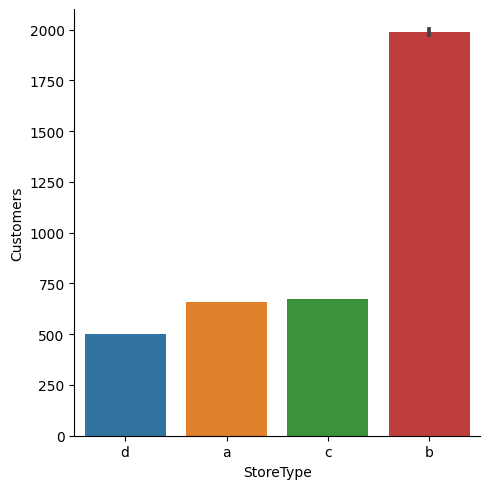
This observation indicates that December stands out as a month with notably higher sales compared to other months. There could be several factors contributing to this increase in sales during December:

1. Holiday Season: December includes major holidays such as Christmas and New Year's Eve, which are traditionally associated with increased consumer spending. Customers may be more inclined to purchase products and gifts during this festive period, leading to higher sales.
2. Seasonal Promotions: Many retailers offer special promotions and discounts during the holiday season to attract customers. These incentives can stimulate higher sales as customers take advantage of the deals available.
3. Gift Shopping: December is a popular month for gift shopping, as individuals purchase presents for their loved ones. This increased demand for gifts and the associated sales can contribute to the higher sales figures observed during this month.

On the other hand, the analysis indicates that the sales figures for the other months are relatively consistent, remaining close to 6000. This suggests that there might be a baseline level of sales throughout the year, with minor variations. Factors such as regular customer demand, ongoing promotions, and the nature of the retail business itself could contribute to this stability in sales for the rest of the months.

### StorType vs Customers

The analysis compares the feature pair of "StoreType" (a, b, c, d) with the number of customers in the Rossman dataset. According to the observation, the store type represented by the letter "b" has the highest number of customers, with a value of nearly 2000.



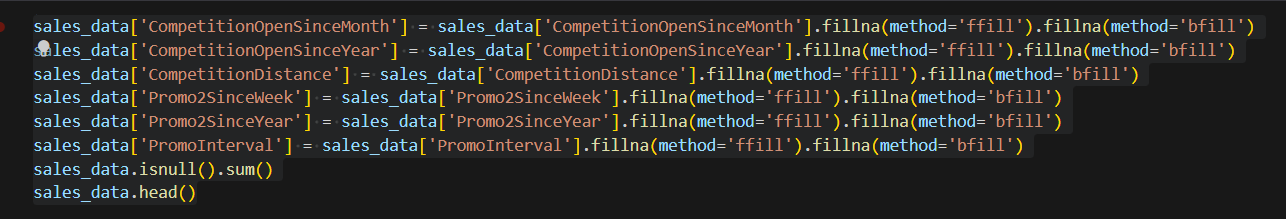
Furthermore, the difference in the number of customers between store types "a," "c," and "d" is relatively small, with a margin of approximately 750 clients. This suggests that these three store types have a relatively similar number of customers, with a slight variation among them.

Based on this analysis, we can infer that store type "b" attracts the largest customer base, resulting in a significantly higher number of customers compared to the other store types. The reasons for this difference in customer count can vary and may be influenced by factors such as store location, size, product offerings, marketing strategies, and customer preferences.

It's important to note that further analysis and investigation into the specific characteristics and attributes of each store type can provide more insights into why store type "b" stands out in terms of customer numbers compared to the other store types.

# CHAPTER 3: MODELING

Before modeling, we need **Data preparation**, also known as data preprocessing, is an essential step that comes after EDA (Exploratory Data Analysis) and before feeding the data into a model. It involves transforming and cleaning the data to make it suitable for analysis and model training.

**Filling null values** by using forward-fill (ffill) and backward-fill (bfill) methods to fill missing values in the selected columns. 

Now we have a non-null data

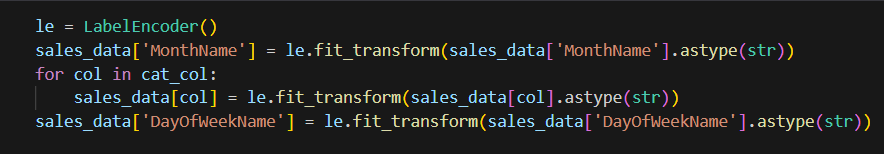


**Transforming categorical data**

Transforming categorical data involves converting categorical variables into a numerical representation that can be used by machine learning algorithms. This process is necessary because many machine learning algorithms require numerical inputs.

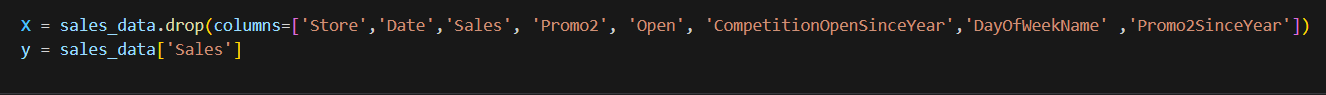
We use Label Encoding technique

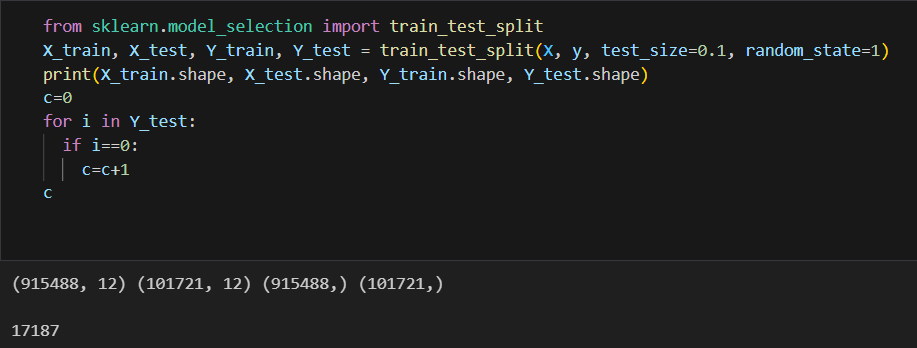
**Label Encoding:** In label encoding, each category is assigned a unique numerical value. For example, if we have a categorical variable "Color" with categories ["Red", "Green", "Blue"], label encoding may assign the values [0, 1, 2] to these categories, respectively. However, it's important to note that label encoding may introduce an unintended ordinal relationship between categories, which may not be appropriate for all situations.



**Splitting the dataset**

Splitting the dataset into training and testing subsets is a crucial step in machine learning model development. It allows us to evaluate the model's performance on unseen data and assess its generalization ability. The typical approach involves randomly dividing the dataset into two parts: the training set and the test set.





**Test\_size** is the proportion of the dataset that should be allocated to the test set. In this case, it is set to 0.1, meaning 10% of the data will be used for testing, while the remaining 90% will be used for training.

## XGB Regressor

XGBRegressor is a class in the XGBoost library that implements the gradient boosting regression algorithm. XGBoost stands for "Extreme Gradient Boosting," and it is a popular machine learning algorithm known for its high performance and accuracy in regression tasks. XGBRegressor is specifically designed for regression problems.

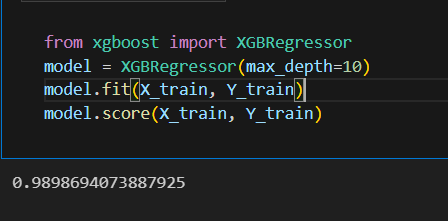


* **Gradient Boosting Algorithm:** XGBRegressor is based on the gradient boosting algorithm, which combines multiple weak prediction models (typically decision trees) to create a strong ensemble model. It iteratively builds new models that predict the residuals (the difference between the actual and predicted values) of the previous models, thereby reducing the overall error.

**XGBoost Features:** XGBRegressor incorporates several key features that contribute to its effectiveness:

* Regularization: XGBoost includes regularization techniques such as L1 regularization (Lasso) and L2 regularization (Ridge), which help prevent overfitting and improve model generalization.
* Tree Pruning: XGBoost applies tree pruning techniques to remove unnecessary branches and reduce model complexity, enhancing its efficiency and reducing memory usage.
* Column Subsampling: XGBoost enables subsampling of columns (features) during each iteration, which helps handle high-dimensional datasets and reduces the risk of overfitting.
* Handling Missing Values: XGBoost provides built-in mechanisms for handling missing values within the dataset, eliminating the need for imputation.

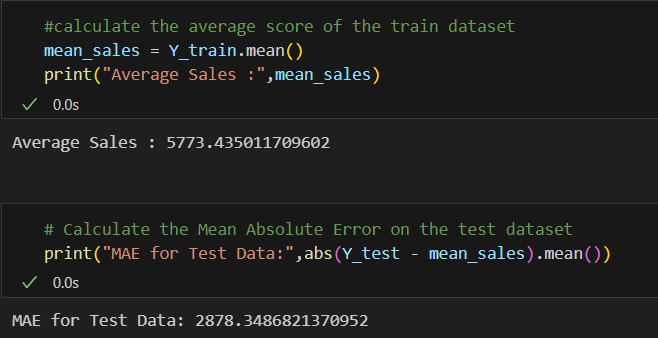
**Implement with score training ~ 0.98 %**



**Calculate the Mean Absolute Error on the test dataset**

MAE stands for Mean Absolute Error. It is a common evaluation metric used in regression tasks to measure the average magnitude of errors between the predicted values and the true values.

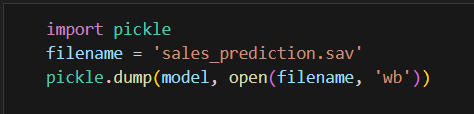
**Limitations**: MAE does not provide information about the direction of errors. It treats all errors equally, which may not be appropriate in some scenarios where certain errors are more critical than others. Additionally, MAE does not account for the scale or magnitude of the target variable, making it difficult to interpret the error in the context of the problem domain.



The MAE represents the average magnitude of the differences between the predicted sales values and the actual sales values in the test set. In this case, the MAE of 2878.3486821370952 indicates that, on average, the predictions deviate from the actual sales values by approximately 2878.35 units.

**And this error is still quite large, next we will try another way for this problem using Deep Learning.**

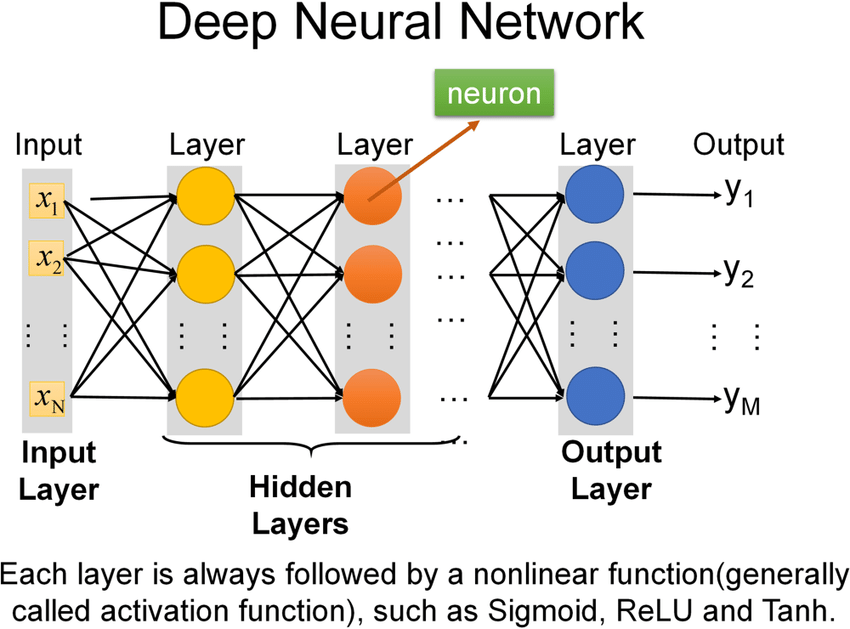
**Saving the pretrained model for re-use and deployment**



## Deep Learning (Keras)

### Deep Neural Network Architecture (DNN)

A Deep Neural Network (DNN) architecture is a type of artificial neural network that consists of multiple hidden layers between the input and output layers. It is designed to learn complex patterns and representations in data by leveraging the hierarchical structure of these layers.



**Detailed explanation of the DNN architecture:**

1. Input Layer: The input layer is the initial layer of the DNN and represents the input features or data. Each neuron in the input layer corresponds to a feature or attribute of the input data.
2. Hidden Layers: DNNs typically have multiple hidden layers sandwiched between the input and output layers. Each hidden layer consists of a set of neurons, also known as units or nodes. The number of hidden layers and the number of neurons in each layer can vary depending on the complexity of the problem and the available resources.
3. Activation Functions: Activation functions are applied to the output of each neuron in the hidden layers to introduce non-linearity and enable the network to learn complex relationships. Popular activation functions include the Rectified Linear Unit (ReLU), sigmoid, and hyperbolic tangent (tanh).
4. Weighted Connections: Each neuron in a layer is connected to every neuron in the subsequent layer. These connections have associated weights, which are adjusted during the training process to optimize the network's performance. The weights determine the strength and impact of the input signals on the neuron's output.
5. Forward Propagation: In the forward propagation phase, the input data is fed into the input layer, and the weighted sum of the inputs is computed for each neuron in the hidden layers. The output of each neuron is then passed through the activation function to produce the activation value or output of the neuron.
6. Backpropagation: Backpropagation is used to train the DNN by iteratively adjusting the weights based on the calculated errors between the predicted output and the actual output. The error is propagated backward through the network, and the weights are updated using optimization algorithms such as gradient descent or its variants.
7. Output Layer: The output layer of the DNN produces the final predictions or outputs. The number of neurons in the output layer depends on the nature of the problem. For example, for a binary classification task, there may be one neuron representing the probability of the positive class, while for a multi-class classification task, there would be multiple neurons representing the probabilities of each class.
8. Loss Function: The loss function quantifies the discrepancy between the predicted output and the true output. It is used to measure the performance of the network and guide the weight updates during training. Common loss functions for different tasks include mean squared error (MSE) for regression, binary cross-entropy for binary classification, and categorical cross-entropy for multi-class classification.
9. Training and Optimization: DNNs are trained using a large labeled dataset through an iterative process. During training, the network learns to optimize its weights to minimize the loss function and improve its predictive performance. Optimization techniques, such as stochastic gradient descent (SGD) or more advanced algorithms like Adam or RMSprop, are commonly used to update the weights efficiently.
10. Model Evaluation: Once the DNN is trained, it can be evaluated on a separate test dataset to assess its generalization performance. Evaluation metrics such as accuracy, precision, recall, and F1 score can be used to measure the model's performance on different tasks.

**DNN architectures** have demonstrated excellent performance in various domains, including image and speech recognition, natural language processing, and recommendation systems. However, designing an effective DNN architecture requires careful consideration of factors such as the dataset size, complexity of the problem, available computational resources, and domain knowledge.

### DNN and Regressor

Using a Deep Neural Network (DNN) as a regressor offers several advantages over traditional regression models. Here are some key points to consider:

1. Non-linear Relationships: DNNs are highly capable of capturing complex non-linear relationships between input features and the target variable. Traditional regression models, such as linear regression or decision trees, often assume linear relationships or limited non-linearities. DNNs can model intricate and non-linear patterns in the data, allowing for more accurate predictions.
2. Feature Learning: DNNs have the ability to automatically learn relevant features from the raw input data. Instead of relying on manual feature engineering, where domain knowledge and expert insights are required, DNNs can learn hierarchical representations of the data through multiple hidden layers. This can be particularly beneficial when dealing with high-dimensional or unstructured data, such as images or text.
3. Handling Complex Data: DNNs are well-suited for handling complex data types, including images, audio, text, and time series. They can effectively process and extract meaningful features from these data types, enabling them to capture intricate patterns and dependencies that might be challenging for traditional regression models.
4. Scalability: DNNs are highly scalable and can handle large datasets with millions of samples and high-dimensional feature spaces. They can efficiently leverage parallel processing and GPU acceleration to train models on massive datasets. This scalability makes DNNs suitable for big data applications and scenarios where traditional regression models may struggle due to computational limitations.
5. Transfer Learning and Pretrained Models: DNNs benefit from transfer learning, where a model trained on a large, diverse dataset can be fine-tuned on a smaller dataset specific to the regression task at hand. This allows the model to leverage the knowledge learned from the larger dataset and achieve good performance even with limited labeled data. Additionally, pretrained DNN models, such as those trained on ImageNet for image recognition, can be used as feature extractors for regression tasks, saving training time and improving performance.
6. Regularization and Generalization: DNNs offer various regularization techniques, such as dropout and L1/L2 regularization, which help prevent overfitting and improve generalization. Regularization is particularly important in regression tasks to avoid overemphasizing noisy or irrelevant features and to encourage the model to focus on meaningful patterns in the data.
7. Ensemble Learning: DNNs can be used as base models in ensemble learning frameworks, such as stacking or boosting, to further improve prediction accuracy. By combining multiple DNN regressors, each trained on different subsets of the data or with different hyperparameters, ensemble models can effectively capture diverse patterns and reduce model bias.

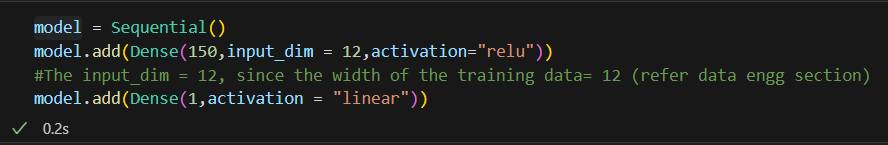
It is worth noting that while DNNs offer powerful capabilities, they also require substantial computational resources, longer training times, and a sufficient amount of labeled data for effective training. Additionally, interpreting the inner workings of DNNs can be challenging due to their black-box nature.

Overall, using DNNs as regressors can provide significant advantages in capturing complex relationships, handling diverse data types, and achieving high predictive accuracy in regression tasks, making them a valuable alternative to traditional regression models.

### Designing the DNN

* Start with small architectures.
* When small architectures (with two layers) fail, increase the size.
* When larger networks with two layers fail, go deeper.
* When larger and deeper networks also fail, go even larger and even deeper.
* When everything fails, revisit the data

**Create Deep Neural Network Architecture**

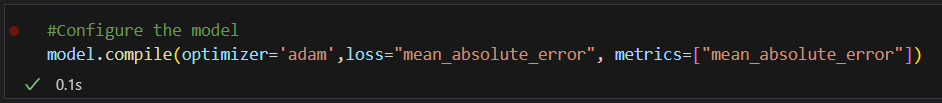


The code provided represents the construction of a neural network model using the Keras library with a Sequential API. Here is a breakdown of the code:

1. **model = Sequential()**: This line initializes a sequential model, which is a linear stack of layers. It allows for the easy construction of neural networks by sequentially adding layers to the model.
2. **model.add(Dense(150, input\_dim=12, activation="relu"))**: This line adds a dense layer to the model. A dense layer is a fully connected layer where each neuron is connected to every neuron in the previous layer. The **150** parameter represents the number of neurons in this layer. The **input\_dim=12** specifies that the input shape to this layer is 12, corresponding to the width of the input data. The **activation="relu"** sets the activation function of the neurons in this layer to Rectified Linear Unit (ReLU), which introduces non-linearity to the model.
3. **model.add(Dense(1, activation="linear"))**: This line adds another dense layer to the model. This layer has a single neuron, as specified by **1**. The **activation="linear"** sets the activation function of this neuron to linear, which means it will produce a continuous output without any non-linearity.

In summary, the code constructs a sequential model with two dense layers. The first dense layer has 150 neurons with ReLU activation, and the second dense layer has a single neuron with linear activation. The model is designed for a regression task where the output is a continuous value.

### Configure the model



The code snippet provided is used to configure the previously defined neural network model. Here's a breakdown of the code:

1. **model.compile()**: This function configures the model for training. It specifies the optimizer, loss function, and optional metrics to be used during training and evaluation.
2. **optimizer='adam'**: The **optimizer** parameter specifies the optimization algorithm used to update the weights of the neural network during training. In this case, "adam" is used, which is a popular optimization algorithm known for its efficiency and effectiveness in training neural networks.
3. **loss="mean\_absolute\_error"**: The **loss** parameter defines the loss function to be used during training. In this case, "mean\_absolute\_error" is chosen as the loss function. It calculates the mean absolute error between the predicted values and the true values of the target variable. The goal of the training process is to minimize this error.
4. **metrics=["mean\_absolute\_error"]**: The **metrics** parameter specifies the evaluation metric(s) to be used during training and evaluation. Here, "mean\_absolute\_error" is used as the metric. It provides the mean absolute error between the predicted values and the true values as an additional metric to monitor the model's performance.

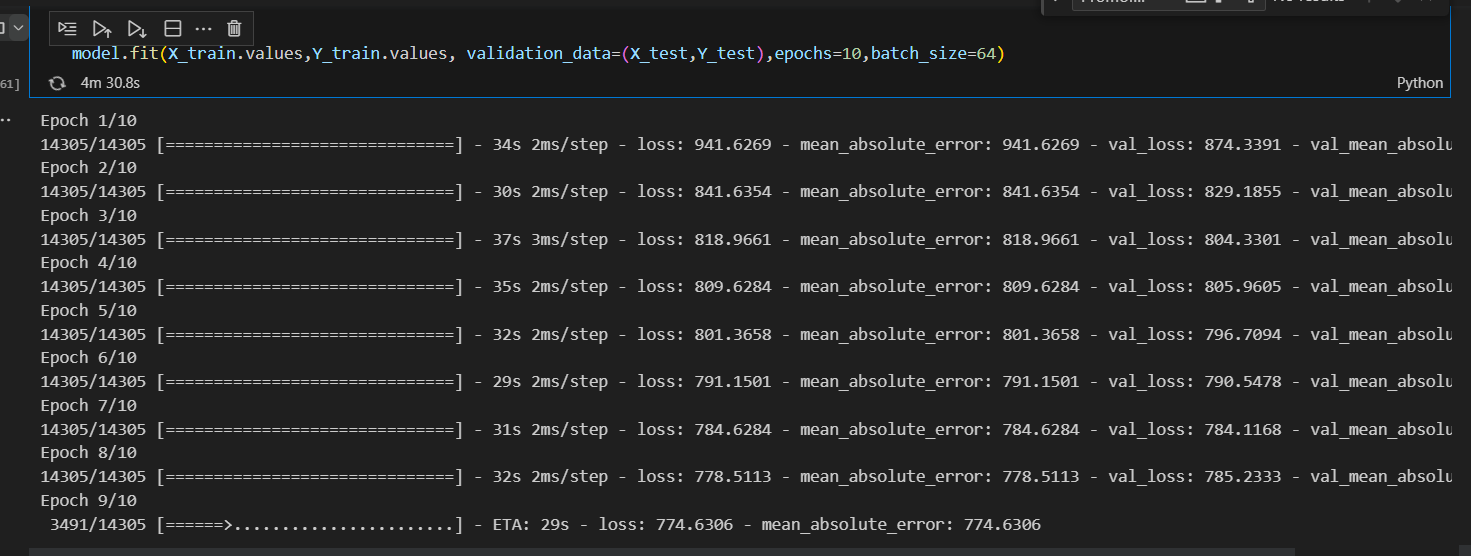
By calling **model.compile()** with the specified optimizer, loss function, and metrics, the model is now ready to be trained using the provided configuration.

### Train the model

The code snippet provided is used to train the previously configured neural network model. Here's an explanation of the code:

1. **model.fit()**: This function is used to train the model on the provided training data. It updates the model's weights based on the input data and target values.
2. **X\_train.values** and **Y\_train.values**: These represent the training data and corresponding target values, respectively. **X\_train.values** contains the input features, and **Y\_train.values** contains the corresponding target variable values. Both are passed as arguments to **model.fit()** for training.
3. **validation\_data=(X\_test, Y\_test)**: This parameter specifies the validation data to be used during training. The model's performance on this data will be evaluated after each epoch. **X\_test** contains the validation input features, and **Y\_test** contains the corresponding target variable values.
4. **epochs=10**: The **epochs** parameter defines the number of times the entire training dataset will be passed through the model during training. In this case, the training process will iterate over the data 10 times.
5. **batch\_size=64**: The **batch\_size** parameter specifies the number of samples that will be propagated through the model at once before the model's weights are updated. It helps in controlling memory usage and computational efficiency. In this case, each batch will contain 64 samples.

By calling **model.fit()** with the provided training data, validation data, number of epochs, and batch size, the model will be trained on the training data. During training, the model's performance will be evaluated on the validation data, and the weights of the model will be updated based on the optimization algorithm specified during model compilation.

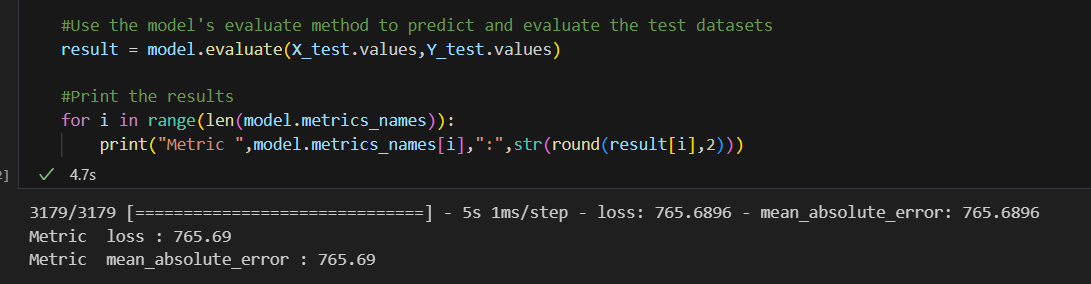


**Training Progress**

### Testing the Model Performance

After training on 10 epochs

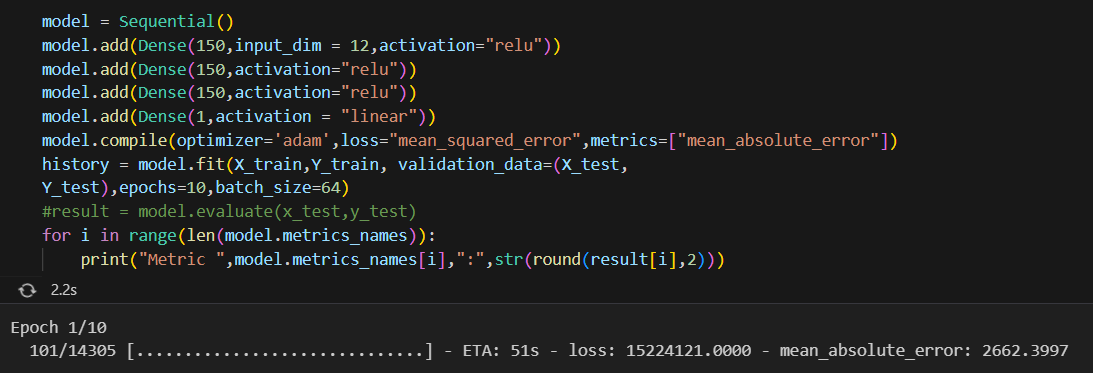
We had evaluation metrics are



**We can see the MAE plummeted from 2878 to 765 showing the clear effectiveness of Deep Learning**

### Improving the Model

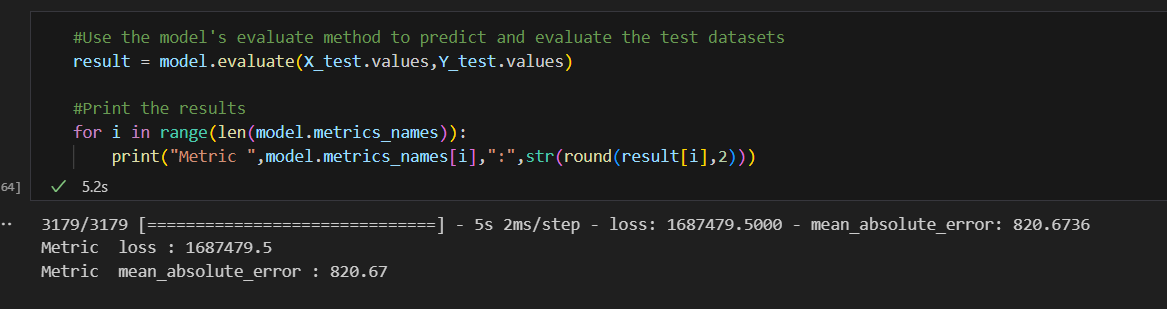
* In the following network, we have added two more layers with similar numbers of neurons
* We will update our loss function to mean squared error instead of MAE



The code snippet provided represents the construction, compilation, training, and evaluation of a neural network model. Here's a breakdown of the code:

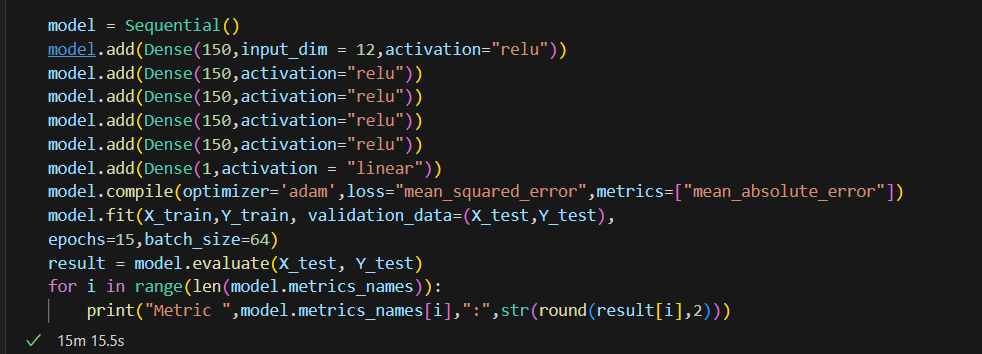
1. Model Architecture:
   * **model = Sequential()**: Initializes a sequential model.
   * **model.add(Dense(150, input\_dim=12, activation="relu"))**: Adds a dense layer with 150 neurons and ReLU activation as the input layer.
   * **model.add(Dense(150, activation="relu"))**: Adds another dense layer with 150 neurons and ReLU activation.
   * **model.add(Dense(150, activation="relu"))**: Adds a third dense layer with 150 neurons and ReLU activation.
   * **model.add(Dense(1, activation="linear"))**: Adds a final dense layer with 1 neuron and linear activation, serving as the output layer.
2. Model Compilation:
   * **model.compile(optimizer='adam', loss="mean\_squared\_error", metrics=["mean\_absolute\_error"])**: Compiles the model using the Adam optimizer, mean squared error as the loss function, and mean absolute error as the evaluation metric.
3. Model Training:
   * **history = model.fit(X\_train, Y\_train, validation\_data=(X\_test, Y\_test), epochs=10, batch\_size=64)**: Trains the model on the provided training data (X\_train and Y\_train) for 10 epochs, with a batch size of 64. The validation data (X\_test and Y\_test) is used to evaluate the model's performance during training.
4. Model Evaluation:
   * **result = model.evaluate(x\_test, y\_test)**: Evaluates the trained model on the test data (x\_test and y\_test) and stores the evaluation results in the **result** variable.
   * **for i in range(len(model.metrics\_names)):**: Iterates over the metrics names of the model.
     + **print("Metric ", model.metrics\_names[i], ":", str(round(result[i], 2)))**: Prints the name of each metric and its corresponding value, rounded to 2 decimal places.

Overall, the code constructs a neural network model with three hidden layers, trains it on the provided training data, and evaluates its performance using mean squared error as the loss function and mean absolute error as the evaluation metric. The evaluation results are then printed for each metric.



We see worse results than the previous model (MAE: 765 -> 820). We will proceed to improve this model more "deep".

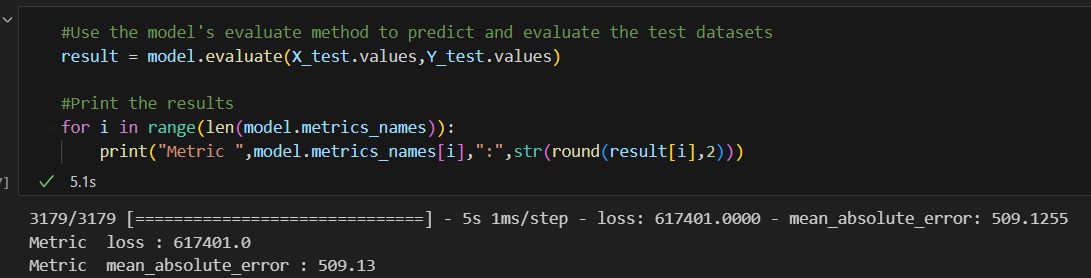
**Improving the Model Continue**



The second code snippet represents an improved version of the neural network model compared to the first code snippet. Here are the differences and improvements:

1. Model Architecture:
   * The second code snippet has an additional dense layer compared to the first code snippet. It adds another **model.add(Dense(150, activation="relu"))** layer, resulting in a deeper neural network with more layers.
   * The second code snippet has a total of five dense layers, while the first code snippet has four dense layers.
2. Model Training:
   * The second code snippet increases the number of epochs from 10 to 15. This allows the model to train for a longer period and potentially capture more complex patterns in the data.
   * The second code snippet does not store the training history in a variable (**history**). It directly fits the model on the training data without assigning the history object.
3. Model Evaluation:
   * The second code snippet includes the evaluation of the model's performance on the test data (**x\_test** and **y\_test**). It calculates the evaluation metrics using the **model.evaluate()** method and stores the results in the **result** variable.
   * The first code snippet does not have the evaluation step or the printing of the evaluation metrics.

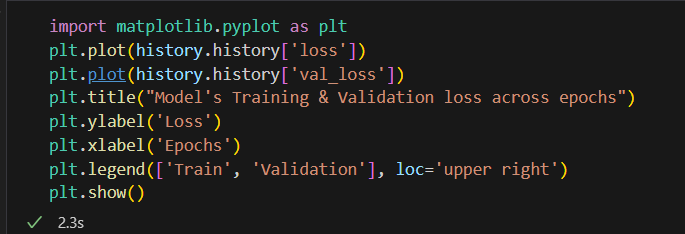
In summary, the second code snippet improves the model by adding an additional dense layer, increasing the number of training epochs, and evaluating the model's performance on the test data. These changes aim to enhance the model's capacity to learn complex patterns and provide a better understanding of its performance through the evaluation metrics.

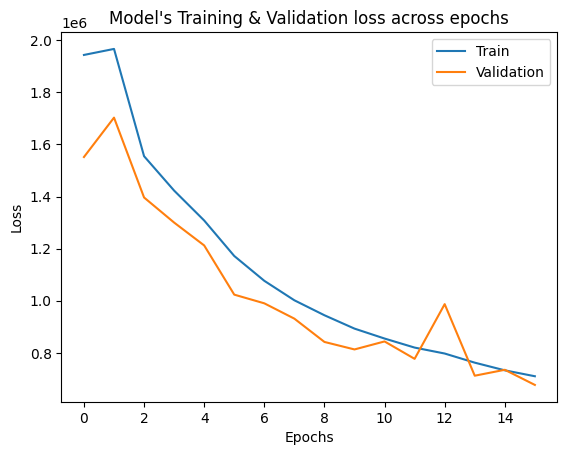


**And after training, we get better result: MAE: 509.13**

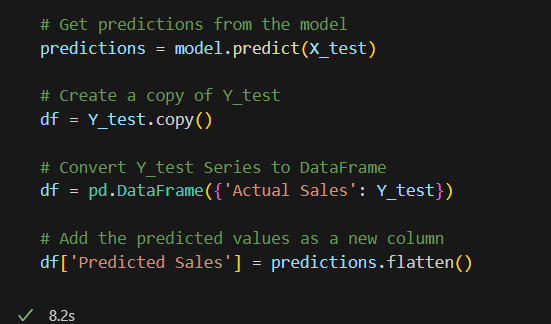
We will use this model as final model to get result

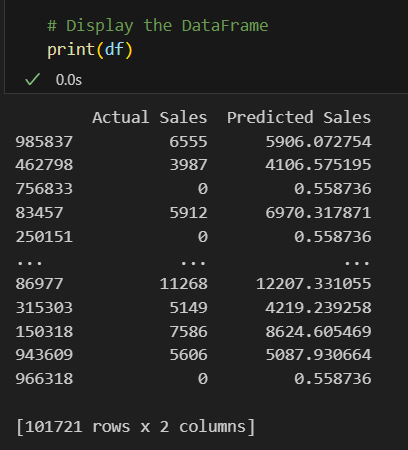
### Plotting the Loss Metric Across Epochs



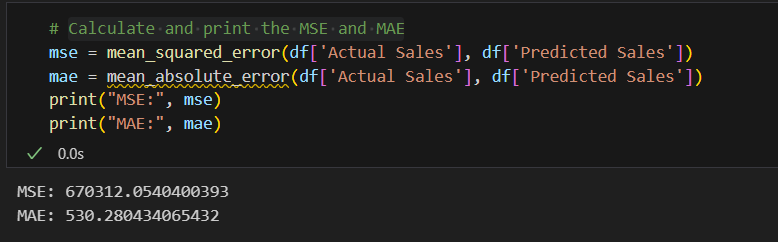


### Testing the Model Manually





### Calculate and print the MSE and MAE



Based on the provided MSE (Mean Squared Error) and MAE (Mean Absolute Error) values, it appears that the DNN model's performance is not optimal for the Rossman dataset.

### Conclusion

The MSE value of 670312.05 suggests that there is a significant amount of variation between the predicted sales values and the actual sales values. A higher MSE indicates larger errors in the predictions, and in this case, the MSE value is quite high.

It is a significant improvement in terms of reducing the average deviation of predicted sales from the actual sales. This indicates that your DNN model has made considerable progress in capturing the patterns and trends in the Rossman dataset, resulting in more accurate predictions.

# CHAPTER 4: DEPLOYMENT

**Now we will build a Graphical User Interface (GUI) to apply visual problem and results using Streamlit**

Deployment in machine learning refers to the process of deploying a trained machine learning model into a production environment for practical use. After developing and training a machine learning model on the training data, deployment allows the model to be used for making predictions on new data in real-world scenarios.

## 4.1. Streamlit



Streamlit is an open-source Python library used for building interactive web applications for data science and machine learning projects. It simplifies the process of creating and sharing interactive dashboards, visualizations, and applications without requiring extensive web development knowledge.

Key features and benefits of Streamlit include:

1. Easy-to-use: Streamlit is designed to be user-friendly and accessible for data scientists and developers. It provides a simple and intuitive API that allows you to quickly create interactive web applications.
2. Rapid prototyping: Streamlit enables rapid development and prototyping of data-driven applications. You can write code in a single Python script and see the results immediately in your browser.
3. Built-in widgets: Streamlit provides a set of built-in widgets and interactive components that allow users to interact with the application. These widgets include sliders, checkboxes, dropdowns, and more, making it easy to create interactive controls for data exploration and analysis.
4. Integration with popular libraries: Streamlit seamlessly integrates with popular data science libraries such as Pandas, Matplotlib, and Plotly. You can leverage these libraries to manipulate and visualize data within your Streamlit application.
5. Automatic reactivity: Streamlit automatically updates the application in real-time as you modify the code. This allows you to easily experiment with different parameters and see the immediate impact on the visualizations or outputs.
6. Sharing and deployment: Streamlit applications can be easily shared and deployed on various platforms, including local servers, cloud services, and containerized environments. You can deploy your applications with a few simple commands, making it easy to share your work with others.

## 4.2. Graphical User Interface (GUI)

Out Streamlit app includes 3 pages: **Home, Data Visualization, Data Prediction**

By dividing our Streamlit app into these three pages, we provide a user-friendly and organized interface for data exploration, visualization, and prediction. Users can seamlessly navigate through the app, gain insights from the data, and leverage the power of machine learning models to make informed decisions or predictions.

**Home Page:**

* The Home page serves as the landing page for our Streamlit app. It provides an overview and introduction to the purpose and functionality of the application.
* On this page, we can include a brief description of the dataset (in this case, the Rossman dataset) and the problem we are trying to solve.

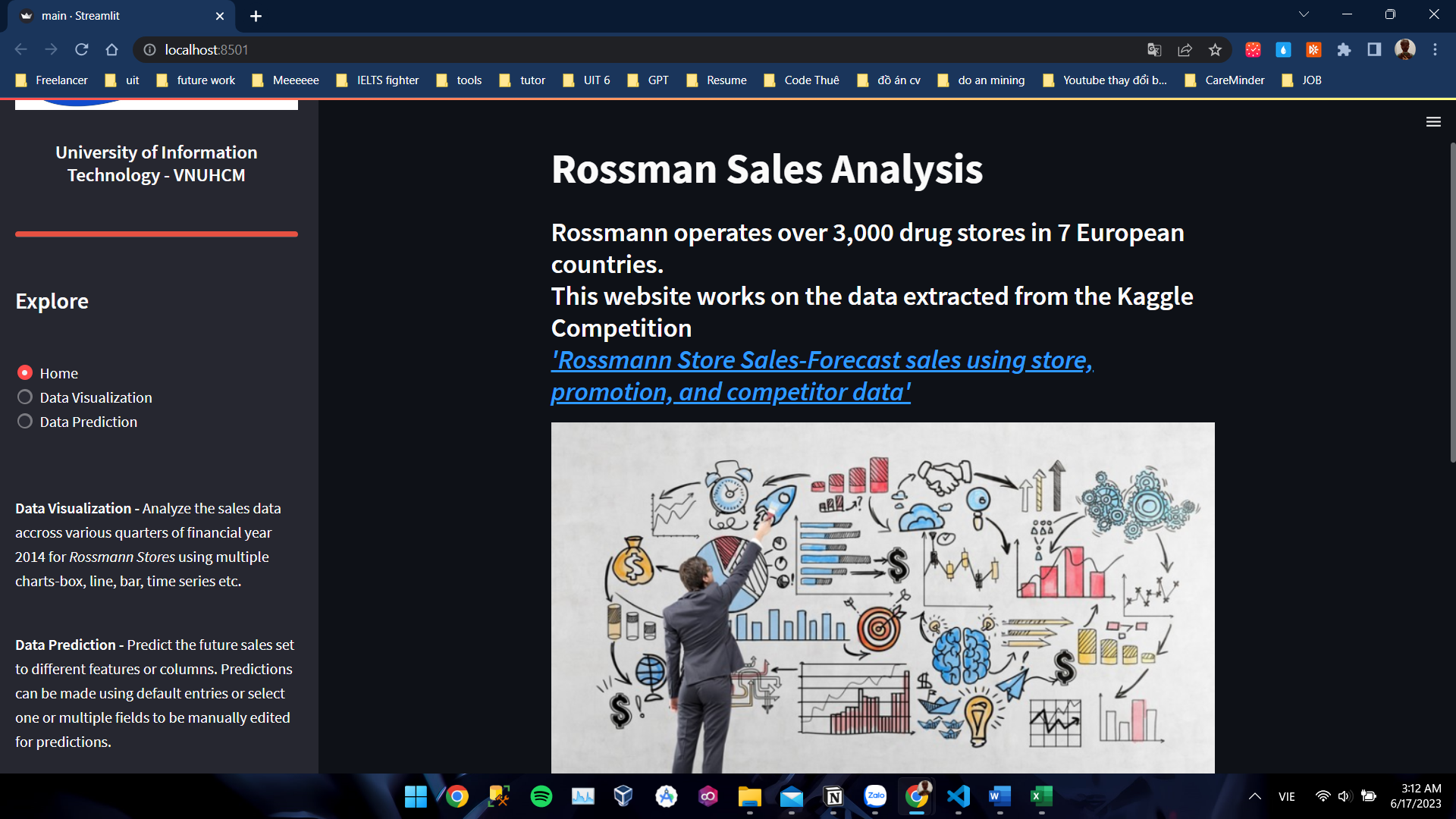
**Data Visualization Page:**

* The Data Visualization page is designed to provide users with interactive visualizations of the dataset. We can leverage the capabilities of libraries like Matplotlib, Seaborn, or Plotly to create various types of charts and graphs.
* Users can explore different aspects of the dataset, such as sales trends over time, correlations between variables, or geographical distribution of stores. They can interact with the visualizations by selecting specific variables, adjusting parameters, or zooming in and out.
* This page allows users to gain insights from the data visually and make informed decisions based on the observed patterns or trends.

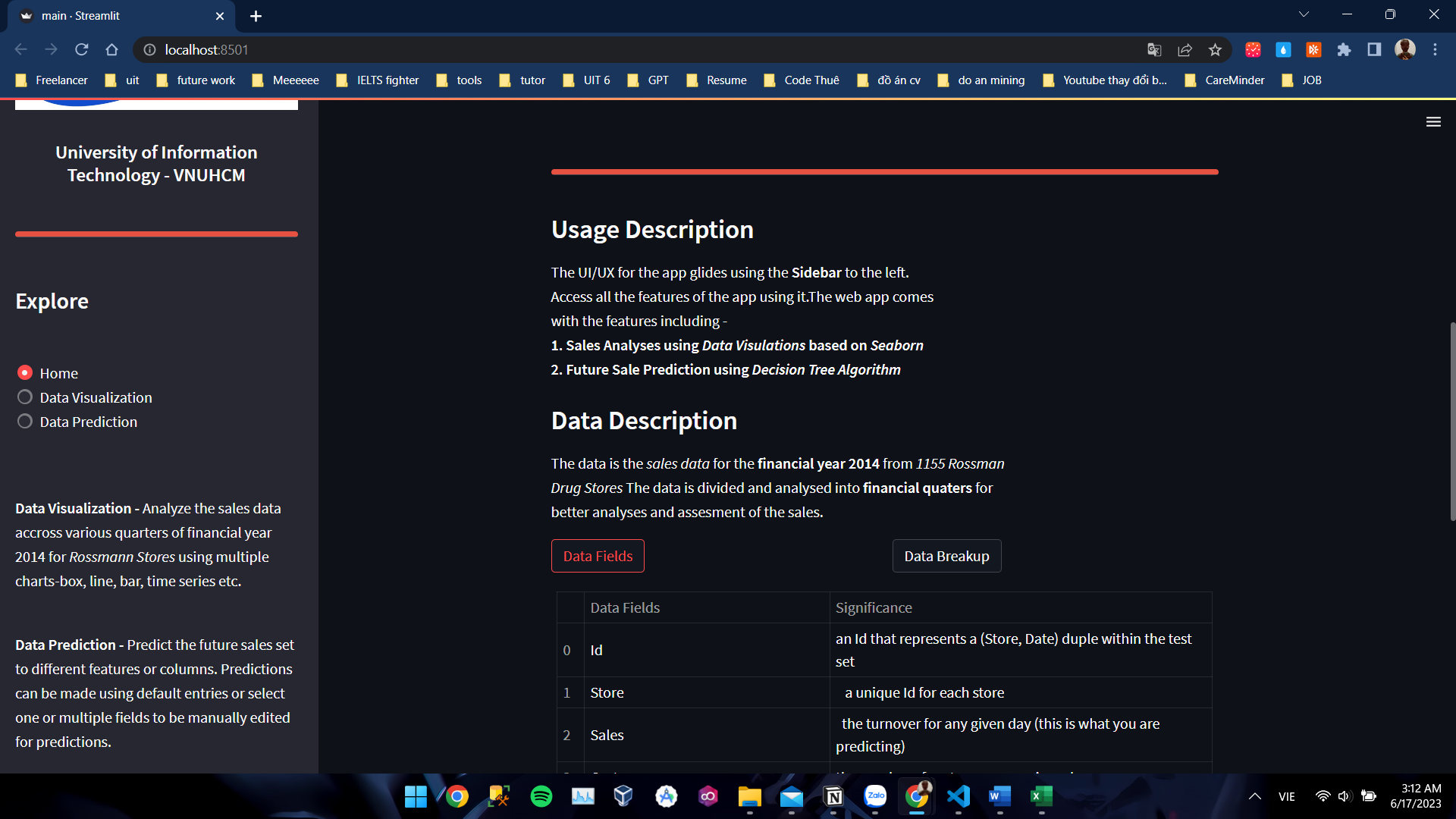
**Data Prediction Page:**

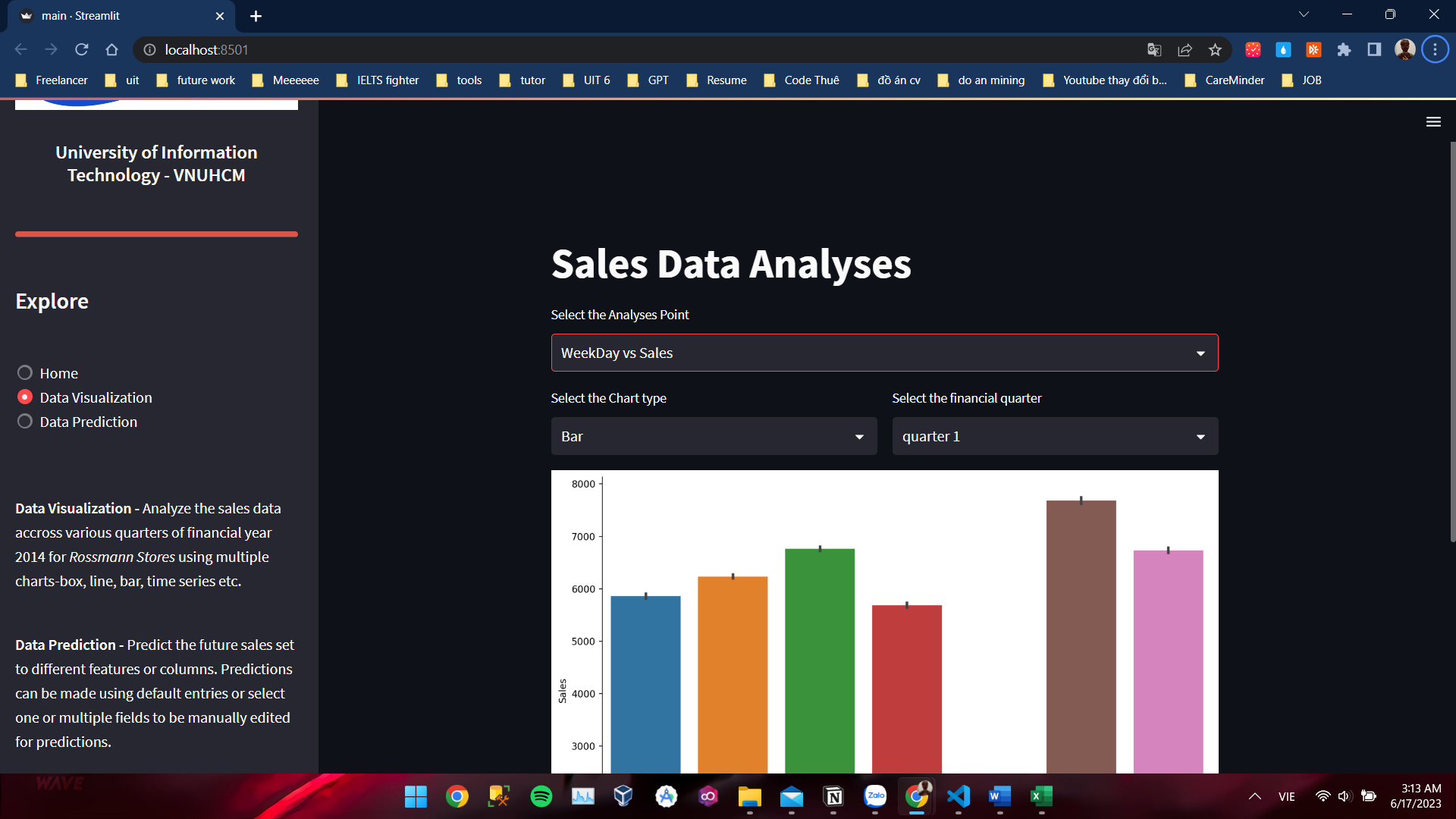
* The Data Prediction page is where users can input specific data points and obtain predictions from our trained machine learning models.
* Users can enter relevant information, such as store features or other variables, and the models will generate predictions for the target variable (e.g., sales).
* The predictions can be displayed in a tabular format, along with the corresponding input values, allowing users to compare the actual and predicted values.

**Some results**

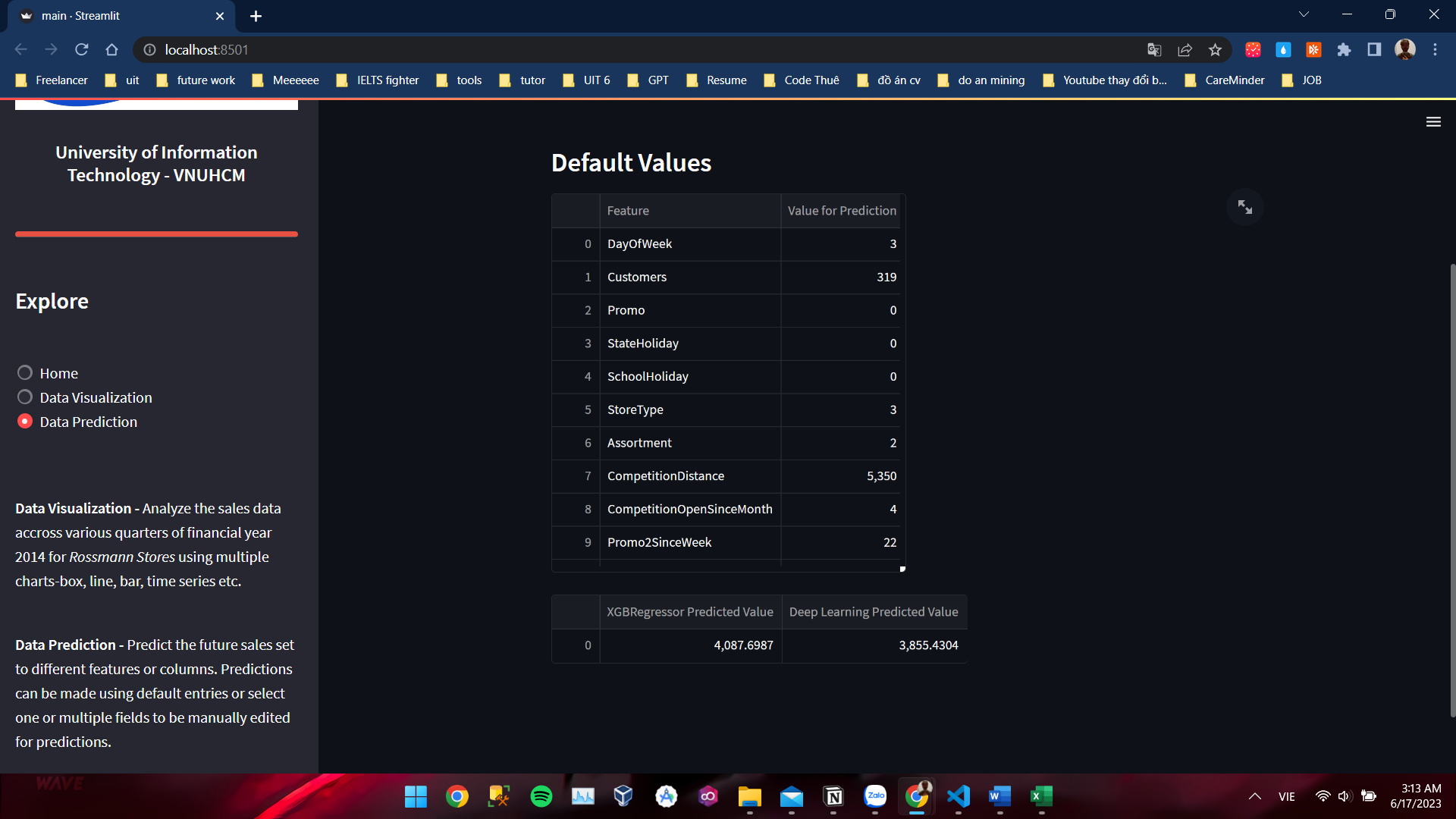


Homepage

**Home page with Usage, Data Description, Data Fields, Data Breakup**



**Data Visualization**



**Data Prediction**

## 4.3. Conclusion

In this project, we focused on the analysis and prediction of sales in the retail industry using the Rossman dataset. We began by identifying the problems and motivations behind this study, followed by an overview of the dataset and the tools and sources utilized.

The data preprocessing process played a crucial role in preparing the dataset for analysis. We performed various cleaning and transformation steps, including merging the train.csv and store.csv data, narrowing down the dataset, and adding additional features for better visualization and labeling. Exploratory data analysis was then conducted, revealing insights into the relationships between variables such as StoreType, WeekDay, MonthName, and sales.

For modeling, we employed two approaches: XGB Regressor and Deep Learning with Keras. The XGB Regressor algorithm provided a powerful method for predicting sales based on the available features. On the other hand, the Deep Learning model, with its multi-layered neural network architecture, offered more flexibility and potential for capturing complex patterns in the data.

We designed, trained, and evaluated the Deep Learning model, continuously improving its performance through iterations. The model's effectiveness was assessed using metrics such as mean squared error (MSE) and mean absolute error (MAE).

In terms of deployment, we utilized Streamlit, a user-friendly framework for creating interactive and visually appealing web applications. We developed a graphical user interface (GUI) that included three main pages: Home, Data Visualization, and Data Prediction. These pages allowed users to explore the dataset visually, analyze sales trends, and make predictions based on the trained models.

Overall, this project demonstrated the value of data preprocessing, exploratory data analysis, and modeling techniques in understanding and predicting sales patterns. The deployment of the models using Streamlit showcased the potential for creating user-friendly interfaces to interact with machine learning models in real-world scenarios.

By leveraging the insights and predictions generated through this project, retailers and businesses can make informed decisions, optimize their operations, and improve their sales performance in a competitive market.

# CHAPTER 5: REFERENCES

1. XGBoost Documentation: The official documentation for XGBoost, an efficient gradient boosting framework
2. Keras Documentation: The official documentation for Keras, a high-level neural networks API
3. Scikit-learn Documentation: The official documentation for Scikit-learn, a popular machine learning library in Python
4. Kaggle: A platform for data science and machine learning, offering a wide range of datasets and competitions
5. Streamlit Documentation: The official documentation for Streamlit, a framework for building interactive web applications for machine learning and data science
6. Python Data Science Handbook: A comprehensive guide to Python for data science, covering various libraries and techniques