Diego Armando Salinas Lugo

CE889
Neural Networks and Deep Learning

Postgraduate : 2401168
username : ds24353

ROCKET LANDING

After collecting data of the game, I have a file to normalize the data.

```
ers > Salin > OneDrive > Documentos > ESSEX > Neural Networks > ds24353_DiegoArmando_SalinasLugo_C
import pandas as pd
from sklearn.preprocessing import MinMaxScaler

# To read my csv with my last weights achieved after training
file_path = r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\OneDrive_2

# Loading it
data = pd.read_csv(file_path, header=None)

# Assigning the column order
data.columns = ['x1', 'x2', 'y1', 'y2']

# Applying Min-Max normalization
scaler = MinMaxScaler()
normalized_data = pd.DataFrame(scaler.fit_transform(data), columns=data.columns)

# Saving the new normalized file to work with
output_file_path = r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\One
normalized_data.to_csv(output_file_path, index=False)

print(f"Normalized data has been saved to {output_file_path}")
```

Then, I have my most important file which I called try5.py. In this file, I have a class called Neural Network in which I split the data in 0.7 for trainning and 0.3 for testing.

In try5

This file contains the methods for forward propagation and backward propagation. Adapted with some functions to ellaborate the whole process.

```
if __name__ == "__main__":
    neurons = 5
    weights = [random.uniform(-1, 1) * (1 / neurons ** 0.5) for _ in range(neurons * 2 + neurons * 2)]
    b = 1  # Initial bias to be updated
    # At the beginning I was trying to work with the parameters i got from matlab but my model wasnt working.
    # They were 5 neurons, lambda 0.05 and trainparam of 0.9, but after trying plenty of times, these values showed more movement in the rocket
    nn = NeuralNetwork(λ=0.08, weights=weights, M=1.05, b=b, neurons=neurons)

    # Path to my CSV file with the normalize data
    csv_file_path = r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\OneDrive_2024-11-25\Assignment Code\normalized_ce889_dataCollection.csv"

    # Trainning and evaluating
    nn.train_and_evaluate(csv_file_path, epochs=150)
```

This part shows the last lines of my code.

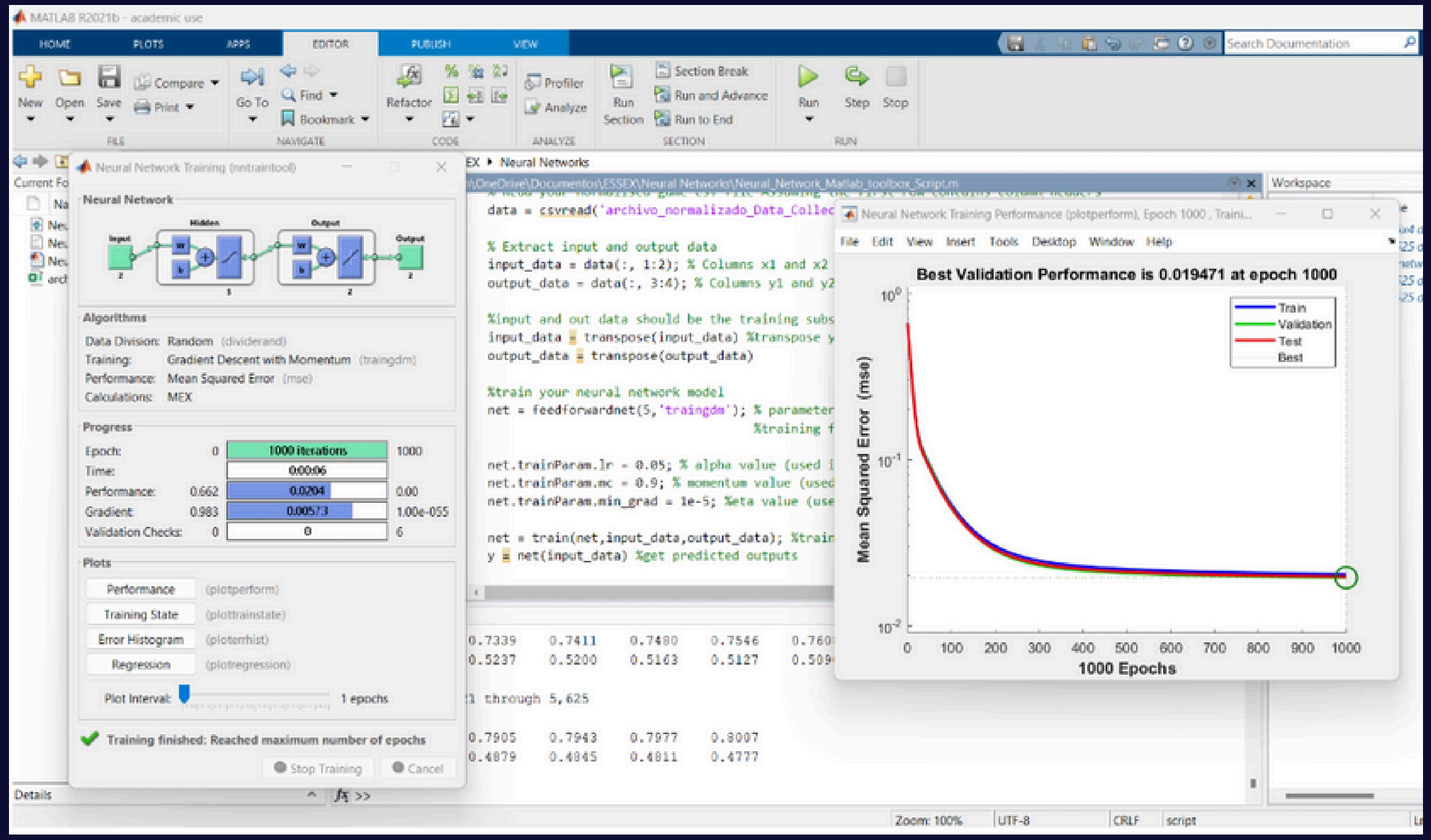
Here I dictate the parameters to use, the direction of my csv and I run the train_and_evaluate function that consists in running all the functions in the class.

At initializing the nodes I implemented the Xavier Initialization principle since I discovered that it ensures balanced gradients for efficient learning and preventing issues like vanishing or exploding gradients.

I started working with 150 epochs since with 100 epochs and other parameters I tought the loss would be better, however, at last moment I got better parameters and noticed that the number of epochs can be less, nevertheless, i didn´t change this number.

ROCKET LANDING

After collecting data of the game, I have a file to normalize the data.



At the beginning I was trying to work with the parameters i got from matlab but my model wasn´t working, the rocket was just falling showing not movement at all.

They were 5 neurons, lambda 0.05 and trainparam of 0.9, but after trying plenty of times, these other values showed more and better movement in the rocket:
neurons = 5, λ = 0.08 and trainparam of 1.05.

The function with which I run the whole performance.

```
# To get into the training and then the testing
def train_and_evaluate(self, file_path, epochs):
    x1_train, x2_train, y_train, x1_test, x2_test, y_test = self.train_test_split(file_path)

    for epoch in range(epochs):
        self.train_batch(x1_train, x2_train, y_train)
        train_loss = self.compute_loss(x1_train, x2_train, y_train)
        print(f"Epoch {epoch + 1}: Train Loss = {train_loss:.6f}")
        print(f"Epoch {epoch + 1}: Updated Weights = {self.weights}")
        print(f"Epoch {epoch + 1}: b = {self.b}")

    test_loss = self.compute_loss(x1_test, x2_test, y_test)
    print(f"Final Test Loss: {test_loss:.6f}")
```

The functions called in the train_evaluate, that are in charge to split the data and train the batch

```
# Trainning the network
def train_batch(self, x1_list, x2_list, y_list):
    for x1, x2, y in zip(x1_list, x2_list, y_list): #Looping through the input and output, using zip
        h, yp = self.forward_propagate(x1, x2) # To perform forward with the inputs
        self.backward_propagate(x1, x2, y, h, yp) # To adjust the weights with new data

# Splitting the data into to its manage
def train_test_split(self, file_path, train_ratio=0.7):
    x1_list, x2_list, y_list = [], [], [] # Initializing the lists to store the data
    with open(file_path, newline='') as csvfile:
        reader = csv.reader(csvfile)
        next(reader) # To skip the header
        for row in reader:
            x1, x2, y1, y2 = map(float, row) # Converting the values to float
            x1_list.append(x1)
            x2_list.append(x2)
            y_list.append([y1, y2]) #Adding the ouput pair to a list. As there are two outputs, I st

    data = list(zip(x1_list, x2_list, y_list)) # zip combines the three lists (x1_list, x2_list, y_l
    random.shuffle(data)
    train_size = int(len(data) * train_ratio)
    train_data = data[:train_size] # data to train
    test_data = data[train_size:] # data to test
    x1_train, x2_train, y_train = zip(*train_data) # Separating the inp and outp and converting to l
    x1_test, x2_test, y_test = zip(*test_data)
    return list(x1_train), list(x2_train), list(y_train), list(x1_test), list(x2_test), list(y_test)
```

ROCKET LANDING

The forward propagation, backward propagation and loss

```
def forward_propagate(self, x1, x2):
    v = []  # Pre activation values
    h = []  # Hidden layer outputs
    yp = []  # Output layer outputs

    # Activations for the hidden layer
    for i in range(self.neurons):
        v_hidden = x1 * self.weights[i] + x2 * self.weights[i + self.neurons] + self.b # Multiplying the first an second in
        v.append(v_hidden) # Storing pre-activation
    for v_hidden in v:
        h.append(self.sigmoid(v_hidden)) # Applying sigmoid activation adn appending

    # Activations for the output layer
    for i in range(2):  # Output layer has 2 neurons
        v_output = sum(h[j] * self.weights[self.neurons * 2 + i * self.neurons + j] for j in range(self.neurons)) + self.b
        v.append(v_output) # Storing pre activation
        yp_output = self.sigmoid(v_output) # Applying sigmoid activation
        yp.append(yp_output) # Adding to its corresponding list

    return h, yp # Returning the hidden layer and output layer activations
```

Following the formulas of:

- Pre-activation and Activation Values
- Sigmoid Activation Function
- Error Calculation
- Backpropagation and Gradients:
  - Output Layer Gradients (Delta) : δoutput = learning rate · error · sigmoid'(y^)
  - Hidden Layer Gradients (Delta) : δhidden = learning rate * h · (1−h) * j∑(δoutputj · wij)
- Weight Update Rule
- Mean Squared Error Loss

```
class NeuralNetwork:
    # To update weights and bias
    def backward_propagate(self, x1, x2, y, h, yp):
        zeta_output = [] # To store the output layer errors
        zeta_hidden = [] # Hidden layer errors

        # Calculating errors for the output layer
        for i in range(2):
            error = y[i] - yp[i]
            delta_output = self.M * error * yp[i] * (1 - yp[i]) # Derivative of sigmoid, i could have adde
            zeta_output.append(delta_output) # Appending the errors

        # For the hidden layer is a more complex procedure
        for i in range(self.neurons):
            sum_delta_weights = sum( #Calculates the sum
                zeta_output[j] * self.weights[self.neurons * 2 + j * self.neurons + i] for j in range(2) #
            )
            delta_hidden = self.M * h[i] * (1 - h[i]) * sum_delta_weights # The formula for getting the de
            zeta_hidden.append(delta_hidden) #Appending to the list

        # Updating the weights for the output layer
        for i in range(2):
            for j in range(self.neurons):
                self.weights[self.neurons * 2 + i * self.neurons + j] += self.M * zeta_output[i] * h[j]

        # Updating weights for the hidden layer
        for i in range(self.neurons):
            self.weights[i] += self.M * zeta_hidden[i] * x1
            self.weights[i + self.neurons] += self.M * zeta_hidden[i] * x2

        # Updating the bias
        for i in range(2):
            self.b += self.M * zeta_output[i]
        for i in range(self.neurons):
            self.b += self.M * zeta_hidden[i]
```

```
# To compute mean squared error loss
def compute_loss(self, x1_list, x2_list, y_list):
    total_error = 0 # Initializing the error
    for x1, x2, y in zip(x1_list, x2_list, y_list):
        _, yp = self.forward_propagate(x1, x2) # Predicting output
        for i in range(2):
            total_error += (y[i] - yp[i]) ** 2 # Sum of squared differences
    return total_error / len(x1_list) # Average error
```

ROCKET LANDING

The outputs showed in the terminal are the weights updated, the loss, the bias by epoch and the final loss.

I save the last weights printed and store them in a csv file, to then in the NeuralNetholder.py call the class and make the forward propagation run with these new weights.

```python
import csv
from try5 import NeuralNetwork  # Import the NeuralNetwork class

class NeuralNetHolder:
    def __init__(self):
        super().__init__()
        # The path file
        self.weights_path = r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\trained_weights_try_5.csv"

        # Initializing the neural network with the parameters declared preoviusly BUT b
        # In training the model, i got a final b of 160.32444.... but if put it here my rocket just falls
        self.network = NeuralNetwork(λ=0.08, weights=[], M=1.05, b = 1, neurons=5)

        # Loading the previously trained weights achieved by training the models
        trained_weights = self.load_weights(self.weights_path) #To load the weights
        self.network.weights = trained_weights # Assigning the loaded weights to the neural network

        # Normalization scale
        self.input_scale = 100

    def load_weights(self, weights_path):

        weights = [] # to store the weights
        with open(weights_path, "r", encoding="utf-8-sig") as file:
            reader = csv.reader(file)
            for row in reader:
```

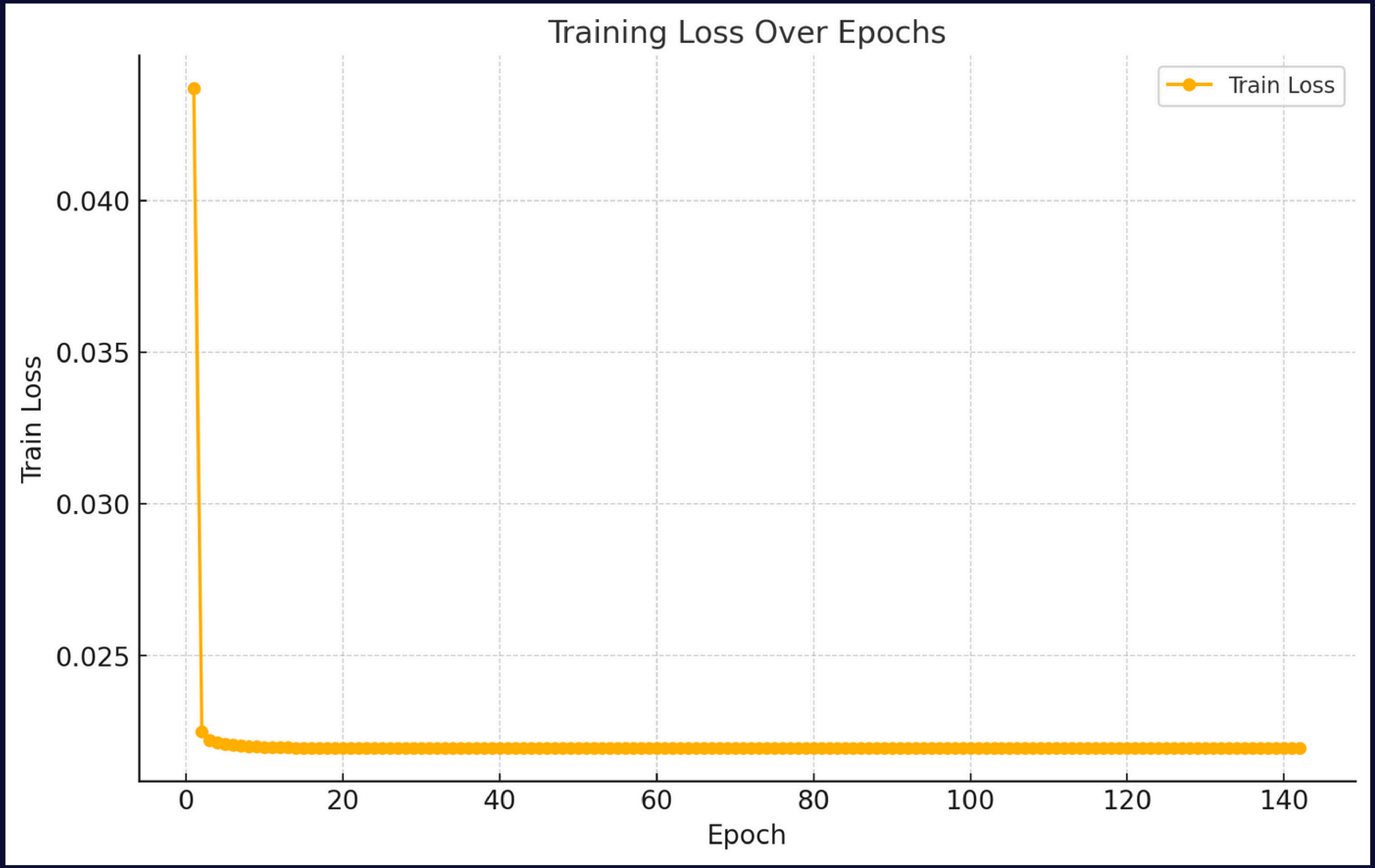
Eventhough the bias was being updating, here I let it as 1 because if I apply the last bias, the rocket would just fall. With one, the rocket moves.

```python
ef load_weights(self, weights_path):

    weights = [] # to store the weights
    with open(weights_path, "r", encoding="utf-8-sig") as file:
        reader = csv.reader(file)
        for row in reader:

            for value in row[0].split(','):  # This splits the rows by commas, to get the individual weights
                try:
                    # I was struggling at the moment of saving my csv file if the first element was negative because excel thinks i want to do a formula
                    # So, when the first is neg, i add an element like "'" and then i apply the following condition
                    # To remove leading/trailing whitespace and single quotes
                    cleaned_value = value.strip().lstrip("'")
                    weights.append(float(cleaned_value))
                except ValueError:
                    print(f"Skipping invalid value: {value}") # in case i typed an invalid value
    return weights
```

```python
def predict(self, input_row):

    x1, x2 = map(float, input_row.split(','))  # Assigning x1,x2 and ensuring its floating type

    # Normalizing the inputs
    x1_normalized = x1 / self.input_scale
    x2_normalized = x2 / self.input_scale

    # After the trainning, for this, only forward_propagate is applied
    _, outputs = self.network.forward_propagate(x1_normalized, x2_normalized)

    # Taking them to their original sacle
    y1 = outputs[0] * self.input_scale
    y2 = outputs[1] * self.input_scale

    # Finally returning the predictions
    return y1, y2
```

ROCKET LANDING

After running my try5.py of my neuralnetwork which is being trained with the normalized data, I obtained 150 losses, that in graphic look llike this:



It shows a strong decrease at the beggining and then it stabalizes, which means for these last parameters used, less epochs can be applied.

The performance of my rocket

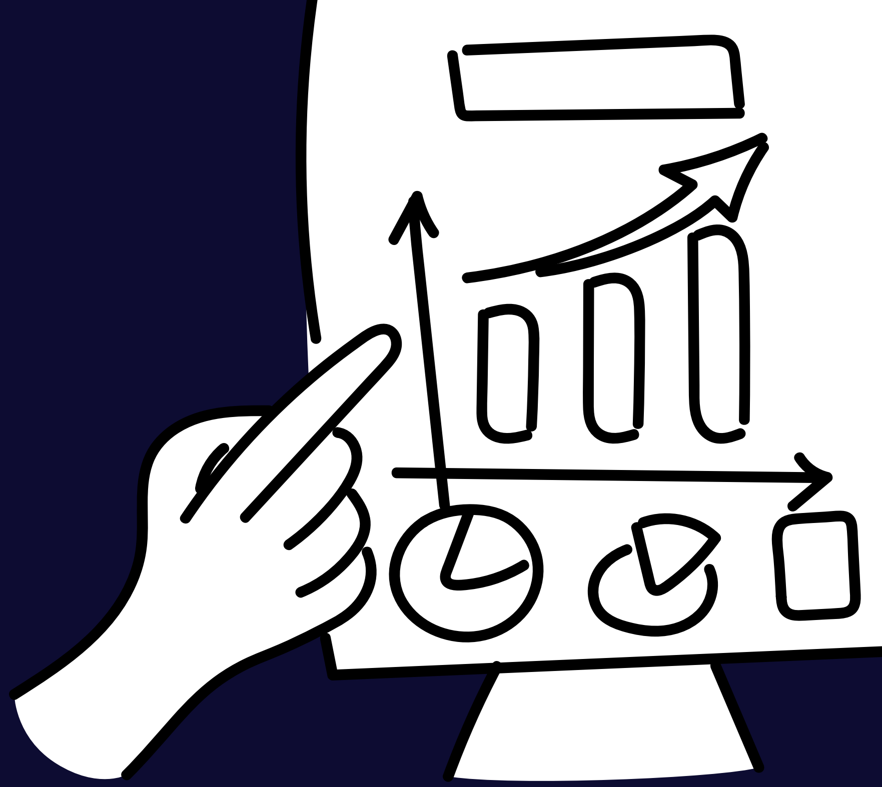
Some videos of the rocket performance:

https://youtube.com/shorts/LeMxOrQFNtU?feature=share

https://youtube.com/shorts/9aWIaSHWF5U?feature=share

https://youtube.com/shorts/XIR4RJvBst4?feature=share

DEEP LEARING TASK



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. Reliable sales forecasts enable store managers to create effective staff schedules that increase productivity and motivation.

Data Explorer

39.85 MB

- sample_submission.csv
- store.csv
- test.csv
- train.csv

The files given are : train (contains historical data), test (data to model and predict) and store (to provide additional information about each store).

First of all, preprocessing the data is truly important in order to proceed with the modelling for the predictions

For that, I created a file called Preprocessed.py, in which:

```python
import pandas as pd  # Data manipulation
import numpy as np  # Numerical operations

# Assigning the csv files to read them and to process the info
store_data = pd.read_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\store.csv")
test_data = pd.read_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\test.csv")
train_data = pd.read_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\train.csv")

# For 'store.csv'
# There are some missing values in 'CompetitionDistance', so they can be replaced with the median value
store_data['CompetitionDistance'] = store_data['CompetitionDistance'].fillna(store_data['CompetitionDistance'].median())

# for 'CompetitionOpenSinceMonth' and 'CompetitionOpenSinceYear', they can be filled with 0 since there is no data
store_data['CompetitionOpenSinceMonth'] = store_data['CompetitionOpenSinceMonth'].fillna(0)
store_data['CompetitionOpenSinceYear'] = store_data['CompetitionOpenSinceYear'].fillna(0)

# for 'Promo2SinceWeek' and 'Promo2SinceYear' same
store_data['Promo2SinceWeek'] = store_data['Promo2SinceWeek'].fillna(0)
store_data['Promo2SinceYear'] = store_data['Promo2SinceYear'].fillna(0)

# for missing values in 'PromoInterval' i fill with an empty string because there is no periodic promtion intervals
store_data['PromoInterval'] = store_data['PromoInterval'].fillna('')

# Now for 'test.csv'
# Filling the blanks in 'Open' with 1 assuming is open
test_data['Open'] = test_data['Open'].fillna(1)

# For easy manipulation, I convert 'Date' column in training and test data to datetime objects
train_data['Date'] = pd.to_datetime(train_data['Date'], format='%d/%m/%Y')
test_data['Date'] = pd.to_datetime(test_data['Date'], format='%d/%m/%Y')

# Applying Featuring Engineering at extracting useful date related features
# LOoping through both train and test datasets in order to create new important columns in the train_data and test_data
for dataset in [train_data, test_data]:
    dataset['Year'] = dataset['Date'].dt.year  # Extracting the year
    dataset['Month'] = dataset['Date'].dt.month  # Extracting the month
    dataset['Day'] = dataset['Date'].dt.day  # Extracting the day
```

```python
# Merging store data into train and test datasets
# For combining and build both files, i use the specific data of 'store' using 'Store' as the key
train_data = pd.merge(train_data, store_data, how='left', on='Store')
test_data = pd.merge(test_data, store_data, how='left', on='Store')

# To calculate Competition Open duration as number of months and leave this as a new feature

train_data['CompetitionOpen'] = 12 * (train_data['Year'] - train_data['CompetitionOpenSinceYear']) + \
                                 (train_data['Month'] - train_data['CompetitionOpenSinceMonth'])
test_data['CompetitionOpen'] = 12 * (test_data['Year'] - test_data['CompetitionOpenSinceYear']) + \
                                (test_data['Month'] - test_data['CompetitionOpenSinceMonth'])

# If it has not started, if there is negative values, replacing with zero
train_data['CompetitionOpen'] = train_data['CompetitionOpen'].apply(lambda x: max(0, x))
test_data['CompetitionOpen'] = test_data['CompetitionOpen'].apply(lambda x: max(0, x))

# Calculating duration for active Promo2 in months to create this new feature

# /4 to convert to months
train_data['Promo2Open'] = 12 * (train_data['Year'] - train_data['Promo2SinceYear']) + \
                            (train_data['WeekOfYear'] - train_data['Promo2SinceWeek']) / 4.0
test_data['Promo2Open'] = 12 * (test_data['Year'] - test_data['Promo2SinceYear']) + \
                           (test_data['WeekOfYear'] - test_data['Promo2SinceWeek']) / 4.0

# If it has not started, if there is negative values, replacing with zero
train_data['Promo2Open'] = train_data['Promo2Open'].apply(lambda x: max(0, x))
test_data['Promo2Open'] = test_data['Promo2Open'].apply(lambda x: max(0, x))

# Setting the now unnecessary columns for modelling
columns_to_drop = ['Date', 'Customers', 'CompetitionOpenSinceMonth', 'CompetitionOpenSinceYear',
                   'Promo2SinceWeek', 'Promo2SinceYear', 'PromoInterval']

# Dropping these columns if they exist in each file
train_data = train_data.drop(columns=[col for col in columns_to_drop if col in train_data.columns])
test_data = test_data.drop(columns=[col for col in columns_to_drop if col in test_data.columns])
store_data = store_data.drop(columns=[col for col in columns_to_drop if col in store_data.columns])
```

```python
# Saving the preprocessed datasets and creating the new cleaned files
train_data.to_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\train_preprocessed.csv", index=False)
test_data.to_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\test_preprocessed.csv", index=False)
store_data.to_csv(r"C:\Users\Salin\OneDrive\Documentos\ESSEX\Neural Networks\store_preprocessed.csv", index=False)

print("Data preprocessing complete. Preprocessed files saved.")
```

- Loads the three provided CSV files into Pandas DataFrames for processing.
- Handles Missing Values
- Converts Date columns to DateTime Format
- Extracts Date Features
- Combines the store-specific information from store.csv into the train and test datasets using the Store column as a key.
- Calculates Competition Open Duration
- Calculates Promo2 Active Duration
- Drops columns that are redundant or irrelevant for modeling.
- Saves the preprocessed data in new files to now work with the cleaned data

The way it was cleaned

For handling Missing Data:

- Fills Competition Distance with the median value.
- Fills CompetitionOpenSinceMoth and Year with 0, same with Promo2SinceWeek and Year
- Replaces missing PromoInterval with an empty string.
- Assumes stores in the test data were open if Open was missing.

Feature Engineering:

- Created time-based features:
  - Year, Month, Day, Week of the Year, Day of the Year.
- Derived new features:
  - CompetitionOpen: Duration (in months) since the store's competitor started operating.
  - Promo2Open: Duration (in months) since the store's promotion started.

The final columns were selected to focus on features that directly affect sales, such as temporal attributes (Year, Month, WeekOfYear), store details (CompetitionDistance, Promo2), and engineered features (CompetitionOpen, Promo2Open) that quantify competition and promotions. The columns no longer existing is beacuase they are redundant or uncosistent in matching, like Date and Customers which were removed to simplify the dataset and ensure consistency.

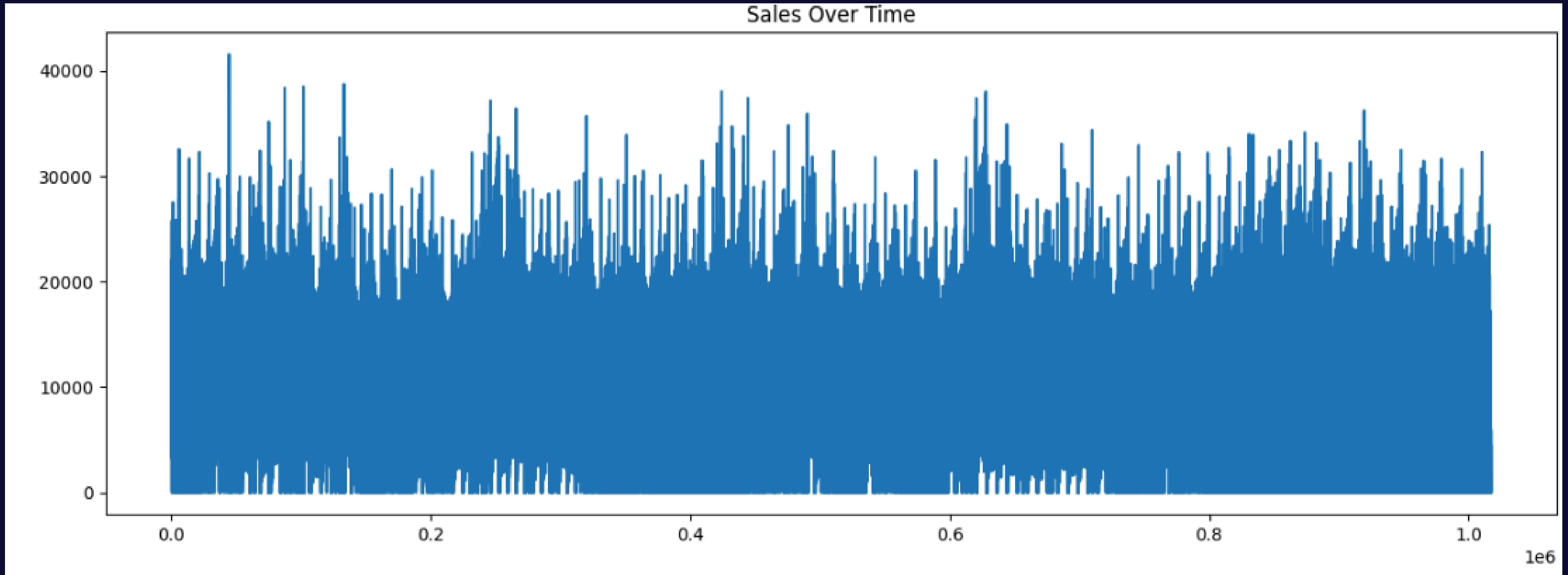
Integration:

- Merged store.csv with train.csv and test.csv with the Store as key.

Columns removed:

- Customers: Not available in the test set; could bias the model.
- Promo2SinceYear, CompetitionOpenSinceMonth: Replaced with engineered features.
- Date: Used for extracting time-based features but not directly relevant for training.

Some visualization and correlations results




```
Sales                  1.000000
Open                   0.678472
Promo                  0.452345
Promo2Open             0.091083
SchoolHoliday          0.085124
WeekOfYear             0.052946
Month                  0.048768
DayOfYear              0.047503
Year                   0.023519
Store                  0.005126
CompetitionOpen       -0.002709
Day                   -0.011612
CompetitionDistance   -0.018869
Promo2                -0.091040
DayOfWeek             -0.462125
Name: Sales, dtype: float64
```

Modelling

With the processed data. I am choosing to work with Feedforward Neural Network (FNN), also known as a Multilayer Perceptron (MLP) for modelling and doing the predictions since after researching I found out that:

- This model is designed specifically for structured/tabular data.
- Neural networks can model complex, non-linear relationships between features and the target variable.
- It works well with large datasets and can handle many features.
- Modern techniques can be applied as: adding dropout, early stopping, and scaling ensures better performance and generalization.

Which suits well with Rossmann Sales proyect, since:

- It has structured/tabular data like the features.
- The dataset is large enough for a neural network to learn meaningful patterns.
- The problem involves complex relationships between features and sales, which FNNs can model effectively.

This model is a Deep Learning model because it has multiple layers, is built using a Deep Learning framework (TensorFlow/Keras), and is designed to learn complex patterns in the data.

FNN : Architecture Overview

- Input Layer: Accepts the preprocessed features, like temporal variables (Year, Month) and store metrics (CompetitionOpen, Promo2Open), ensuring all key factors influencing sales are included.

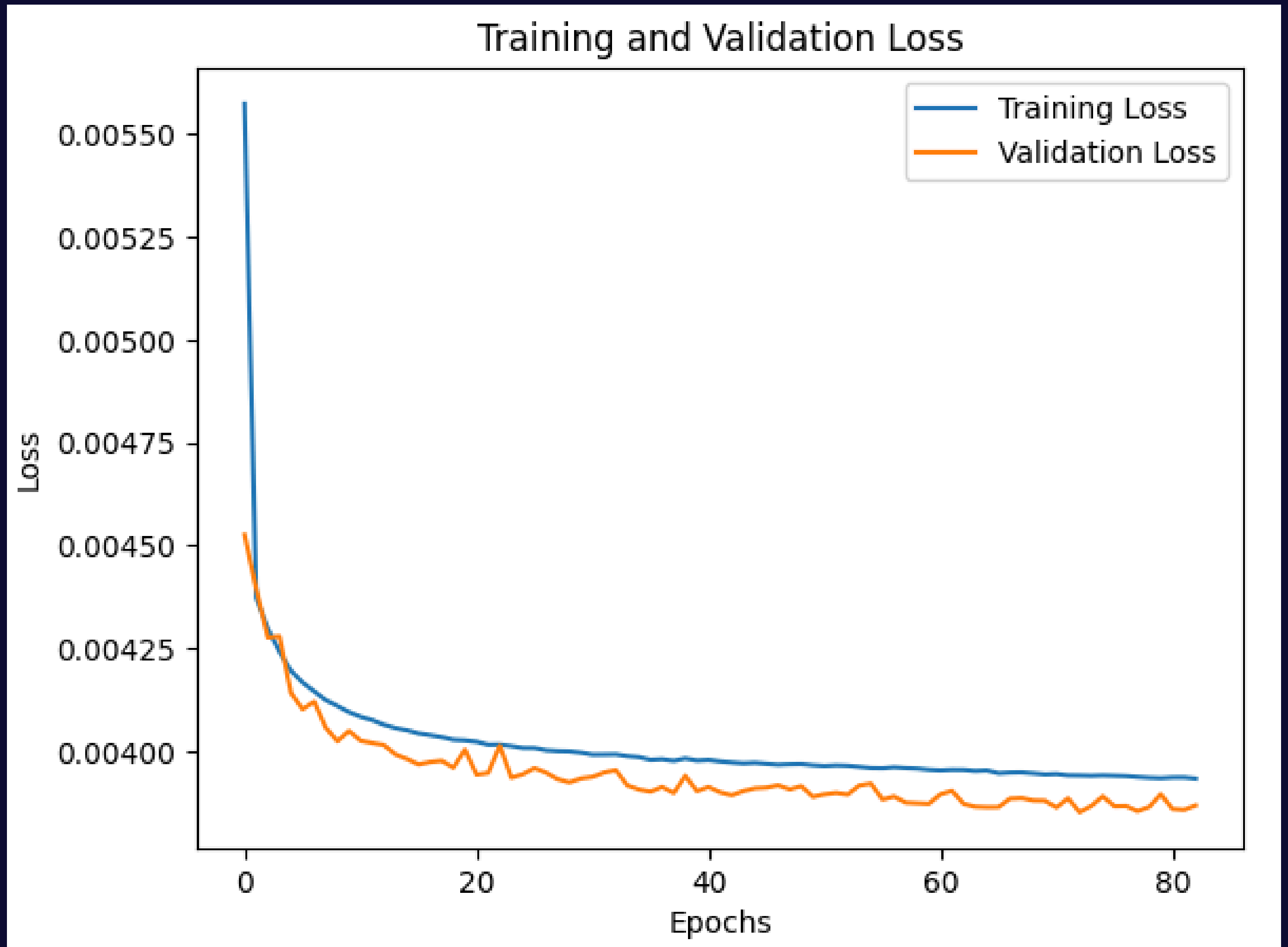
- Hidden Layers:
  - Structure: Three dense layers with decreasing nodes (128, 64, 32).
  - Activation: ReLU for capturing non-linear patterns in sales trends.
  - Regularization: Dropout layers to prevent overfitting by randomly deactivating nodes during training.
- Output Layer: A single neuron with no activation to predict continuous sales values.
  - Optimization:
    - Loss Function: Mean Squared Error (MSE) to minimize large prediction errors.
    - Optimizer: Adam, for efficient learning and adaptive adjustments to the model's parameters. A popular optimization algorithm that adjusts the learning rate during training.
- Hyperparameters:
  - Epochs: The model is trained for a maximum of 100 epochs, with early stopping implemented to prevent overfitting if the validation loss stops improving.
  - Batch Size: Set to 32, meaning the model processes 32 samples at a time before updating weights. This balances training efficiency and gradient stability.
  - Callbacks: Early stopping is included to monitor validation loss and stop training early if no improvement is observed, optimizing training time.

RESULTS

```
Epoch 83/100
25431/25431 ━━━━━━━━━━━━━━━━━━━━ 40s 2ms/step - loss: 0.0039 - mse: 0.0039 - val_loss: 0.0039 - val_mse: 0.0039
```

```python
# Evaluating the model

val_loss, val_mse = nn_model.evaluate(X_val_scaled, y_val_scaled, verbose=0)
val_rmse_scaled = val_mse ** 0.5
print(f"Validation RMSE (scaled): {val_rmse_scaled:.2f}")

# Converting scaled MSE back to RMSE in the original scale
# Scaler's `data_max_` and `data_min_` to calculate the range
data_range = target_scaler.data_max_[0] - target_scaler.data_min_[0]  # Extract scalar values
val_rmse = (val_mse ** 0.5) * data_range  # Scale back RMSE
print(f"Validation RMSE (original scale): {val_rmse:.2f}")
```

```
Validation RMSE (scaled): 0.06
Validation RMSE (original scale): 2388.28
```




```python
# Scaling the test data

test_preprocessed = pd.read_csv("test_preprocessed.csv")
X_test_scaled = scaler.transform(test_preprocessed[features])

# Predicting sales on test data
test_predictions_scaled = nn_model.predict(X_test_scaled)

# Scaling predictions back to original range
test_predictions = target_scaler.inverse_transform(test_predictions_scaled)

# Preparing the output file
submission = pd.DataFrame({
    'Id': test_preprocessed['Id'],
    'Sales': test_predictions.flatten()
})
submission.to_csv("Sales_Prediction.csv", index=False) # The final output
print("Sales Prediction file has been created")
```

I create a file with the predictions and I upload it to Kaggle which gives me the result:

Submission and Description	Private Score	Public Score	Selected
Sales_Prediction.csv Complete (after deadline) · 2d ago · This csv files contains the prediction of 6 weeks of daily sales for the stores in ...	**0.40065**	**0.40049**	☐

Thank You

Diego Armando Salinas Lugo

ds24353