Tecnológico de Monterrey

Campus Querétaro

Implementación de una técnica de aprendizaje máquina sin el uso de un framework

Gamaliel Marines Olvera
A01708746

Advanced AI

23 de Agosto del 2024

Introduction

In this document, you will find the necessary information about the problem proposed in the challenge "*Implementación de una técnica de aprendizaje máquina sin el uso de un framework*." My name is Gamaliel Marines, and through this document, I provide the context regarding the dataset, implementation, and the issue to be addressed in this challenge. I also explain the thought process, algorithms, and mathematics that bring the program to life and that will ultimately lead to solving the challenge

Context

For this challenge, I chose the following dataset - “Metro Interstate Traffic Volume” from UC Irvine. This dataset contains 8 features and 48,204 instances. This means that the dataset has 8 columns and 48,204 rows. In this activity, I analyze how the features or inputs influence the model and how they relate to the predictions, which will be compared to the labels, or actual results. This activity will focus on the area of AI known as Machine Learning, specifically in the branch of supervised learning using the linear regression method.

Documentation

Logic

Once I understand the dataset and its nature with the help of the documentation, the next step is to analyze the variables and determine which variables are useful and relevant for the model. This means I need to clean the dataset to avoid having "garbage" data that could negatively affect the model's training.

By studying the dataset, I conclude that "traffic_volume" is the label or output of the model, as it is the actual data against which the predictions are compared.

For the model, I use at least one of the features present in the dataset. In this case, the most obvious choice is temperature, as it is a numerical value. I can also use cloud coverage, given the nature of the data type.

Features that are discarded due to the absence of values are "holiday" and "snow_h1."

On my first approach to the dataset, I instantly discarded variables such as "weather_main" and "weather_description because they are not numerical values and therefore could not be used for the model. However, using ***One-Hot Encode***, their nature could be changed for each category and have new features of numerical nature.

In order to prevent having outliers that negatively affect the model, instead of deleting the rows containing outliers, I use **Imputation**, which helps the model to have a more accurate prediction by imputing the “distracting” values of the outliers.

The step where I search, select, and download the dataset corresponds to the data *extraction* phase of the process known as **ETL** (Extraction, Transformation, and Load). The part where I “clean” the dataset and pre-process the data, choose the columns which I work with, discard those which I do not use, and clean the rows corresponds to the *transformation* phase. Once these first two phases are completed, the next step is the *load* phase, which involves implementing the machine learning algorithm and loading the data into the model.

After preparing the dataset, I choose a model adequate for the prediction I am trying to accomplish. In this case a ***linear regression model***. Before training the model with the dataset, I split the dataset into a *training set* and a *test set* and use **scaling** for faster training. This part corresponds to the *Load Phase.* Once the training has been done, I compare the results of the training set against the test set resulting in the training error and test error. With that data I can interpret the model and its results.

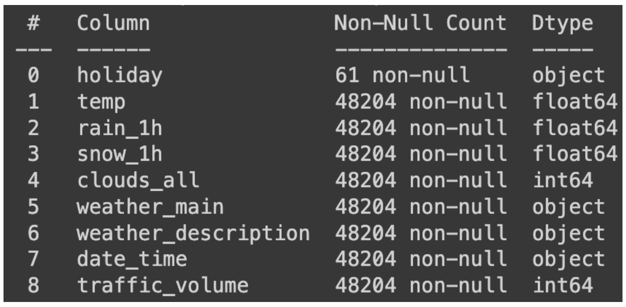
Algorithms and Mathematical Methods

As I mentioned before, I propose **One-Hot Encoding** and use **Imputation** and **Scaling**. Other interesting tools that I make use of are **Mean Square Error**, **Gradient descent**, **R-square**, **Training Error**, and **Test Error.** Following is the reasoning and my arguments on why I use each one of them:

1. **One-Hot Encoding**

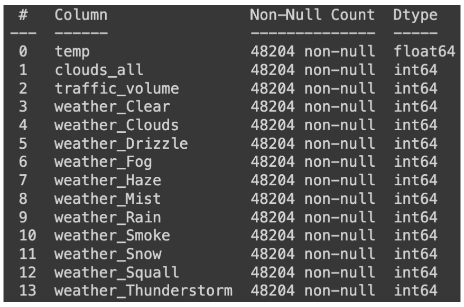
It converts *categorical* variables into a form that can be provided to machine learning algorithms to make predictions, numerical values. In this case, since features like `weather_main` and `weather_description` are categorical, they cannot be directly used in a linear regression model, which requires numerical input. By applying One-Hot Encoding, I would have transformed these categorical features into a series of binary (0 or 1) columns, each representing a different category.

This method is helpful because without converting these categorical variables into a numerical format, the model could not use the information contained within these features.



```
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   holiday              61 non-null     object
 1   temp                 48204 non-null  float64
 2   rain_1h              48204 non-null  float64
 3   snow_1h              48204 non-null  float64
 4   clouds_all           48204 non-null  int64
 5   weather_main         48204 non-null  object
 6   weather_description  48204 non-null  object
 7   date_time            48204 non-null  object
 8   traffic_volume       48204 non-null  int64
```

before



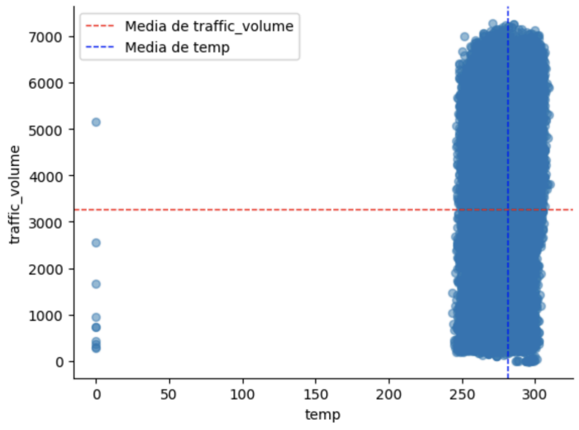
```
 #   Column                 Non-Null Count  Dtype
---  ------                 --------------  -----
 0   temp                   48204 non-null  float64
 1   clouds_all             48204 non-null  int64
 2   traffic_volume         48204 non-null  int64
 3   weather_Clear          48204 non-null  int64
 4   weather_Clouds         48204 non-null  int64
 5   weather_Drizzle        48204 non-null  int64
 6   weather_Fog            48204 non-null  int64
 7   weather_Haze           48204 non-null  int64
 8   weather_Mist           48204 non-null  int64
 9   weather_Rain           48204 non-null  int64
 10  weather_Smoke          48204 non-null  int64
 11  weather_Snow           48204 non-null  int64
 12  weather_Squall         48204 non-null  int64
 13  weather_Thunderstorm   48204 non-null  int64
```

after

2. **Imputation**

It replaces inconsistent data with substituted values to maintain the integrity of the dataset. During data cleaning, instead of discarding rows with missing or outlier values, I use imputation to “fix” these disruptions. This helps in ensuring that the model is trained on a complete dataset, without the noise or distractions caused by missing or erroneous data.

Removing rows with missing values leads to a significant reduction in the size of the dataset, thereby losing potentially valuable information. By imputing the missing data, I maintained the dataset's size and variability, which is important for the robustness of the model.



As shown in the graph, there are some outliers in the dataset which make no sense (the temperature is expressed in kelvin, meaning they would have reached the absolute zero). This is why imputation is necessary. These outliers could confuse the model and cause inaccurate predictions.

For temp (temperature): The variance is low. The points are very concentrated around the higher values, indicating little variability in this variable.

For traffic_volume: The variance is medium. There is greater dispersion of the points along the vertical axis, suggesting moderate variability in the traffic volume data.

3. **Scaling**

Scaling is the process of normalizing the range of independent variables (features of data). In this case, I use scaling to ensure that all the features contribute equally to the model and that the gradient descent algorithm used for training can converge more quickly and reliably. Without scaling, features with larger ranges would dominate the model, leading to skewed results.

Scaling speeds up the convergence of the gradient descent algorithm, leading to faster training and more accurate predictions. It also helped prevent the model from being biased toward features with larger numerical values.

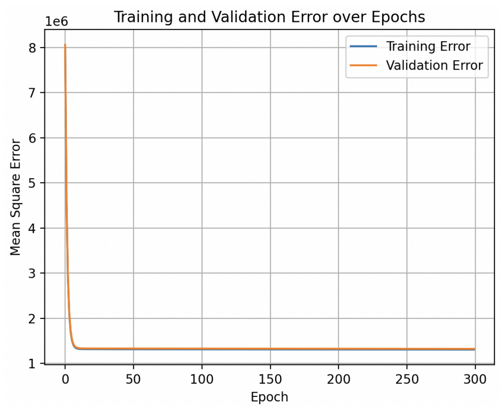
In this case, I chose mean scaling over min-max scaling for a few important reasons:

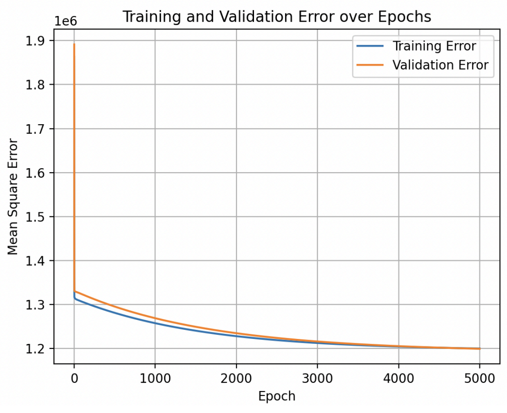
1. Mean scaling standardizes the features to have a mean of zero and a standard deviation of one, which aligns well with the assumptions of linear regression, ensuring that each feature contributes equally to the model.
2. Mean scaling is less sensitive to outliers compared to min-max scaling. Min-max scaling compresses the entire range of data to a fixed range (usually 0 to 1), which can cause extreme values (outliers) to disproportionately influence the scaled data. Mean scaling, on the other hand, standardizes the data by subtracting the mean and dividing by the standard deviation, which mitigates the impact of outliers.

4. **Mean Square Error** (MSE):

Mean Square Error (MSE) is a measure of the quality of an estimator—it is the average of the squares of the errors between the predicted and actual values. In this project, MSE was used to evaluate the performance of the linear regression model by comparing the predictions with the actual \`traffic_volume\` values.

Lower MSE values indicate better model performance, making it an effective tool for evaluating and improving the model.



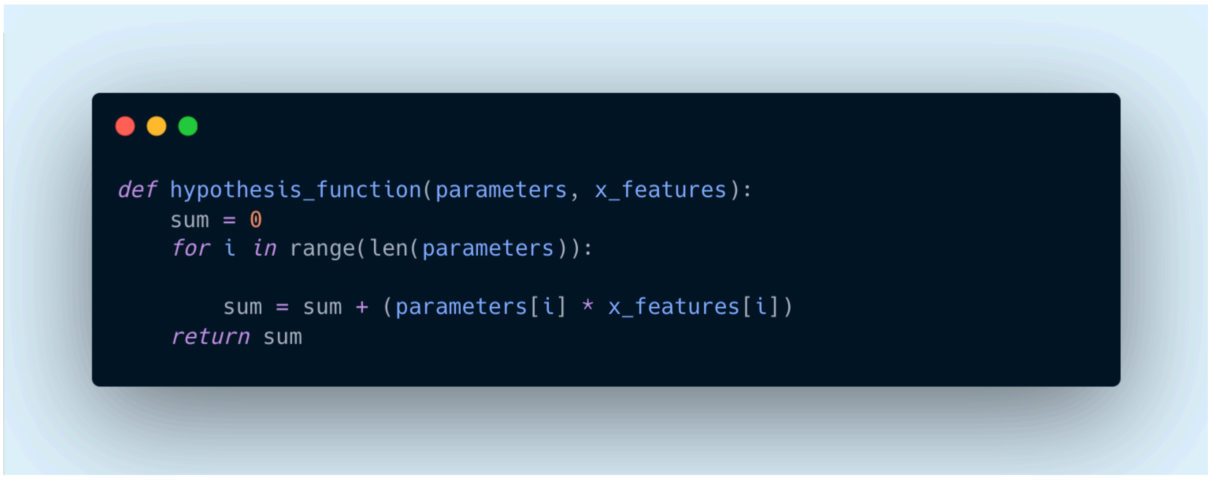


5. **Gradient Descent**:

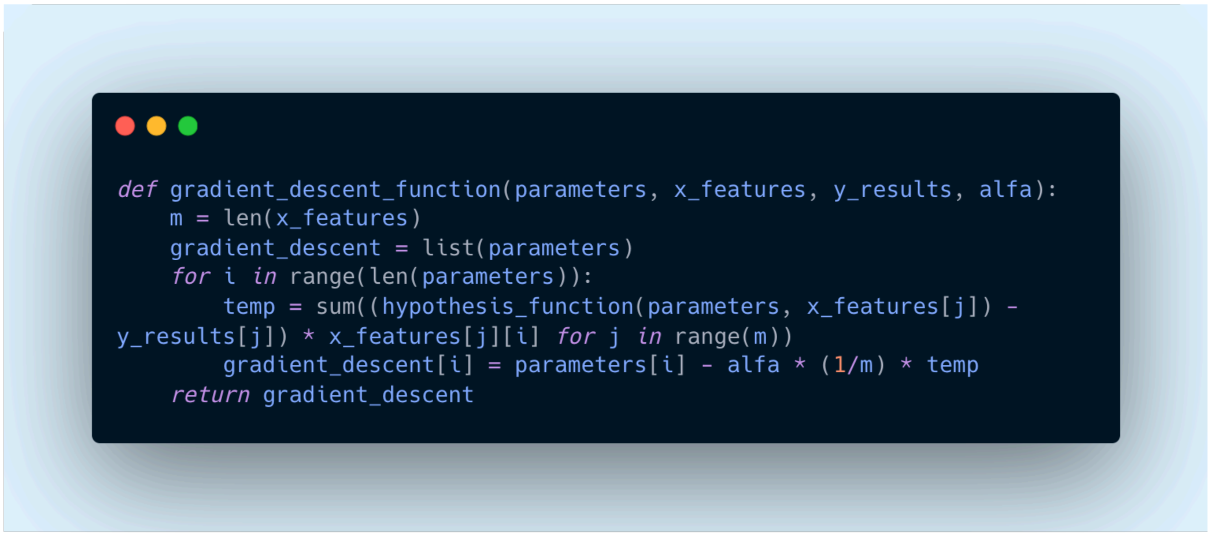
Gradient Descent is an optimization algorithm used to minimize the cost function in linear regression models. In this case, I applied gradient descent to iteratively adjust the model's parameters to reduce the difference between predicted and actual traffic volumes, thereby minimizing the MSE.

Gradient Descent was crucial for finding the optimal parameters for the linear regression model. Without it, the model might not have converged to the best possible solution, leading to less accurate predictions. This method allowed for efficient training, even with a relatively large dataset.

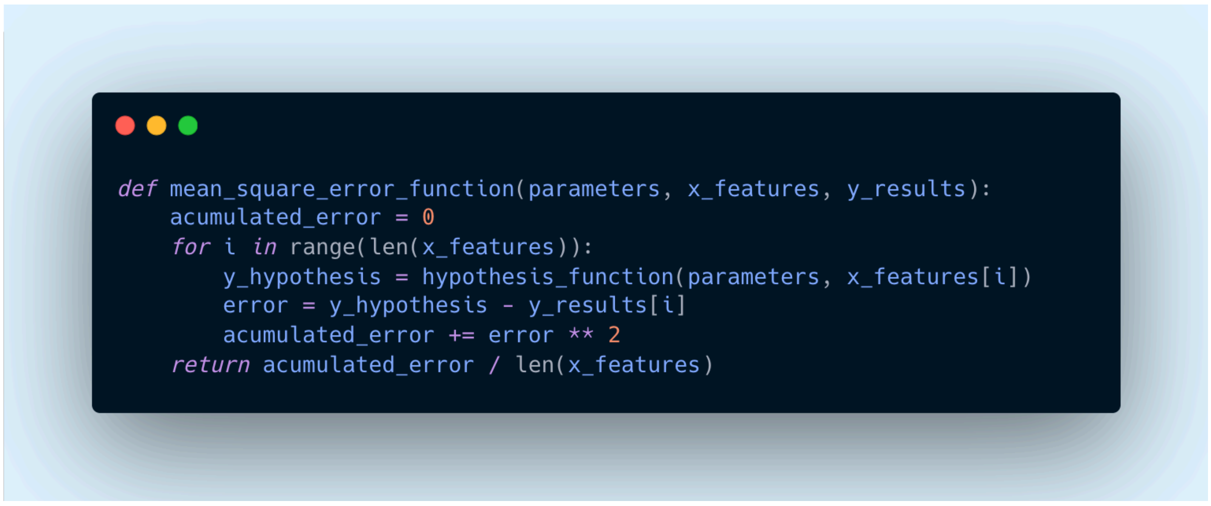
Code



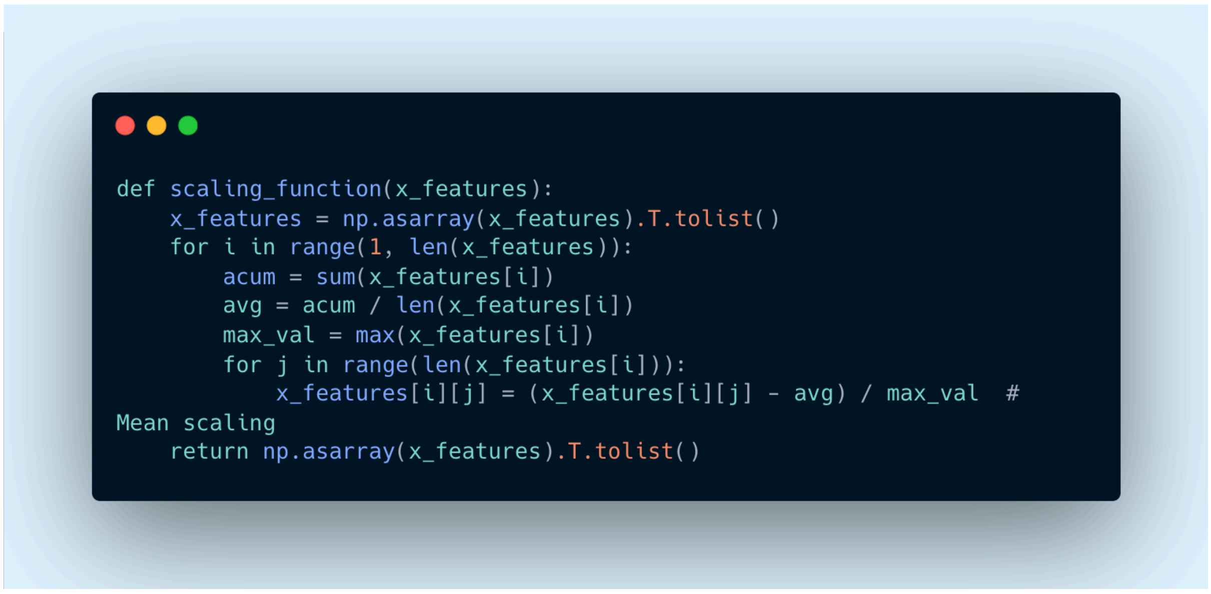
```python
def hypothesis_function(parameters, x_features):
    sum = 0
    for i in range(len(parameters)):

        sum = sum + (parameters[i] * x_features[i])
    return sum
```



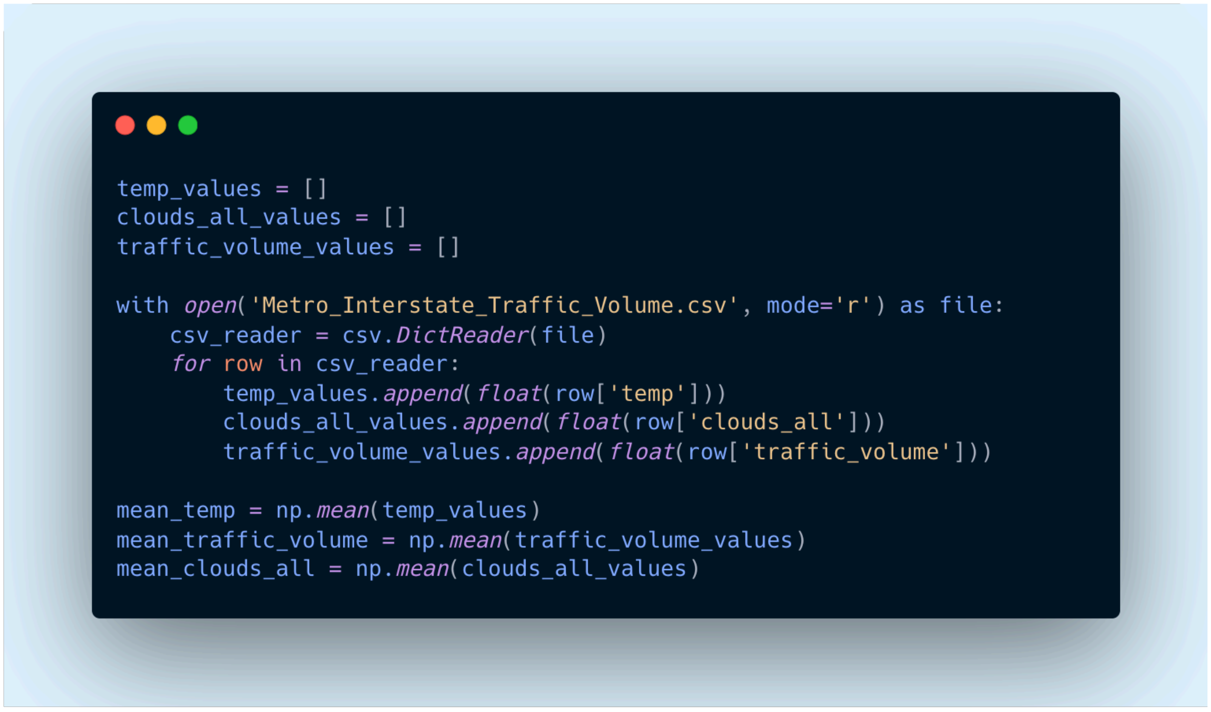
```python
def gradient_descent_function(parameters, x_features, y_results, alfa):
    m = len(x_features)
    gradient_descent = list(parameters)
    for i in range(len(parameters)):
        temp = sum((hypothesis_function(parameters, x_features[j]) -
y_results[j]) * x_features[j][i] for j in range(m))
        gradient_descent[i] = parameters[i] - alfa * (1/m) * temp
    return gradient_descent
```



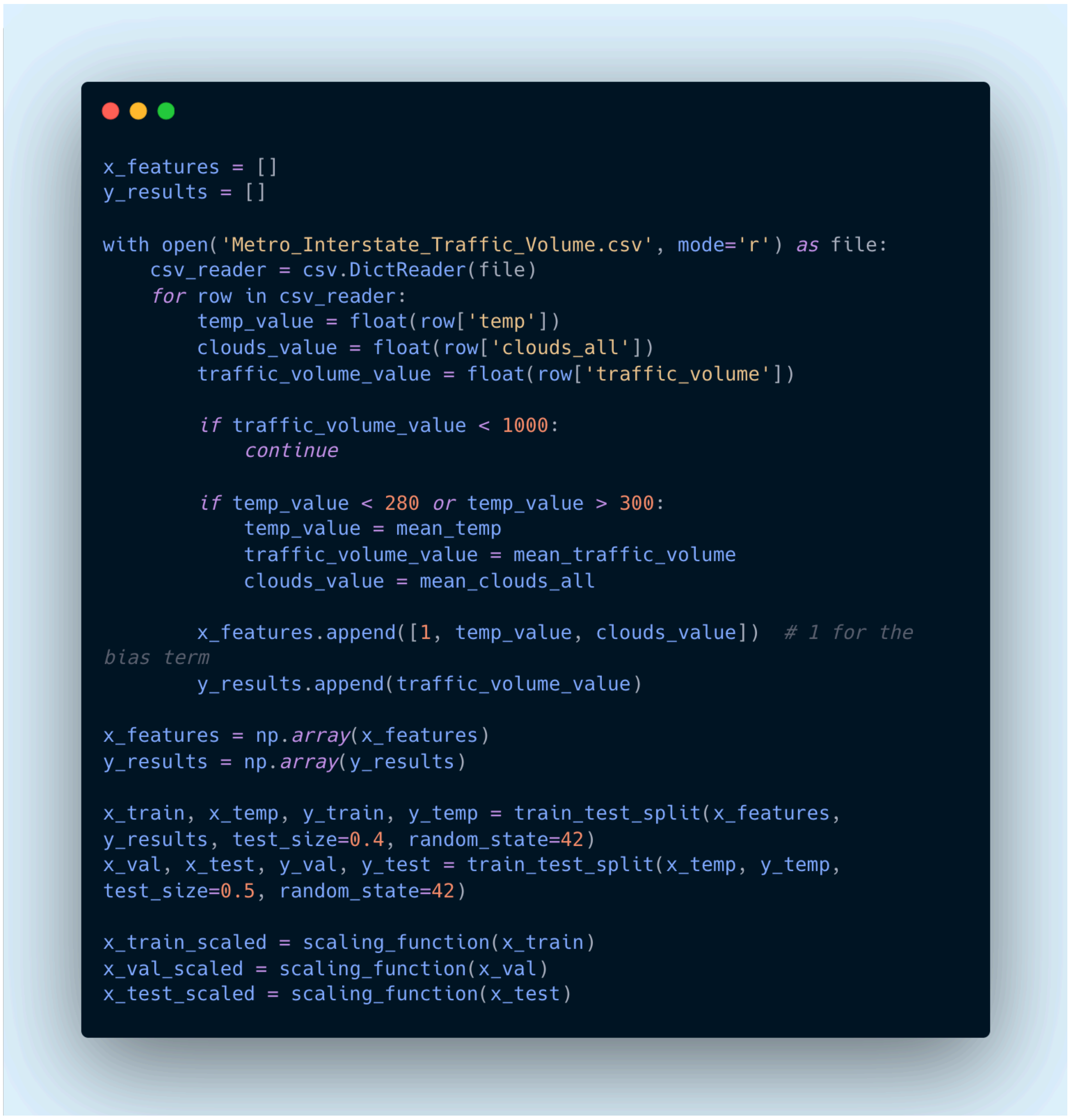
```python
def mean_square_error_function(parameters, x_features, y_results):
    acumulated_error = 0
    for i in range(len(x_features)):
        y_hypothesis = hypothesis_function(parameters, x_features[i])
        error = y_hypothesis - y_results[i]
        acumulated_error += error ** 2
    return acumulated_error / len(x_features)
```



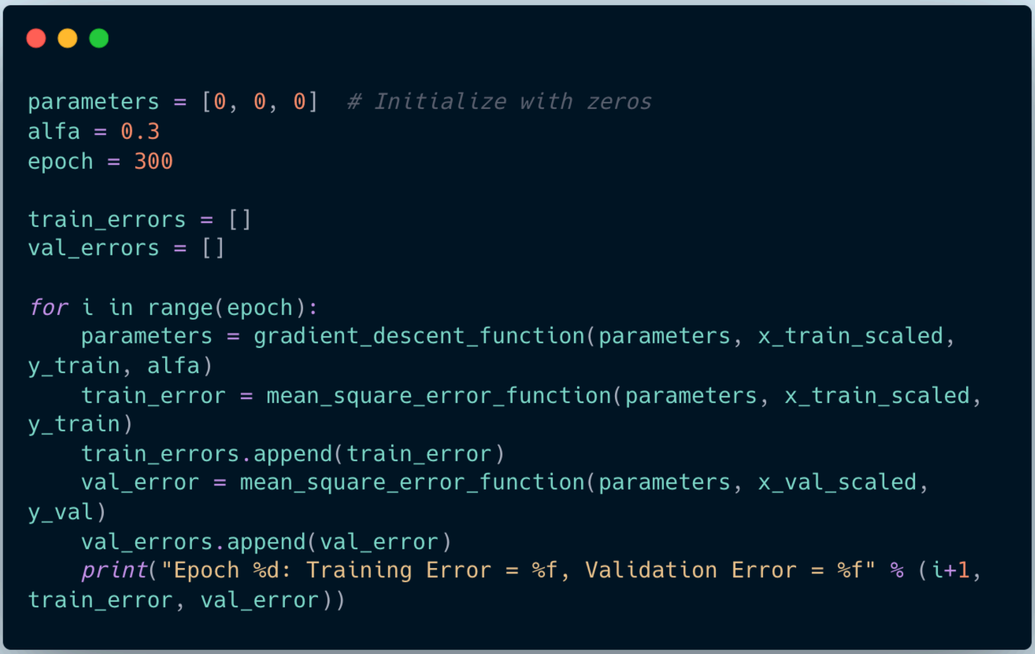
```
def scaling_function(x_features):
    x_features = np.asarray(x_features).T.tolist()
    for i in range(1, len(x_features)):
        acum = sum(x_features[i])
        avg = acum / len(x_features[i])
        max_val = max(x_features[i])
        for j in range(len(x_features[i])):
            x_features[i][j] = (x_features[i][j] - avg) / max_val  #
Mean scaling
    return np.asarray(x_features).T.tolist()
```



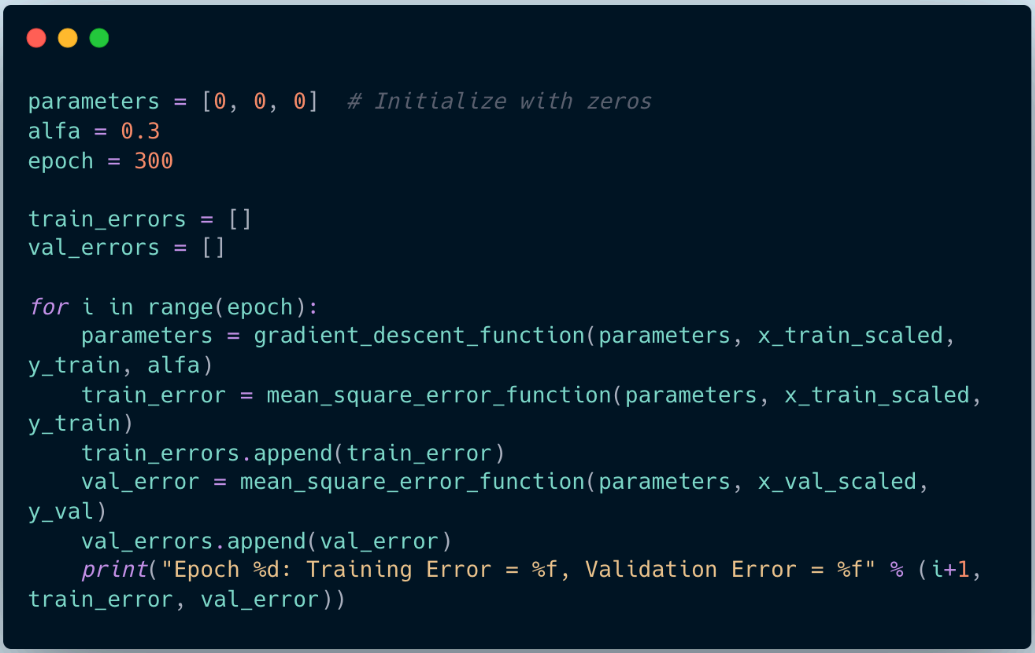
```
temp_values = []
clouds_all_values = []
traffic_volume_values = []

with open('Metro_Interstate_Traffic_Volume.csv', mode='r') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
        temp_values.append( float(row['temp']))
        clouds_all_values.append( float(row['clouds_all']))
        traffic_volume_values.append( float(row['traffic_volume']))

mean_temp = np.mean(temp_values)
mean_traffic_volume = np.mean(traffic_volume_values)
mean_clouds_all = np.mean(clouds_all_values)
```



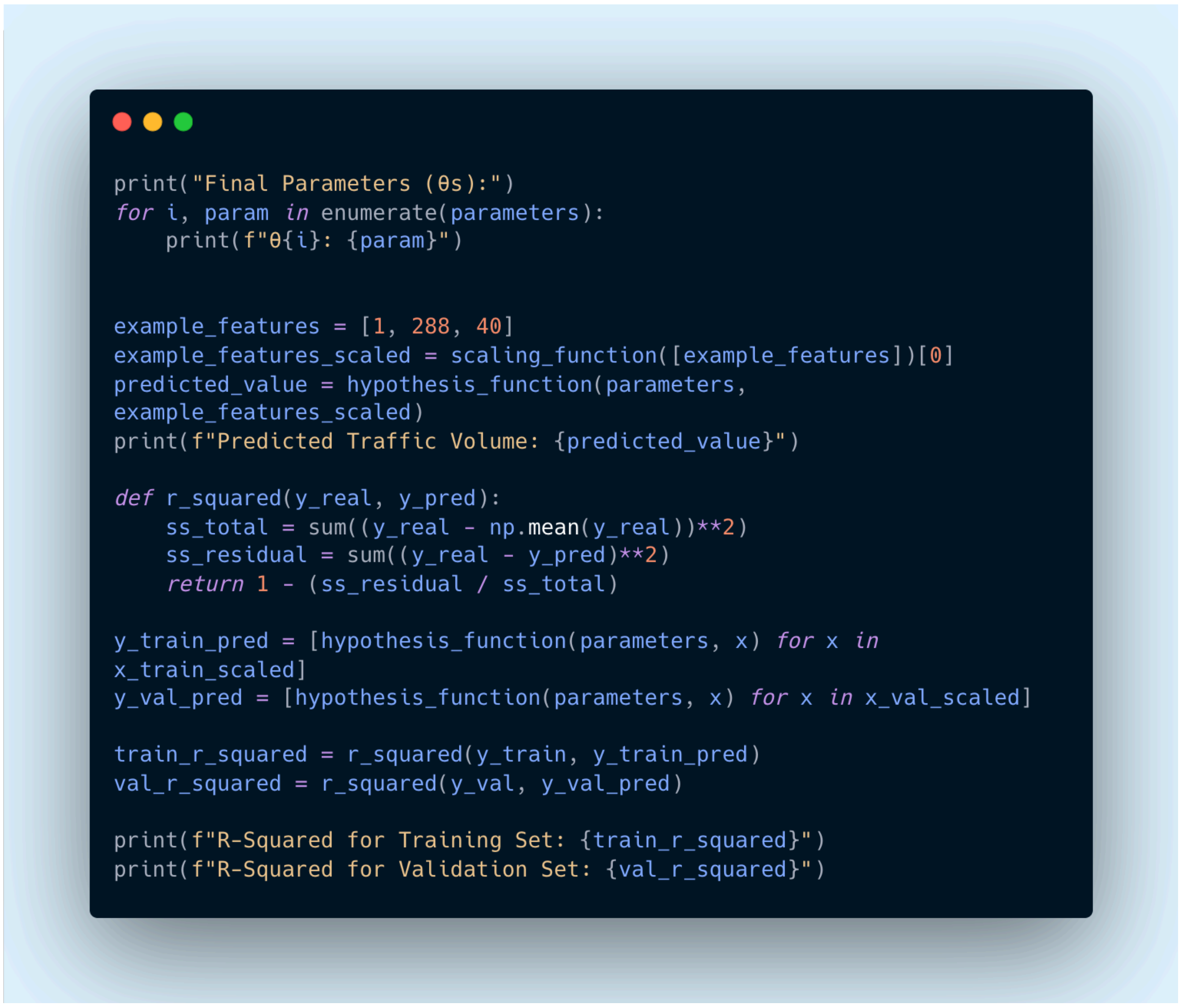
```python
x_features = []
y_results = []

with open('Metro_Interstate_Traffic_Volume.csv', mode='r') as file:
    csv_reader = csv.DictReader(file)
    for row in csv_reader:
        temp_value = float(row['temp'])
        clouds_value = float(row['clouds_all'])
        traffic_volume_value = float(row['traffic_volume'])

        if traffic_volume_value < 1000:
            continue

        if temp_value < 280 or temp_value > 300:
            temp_value = mean_temp
            traffic_volume_value = mean_traffic_volume
            clouds_value = mean_clouds_all

        x_features.append([1, temp_value, clouds_value])  # 1 for the
bias term
        y_results.append(traffic_volume_value)

x_features = np.array(x_features)
y_results = np.array(y_results)

x_train, x_temp, y_train, y_temp = train_test_split(x_features,
y_results, test_size=0.4, random_state=42)
x_val, x_test, y_val, y_test = train_test_split(x_temp, y_temp,
test_size=0.5, random_state=42)

x_train_scaled = scaling_function(x_train)
x_val_scaled = scaling_function(x_val)
x_test_scaled = scaling_function(x_test)
```



```
parameters = [0, 0, 0]  # Initialize with zeros
alfa = 0.3
epoch = 300

train_errors = []
val_errors = []

for i in range(epoch):
    parameters = gradient_descent_function(parameters, x_train_scaled,
y_train, alfa)
    train_error = mean_square_error_function(parameters, x_train_scaled,
y_train)
    train_errors.append(train_error)
    val_error = mean_square_error_function(parameters, x_val_scaled,
y_val)
    val_errors.append(val_error)
    print("Epoch %d: Training Error = %f, Validation Error = %f" % (i+1,
train_error, val_error))
```

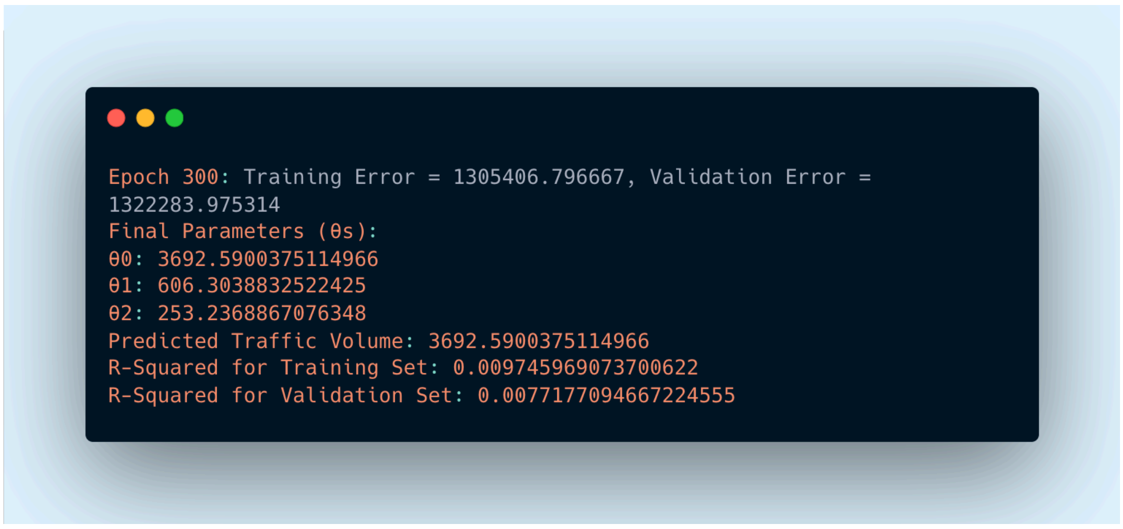


```
parameters = [0, 0, 0]  # Initialize with zeros
alfa = 0.3
epoch = 300

train_errors = []
val_errors = []

for i in range(epoch):
    parameters = gradient_descent_function(parameters, x_train_scaled,
y_train, alfa)
    train_error = mean_square_error_function(parameters, x_train_scaled,
y_train)
    train_errors.append(train_error)
    val_error = mean_square_error_function(parameters, x_val_scaled,
y_val)
    val_errors.append(val_error)
    print("Epoch %d: Training Error = %f, Validation Error = %f" % (i+1,
train_error, val_error))
```

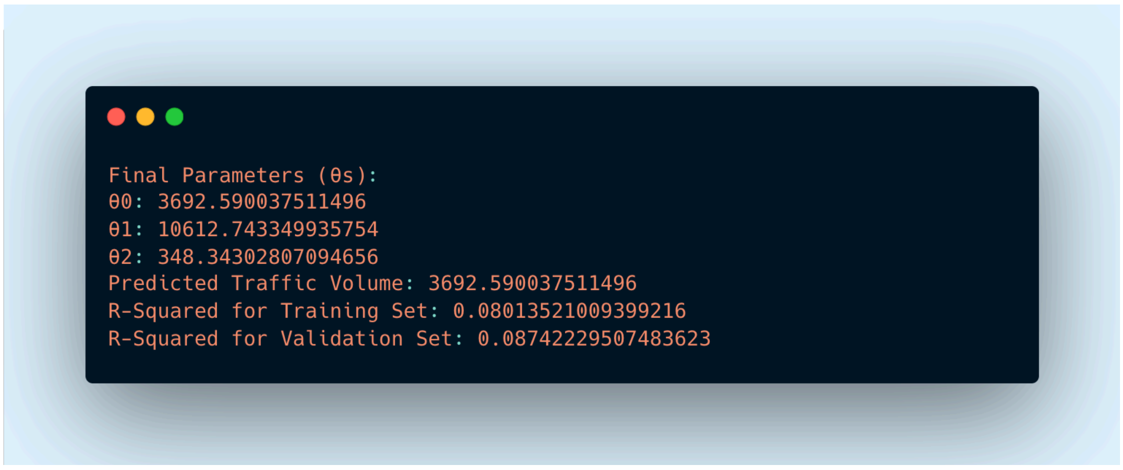


```
print("Final Parameters (θs):")
for i, param in enumerate(parameters):
    print(f"θ{i}: {param}")


example_features = [1, 288, 40]
example_features_scaled = scaling_function([example_features])[0]
predicted_value = hypothesis_function(parameters,
example_features_scaled)
print(f"Predicted Traffic Volume: {predicted_value}")

def r_squared(y_real, y_pred):
    ss_total = sum((y_real - np.mean(y_real))**2)
    ss_residual = sum((y_real - y_pred)**2)
    return 1 - (ss_residual / ss_total)

y_train_pred = [hypothesis_function(parameters, x) for x in
x_train_scaled]
y_val_pred = [hypothesis_function(parameters, x) for x in x_val_scaled]

train_r_squared = r_squared(y_train, y_train_pred)
val_r_squared = r_squared(y_val, y_val_pred)

print(f"R-Squared for Training Set: {train_r_squared}")
print(f"R-Squared for Validation Set: {val_r_squared}")
```

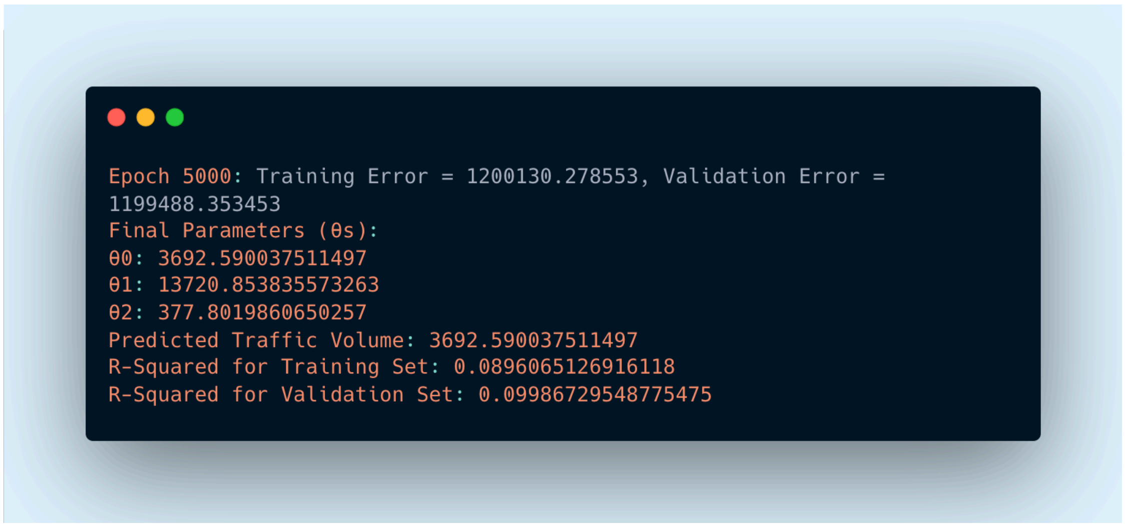
Result



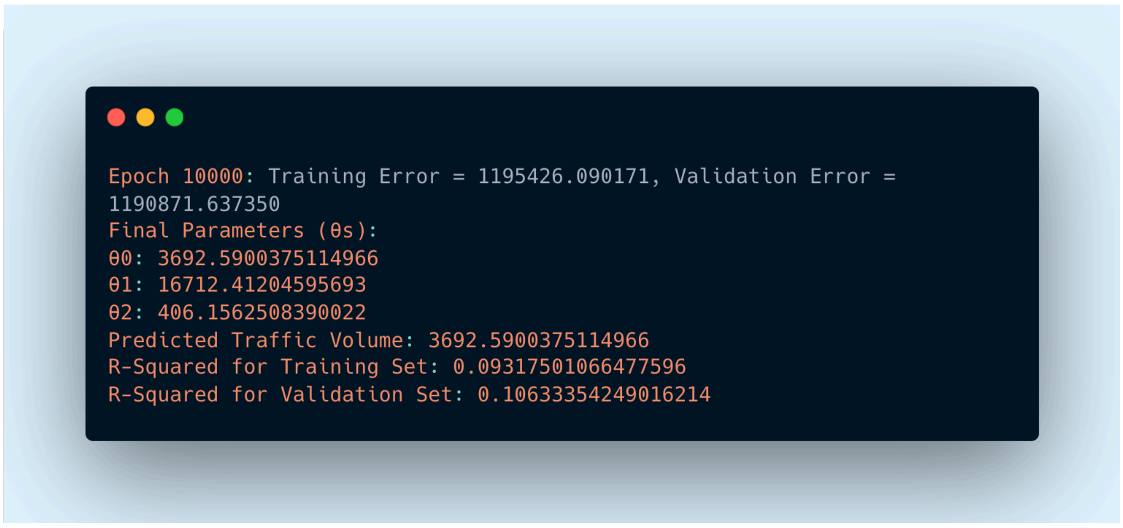
```
Epoch 300: Training Error = 1305406.796667, Validation Error =
1322283.975314
Final Parameters (θs):
θ0: 3692.5900375114966
θ1: 606.3038832522425
θ2: 253.2368867076348
Predicted Traffic Volume: 3692.5900375114966
R-Squared for Training Set: 0.009745969073700622
R-Squared for Validation Set: 0.0077177094667224555
```



```
Final Parameters (θs):
θ0: 3692.590037511496
θ1: 10612.743349935754
θ2: 348.34302807094656
Predicted Traffic Volume: 3692.590037511496
R-Squared for Training Set: 0.080135210093992l6
R-Squared for Validation Set: 0.08742229507483623
```



```
Epoch 5000: Training Error = 1200130.278553, Validation Error =
1199488.353453
Final Parameters (θs):
θ0: 3692.590037511497
θ1: 13720.853835573263
θ2: 377.8019860650257
Predicted Traffic Volume: 3692.590037511497
R-Squared for Training Set: 0.0896065126916118
R-Squared for Validation Set: 0.09986729548775475
```



```
Epoch 10000: Training Error = 1195426.090171, Validation Error =
1190871.637350
Final Parameters (θs):
θ0: 3692.5900375114966
θ1: 16712.41204595693
θ2: 406.1562508390022
Predicted Traffic Volume: 3692.5900375114966
R-Squared for Training Set: 0.09317501066477596
R-Squared for Validation Set: 0.10633354249016214
```

I also wanted to check how it would work when using frameworks: Implementación de una técnica de aprendizaje máquina sin el uso de un framework_…

Inside that program, at the end are interpretations about the results obtained using frameworks.

Interpretation

Model Performance Over Epochs

- **Training and Validation Errors**: Mean Square Error (MSE) for both the training and validation datasets over 5000 epochs, both errors decrease consistently as the number of epochs increases, indicating that the model is learning from the data. However, even at epoch 5000, the errors are still quite high (approximately 1.2 million for both training and validation), which suggests that the model is not fitting the data well.

R-Squared Values

- **R-Squared for Training and Validation Sets**:
    - **Training Set**: R2=0.0896
    - **Validation Set**: R2=0.0999
- These R-squared values are very low, indicating that the model explains only a small percentage of the variance in the traffic volume data. This further reinforces the conclusion that the linear regression model is underfitted.

- **Underfitting**: The model appears to be underfitting the data, as indicated by the high errors and low R-squared values. Despite the reduction in error over time, the model does not capture the underlying patterns in the data effectively, leading to poor predictive performance.

Conclusion

Considering all the factors, I have concluded that the problem is not merely a *skill issue*… Jokes aside, and understanding that this is an academic paper, I recognize that the problem requires a more powerful model than just a linear regression. Before delving into the details and reasoning behind this statement, I want to highlight that the model is severely underfitted. Here's why: both the validation and training errors are significantly large, which is a clear indication of underfitting. If the errors had shown a low training error and a high test error, that would suggest overfitting. Additionally, we can observe that the R-squared value, which measures the accuracy of the model, increases significantly as the epochs progress, indicating that the model needs more training.

I sustain my statement that a simple linear regression model is insufficient for this dataset because I had to rely heavily on imputation and ignore valuable variables that could have provided more context for prediction. Furthermore, comparing the errors and accuracy achieved by the model built without a framework against those of a model using frameworks supports the argument that the problem is not a *skill issue*... or maybe it is. The program needs to run for many more epochs to achieve higher accuracy, and based on those results, I might need to change my previous statements.

Improvements

Given the current performance, it may be necessary to explore more complex models, introduce additional relevant features, or preprocess the data differently to improve the model's accuracy and fit to the data. The current linear regression model, as configured, is insufficient for making reliable predictions on this dataset.

References

https://stackoverflow.com/questions/51237635/difference-between-standard-scaler-and-minmaxscaler

https://dataheadhunters.com/academy/scaling-and-normalization-preparing-data-for-analysis/

https://www.kaggle.com/code/alexisbcook/scaling-and-normalization