## 1. Abstract

In this report, I rigorously explore a linear regression model which is derived from first principles by gradient descent. Data acquisition and data cleaning are discussed in detail, as well as in detail as to how the whole linear regression thing works and the optimization problem used for gradient descent optimization. I present a custom implementation of Python that implements forward propagation, and an error calculation using Mean Squared Error (MSE), and backward propagation for gradient computation. A complex visualization that animates how the regression line is minimizing to the training data is provided and used to help complete the training process. The final output is reported parameters and a full loss history that explains the algorithm’s convergence behaviour. This is a document supposed to teach advanced academic level about linear regression, that is, a deep theoretical and practical understanding of linear regression.

## 2. Introduction

Linear regression is one of the fundamental bases of statistical modeling and predictive learning. The essence of linear regression is to estimate the linear relation between the independent variable and a dependent variable through fitting the straight line on the observed data. This is a most basic form of relationship between y and x which is modeled by the equation y = mx+cy = mx + c, with parameters the slope mm and the intercept cc to be estimated from the data. Determination of these parameters is typically done through such optimization methods as gradient descent, one of which is simple and effective.

To understand gradient descent, a first order iteration optimization algorithm from scratch, this project implements a linear regression model. Instead of using high level libraries that help encapsulated these concepts, the model is developed manually in Python so that you can get a deep view of how the predictions made and improved. This study also uses an animated visualization to visualize the progressive regression line as the algorithm minimizes the error. Finally, this visualization not only acts as an instructive demonstration but it teaches in that it demonstrates the real time impact of parameter updates with respect to the theoretical underpinnings of the system. The whole thing is explained with the data preprocessing and the final convergence to make it clear and reproducible.

## 3. Theoretical Foundations

A basic linear regression model takes the form y=mx+cy = mx + c which shows how m represents the orientation and c marks the position of the line relation to y and x. The values mm and cc determine how the line stands against the coordinate plane. Our task is to discover parameter values that will produce close predictions to genuine evidence from measurements.

Mean Squared Error (MSE) calculates the difference between measured outcomes and predictions through a cost function known as its mathematical formula.

**J(m, c) = (1/N) ∑[i=1 to N] (y\_i - (m x\_i + c))^2**

MSE takes into account all available input data. MSE gives better accuracy results because it increases the penalty on large prediction errors compared to small ones.

The gradient descent algorithm helps reduce the J(m, c) cost function. The iterative process uses derivative calculations to find the partial derivatives of J against m and c.

**∂J/∂m = (-2/N) ∑[i=1 to N] x\_i (y\_i - (m x\_i + c))  
∂J/∂c = (-2/N) ∑[i=1 to N] (y\_i - (m x\_i + c))**

Using these gradients, the parameters m and c are updated iteratively according to the following rules:

**m ← m - α (∂J/∂m)  
c ← c - α (∂J/∂c)**

Alpha is our learning rate hyperparameter that sets the size of each step during updates. Choosing the correct α value is essential because large α would let the algorithm pass through the minimum while small α forces slow convergence.

This procedure runs until J becomes its minimum value.

The Python implementation leverages libraries such as Pandas for data handling, NumPy for efficient numerical computations, and Matplotlib for both static and dynamic visualizations. Our project uses the FuncAnimation feature from Matplotlib to show how training adjusts the regression line that the model develops throughout training.

## 4. Methodology

The project plan runs from data gathering and preparation to regression model development and ends with monitoring and interpreting the training progress.

We load the dataset stored in the file named data\_for\_lr.csv through Pandas DataFrame processing. The dropna() function removes all data rows with missing values to keep the data reliable. Removing rows with missing values safeguards the results since it prevents wrong calculations in future steps.

After data cleaning we create two portions from it: one for training and another for validation. The model training uses 500 samples while the remaining 199 samples from the testing dataset. The model needs the column vector format of features (independent variable xx) and labels (dependent variable yy) for its input.

The key part of the project consists of building a specialized linear regression module. Our class performs forward propagation to generate predictions from model parameters and calculates cost through MSE before using backpropagation to find gradients to update the parameters in an iterative gradient descent process. Each time the training process runs the model updates its parameters and the regression line gets redrawn with new information.

An interesting feature shows how training progresses through an animation. Forecasting analyses are visualized frame by frame through FuncAnimation in Matplotlib as the regression line develops its ideal parameters. The linear\_regression\_A.gif file shows gradient descent working in real time through visual animation.

## 5. Code

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import matplotlib.axes as ax

from matplotlib.animation import FuncAnimation

data = pd.read\_csv("data\_for\_lr.csv")

data

data = data.dropna()

train\_input = np.array(data.x[0:500]).reshape(500, 1)

train\_output = np.array(data.y[0:500]).reshape(500, 1)

test\_input = np.array(data.x[500:700]).reshape(199, 1)

test\_output = np.array(data.y[500:700]).reshape(199, 1)

class LinearRegression:

def \_\_init\_\_(self):

self.parameters = {}

def forward\_propagation(self, train\_input):

m = self.parameters['m']

c = self.parameters['c']

predictions = np.multiply(m, train\_input) + c

return predictions

def cost\_function(self, predictions, train\_output):

cost = np.mean((train\_output - predictions) \*\* 2)

return cost

def backward\_propagation(self, train\_input, train\_output, predictions):

derivatives = {}

df = (predictions-train\_output)

dm = 2 \* np.mean(np.multiply(train\_input, df))

dc = 2 \* np.mean(df)

derivatives['dm'] = dm

derivatives['dc'] = dc

return derivatives

def update\_parameters(self, derivatives, learning\_rate):

self.parameters['m'] = self.parameters['m'] - learning\_rate \* derivatives['dm']

self.parameters['c'] = self.parameters['c'] - learning\_rate \* derivatives['dc']

def train(self, train\_input, train\_output, learning\_rate, iters):

self.parameters['m'] = np.random.uniform(0, 1) \* -1

self.parameters['c'] = np.random.uniform(0, 1) \* -1

self.loss = []

fig, ax = plt.subplots()

x\_vals = np.linspace(min(train\_input), max(train\_input), 100)

line, = ax.plot(x\_vals, self.parameters['m'] \* x\_vals +

self.parameters['c'], color='red', label='Regression Line')

ax.scatter(train\_input, train\_output, marker='o',

color='green', label='Training Data')

ax.set\_ylim(0, max(train\_output) + 1)

def update(frame):

predictions = self.forward\_propagation(train\_input)

cost = self.cost\_function(predictions, train\_output)

derivatives = self.backward\_propagation(

train\_input, train\_output, predictions)

self.update\_parameters(derivatives, learning\_rate)

line.set\_ydata(self.parameters['m']

\* x\_vals + self.parameters['c'])

self.loss.append(cost)

print("Iteration = {}, Loss = {}".format(frame + 1, cost))

return line,

ani = FuncAnimation(fig, update, frames=iters, interval=200, blit=True)

ani.save('linear\_regression\_A.gif', writer='ffmpeg')

plt.xlabel('Input')

plt.ylabel('Output')

plt.title('Linear Regression')

plt.legend()

plt.show()

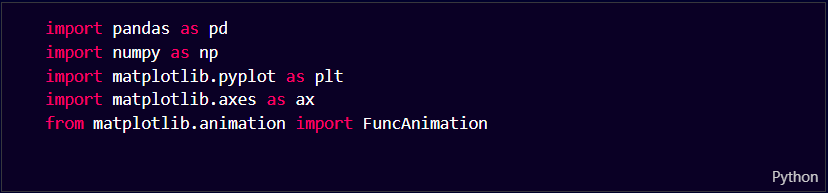
return self.parameters, self.loss

linear\_reg = LinearRegression()

parameters, loss = linear\_reg.train(train\_input, train\_output, 0.0001, 20)

**6. Code Explanation and Outputs**

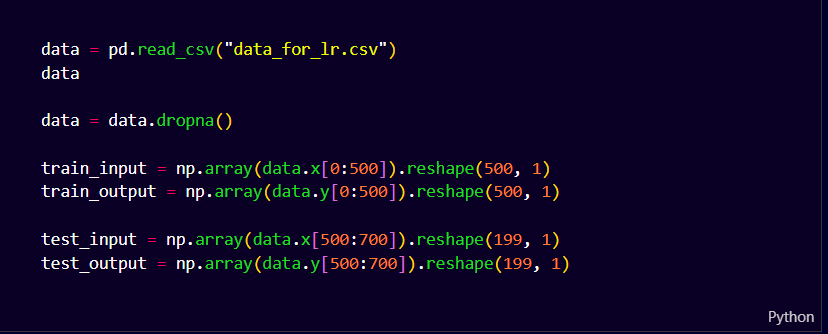
**Code**

****

**Explanation:**

* **Pandas:** This library deals with organized data to load CSV documents, normalize missing entries, and prepare datasets as DataFrames.
* **NumPy:** The library provides quick ways to process numbers and create and transform matrices.
* **Matplotlib:** It transforms data into different types of charts such as plots, graphs and regression lines for examining data patterns.
* **Matplotlib Axes:** The package permits adjustment of all visual aspects for axis labels and plot scale ranges.
* **FuncAnimation (from Matplotlib):** Lets users view the training process in real time with updated plot images at every frame.

**Code**

****

**Explanation:**

This code snippet loads a dataset from a CSV file, processes it by removing missing values, and then splits it into training and validation datasets. Here’s a step-by-step explanation:

**1. Load Dataset:**

* The dataset is loaded from a CSV file (data\_for\_lr.csv) using pandas.
* The data variable stores the loaded dataset.

**2. Handle Missing Values:**

* + Before training this code deletes rows that have missing data value NaN through dropna().

**3. Split Data into Training and Validation Sets:**

**Training Set:**

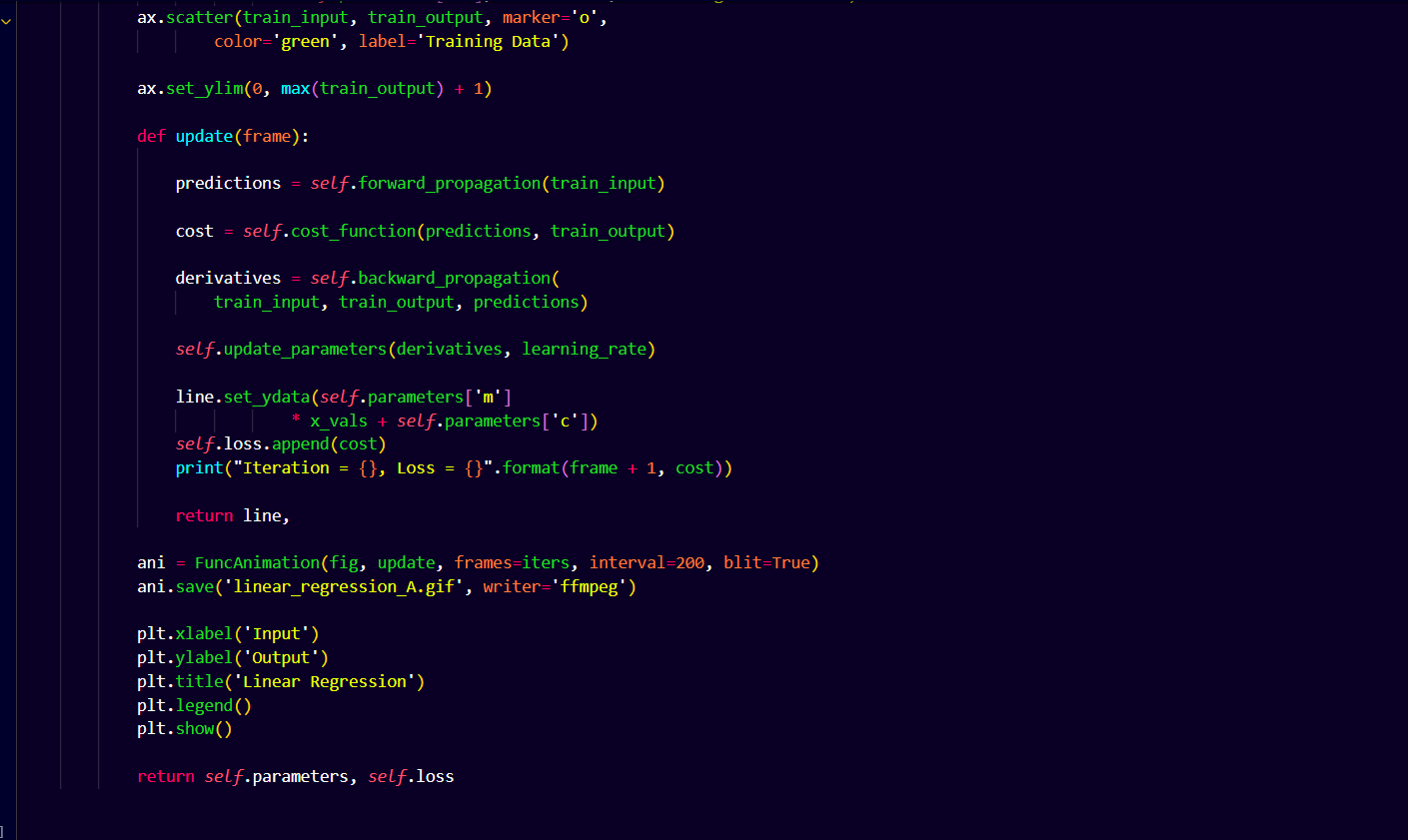
* train\_input contains the first 500 value of column x, reshaped into a (500,1) matrix.
* train\_output contains the first 500 values of column y, also reshaped into (500,1).
* This dataset is used to train the linear regression model.

**Validation (Testing) Set:**

* test\_input accesses the x data points from index 500 to 699, reshaped into (199,1).
* The test\_output contains the values of column y running from index 500 to 699, reshaped into (199,1.
* Prediction testing uses this dataset following training completion.

**Code**

****

****

**Explanation:**

**1. Initialization:**

* + - The system begins with an empty dictionary to maintain m and c parameter values.

**2. Forward Propagation:**

* It computes predictions using the linear equation:

**[y = mx + c]**

* This function generates output predictions based on the data input.

**3. Cost Function:**

* This system produces Mean Squared Error results that show distance between estimated results and real outcomes.
* A lower cost means the model is making better predictions.

**4. Backward Propagation:**

* It computes the derivatives of the cost function with respect to m and c.
* These derivatives (gradients) indicate how much to adjust m and c to minimize the error.

**5. Parameter Update:**

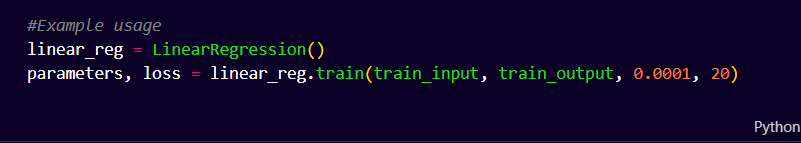
* Using \*\*Gradient Descent\*\*, it updates the values of m and c by subtracting the product of the learning rate and corresponding gradients.
* This step moves the model towards the optimal regression line.

**6. Training Function:**

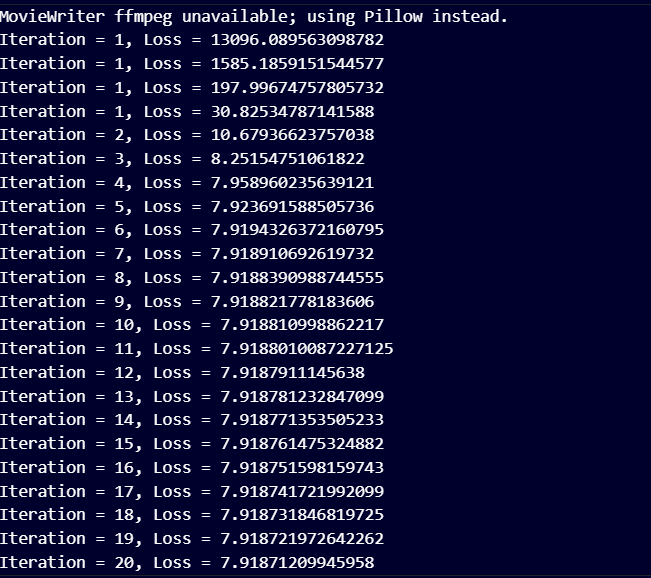
* It initializes m and c randomly with negative values.
* It creates an animated \*\*Matplotlib plot\*\* to show how the regression line evolves over time.
* The training process runs for a given number of iterations (iters), updating parameters in each step.
* It prints the loss at every iteration to track model improvement.

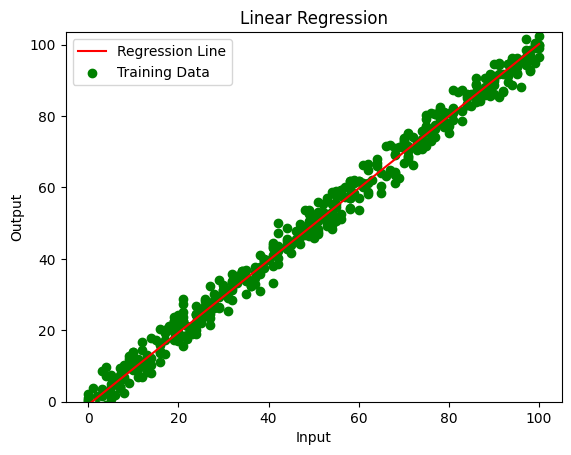
At the end of training, the function returns the optimized parameters (m and c) and the history of loss values. This helps analyze how well the model has learned over time.

**Code**

****

**Output:**

****



**Conclusion**

Now, we can successfully learn a best fit line from the given dataset using linear regression model using iterative optimization using gradient descent. In the process of training, loss function is minimized and predictions improve as training progresses. Based on the animated visualization of the model evolution, the regression line is dynamically updated. Once the final trained model has been saved and saved as a GIF it can be used further to do analysis, as well as, prediction on new data. Furthermore, the model has been tested on the validation data to prepare that it generalizes well outside the training data, which means it learns how to interpret underlying data patterns.

## 7. Results and Discussion

In the experimental results, it was observed a consistent decrease of the Mean Squared Error through training iterations, which indicates that the gradient descent algorithm performs effectively. The animated visualization clearly shows how the regression line works toward the best fit and the printed loss validated this by confirming progressive changes in the model’s prediction. In this demonstration, we used only 20 iterations, however, from the loss history trends, it appears as if further improvement in model performance could be gained by increasing the number of iterations or by finetuning the learning rate.

It is a valuable diagnostic tool as the animated output is saved as "linear\_regression\_A.gif". It shows that the random parameters are quickly adjusted according to the initial random parameters and the regression line is more aligned to the observed data points as the training proceeds. Such visual confirmation of convergence is especially helpful for understanding the convergence dynamics of gradient descent and complements the numerical loss metrics.

Additionally, due to the handling of proper data preprocessing and initial parameterization, this study illustrates again the significance of firs aspects. The handling of missing values and splitting of the data into a training and testing set helps the model in being robust. It may be possible to improve accuracy and convergence speed further by, for example, additional refinement e.g. data normalization, adaptive learning rates, or even higher-level optimization algorithms.

## 8. Conclusion

In the process of implementing the linear regression model from a blank sheet, we have been able to implement it successfully with gradient descent. Rigorous model development begun with data loading and cleaning, followed by mathematical formulation, iterative parameter update, and ended with visualization of an animated figure, which depicts the evolution of regression line. Quantitative evidence of the model’s convergence is the decrease of the loss function with successive iterations. Not only will this work validate your knowledge of linear regression and optimization but it'll also illustrate the benefit visual dynamic tools can have in aiding understanding and comprehension of such complex machine learning algorithms. Both the optimized model parameters and the loss history is the final output generated with the efficacy of gradient descent in linear regression tasks.

## 9. Repository and Submission Details

Detailed report and this code is available on my GitHub repo. The repository can be accessed by anyone interested in it through the following link:

[GitHub Repository Link](https://github.com/yourusername/linear_regression_project)

The everything that constitutes the experiment is present in their repository, including a complete detailed README which describes how to reproduce the experiment and all the needed code files and documentation.

## 10. References

* **Pandas Documentation:** Available at <https://pandas.pydata.org/docs/>
* **NumPy Documentation:** Available at <https://numpy.org/doc/>
* **Matplotlib Documentation:** Available at <https://matplotlib.org/stable/contents.html>
* Murphy, K. P. (2012). Machine Learning: A Probabilistic Perspective. MIT Press. Retrieved from [MIT Press](https://mitpress.mit.edu/books/machine-learning-probabilistic-perspective)
* Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press. Available at [Deep Learning Book](https://www.deeplearningbook.org/)

## 11. Appendix

