Advanced Java and Advanced Python Assignment

Multi-Linear Regression

Cranfield University

Vu Ta Duong; Paul Fontanges

Prepared for 28 OCT, 2024

Abstract

This assignment explores the application of multi-linear regression, specifically focusing on gradient descent as a method for optimizing predictive models. Linear regression is a frequent approach for predicting a single real-valued variable based on multiple predictors. In this project, we implement a Python script for gradient descent to approximate the parameters $(\theta_0, \dots, \theta_r)$ in a multi-linear regression model. Using the mean square error as a cost function, we iteratively adjust our model to minimize the error, guided by the gradient, to identify the parameter values that most closely align with the observed data.

To test our model, we utilize two datasets : dataEnergy, predicting the energy output of a power plant based on ambient conditions, and dataLoans, predicting loan interest rates based on FICO scores and loan amounts. By experimenting with different learning rates and iteration counts, we examine convergence behavior and optimize our model's performance. We validate our results using the scikit-learn package's regression model, comparing outcomes to measure accuracy, and asses the model's effectiveness through the r-squared metric.

This work demonstrates the impact of training size on model performance, illustrating both the potential and limitations of linear regression in predictive modeling and emphasizing the value of gradient descent as a foundational optimization tool in machine learning.

Contents

0.1 Introduction

0.1.1 Linear regression introduction

Linear regression is a fundamental statistical technique used to model the relationship between a dependent variable (often called the response or target) and one or more independent variables (also known as predictors or features). The goal of linear regression is to find the best-fitting line that explains how changes in the independent variables affect the dependent variable. This is achieved by minimizing the difference (error) between the predicted values and the actual values in the dataset.[1]

0.1.2 Applications of Linear Regression

Linear regression is widely used across various domains for its simplicity and interpretability. Here are a few common applications[2]:

- Economics: It is used to model relationships such as how changes in price influence demand or how income levels impact consumer spending.
- Analyze risk: For instance insurance companies use linear regression to develop a model for estimating claim costs. This analysis support company leaders in making strategic decisions regarding risk management.
- Real Estate: It's commonly used to predict house prices based on factors like location, square footage, and the number of bedrooms.
- Marketing: Businesses use linear regression to assess how factors such as advertising spend, pricing, and market conditions influence sales.

0.1.3 Aim of this Assignment

The main objective here is to find the most accurate prediction model ($\hat{y}$) possible for a given set of a data ($x_0, \dots, x_r, y$). This means finding the following parameters ($\theta_0, \dots, \theta_r$) such as :

$$\hat{y} = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \ldots + \theta_r x_r \approx y$$

To find these parameters, we will calculate the minimum of the cost function by implementing and using the gradient descent method in a Python script.
First, we'll look at the methodology for dealing with this problem. Then we'll explain how the Python code works, step by step. We'll present the results we've obtained, analyze them and discuss them. Then we'll conclude.

0.2 Methodology

0.2.1 Data

We need to find a prediction model for the following two datasets.

dataEnergy set

Firstly, the "dataEnergy" dataset contains 9568 data points collected over a 6-year period (2006-2011) from a combined-cycle power plant operating at full load. The main objective is to predict the plant's net hourly electrical Energy Production (EP, in MW) as a function of four environmental variables measured every hour.

The four explanatory variables are:

- Ambient temperature (T): Outdoor ambient temperature measured in degrees Celsius, which influences the plant's thermodynamic performance.
- Ambient pressure (AP): Atmospheric pressure in millibars, which has an impact on air density and therefore on turbine efficiency.
- Relative humidity (RH): The percentage of humidity in the air, influencing steam cooling and, consequently, system efficiency.
- Exhaust gas vacuum (V): The vacuum created by the exhaust gases, reflecting the pressure level in the condenser, affecting the overall efficiency of the production cycle.

dataLoans set

This second dataset comes from a peer-to-peer lending platform where each user can lend money to others. This dataset includes a sample of 2,500 loans and contains various variables about them. Here, the ones of interest to us are the FICO score, the loan amount, and the interest rate. We will use the FICO Score (FS) and the Loan Amount (LA) to predict the Interest Rate (IR).

- FICO Score (FS): This credit score reflects the borrower's creditworthiness and can influence the interest rate applied to their loan.
- Loan Amount (LA): The total loan amount requested by the borrower, which may also be related to the risk level and therefore the applied interest rate.

However, a preliminary analysis needs to be conducted on this dataset. There are 5 column names for 6 columns of data. This indicates that one of the columns is unnecessary, and upon examining the "dataLoans.csv" file, it appears that the "first" column follows an incremental pattern. We can deduce that it likely represents the loan number, so we will remove it beforehand in our implementation.

Analyzing these datasets allows us to better understand them, establish specific relationships between certain parameters, and assess the impact of some of them on the output value, enabling us to predict it based on the input parameters.

0.2.2 Matrix approach

Here in our Python implementations, we will use a Matrix approach as shown below.

0.2.3 Cost Function and Mean Square Error

For a multi-linear function, the most appropriate cost function is the mean square error. Firstly, the cost function is a measure of how well the regression model fits the data. It is used to quantify the error between the values predicted by the model and the actual values observed. The aim of the regression is to find the parameters θ that minimize the cost function. The smaller the cost function, the closer the predicted values are to the real values y, indicating model accuracy.

The mean square error is defined by:

$$J(\theta) = \frac{1}{n}\sum_{i=1}^{n}\left(\hat{y}^{(i)} - y^{(i)}\right)^2 = \frac{1}{n}\sum_{i=1}^{n}\left(\theta_0 + \theta_1 x_1^{(i)} + \theta_2 x_2^{(i)} + \cdots + \theta_r x_r^{(i)} - y^{(i)}\right)^2$$

0.2.4 Gradient Descent Method

We use the gradient descent method as an optimization algorithm to minimize the cost function. It is based on an iterative calculation of the gradient. The gradient is a vector that indicates the direction in which the cost function increases most rapidly. It is calculated by taking the partial derivatives of the cost function with respect to each parameter θ. Parameters are adjusted by following the gradient in the opposite direction. Updating is done as follows:

$$\theta_j \leftarrow \theta_j - \eta \cdot \frac{\partial J(\theta)}{\partial \theta_j}$$

Here η is called the 'learning rate' parameter, which will be adjusted to find the best results.

0.2.5 $\mathbf{R^2}$ Method of Validation

After finding our prediction model, we check its credibility using the coefficient of determination (R^2). The R^2 is a statistical measure indicating how well a regression model fits the data. It quantifies the proportion of variance in the dependent variable explained by the independent variables[3]. The formula for R^2 is:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}\left(y^{(i)} - \hat{y}(x_1^{(i)}, \ldots, x_r^{(i)})\right)^2}{n \cdot \mathrm{var}(y)}$$

0.2.6 Scikit-learn Validation

After using the R^2 method, we employ the Scikit-learn library for a second validation of our model.

0.2.7 External Libraries

- $Numpy$: Used for numerical operations and handling arrays, essential for performing mathematical computations in Python.
- $Pandas$: A powerful library for handling data manipulation from many different data formats.
- $Matplotlib$: A plotting library to visualize data.
- $Scikit - learn$: A machine learning library used to implement and train machine learning models.
- $Seaborn$: Used for data visualization, providing a high-level interface for drawing attractive and informative statistical graphics.

0.3 Implementation

0.4 Results

0.4.1 dataEnergy Results

Once the script implementing the gradient descent method has been run on our "dataEnergy" dataset. We decide to run it for several different learning rate (η) values and for several different numbers of iterations.
Specifically, we tested the following learning rates: 0.1, 0.01, 0.001, and 0.0001, along with the following iteration counts: 10, 50, 100, 1000, and 10000. The results were as follows:

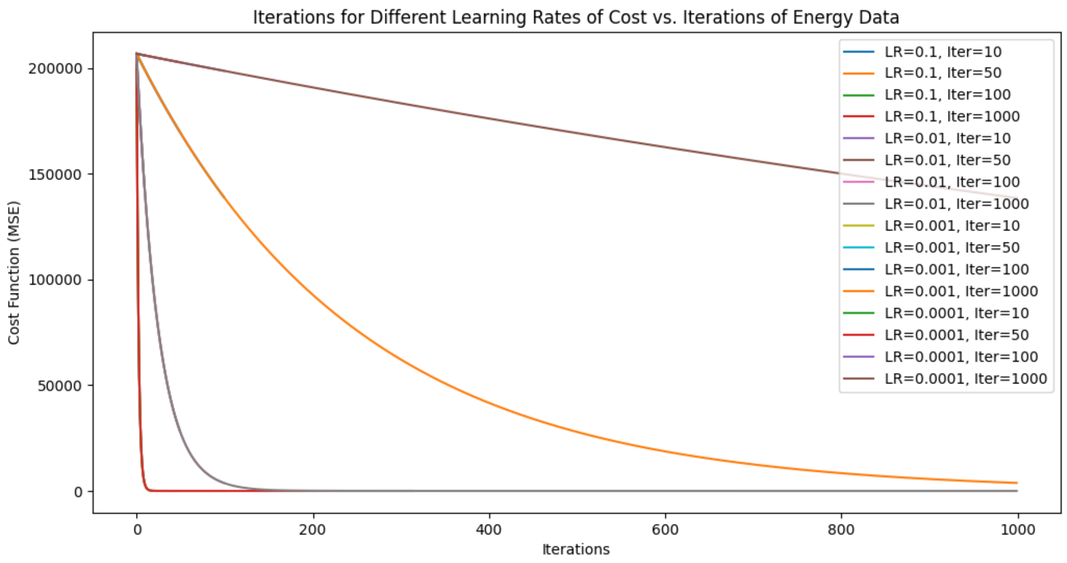
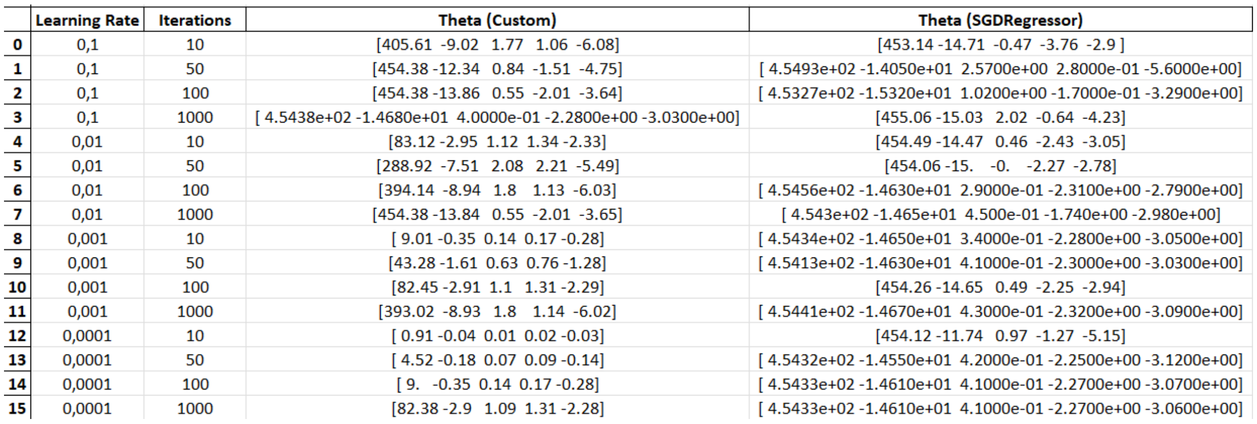


Figure 1: The convergence of the cost function for several learning rates and iteration counts

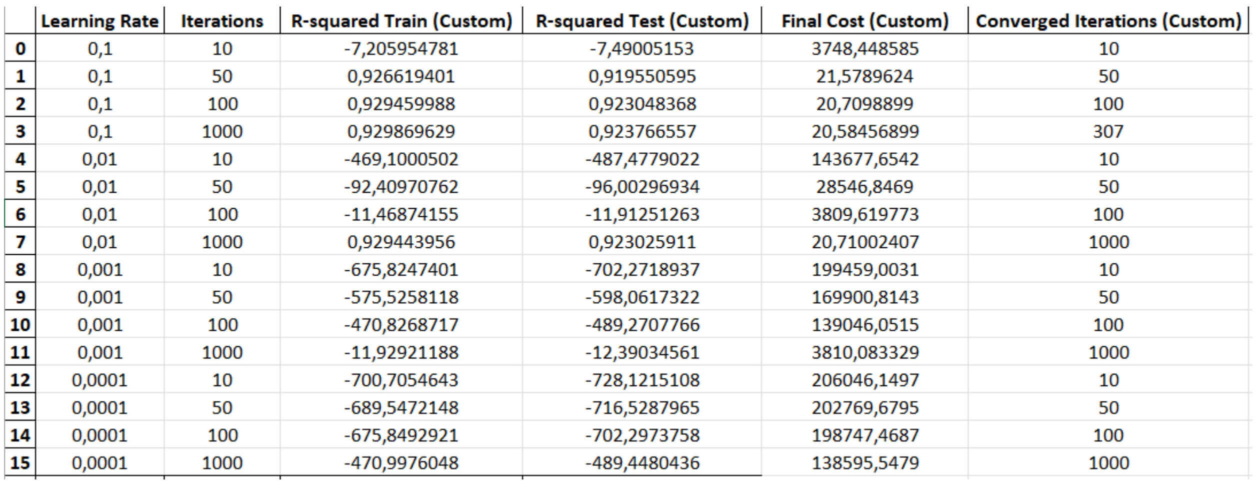
Here we have several values of theta based on different values of the learning rate and the number of iterations:



	Learning Rate	Iterations	Theta (Custom)	Theta (SGDRegressor)
0	0,1	10	[405.61 -9.02 1.77 1.06 -6.08]	[453.14 -14.71 -0.47 -3.76 -2.9]
1	0,1	50	[454.38 -12.34 0.84 -1.51 -4.75]	[4.5493e+02 -1.4050e+01 2.5700e+00 2.8000e-01 -5.6000e+00]
2	0,1	100	[454.38 -13.86 0.55 -2.01 -3.64]	[4.5327e+02 -1.5320e+01 1.0200e+00 -1.7000e-01 -3.2900e+00]
3	0,1	1000	[4.5438e+02 -1.4680e+01 4.0000e-01 -2.2800e+00 -3.0300e+00]	[455.06 -15.03 2.02 -0.64 -4.23]
4	0,01	10	[83.12 -2.95 1.12 1.34 -2.33]	[454.49 -14.47 0.46 -2.43 -3.05]
5	0,01	50	[288.92 -7.51 2.08 2.21 -5.49]	[454.06 -15. -0. -2.27 -2.78]
6	0,01	100	[394.14 -8.94 1.8 1.13 -6.03]	[4.5456e+02 -1.4630e+01 2.9000e-01 -2.3100e+00 -2.7900e+00]
7	0,01	1000	[454.38 -13.84 0.55 -2.01 -3.65]	[4.543e+02 -1.465e+01 4.500e-01 -1.740e+00 -2.980e+00]
8	0,001	10	[9.01 -0.35 0.14 0.17 -0.28]	[4.5434e+02 -1.4650e+01 3.4000e-01 -2.2800e+00 -3.0500e+00]
9	0,001	50	[43.28 -1.61 0.63 0.76 -1.28]	[4.5413e+02 -1.4630e+01 4.1000e-01 -2.3000e+00 -3.0300e+00]
10	0,001	100	[82.45 -2.91 1.1 1.31 -2.29]	[454.26 -14.65 0.49 -2.25 -2.94]
11	0,001	1000	[393.02 -8.93 1.8 1.14 -6.02]	[4.5441e+02 -1.4670e+01 4.3000e-01 -2.3200e+00 -3.0900e+00]
12	0,0001	10	[0.91 -0.04 0.01 0.02 -0.03]	[454.12 -11.74 0.97 -1.27 -5.15]
13	0,0001	50	[4.52 -0.18 0.07 0.09 -0.14]	[4.5432e+02 -1.4550e+01 4.2000e-01 -2.2500e+00 -3.1200e+00]
14	0,0001	100	[9. -0.35 0.14 0.17 -0.28]	[4.5433e+02 -1.4610e+01 4.1000e-01 -2.2700e+00 -3.0700e+00]
15	0,0001	1000	[82.38 -2.9 1.09 1.31 -2.28]	[4.5433e+02 -1.4610e+01 4.1000e-01 -2.2700e+00 -3.0600e+00]

Figure 2: Theta value's from our Custom method and from SGDRegressor method

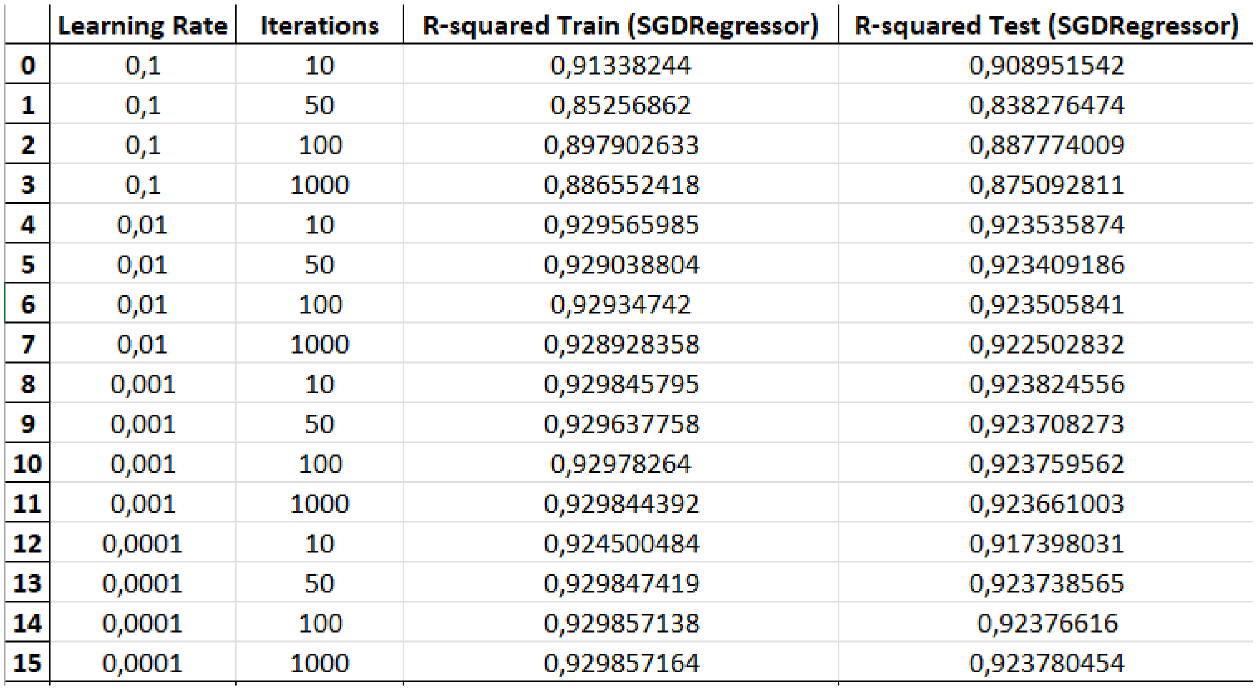
Here we have now R squared values from the same learning rate and number of iterations:



	Learning Rate	Iterations	R-squared Train (Custom)	R-squared Test (Custom)	Final Cost (Custom)	Converged Iterations (Custom)
0	0,1	10	-7,205954781	-7,49005153	3748,448585	10
1	0,1	50	0,926619401	0,919550595	21,5789624	50
2	0,1	100	0,929459988	0,923048368	20,7098899	100
3	0,1	1000	0,929869629	0,923766557	20,58456899	307
4	0,01	10	-469,1000502	-487,4779022	143677,6542	10
5	0,01	50	-92,40970762	-96,00296934	28546,8469	50
6	0,01	100	-11,46874155	-11,91251263	3809,619773	100
7	0,01	1000	0,929443956	0,923025911	20,71002407	1000
8	0,001	10	-675,8247401	-702,2718937	199459,0031	10
9	0,001	50	-575,5258118	-598,0617322	169900,8143	50
10	0,001	100	-470,8268717	-489,2707766	139046,0515	100
11	0,001	1000	-11,92921188	-12,39034561	3810,083329	1000
12	0,0001	10	-700,7054643	-728,1215108	206046,1497	10
13	0,0001	50	-689,5472148	-716,5287965	202769,6795	50
14	0,0001	100	-675,8492921	-702,2973758	198747,4687	100
15	0,0001	1000	-470,9976048	-489,4480436	138595,5479	1000

Figure 3: R-squared values from our Custom method, Final cost values and the convergence iteration number

And then the r-squared values from the SGDRegressor :



	Learning Rate	Iterations	R-squared Train (SGDRegressor)	R-squared Test (SGDRegressor)
0	0,1	10	0,91338244	0,908951542
1	0,1	50	0,85256862	0,838276474
2	0,1	100	0,897902633	0,887774009
3	0,1	1000	0,886552418	0,875092811
4	0,01	10	0,929565985	0,923535874
5	0,01	50	0,929038804	0,923409186
6	0,01	100	0,92934742	0,923505841
7	0,01	1000	0,928928358	0,922502832
8	0,001	10	0,929845795	0,923824556
9	0,001	50	0,929637758	0,923708273
10	0,001	100	0,92978264	0,923759562
11	0,001	1000	0,929844392	0,923661003
12	0,0001	10	0,924500484	0,917398031
13	0,0001	50	0,929847419	0,923738565
14	0,0001	100	0,929857138	0,92376616
15	0,0001	1000	0,929857164	0,923780454

Figure 4: R-squared values from the SGDRegressor method

0.4.2 dataLoan Results

Here, for the "dataLoans" dataset, we apply the same method as for the "dataEnergy" dataset. We will run our script for several values of the learning rate and for various numbers of iterations (using the same set). This yields the following results:

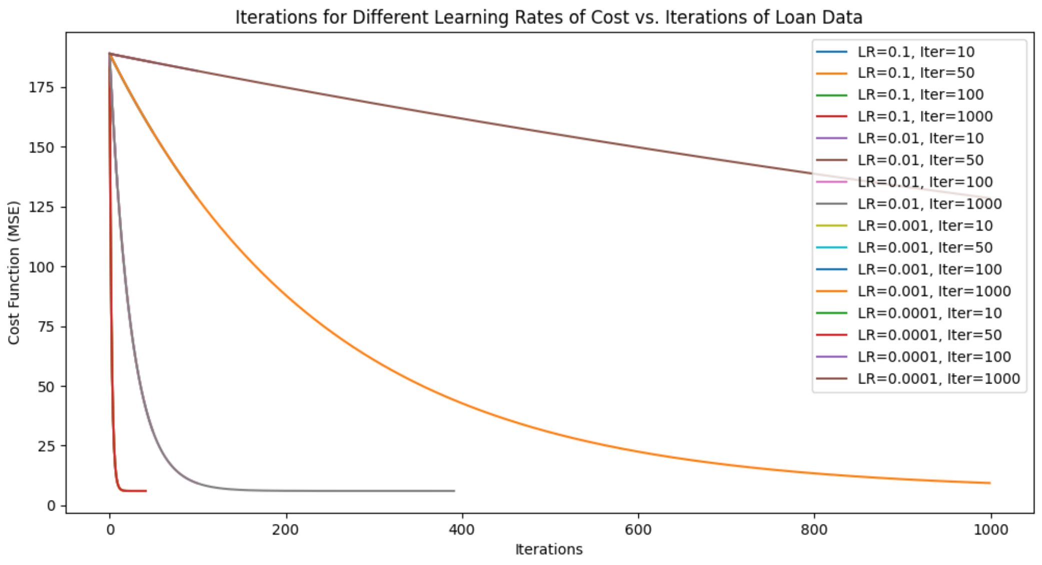
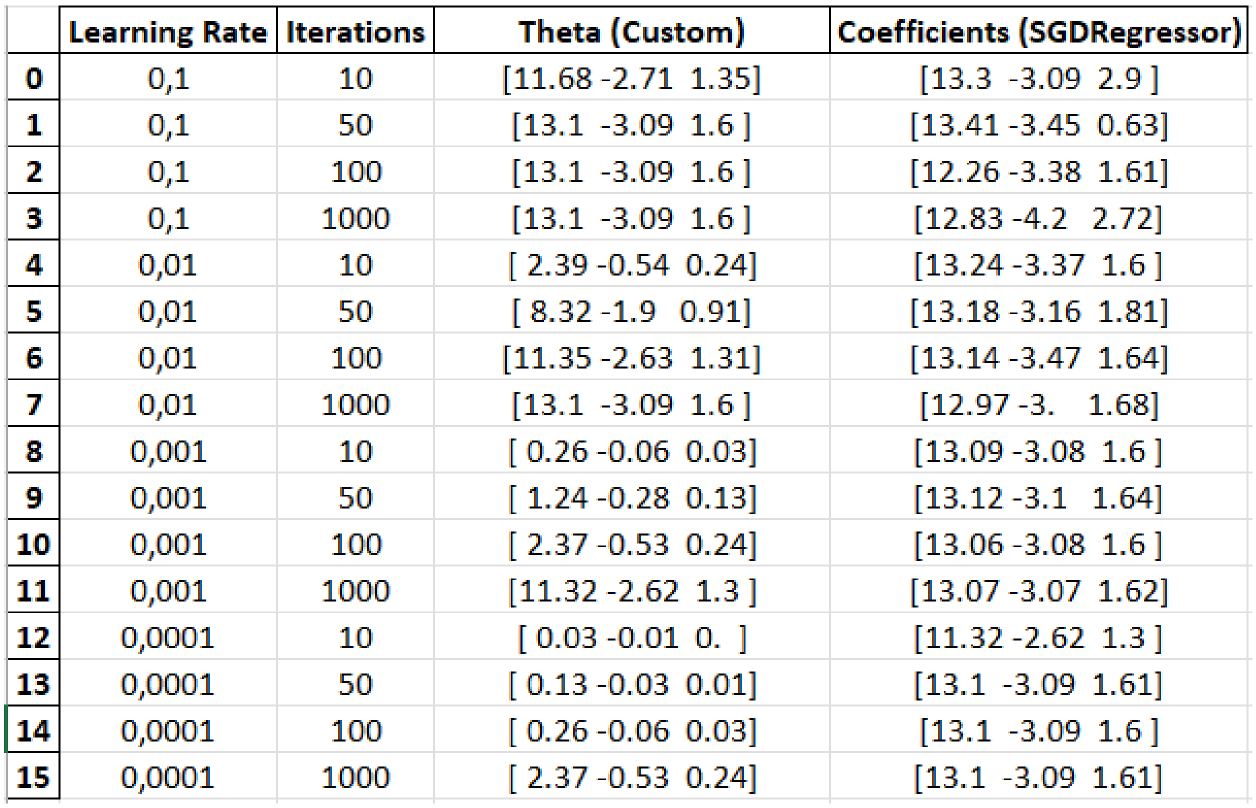


Figure 5: The convergence of the cost function for several learning rate and iteration depending on number of iterations

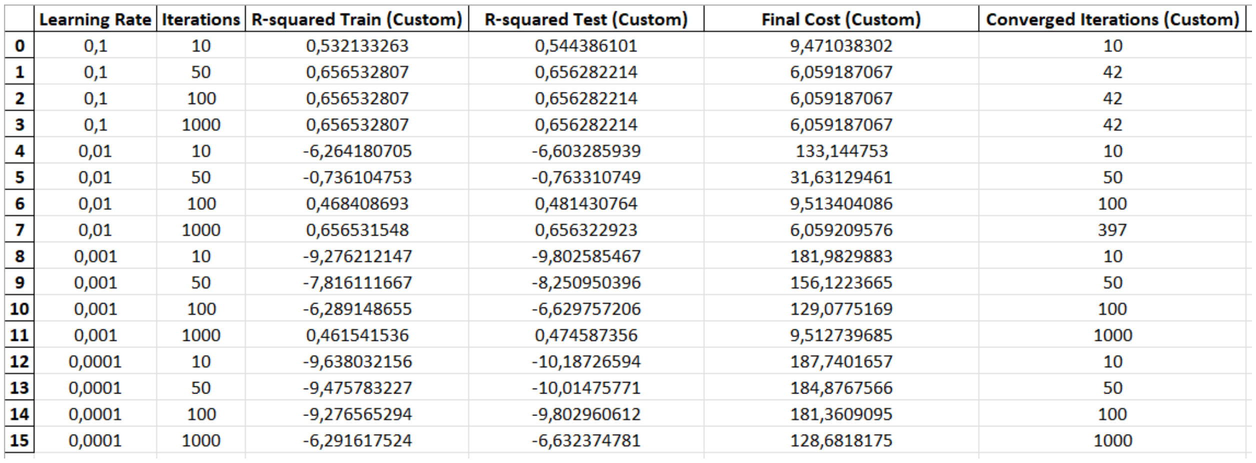
Here we have several values of theta based on different values of the learning rate and the number of iterations:



	Learning Rate	Iterations	Theta (Custom)	Coefficients (SGDRegressor)
0	0,1	10	[11.68 -2.71 1.35]	[13.3 -3.09 2.9]
1	0,1	50	[13.1 -3.09 1.6]	[13.41 -3.45 0.63]
2	0,1	100	[13.1 -3.09 1.6]	[12.26 -3.38 1.61]
3	0,1	1000	[13.1 -3.09 1.6]	[12.83 -4.2 2.72]
4	0,01	10	[2.39 -0.54 0.24]	[13.24 -3.37 1.6]
5	0,01	50	[8.32 -1.9 0.91]	[13.18 -3.16 1.81]
6	0,01	100	[11.35 -2.63 1.31]	[13.14 -3.47 1.64]
7	0,01	1000	[13.1 -3.09 1.6]	[12.97 -3. 1.68]
8	0,001	10	[0.26 -0.06 0.03]	[13.09 -3.08 1.6]
9	0,001	50	[1.24 -0.28 0.13]	[13.12 -3.1 1.64]
10	0,001	100	[2.37 -0.53 0.24]	[13.06 -3.08 1.6]
11	0,001	1000	[11.32 -2.62 1.3]	[13.07 -3.07 1.62]
12	0,0001	10	[0.03 -0.01 0.]	[11.32 -2.62 1.3]
13	0,0001	50	[0.13 -0.03 0.01]	[13.1 -3.09 1.61]
14	0,0001	100	[0.26 -0.06 0.03]	[13.1 -3.09 1.6]
15	0,0001	1000	[2.37 -0.53 0.24]	[13.1 -3.09 1.61]

Figure 6: Theta value's from our Custom method and from SGDRegressor method

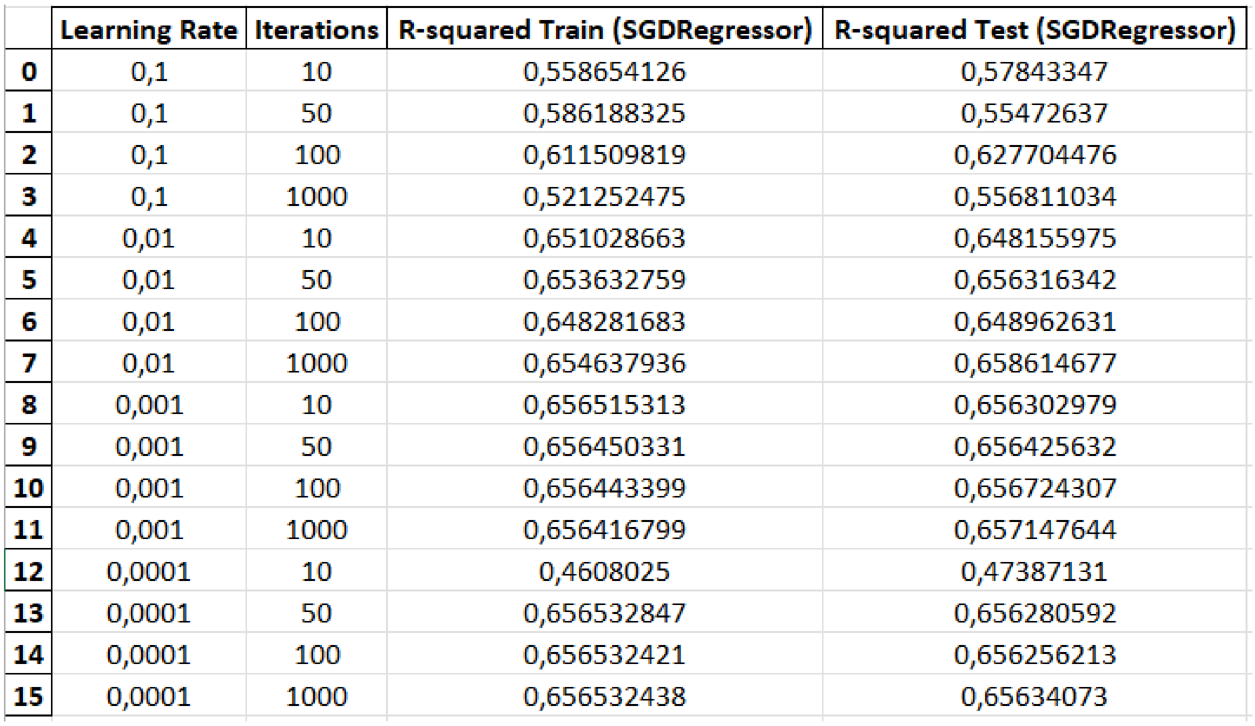
Here we have now R squared values from the same learning rate and number of iterations:



	Learning Rate	Iterations	R-squared Train (Custom)	R-squared Test (Custom)	Final Cost (Custom)	Converged Iterations (Custom)
0	0,1	10	0,532133263	0,544386101	9,471038302	10
1	0,1	50	0,656532807	0,656282214	6,059187067	42
2	0,1	100	0,656532807	0,656282214	6,059187067	42
3	0,1	1000	0,656532807	0,656282214	6,059187067	42
4	0,01	10	-6,264180705	-6,603285939	133,144753	10
5	0,01	50	-0,736104753	-0,763310749	31,63129461	50
6	0,01	100	0,468408693	0,481430764	9,513404086	100
7	0,01	1000	0,656531548	0,656322923	6,059209576	397
8	0,001	10	-9,276212147	-9,802585467	181,9829883	10
9	0,001	50	-7,816111667	-8,250950396	156,1223665	50
10	0,001	100	-6,289148655	-6,629757206	129,0775169	100
11	0,001	1000	0,461541536	0,474587356	9,512739685	1000
12	0,0001	10	-9,638032156	-10,18726594	187,7401657	10
13	0,0001	50	-9,475783227	-10,01475771	184,8767566	50
14	0,0001	100	-9,276565294	-9,802960612	181,3609095	100
15	0,0001	1000	-6,291617524	-6,632374781	128,6818175	1000

Figure 7: R-squared values from our Custom method, Final cost values and the convergence iteration number

And then the r-squared values from the SGDRegressor :



	Learning Rate	Iterations	R-squared Train (SGDRegressor)	R-squared Test (SGDRegressor)
0	0,1	10	0,558654126	0,57843347
1	0,1	50	0,586188325	0,55472637
2	0,1	100	0,611509819	0,627704476
3	0,1	1000	0,521252475	0,556811034
4	0,01	10	0,651028663	0,648155975
5	0,01	50	0,653632759	0,656316342
6	0,01	100	0,648281683	0,648962631
7	0,01	1000	0,654637936	0,658614677
8	0,001	10	0,656515313	0,656302979
9	0,001	50	0,656450331	0,656425632
10	0,001	100	0,656443399	0,656724307
11	0,001	1000	0,656416799	0,657147644
12	0,0001	10	0,4608025	0,47387131
13	0,0001	50	0,656532847	0,656280592
14	0,0001	100	0,656532421	0,656256213
15	0,0001	1000	0,656532438	0,65634073

Figure 8: R-squared values from the SGDRegressor method

0.5 Analysis and Discussion

0.5.1 Energy Dataset Analysis

Descriptive Statistics and Initial Observations

When observing for distribution of the energy dataset, the histogram revealed that there is slight skewness for some variables but overall the distribution for each variable is fairly normal.

As shown in Appendix **??** outliers are present in some of the features, especially for Ambient Pressure, and Relative Humidity. Whether we should handle these outliers depends on whether these outliers represent genuine variability in the data or if these outliers are due to measurement errors.

We analyzed the relationships between the independent and dependent variables using a correlation matrix and heat map (Show reference here). Statistically speaking, the correlation analysis highlights that Ambient Temperature (AT) and Exhaust Vacuum (V) is a critical predictors of Electrical Energy Output (PE). Increases in either of the 2 variables can result in a decrease in energy output. The weak correlations with Relative Humidity (RH) and Ambient Pressure (AP) indicate their lesser influence on PE.

Analysis of Gradient Descent Results

Our manual gradient decent results are comparable to the SGDRegressor in terms of R^2 for training and testing sets of approximately 0.93. The gradual reduction in cost values over increasing iterations further supports the reliability of finding the optimal coefficient.

Adjustment in learning rate and iteration significantly impacted the model's convergence. In the manual implementation, when the learning rate is too low the convergence point cannot be reached resulting in a negative R^2 score.

This showed that the manual implementation is sensitive to hyperparameters and implies a trade-off between model complexity and performance. Scikit-learn's model is more robust and is not sensitive to hyperparameters.

0.5.2 Loan Dataset Analysis

Descriptive Statistics and Initial Observations

Upon examining the distribution of the *Loan Dataset*, the histograms indicated that *FICO Score*, *Loan Amount*, and *Interest Rate* are fairly normally distributed, but displayed slight right-skewness (see Appendix **??**). This skewness suggests a higher concentration of lower FICO Score, loan amounts, and interest rates in the dataset.

Boxplots (Appendix **??**) identified the presence of outliers for all independent and dependent variables. The data is taken from Lending Club, a US peer-to-peer lending group (add Reference). This suggests genuine variability, such as high-risk loans or high FICO Scores.

When checking for correlation Appendix **??**), the analysis highlighted a *moderate negative correlation* between *FICO Score* and *Interest Rate* (-0.53), indicating that higher credit scores are associated with lower interest rates. However, there was a *weak positive correlation* between *Loan Amount* and *Interest Rate*, suggesting that larger loan amounts only slightly affect the interest rates.

In summary, *FICO Score* is the key predictor for *Interest Rate*, which aligns with industry practices where credit scores play a significant role in determining loan terms (add reference).

Analysis of Gradient Descent Results

The best R^2 score for the Loan dataset is around 0.68. Which is consistent with the results from SGDRegressor. This highlights that the manual model works well even with different datasets. The persistent problem still occurs with the manual model when assigning low learning rates as the iteration ends before reaching the convergence point, resulting in bad R^2 scores.

0.5.3 Discussion

Our manual implementation of gradient descent and Scikit-learn's SGDRegressor method demonstrated that while both models achieve similar R^2 scores for certain learning rates and iterations, they still do worse for lower learning rates. This could be due to how Scikit-learn calculates gradient descent versus ours. Scikit-learn uses Stochastic Gradient Descent which samples the data to calculate the cost function. Usually, this is preferred when working with large datasets, but in our case, when the value of learning rates is low using a smaller sample works well to calculate the gradient of the cost function. The limitation in our implementation of gradient descent is that it gets the gradient by using the entire dataset, so a lower value of learning rates or iteration will hurt it.

0.6 Conclusions

The goal of the assignment was to implement gradient descent according to the formula provided, test out different learning rates and interactions to find the best coefficient, and compare our R^2 result with Scikit-learn's packages. Despite not outperforming the model in Scikit-learn and on certain occasions performing worse, our overall objective is still achieved. After implementing our manual gradient decent model we got similar R^2 scores with Scikit-learn's model. Future work could explore dynamic learning rates and iteration. It would also be interesting to implement the same model as in Scikit-learn to see if the result would be the same.

Appendix A

Individual Contributions

A.1 Duong's Contributions

During the project Paul has helped in the implementating the validation method for the slope and intercept using scikit-learn. He was able to explain to me the inner working of his code as well as contributed greatly in the making of the report mainly at the methodology Scikit-learn, Plotting section and the Results section.

A.2 Paul's Contributions

For this project, I wrote an initial script that implements the gradient descent method for the "dataEnergy" dataset using lists. Duong then adapted the code using a matrix-based approach, which provided us with more usable results. He also set up additional analysis tools to better interpret our results. I then completed the Abstract, Introduction, Methodology, and Results sections of the report.

Appendix B

Source Code

B.1 Source Code

B.2 Results of the Learning Process

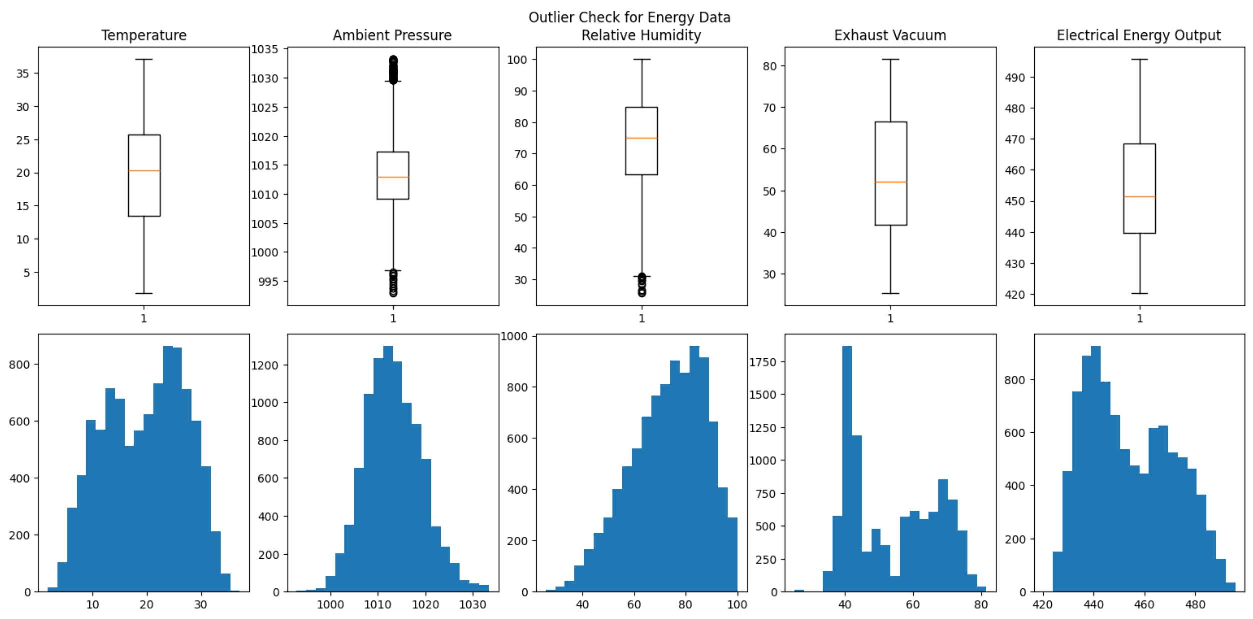


Figure B.1: Distribution of input values for the dataEnergy dataset via boxplot

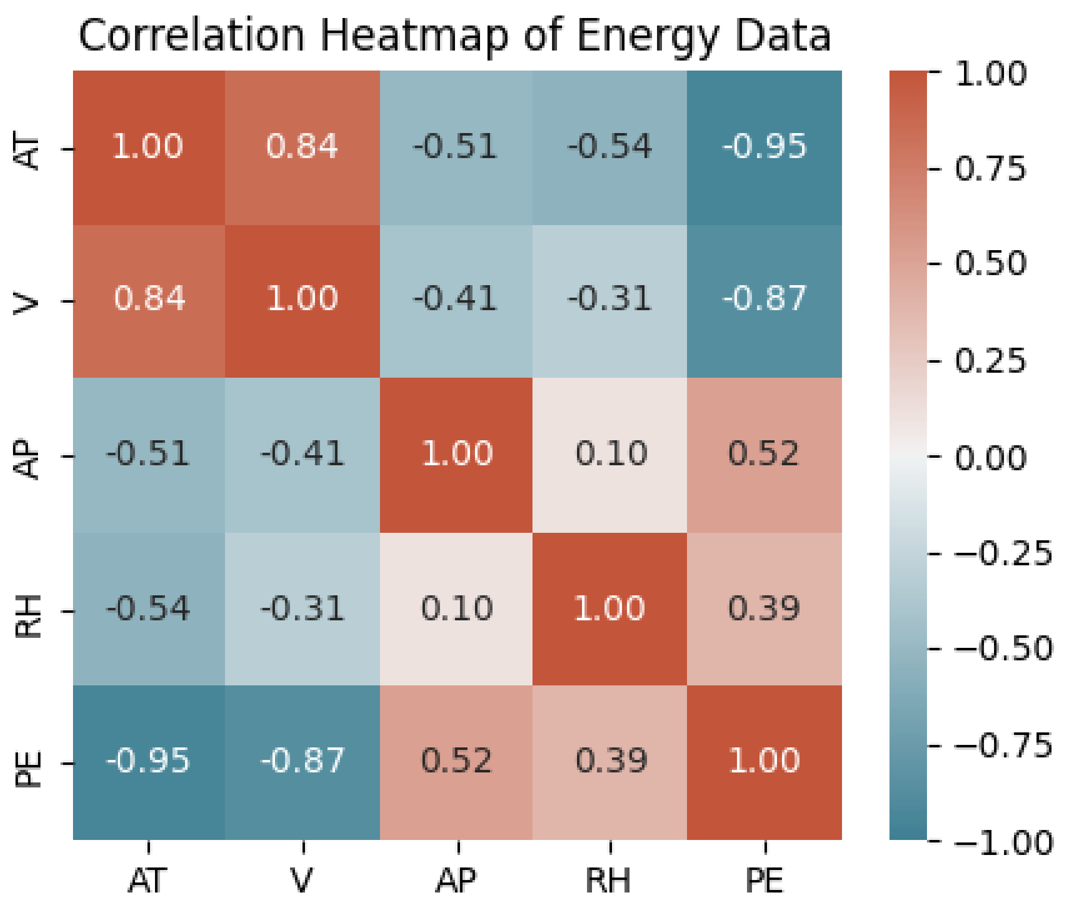


Figure B.2: Distribution of input values for the dataEnergy dataset via heatmap

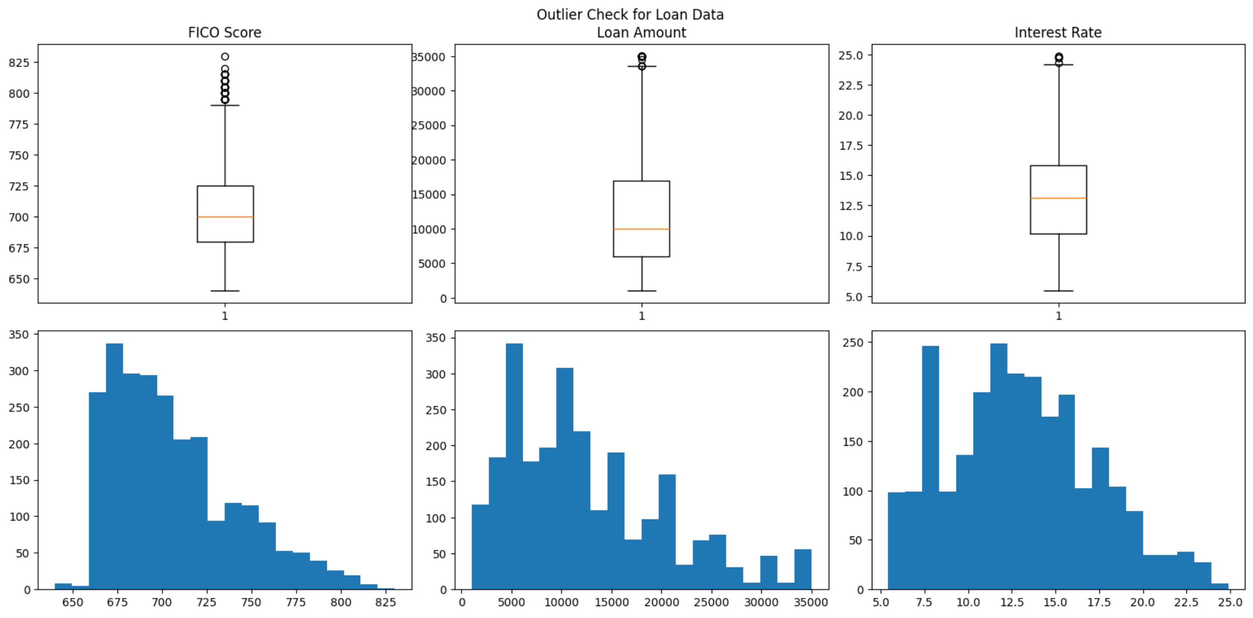


Figure B.3: Distribution of input values for the dataLoan dataset via boxplot

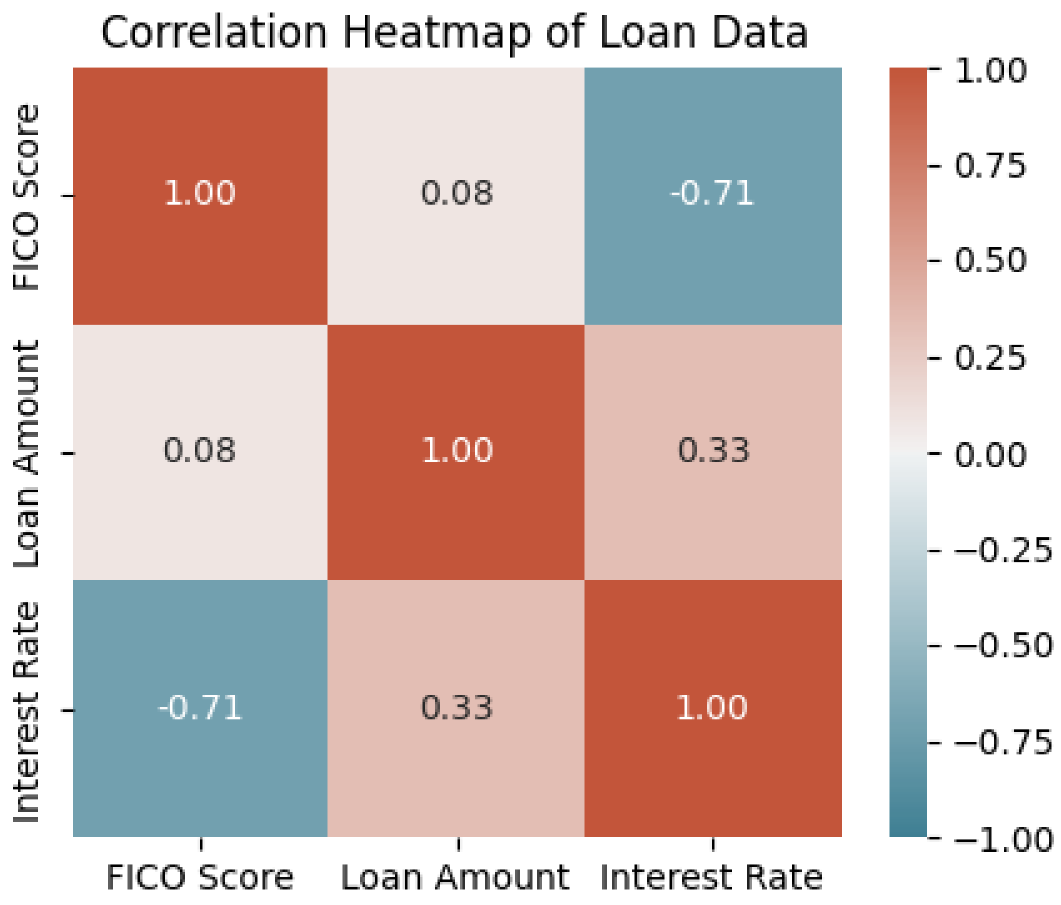


Figure B.4: Distribution of input values for the dataLoan dataset via heatmap

References

[1] N. Amral, C.S. Özveren, D. King. *Short Term Load Forecasting using Multiple Linear Regression.* [Internet]. ResearchGate; October 2007 [cited 2024 Oct 18]. Available from: `https://www.researchgate.net/publication/224309206_Short_term_load_forecasting_using_Multiple_Linear_Regression`.

[2] IBM. *What is Linear Regression?* [Internet]. IBM; n.d. [cited 2024 Oct 18]. Available from: `https://www.ibm.com/topics/linear-regression`.

[3] Jim Frost. *How To Interpret R-squared in Regression Analysis.* [Internet]. Statistics by Jim; [cited 2024 Oct 18]. Available from: `https://statisticsbyjim.com/regression/interpret-r-squared-regression/`.