**FORMAN CHRISTIAN COLLEGE (A CHARTERED UNIVERSITY)**

****

**Computer Organization with Assembly Language**

**COMP-300 (Section C)**

**Project Solution**

**Linear Regression in Assembly Language**

**Abdullah Mehtab || 241607845**

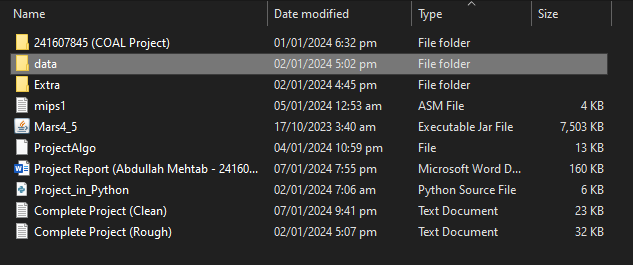
**Introduction**

This lab focuses on implementing various functions in MIPS Assembly Language to handle linear regression. The main tasks include reading data from a file, splitting it into training and testing sets, performing linear regression, and testing the model. The following report provides a step-by-step tutorial on understanding and implementing the provided MIPS code.

The purpose of **UDF-1**, named ReadnStore, is to read a CSV file containing input data into the program. A CSV file is a text file where data is separated by commas, representing a matrix. The task involves reading the file, saving the data, and displaying it as a 2D array**.   
UDF-2**, named test\_train\_data, aims to randomly select 80% of the input data for training the model and save the remaining 20% as test data. The function is designed to handle a 2D array, where each row represents a data point and each column represents a feature. The training and test data are then displayed and saved in separate arrays.   
**UDF-3**, named linRegression, focuses on implementing the Linear Regression algorithm to compute the weights of the linear regression model for a given data set using training data. The function aims to output an equation representing the linear regression model.   
**UDF-4**, named testModel, is tasked with testing the linear regression model created in UDF-3 using test data. This function fills the predicted array based on the model's weights and the test data, allowing for a comparison between the actual and predicted results.

**Important Need-to-Know disclaimer.**

* Input files are in the data folder placed in the same directory as the code.
* To successfully execute the code, the MARS.exe file should also be in the same directory to be able to detect path for the file to be read.



**Logic/Algorithm**

**(Details of Algorithm explained with code snippets)**

**UDF-0:**

Setup a main function which will call the following functions, then prompt to end.

**ReadnStore (UDF-1):** Reads data from a file (\*.csv) and stores it in memory as integer array.

**test\_train\_data (UDF-2):** Splits the data into training and testing sets.

**linRegression (UDF-3):** Implements linear regression on the training set to determine weights.

**testModel (UDF-4):** Tests the linear regression model on the testing set.

**UDF-1:**

1. **Open File:**
   * Use system call 13 to open the file (**filename**) for reading.
   * Save the file descriptor in register **$s0**.
2. **Read from File:**
   * Use system call 14 to read the file (**buffer**) into memory.
   * Print the raw data read from the file.
3. **Close File:**
   * Use system call 16 to close the file.
4. **Extract data (Number of Rows and Columns):**
   * Read the first byte to determine the number of rows.
   * Handle cases where the number of rows is a two-digit number. (cannot be bigger)
5. **Display data:**
   * Print the number of rows and columns.
6. **Parse and Display Matrix:**
   * Convert characters to integers and save them in a 2D array (**all\_data**).
   * Display the matrix.
7. **Return:**
   * Return to the calling function using **jr $ra** in MAIN.

**UDF-2:**

1. **Calculate Split Percentages:**
   * Calculate 80% of the rows for training (**$t4**).
   * Calculate 20% of the rows for testing (**$t5**).
2. **Loop through All Data:**
   * Use a loop to iterate through each row of the input matrix.
   * For each row, generate a random number (**$s7**) to decide whether to include it in training or testing.
3. **Include Rows in Training or Testing:**
   * If the random number is 0, include the row in training data.
   * If the random number is 1, include the row in test data.
4. **Check Completion:**
   * Check if 80% of training data is complete (**$t4**).
   * Check if 20% of test data is complete (**$t5**).
   * If not complete, continue the loop.
5. **Display and Save Data:**
   * Display and save the training data.
   * Display and save the test data.
6. **Return:**
   * Return to the calling function using **jr $ra** in MAIN.

**UDF-3:**

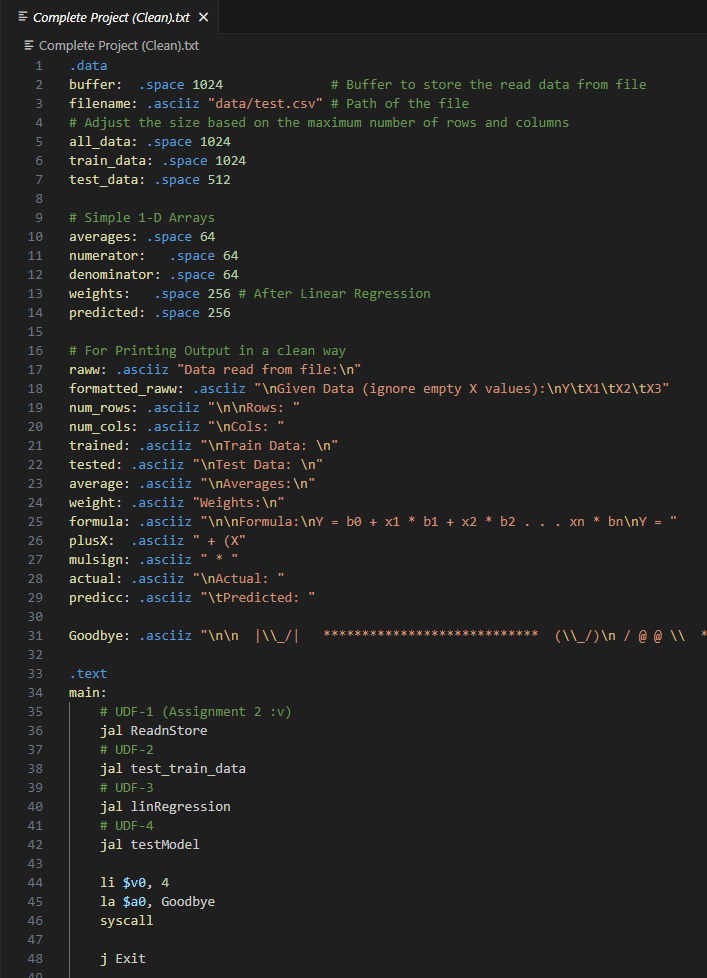
1. **Compute Column Averages:**
   * Calculate the average value for each column in the training data.
   * Print the averages and save them in an array.
2. **Calculate Numerator and Denominator:**
   * Traverse through the training data and compute the numerator and denominator for each weight. (details explained with code snippets, and formulas for calculating through reference links in given question paper)
   * Save the results in separate arrays.
3. **Calculate and Store Weights:**
   * Use the computed numerator and denominator to calculate weights.
   * Store the weights in an array.
4. **Find Y Intercept (b0):**
   * Compute the Y intercept (b0) using the calculated weights and column averages. (since it is dependent on the other values)
   * Save the Y intercept in the weights array at the first index.
5. **Display Weights:**
   * Display the calculated weights.
6. **Return:**
   * Return to the calling function using **jr $ra** in MAIN.

**UDF-4:**

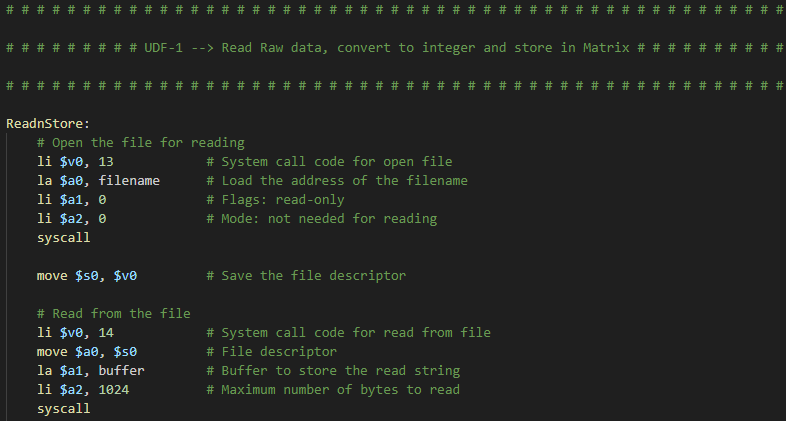
1. **Fill Predicted Array:**
   * Fill the predicted array with the Y-intercept (b0) value, as it is always added.
2. **Test the Model using Weights:**
   * Iterate through the test data and calculate predictions using the weights obtained from UDF-3.
   * Display the linear regression formula.
3. **Display Actual Data vs. Predicted Data:**
   * Compare and display the actual and predicted values for each row in the test data.
4. **Return:**
   * Return to the calling function using **jr $ra** in MAIN.

**Code Snippets with Explanation**

**Data segment and MAIN function:**

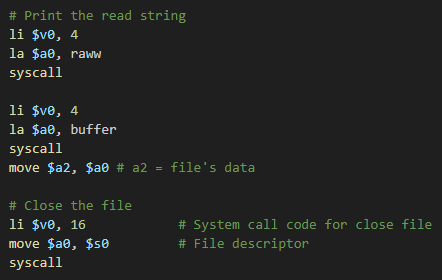
****

**UDF-1**

**Opening and Reading File**

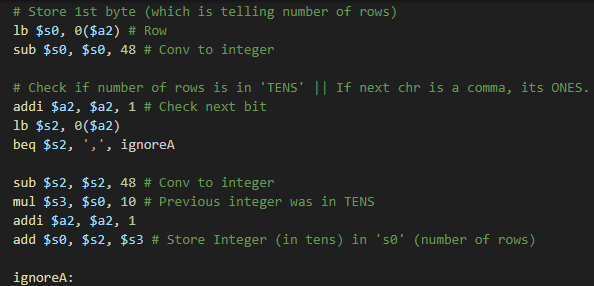
* Opens the file (filename) using system call 13.
* Loads the address of the filename into $a0.
* Sets flags for read-only ($a1) and no mode needed ($a2).
* Saves the file descriptor in $s0.
* Reads the file (**buffer**) using system call 14.
* Sets file descriptor (**$a0**), buffer address (**$a1**), and maximum bytes to read (**$a2**).

**Display Raw data once and close the file**



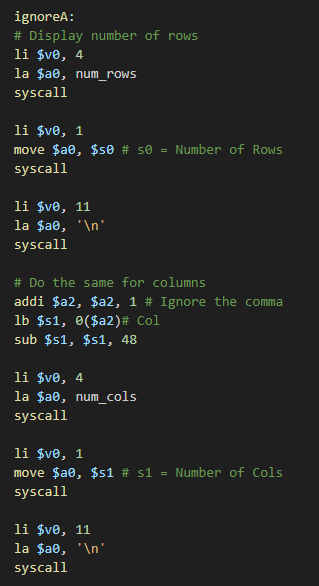
* Print the data (buffer) which is now a simple character array (aka STRING) using system call 4.
* Close the file (filename) using system call 16.

**Extract Number of Row/Col**

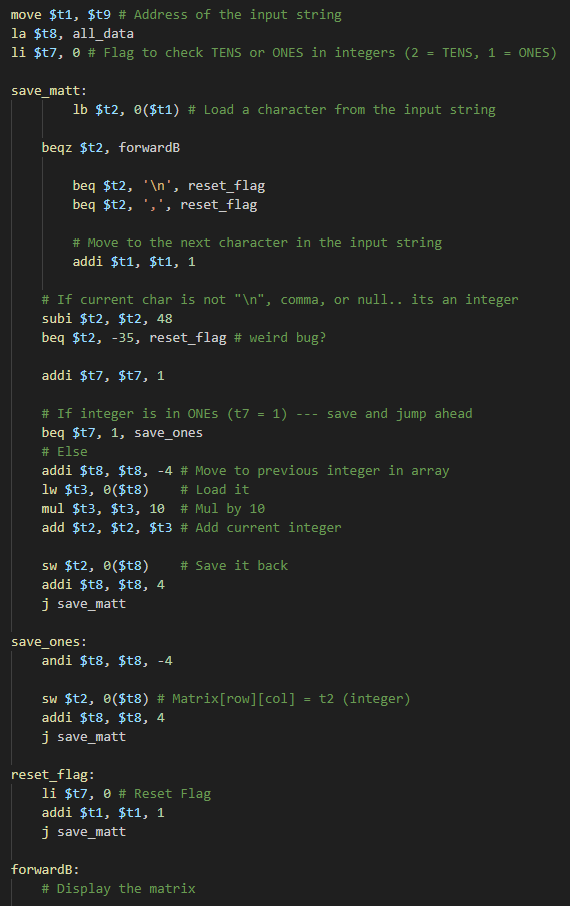


* Reads the first byte to determine the number of rows.
* Converts the ASCII character to an integer.
* Checks if the next character is a comma to handle two-digit numbers.
* Skips to the next character if it's not a comma.
* Converts the second digit of the two-digit number to an integer.
* Calculates the total number of rows.

**Display number of Row/Col**



**Parsing and Displaying Matrix**

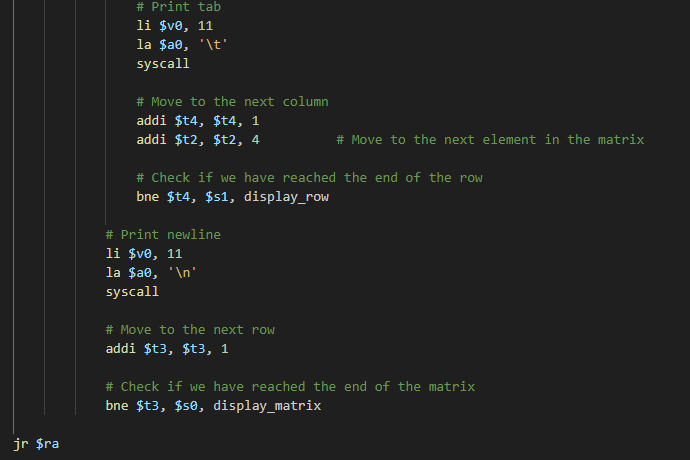


* Loads the address of the matrix (**all\_data**).
* Initializes a flag to check tens or ones in integers.
* Loads a character from the input string
* Jumps forward if the character is null
* Jumps to reset\_flag if the character is newline or comma
* Converts the character into an integer and handles a weird bug (it was detecting commas as ascii, so skipping ahead by ignoring it)
* Increment the flag for tens (or ones)
* Jumps to save\_one if it’s a ones (comma is detected)
* Handles cases where flag is 2 (not in ones, by iterating it again but not saving and jumping ahead)
* If comma or newline is found, save the previous calculated integer in array and reset the flag of checking ones or tens
* Move forward in program

**Display Matrix**

A screenshot of a computer program

Description automatically generated

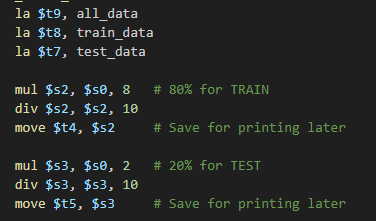


* Displays the matrix in a row-wise fashion.

Then return to the main function (where it was called)

**UDF-2**

**Calculating Split Percentages**

****

* Loads addresses of input matrix (all\_data), training data, and test data.
* Calculates 80% and 20% of the rows for training and testing, respectively.
* Saves these percentages in $t4 and $t5.

**Loop through All Data**

A screenshot of a computer program

Description automatically generated

* Uses the MARS random number generator (**syscall 42**) to generate a random number between 0 and 1.
* Saves the random number in **$s7**.

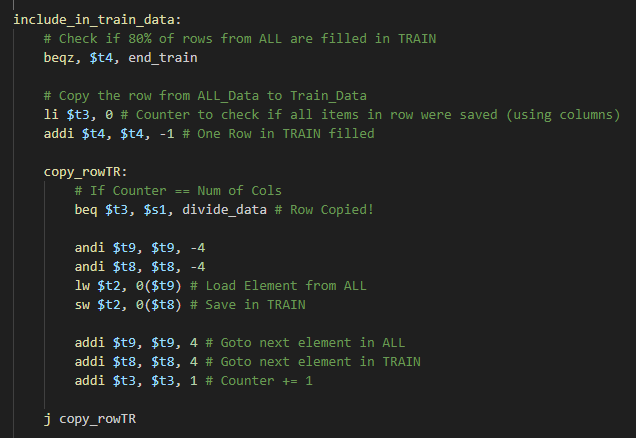
**Include Rows in Training or Testing**

A black screen with white text

Description automatically generated

* Branches to include in training if the random number is 0.
* Jumps to include in testing if the random number is 1.

**Include in Training Data**



* Branches to end if 80% of training rows are filled.
* Copies the row from **all\_data** to **train\_data**.
* Uses a loop to copy each element in the row.

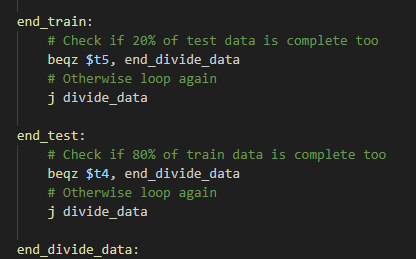
**Include in Test Data**

A screenshot of a computer program

Description automatically generated

* Branches to end if 20% of test rows are filled.
* Copies the row from **all\_data** to **test\_data**.
* Uses a loop to copy each element in the row.

**End Conditions**

****

* Checks if either training or test data is complete.
* Jumps back to the main loop if not complete.

**Display Train Matrix**

**A screenshot of a computer program

Description automatically generated**

**Display Test Matrix**

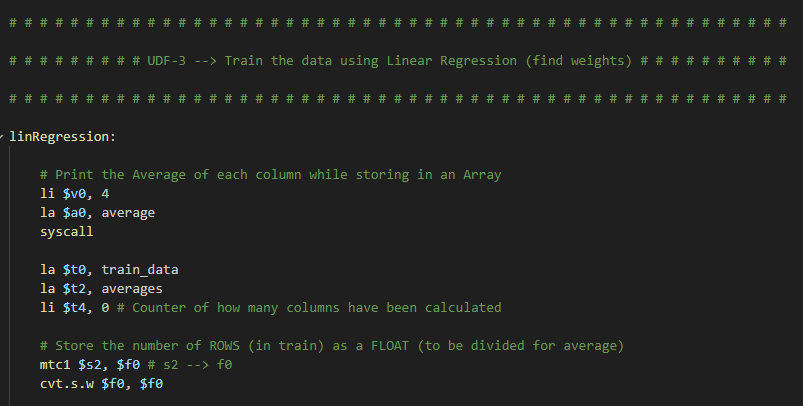
A screenshot of a computer program

Description automatically generated

Jump back to the main function

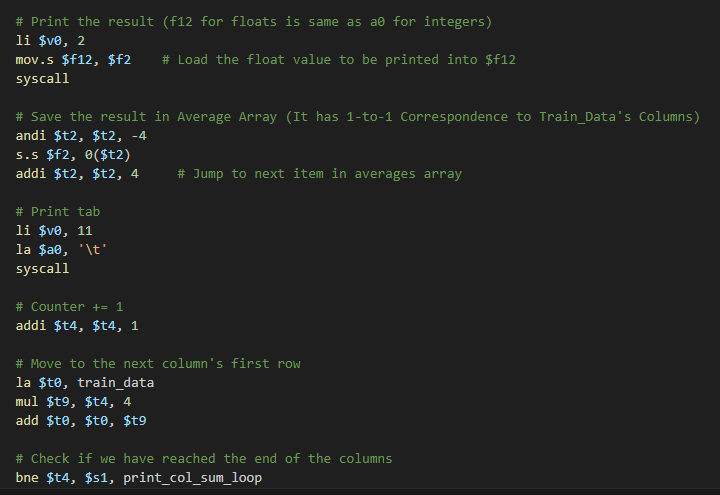
**UDF-3**

**Compute Column Averages**

****

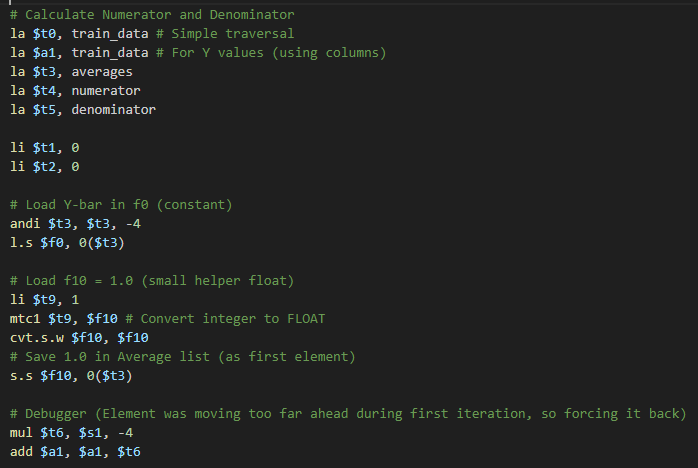
**A screenshot of a computer program

Description automatically generated**

****

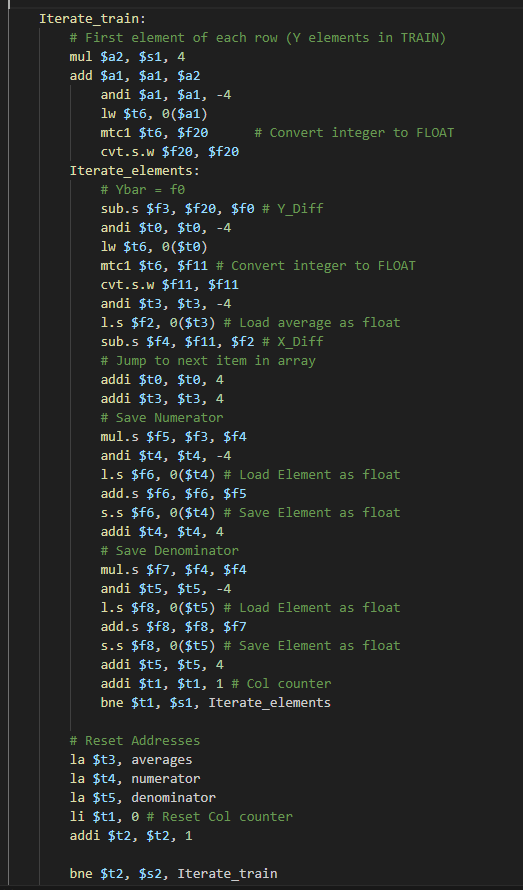
**Calculate Numerator and Denominator**

**Setup before looping**

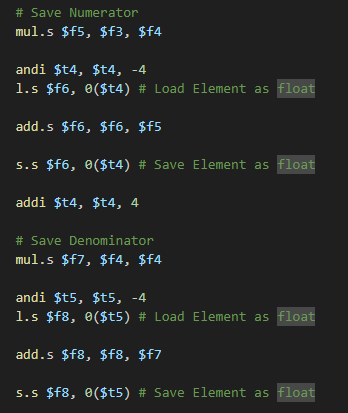
****

**Iterate through Training Data and elements to calculate**

**numerator/denominator for each column**

****

**Save Numerator and Denominator**

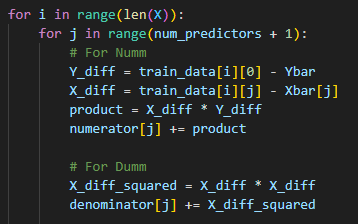
****

(repeated snippet)

NOTE: The formula and method to calculate numerators and denominators was extracted from given links provided in question paper file.

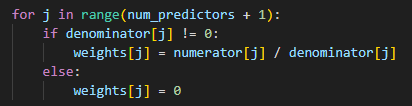
First Solved in PYTHON (since easier to translate from maths)

Then PYTHON translated to ASSEMBLY



This exact thing is happening in MIPS

**Calculate and Store Weights**

****

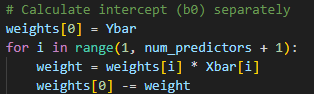
**V**

**A screenshot of a computer program

Description automatically generated**

**Find Y slope-intercept (b0) || First element in weights matrix**

**It is dependent on other weights hence it needs to be re-calculated**

****

**A computer screen shot of a program

Description automatically generated**

**Display Weights:**

**A screenshot of a computer program

Description automatically generated**

Jump back to main

**A screenshot of a computer screen

Description automatically generatedUDF-4**

**Fill Predicted Array**

(predicted values will always have the b0 intercept added in it)

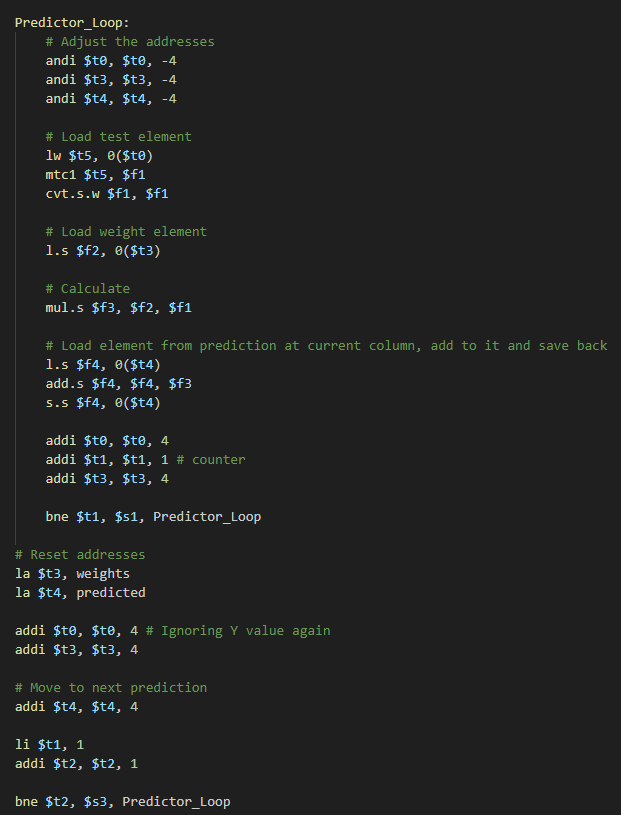
A screenshot of a computer program

Description automatically generated$f30 = b0

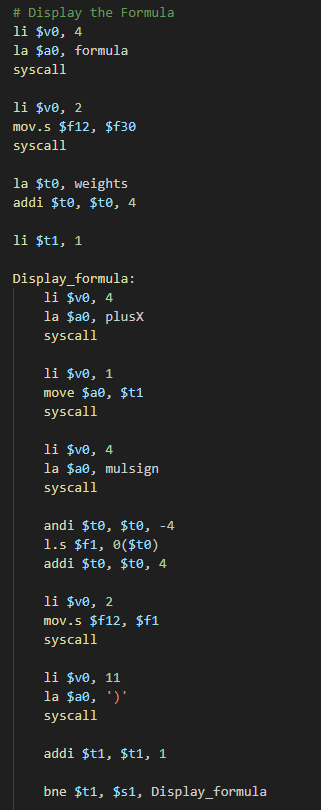
**Test model using weights and test matrix**

A screen shot of a computer code

Description automatically generated



**Display Linear Regression Formula**



**Display Actual Data vs. Predicted Data**

**A screen shot of a computer program

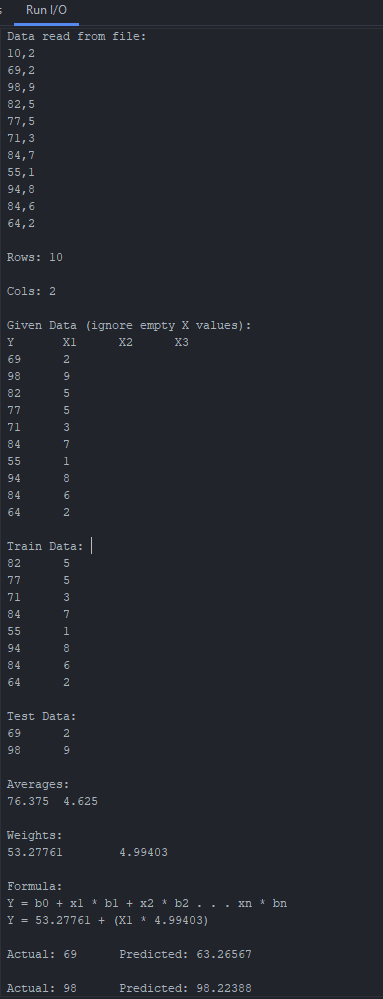
Description automatically generated**

**Screenshot of Output**

**Input CSV File:**

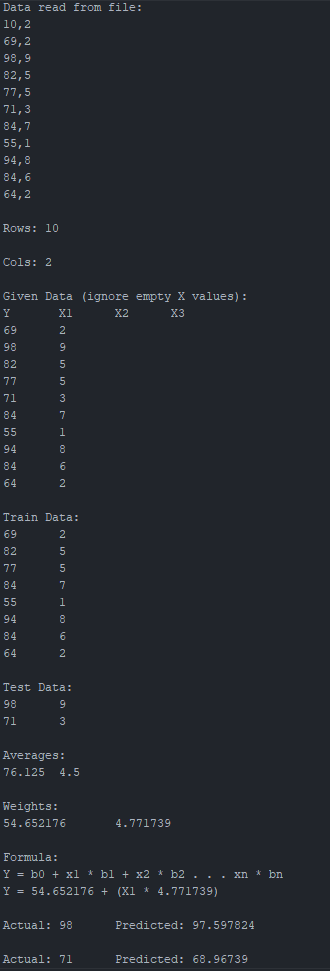
**A screenshot of a computer

Description automatically generated**

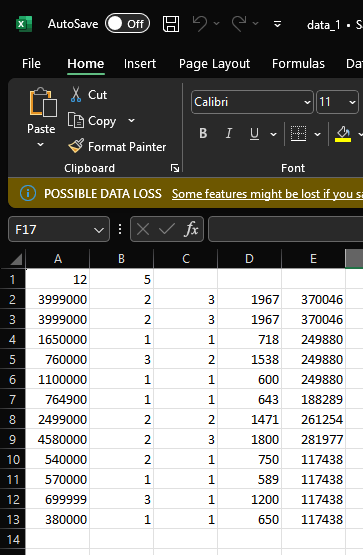
**Output Instance 1:**

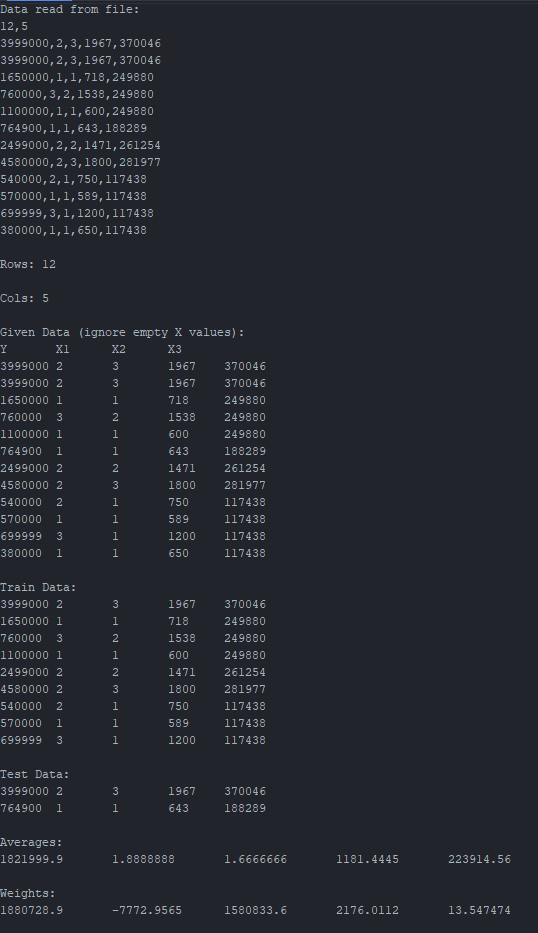
**A screenshot of a computer

Description automatically generatedOutput Instance 2**

**Output Instance 3**

**Input CSV\_File (data\_1) --- Provided by Sir**

****

****

**A screenshot of a computer

Description automatically generated**

**A screen shot of a computer screen

Description automatically generated**

**A screen shot of a computer screen

Description automatically generated**

**A screen shot of a computer screen

Description automatically generated**