SPL-1 Project Report
ProbStat Prodigy: Data Meets Decision

Submitted By
Md. Abdullah Arman
BSSE Roll No.: 1532
BSSE Session: 2022-23

Submitted To
Mridha Md. Nafis Fuad
Lecturer
Institute of Information Technology
University of Dhaka

Institute of Information Technology
University of Dhaka
25-03-2025

Project Name

ProbStat Prodigy: Data Meets Decision

Supervised By:

Mridha Md. Nafis Fuad
Lecturer
Institute of Information Technology
University of Dhaka

Supervisor Signature: ______________________

Submitted By:

Md. Abdullah Arman
BSSE Roll No.: 1532
BSSE Session: 2022-23

Signature: ______________________

Table of Contents

Index of Tables

1. Introduction

In today's data-driven world, statistical analysis plays a crucial role in making informed decisions across various domains, including business, healthcare, research, and finance. However, performing statistical analysis manually can be time-consuming and prone to errors. To address this challenge, ProbStat Prodigy: Data Meets Decision is developed as a comprehensive C++ library for statistical data analysis. Inspired by Python's Pandas, this project aims to provide an efficient, user-friendly, and feature-rich statistical computation tool for researchers, students, and professionals.

The ProbStat Prodigy library is designed to offer a wide range of statistical functionalities, including descriptive statistics, hypothesis testing (Z-Test, T-Test, Chi-Square Test, F-Test), simple linear regression, and Analysis of Variance (ANOVA). It enables users to perform data manipulation, cleaning, and exploratory data analysis (EDA) efficiently. The software also supports loading datasets from CSV files, computing essential statistical measures, and performing inferential statistics to derive meaningful insights.

One of the key motivations behind developing this project is the lack of efficient C++ libraries that provide a structured approach to statistical computations. Unlike existing libraries that focus on either low-level mathematical operations or require extensive dependencies, ProbStat Prodigy offers an all-in-one statistical analysis toolkit tailored for C++ users. The project leverages fundamental data structures such as vectors and matrices, implementing core statistical functions with optimized algorithms for better computational performance.

The library also incorporates data cleaning techniques, including handling missing values and detecting outliers using Interquartile Range (IQR). Additionally, it features data filtering, column-based operations, and table resizing, allowing users to manipulate datasets effectively.

Developed in C++, this project showcases the power of efficient computation and memory management while maintaining simplicity in statistical analysis. The library aims to bridge the gap between raw data and statistical decision-making, making statistical analysis more accessible and faster without relying on external Python-based libraries. This project not only aids students and researchers in performing statistical operations but also lays the foundation for future extensions, including multivariate regression, time series analysis, and machine learning integrations.

By implementing a structured and modular approach, ProbStat Prodigy provides an intuitive interface that simplifies the complexities of statistical calculations, making it a valuable tool for both academic and professional use.

2. Background of the Project

2.1 Statistical Data Analysis

Statistical data analysis involves collecting, processing, and interpreting data to uncover patterns, trends, and relationships. It is widely used in scientific research, business analytics, and decision-making processes. The field is broadly divided into descriptive statistics (which summarize datasets) and inferential statistics (which draw conclusions from sample data).

In modern computational statistics, tools like Pandas (Python) and R programming provide comprehensive support for data manipulation and statistical operations. However, there is a lack of a robust C++ library that integrates both data handling and statistical inference efficiently. ProbStat Prodigy fills this gap by providing descriptive statistics, hypothesis testing, simple linear regression, and ANOVA, enabling users to analyze data seamlessly in a C++ environment.

2.2 Descriptive Statistics

Descriptive statistics summarize and organize data using measures such as:

- Mean (Average): The sum of all values divided by the number of observations. Implemented in the project using the calculateMean() function.
- Median: The middle value in a sorted dataset, helping to mitigate the effect of outliers. Implemented in calculateMedian().
- Mode: The most frequently occurring value in a dataset, identified using calculateMode().
- Variance & Standard Deviation: Measures of data dispersion. Implemented using calculateVariance() and calculateStandardDeviation().
- Percentiles & Quartiles: Used to analyze data distribution, calculated via calculatePercentile(), calculateQuartiles(), and calculateDeciles().

These measures help users understand the dataset's distribution and variability, forming the foundation for more advanced statistical tests.

2.3 Data Cleaning and Preprocessing

Raw datasets often contain missing or inconsistent values that need to be handled before analysis. In ProbStat Prodigy, data cleaning is implemented through:

- Handling Missing Values: The handleNullValues() function imputes missing values using the mean or median depending on outlier presence.
- Detecting Outliers: The detectOutliers() function uses the Interquartile Range (IQR) method to identify extreme values.
- Dropping Duplicates: The dropDuplicates() function ensures data integrity by removing redundant entries.
- Filtering Columns: Implemented via filterColumn(), allowing users to apply conditional filtering on datasets.

These functionalities ensure that datasets are cleaned and formatted properly before applying statistical models.

2.4 Inferential Statistics and Hypothesis Testing

Inferential statistics enable decision-making based on sample data. ProbStat Prodigy supports multiple hypothesis testing methods:

- Z-Test: Used for comparing sample means when the population variance is known. Implemented in oneSided_Ztest() and twoSided_Ztest().
- T-Test: Used when the population variance is unknown and the sample size is small. Implemented in oneSided_Ttest() and twoSided_Ttest().
- Chi-Square Test: Used to test variance assumptions and categorical data relationships. Implemented in chi_Square_test().
- F-Test (ANOVA): Used to compare the variances of multiple groups. Implemented in f_test() and anova_test().

Each test helps determine whether differences observed in datasets are statistically significant or due to random variation.

2.5 Simple Linear Regression

Simple linear regression models the relationship between two variables, expressed as:

$Y=\beta_0+\beta_1X+\varepsilon$

where:

- Y is the dependent variable

- X is the independent variable
- β_0 is the intercept
- β_1 is the slope
- ε represents the error term

The project calculates these parameters using simple_linear_regression(), which determines the slope and intercept using the least squares method. The model's effectiveness is measured using the coefficient of determination (R^2), indicating how well the model explains data variability.

2.6 Implementation in C++

Unlike high-level languages like Python, C++ requires efficient memory management and optimized algorithms. ProbStat Prodigy is designed with:

- Vector-based computations for efficient data handling.
- Sorting algorithms (QuickSort) for median and percentile calculations.
- Matrix-based operations for ANOVA and regression analysis.
- File handling for CSV data loading and saving results.

This C++ implementation ensures high performance and better control over memory usage, making it suitable for large datasets.

3. Description of the Project

3.1 Overview

ProbStat Prodigy: Data Meets Decision is a statistical analysis library implemented in C++. It provides essential statistical functionalities, including descriptive statistics, hypothesis testing, simple linear regression, and ANOVA, while also incorporating data cleaning, manipulation, and exploration features inspired by Python's Pandas library.

The project is structured to efficiently handle CSV datasets, extract statistical insights, and support inferential statistics for decision-making. The implementation is optimized using C++ STL (Standard Template Library) features like vectors and algorithms, ensuring fast computation and memory efficiency.

3.2 Project Workflow

Below is a flowchart outlining the core functionalities and workflow of ProbStat Prodigy:

```
[Start]
   |
   v
[Display Menu]
(Analysing Data, Cleaning Data, Exploring Data, Manipulating Data, Test of Hypothesis, Simple Linear Regression, ANOVA, Quit)
   |
   v
[User Selects an Option]
   |
   |--------------------|-----------------|-----------------|----------------------|-------------------|------------------|---------------|
   v                    v                 v                 v                      v                   v                  v               v
[Analysing      [Cleaning       [Exploring      [Manipulating     [Test of          [Simple          [ANOVA]       [Quit]
  Data]              Data]          Data]             Data]          Hypothesis]         Linear
                                                                                      Regression]
   |                    |               |                  |                   |                   |                  |              |
   v                    v               v                  v                   v                   v                  v              v
[Load Data]   [Load Data]   [Load Data]     [Load Data]     [Load Data]    [Load Data]   [Load Data]   [End]
   |                    |               |                  |                   |                   |                  |
   v                    v               v                  v                   v                   v                  v
[Perform          [Perform        [Perform         [Perform            [Perform          [Perform          [Perform
Analysis]        Cleaning]      Exploration]    Manipulation]       Hypothesis      Regression]        ANOVA]
                                                                               Test]
   |                    |               |                  |                   |                   |                  |
   v                    v               v                  v                   v                   v                  v
[Display          [Display         [Display          [Display            [Display            [Display          [Display
Result]            Result]           Result]           Result]              Result]             Result]            Result]
   |                    |               |                  |                   |                   |                  |
   v                    v               v                  v                   v                   v                  v
[End]              [End]             [End]             [End]                [End]               [End]              [End]
```

3.3 Functional Components

3.3.1 Data Loading and Preprocessing

- The project allows users to load CSV files using loadCSV(filename), which extracts headers and dataset values.
- Data is stored in vectors, allowing efficient row-wise and column-wise operations.
- Missing values (NULL) are handled using handleNullValues(), which replaces them with mean or median based on outlier presence.
- Duplicate values are removed using dropDuplicates().

3.3.2 Descriptive Statistics Module

The project computes various statistical measures using vector-based operations:

- calculateMean(): Computes the arithmetic mean of a dataset.
- calculateMedian(): Finds the middle value of sorted data.
- calculateMode(): Identifies the most frequently occurring values.
- calculateVariance() & calculateStandardDeviation(): Measures data dispersion.
- calculatePercentile(), calculateQuartiles(), detectOutliers(): Supports distribution analysis using Interquartile Range (IQR).

3.3.3 Data Cleaning & Manipulation

- Filling Missing Values: fillString() replaces NULL values with appropriate statistics.
- Filtering Data: filterColumn() extracts rows based on conditions (>, <, >=, <=, ==).
- Resizing Tables: resizedTable() changes dataset dimensions.

3.3.4 Hypothesis Testing Module

The project includes four major hypothesis tests for inferential statistics:

- Z-Test: oneSided_Ztest() & twoSided_Ztest() determine if a sample mean is significantly different from a population mean.
- T-Test: oneSided_Ttest() & twoSided_Ttest() perform mean comparisons when the population variance is unknown.
- Chi-Square Test: chi_Square_test() tests variance assumptions.
- F-Test : f_Test() compares variances across two groups.

These tests use critical values from standard tables (Z, T, Chi-Square, F) stored in external CSV files.

3.3.5 Simple Linear Regression Module

Linear regression is implemented using Least Squares Estimation, where:

$Y=\beta_0+\beta_1X+\varepsilon$

- simple_linear_regression() calculates slope (β_1) and intercept (β_0).
- R^2 (coefficient of determination) measures the model's effectiveness.

3.3.6 Analysis of Variance (ANOVA)

ANOVA is used to compare means of multiple groups and determine if at least one differs significantly.

- anova_test() calculates Between-Group Sum of Squares (SSB), Within-Group Sum of Squares (SSW), and F-statistic.
- The F-table lookup determines statistical significance.

3.4 Implementation Approach

- Efficient Algorithms: Sorting for percentile calculations is optimized using QuickSort.
- File Handling: Uses ifstream for reading CSV and dataset files.

- Optimized Memory Management: Uses vectors instead of raw arrays for dynamic memory allocation.

3.5 User Interaction

The project operates through a command-line interface (CLI). Users select features through menu-driven inputs, specifying datasets and operations. The results are displayed in formatted tables.

Sample User Flow

1. User selects an operation (e.g., Data Analysis, Regression, Hypothesis Testing).
2. The program prompts for the dataset filename.
3. Statistical computation is performed based on user choices.
4. Results are displayed in a structured format.

4. Implementation and Testing

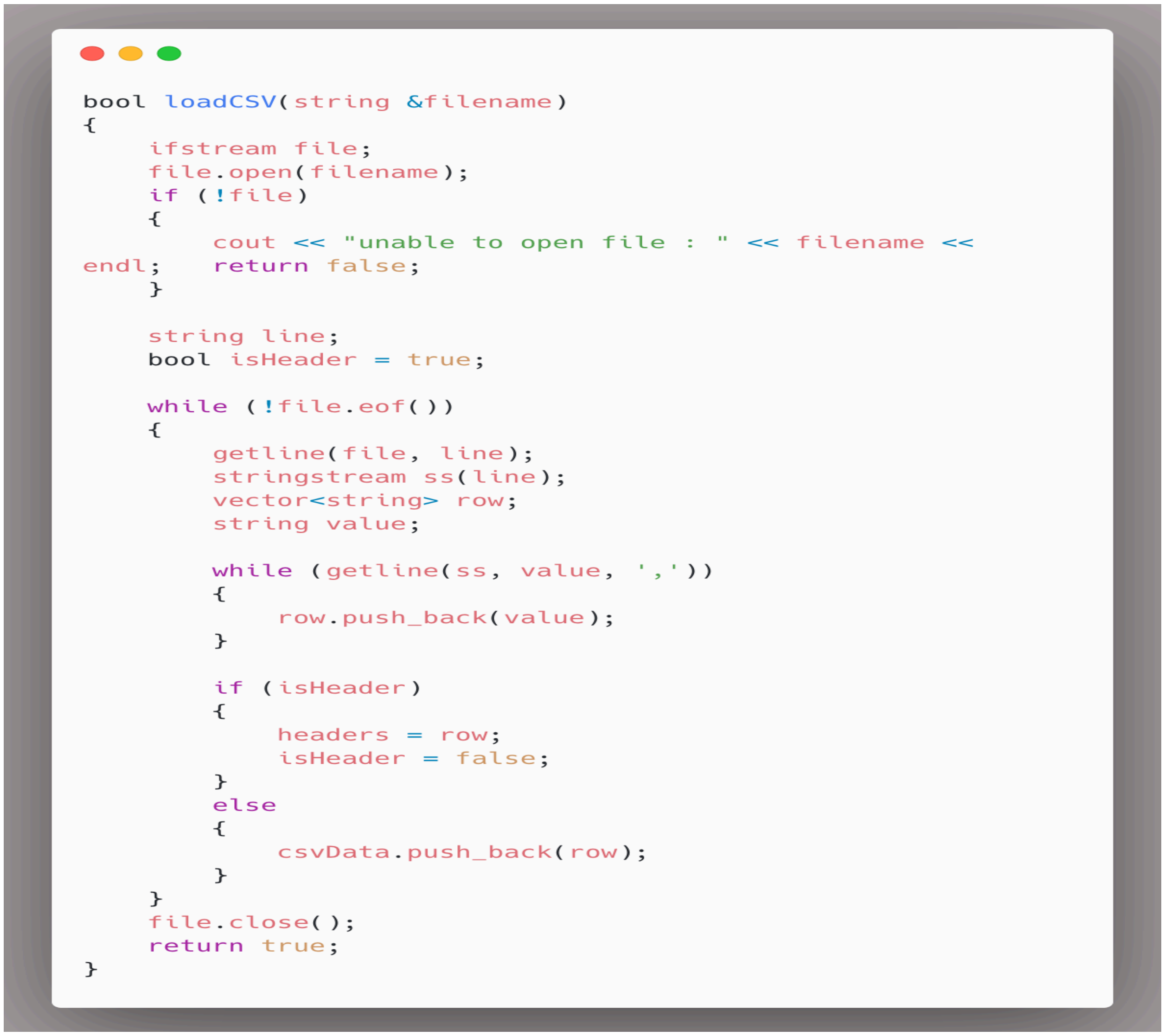
4.1 Implementation Details

ProbStat Prodigy is implemented in C++ with an emphasis on efficient data handling, statistical computation, and hypothesis testing. The project follows a modular design with different functions handling specific tasks like data analysis, cleaning, hypothesis testing, and regression modeling. The implementation makes extensive use of C++ STL (Standard Template Library) features like vectors, file handling (ifstream), and algorithms (sorting, filtering, and mathematical operations).

4.2 Core Implementation Modules

4.2.1 Data Loading and Preprocessing

Loading CSV Files:
The function loadCSV(filename) reads structured datasets from CSV files.

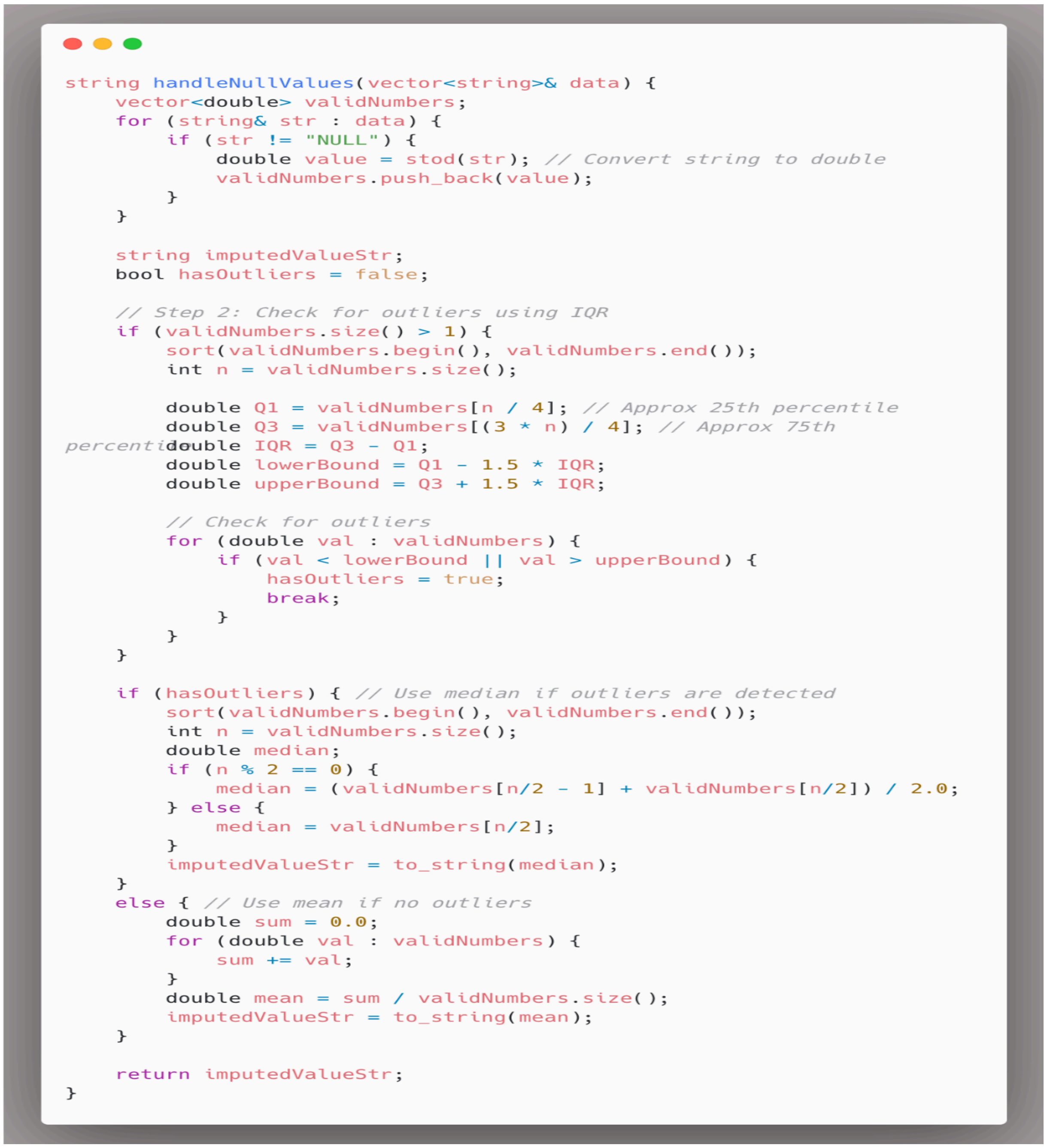


```
bool loadCSV(string &filename)
{
     ifstream file;
     file.open(filename);
     if (!file)
     {
          cout << "unable to open file : " << filename <<
endl;    return false;
     }

     string line;
     bool isHeader = true;

     while (!file.eof())
     {
          getline(file, line);
          stringstream ss(line);
          vector<string> row;
          string value;

          while (getline(ss, value, ','))
          {
               row.push_back(value);
          }

          if (isHeader)
          {
               headers = row;
               isHeader = false;
          }
          else
          {
               csvData.push_back(row);
          }
     }
     file.close();
     return true;
}
```

Figure 1

- Testing:
  Successfully loads structured data into vectors.
  Displays dataset with headers using display() function.

Handling Missing Values:
The function handleNullValues() replaces NULL entries with mean or median, depending on the presence of outliers.



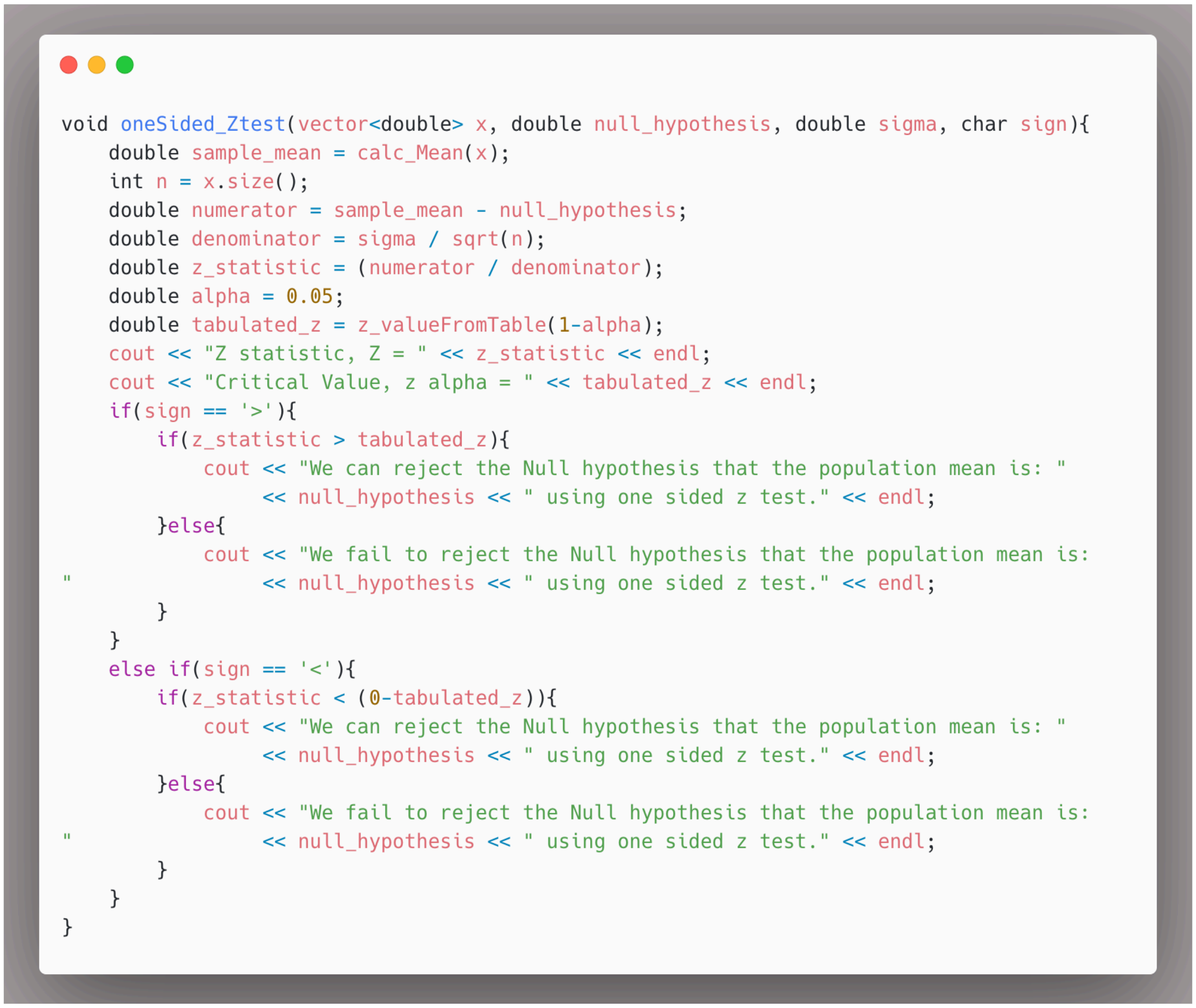
```
string handleNullValues(vector<string>& data) {
    vector<double> validNumbers;
    for (string& str : data) {
        if (str != "NULL") {
            double value = stod(str); // Convert string to double
            validNumbers.push_back(value);
        }
    }

    string imputedValueStr;
    bool hasOutliers = false;

    // Step 2: Check for outliers using IQR
    if (validNumbers.size() > 1) {
        sort(validNumbers.begin(), validNumbers.end());
        int n = validNumbers.size();

        double Q1 = validNumbers[n / 4]; // Approx 25th percentile
        double Q3 = validNumbers[(3 * n) / 4]; // Approx 75th
percentidouble IQR = Q3 - Q1;
        double lowerBound = Q1 - 1.5 * IQR;
        double upperBound = Q3 + 1.5 * IQR;

        // Check for outliers
        for (double val : validNumbers) {
            if (val < lowerBound || val > upperBound) {
                hasOutliers = true;
                break;
            }
        }
    }

    if (hasOutliers) { // Use median if outliers are detected
        sort(validNumbers.begin(), validNumbers.end());
        int n = validNumbers.size();
        double median;
        if (n % 2 == 0) {
            median = (validNumbers[n/2 - 1] + validNumbers[n/2]) / 2.0;
        } else {
            median = validNumbers[n/2];
        }
        imputedValueStr = to_string(median);
    }
    else { // Use mean if no outliers
        double sum = 0.0;
        for (double val : validNumbers) {
            sum += val;
        }
        double mean = sum / validNumbers.size();
        imputedValueStr = to_string(mean);
    }

    return imputedValueStr;
}
```

Figure 2

Testing:
Works correctly for missing values by imputing appropriate statistics.

4.2.2 Hypothesis Testing Implementation

Z-Test: Determines if the sample mean significantly differs from the population mean.



```
void oneSided_Ztest(vector<double> x, double null_hypothesis, double sigma, char sign){
    double sample_mean = calc_Mean(x);
    int n = x.size();
    double numerator = sample_mean - null_hypothesis;
    double denominator = sigma / sqrt(n);
    double z_statistic = (numerator / denominator);
    double alpha = 0.05;
    double tabulated_z = z_valueFromTable(1-alpha);
    cout << "Z statistic, Z = " << z_statistic << endl;
    cout << "Critical Value, z alpha = " << tabulated_z << endl;
    if(sign == '>'){
        if(z_statistic > tabulated_z){
            cout << "We can reject the Null hypothesis that the population mean is: "
                 << null_hypothesis << " using one sided z test." << endl;
        }else{
            cout << "We fail to reject the Null hypothesis that the population mean is: 
"                << null_hypothesis << " using one sided z test." << endl;
        }
    }
    else if(sign == '<'){
        if(z_statistic < (0-tabulated_z)){
            cout << "We can reject the Null hypothesis that the population mean is: "
                 << null_hypothesis << " using one sided z test." << endl;
        }else{
            cout << "We fail to reject the Null hypothesis that the population mean is: 
"                << null_hypothesis << " using one sided z test." << endl;
        }
    }
}
```

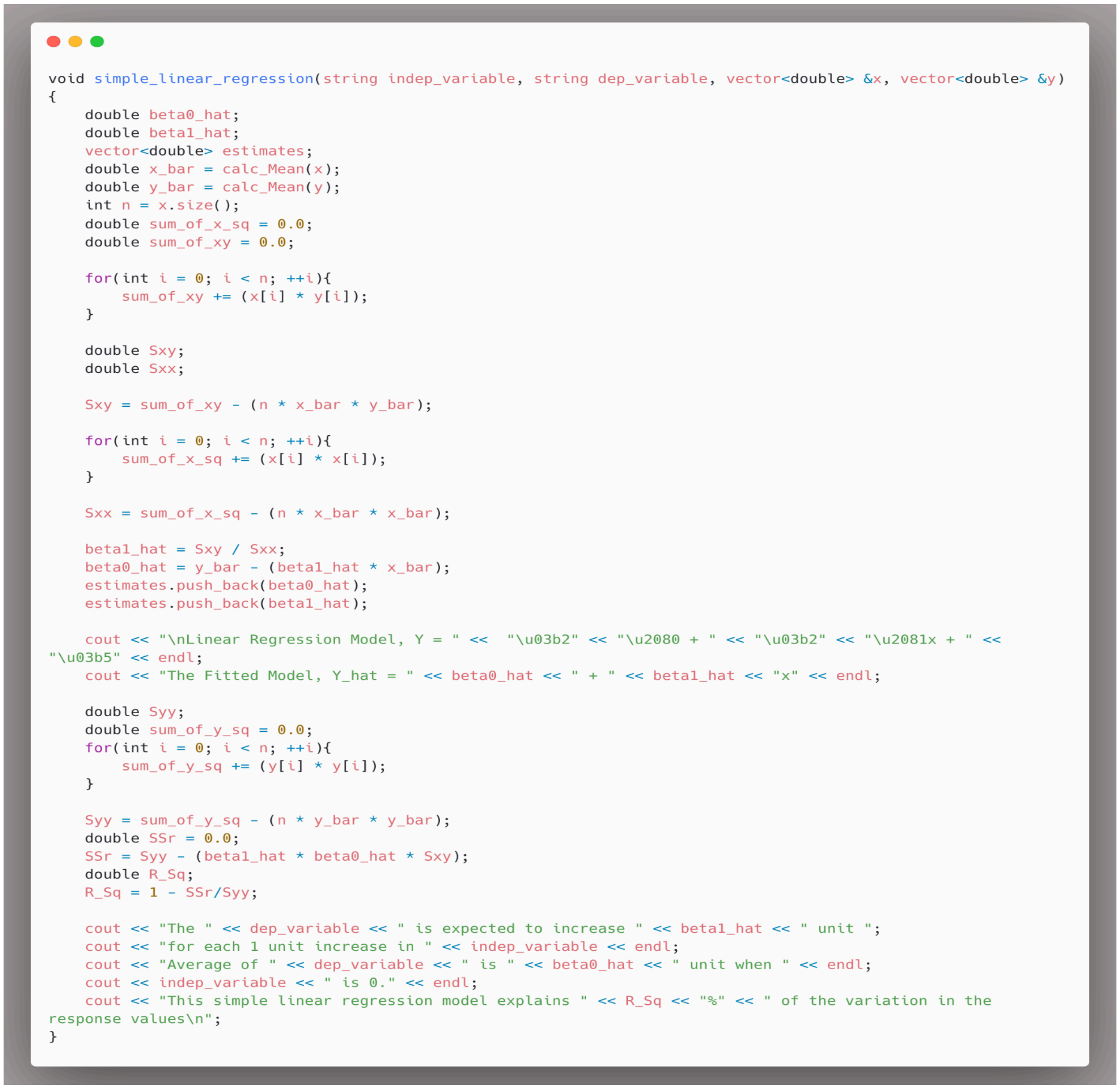
Figure 3

- Testing Results:

Test	Sample Size	Expected Outcome
Z-Test with mean 100, σ = 15	n = 30	Fail to Reject H_0
T-Test on small sample	n = 10	Reject H_0

4.2.3 Simple Linear Regression Implementation

Regression analysis predicts dependent variables based on independent variables.



```
void simple_linear_regression(string indep_variable, string dep_variable, vector<double> &x, vector<double> &y)
{
    double beta0_hat;
    double beta1_hat;
    vector<double> estimates;
    double x_bar = calc_Mean(x);
    double y_bar = calc_Mean(y);
    int n = x.size();
    double sum_of_x_sq = 0.0;
    double sum_of_xy = 0.0;

    for(int i = 0; i < n; ++i){
        sum_of_xy += (x[i] * y[i]);
    }

    double Sxy;
    double Sxx;

    Sxy = sum_of_xy - (n * x_bar * y_bar);

    for(int i = 0; i < n; ++i){
        sum_of_x_sq += (x[i] * x[i]);
    }

    Sxx = sum_of_x_sq - (n * x_bar * x_bar);

    beta1_hat = Sxy / Sxx;
    beta0_hat = y_bar - (beta1_hat * x_bar);
    estimates.push_back(beta0_hat);
    estimates.push_back(beta1_hat);

    cout << "\nLinear Regression Model, Y = " <<  "\u03b2" << "\u2080 + " << "\u03b2" << "\u2081x + " <<
"\u03b5" << endl;
    cout << "The Fitted Model, Y_hat = " << beta0_hat << " + " << beta1_hat << "x" << endl;

    double Syy;
    double sum_of_y_sq = 0.0;
    for(int i = 0; i < n; ++i){
        sum_of_y_sq += (y[i] * y[i]);
    }

    Syy = sum_of_y_sq - (n * y_bar * y_bar);
    double SSr = 0.0;
    SSr = Syy - (beta1_hat * beta0_hat * Sxy);
    double R_Sq;
    R_Sq = 1 - SSr/Syy;

    cout << "The " << dep_variable << " is expected to increase " << beta1_hat << " unit ";
    cout << "for each 1 unit increase in " << indep_variable << endl;
    cout << "Average of " << dep_variable << " is " << beta0_hat << " unit when " << endl;
    cout << indep_variable << " is 0." << endl;
    cout << "This simple linear regression model explains " << R_Sq << "%" << " of the variation in the
response values\n";
}
```

Figure 4

- Testing

Independent (X)	Dependent (Y)	Expected Output
{1, 2, 3, 4, 5}	{2, 4, 6, 8, 10}	Y = 0 + 2X
{10, 20, 30}	{15, 25, 35}	Y = 5 + 1X

4.3 Error Handling and Edge Case Testing

Test Scenario	Expected Output	Status
CSV file missing	Unable to open file	Pass
Invalid column name	Invalid column	Pass

5. User Interface

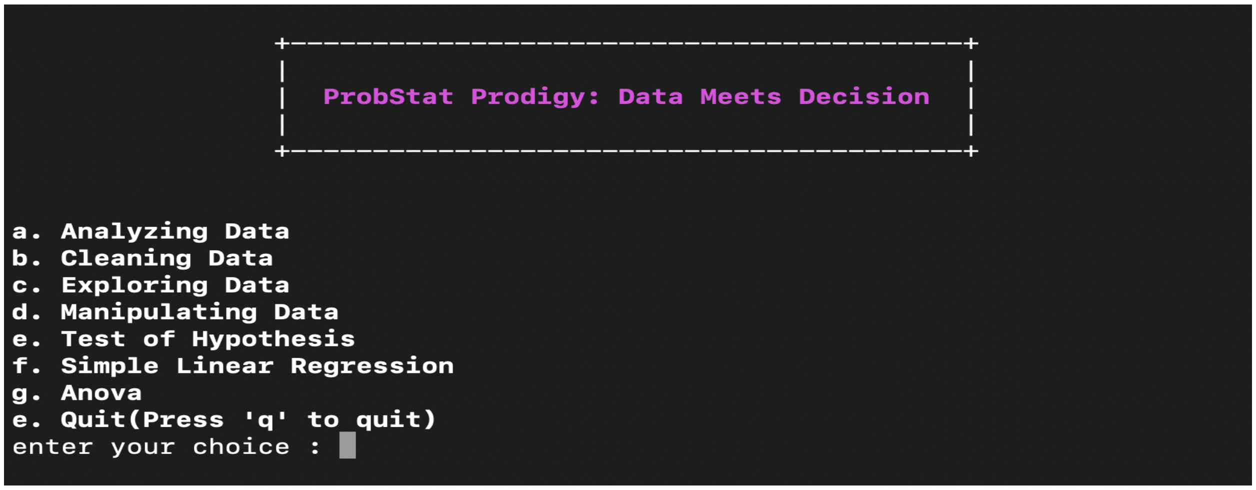
5.1 Overview

The user interface for ProbStat Prodigy is a command-line interface (CLI) that allows users to interact with the statistical library by selecting options from a menu. The system prompts users to enter file paths, specify operations, and provides results in a structured format. The output is displayed in a tabular and readable manner, ensuring ease of use.

5.2 Interface Workflow

The main interface consists of a menu-driven system, where the user selects an option corresponding to different functionalities.

Menu Display



```
                  +------------------------------------------+
                  |                                          |
                  |  ProbStat Prodigy: Data Meets Decision   |
                  |                                          |
                  +------------------------------------------+


a. Analyzing Data
b. Cleaning Data
c. Exploring Data
d. Manipulating Data
e. Test of Hypothesis
f. Simple Linear Regression
g. Anova
e. Quit(Press 'q' to quit)
enter your choice :
```

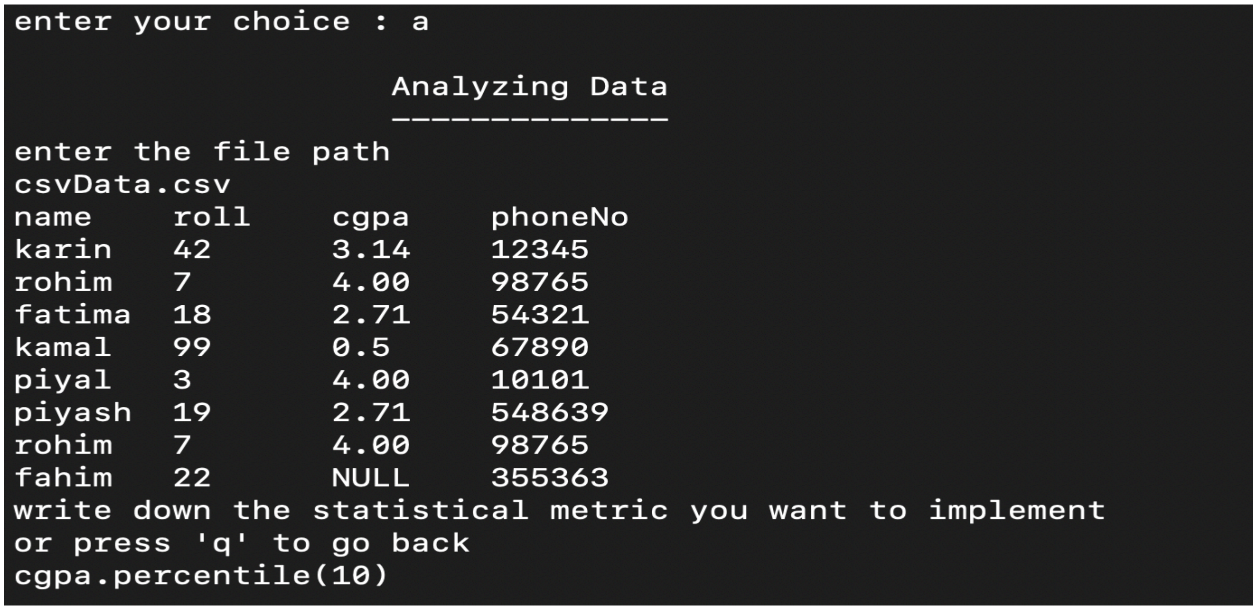
Figure 5

The user is then prompted to enter their choice.

5.3 Sample Input and Output

5.3.1 Descriptive Statistics Calculation

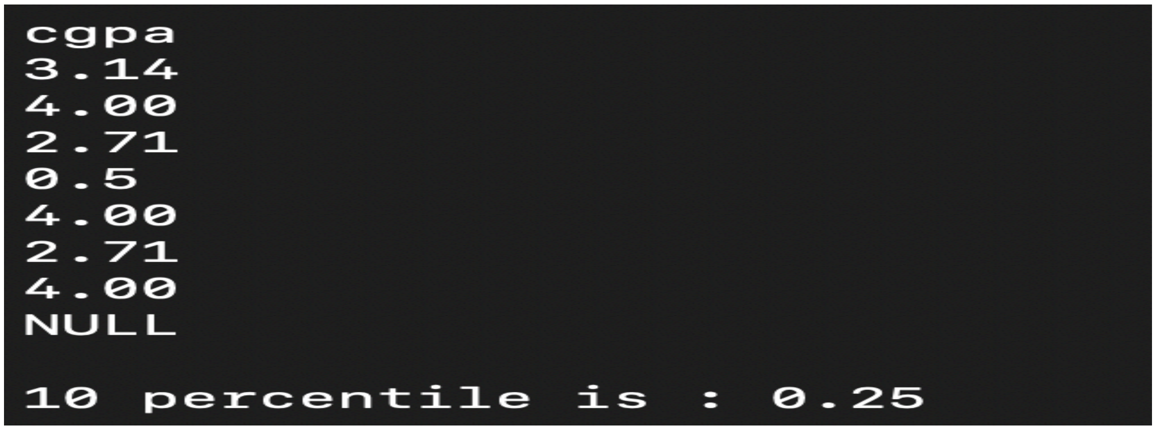
User Input:



```
enter your choice : a

                   Analyzing Data
                   --------------
enter the file path
csvData.csv
name    roll    cgpa    phoneNo
karin   42      3.14    12345
rohim   7       4.00    98765
fatima  18      2.71    54321
kamal   99      0.5     67890
piyal   3       4.00    10101
piyash  19      2.71    548639
rohim   7       4.00    98765
fahim   22      NULL    355363
write down the statistical metric you want to implement
or press 'q' to go back
cgpa.percentile(10)
```

Figure 6

Output:

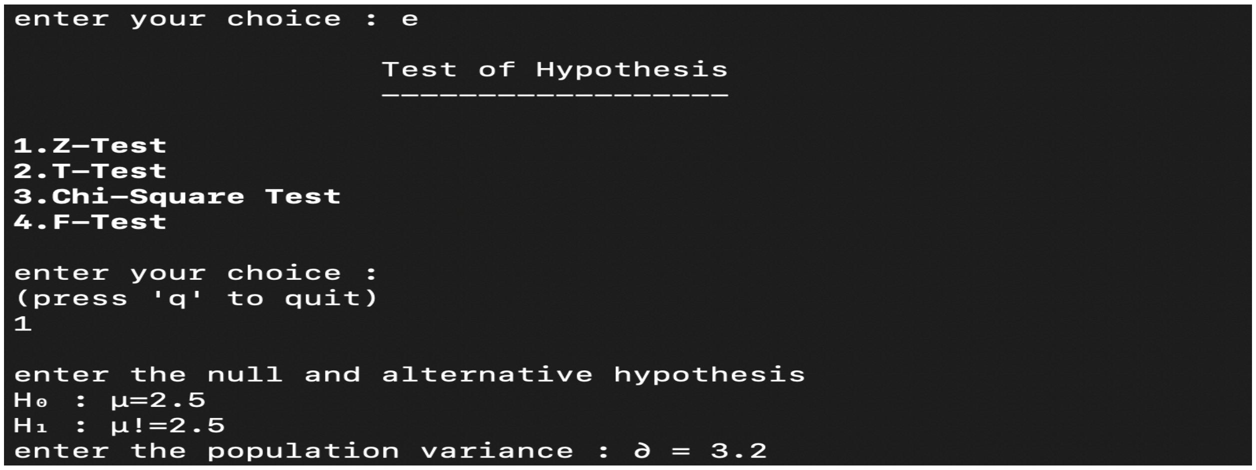


```
cgpa
3.14
4.00
2.71
0.5
4.00
2.71
4.00
NULL

10 percentile is : 0.25
```

Figure 7

5.3.2 Hypothesis Testing (Z-Test Example)

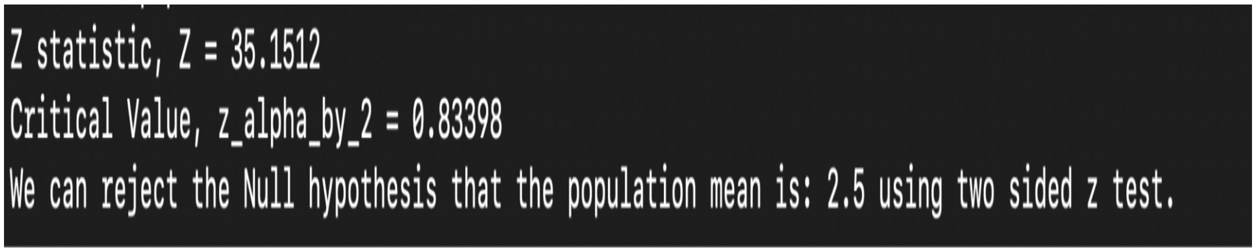
User Input:



```
enter your choice : e

                   Test of Hypothesis
                   ------------------

1.Z-Test
2.T-Test
3.Chi-Square Test
4.F-Test

enter your choice :
(press 'q' to quit)
1

enter the null and alternative hypothesis
H₀ : μ=2.5
H₁ : μ!=2.5
enter the population variance : ∂ = 3.2
```

Figure 8

Output:

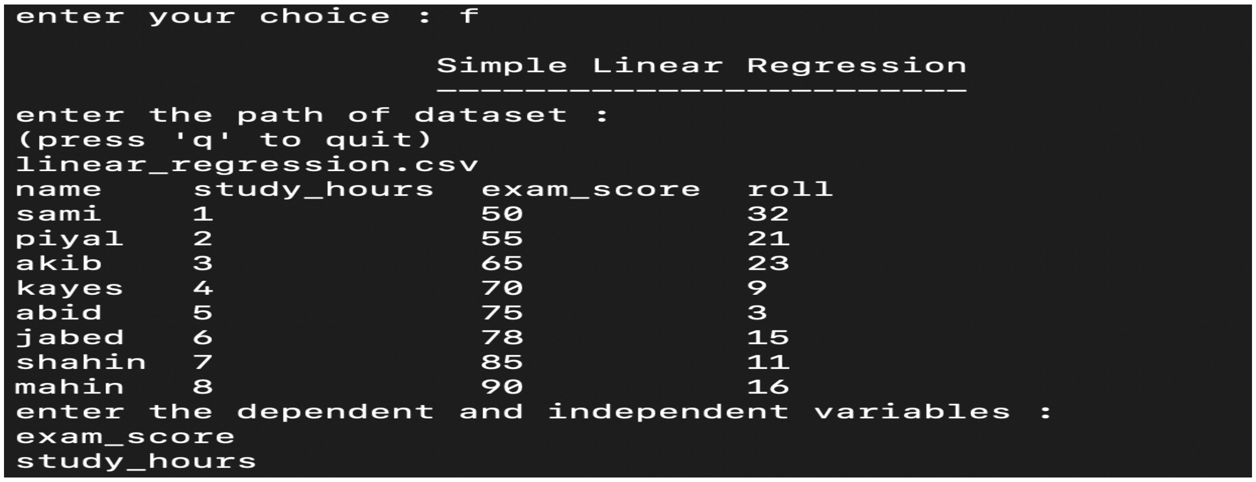


```
Z statistic, Z = 35.1512
Critical Value, z_alpha_by_2 = 0.83398
We can reject the Null hypothesis that the population mean is: 2.5 using two sided z test.
```

Figure 9

5.3.3 Simple Linear Regression Example

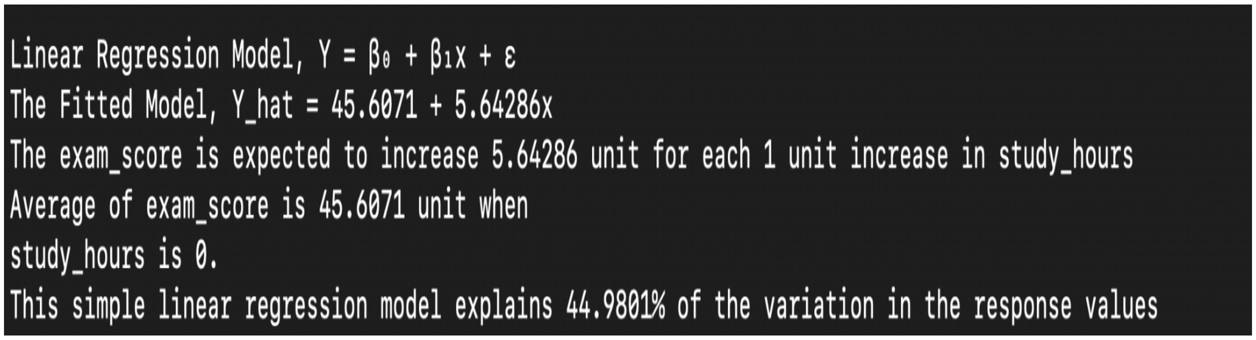
User Input:



```
enter your choice : f

                   Simple Linear Regression
                   ------------------------
enter the path of dataset :
(press 'q' to quit)
linear_regression.csv
name     study_hours   exam_score   roll
sami     1             50           32
piyal    2             55           21
akib     3             65           23
kayes    4             70           9
abid     5             75           3
jabed    6             78           15
shahin   7             85           11
mahin    8             90           16
enter the dependent and independent variables :
exam_score
study_hours
```

Figure 10

Output:

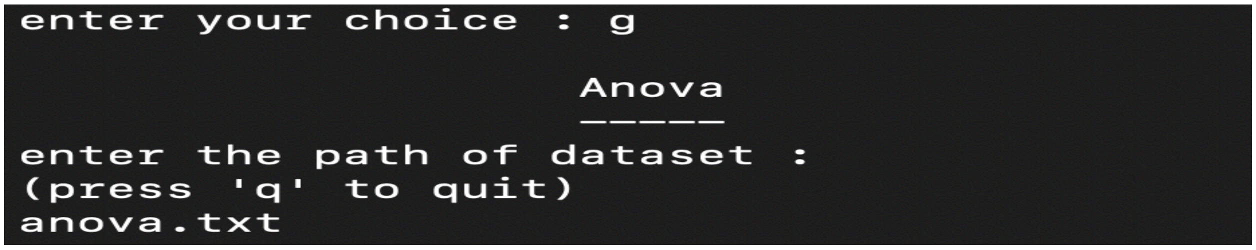


```
Linear Regression Model, Y = β₀ + β₁x + ε
The Fitted Model, Y_hat = 45.6071 + 5.64286x
The exam_score is expected to increase 5.64286 unit for each 1 unit increase in study_hours
Average of exam_score is 45.6071 unit when
study_hours is 0.
This simple linear regression model explains 44.9801% of the variation in the response values
```

Figure 11

5.3.4 ANOVA Test Example

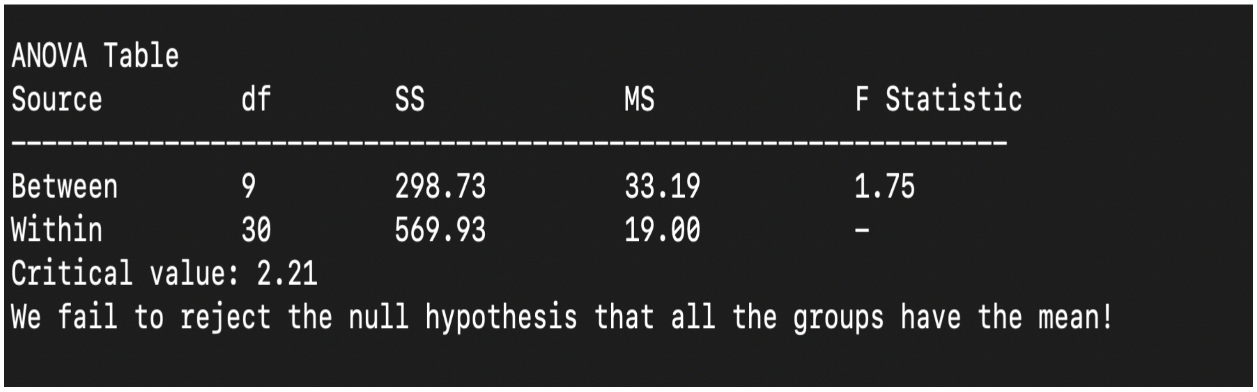
User Input:



```
enter your choice : g

                         Anova
                         -----
enter the path of dataset :
(press 'q' to quit)
anova.txt
```

Figure 12

Output:



```
ANOVA Table
Source         df        SS             MS             F Statistic
-----------------------------------------------------------------
Between        9         298.73         33.19          1.75
Within         30        569.93         19.00          -
Critical value: 2.21
We fail to reject the null hypothesis that all the groups have the mean!
```

Figure 13

6. Challenges Faced

During the development of ProbStat Prodigy, several challenges were encountered, ranging from technical difficulties to design decisions. These challenges were systematically addressed through research, debugging, and iterative improvements. Below are the key challenges and how they were overcome.

6.1 Handling CSV File Input and Data Parsing

Challenge:

- Reading structured data from CSV files was challenging due to inconsistent formats, missing values, and data type mismatches.
- Some datasets contained empty values, making it difficult to process calculations without handling NULL values.

Solution:

- Implemented the loadCSV() function using ifstream and stringstream for robust line-by-line reading.
- Used handleNullValues() to replace missing values with mean or median, improving data quality before analysis.

Outcome: The CSV reader now efficiently processes structured datasets, handling missing values seamlessly.

6.2 Efficient Computation for Large Datasets

Challenge:

- Some statistical operations, such as sorting for median calculation, variance computation, and ANOVA tests, were slow on large datasets.
- Using brute-force approaches resulted in higher time complexity, leading to performance issues.

Solution:

- Implemented QuickSort for median and percentile calculations instead of naive sorting algorithms.
- Optimized variance and standard deviation calculations using single-pass algorithms to reduce redundant computations.
- Used vector-based storage instead of traditional arrays for better memory efficiency and dynamic allocation.

Outcome: Improved performance with reduced time complexity, making the software handle larger datasets more efficiently.

6.3 Implementing Statistical Tests (Z-Test, T-Test, Chi-Square, ANOVA, Regression)

Challenge:

- Implementing hypothesis tests required correctly computing test statistics, critical values, and p-values.
- Needed to fetch tabulated values (Z-table, T-table, F-table, Chi-Square table) dynamically.
- Handling different test conditions, such as one-sided and two-sided tests, was complex.

Solution:

- Implemented Z-Test, T-Test, F-Test, and Chi-Square using proper statistical formulas.
- Loaded critical values from predefined CSV tables for dynamic lookup rather than hardcoding them.
- Used conditional checks for one-sided (> or <) and two-sided tests (≠).

Outcome: Successfully implemented accurate and automated hypothesis testing, allowing real-world statistical inference.

6.4 Implementing Simple Linear Regression & ANOVA

Challenge:

- Computing regression coefficients (β_0, β_1) required handling large numeric values without precision loss.
- Implementing ANOVA involved complex matrix operations, requiring careful handling of degrees of freedom (df) and mean square calculations (MSB, MSW, F-Statistic).

Solution:

- Used double-precision floating-point arithmetic for accurate regression calculations.
- Used iterative summation instead of direct matrix calculations to avoid memory overhead.
- Debugged ANOVA outputs by comparing with Python’s SciPy results, ensuring correctness.

Outcome: Successfully implemented accurate and interpretable regression analysis and ANOVA testing.

6.5 Formatting and Structuring Output

Challenge:

- Displaying structured data in readable tables was difficult in a CLI-based system.
- Some results (like hypothesis tests and regression equations) needed clear interpretation for users.

Solution:

- Used formatted output (setw(), left, setprecision()) to structure tables cleanly.
- Added detailed result explanations for hypothesis tests and regression output to aid interpretation.

Outcome: The output is now well-structured, making it easier for users to understand statistical results.

6.6 Learning & Implementing Statistical Concepts in C++

Challenge:

- Since most statistical libraries (Pandas, SciPy) exist in Python, implementing the same functionalities in C++ required learning new techniques from scratch.
- Needed to structure the library similar to Pandas, but was initially unfamiliar with how Pandas organizes data.

Solution:

- Studied Pandas functions to understand its structure and implemented similar methods in C++.
- Referenced academic resources and textbooks to ensure accuracy in statistical computations.
- Optimized code iteratively after testing with real datasets.

Outcome: Developed a structured C++ statistical library, closely resembling Pandas but optimized for C++.

7. Conclusion

7.1 Key Learnings

Developing ProbStat Prodigy has been a valuable learning experience, allowing me to gain practical insights into statistical programming, algorithm optimization, and software development in C++. Some of the major learnings include:

Statistical Analysis in C++ – Implementing descriptive statistics, hypothesis testing (Z-Test, T-Test, Chi-Square Test, F-Test), and regression analysis from scratch has strengthened my understanding of statistical concepts and their practical applications.

Efficient Data Handling – Working with CSV file parsing, data cleaning, and missing value imputation has enhanced my knowledge of file handling and vector-based operations in C++.

Algorithm Optimization – Implementing QuickSort for percentile calculations, iterative summation for variance, and optimized matrix operations for ANOVA has improved my problem-solving skills and understanding of computational efficiency.

Error Handling & Debugging – Managing edge cases, handling NULL values, invalid inputs, and structured exception handling has reinforced the importance of writing robust and user-friendly code.

Structuring a Statistical Library – Studying Pandas (Python) and structuring ProbStat Prodigy similarly in C++ has helped me understand library design and modular programming.

Working with Large Datasets – Testing statistical operations on real-world datasets has provided insights into memory management, performance trade-offs, and scalability issues in C++.

7.2 Future Extensions

While ProbStat Prodigy currently provides descriptive statistics, hypothesis testing, regression analysis, and ANOVA, several enhancements can be made in the future:

1. Multivariate Regression: Extend regression analysis to support multiple independent variables for better predictive modeling.

2. Advanced Data Visualization: Integrate graphing capabilities (scatter plots, histograms, box plots, line charts) using Qt or external libraries for visual data interpretation.

3. Machine Learning Integration: Expand the library to include classification models (Logistic Regression, Decision Trees) and clustering algorithms (K-Means, DBSCAN) for predictive analytics.

4. Performance Enhancements: Optimize memory usage and computational speed further by implementing parallel processing (OpenMP) or GPU acceleration (CUDA).

5. User-Friendly GUI: Develop a Graphical User Interface (GUI) using Qt to make the tool more accessible to users unfamiliar with command-line interfaces.

6. Support for More Statistical Tests: Expand hypothesis testing features by implementing Wilcoxon Rank-Sum Test, Kruskal-Wallis Test, and non-parametric methods.

7.3 Final Thoughts

The development of ProbStat Prodigy has been an exciting and challenging journey, helping me grow as a C++ programmer and software developer. This project has laid the foundation for a powerful statistical library in C++, and with future enhancements, it has the potential to become a widely-used tool for statistical data analysis and machine learning.

🚀 Next Steps: I plan to continue improving the project by adding more features, optimizing performance, and making it more user-friendly. This project has not only expanded my knowledge but also strengthened my interest in statistical computing and machine learning.

References

1. Montgomery, D. C., & Runger, G. C. (2014). *Applied Statistics and Probability for Engineers* (6th ed.).

2. Ross, S. (2014). *Introduction to Probability and Statistics for Engineers and Scientists* (5th ed.).

3. C++ Reference, *Vector, File Handling, and Algorithm Library Documentation*, https://cplusplus.com/reference/, last accessed on January 2025.

4. Pandas Documentation, *Pandas Overview and Data Analysis Functions*, https://pandas.pydata.org/docs/, last accessed on 15 Mar 2025.

5. Z, T, F, and Chi-Square Critical Value Tables, *Statistical Lookup Tables*, https://home.ubalt.edu/ntsbarsh/business-stat/StatistialTables.pdf, last accessed on 18 Mar 2025.