Numpy and Pandas in C++

Generated by Doxygen 1.12.0

Chapter 1

Introduction

1.1 About Me

I am **Aryan Gupta**. Roll No. - **230150003**.
I was tasked to implement core functionalities of **NumPy** and **Pandas** libraries in C++. These libraries replicate the data manipulation and analysis capabilities provided in Python, tailored to the needs of C++ developers.

You can view the html documentation by opening the index.html in html folder in the source directory

1.2 Compilation and Running Instructions

The source code for the libraries resides in the `source` directory. To compile and execute, use the following steps:

1. Navigate to the project directory.
2. Use the `Makefile` provided to build and run the project:
   ```
   make run
   ```
   This will compile the code in the `source` directory and execute the main program.

1.3 Features of the Libraries

NumPy Implementation:

- Includes mathematical and statistical operations like sum, mean, and variance.
- Offers slicing, indexing, and reshaping of arrays.

Pandas Implementation:

- Mimics Python's DataFrame functionality.
- Allows column-wise operations.
- Provides methods for sorting, filtering, and custom transformations.
- Supports exporting and importing from CSV format.

EDA Features:

- Comprehensive statistical summary of datasets.
- Filtering and visualization of data.

1.4 Project Objective

The primary goal was to bring the capabilities of popular Python data libraries like NumPy and Pandas to the C++ ecosystem, allowing developers to perform seamless EDA in C++.

1.5 Author

- Name: Aryan Gupta
- Roll No.: 230150003

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 4

Class Documentation

4.1 Array< T > Class Template Reference

A template class for a dynamic array that supports various mathematical and array operations. Only handles double and string.

```
#include <Array.h>
```

Public Member Functions

- Array ()

  Default constructor that initializes an empty array.
- Array (const vector< T > &data)

  Constructor that initializes the array with the given vector.
- T & operator[ ] (size_t index)

  Access an element by index.
- const T & operator[ ] (size_t index) const

  Access an element by index (const version).
- void **print** () const

  Prints the contents of the array.
- vector< T > getData () const

  Returns the internal data vector.
- size_t search (const T &value) const

  Searches for a value in the array and returns its index.
- size_t size () const

  Returns the size of the array.
- void **sort** ()

  Sorts the array in place.
- Array< T > sample (size_t sampleSize) const

  Samples a specified number of elements from the array.
- Array< T > slice (size_t start, size_t end) const

  Slices the array from start to end.
- Array< T > unique () const

  Returns unique values from the array.
- size_t count (const T &value) const

Counts occurrences of a specified value in the array.
- Array< double > add (const Array< double > &other) const

  Adds two arrays element-wise.
- Array< double > subtract (const Array< double > &other) const

  Subtracts one array from another element-wise.
- Array< double > multiply (const Array< double > &other) const

  Multiplies two arrays element-wise.
- Array< double > divide (const Array< double > &other) const

  Divides one array by another element-wise.
- Array< double > power (double exponent) const

  Raises each element of the array to a specified exponent.
- double sum () const

  Computes the sum of the elements in the array.
- double mean () const

  Computes the mean of the elements in the array.
- double std () const

  Computes the standard deviation of the elements in the array.
- double var () const

  Computes the variance of the elements in the array.
- Array< double > cumsum () const

  Computes the cumulative sum of the elements in the array.
- Array< double > cumprod () const

  Computes the cumulative product of the elements in the array.
- Array< double > sin () const

  Computes the sine of each element in the array.
- Array< double > cos () const

  Computes the cosine of each element in the array.
- Array< double > exp () const

  Computes the exponential of each element in the array.
- Array< double > log () const

  Computes the natural logarithm of each element in the array.
- std::tuple< double, double, double > quartiles () const

  Computes the quartiles of the elements in the array.
- double max () const

  Finds the maximum value in the array.
- double min () const

  Finds the minimum value in the array.

Static Public Member Functions

- static Array< T > zeros (size_t size)

  Creates an array filled with zeros.
- static Array< T > ones (size_t size)

  Creates an array filled with ones.
- static Array< T > arange (double start, double end, double step)

  Creates an array with a range of values from start to end with a specified step.
- static Array< T > linspace (double start, double end, size_t num)

  Creates an array with evenly spaced values between start and end.
- static Array< T > concatenate (const Array< T > &a, const Array< T > &b)

  Concatenates two arrays.
- static vector< Array< T > > split (const Array< T > &a, size_t num)

  Splits an array into multiple arrays.
- static Array< T > randomRand (size_t size)

  Generates an array of random samples.

4.1.1 Detailed Description

template<typename T>
class Array< T >

A template class for a dynamic array that supports various mathematical and array operations. Only handles double and string.

Template Parameters

T	The type of elements in the array.

Definition at line 13 of file Array.h.

4.1.2 Constructor & Destructor Documentation

4.1.2.1 Array() [1/2]

```
template<typename T >
Array< T >::Array ( )  [inline]
```

Default constructor that initializes an empty array.

Definition at line 27 of file Array.h.

```
00027 : data() {}
```

4.1.2.2 Array() [2/2]

```
template<typename T >
Array< T >::Array (
             const vector< T > & data)  [inline]
```

Constructor that initializes the array with the given vector.

Parameters

data	The vector to initialize the array with.

Definition at line 34 of file Array.h.

```
00034 : data(data) {}
```

4.1.3 Member Function Documentation

4.1.3.1 add()

```
template<typename T >
Array< double > Array< T >::add (
             const Array< double > & other) const
```

Adds two arrays element-wise.

Parameters

other	The array to add.

Returns

An Array<double> containing the result of the addition.

4.1.3.2 arange()

```
template<typename T >
static Array< T > Array< T >::arange (
             double start,
             double end,
             double step)  [static]
```

Creates an array with a range of values from start to end with a specified step.

Parameters

start	The starting value.
end	The ending value.
step	The step size.

Returns

An Array<T> containing the range of values.

4.1.3.3 concatenate()

```
template<typename T >
static Array< T > Array< T >::concatenate (
             const Array< T > & a,
             const Array< T > & b)  [static]
```

Concatenates two arrays.

Parameters

a	The first array.
b	The second array.

Returns

An Array<T> containing the concatenated result.

4.1.3.4 cos()

```
template<typename T >
Array< double > Array< T >::cos () const
```

Computes the cosine of each element in the array.

Returns

An Array<double> containing the cosine values.

4.1.3.5 count()

```
template<typename T >
size_t Array< T >::count (
            const T & value) const
```

Counts occurrences of a specified value in the array.

Parameters

value	The value to count.

Returns

The number of occurrences of the value.

4.1.3.6 cumprod()

```
template<typename T >
Array< double > Array< T >::cumprod () const
```

Computes the cumulative product of the elements in the array.

Returns

An Array<double> containing the cumulative product.

4.1.3.7 cumsum()

```
template<typename T >
Array< double > Array< T >::cumsum () const
```

Computes the cumulative sum of the elements in the array.

Returns

An Array<double> containing the cumulative sum.

4.1.3.8 divide()

```
template<typename T >
Array< double > Array< T >::divide (
            const Array< double > & other) const
```

Divides one array by another element-wise.

Parameters

other	The array to divide by.

Returns

An Array<double> containing the result of the division.

4.1.3.9 exp()

```
template<typename T >
Array< double > Array< T >::exp () const
```

Computes the exponential of each element in the array.

Returns

An Array<double> containing the exponential values.

4.1.3.10 getData()

```
template<typename T >
vector< T > Array< T >::getData () const
```

Returns the internal data vector.

Returns

A vector<T> containing the data.

4.1.3.11 linspace()

```
template<typename T >
static Array< T > Array< T >::linspace (
             double start,
             double end,
             size_t num)  [static]
```

Creates an array with evenly spaced values between start and end.

Parameters

start	The starting value.
end	The ending value.
num	The number of samples to generate.

Returns

An Array<T> containing the evenly spaced values.

4.1.3.12 log()

```
template<typename T >
Array< double > Array< T >::log () const
```

Computes the natural logarithm of each element in the array.

Returns

An Array<double> containing the logarithm values.

4.1.3.13 max()

```
template<typename T >
double Array< T >::max () const
```

Finds the maximum value in the array.

Returns

The maximum value.

4.1.3.14 mean()

```
template<typename T >
double Array< T >::mean () const
```

Computes the mean of the elements in the array.

Returns

The mean of the elements.

4.1.3.15 min()

```
template<typename T >
double Array< T >::min () const
```

Finds the minimum value in the array.

Returns

The minimum value.

4.1.3.16 multiply()

```
template<typename T >
Array< double > Array< T >::multiply (
             const Array< double > & other) const
```

Multiplies two arrays element-wise.

Parameters

other	The array to multiply.

Returns

An Array<double> containing the result of the multiplication.

4.1.3.17 ones()

```
template<typename T >
static Array< T > Array< T >::ones (
            size_t size)  [static]
```

Creates an array filled with ones.

Parameters

size	The size of the array.

Returns

An Array<T> filled with ones.

4.1.3.18 operator[]() [1/2]

```
template<typename T >
T & Array< T >::operator[] (
            size_t index)
```

Access an element by index.

Parameters

index	The index of the element to access.

Returns

A reference to the element at the specified index.

4.1.3.19 operator[]() [2/2]

```
template<typename T >
const T & Array< T >::operator[] (
            size_t index) const
```

Access an element by index (const version).

Parameters

index	The index of the element to access.

Returns

A const reference to the element at the specified index.

4.1.3.20 power()

```
template<typename T >
Array< double > Array< T >::power (
                double exponent) const
```

Raises each element of the array to a specified exponent.

Parameters

exponent	The exponent to raise the elements to.

Returns

An Array<double> containing the result of the power operation.

4.1.3.21 quartiles()

```
template<typename T >
std::tuple< double, double, double > Array< T >::quartiles () const
```

Computes the quartiles of the elements in the array.

Returns

A tuple containing Q1, Q2, and Q3 quartiles.

4.1.3.22 randomRand()

```
template<typename T >
static Array< T > Array< T >::randomRand (
                size_t size) [static]
```

Generates an array of random samples.

Parameters

size	The number of random samples.

Returns

An Array<T> containing random samples.

4.1.3.23 sample()

```
template<typename T >
Array< T > Array< T >::sample (
                size_t sampleSize) const
```

Samples a specified number of elements from the array.

Parameters

sampleSize	The number of elements to sample.

Returns

An Array<T> containing the sampled elements.

4.1.3.24 search()

```
template<typename T >
size_t Array< T >::search (
             const T & value) const
```

Searches for a value in the array and returns its index.

Parameters

value	The value to search for.

Returns

The index of the value, or size() if not found.

4.1.3.25 sin()

```
template<typename T >
Array< double > Array< T >::sin () const
```

Computes the sine of each element in the array.

Returns

An Array<double> containing the sine values.

4.1.3.26 size()

```
template<typename T >
size_t Array< T >::size () const
```

Returns the size of the array.

Returns

The number of elements in the array.

4.1.3.27 slice()

```
template<typename T >
Array< T > Array< T >::slice (
             size_t start,
             size_t end) const
```

Slices the array from start to end.

Parameters

start	The starting index of the slice.
end	The ending index of the slice.

Returns

An Array<T> containing the sliced elements.

4.1.3.28 split()

```
template<typename T >
static vector< Array< T > > Array< T >::split (
            const Array< T > & a,
            size_t num) [static]
```

Splits an array into multiple arrays.

Parameters

a	The array to split.
num	The number of splits.

Returns

A vector of Array<T> containing the split arrays.

4.1.3.29 std()

```
template<typename T >
double Array< T >::std () const
```

Computes the standard deviation of the elements in the array.

Returns

The standard deviation of the elements.

4.1.3.30 subtract()

```
template<typename T >
Array< double > Array< T >::subtract (
            const Array< double > & other) const
```

Subtracts one array from another element-wise.

Parameters

other	The array to subtract.

Returns

An Array<double> containing the result of the subtraction.

4.1.3.31 sum()

```
template<typename T >
double Array< T >::sum () const
```

Computes the sum of the elements in the array.

Returns

The sum of the elements.

4.1.3.32 unique()

```
template<typename T >
Array< T > Array< T >::unique () const
```

Returns unique values from the array.

Returns

An Array<T> containing unique values.

4.1.3.33 var()

```
template<typename T >
double Array< T >::var () const
```

Computes the variance of the elements in the array.

Returns

The variance of the elements.

4.1.3.34 zeros()

```
template<typename T >
static Array< T > Array< T >::zeros (
            size_t size) [static]
```

Creates an array filled with zeros.

Parameters

size	The size of the array.

Returns

An Array< T > filled with zeros.

The documentation for this class was generated from the following file:

- include/Array.h

4.2 DataFrame Class Reference

A class representing a DataFrame, which can hold multiple columns of data.

```
#include <df.h>
```

Public Member Functions

- **DataFrame** ()

  Default constructor that initializes an empty DataFrame.
- DataFrame (const vector< vector< variant< double, string > > > &inputData, const vector< string > &col↩ Names)

  Constructor that initializes the DataFrame with input data and column names.
- DataFrame (const string &csvFilePath)

  Constructor that initializes the DataFrame from a CSV file.
- void addColumn (const string &name, const ColumnType &data)

  Adds a new column to the DataFrame.
- void addColumn (const string &name, const Array< double > &data)

  Adds a new column of double values to the DataFrame.
- void addColumn (const string &name, const Array< string > &data)

  Adds a new column of string values to the DataFrame.
- void **print** () const

  Prints the entire DataFrame to the console.
- void head (size_t n=5) const

  Returns the top n rows of the DataFrame.
- void tail (size_t n=5) const

  Returns the bottom n rows of the DataFrame.
- vector< string > getColumns () const

  Returns the names of the columns in the DataFrame.
- vector< size_t > getIndex () const

  Returns the row index labels of the DataFrame.
- DataFrame copy () const

  Creates a copy of the DataFrame.
- variant< double, string > iloc (size_t row, size_t col) const

  Accesses a specific entry in the DataFrame using integer-location based indexing.
- variant< double, string > loc (size_t rowLabel, const string &colLabel) const

Accesses a specific entry in the DataFrame using label-based indexing.
- int searchRowByColumn (const string &colLabel, const variant< double, string > &value) const

  Searches for a value in a specified column and returns the row index.
- int searchColumnByRow (size_t rowIndex, const variant< double, string > &value) const

  Searches for a value in a specified row and returns the column index.
- void to_csv (const string &filePath) const

  Exports the DataFrame to a CSV file.
- void describe (int col=-1) const

  Describes the DataFrame or a specific column.
- pair< size_t, size_t > shape () const

  Returns the shape of the DataFrame as a pair of (rows, columns).
- ColumnType unique (size_t col) const

  Returns unique values from a specified column.
- size_t nunique (size_t col) const

  Returns the number of unique values in a specified column.
- double sum (size_t col) const

  Computes the sum of a specified column (for double columns).
- double mean (size_t col) const

  Computes the mean of a specified column (for double columns).
- tuple< double, double, double > quartiles (size_t col) const

  Computes the quartiles of a specified column.
- DataFrame filterString (size_t col, string threshold, bool ifMinimumLimit=true) const

  Filters the DataFrame based on a string threshold.
- DataFrame filterDouble (size_t col, double threshold, bool ifMinimumLimit=true) const

  Filters the DataFrame based on a double threshold.
- void drop (size_t col)

  Drops a specified column from the DataFrame.
- DataFrame concat (const DataFrame &other, bool axis=0) const

  Concatenates another DataFrame to the current one.
- void sort_values (size_t col, bool ascending=true)
- DataFrame apply (size_t col, function< variant< double, string >(const variant< double, string > &)> func) const

  Applies a function to a specified column of the DataFrame.
- void plot (size_t col) const

  Plots the values of a specified column.
- void hist (size_t col) const

  Creates a histogram of the values in a specified column.
- void boxplot (size_t col) const

  Creates a boxplot of the values in a specified column.

4.2.1 Detailed Description

A class representing a DataFrame, which can hold multiple columns of data.

This class can handle both numeric and string data types and provides various methods for data manipulation, statistical analysis, and visualization.

Definition at line 18 of file df.h.

4.2.2 Constructor & Destructor Documentation

4.2.2.1 DataFrame() [1/2]

```
DataFrame::DataFrame (
            const vector< vector< variant< double, string > > > & inputData,
            const vector< string > & colNames)
```

Constructor that initializes the DataFrame with input data and column names.

Parameters

inputData	A 2D vector containing the data for the DataFrame.
colNames	A vector of strings containing the names of the columns.

4.2.2.2 DataFrame() [2/2]

```
DataFrame::DataFrame (
            const string & csvFilePath)
```

Constructor that initializes the DataFrame from a CSV file.

Parameters

csvFilePath	The path to the CSV file.

4.2.3 Member Function Documentation

4.2.3.1 addColumn() [1/3]

```
void DataFrame::addColumn (
            const string & name,
            const Array< double > & data)
```

Adds a new column of double values to the DataFrame.

Parameters

name	The name of the new column.
data	The Array<double> containing the data for the new column.

4.2.3.2 addColumn() [2/3]

```
void DataFrame::addColumn (
            const string & name,
            const Array< string > & data)
```

Adds a new column of string values to the DataFrame.

Parameters

name	The name of the new column.
data	The Array<string> containing the data for the new column.

4.2.3.3 addColumn() [3/3]

```
void DataFrame::addColumn (
            const string & name,
            const ColumnType & data)
```

Adds a new column to the DataFrame.

Parameters

name	The name of the new column.
data	The data for the new column.

4.2.3.4 apply()

```
DataFrame DataFrame::apply (
            size_t col,
            function< variant< double, string >(const variant< double, string > &)> func)
const
```

Applies a function to a specified column of the DataFrame.

Parameters

col	The index of the column to apply the function to.
func	The function to apply to each element in the column.

Returns

A new DataFrame containing the results of the function application.

4.2.3.5 boxplot()

```
void DataFrame::boxplot (
            size_t col) const
```

Creates a boxplot of the values in a specified column.

Parameters

col	The index of the column to create a boxplot for.

4.2.3.6 concat()

```
DataFrame DataFrame::concat (
            const DataFrame & other,
            bool axis = 0) const
```

Concatenates another DataFrame to the current one.

Parameters

other	The DataFrame to concatenate.
axis	The axis along which to concatenate (0 for rows, 1 for columns).

Returns

A new DataFrame containing the concatenated results.

4.2.3.7 copy()

```
DataFrame DataFrame::copy () const
```

Creates a copy of the DataFrame.

Returns

A new DataFrame that is a copy of the current one.

4.2.3.8 describe()

```
void DataFrame::describe (
             int col = -1) const
```

Describes the DataFrame or a specific column.

Parameters

col	The index of the column to describe (default is -1 for the entire DataFrame).

4.2.3.9 drop()

```
void DataFrame::drop (
             size_t col)
```

Drops a specified column from the DataFrame.

Parameters

col	The index of the column to drop.

4.2.3.10 filterDouble()

```
DataFrame DataFrame::filterDouble (
             size_t col,
             double threshold,
             bool ifMinimumLimit = true) const
```

Filters the DataFrame based on a double threshold.

Parameters

col	The index of the column to filter.
threshold	The threshold value for filtering.
ifMinimumLimit	If true, filters for values greater than or equal to the threshold.

Returns

A new DataFrame containing the filtered results.

4.2.3.11 filterString()

```
DataFrame DataFrame::filterString (
             size_t col,
             string threshold,
             bool ifMinimumLimit = true) const
```

Filters the DataFrame based on a string threshold.

Parameters

col	The index of the column to filter.
threshold	The threshold value for filtering.
ifMinimumLimit	If true, filters for values greater than or equal to the threshold.

Returns

A new DataFrame containing the filtered results.

4.2.3.12 getColumns()

```
vector< string > DataFrame::getColumns () const
```

Returns the names of the columns in the DataFrame.

Returns

A vector of strings containing the column names.

4.2.3.13 getIndex()

```
vector< size_t > DataFrame::getIndex () const
```

Returns the row index labels of the DataFrame.

Returns

A vector of size_t containing the index labels.

4.2.3.14 head()

```
void DataFrame::head (
             size_t n = 5) const
```

Returns the top n rows of the DataFrame.

Parameters

n	The number of rows to return (default is 5).

4.2.3.15 hist()

```
void DataFrame::hist (
             size_t col) const
```

Creates a histogram of the values in a specified column.

Parameters

col	The index of the column to create a histogram for.

4.2.3.16 iloc()

```
variant< double, string > DataFrame::iloc (
             size_t row,
             size_t col) const
```

Accesses a specific entry in the DataFrame using integer-location based indexing.

Parameters

row	The row index.
col	The column index.

Returns

A variant containing the value at the specified location.

4.2.3.17 loc()

```
variant< double, string > DataFrame::loc (
             size_t rowLabel,
             const string & colLabel) const
```

Accesses a specific entry in the DataFrame using label-based indexing.

Parameters

rowLabel	The label of the row.
colLabel	The label of the column.

Returns

A variant containing the value at the specified location.

4.2.3.18 mean()

```
double DataFrame::mean (
             size_t col) const
```

Computes the mean of a specified column (for double columns).

Parameters

col	The index of the column to compute the mean.

Returns

The mean of the column values.

4.2.3.19 nunique()

```
size_t DataFrame::nunique (
            size_t col) const
```

Returns the number of unique values in a specified column.

Parameters

col	The index of the column to count unique values.

Returns

The number of unique values in the column.

4.2.3.20 plot()

```
void DataFrame::plot (
            size_t col) const
```

Plots the values of a specified column.

Parameters

col	The index of the column to plot.

4.2.3.21 quartiles()

```
tuple< double, double, double > DataFrame::quartiles (
            size_t col) const
```

Computes the quartiles of a specified column.

Parameters

col	The index of the column to compute quartiles.

Returns

A tuple containing the first, second (median), and third quartiles.

4.2.3.22 searchColumnByRow()

```
int DataFrame::searchColumnByRow (
            size_t rowIndex,
            const variant< double, string > & value) const
```

Searches for a value in a specified row and returns the column index.

Parameters

rowIndex	The index of the row to search.
value	The value to search for.

Returns

The index of the column containing the value, or -1 if not found.

4.2.3.23 searchRowByColumn()

```
int DataFrame::searchRowByColumn (
             const string & colLabel,
             const variant< double, string > & value) const
```

Searches for a value in a specified column and returns the row index.

Parameters

colLabel	The label of the column to search.
value	The value to search for.

Returns

The index of the row containing the value, or -1 if not found.

4.2.3.24 shape()

```
pair< size_t, size_t > DataFrame::shape () const
```

Returns the shape of the DataFrame as a pair of (rows, columns).

Returns

A pair containing the number of rows and columns.

4.2.3.25 sort_values()

```
void DataFrame::sort_values (
             size_t col,
             bool ascending = true)
```

@ brief Sorts the DataFrame based on the values of a specified column.

Parameters

col	The index of the column to sort by.
ascending	If true, sorts in ascending order; otherwise, sorts in descending order.

4.2.3.26 sum()

```
double DataFrame::sum (
             size_t col) const
```

Computes the sum of a specified column (for double columns).

Parameters

col	The index of the column to sum.

Returns

The sum of the column values.

4.2.3.27 tail()

```
void DataFrame::tail (
             size_t n = 5) const
```

Returns the bottom n rows of the DataFrame.

Parameters

n	The number of rows to return (default is 5).

4.2.3.28 to_csv()

```
void DataFrame::to_csv (
             const string & filePath) const
```

Exports the DataFrame to a CSV file.

Parameters

filePath	The path to the output CSV file.

4.2.3.29 unique()

```
ColumnType DataFrame::unique (
             size_t col) const
```

Returns unique values from a specified column.

Parameters

col	The index of the column to get unique values from.

Returns

A ColumnType containing the unique values.

The documentation for this class was generated from the following file:

- include/df.h

Chapter 5

File Documentation

5.1 EDA.cpp

```cpp
// author: Aryanthepain
#include "include/testing.h"

int main()
{
    // Load the dataframe from the CSV file
    DataFrame df("mpg.csv");

    // Display the first 5 rows of the dataset
    cout « "First 5 rows of the dataset:" « endl;
    df.head();
    cout « endl;

    // Display dataset information
    cout « "Dataset Info:" « endl;
    cout « "Shape of the dataset:" « df.shape().first « " X " « df.shape().second « "\n"
         « endl;
    df.describe();

    // Analyze the 'name' column
    cout « "Analyzing the name column:\n"
         « endl;
    size_t nameColumn = 8; // Index of the 'name' column
    df.describe(nameColumn);
    cout « "All of the names should be common, however they are not.\n"
         « "This is because there are multiple entries for a car with multiple model years\n"
         « endl;

    // Display unique values in the 'origin' column
    cout « "\nUnique values in 'origin' column:" « endl;
    size_t originColumn = 7; // Index of the 'origin' column
    auto uniqueOrigin = df.unique(originColumn);
    auto originData = get<Array<string»(uniqueOrigin);
    originData.print();

    // Filter rows where 'origin' is 'USA'
    size_t mpgColumn = 0; // Index of the 'mpg' column
    for (size_t i = 0; i < originData.size(); i++)
    {
        string targetOrigin = originData[i];
        cout « "\nTop Cars in '" « targetOrigin « "':" « endl;
        DataFrame originFilteredDf = df.filterString(originColumn, targetOrigin);
        originFilteredDf.sort_values(mpgColumn);
        originFilteredDf.head();
    }

    // Perform statistical analysis for the 'mpg' column
    cout « "\nStatistical analysis of 'mpg' column:" « endl;
    cout « "Sum: " « df.sum(mpgColumn) « endl;
    cout « "Mean: " « df.mean(mpgColumn) « endl;

    auto [q1, q2, q3] = df.quartiles(mpgColumn);
    cout « "Quartiles - Q1: " « q1 « ", Q2 (Median): " « q2 « ", Q3: " « q3 « endl;

    // Sorting by 'mpg' column in descending order
    cout « "\nSorting by 'mpg' column (descending):" « endl;
    DataFrame sortedDf = df.copy();
    sortedDf.sort_values(mpgColumn, false); // Sort in descending order
    sortedDf.head();

    // Filter rows where 'mpg' > 30
    double mpgThreshold = 30.0;
    cout « "\nFiltering rows where 'mpg' > " « mpgThreshold « ":" « endl;
    DataFrame filteredDf = df.filterDouble(mpgColumn, mpgThreshold, true);
    filteredDf.head();

    // Flagging rows where 'mpg' > 30
    cout « "\nFlagging rows where 'mpg' > 30:" « endl;
    auto flaggedDf = df.apply(
        mpgColumn,
        [](const variant<double, string> &value) -> variant<double, string>
        {
            return get<double>(value) > 30.0 ? "High MPG" : "Low MPG";
        });
    flaggedDf.head();

    // Plotting the distribution of the 'mpg' column
    cout « "\nPlotting 'mpg' column distribution:" « endl;
    df.hist(mpgColumn);

    // Line plot for the sorted 'mpg' column
    cout « "\nLine plot for sorted 'mpg' column:" « endl;
    sortedDf.plot(mpgColumn);

    return 0;
}
```

5.2 Array.h

```
// author: Aryanthepain
#ifndef ARRAY_H
#define ARRAY_H
#include <bits/stdc++.h>
using namespace std;

template <typename T>
class Array
{
private:
    vector<T> data;

    // Helper functions
    Array<T> elementWiseOperation(const Array<T> &other, std::function<T(T, T)> op) const;
    Array<T> elementWiseOperation(std::function<T(T)> op) const;

public:
    // Constructors
    Array() : data() {}

    Array(const vector<T> &data) : data(data) {}

    static Array<T> zeros(size_t size);

    static Array<T> ones(size_t size);

    static Array<T> arange(double start, double end, double step);

    static Array<T> linspace(double start, double end, size_t num);

    // Access methods
    T &operator[](size_t index);

    const T &operator[](size_t index) const;

    void print() const;

    vector<T> getData() const;

    size_t search(const T &value) const;

    size_t size() const;

    // Array Manipulation
    static Array<T> concatenate(const Array<T> &a, const Array<T> &b);

    static vector<Array<T>> split(const Array<T> &a, size_t num);

    static Array<T> randomRand(size_t size);

    void sort();

    Array<T> sample(size_t sampleSize) const;

    Array<T> slice(size_t start, size_t end) const;

    Array<T> unique() const;

    size_t count(const T &value) const;

    // Mathematical Operations for double
    Array<double> add(const Array<double> &other) const;

    Array<double> subtract(const Array<double> &other) const;

    Array<double> multiply(const Array<double> &other) const;

    Array<double> divide(const Array<double> &other) const;

    Array<double> power(double exponent) const;

    double sum() const;

    double mean() const;

    double std() const;

    double var() const;

    Array<double> cumsum() const;

    Array<double> cumprod() const;

    Array<double> sin() const;

    Array<double> cos() const;

    Array<double> exp() const;

    Array<double> log() const;

    std::tuple<double, double, double> quartiles() const;

    double max() const;

    double min() const;
};

#endif // ARRAY_H
```

5.3 df.h

```
// author: Aryanthepain
#ifndef DF_H
#define DF_H

#include <bits/stdc++.h>
#include "Array.h"
using namespace std;

// Define a variant type for the DataFrame columns
using ColumnType = variant<Array<double>, Array<string>>;

class DataFrame
{
private:
    vector<ColumnType> columns;
    vector<string> columnNames;
    vector<size_t> indexLabels;

    // Helper functions
    void describeNumericColumn(size_t index) const;
    void describeStringColumn(size_t index) const;
    bool OutOfBounds(size_t num) const;
    DataFrame concatColumns(const DataFrame &other) const;
    DataFrame concatRows(const DataFrame &other) const;

public:
    // Constructors

    DataFrame();

    DataFrame(const vector<vector<variant<double, string»> &inputData,
              const vector<string> &colNames);

```

```cpp
    DataFrame(const string &csvFilePath);

    void addColumn(const string &name, const ColumnType &data);

    void addColumn(const string &name, const Array<double> &data);

    void addColumn(const string &name, const Array<string> &data);

    // Access methods

    void print() const;

    void head(size_t n = 5) const;

    void tail(size_t n = 5) const;

    vector<string> getColumns() const;

    vector<size_t> getIndex() const;

    DataFrame copy() const;

    variant<double, string> iloc(size_t row, size_t col) const;

    variant<double, string> loc(size_t rowLabel, const string &colLabel) const;

    int searchRowByColumn(const string &colLabel, const variant<double, string> &value) const;

    int searchColumnByRow(size_t rowIndex, const variant<double, string> &value) const;

    void to_csv(const string &filePath) const;

    // Describe the DataFrame

    void describe(int col = -1) const;

    pair<size_t, size_t> shape() const;

    ColumnType unique(size_t col) const;

    size_t nunique(size_t col) const;

    // Statistical methods

    double sum(size_t col) const;

    double mean(size_t col) const;

    tuple<double, double, double> quartiles(size_t col) const;

    // Manipulate DataFrame

    DataFrame filterString(size_t col, string threshold, bool ifMinimumLimit = true) const;

    DataFrame filterDouble(size_t col, double threshold, bool ifMinimumLimit = true) const;

    void drop(size_t col);

    DataFrame concat(const DataFrame &other, bool axis = 0) const;

    void sort_values(size_t col, bool ascending = true);

    DataFrame apply(size_t col, function<variant<double, string>(const variant<double, string> &)> func) const;

    // Plotting methods

    void plot(size_t col) const;

    void hist(size_t col) const;

    void boxplot(size_t col) const;
};

#endif // DF_H
```

5.4 testing.h

```cpp
// author: Aryanthepain
#ifndef TESTING_H
#define TESTING_H
#include <bits/stdc++.h>
#include "df.h"
```

```
#include "Array.h"
using namespace std;

#endif
```

5.5 makefile

```
# \file
# \brief Makefile for building the project
#
# This Makefile compiles the source files located in the specified directories,
# links them into an executable, and provides targets for running the executable,
# cleaning up build artifacts, and performing Git operations.

# Compiler settings
CC = g++
CXXFLAGS = -Iinclude -Wall -Wextra -pedantic

# \brief Directories for source files and object files
# \var SRCDIRS
# Directories containing source files
SRCDIRS = array df

# \var OBJDIR
# Directory for storing object files
OBJDIR = obj

# \var EXECUTABLE
# The name of the final executable
EXECUTABLE = $(OBJDIR)/heyyo

# \brief Gather all source files from specified directories
# \var SOURCES
# List of source files to be compiled
SOURCES = EDA.cpp $(foreach dir, $(SRCDIRS), $(wildcard $(dir)/*.cpp))

# \var OBJECTS
# List of object files generated from the source files
OBJECTS = $(patsubst %.cpp, $(OBJDIR)/%.o, $(SOURCES))

# \brief Temporary files and plots that need to be deleted
# \var TEMPFILES
# List of temporary files to be cleaned
TEMPFILES = temp_data.txt

# \var PLOT_DIR
# Directory for storing plot files
PLOT_DIR = plots

# \brief Default target to build the executable
all: $(EXECUTABLE)
    @echo "Build complete. Executable: $(EXECUTABLE)"

# \brief Target to run the executable
run: all
    time ./$(EXECUTABLE)

# \brief Link object files to create the executable
# \param $(EXECUTABLE) The name of the executable to create
$(EXECUTABLE): $(OBJECTS)
    $(CC) $(OBJECTS) -o $@
    @echo "Linking complete. Created executable: $@"

# \brief Compile source files into object files
# \param $(OBJDIR)/%.o The object file to be created
# \param %.cpp The source file to compile
$(OBJDIR)/%.o: %.cpp | $(OBJDIR)
    @mkdir -p $(@D)
    $(CC) $(CXXFLAGS) -c $< -o $@
    @echo "Compiled: $< -> $@"

# \brief Ensure the object directory exists
$(OBJDIR):
    @mkdir -p $(OBJDIR)
    @echo "Created directory: $(OBJDIR)"

# \brief Git operations for version control
# \var branch
# The branch to push changes to (default: main)
branch?=main

# \var message
# The commit message (default: current date and time)
```

```
message?=$(shell date '+%d-%m-%Y %H:%M:%S')

# \brief Perform Git operations: add, commit, and push
git:
    git add .
    git commit -m "$(message)"
    git push origin $(branch)

# \brief Push changes to the remote Git repository
push:
    git push origin $(branch)

# \brief Clean up build artifacts
clean:
    rm -rf $(OBJDIR) $(PLOT_DIR)/* $(TEMPFILES)
    @echo "Cleaned up generated files."

# \brief Declare phony targets
.PHONY: all run git push clean
```

Index