

Numpy and Pandas in C++

Generated by Doxygen 1.12.0

1 Introduction	1
1.1 About Me	1
1.2 Compilation and Running Instructions	1
1.3 Features of the Libraries	1
1.4 Project Objective	2
1.5 Author	2
2 Class Index	3
2.1 Class List	3
3 File Index	5
3.1 File List	5
4 Class Documentation	7
4.1 Array< T > Class Template Reference	7
4.1.1 Detailed Description	9
4.1.2 Constructor & Destructor Documentation	9
4.1.2.1 Array() [1/2]	9
4.1.2.2 Array() [2/2]	9
4.1.3 Member Function Documentation	9
4.1.3.1 add()	9
4.1.3.2 arange()	10
4.1.3.3 concatenate()	10
4.1.3.4 cos()	11
4.1.3.5 count()	11
4.1.3.6 cumprod()	11
4.1.3.7 cumsum()	11
4.1.3.8 divide()	11
4.1.3.9 exp()	12
4.1.3.10 getData()	12
4.1.3.11 linspace()	12
4.1.3.12 log()	13
4.1.3.13 max()	13
4.1.3.14 mean()	13
4.1.3.15 min()	13
4.1.3.16 multiply()	13
4.1.3.17 ones()	14
4.1.3.18 operator[]() [1/2]	14
4.1.3.19 operator[]() [2/2]	14
4.1.3.20 power()	15
4.1.3.21 quartiles()	15
4.1.3.22 randomRand()	15
4.1.3.23 sample()	15

4.1.3.24 search()	16
4.1.3.25 sin()	16
4.1.3.26 size()	16
4.1.3.27 slice()	16
4.1.3.28 split()	17
4.1.3.29 std()	17
4.1.3.30 subtract()	17
4.1.3.31 sum()	18
4.1.3.32 unique()	18
4.1.3.33 var()	18
4.1.3.34 zeros()	18
4.2 DataFrame Class Reference	19
4.2.1 Detailed Description	20
4.2.2 Constructor & Destructor Documentation	21
4.2.2.1 DataFrame() [1/2]	21
4.2.2.2 DataFrame() [2/2]	21
4.2.3 Member Function Documentation	21
4.2.3.1 addColumn() [1/3]	21
4.2.3.2 addColumn() [2/3]	21
4.2.3.3 addColumn() [3/3]	22
4.2.3.4 apply()	22
4.2.3.5 boxplot()	22
4.2.3.6 concat()	22
4.2.3.7 copy()	23
4.2.3.8 describe()	23
4.2.3.9 drop()	23
4.2.3.10 filterDouble()	23
4.2.3.11 filterString()	24
4.2.3.12 getColumns()	24
4.2.3.13 getIndex()	24
4.2.3.14 head()	24
4.2.3.15 hist()	25
4.2.3.16 iloc()	25
4.2.3.17 loc()	25
4.2.3.18 mean()	25
4.2.3.19 nunique()	26
4.2.3.20 plot()	26
4.2.3.21 quartiles()	26
4.2.3.22 searchColumnByRow()	26
4.2.3.23 searchRowByColumn()	27
4.2.3.24 shape()	27
4.2.3.25 sort_values()	27

4.2.3.26 sum()	27
4.2.3.27 tail()	28
4.2.3.28 to_csv()	28
4.2.3.29 unique()	28
5 File Documentation	29
5.1 EDA.cpp	29
5.2 Array.h	30
5.3 df.h	31
5.4 testing.h	32
5.5 makefile	33
Index	35

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:

[Array< T >](#)

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

7

[DataFrame](#)

A class representing a [DataFrame](#), which can hold multiple columns of data

19

Chapter 3

File Index

3.1 File List

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

EDA.cpp	29
makefile	33
include/Array.h	30
include/df.h	31
include/testing.h	32

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

<i>T</i>	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

<i>data</i>	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

<i>other</i>	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

<i>start</i>	The starting value.
<i>end</i>	The ending value.
<i>step</i>	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

<i>a</i>	The first array.
<i>b</i>	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

<i>value</i>	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

<i>other</i>	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

<i>start</i>	The starting value.
<i>end</i>	The ending value.
<i>num</i>	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

<i>other</i>	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

<i>size</i>	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

<i>index</i>	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

<i>index</i>	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

<i>exponent</i>	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

<i>size</i>	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

<i>sampleSize</i>	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

<i>value</i>	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

<i>start</i>	The starting index of the slice.
<i>end</i>	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

<i>a</i>	The array to split.
<i>num</i>	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

<i>other</i>	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

<i>size</i>	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 > &columnNames)
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

<i>inputData</i>	A 2D vector containing the data for the DataFrame .
<i>colNames</i>	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

<i>csvFilePath</i>	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

<i>name</i>	The name of the new column.
<i>data</i>	The <code>Array<double></code> 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

<i>name</i>	The name of the new column.
<i>data</i>	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

<i>name</i>	The name of the new column.
<i>data</i>	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

<i>col</i>	The index of the column to apply the function to.
<i>func</i>	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

<i>col</i>	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

<i>other</i>	The DataFrame to concatenate.
<i>axis</i>	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

<i>col</i>	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

<i>col</i>	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

<i>col</i>	The index of the column to filter.
<i>threshold</i>	The threshold value for filtering.
<i>ifMinimumLimit</i>	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

<i>col</i>	The index of the column to filter.
<i>threshold</i>	The threshold value for filtering.
<i>ifMinimumLimit</i>	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

<i>n</i>	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

<i>col</i>	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

<i>row</i>	The row index.
<i>col</i>	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

<i>rowLabel</i>	The label of the row.
<i>colLabel</i>	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

<i>col</i>	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

<i>col</i>	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

<i>col</i>	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

<i>col</i>	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

<i>rowIndex</i>	The index of the row to search.
<i>value</i>	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

<i>colLabel</i>	The label of the column to search.
<i>value</i>	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

<i>col</i>	The index of the column to sort by.
<i>ascending</i>	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

<i>col</i>	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

<i>n</i>	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

<i>filePath</i>	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

<i>col</i>	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

```
00001 // author: Aryanthepain
00002 #include "include/testing.h"
00003
00013 int main()
00014 {
00015     // Load the dataframe from the CSV file
00016     DataFrame df("mpg.csv");
00017
00018     // Display the first 5 rows of the dataset
00019     cout << "First 5 rows of the dataset:" << endl;
00020     df.head();
00021     cout << endl;
00022
00023     // Display dataset information
00024     cout << "Dataset Info:" << endl;
00025     cout << "Shape of the dataset:" << df.shape().first << " X " << df.shape().second << "\n"
00026         << endl;
00027     df.describe();
00028
00029     // Analyze the 'name' column
00030     cout << "Analyzing the name column:\n"
00031         << endl;
00032     size_t nameColumn = 8; // Index of the 'name' column
00033     df.describe(nameColumn);
00034     cout << "All of the names should be common, however they are not.\n"
00035         << "This is because there are multiple entries for a car with multiple model years\n"
00036         << endl;
00037
00038     // Display unique values in the 'origin' column
00039     cout << "\nUnique values in 'origin' column:" << endl;
00040     size_t originColumn = 7; // Index of the 'origin' column
00041     auto uniqueOrigin = df.unique(originColumn);
00042     auto originData = get<Array<string>>(uniqueOrigin);
00043     originData.print();
00044
00045     // Filter rows where 'origin' is 'USA'
00046     size_t mpgColumn = 0; // Index of the 'mpg' column
00047     for (size_t i = 0; i < originData.size(); i++)
00048     {
00049         string targetOrigin = originData[i];
00050         cout << "\nTop Cars in '" << targetOrigin << "':" << endl;
00051         DataFrame originFilteredDf = df.filterString(originColumn, targetOrigin);
00052         originFilteredDf.sort_values(mpgColumn);
00053         originFilteredDf.head();
00054     }
00055
00056     // Perform statistical analysis for the 'mpg' column
00057     cout << "\nStatistical analysis of 'mpg' column:" << endl;
00058     cout << "Sum: " << df.sum(mpgColumn) << endl;
00059     cout << "Mean: " << df.mean(mpgColumn) << endl;
00060
00061     auto [q1, q2, q3] = df.quantiles(mpgColumn);
00062     cout << "Quartiles - Q1: " << q1 << ", Q2 (Median): " << q2 << ", Q3: " << q3 << endl;
00063
00064     // Sorting by 'mpg' column in descending order
00065     cout << "\nSorting by 'mpg' column (descending):" << endl;
00066     DataFrame sortedDf = df.copy();
00067     sortedDf.sort_values(mpgColumn, false); // Sort in descending order
```

```

00068     sortedDf.head();
00069
00070     // Filter rows where 'mpg' > 30
00071     double mpgThreshold = 30.0;
00072     cout << "\nFiltering rows where 'mpg' > " << mpgThreshold << ":" << endl;
00073     DataFrame filteredDf = df.filterDouble(mpgColumn, mpgThreshold, true);
00074     filteredDf.head();
00075
00076     // Flagging rows where 'mpg' > 30
00077     cout << "\nFlagging rows where 'mpg' > 30:" << endl;
00078     auto flaggedDf = df.apply(
00079         mpgColumn,
00080         [](const variant<double, string> &value) -> variant<double, string>
00081         {
00082             return get<double>(value) > 30.0 ? "High MPG" : "Low MPG";
00083         });
00084     flaggedDf.head();
00085
00086     // Plotting the distribution of the 'mpg' column
00087     cout << "\nPlotting 'mpg' column distribution:" << endl;
00088     df.hist(mpgColumn);
00089
00090     // Line plot for the sorted 'mpg' column
00091     cout << "\nLine plot for sorted 'mpg' column:" << endl;
00092     sortedDf.plot(mpgColumn);
00093
00094     return 0;
00095 }

```

5.2 Array.h

```

00001 // author: Aryanthepain
00002 #ifndef ARRAY_H
00003 #define ARRAY_H
00004 #include <bits/stdc++.h>
00005 using namespace std;
00006
00012 template <typename T>
00013 class Array
00014 {
00015 private:
00016     vector<T> data;
00017
00018     // Helper functions
00019     Array<T> elementWiseOperation(const Array<T> &other, std::function<T(T, T)> op) const;
00020     Array<T> elementWiseOperation(std::function<T(T)> op) const;
00021
00022 public:
00023     // Constructors
00027     Array() : data() {}
00028
00034     Array(const vector<T> &data) : data(data) {}
00035
00042     static Array<T> zeros(size_t size);
00043
00050     static Array<T> ones(size_t size);
00051
00060     static Array<T> arange(double start, double end, double step);
00061
00070     static Array<T> linspace(double start, double end, size_t num);
00071
00072     // Access methods
00079     T &operator[](size_t index);
00080
00087     const T &operator[](size_t index) const;
00088
00092     void print() const;
00093
00099     vector<T> getData() const;
00100
00107     size_t search(const T &value) const;
00108
00114     size_t size() const;
00115
00116     // Array Manipulation
00124     static Array<T> concatenate(const Array<T> &a, const Array<T> &b);
00125
00133     static vector<Array<T>> split(const Array<T> &a, size_t num);
00134
00141     static Array<T> randomRand(size_t size);
00142
00146     void sort();
00147

```

```

00154     Array<T> sample(size_t sampleSize) const;
00155
00163     Array<T> slice(size_t start, size_t end) const;
00164
00170     Array<T> unique() const;
00171
00178     size_t count(const T &value) const;
00179
00180     // Mathematical Operations for double
00187     Array<double> add(const Array<double> &other) const;
00188
00195     Array<double> subtract(const Array<double> &other) const;
00196
00203     Array<double> multiply(const Array<double> &other) const;
00204
00211     Array<double> divide(const Array<double> &other) const;
00212
00219     Array<double> power(double exponent) const;
00220
00226     double sum() const;
00227
00233     double mean() const;
00234
00240     double std() const;
00241
00247     double var() const;
00248
00254     Array<double> cumsum() const;
00255
00261     Array<double> cumprod() const;
00262
00268     Array<double> sin() const;
00269
00275     Array<double> cos() const;
00276
00282     Array<double> exp() const;
00283
00289     Array<double> log() const;
00290
00296     std::tuple<double, double, double> quartiles() const;
00297
00303     double max() const;
00304
00310     double min() const;
00311 };
00312
00313 #endif // ARRAY_H

```

5.3 df.h

```

00001 // author: Aryanthepain
00002 #ifndef DF_H
00003 #define DF_H
00004
00005 #include <bits/stdc++.h>
00006 #include "Array.h"
00007 using namespace std;
00008
00009 // Define a variant type for the DataFrame columns
00010 using ColumnType = variant<Array<double>, Array<string>>;
00011
00018 class DataFrame
00019 {
00020 private:
00021     vector<ColumnType> columns;
00022     vector<string> columnNames;
00023     vector<size_t> indexLabels;
00024
00025     // Helper functions
00026     void describeNumericColumn(size_t index) const;
00027     void describeStringColumn(size_t index) const;
00028     bool OutOfBounds(size_t num) const;
00029     DataFrame concatColumns(const DataFrame &other) const;
00030     DataFrame concatRows(const DataFrame &other) const;
00031
00032 public:
00033     // Constructors
00034
00038     DataFrame();
00039
00046     DataFrame(const vector<vector<variant<double, string>>> &inputData,
00047               const vector<string> &colNames);
00048

```

```

00054     DataFrame(const string &csvFilePath);
00055
00062     void addColumn(const string &name, const ColumnType &data);
00063
00070     void addColumn(const string &name, const Array<double> &data);
00071
00078     void addColumn(const string &name, const Array<string> &data);
00079
00080     // Access methods
00081
00085     void print() const;
00086
00092     void head(size_t n = 5) const;
00093
00099     void tail(size_t n = 5) const;
00100
00106     vector<string> getColumns() const;
00107
00113     vector<size_t> getIndex() const;
00114
00120     DataFrame copy() const;
00121
00129     variant<double, string> iloc(size_t row, size_t col) const;
00130
00138     variant<double, string> loc(size_t rowLabel, const string &colLabel) const;
00139
00147     int searchRowByColumn(const string &colLabel, const variant<double, string> &value) const;
00148
00156     int searchColumnByRow(size_t rowIndex, const variant<double, string> &value) const;
00157
00163     void to_csv(const string &filePath) const;
00164
00165     // Describe the DataFrame
00166
00172     void describe(int col = -1) const;
00173
00179     pair<size_t, size_t> shape() const;
00180
00187     ColumnType unique(size_t col) const;
00188
00195     size_t nunique(size_t col) const;
00196
00197     // Statistical methods
00198
00205     double sum(size_t col) const;
00206
00213     double mean(size_t col) const;
00214
00221     tuple<double, double, double> quartiles(size_t col) const;
00222
00223     // Manipulate DataFrame
00224
00233     DataFrame filterString(size_t col, string threshold, bool ifMinimumLimit = true) const;
00234
00243     DataFrame filterDouble(size_t col, double threshold, bool ifMinimumLimit = true) const;
00244
00250     void drop(size_t col);
00251
00259     DataFrame concat(const DataFrame &other, bool axis = 0) const;
00260
00267     void sort_values(size_t col, bool ascending = true);
00268
00276     DataFrame apply(size_t col, function<variant<double, string>(const variant<double, string> &)>
func) const;
00277
00278     // Plotting methods
00279
00285     void plot(size_t col) const;
00286
00292     void hist(size_t col) const;
00293
00299     void boxplot(size_t col) const;
00300 };
00301
00302 #endif // DF_H

```

5.4 testing.h

```

00001 // author: Aryanthepain
00002 #ifndef TESTING_H
00003 #define TESTING_H
00004 #include <bits/stdc++.h>
00005 #include "df.h"

```

```

00006 #include "Array.h"
00007 using namespace std;
00008
00009 #endif

```

5.5 makefile

```

00001 # \file
00002 # \brief Makefile for building the project
00003 #
00004 # This Makefile compiles the source files located in the specified directories,
00005 # links them into an executable, and provides targets for running the executable,
00006 # cleaning up build artifacts, and performing Git operations.
00007
00008 # Compiler settings
00009 CC = g++
00010 CXXFLAGS = -Iinclude -Wall -Wextra -pedantic
00011
00012 # \brief Directories for source files and object files
00013 # \var SRCDIRS
00014 # Directories containing source files
00015 SRCDIRS = array df
00016
00017 # \var OBJDIR
00018 # Directory for storing object files
00019 OBJDIR = obj
00020
00021 # \var EXECUTABLE
00022 # The name of the final executable
00023 EXECUTABLE = $(OBJDIR)/heyyo
00024
00025 # \brief Gather all source files from specified directories
00026 # \var SOURCES
00027 # List of source files to be compiled
00028 SOURCES = EDA.cpp $(foreach dir, $(SRCDIRS), $(wildcard $(dir)/*.cpp))
00029
00030 # \var OBJECTS
00031 # List of object files generated from the source files
00032 OBJECTS = $(patsubst %.cpp, $(OBJDIR)/%.o, $(SOURCES))
00033
00034 # \brief Temporary files and plots that need to be deleted
00035 # \var TEMPFILES
00036 # List of temporary files to be cleaned
00037 TEMPFILES = temp_data.txt
00038
00039 # \var PLOT_DIR
00040 # Directory for storing plot files
00041 PLOT_DIR = plots
00042
00043 # \brief Default target to build the executable
00044 all: $(EXECUTABLE)
00045     @echo "Build complete. Executable: $(EXECUTABLE)"
00046
00047 # \brief Target to run the executable
00048 run: all
00049     time ./$(EXECUTABLE)
00050
00051 # \brief Link object files to create the executable
00052 # \param $(EXECUTABLE) The name of the executable to create
00053 $(EXECUTABLE): $(OBJECTS)
00054     $(CC) $(OBJECTS) -o $@
00055     @echo "Linking complete. Created executable: $@"
00056
00057 # \brief Compile source files into object files
00058 # \param $(OBJDIR)/%.o The object file to be created
00059 # \param %.cpp The source file to compile
00060 $(OBJDIR)/%.o: %.cpp | $(OBJDIR)
00061     @mkdir -p $(@D)
00062     $(CC) $(CXXFLAGS) -c $< -o $@
00063     @echo "Compiled: $< -> $@"
00064
00065 # \brief Ensure the object directory exists
00066 $(OBJDIR):
00067     @mkdir -p $(OBJDIR)
00068     @echo "Created directory: $(OBJDIR)"
00069
00070 # \brief Git operations for version control
00071 # \var branch
00072 # The branch to push changes to (default: main)
00073 branch?=main
00074
00075 # \var message
00076 # The commit message (default: current date and time)

```

```
00077 message?=$(shell date '+%d-%m-%Y %H:%M:%S')
00078
00079 # \brief Perform Git operations: add, commit, and push
00080 git:
00081     git add .
00082     git commit -m "$(message)"
00083     git push origin $(branch)
00084
00085 # \brief Push changes to the remote Git repository
00086 push:
00087     git push origin $(branch)
00088
00089 # \brief Clean up build artifacts
00090 clean:
00091     rm -rf $(OBJDIR) $(PLOT_DIR)/* $(TEMPFILES)
00092     @echo "Cleaned up generated files."
00093
00094 # \brief Declare phony targets
00095 .PHONY: all run git push clean
```


Index

- add
 - Array< T >, [9](#)
- addColumn
 - DataFrame, [21](#), [22](#)
- apply
 - DataFrame, [22](#)
- arange
 - Array< T >, [10](#)
- Array
 - Array< T >, [9](#)
- Array< T >, [7](#)
 - add, [9](#)
 - arange, [10](#)
 - Array, [9](#)
 - concatenate, [10](#)
 - cos, [10](#)
 - count, [11](#)
 - cumprod, [11](#)
 - cumsum, [11](#)
 - divide, [11](#)
 - exp, [12](#)
 - getData, [12](#)
 - linspace, [12](#)
 - log, [12](#)
 - max, [13](#)
 - mean, [13](#)
 - min, [13](#)
 - multiply, [13](#)
 - ones, [14](#)
 - operator[], [14](#)
 - power, [15](#)
 - quartiles, [15](#)
 - randomRand, [15](#)
 - sample, [15](#)
 - search, [16](#)
 - sin, [16](#)
 - size, [16](#)
 - slice, [16](#)
 - split, [17](#)
 - std, [17](#)
 - subtract, [17](#)
 - sum, [18](#)
 - unique, [18](#)
 - var, [18](#)
 - zeros, [18](#)
- boxplot
 - DataFrame, [22](#)
- concat
- DataFrame, [22](#)
- concatenate
 - Array< T >, [10](#)
- copy
 - DataFrame, [23](#)
- cos
 - Array< T >, [10](#)
- count
 - Array< T >, [11](#)
- cumprod
 - Array< T >, [11](#)
- cumsum
 - Array< T >, [11](#)
- DataFrame, [19](#)
 - addColumn, [21](#), [22](#)
 - apply, [22](#)
 - boxplot, [22](#)
 - concat, [22](#)
 - copy, [23](#)
 - DataFrame, [21](#)
 - describe, [23](#)
 - drop, [23](#)
 - filterDouble, [23](#)
 - filterString, [24](#)
 - getColumns, [24](#)
 - getIndex, [24](#)
 - head, [24](#)
 - hist, [25](#)
 - iloc, [25](#)
 - loc, [25](#)
 - mean, [25](#)
 - nunique, [26](#)
 - plot, [26](#)
 - quartiles, [26](#)
 - searchColumnByRow, [26](#)
 - searchRowByColumn, [27](#)
 - shape, [27](#)
 - sort_values, [27](#)
 - sum, [27](#)
 - tail, [28](#)
 - to_csv, [28](#)
 - unique, [28](#)
- describe
 - DataFrame, [23](#)
- divide
 - Array< T >, [11](#)
- drop
 - DataFrame, [23](#)

- exp
 - Array< T >, [12](#)
- filterDouble
 - DataFrame, [23](#)
- filterString
 - DataFrame, [24](#)
- getColumns
 - DataFrame, [24](#)
- getData
 - Array< T >, [12](#)
- getIndex
 - DataFrame, [24](#)
- head
 - DataFrame, [24](#)
- hist
 - DataFrame, [25](#)
- iloc
 - DataFrame, [25](#)
- include/Array.h, [30](#)
- include/df.h, [31](#)
- include/testing.h, [32](#)
- Introduction, [1](#)
- linspace
 - Array< T >, [12](#)
- loc
 - DataFrame, [25](#)
- log
 - Array< T >, [12](#)
- max
 - Array< T >, [13](#)
- mean
 - Array< T >, [13](#)
 - DataFrame, [25](#)
- min
 - Array< T >, [13](#)
- multiply
 - Array< T >, [13](#)
- nunique
 - DataFrame, [26](#)
- ones
 - Array< T >, [14](#)
- operator[]
 - Array< T >, [14](#)
- plot
 - DataFrame, [26](#)
- power
 - Array< T >, [15](#)
- quartiles
 - Array< T >, [15](#)
 - DataFrame, [26](#)
- randomRand
 - Array< T >, [15](#)
- sample
 - Array< T >, [15](#)
- search
 - Array< T >, [16](#)
- searchColumnByRow
 - DataFrame, [26](#)
- searchRowByColumn
 - DataFrame, [27](#)
- shape
 - DataFrame, [27](#)
- sin
 - Array< T >, [16](#)
- size
 - Array< T >, [16](#)
- slice
 - Array< T >, [16](#)
- sort_values
 - DataFrame, [27](#)
- split
 - Array< T >, [17](#)
- std
 - Array< T >, [17](#)
- subtract
 - Array< T >, [17](#)
- sum
 - Array< T >, [18](#)
 - DataFrame, [27](#)
- tail
 - DataFrame, [28](#)
- to_csv
 - DataFrame, [28](#)
- unique
 - Array< T >, [18](#)
 - DataFrame, [28](#)
- var
 - Array< T >, [18](#)
- zeros
 - Array< T >, [18](#)