Making the assignment

You should make this assignment **individually**. You are allowed and encouraged to discuss ideas necessary to solve the assignment with your peers, but you should write the code by yourself. We will employ fraud checking software specifically created for programming classes such as this.

In case we suspect fraud, we can invite you to explain your code. Due to university policy cases of (suspected) fraud must be reported to the exam committee. The exam committee will then decide on possible disciplinary actions.

If you prefer that your assignment is graded before the exam of the course, the deadline for this assignment is on **Thursday, 15th of October 2020, 23:00, CEST**. If you desire this, make sure you hand in your work to the Early Deadline version of the assignment on Codegrade, **and also** send an e-mail to `eb-feb22012@ese.eur.nl` to confirm that you wish your work to be graded early. **If you forget to send an e-mail, your work will not be graded early!**. If you don't care about early grading, the final deadline is on **Monday, the 25th of October 2020, 23:00 CEST**. You should then hand in your work in the regular version of the assignment.

Grade and Scores

The grade of your assignment will depend on two important criteria:

1. Correctness: does your code work according to the specification in the assignment?
2. Code Quality and Style: is your code well written and understandable?

This assignment contains **four possible extensions**. You can **choose** which extensions you want to implement, but all work you put into the extensions can only provide you with at most 2 points. It is not required to implement all extensions to obtain a high grade.

Part of Assignment	Points
Data Frame Core	2
DataVector Methods	1
Restructuring Methods	2
Analysis Methods	1
Subtotal	6
Extension 1 – Random Data	1
Extension 2 – Statistics Part 1	1
Extension 2 – Statistics Part 2	1
Extension 3 – Plotting Scatterplot	1
Extension 3 – Plotting Histogram	1
Extension 4 – Excel IO (Reading)	1
Extension 4 – Excel IO (Writing)	1
Subtotal / Maximum	2
Code Quality and Style	2
Total	10

When we ask you to implement a function using a **specific** algorithm, you will get no points if you use a different algorithm.

Handing in your assignment

- Any scores generated by **Codegrade** only give an indication of your possible grade. Final grades will only be given after your code has been checked manually, and may differ from the preliminary automated feedback provided. In some cases there may be additional testcases that are only executed after you hand in your final code.
- Deliberate attempts to trick **Codegrade** into accepting answers that are clearly not a solution to the assignments will be considered *fraud*.

Code Quality and Style

Part of your grade will also depend on the quality and style of your code. **Codegrade** will check a number of rules automatically and compute a tentative score based on these rules. **Note that in the actual assignments, this score will likely be adjusted based upon manual inspection by human graders**. The automated tools do not find all issues, in and rare cases the automated graders may be too strict.

The rules applied by the automated grader come in four categories: *mandatory rules*, *important rules*, *slopy code rules* and *layout, style and naming rules*. This score is only an indication and is computed as follows:

- If any *mandatory rules* are broken, you get no points for code quality and style.
- If no *important rules* and *sloppy code rules* are broken, you get 1.5 points. For each *important rule* you break, 0.2 points are deducted. For each *sloppy code rule* you break, 0.1 points are deducted. It doesn't matter how often a particular rule is broken.
- If no *layout, style and naming rules* are broken, you get 0.5 points. For the first three violations, no points are deducted, but for each violation beyond that 0.1 point is deducted.
- Note that this score is only an indication of your code quality. Manual grading will always overrule the score indicated by Codegrade!

The detailed rules are explained in the "*Code quality and style guide*" of this course. They can be found on `https://erasmusuniversityautolab.github.io/FEB22012-StyleGuide/` There is a separate document on implemeting correct Javadoc comments. This can be found at: `https://erasmusuniversityautolab.github.io/FEB22012-StyleGuide/javadoc.html`

General Remarks

- The lecture slides usually contain a number of useful hints and examples for these assignments. If you get stuck, always check if something useful was discussed during the lectures. If you have question, feel free to contact us by e-mail.
- You can write your Java code with any editor your like. In the computer labs you will have access to BlueJ and Eclipse. We encourage you to use a good IDE, such as Eclipse, Netbeans or IntelliJ, as those have better features to help you with simple tasks and warn you in case of possible mistakes.

The Assignment

In this assignment you will create a small library that implements a *data frame*, a common type of data structure popular in *data science*. Data frames are a core concept in the R language and environment for statistical computing. In the popular Python programming language, pandas is a very popular library that implements a data frame implementation. Within the Java ecosystem, there are a number of projects that also provide data frame functionality, for example Morpheus and Tablesaw.

Since data frames are a helpful tool in data analytics, it is useful to study how they work internally. It is not a goal of this assignment to implement the most efficient or feature-rich data frame library possible. As a first step, you will implement the basic functionality of a data frame. Then you will implement a number of methods that can be used to manipulate data frames. Finally, you can choose from a number of extensions based on the use of additional Java libraries.

Design of the Data Frame

A data frame is a two-dimensional table of data, similar to a matrix. But where matrices have integer indices associated with both rows and columns, our data frame will have *names* of type `String` associated with the columns. An example of data frame with three columns and two rows is given in Table 1. Data scientists find it often convenient to talk about the columns as *variables* that represent all kinds of interesting quantities, e.g. profit, revenue, age, income, etcetera. The rows typically represent observations where the different variables were measured jointly. Under this interpretation, the example in Table 1 represents the measurements of the revenue and costs of a company in two distinct years.

In the data frame you need to implement for this assignment, we assume that the dimensions (i.e. number of rows and columns) of the data frame are immutable: once a `DataFrame` object is successfully constructed, its dimensions will never change. The values of the entries of the data frame are mutable: it is possible to replace values in an existing data frame by new values. This means that all operations that may result in a data frame of different dimensions needs to construct a new `DataFrame` object and return it, rather than mutate the current object.

At the heart of our library, there will be two interfaces: `DataFrame<E>` and `DataVector<E>`. The `DataFrame` interface is designed to represent any kind of data frame, agnostic for the type of data stored in it. The `DataVector` interface is designed to represent a single vector of data. Within this interface, no distinction is made between row or column vectors: the `DataVector` has a certain length, has a name associated which each entry (which is useful when it represents a row of data in the data frame) and also has a name by itself (which is useful when it represents a column of data in the data frame).

For the implementation of this assignment you need to create classes that implement `DataFrame<Double>` and `DataVector<Double>`. That means that the data stored in your `DataFrame` and `DataVector` objects will always be of type. To make life slightly more convenient, a number of `default` methods were implemented in both those interfaces. First, the is the `print()` method that prints a `DataFrame` to the console, with a column width of 12 characters wide. This is performed by a call to the default method `formatMatrix(int colWidth)`, that generates a `String` representation of the matrix, where the width of the columns is set to the argument. The other default methods will be discussed when the methods they related to introduced.

Table 1: An example data frame with three columns and two rows

year	revenue	costs
2017	10632.83	37816.80
2018	85216.33	31863.73

Test Data

In the example code we provide with this document, we do not repeat the same data every time. To produce a data frame, we will call the function `testDataFrame()`, that produces a new `DataFrame` as follows:

```
1 public static DataFrame<Double> testDataFrame() {
2    List<String> colNames = Arrays.asList("year", "revenue", "costs");
3    double [][] data = {{ 2015, 70021.35, 25071.12 },
4                         { 2016, 67008.12, 108632.80 },
5                         { 2017, 10632.83, 37816.8 },
6                         { 2018, 85216.33, 31863.73 }};
7    return new DoubleDataFrame(colNames, data);
8 }
```

Maven Dependency Management

For the extensions of this assignment, you should make use of a number of Java libraries that are not included with Java. It is recommend to download the `week_67_student_package.zip` file, and open the data within this zip file as a *Maven project*.

With **IntelliJ**, you can unzip the package and use *File* and then *Open...* and select the folder where you unzippen the package.

In **Eclipse**, this can be done via *Import...* options in the *File* menu. You should open the *Maven* category and select the option *Existing Maven project*. Click the *Browse* button and select the folder that contains the `pom.xml` included in the zip package. For the extension assignments, you need to include additional dependencies that you should use. The dependencies that you are allowed to use in this assignment are:

- Apache Math Commons version 3.6.1. Javadocs can be viewed here. The dependency details are: **groupId**: `org.apache.commons`, **artifactId**: `commons-math3`, **version**: `3.6.1`.
- Apache POI 4.1.2. Javadocs can be be viewed here. To use this library you need to include two dependencies that both have **groupId**: `org.apache.poi` and **version**: `4.1.2`. The **artifactId**s are: `poi` and `poi-ooxml`.
- XChart 3.6.5. Javadoc links: important classes, all classes. The Github page is also helpful with some examples. The dependency details are: **groupId**: `org.knowm.xchart`, **artifactId**: `xchart` and **version**: `3.6.5`.

Note: when you import a Maven project in Eclipse, the project will live in the folder your important, and changes you make to the file will occur there. Eclipse does not copy the project to your workspace folder, so be aware of this.

Maven Troubleshooting

In a few cases students have issues importing the maven project. The maven project is only required for the extension part of the assignment. If you start very close to the deadline and have issues with maven, you should consider not making an extension, and just work in a plain Java project.

If Eclipse complains project "assignment3" already exists when you import the maven project, you have a project with that name already in your workspace. Delete that project from your workspace before you import, or switch to a new clean workspace before you do the import.

If you get long or strange maven related errors from Eclipse, it is likely there is an issue with downloading the packages. Make sure you have a stable network connection before you attempt to import the project, and that no firewalls are active that block network traffic from Eclipse. If you network works, you can try to do an update of your project, to let Eclipse attempt downloading the packages again. This is done by clicking with the right mouse button on the project root folder "assignment3", then in submenu "Maven" choose "Update Maven Project". In the dialog window make sure project assignment3 and the checkbox 'Force Update of Snapshots/Releases" are checked. Then click OK, and Eclipse should reattempt to download missing packages.

If updating the project does not help some of the packages may have become corrupt because of an incomplete download. In that case, you should search for the home folder of your user (e.g. `C:\Users\YourName` on Windows) and look for the `.m2` folder. You can delete the `repositories` folder from the `.m2` folder to remove everything that was previously downloaded by Maven. If you perform an update of your project afterward (see previous paragraph), it will make sure all packages are downloaded again. That should help in case some packages got corrupted.

If you can't get maven to work properly on your computer, feel free to send an e-mail to ebfeb22012@ese.eur.nl.

Data Frame Core Functionality

In the first part of this assignment, you need to implement the core functionality of the data frame. For this purpose, you must implement the class `DoubleDataFrame` with at least one constructor that accepts a `List<String>` that contains the names of the columns and two dimensional `double` array, that contains the data to be stored in the data frame. It is assumed that entry `data[i][j]` contains the element at the `i`th row and `j`th column.

```
1 public class DoubleDataFrame implements DataFrame<Double> {
2    public DoubleDataFrame(List<String> columnNames, double [][] data) { ... }
3    // Class implementation goes here
4 }
```

The internal representation that you use to store all data within your `DoubleDataFrame` is up to you. Possible choices are a `double[][]`, a `List<Map<String,Double>>`, a `Map<Integer,Map<String,Double>>` or perhaps even a combination of a `List<String>` for the column names and a `List<List<Double>>` for the actual data.

For the first part of the assignment, you need to implement the following four methods of the `DataFrame` interface correctly:

```
1 public int getRowCount();
2 public int getColumnCount();
3
4 public List<String> getColumnNames();
5
6 public void setValue(int rowIndex, String colName, E value);
7 public E getValue(int rowIndex, String colName);
```

The methods `getRowCount()` and `getColumnCount()` should return the dimensions of the data frame. The method `getColumnNames()` should return a `List<String>` with the names of the columns in the same order as they were provided when the data frame was created. This list should either be *unmodifiable*, or a copy of the original list, to avoid that changes to this list affect the internal structure of the data frame.

Finally, the methods `setValue` and `getValue` can be used to retrieve and change values of the data frame. In the case of the `DoubleDataFrame` the type associated with type variable `E` is `Double`. These methods should throw an `IndexOutOfBoundException` if an invalid row index is provided, or a `IllegalArgumentException` if a non-existing column name is provided.

Note: while you do not need to implement the most efficient data frame possible, the `getValue()` and `setValue()` should be efficient. It is not allowed to use an approach that searches through a `List` or array to find an index associated with a certain column name, such as the `List.indexOf` method or an explicit loop that does it. If you use an internal structure that needs you to convert the column name to a column index, you should use a `Map<String,Integer>` to do this conversion in a fast way.

Example: The following example code:

```
1 DataFrame<Double> df = testDataFrame();
2 System.out.println("Number of rows: "+df.getRowCount());
3 System.out.println("Number of columns: "+df.getColumnCount());
4 System.out.println(df.getValue(1, "year"));
5 df.print();
6 df.setValue(2, "revenue", 0d);
7 df.print();
```

should produce the following output.

```
Number of rows: 4
Number of columns: 3
2016.0
             year         revenue      costs
row_0        2015.0       70021.35     25071.12
row_1        2016.0       67008.12     108632.8
row_2        2017.0       10632.83     37816.8
row_3        2018.0       85216.33     31863.73

             year         revenue      costs
row_0        2015.0       70021.35     25071.12
row_1        2016.0       67008.12     108632.8
row_2        2017.0       0.0          37816.8
row_3        2018.0       85216.33     31863.73
```

Note: the `print()` method is a default method of the `DataFrame` interface and has already been implemented for you.

Speed test: the following code can be used to check the speed of your `getValue()` and `setValue()` methods.

```
int size = 10000;
double [][] data = new double[1][size];
List<String> header = new ArrayList<>(size);
for (int j=0; j < size; j++)
{
   data[0][j] = j;
   header.add("x_"+j);
}
DoubleDataFrame df = new DoubleDataFrame(header, data);
long time = System.currentTimeMillis();
for (int j=0; j < size; j++)
{
   df.getValue(0, header.get(j));
   df.setValue(0, header.get(j), 0d);
}
time = System.currentTimeMillis() - time;
System.out.println("Running time: "+time+"ms");
```

If this takes more than 100 milliseconds, you should work on the efficiency of your approach.

DataVector methods

As a next step, you should create a class that implements the `DataVector<Double>` interface and implement the methods in `DoubleDataFrame` that produce `DataVectors`. Our `DoubleDataFrame` should provide one or more `DataVector<Double>` objects when one of the following methods from the `DataFrame` interface are called:

```
1 public DataVector<E> getRow(int rowIndex);
2 public DataVector<E> getColumn(String colName);
3 public List<DataVector<E>> getRows();
4 public List<DataVector<E>> getColumns();
```

Note that it is allowed to let the `getRow` throws an `IndexOutOfBoundException` if an illegal index is provided, while the `getColumn` method is allowed to throw an `IllegalArgumentException` if the provided column name does not exist.

You can choose any name for the class you write to implement the `DataVector<Double>` interface, but we suggest `DoubleDataVector`, for example with the header:

```
1 public class DoubleDataVector implements DataVector<Double>
```

You choose design a constructor that can be used within your `DoubleDataFrame` to store the necessary data in the `DoubleDataVector`. A `DoubleDataVector` should implement the following methods:

```
1 public String getName();
2 public List<String> getEntryNames();
3 public E getValue(String entryName);
4 public List<E> getValues();
5 public Map<String,E> asMap();
```

What is returned by the methods depends on whether the `DataVector` represents a column or a row in the `DataFrame` that produced it.

If a `DataVector` is obtained for a **row**, the `getName()` method should return `"row_0"` if it represents the row with index 0, `"row_1"` if it represents the row with index 1, etcetera. The `getEntryNames()` method should return a list that contains the names of the column names of the original data frame (in the original order). The `getValue()` method should return the value associated with the column name that is passed as an argument. The method `getValues()` should return a `List` of all that are stored in this row in the original data frame. Finally, the method `asMap()` should provide a `Map` object that contains key-value pairs of the column names as they would be produced by the `getValue()` method. The result of the call `vec.asMap().get("someColumnName")` on a `DataVector<E> vec` should produce the same result as `vec.getValue("someColumnName")`.

If a `DataVector` is obtained for a **column**, the `getName()` method should return the name of the column. The `getEntryNames()` should a list with `"row_0"`, `"row_1`, ...`"row_n"` where `n` is the number of rows in the matrix. The method `getValue()` should associate `entryName` `"row_0"` with the first element in the column, `"row_1"` with the second row in the column, etcetera. The method `getValues()` should return a list with all the values in the column. Finally, the `asMap()` method should return the key-value pairs as they would be produced by the `getValue()` method, similar to the behavior for *row*-based `DataVector` objects.

Example: when the following code is executed:

```
1  DataFrame<Double> df = testDataFrame();
2  DataVector<Double> row = df.getRow(1);
3  System.out.println(row.getName());
4  System.out.println(row.getEntryNames());
5  System.out.println(row.getValue("costs").equals(df.getValue(1, "costs")));
6  System.out.println(row.getValues());
7  System.out.println(row.asMap());
8  System.out.println();
9
10 DataVector<Double> col = df.getColumn("costs");
11 System.out.println(col.getName());
12 System.out.println(col.getEntryNames());
13 System.out.println(col.getValue("row_1").equals(df.getValue(1, "costs")));
14 System.out.println(col.getValues());
15 System.out.println(col.asMap());
16 System.out.println();
17
18 System.out.println(df.getColumns().size() == df.getColumnCount());
19 System.out.println(df.getRows().size() == df.getRowCount());
20 for (DataVector<Double> vec : df)
21 {
22    System.out.println(vec.getName());
23 }
```

the following should be printed (be aware that the order of the key-value pairs of a map can be different, which is allowed):

```
row_1
[year, revenue, costs]
true
[2016.0, 67008.12, 108632.8]
{costs=108632.8, revenue=67008.12, year=2016.0}

costs
[row_0, row_1, row_2, row_3]
true
[25071.12, 108632.8, 37816.8, 31863.73]
{row_0=25071.12, row_3=31863.73, row_1=108632.8, row_2=37816.8}

true
true
row_0
row_1
row_2
row_3
```

Note that it is possible to use the *enhanced for loop* on your `DataFrame` because the `DataFrame` interface provides a default implementation of the `Iterable` interface. This default implementation iterations over the `DataVector` objects contained in the `List` returned by the `getRows()` method.

Restructuring Methods

As mentioned before, in this assignment we choose to consider the dimensions of a `DataFrame` object as immutable: once a `DataFrame` is created, we can not resize it. However, it is allowed to implement methods that construct a new `DataFrame` object, copy relevant data from our current `DataFrame` into the new object, and return the new `DataFrame` object. These kinds of operations, where we change the structure of the `DataFrame` are called *restructuring* operations. For this part of the assignment, you need to correctly need implement the following three methods of the `DataFrame` interface in the `DoubleDataFrame` class:

```
1 public DataFrame<E> expand(int additionalRows, List<String> newCols);
2 public DataFrame<E> project(Collection<String> retainColumns);
3 public DataFrame<E> select(Predicate<DataVector<E>> rowFilter);
```

The `expand` method should construct a bigger `DataFrame` object, where `additionalRows` extra rows are added to the bottom of the matrix and additional columns with the names defined in `newCols` are added to the right side of the `DataFrame`. The data currently stored in the smaller `DataFrame` object should be copied to the bigger object. The values associated with the new rows and columns should be 0.

The `project` method can be used to obtain a `DataFrame` object with fewer columns: only the columns with the names that are contained in the `Collection<String> retainColumns` should be contained in the resulting `DataFrame` object. Note that the relative order of the columns of the original `DataFrame` should be maintained. Since `retainColumns` may be an unordered `Set`, the order of the column names in `retainColumns` can not be relied upon.

The `select` method can be used to obtain a `DataFrame` with fewer rows. To determine which rows must be included in the output `DataFrame`, you should make use of the Predicate object called `rowFilter`. This `Predicate` has a method `test` that accepts a `DataVector` as input, and returns a boolean that indicates if the row associated with the `DataVector` object should be included in the output `DataFrame`.

Example: When the following code is executed:

```
1  DataFrame<Double> df = testDataFrame();
2  DataFrame<Double> bigger = df.expand(1, "profit", "loss");
3  bigger.print();
4  bigger.setValue(1, "costs", 0d);
5  System.out.println(!df.getValue(1, "costs").equals(0d));
6
7  DataFrame<Double> smaller1 = df.project("year", "costs");
8  smaller1.print();
9  smaller1.setValue(1, "costs", 0d);
10 System.out.println(!df.getValue(1, "costs").equals(0d));
11
12 DataFrame<Double> smaller2 = df.select(row -> !row.getValue("year").equals(2017d));
13 smaller2.print();
14 smaller2.setValue(1, "costs", 0d);
15 System.out.println(!df.getValue(1, "costs").equals(0d));
```

Note that in this example, the calls to `expand` and `project` occur via variants of `expand` and `project` that are provided as default implementations in the `DataFrame` interface. These default implementations wrap the arguments in a `List<String>` object and passes this object to the implementation you have to write.

When the example code runs the following should be printed:

```
            year        revenue     costs       profit      loss
row_0       2015.0      70021.35    25071.12    0.0         0.0
row_1       2016.0      67008.12    108632.8    0.0         0.0
row_2       2017.0      10632.83    37816.8     0.0         0.0
row_3       2018.0      85216.33    31863.73    0.0         0.0
row_4       0.0         0.0         0.0         0.0         0.0

true
            year        costs
row_0       2015.0      25071.12
row_1       2016.0      108632.8
row_2       2017.0      37816.8
row_3       2018.0      31863.73

true
            year        revenue     costs
row_0       2015.0      70021.35    25071.12
row_1       2016.0      67008.12    108632.8
row_2       2018.0      85216.33    31863.73

true
```

Analysis Methods

As a last step, you need to implement methods that are helpful in the analysis of the data set currently held by the data frame. The two analysis methods from the `DataFrame` interface you have to implement are:

```
1 public DataFrame<E> computeColumn(String columnName,
      Function<DataVector<E>,Double> function);
2 public DataVector<E> summarize(String name, BinaryOperator<E> summaryFunction);
```

The `computeColumn` method should produce a larger `DataFrame` object with the same number of rows, but one additional column with name `columnName`. The values stored in this column should be computed by the the `Function` object passed as the second argument. This `Function` has a method `apply` that accepts a `DataVector` as input, and produces the value of data stored in the `DataFrame` (which is `Double`) in our case. The method should `apply` the `Function` to each row in the original `DataFrame`, and store the output under the column that is being constructed.

The `summarize` method should produce a `DataVector` which holds the `name` argument as the name that is being returned by `DataVector.getName()`. Furthermore, this `DataVector` should hold aggregated values for every column in the original `DataFrame`. These aggregate values should be computed using the `BinaryOperator` argument. A `BinaryOperator<Double>` has a method `apply` that accepts two `Double` arguments, and return a single `Double` output. This operator should be first applied to the first and second element in the column, then be applied to the output of the first step and the third element, then be applied to the output of the second step and the fourth element, etcetera. You can either write a loop that does this manually, or make clever use of the `reduce` method from the `Stream` class.

Example: when you execute the following code

```
1  DataFrame<Double> df = testDataFrame();
2  Function<DataVector<Double>,Double> profitFunction;
3  profitFunction = row -> row.getValue("revenue") - row.getValue("costs");
4  DataFrame<Double> df2 = df.computeColumn("profit", profitFunction);
5  df2.print();
6
7  BinaryOperator<Double> sumOp = Double::sum;
8  DataVector<Double> dv = df2.summarize("sum", sumOp);
9  dv.print();
10 df2.summarize("max", Math::max).print();
11 df2.summarize("min", Math::min).print();
```

the following output is expected

```
            year        revenue      costs         profit
row_0       2015.0      70021.35     25071.12      44950.230000
row_1       2016.0      67008.12     108632.8      -41624.68000
row_2       2017.0      10632.83     37816.8       -27183.97
row_3       2018.0      85216.33     31863.73      53352.600000

sum         8066.0      232878.63    203384.45000  29494.180000
max         2018.0      85216.33     108632.8      53352.600000
min         2015.0      10632.83     25071.12      -41624.68000
```

Extension 1 – Generate Random Data

In this extension you work with some of the functionality available in the Apache Commons Math library. In particular the package `org.apache.commons.math3.distribution` (documentation) provides a number of useful classes to generate a `DataFrame` that is filled with data sampled from a given distribution.

You need to implement a class `RandomTools` that has the following public interface:

```
public RandomTools(RealDistribution distr) {...}
public DataFrame<Double> generate(long seed, int rows, List<String> colnames) {...}

// Utility methods that provide default distributions
public static RandomTools uniform(double lb, double ub) {...}
public static RandomTools gaussian(double mu, double sigma) {...}
public static RandomTools exponential(double mean) {...}
```

The constructor should accept any object of type `RealDistribution` (documentation) and store it in the `RandomTools` instance. When the `generate` method is then called on that `RandomTools` instance, it should set the random `seed` of that `RealDistribution` and then fill a `DataFrame` with `rows` rows and columns corresponding to the `colnames` list with `sample` data that is acquired from the `RealDistribution` object.

The class should also contain three static methods that construct a `RandomTools` object with certain distribution types. The `uniform` method should provide a `RandomTools` instance that uses a uniform distribution with lower-bound `lb` and upper-bound `ub`. The `gaussian` method should provide a `RandomTools` instance that uses a gaussian or normal distribution with mean `mu` and standard deviation `sigma`. The `exponential` method should provide a `RandomTools` instance that samples from an exponential distribution with mean `mean`. Note that the documentation of the `RealDistribution` interface contains a list of classes than implement various distributions. You should be able to find some classes you can use to help in implementing these static methods.

Example: if you run the following code

```
int rows = 10;
RandomTools rt = RandomTools.uniform(10, 20);
DataFrame<Double> df;
df = rt.generate(12345, rows, Arrays.asList("uniform1", "uniform2"));
rt = RandomTools.gaussian(0, 1);
df = df.concat(rt.generate(54321, rows, Arrays.asList("normal1", "normal2")));
rt = RandomTools.exponential(5);
df = df.concat(rt.generate(1337, rows, Arrays.asList("exponential")));
df.print();
```

it should produce the following output (or something similar)

Listing 1: Example random data

	uniform1	uniform2	normal1	normal2	exponential
row_0	15.205536706	11.752004338	0.4877097439	1.6460483766	1.0067010307
row_1	19.468181054	12.824796079	0.6168264465	0.8657733603	8.3139206690
row_2	11.698200571	10.619443985	-0.148842591	0.4027376041	3.8114547894
row_3	17.561438860	10.624647725	-0.733937611	0.9253232041	0.9869376771
row_4	14.181868249	11.597424697	-0.641166986	0.8604259673	0.2704712853
row_5	10.109201456	15.875627821	1.6100325183	1.4804334261	1.5238785432
row_6	16.919577952	18.557133663	-1.121124394	1.4778496350	1.9518087496
row_7	11.999833526	13.249142323	0.9811540682	-0.243741325	4.4602185170
row_8	14.694905619	16.025124173	0.2002095654	-0.648235107	1.7330072668
row_9	17.954139135	13.259821185	0.0303474047	-1.493674190	3.3773059197

Extension 2 – Statistics

In this extension you work with some of the functionality available in the Apache Commons Math library. You will provide an implementation of the `DataFrameStatistics` interface that provides easy access to a number of statistical methods of Apache Commons Math.

First, you need to override the following method of the `DataFrame` interface, as the default implementation throws an `UnsupportedOperationException`:

```
1 public DataFrameStatistics statistics();
```

When a user calls this method, they should get an object that implements the `DataFrameStatistics` interface, which contains the following methods:

```
1 // Part 1 methods
2 public double tTest(String var, double mu);
3 public double tTest(String var1, String var2);
4 public double pearsonsCorrelation(String var1, String var2);
5 public DescriptiveStatistics describe(String var);
6 // Part 2 method
7 public Map<String,Double> estimateLinearModel(String dep, List<String> indep);
```

Part 1

The first `tTest` method should perform a unpaired Student's t-test with the numbers stored in column with name `var` of the `DataFrame` that provided this `DataFrameStatistics` that tells if the observed values are different from a normal distribution around a given mean `mu`. The output should be the p-value of this test. The second `tTest` method should perform an unpaired t-test that tests if the observed values in column `var1` have a different mean than the observed values in column `var2`. For this part of the assignment, it is probably helpful to study the `TTest` class (documentation) or the `TestUtil` class (documentation) of the `org.apache.commons.math3.stat.inference` package.

The `pearsonsCorrelation` method should compute the *Pearson correlation coefficient* between the observations of the column with name `var1` and the column with name `var2` of the underlying `DataFrame` object. For this part of the assignment, it is probably helpful to study the `PearsonsCorrelation` class (documentation).

Finally, it should be possible to obtain a `DescriptiveStatistics` object for a given column in the underlying data frame. The `DescriptiveStatistics` (documentation) class contains methods that compute a number of statistics for a data vector. The `describe` method should feed the data of the column with name `var` to such an object, and return the result.

Example: note that the `DataFrame` returned by `sampledDataFrame()` is based on the sampled data presented in Listing 1. If you would run the following example code:

```
1 DataFrame<Double> df = sampledDataFrame();
2 df = df.computeColumn("neg", row -> -row.getValue("normal1"));
3 DataFrameStatistics stats = df.statistics();
4 System.out.println(stats.tTest("normal1", 0));
5 System.out.println(stats.tTest("normal1", "normal2"));
6 System.out.println(stats.pearsonsCorrelation("normal1", "neg"));
7 System.out.println(stats.pearsonsCorrelation("uniform1", "normal1"));
8 System.out.println(stats.describe("exponential"));
```

The printed output should be as follows:

```
0.6391596704986711
0.35395105736406973
-0.9999999999999999
-0.47751103680106
DescriptiveStatistics:
n: 10
min: 0.27
max: 8.313
mean: 2.743
std dev: 2.3774701774037976
median: 1.842
skewness: 1.5524798622096312
kurtosis: 2.7132073055779546
```

Part 2

In this part of the assignment we will make use of the Apache Math Commons library to estimate a linear model, with a single *dependent* variable, and one or multiple *independent* variables. You should make use of the `OLSMultipleLinearRegression` class (documentation) to perform this task. When the `estimateLinearModel` linear model method is called, we assume the column with name `dep` contains observations of the dependent random variable Y, and that the columns with their names in list `indep` contain observations from the independent random variables X_1, X_2, etcetera. The `OLSMultipleLinearRegression` should then be used to estimate the following model:

$$Y = \beta_1 X_1 + \beta_2 X_2 + \cdots + c + \epsilon \tag{1}$$

where ϵ is the error term and c is the intercept. You should prepare the data, feed it to an object of `OLSMultipleLinearRegression` in the correct way, and use that object to to compute the intercept c and the β coefficients. The method `estimateRegressionParameters` provides these values in a single array, where the first entry is the intercept, and the other entries are the β coefficients. The output of `estimateLinearModel` should be a `Map<String,Double>` that contains the β coefficient for every column in list `indep`, as well as an entry with key `"intercept"` that contains the value of the intercept.

Example: if you would execute the following code:

```
1 DataFrame<Double> df = sampledDataFrame();
2 Function<DataVector<Double>,Double> f;
3 f = row -> 17 * row.getValue("uniform1") + 3 * row.getValue("uniform2")
4         + row.getValue("normal1");
5 df = df.computeColumn("dep", f);
6 Map<String,Double> model;
7 model = df.statistics().estimateLinearModel("dep", "uniform1", "uniform2");
8 System.out.println(model);
```

the output should be

```
{intercept=1.9355845153530227, uniform1=16.961741766120387, uniform2=3.017948152625097}
```

Note: the call to `estimateLinearModel` is a default method implementation provided in the interface, that conveniently converts the arguments into a `List` and calls the `estimateLinearModel` method you need to implement.

Extension 3 – Plotting

In this extension you work with some of the functionality available in the XChart library for plotting and data visualization. While XChart has many chart types, you will work on creating a *scatter plot* and a *histogram* using this library.

First, you need to override the following method of the `DataFrame` interface, as the default implementation throws an `UnsupportedOperationException`:

```
1 public DataFramePlotting plotting();
```

When a user calls this method, they should get an object that implements the `DataFramePlotting` interface, which contains the following methods:

```
1 public XYChart scatter(String title, String xVar, String yVar);
2 public void saveScatter(File outputFile, String title, String xVar, String yVar)
      throws IOException;
3
4 public CategoryChart histogram(String title, String varName, int bins);
5 public void saveHistogram(File outputFile, String title, String varName, int bins)
      throws IOException;
```

The `scatter` method should construct a `XYChart` object with a data series that has the column with name `xVar` from the underlying `DataFrame` as the x data and the column with name `yVar` from the underlying `DataFrame`. Furthermore, the following style properties should hold:

- The title of the chart should be the provided `title`
- The title on the x axis should be the name `xVar`
- The title on the y axis should be the name `yVar`
- The theme of the chart should be `ChartTheme.GGPlot2`
- The series render style should be `XYSeriesRenderStyle.Scatter`
- The size of the markers should be 10
- There should be no legend visible

Note that it is not necessary to define the size or dimensions of the plot itself: it is sufficient to rely on the default settings provided by XChart.

The `histogram` method should construct a `CategoryChart` object with a data series computed using the `Histogram` class from `XChart` based on the data stored in the `varName` column of the underlying `DataFrame` object. The histogram should have a number of bins equal to `bins`. Furthermore, the following style properties should hold:

- The title of the chart should be the provided `title`
- The title on the x axis should be equal to the variable `xVar`
- The title on the y axis should be the string `"frequency"`
- The theme of the chart should be `ChartTheme.GGPlot2`
- There should be no legend visible

Note that it is not necessary to define the size or dimensions of the plot itself: it is sufficient to rely on the default settings provided by XChart.

The `saveScatter` method should call the `scatter` method to compute the relevant `XYChart` object, and write an image of the plot to the file provided as argument `outputFile`. For this purpose, you should use the class `BitmapEncoder` from the XChart library. You should use the format `BitmapFormat.PNG` as the output type, since `png` files are a lossless graphics format, in contrast to formats such as `jpg`.

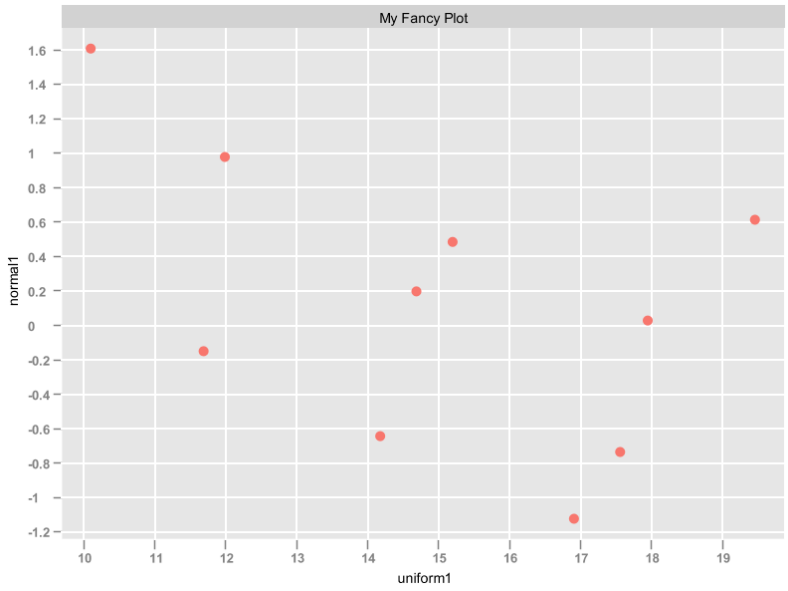
Similarly, the `saveHistogram` method should call the `histogram` method to compute the relevant `CategoryChart` object, and write an image of the plot to the file provided as argument `outputFile` using the `BitmapEncoder` class.

Example: suppose the following code is ran:

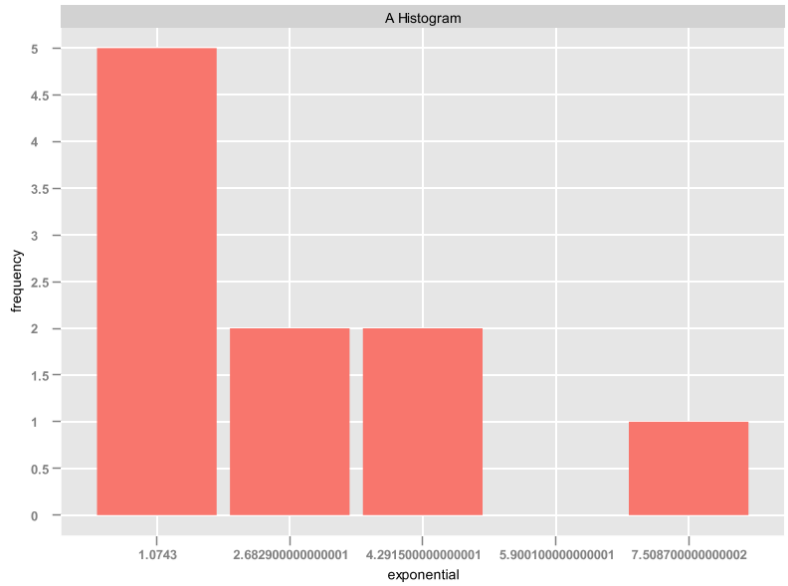
```
DataFrame<Double> df = sampledDataFrame();
df.plotting().showScatter("My Fancy Plot", "uniform1", "normal1");
df.plotting().showHistogram("A Histogram", "exponential", 5);
try {
    File scatter = new File("scatter.png");
    File histogram = new File("histogram.png");
    df.plotting().saveScatter(scatter, "My Fancy Plot", "uniform1", "normal1");
    df.plotting().saveHistogram(histogram, "A Histogram", "exponential", 5);
}
catch (IOException ex) {
    ex.printStackTrace();
}
```

Two windows should have been opened, one of them containing the plot that is shown in Figure 1a and the other the plot that is shown in Figure 1b. In addition to that, two files with names `scatter.png` and `histogram.png` should have been written, that contain images equal to the plots shown in the windows.

Note that the `showScatter` and `showHistogram` methods are default methods provided in the `DataFramePlotting` interface. They call the methods you are supposed to implement, but you do not need to implement these methods.



(a) Scatter plot example



(b) Histogram example

Figure 1: Example plots generated with the XChart library

	A	B	C	D	E
1	year	revenue	costs		
2	2015	70021.35	25071.12		
3	2016	67008.12	108632.8		
4	2017	10632.83	37816.8		
5	2018	85216.33	31863.73		
6					
7					

Figure 2: Example of a data frame that was written to the Microsoft Excel file format

Extension 4 – Excel IO

In this extension you work with some of the functionality available in the Apache POI library that can read and write files in the Microsoft Office format. Since Microsoft Excel is a very popular application used by many financial institutions and other companies that perform data analysis, it is useful to be able to read and write files in the format of this application. While it is possible to work with `csv` files, this is very error prone, as depending on the regional settings (i.e. United States or The Netherlands) a `csv` may or not be read in correctly. If you write an actual `xlsx` file, the format used by Microsoft Excel, there is no risk that numbers are not correctly interpreted.

For this assignment you should implement the `FileUtils` class, which should have two static methods in its public interface:

```
1 public static void writeToXLSX(DataFrame<Double> data, File output) throws
      IOException {...}
2 public static DataFrame<Double> readFromXLSX(File in) throws IOException {...}
```

The `writeToXLSX` method should take a `DataFrame<Double>` and write the contents of that `DataFrame` to the given `output` file using Apache POI. The workbook written should contain a single sheet, with the data from the `DataFrame` on that sheet. The first line of the sheet should contain the column names in the order in which the column names are stored in the data frame. Then, numeric values of the data frame should be written to the consecutive rows of the sheet.

The `readFromXLSX` should read the contents of the provided `xlsx` file which is provided as variable `in` and convert its contents to a `DataFrame<Double>` object. You may assume that the workbook in the file contains only a single sheet, and that the contents of that sheet are written as is explained for the `writeToXLSX` method: the first row contains the column names (which should retrieved from the `Cell` objects using the `getStringCellValue()` method of the `Cell` class, while all rows beyond the first one contain numeric values that can be retrieved from the `Cell` objects using the `getNumericCellValue()` method of the `Cell` class.

Example: when you would execute the following code

```
1  DataFrame<Double> df = testDataFrame();
2  try {
3     File f = new File("mydata.xlsx");
4     FileTools.writeToXLSX(df, new File("mydata.xlsx"));
5     DataFrame<Double> read = FileTools.readFromXLSX(f);
6     read.print();
7  }
8  catch (IOException ex) {
9     ex.printStackTrace();
10 }
```

it should have written the `DataFrame` to a file called `mydata.xlsx` that looks as is shown in Figure 2. Additionally, it should print the following to indicate that the data was read back in successfully:

```
          year        revenue      costs
row_0     2015.0      70021.35     25071.12
row_1     2016.0      67008.12     108632.8
row_2     2017.0      10632.83     37816.8
row_3     2018.0      85216.33     31863.73
```