Handout

for

One day Workshop with Hands-on Training

on

"Machine Learning and Its Applications"

Prepared and Presented
By

Dr. D. Asir Antony Gnana Singh, B.E., M.E., M.B.A., Ph.D.,
Department of CSE, Anna University,
BIT Campus, Tiruchirappalli–620 0024

at
UCE, Anna University, BIT-Campus,
Tiruchirappalli-620 0024
on
28th September 2019

Machine Learning Codes Using Java

Contents

Chapter 1

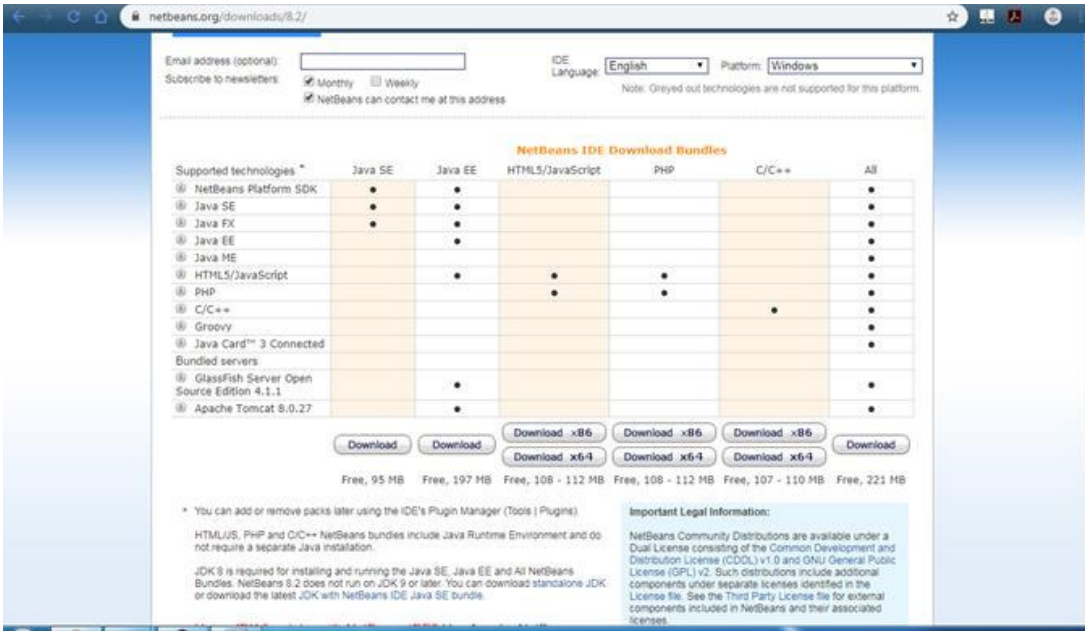
Software Installation and Configuration

Part I
Downloading and Installing NetBeans IDE 8.2

Step I: Download the NetBeans IDE 8.2

1. Go to the official website of NetBeans IDE 8.2 using the following link to download:

   https://netbeans.org/downloads/8.2/

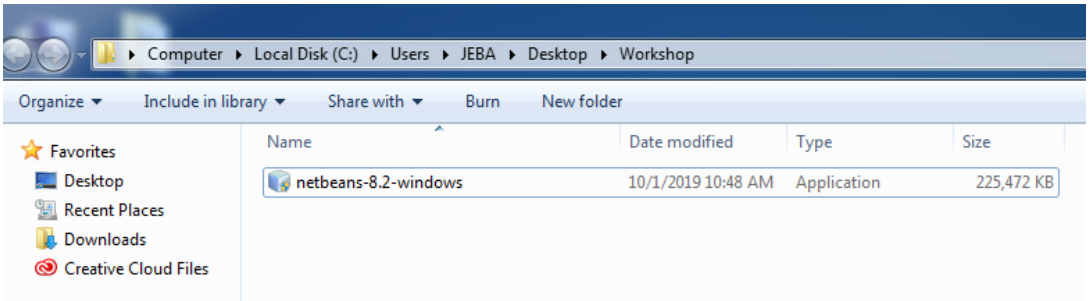


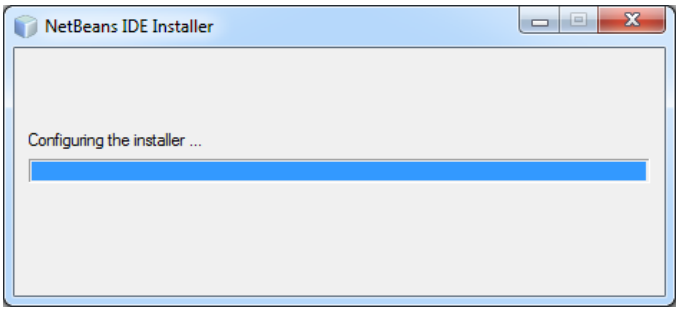
Snapshot of web page (https://netbeans.org/downloads/8.2/)

2. Choose the option "English" for IDE Language: and choose the option "Windows" option for Platform:
3. Click the download button provided for the option "All" for the NetBeans IDE Download Bundles to download the IDE.
4. (netbeans-8.2-windows)

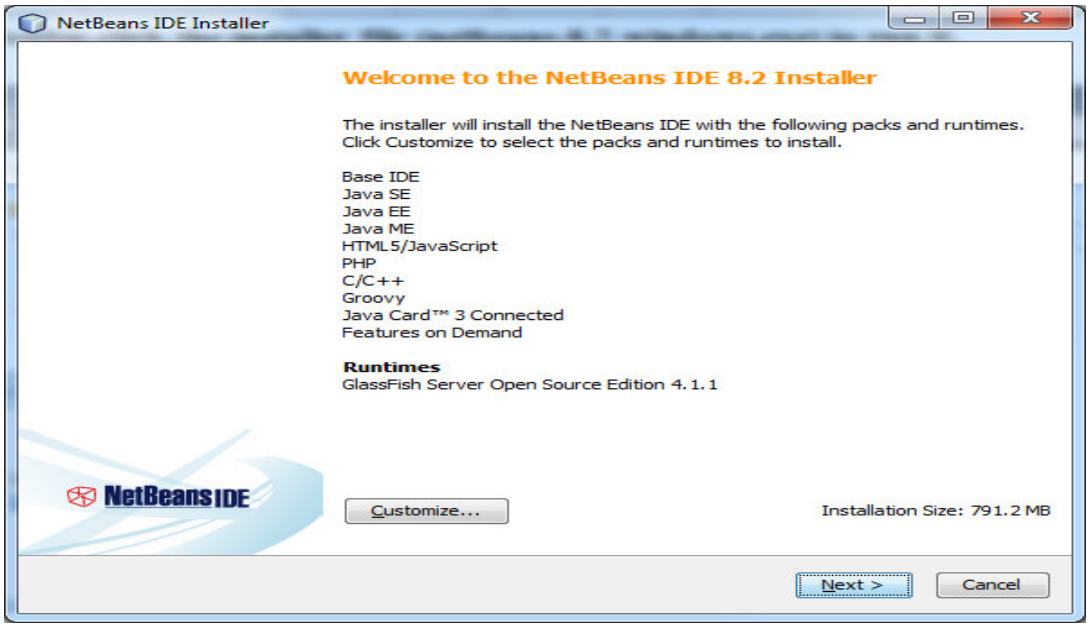
Step II: Install the IDE

Double-click the installer file (netbeans-8.2-windows.exe) to run it.

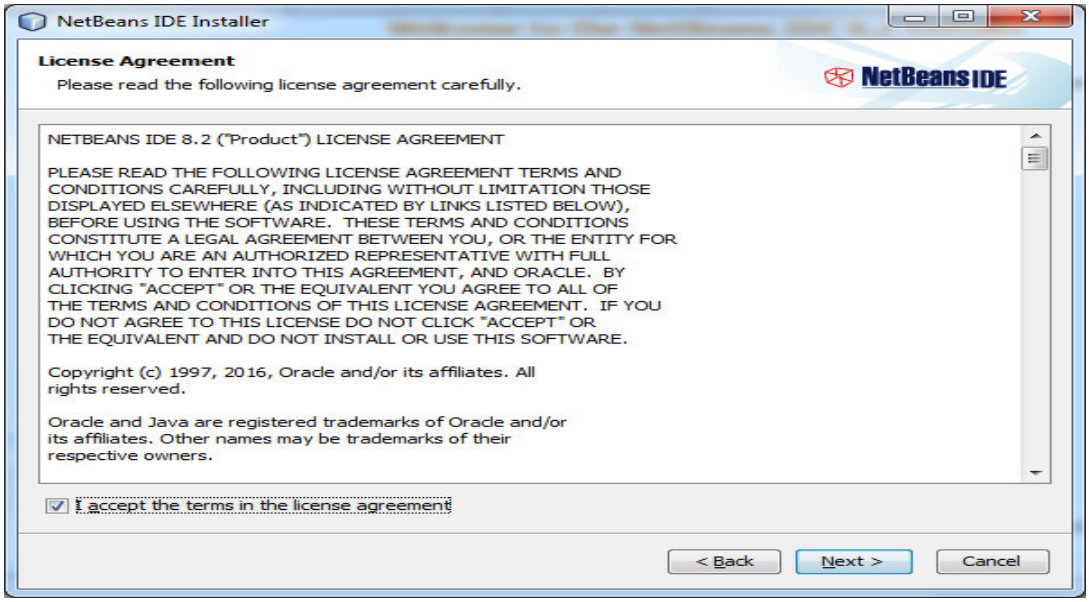




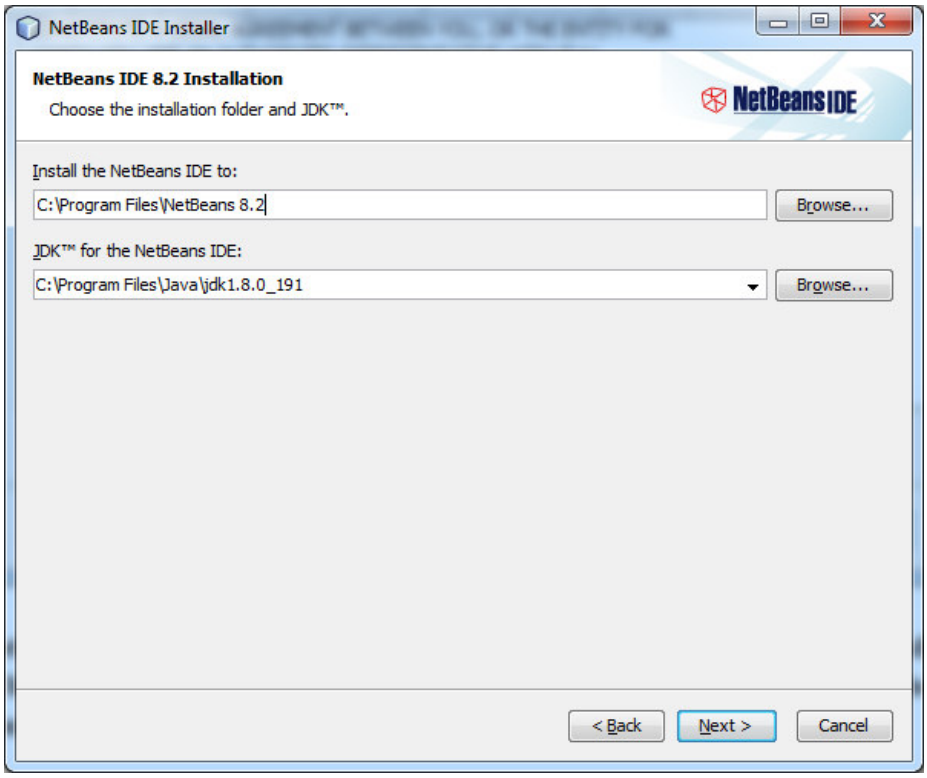
1. Click the Next button to continue

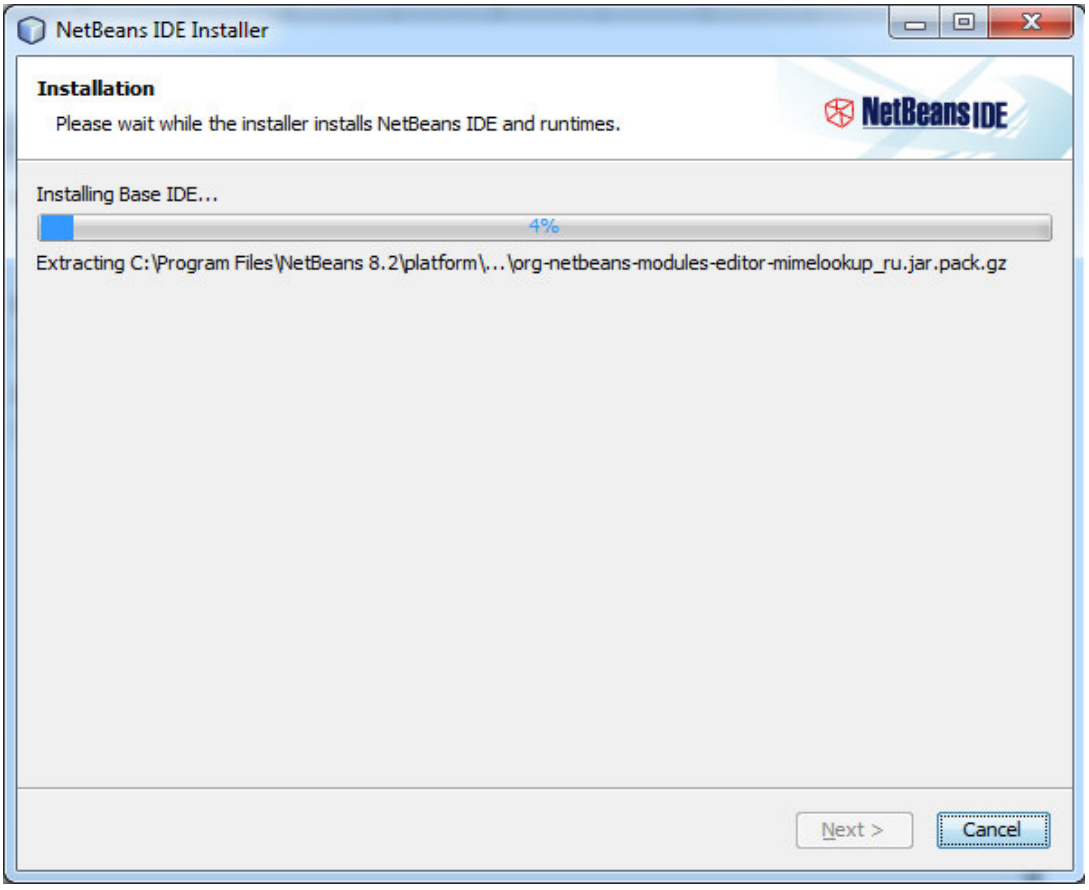


2. Click the button I Agree (That means you accept the terms of agreement) to continue

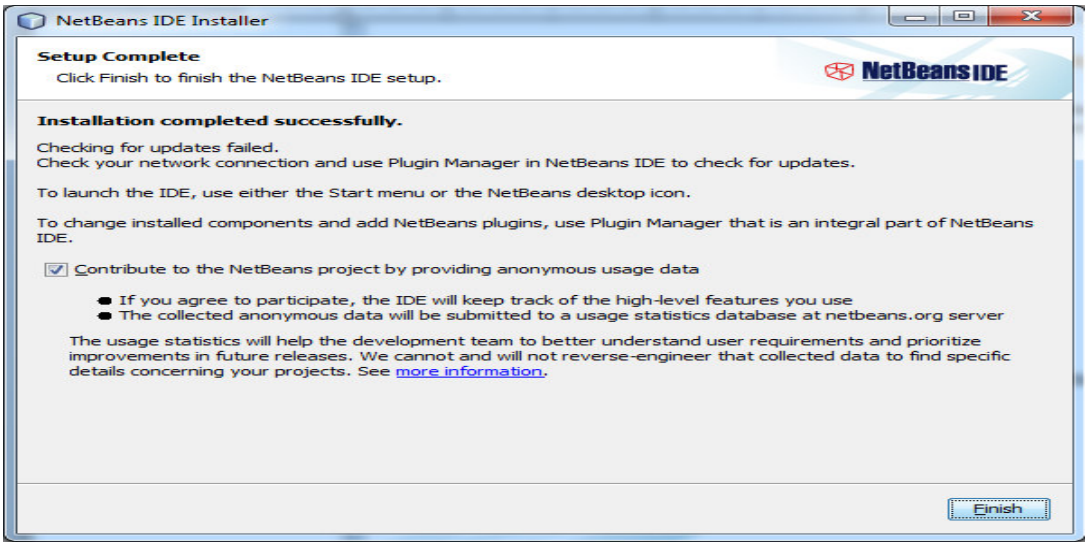


3. Choose the installation folder to install the IDE 8.2 and click next button to continue



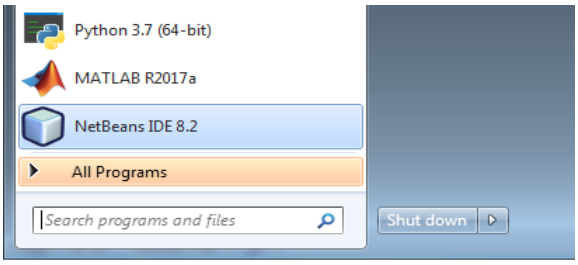


4. After Installation complete the finish button to complet the installation

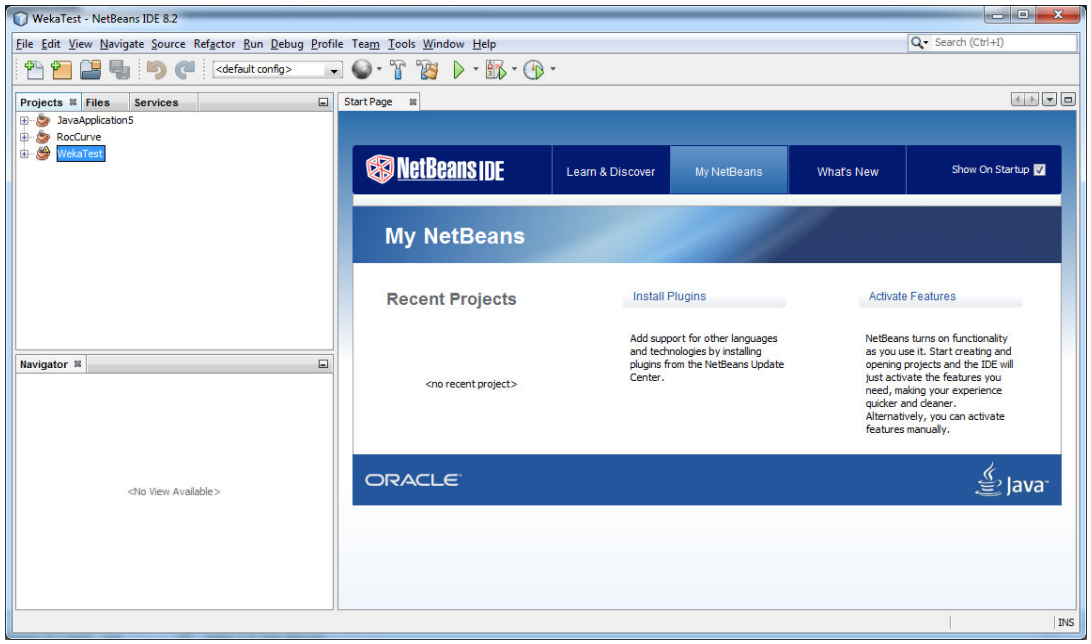


Step III: Explore the IDE

1. Double-click the icon of the IDE to open the IDE.



2. The IDE will appear as shown in the following screenshot :

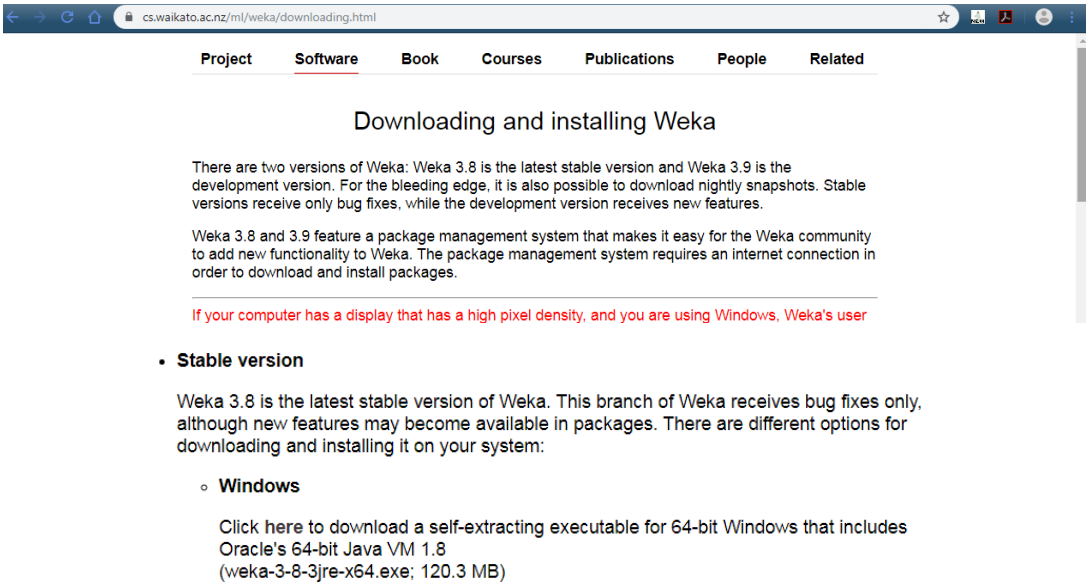


Part II
Downloading and Installing Weka 3.8.3

Step I: Download the Weka 3.8.3

1. Go to the official website of Weka using the following link for downloading:

   https://www.cs.waikato.ac.nz/ml/weka/downloading.html

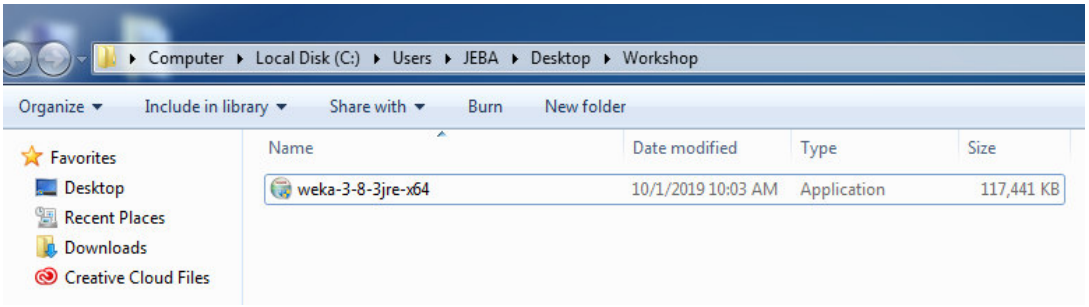


Snapshot of Web page (https://www.cs.waikato.ac.nz/ml/weka/downloading.html)

2. Click the down load link of (Click here to Down load the download a self-extracting executable for 64-bit Windows that includes Oracle's 64-bit Java VM 1.8) from the Window section of Stable version to download the weka-3-8-3jre-x64.exe. (120.3 MB)

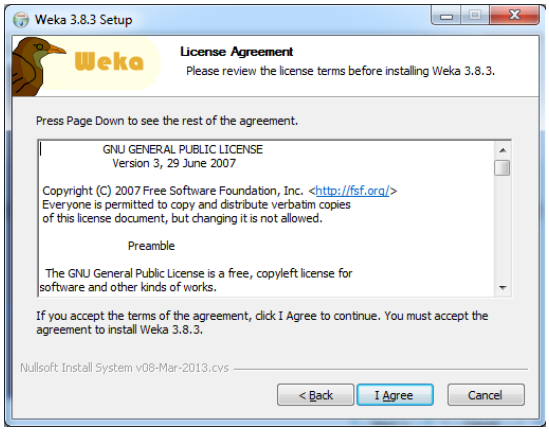
Step II: Install the Weka 3.8.3

1. Double-click the installer file (weka-3-8-3jre-x64.exe.) to run it.

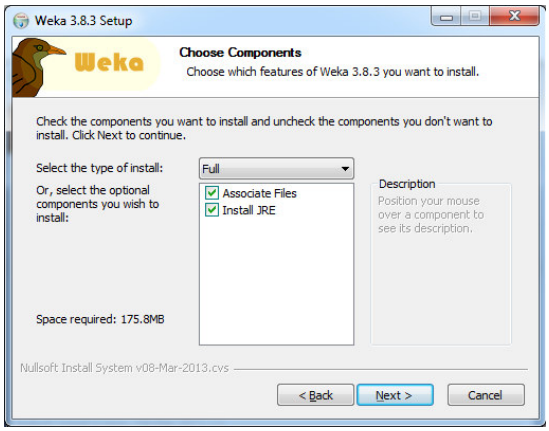


2. Click the Next button to continue

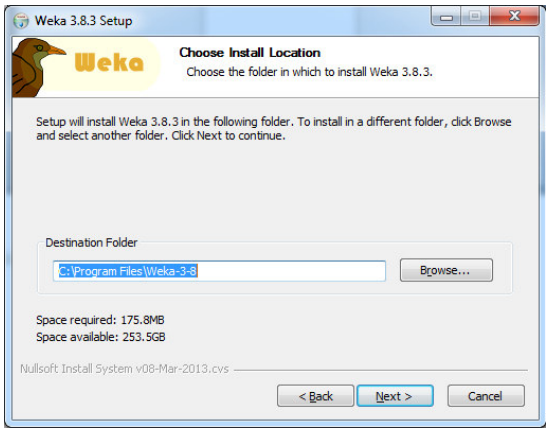
3. Click the button I Agree (That means you accept the terms of agreement) to continue



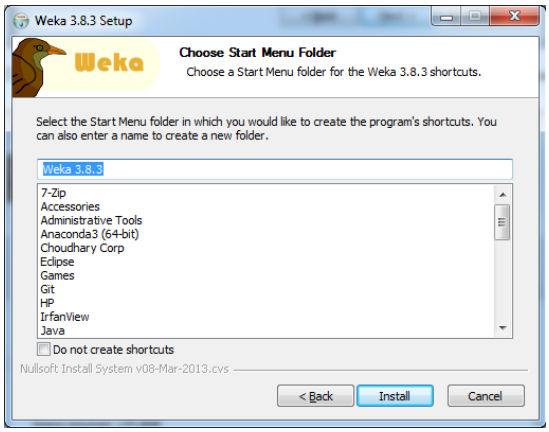
4. Select the option “Full” from the dropdown menu “Select the type of Installation:” Click on the checkboxes “Associated Files” and “Install JRE” and Click the next button to continue

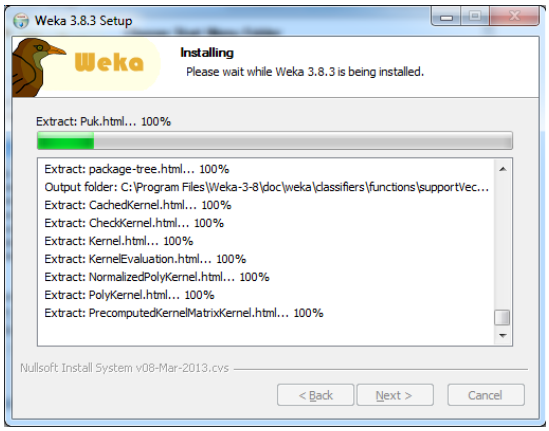


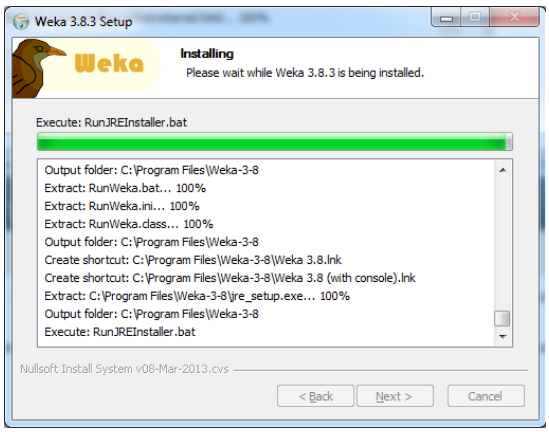
5. Choose the directory to install the Weka 3.8.3 and click next button to continue



6. Click the Install button to install



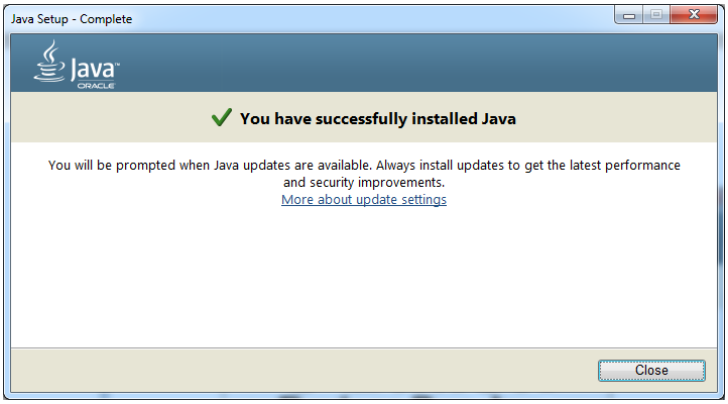




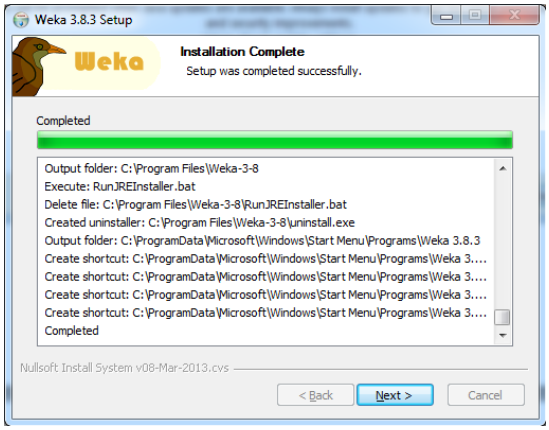
7. At the end of the installation a "Java Setup-Welcome" popup window will appear to setup the compatible version of Java for Weka 3.8.3. Click the Install button to setup the Java for Weka 3.8.3

8. Installation of Java is started as following snapshot

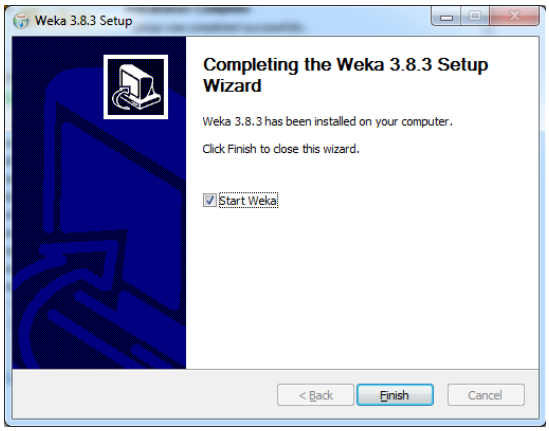
9. After completion of the installation of the Java the popup window will appear as follow snapshot and click the close button to close the popup window



10. Click the next button of the Weka 3.8.3 Setup window to complete the installation



11. Clock on the Start Weka checkbox and Click the finish button to launch the Weka GUI Chooser

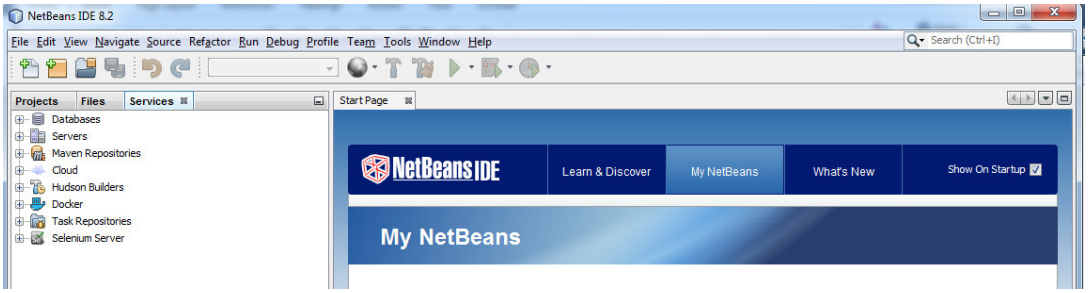


12. The Weka GUI Chooser will appear as the screenshot given below:

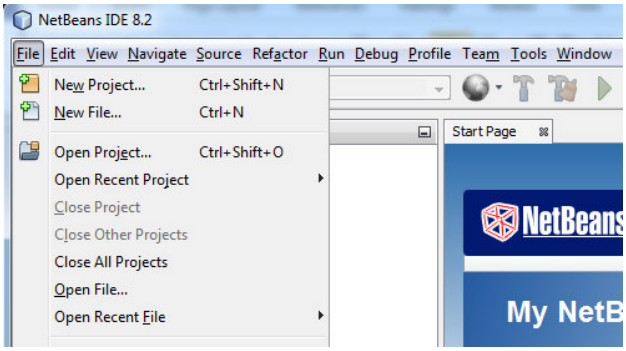
Part III
Creating a new Java project space in NetBeans IDE 8.2 to develop Machine learning model

Step I:

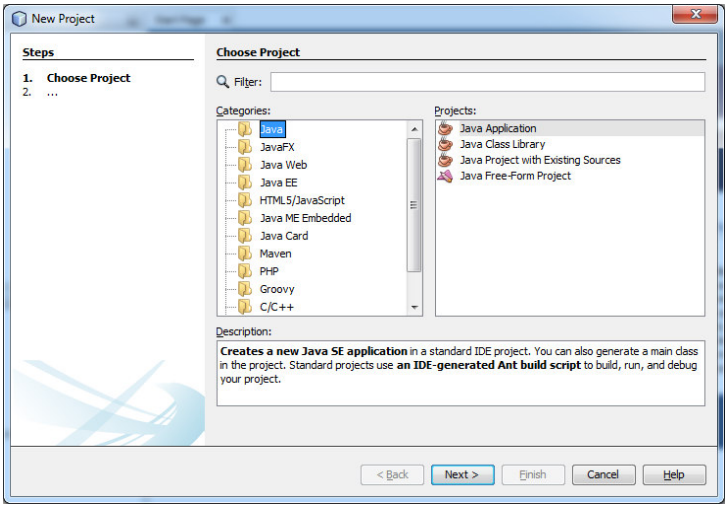
1. Explore the NteBeans IDE 8.2



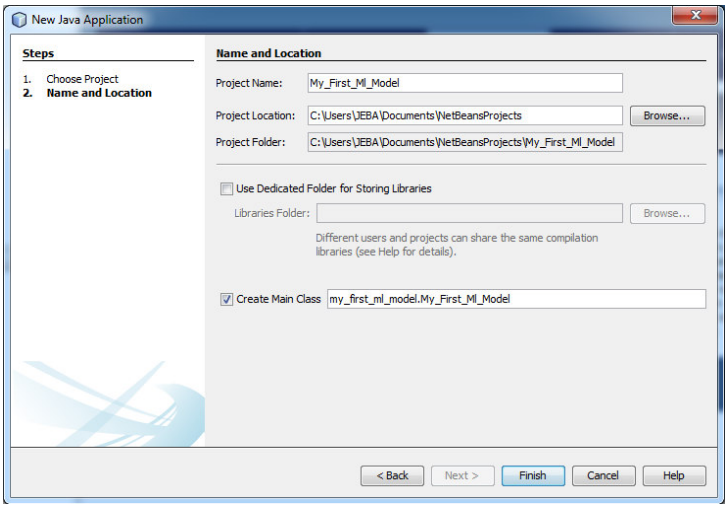
2. Select the File menu (New Project popup window will appear as shown in below screenshot)



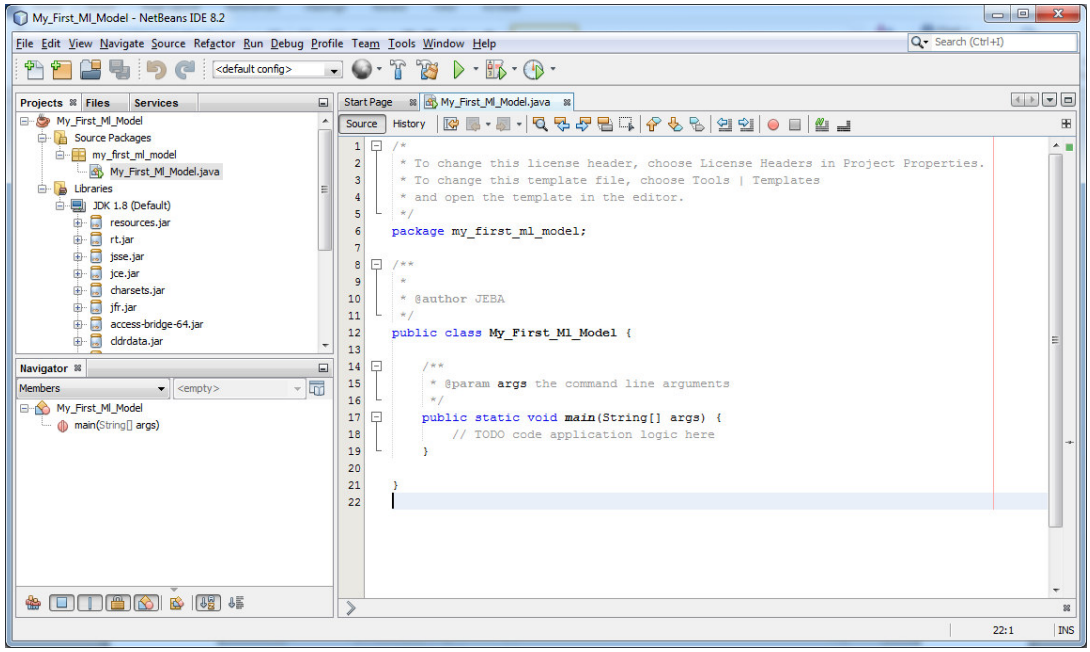
3. Choose the "Java menu" from the dropdown menu "Categories" and choose "Java Application" from the dropdown menu "Project" and click on the "New Project" Tab (New Java Application popup window will appear)



4. Provide the Project Name as "My_First_Ml_Model" and click finish

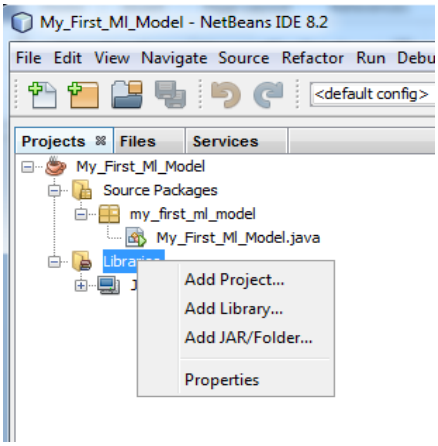


5. Choose the "Projects" under this tab you can see:
   a. The Java project is created with name of "My_First_Ml_Model" and inside of the project the two directories are created one for Source package and another for Libraries.
   b. Inside of the source package the main package is named as "My_First_Ml_Model".
   c. Inside of the main package a main java class file is created with the file name of "My_First_Ml_Model.java
   d. Inside of the libraries the JAR (Java ARchive) files for the JDK (Java Development Kit) are available.

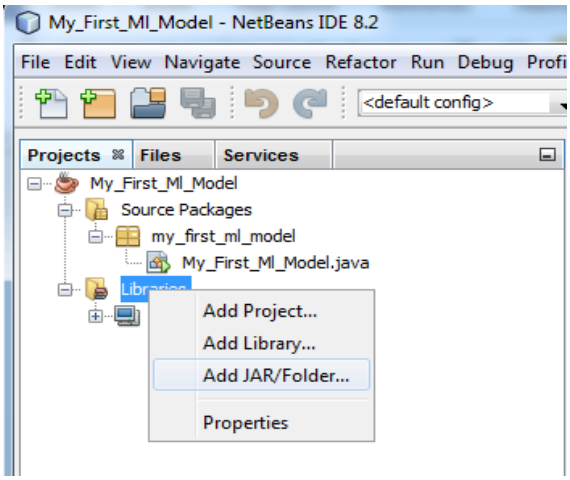


Step II: (Configure the Weka API 3.8.3 and NetBeans IDE 8.2) Add the Java jar file into the Project Libraries using the following steps:

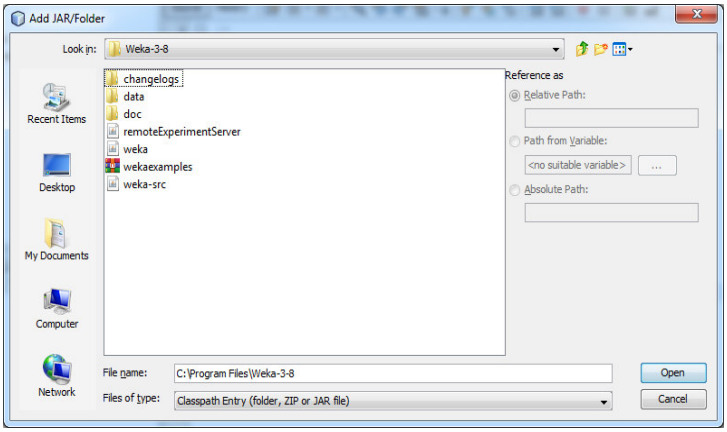
1. Do right-click on the Libraries folder as shown below screenshot



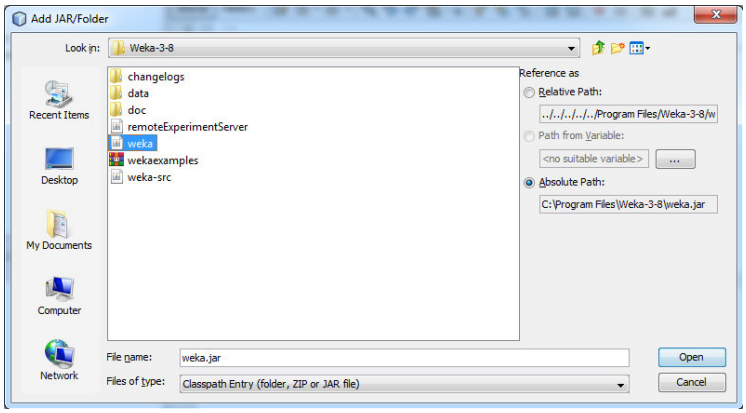
2. Click on the menu "Add JAR/Folder" as shown in the below screenshot



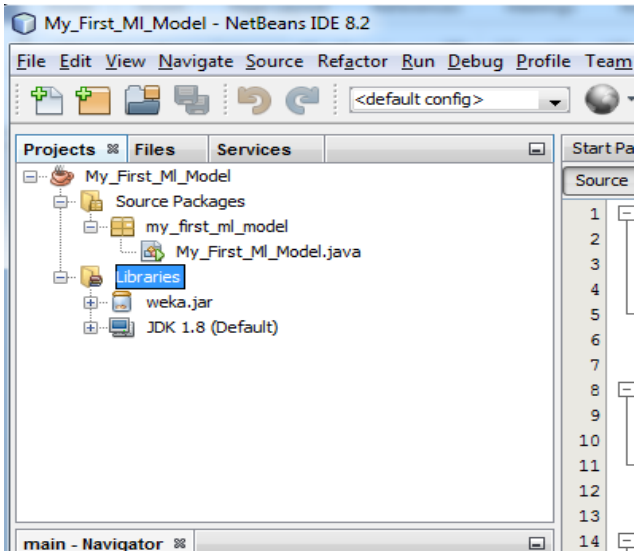
3. Add JAR/Folder popup window will appear and Select the Weka installed folder (The path of where the Weka is installed) (For example: "C:\Program Files\Weka-3-8") as shown in below screenshot.



4. Click on the Weka jar file from the directory (C:\Program Files\Weka-3-8) as shown in the below screenshot and click on the open button



5. Now the Weka.jar is included in the Libraries as shown in the blow screenshot.



Chapter 2
Dataset Preparation

Part I
Preparation of Dataset

Step I: Prepare the dataset (in ARFF (attribute relation file format))

Dataset consist of M instances with N attributes followed by the class label as shown below:

Attributes					Class/Label
X_1	X_2	X_3	...	X_N	C
x_{11}	x_{21}	x_{31}	...	x_{N1}	c_1
x_{12}	x_{22}	x_{32}	...	x_{N2}	c_2
x_{13}	x_{23}	x_{33}	...	x_{N3}	c_1
.	.	.		.	.
.	.	.	...	.	.
.	.	.		.	.
x_{1M}	x_{2M}	x_{3M}	...	x_{NM}	c_2

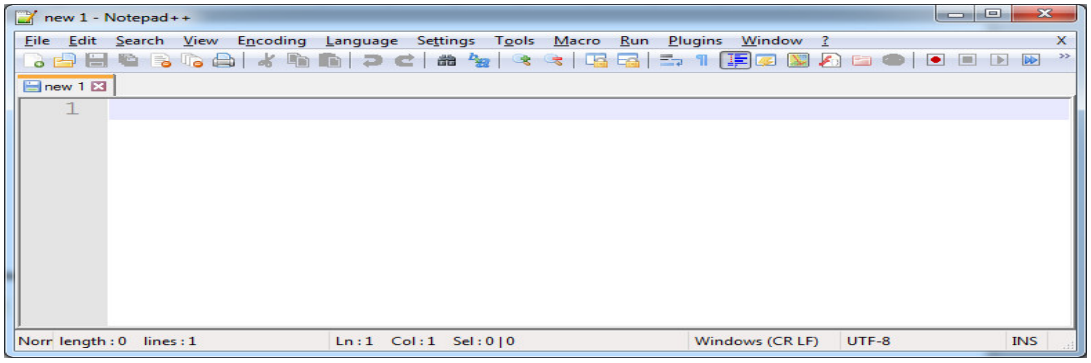
Representation of Dataset

Note: The step by step procedure is given to prepare medical dataset entitled diabetes dataset. Dataset contains eight attribute and one class for 768 instances. This dataset is used to predict whether a person has diabetes or not.

Description of Attribute and class label

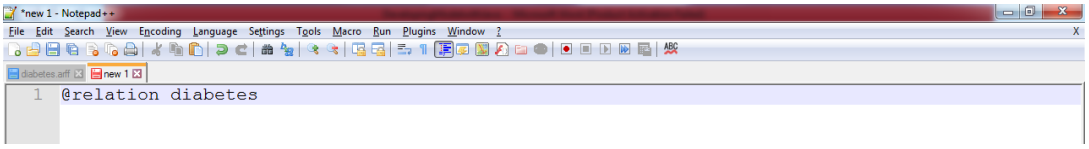
1. Number of times pregnant
1. Plasma glucose concentration a 2 hours in an oral glucose tolerance test
2. Diastolic blood pressure (mm Hg)
3. Triceps skin fold thickness (mm)
4. 2-Hour serum insulin (mu U/ml)
5. Body mass index (weight in kg/(height in m)^2)
6. Diabetes pedigree function
7. Age (years)
8. Class variable (0 or 1)

1. Open Notepad/Notepad++ and



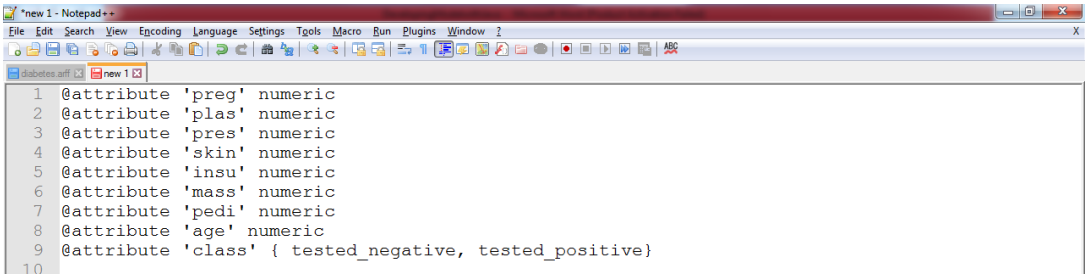
2. Define the name of the dataset:

   Syntax: @ relation <dataset_name>
   Code: @relation diabetes



3. Define the attributes and class label of the dataset:

   Syntax: @ attribute <name of the attribute > <attribute type>
   Syntax: @ attribute <class label name > <attribute type>
   Code: @attribute 'preg' numeric
   @attribute 'plas' numeric
   @attribute 'pres' numeric
   @attribute 'skin' numeric
   @attribute 'insu' numeric
   @attribute 'mass' numeric
   @attribute 'pedi' numeric
   @attribute 'age' numeric
   @attribute 'class' { tested_negative, tested_positive}



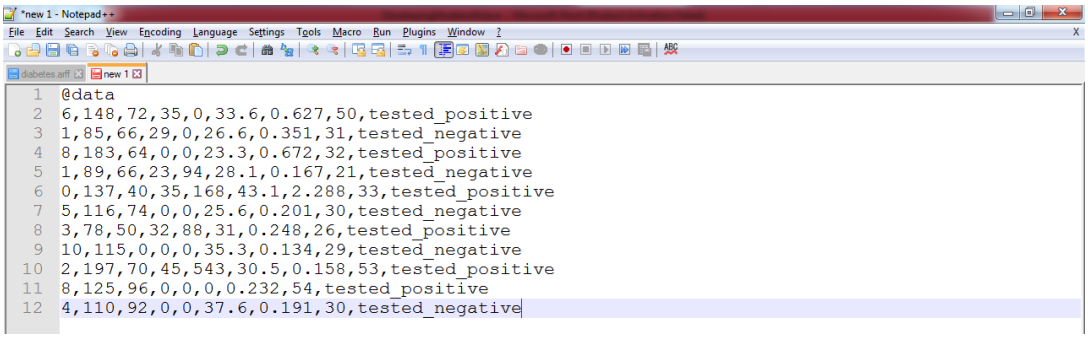
4. Define the instances of the dataset:

Syntax: @data

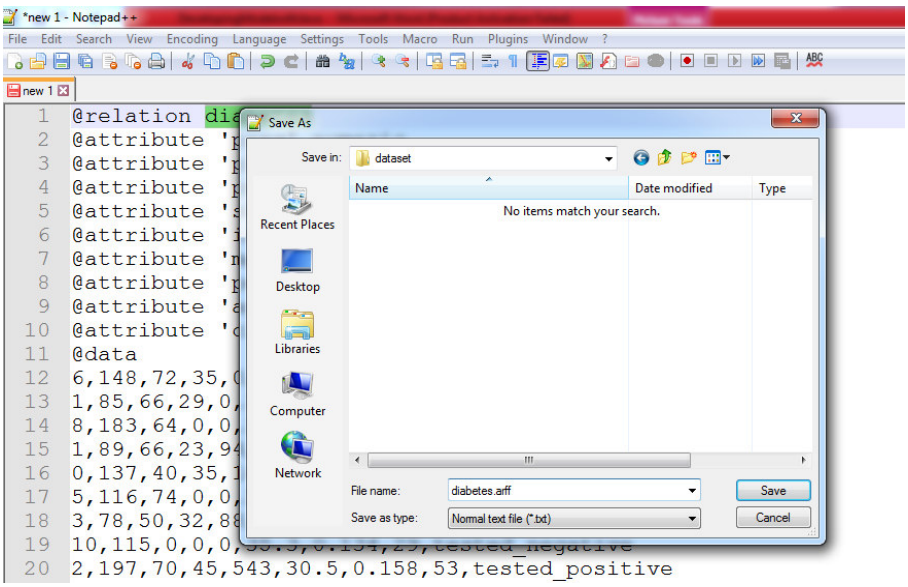
< value for attribute 1, value for attribute 2, ….. value for attribute n>

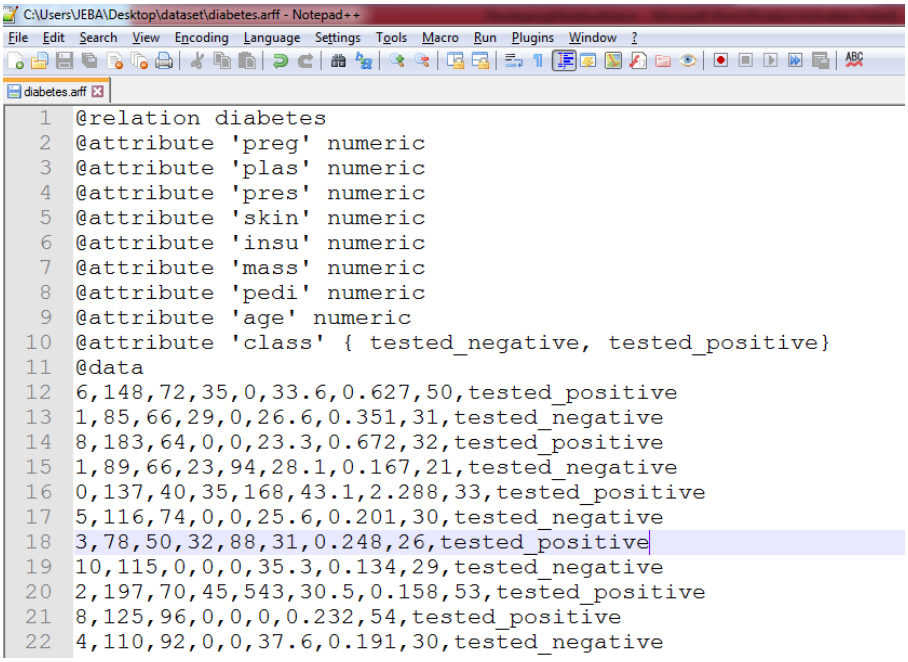
Code:

```
@data
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
0,137,40,35,168,43.1,2.288,33,tested_positive
5,116,74,0,0,25.6,0.201,30,tested_negative
3,78,50,32,88,31,0.248,26,tested_positive
10,115,0,0,0,35.3,0.134,29,tested_negative
2,197,70,45,543,30.5,0.158,53,tested_positive
8,125,96,0,0,0,0.232,54,tested_positive
4,110,92,0,0,37.6,0.191,30,tested_negative
10,168,74,0,0,38,0.537,34,tested_positive
```



5. Save the file with the file extension of arff shown in screenshots below:




C:\Users\JEBA\Desktop\dataset\diabetes.arff - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins Window ?
diabetes.arff
@relation diabetes
@attribute 'preg' numeric
@attribute 'plas' numeric
@attribute 'pres' numeric
@attribute 'skin' numeric
@attribute 'insu' numeric
@attribute 'mass' numeric
@attribute 'pedi' numeric
@attribute 'age' numeric
@attribute 'class' { tested_negative, tested_positive}
@data
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
0,137,40,35,168,43.1,2.288,33,tested_positive
5,116,74,0,0,25.6,0.201,30,tested_negative
3,78,50,32,88,31,0.248,26,tested_positive
10,115,0,0,0,35.3,0.134,29,tested_negative
2,197,70,45,543,30.5,0.158,53,tested_positive
8,125,96,0,0,0,0.232,54,tested_positive
4,110,92,0,0,37.6,0.191,30,tested_negative

Chapter 3
Developing Ml models and performance evaluation

Part I
Developing Java Code to construct and evaluate the Machine learning models

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. The dataset is loaded to develop the machine learning mode. Machine learning model is evaluated based any one of the criteria such a percentage split, n fold cross validation method, and training set as the test set. The evaluated results are displayed in terms of performance evaluation metrics such as accuracy, TP Rate, FP Rate, Precision, Recall, F-Measure and ROC Area.

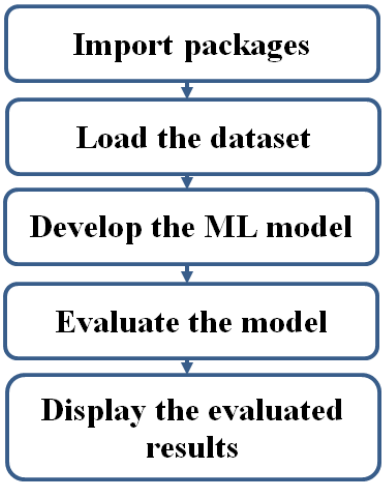
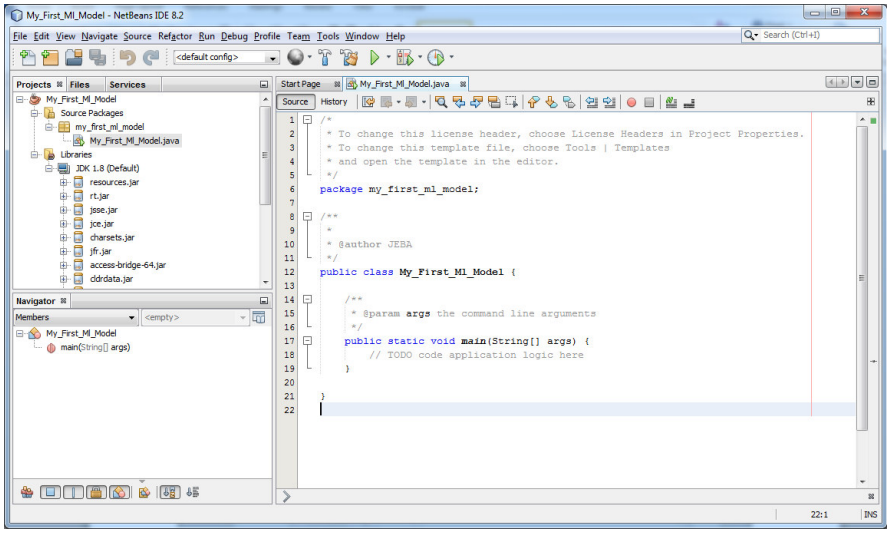


Figure: Steps to develop a Java code for construct and evaluate the Ml models

Note: Open the Code editing space of the main class file "My_First_Ml_Model.java" (as shown in below screenshot) to write the code for developing machine learning models



Step I: Develop the Java code to construct and evaluate the Machine learning models

1. Import the required packages:

a. Import Package to develop the Ml model (J48)

Definition:	`J48` class is used to generate decision tree-based machine learning model.
Code:	`import weka.classifiers.trees.J48;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Import package to access the dataset from a file

Definition:	`DataSource` class is used loading data from files
Code:	`Import weka.core.converters.ConverterUtils.DataSource;`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

c. Import package to evaluating the performance of the Ml model

Definition:	`Evaluation` class for evaluating machine learning models
Code:	`import weka.classifiers.Evaluation;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

d. Import package to generate the random number

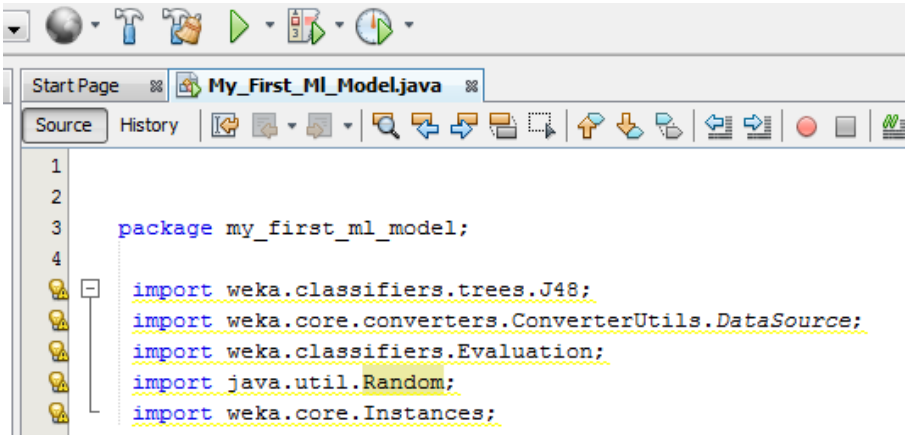
Definition:	`Random` class is used to generate a stream of pseudorandom numbers
Code:	`import java.util.Random;`
Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

e. Import package to handle the instance of the dataset

Definition:	`Instances` class is used to handle the ordered set of weighted instances.
Code:	`import weka.core.Instances;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to import required classes:

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
```



Step II: Define the main class and the main method

Note: Below a Java come is provided for displaying the string "HelloWorld"

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the
string.
    }
}
```

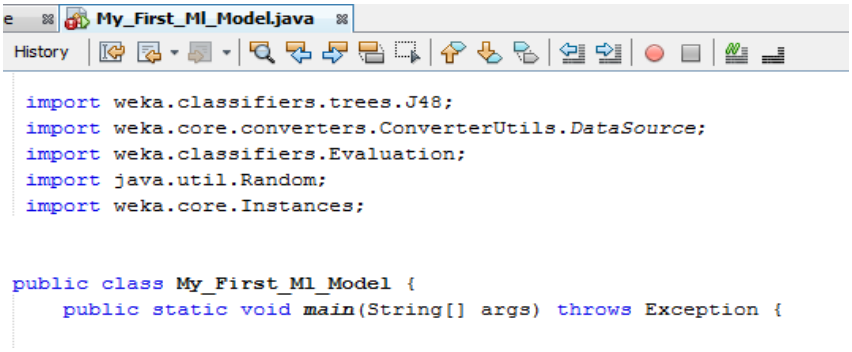
- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Definition:	`public` class: Any other class can access a public field or method. `public` method: This method is public and therefore available to anyone. `static` method: This method can be run without having to create an instance of the class MyClass. `void` method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.
Code:	`public class My_First_Ml_Model {` `public static void main(String[] args) throws Exception {`
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {
```



Step III: Load the dataset from the file

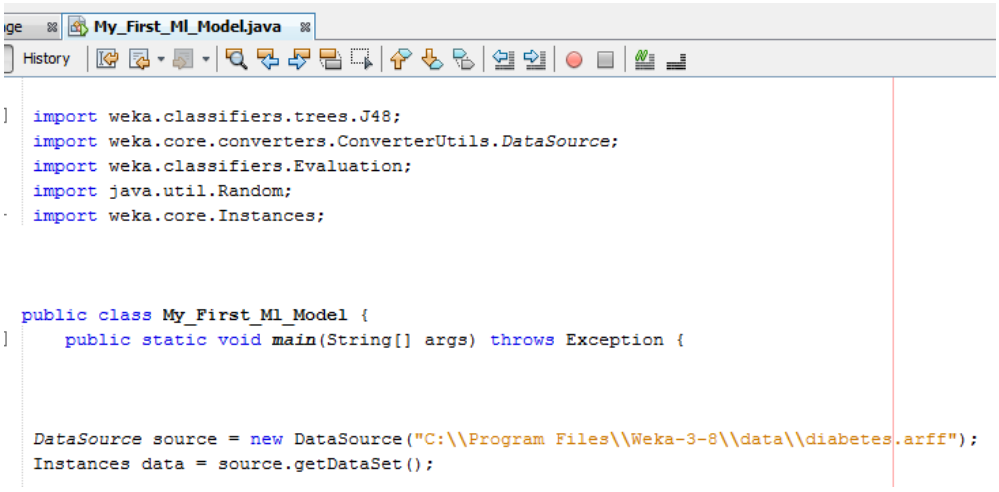
Definition:	`DataSource` class is used to loading data from files `getDataSet()` method returns the full dataset, can be null in case of an error. In this code, an object `source` is created for the class `DataSource` and the object `source` is used to invoke the method `getDataSet()` using dot operator Note: Provide the entire path of the dataset which should be loaded

Code:	`DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");`
	
	`Instances data = source.getDataSet();`

Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();
```

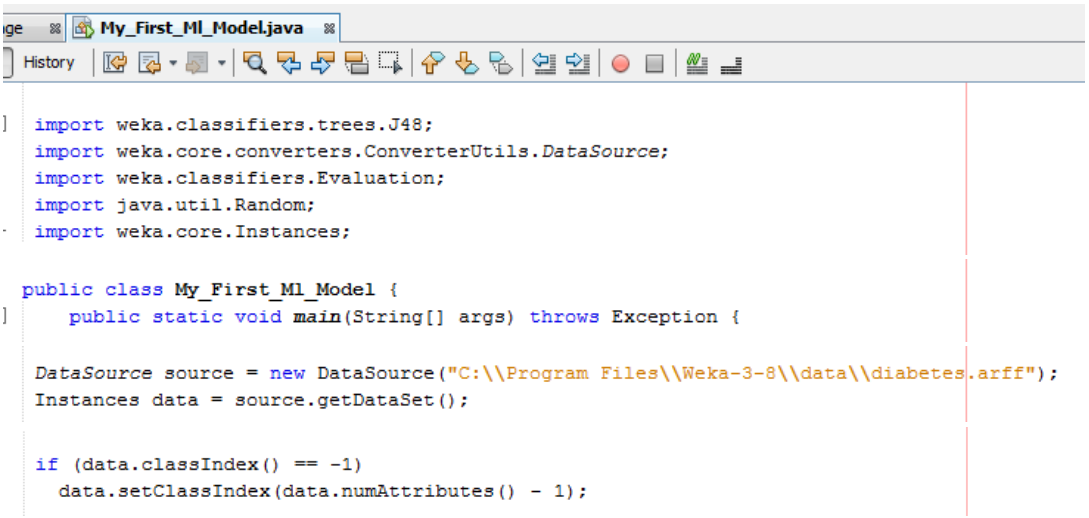


Step IV: Set the class attribute for the dataset

Definition:	`classIndex()` Returns the class attribute's index (default it is set -1). `setClassIndex(int classIndex)` Sets the class index of the dataset. `numAttributes()` Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object `data` which is created for the class `Instances`.
Code:	`if (data.classIndex() == -1)` `  data.setClassIndex(data.numAttributes() - 1);`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset

```
if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
```



Step V: Develop the machine learning model from the dataset

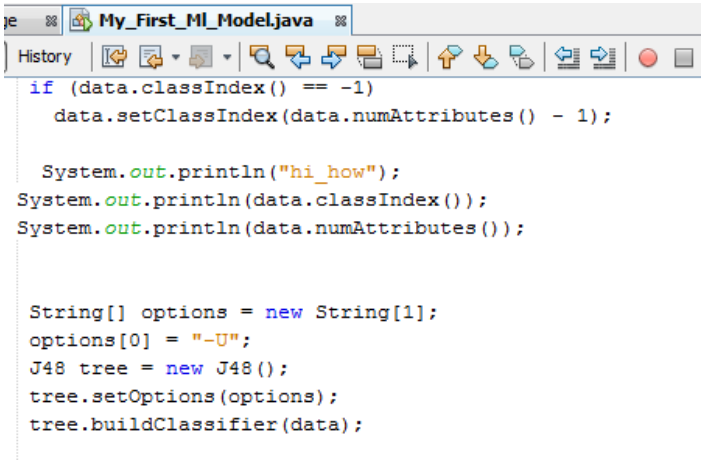
Definition:	`String` class represents character strings `setOptions` method sets the options (Option –U for unpruned tree) for `J48` class `J48` class for developing a pruned or unpruned C4.5 decision tree `buildClassifier` method generates the classifier (Machine learning model) Note: `tree` is the object of the class `J48` and the data is the object of the class `DataSource`

Code:	`String[] options = new String[1];`
	`options[0] = "-U";`
	`J48 tree = new J48();`
	`tree.setOptions(options);`
	`tree.buildClassifier(data);`

Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/lang/String.html http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

Snapshot of code to develop the Machine Learning Model from the dataset

```
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);
```

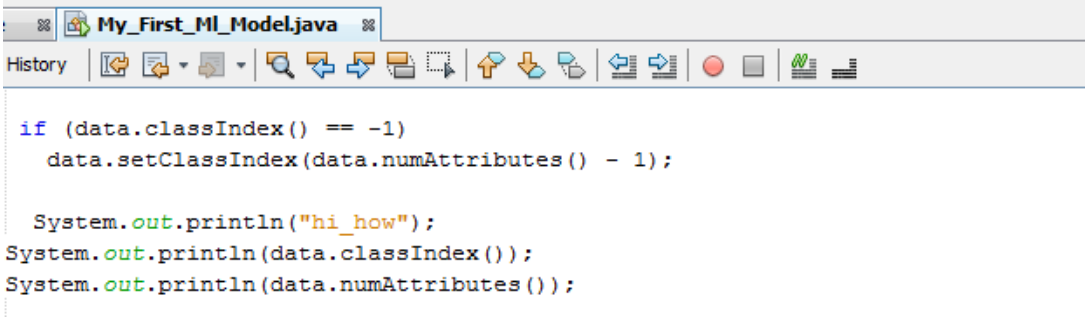


Step VI: Evaluate the performance of Machine learning model and display the result in console window

Definition:	`Evaluation` class is used for evaluating machine learning models. `crossValidateModel` method Performs a (stratified if class is nominal) cross-validation for a classifier on a set of instances. `System.out.println(data)` is standard output stream that displays value of the data for output `toSummaryString` method outputs the performance statistics in summary form.
Code:	`Evaluation eval = new Evaluation(data); eval.crossValidateModel(tree, data, 10, new Random(1)); System.out.println(eval.toSummaryString("\nResults\n======\n", false));`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
Evaluation eval = new Evaluation(data);
eval.crossValidateModel(tree, data, 10, new Random(1));
   System.out.println(eval.toSummaryString("\nResults\n======\n", false));
```



```
String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);


Evaluation eval = new Evaluation(data);
eval.crossValidateModel(tree, data, 10, new Random(1));
   System.out.println(eval.toSummaryString("\nResults\n======\n", false));
   }
}
```

Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

Snapshot of complete code to develop and evaluate the Machine Learning Model from the dataset

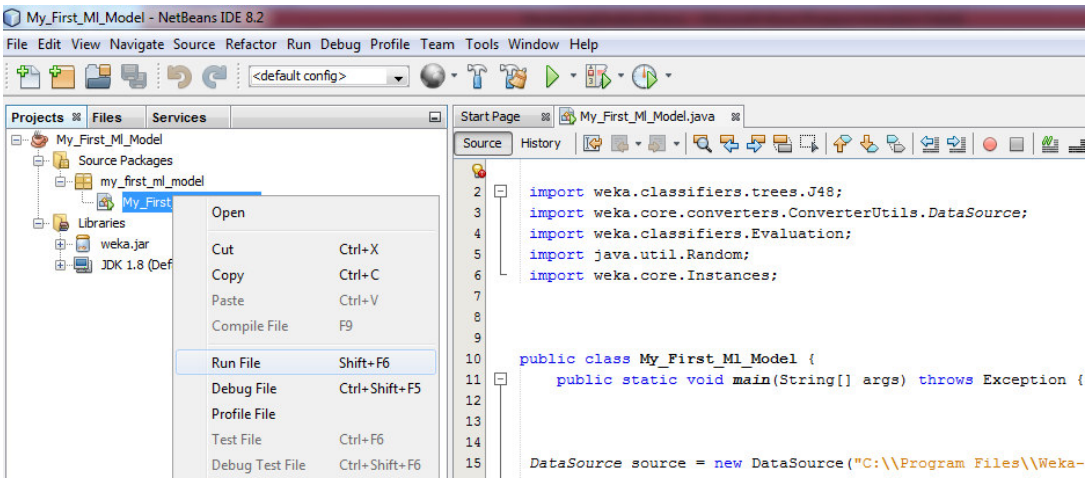
```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;


public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {


DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);


String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);

   Evaluation eval = new Evaluation(data);
   eval.crossValidateModel(tree, data, 10, new Random(1));
      System.out.println(eval.toSummaryString("\nResults\n======\n", false));

   }
}
```

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
public class My_First_Ml_Model {
public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-
8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
```

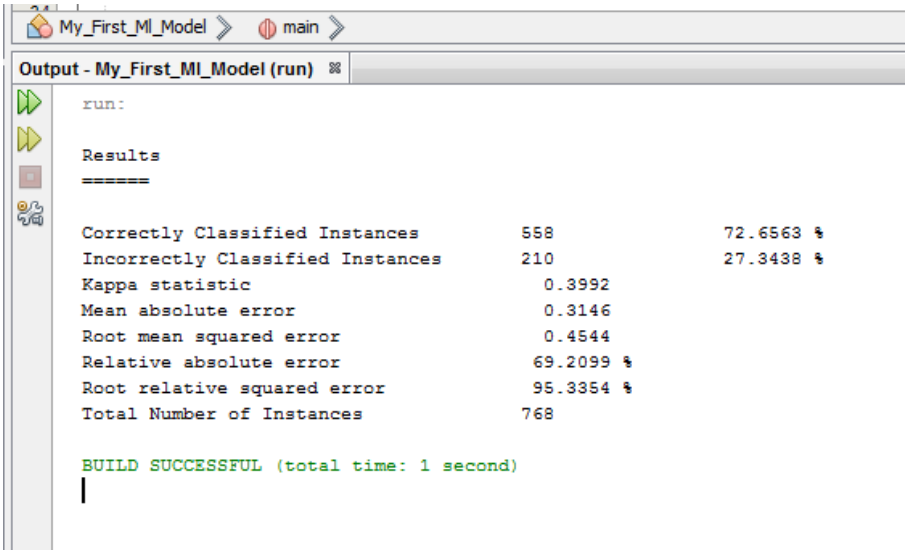
```
 String[] options = new String[1];
 options[0] = "-U";
 J48 tree = new J48();
 tree.setOptions(options);
 tree.buildClassifier(data);

 Evaluation eval = new Evaluation(data);
 eval.crossValidateModel(tree, data, 10, new Random(1));
    System.out.println(eval.toSummaryString("\nResults\n======\n",
false));

    }
}
```

Step VII: Run the code and display the results

Do right-click on the Java class file named as "My_First_Ml_Model" one drop down menu is appeared and click on the "Run File" option to run the code and display the results as shown in the screenshot below. Or Simply Press the keys (Shift+F6) to run the code display the results.



Note: The results are appeared in the output window as shown in screenshot below



Part II
Additional Workouts

1. Displaying the dataset

a. Display the entire dataset

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

System.out.println(data);
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
@relation pima_diabetes

@attribute preg numeric
@attribute plas numeric
@attribute pres numeric
@attribute skin numeric
@attribute insu numeric
@attribute mass numeric
@attribute pedi numeric
@attribute age numeric
@attribute class {tested_negative,tested_positive}

@data
6,148,72,35,0,33.6,0.627,50,tested_positive
1,85,66,29,0,26.6,0.351,31,tested_negative
8,183,64,0,0,23.3,0.672,32,tested_positive
1,89,66,23,94,28.1,0.167,21,tested_negative
```

Explanation:

Definition:	`System.out.println(data)` is standard output stream that displays value of the data for output `data` is the object of the class `Instances` and it invokes the function `getDataSet()` to load the dataset into memory with the reference of `data`.
Code:	`System.out.println(data);`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

b. Display the values of the first instance from the dataset

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

System.out.println(data.instance(1));
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
1,85,66,29,0,26.6,0.351,31,tested_negative
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

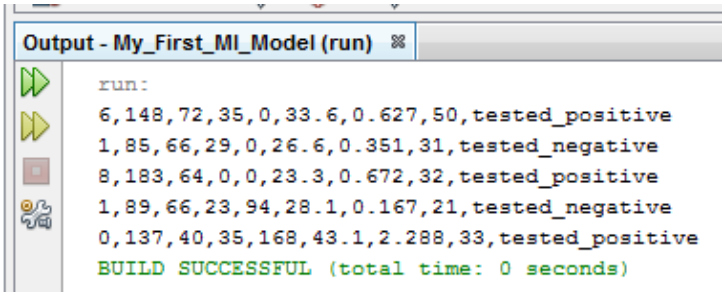
Definition:	`System.out.println(data)` is standard output stream that displays value of the data for output `data` is the object of the class `Instances` and it invokes the function `instance(1)` by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset.
Code:	`System.out.println(data.instance(1));`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

c. Display the first five instances from the dataset

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

 for (int i=0; i<=4; i++)
{
System.out.println(data.instance(i));
   }
```

Screenshot of output:



Explanation:

Definition:	`System.out.println(data)` is standard output stream that displays value of the data for output `data` is the object of the class `Instances` and it invokes the function `instance(1)` by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. `for` is the looping statement. Here, it helps to invoke the instances by passing the parameter 'i' for representing the index of the instance to the method `instance(i)`.

Code:	`for (int i=0; i<=4; i++)`
	`{`
	`System.out.println(data.instance(i));`
	`}`

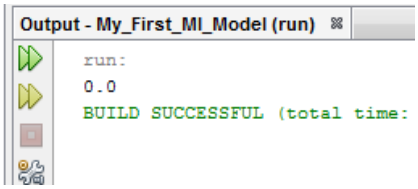
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html https://docs.oracle.com/javase/8/docs/technotes/guides/language/foreach.html

d. Display the class value of the first instance

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);


System.out.println(data.instance(1).classValue());
```

Screenshot of output:



Explanation:

Definition:	`System.out.println(data)` is standard output stream that displays value of the data for output `data` is the object of the class `Instances` and it invokes the function `instance(1)` by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. `classValue()` returns the class value of a particular index of an instance.
Code:	`System.out.println(data.instance(1).classValue());`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

e. Display the total number of attributes present in the dataset

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);


System.out.println(data.instance(1).numAttributes());
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
9
BUILD SUCCESSFUL (total time: 0 seconds)
```

Explanation:

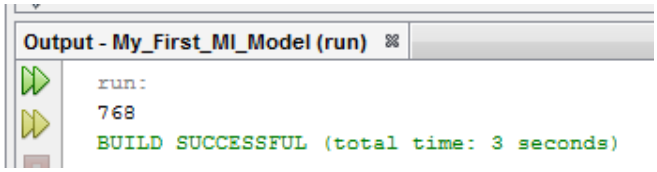
Definition:	`data` is the object of the class `Instances` and it invokes the function `instance(1)` by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. `numAttributes()` returns the total number of attributes present in the dataset.
Code:	`System.out.println(data.numAttributes());`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

f. Display the total number of instances present in the dataset

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);


System.out.println(data.numInstances());

   }}
```

Screenshot of output:



Explanation:

Definition:	`data` is the object of the class `Instances` and it invokes the function `instance(1)` by passing the parameter value '1' ('1' denotes the index number of instance of the dataset) to return the instance of index number '1' from the dataset. `numInstances()` returns the total number of instances present in the dataset.
Code:	`System.out.println(data. numInstances());`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

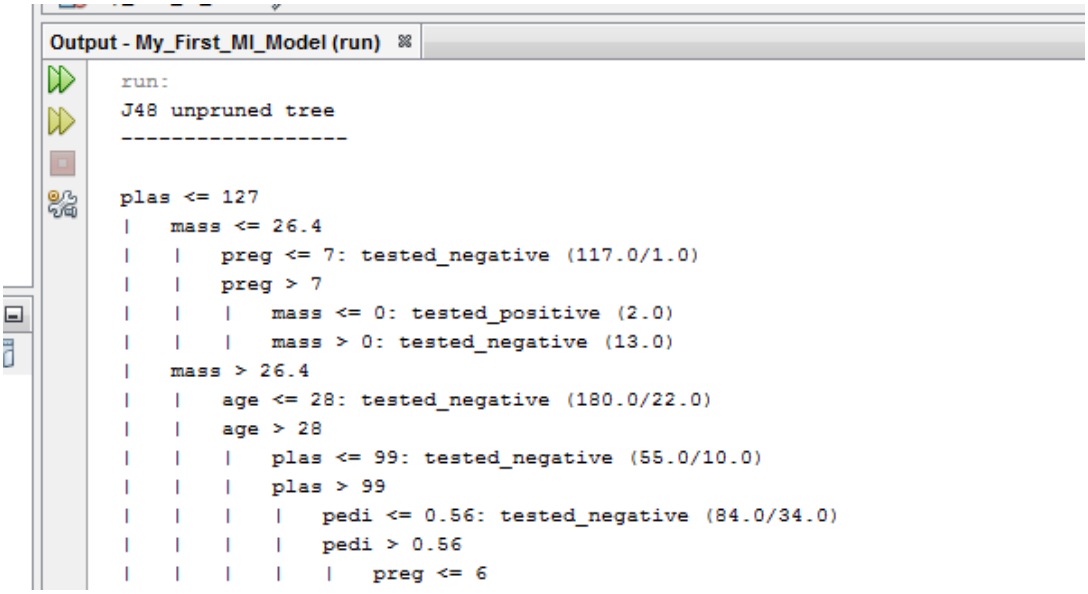
2. Develop and display different types of Ml modes

a. Develop and display the tree-based J48 Ml model

Screenshot of code:

```
public class My_First_Ml_Model {
    public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

String[] options = new String[1];
options[0] = "-U";
J48 tree = new J48();
tree.setOptions(options);
tree.buildClassifier(data);

System.out.println(tree);

   }}
```

Screenshot of output:



Explanation:

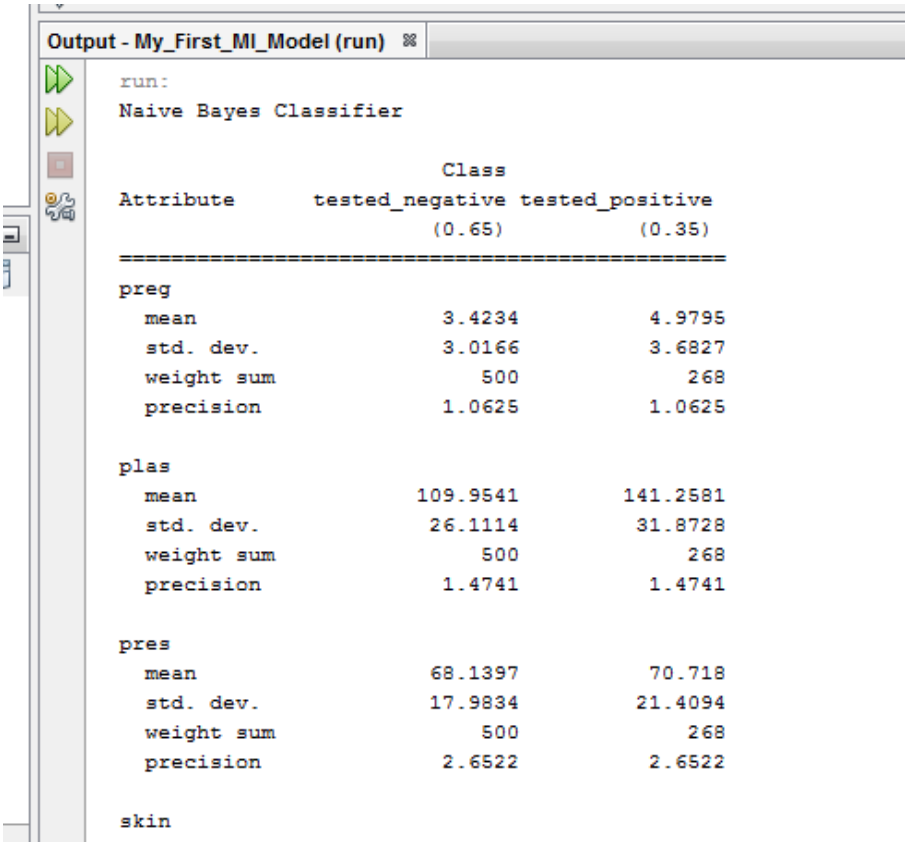
Definition:	`tree` is the object of the `J48` class that invokes the `buildClassifier(data)` method to develop the J48 Ml model
Code:	`System.out.println(tree);`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Develop and display the naïve Bayes Ml model

Screenshot of code:

```
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.classifiers.bayes.NaiveBayes;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instance;
import weka.core.Instances;

public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);
System.out.println(nb);

   }}
```

Screenshot of output:



Explanation:

Definition:	`nb` is the object of the `NaiveBayes` class that invokes the `buildClassifier(data)` method to develop the NaiveBayes Ml model Note: the following naïve Bayes classifier package must be imported before run the code. `import weka.classifiers.bayes.NaiveBayes;`
Code:	`import weka.classifiers.bayes.NaiveBayes;` `System.out.println(nb);`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/bayes/NaiveBayes.html

3. Perform the evaluation with different mode of test options

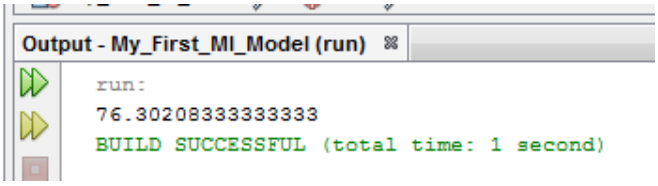
a. Develop naïve Bayes Ml model and display the accuracy (percentage of instances correctly predicted) in 10 fold cross validation

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
```

```
NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);


Evaluation eval = new Evaluation(data);
eval.crossValidateModel(nb, data, 10, new Random(1));

  System.out.println(eval.pctCorrect());
```

Screenshot of output:



Explanation:

Definition:	`eval` is the object of the `Evaluation` class that invokes the `crossValidateModel()` method and `pctCorrect()` method. `crossValidateModel(Classifier classifier, Instances data, int numFolds, java.util.Random random)` performs a cross-validation for a classifier on a set of instances. `crossValidateModel buildClas()to` return the percentage of instances correctly classified (that is, for which a correct prediction was made).

Code:	`import weka.classifiers.bayes.NaiveBayes;`
	`Evaluation eval = new Evaluation(data);`
	`eval. (nb, data, 10, new Random(1));`
	`System.out.println(eval.pctCorrect());`
	`System.out.println(nb);`

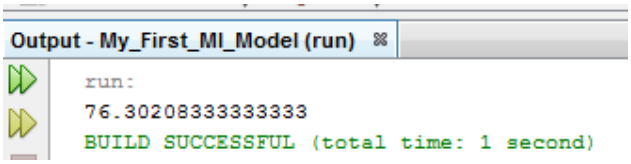
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

b. Develop naïve Bayes Ml model and display the accuracy (percentage of instances correctly predicted) and use the test set as training set.

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);


Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);

  System.out.println(eval.pctCorrect());
    }}
```

Screenshot of output:



Explanation:

Definition:	`eval.evaluateModel()` method evaluates the classifier on a given set of instances.
Code:	`eval.evaluateModel(nb, data);`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

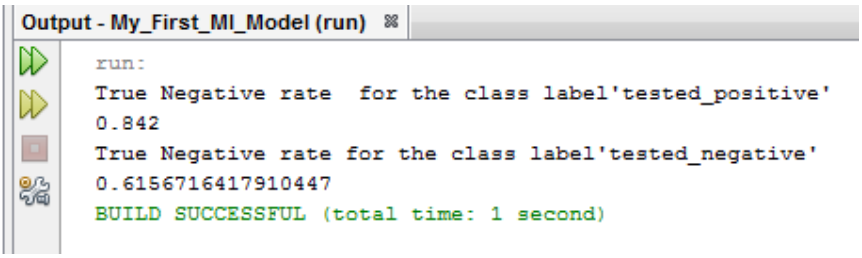
4. Display the performance of the Ml model using different performance metrics

a. Develop naïve Bayes Ml model and display the Develop naïve Bayes Ml model and display the TP Rate for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("True Negative rate  for the class label" + "'tested_positive'");
  System.out.println(eval.trueNegativeRate(1));
  System.out.println("True Negative rate for the class label" + "'tested_negative'");
  System.out.println(eval.trueNegativeRate(0));

    }}
```

Screenshot of output:



Explanation:

Definition:	`truePositiveRate(int classIndex)` calculates the true positive rate with respect to a particular class.
Code:	`Eval.truePositiveRate (1)` `Eval.truePositiveRate (0)`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

b. Develop naïve Bayes Ml model and display the Develop naïve Bayes Ml model and display the TN Rate for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("True Negative rate  for the class label" + "'tested_positive'");
  System.out.println(eval.trueNegativeRate(1));
  System.out.println("True Negative rate for the class label" + "'tested_negative'");
  System.out.println(eval.trueNegativeRate(0));

    }}
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
True Negative rate  for the class label'tested_positive'
0.842
True Negative rate for the class label'tested_negative'
0.6156716417910447
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition:	`trueNegativeRate(int classIndex)` calculates the true negative rate with respect to a particular.
Code:	`Eval.trueNegativeRate(1)` `eval.trueNegativeRate(0)`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

c. Develop naïve Bayes Ml model and display the Develop naïve Bayes Ml model and display the FP for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("False positive rate  for the class label" + "'tested_positive'");
  System.out.println(eval.falseNegativeRate(1));
  System.out.println("False positive rate for the class label" + "'tested_negative'");
  System.out.println(eval.falseNegativeRate(0));

    }}
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
False positive rate  for the class label'tested_positive'
0.3843283582089552
False positive rate for the class label'tested_negative'
0.158
BUILD SUCCESSFUL (total time: 1 second)
```

Explanation:

Definition	`falsePositiveRate(int classIndex)` calculate the false positive rate with respect to a particular class.
Part of the Code:	`System.out.println(eval.falseNegativeRate(1));` `System.out.println(eval.falseNegativeRate(0));`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

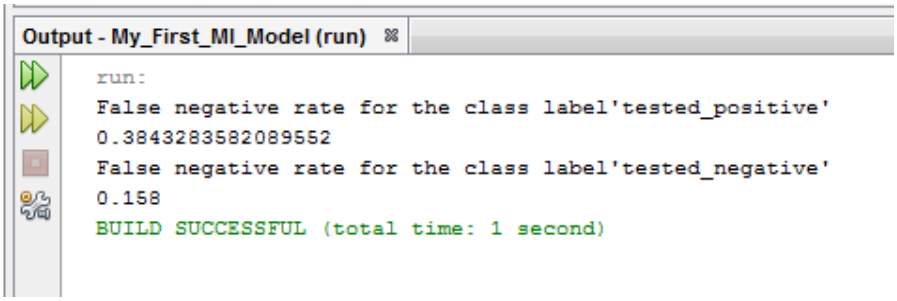
d. Develop naïve Bayes Ml model and display the Develop naïve Bayes Ml model and display the FN Rate for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {
```

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("False negative rate for the class label" + "'tested_positive'");
  System.out.println(eval.falseNegativeRate(1));
  System.out.println("False negative rate for the class label" + "'tested_negative'");
  System.out.println(eval.falseNegativeRate(0));

   }}
```

Screenshot of output:



Explanation:

Definition:	`falseNegativeRate(int classIndex)` method calculates the false negative rate with respect to a particular class.
Code:	`eval.falseNegativeRate(1)` `eval.falseNegativeRate(0)`
Ref. URL;	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

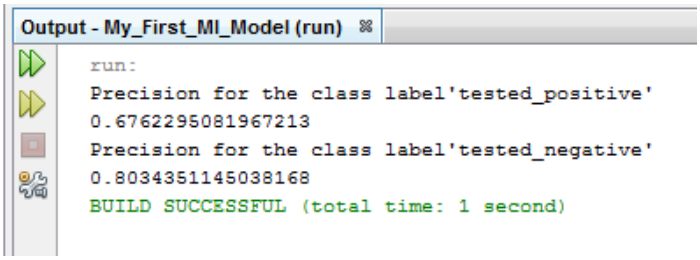
a. Develop naïve Bayes Ml model and display the Precision for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);
```

```
Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("Precision for the class label" + "'tested_positive'");
  System.out.println(eval.precision(1));
  System.out.println("Precision for the class label" + "'tested_negative'");
  System.out.println(eval.precision(0));

   }}
```

Screenshot of output:



Explanation:

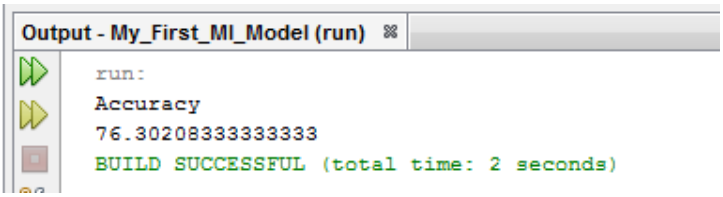
Definition:	`precision(int classIndex)` method calculates the precision with respect to a particular class.
Code:	`System.out.println(eval.precision(1));` `System.out.println(eval.precision(0));`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

a. Develop naïve Bayes Ml model and display the accuracy (percentage of the correctly predicted instance) for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("Accuracy ");

  System.out.println(eval. pctCorrect());

   }}
```

Screenshot of output:



Explanation:

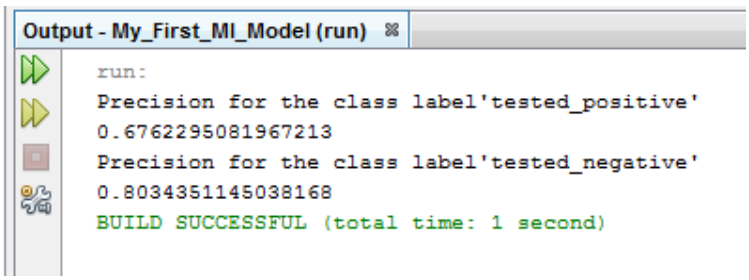
Definition:	`pctCorrect()` returns the percentage of instances correctly classified (that is, for which a correct prediction was made).
Code:	`System.out.println(eval. pctCorrect());`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

b. Develop naïve Bayes Ml model and display the Precision for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("Precision for the class label" + "'tested_positive'");
  System.out.println(eval.precision(1));
  System.out.println("Precision for the class label" + "'tested_negative'");
  System.out.println(eval.precision(0));

    }}
```

Screenshot of output:



Explanation:

Definition:	`precision(int classIndex)` method calculates the precision with respect to a particular class.
Code:	`System.out.println(eval.precision(1));`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

e. evelop naïve Bayes Ml model and display the Recall for the class labels (test mode is use training set as test set)

Screenshot of code:

```
public class My_First_Ml_Model {
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

NaiveBayes nb = new NaiveBayes();
nb.buildClassifier(data);

Evaluation eval = new Evaluation(data);
eval.evaluateModel(nb, data);
  System.out.println("Recall rate for the class label" + "'tested_positive'");

  System.out.println(eval.recall(1));
  System.out.println("Recall rate for the class label" + "'tested_negative'");
  System.out.println(eval.recall(0));
   }}
```

Screenshot of output:

```
Output - My_First_Ml_Model (run)
run:
Recall rate for the class label'tested_positive'
0.6156716417910447
Recall rate for the class label'tested_negative'
0.842
BUILD SUCCESSFUL (total time: 1 second)
```

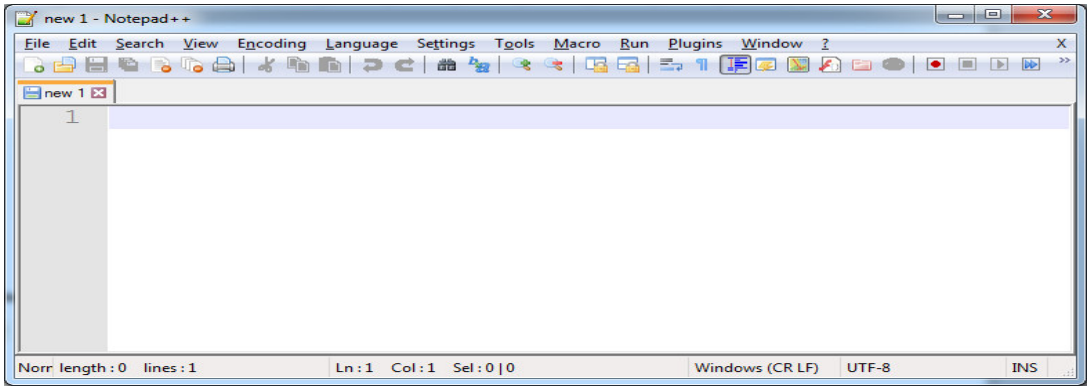
Explanation:

Definition:	`recall(int classIndex)` method Calculate the recall with respect to a particular class.
Code:	`System.out.println(eval.recall(1));`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

Chapter 4
Prediction using Ml model

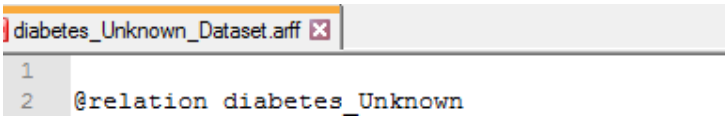
Part I
Preparing the unknown (unlabeled) dataset

1. **Open Notepad/Notepad++ and**



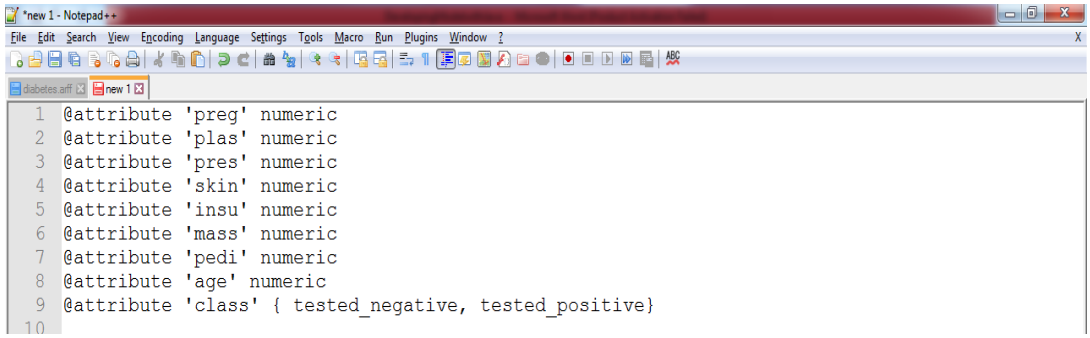
2. **Define the name of the dataset:**

   Syntax: @ relation <dataset_name>
   Code: @relation diabetes_Unknown



3. **Define the attributes and class label of the dataset:**

   Syntax: @ attribute <name of the attribute > <attribute type>
   Syntax: @ attribute <class label name > <attribute type>
   Code: @attribute 'preg' numeric
   @attribute 'plas' numeric
   @attribute 'pres' numeric
   @attribute 'skin' numeric
   @attribute 'insu' numeric
   @attribute 'mass' numeric
   @attribute 'pedi' numeric
   @attribute 'age' numeric
   @attribute 'class' { tested_negative, tested_positive}



4. Define the instances of the dataset:

Syntax:

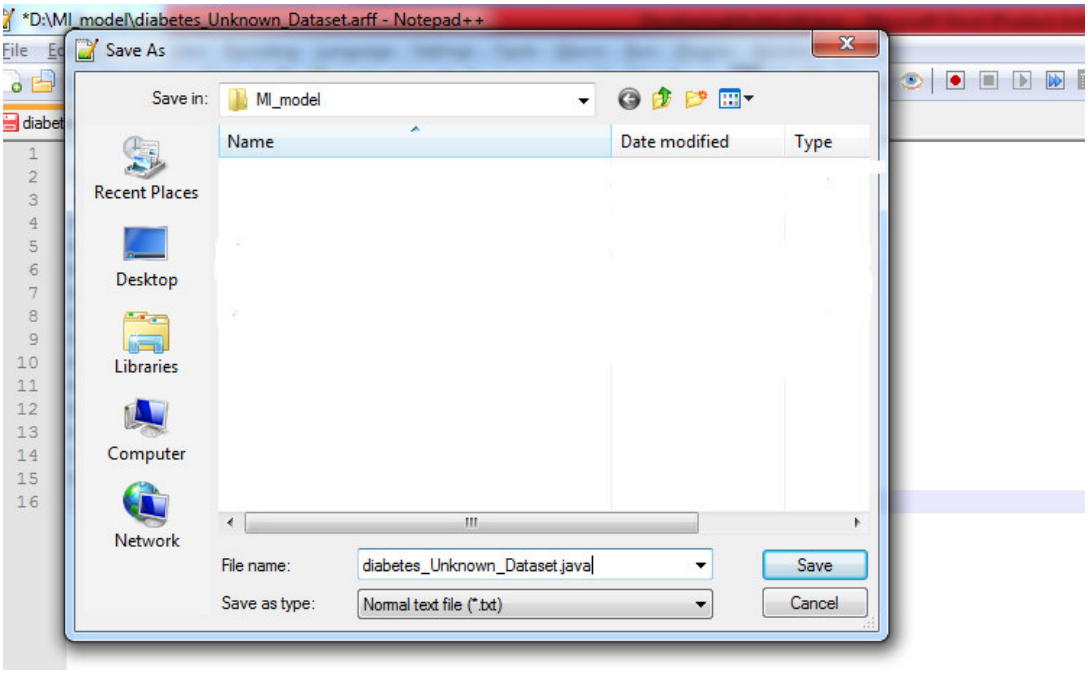
@data
< value for attribute 1, value for attribute 2, ….. value for attribute n>

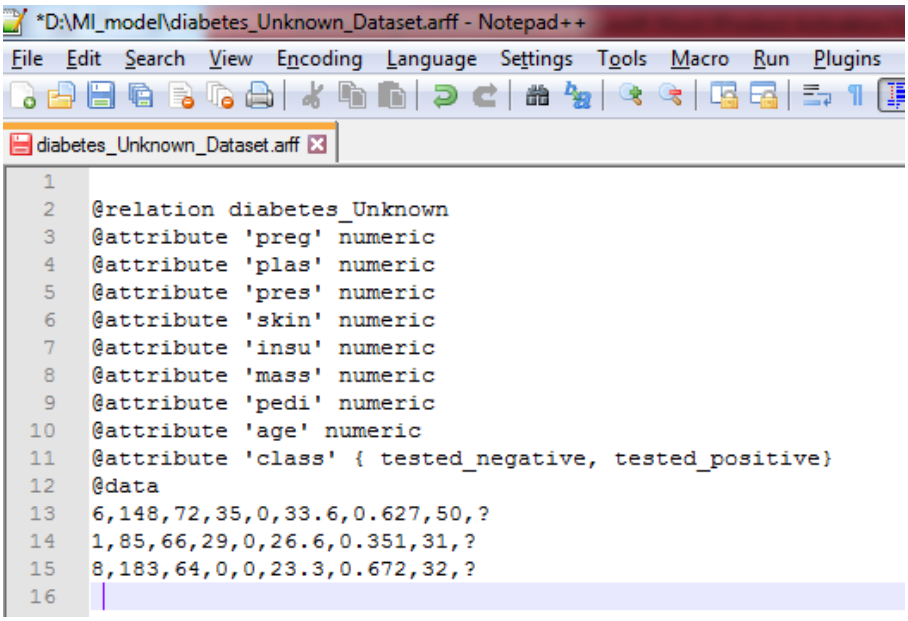
Code:

@data
6,148,72,35,0,33.6,0.627,50,?
1,85,66,29,0,26.6,0.351,31,?
8,183,64,0,0,23.3,0.672,32,?

```
@relation diabetes_Unknown
@attribute 'preg' numeric
@attribute 'plas' numeric
@attribute 'pres' numeric
@attribute 'skin' numeric
@attribute 'insu' numeric
@attribute 'mass' numeric
@attribute 'pedi' numeric
@attribute 'age' numeric
@attribute 'class' { tested_negative, tested_positive}
@data
6,148,72,35,0,33.6,0.627,50,?
1,85,66,29,0,26.6,0.351,31,?
8,183,64,0,0,23.3,0.672,32,?
```

5. Save the file with the file extension of arff shown in screenshots below:




*D:\MI_model\diabetes_Unknown_Dataset.arff - Notepad++
File Edit Search View Encoding Language Settings Tools Macro Run Plugins
diabetes_Unknown_Dataset.arff
1
2 @relation diabetes_Unknown
3 @attribute 'preg' numeric
4 @attribute 'plas' numeric
5 @attribute 'pres' numeric
6 @attribute 'skin' numeric
7 @attribute 'insu' numeric
8 @attribute 'mass' numeric
9 @attribute 'pedi' numeric
10 @attribute 'age' numeric
11 @attribute 'class' { tested_negative, tested_positive}
12 @data
13 6,148,72,35,0,33.6,0.627,50,?
14 1,85,66,29,0,26.6,0.351,31,?
15 8,183,64,0,0,23.3,0.672,32,?
16

Part II
Developing the Ml Models and Perform Prediction

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. Machine learning model is developed from the loaded training dataset and the model is saved in the local disk or cloud. Then, model and unknown (unlabeled) dataset are loaded. The prediction is performed using the model on the unknown dataset and the predicted results are displayed.

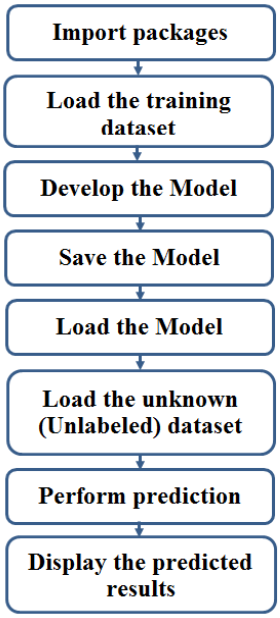


Figure: Steps to develop a Java code for performing prediction using Ml models

Step I: Develop the Java code to construct and predict label of the unknown data Ml models

1. Import the required packages:

a. Import package to handle the files

Definition:	`File(String pathname)` creates a new File instance by converting the given pathname string into an abstract pathname.
Code:	`import java.io.File;`
Ref. URL	https://docs.oracle.com/javase/8/docs/api/java/io/File.html

b. Import Package to develop the Ml model (J48)

Definition:	`J48` class is used to generate decision tree-based machine learning model.
Code:	`import weka.classifiers.trees.J48;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

c. Import package to access the dataset from a file

Definition:	`DataSource` class is used loading data from files
Code:	`Import weka.core.converters.ConverterUtils.DataSource;`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

d. Import package to load the Ml models

Definition:	`SerializedClassifier` loads serialized models and uses it to make predictions.
Code:	`import weka.classifiers.misc.SerializedClassifier;`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/misc/SerializedClassifier.html

e. Import package for debugging

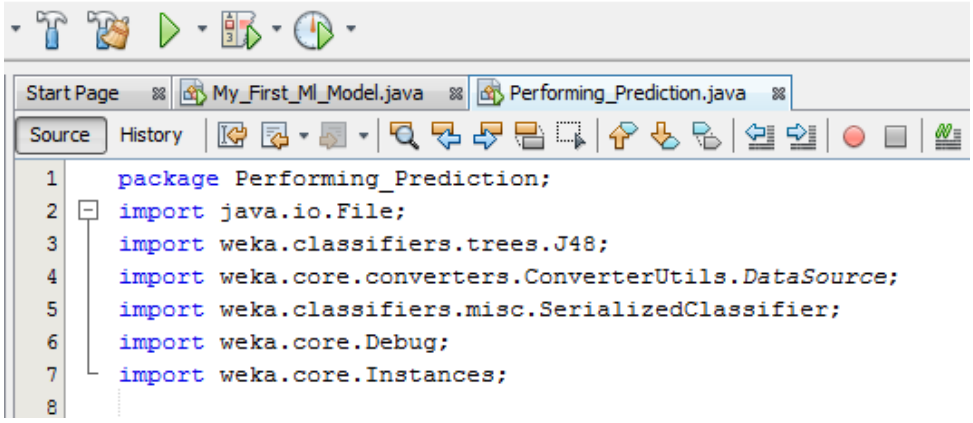
Definition:	`Debug` class is used for debug output, logging, clocking, etc.
Code:	`import weka.core.Debug;`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

f. Import package to handle the instance of the dataset

Definition	`Instances` class is used to handle the ordered set of weighted instances.
Code:	`import weka.core.Instances;`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to import required classes:

```
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;
```



Step II: Define the Main class and the Main method

Note: Below a Java come is provided for displaying the string "HelloWorld"

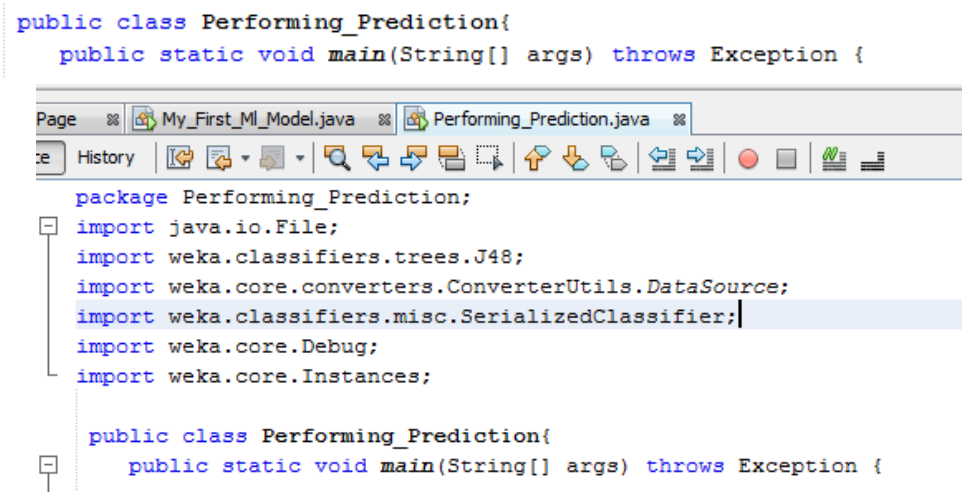
```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the
string.
    }
}
```

- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Definition:	`public` class: Any other class can access a public field or method. `public` method: This method is public and therefore available to anyone. `static` method: This method can be run without having to create an instance of the class MyClass. `void` method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.
Code:	`public class Performing_Prediction{` `  public static void main(String[] args) throws Exception {`
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

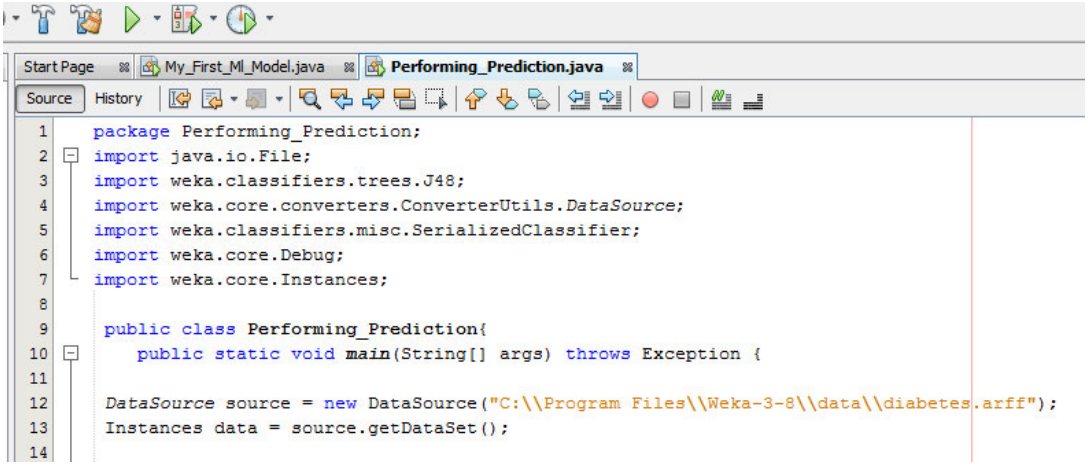


Step III: Load the dataset from the file

Definition:	`DataSource` class is used to loading data from files `getDataSet()` method returns the full dataset, can be null in case of an error. In this code, an object `source` is created for the class `DataSource` and the object `source` is used to invoke the method `getDataSet()` using dot operator Note: Provide the entire path of the dataset which should be loaded
Code:	`DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");` `Instances data = source.getDataSet();`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();
```

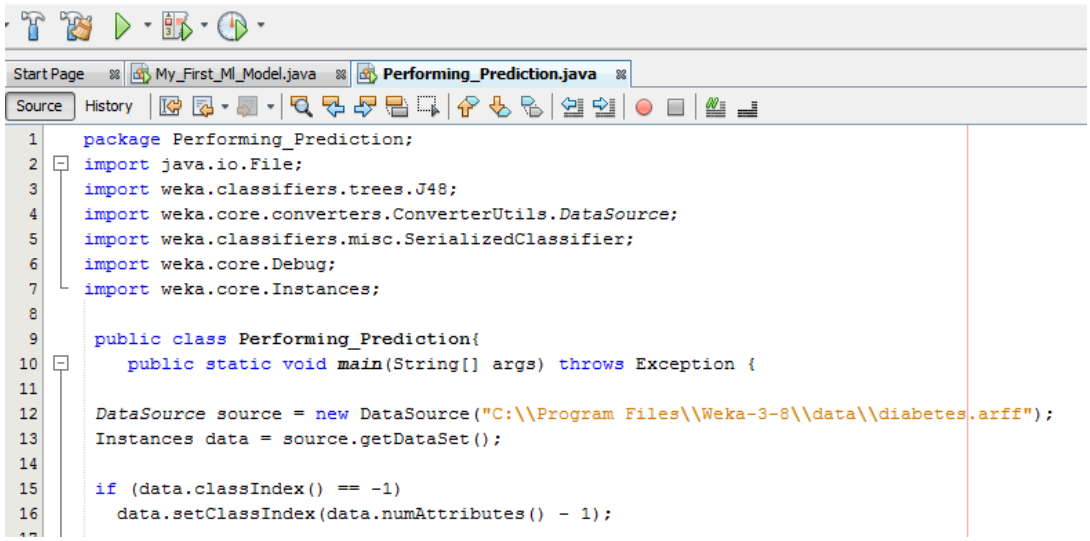


Step IV: Set the class attribute for the dataset

Definition:	`classIndex()` Returns the class attribute's index (default it is set -1). `setClassIndex(int classIndex)` Sets the class index of the dataset. `numAttributes()` Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object `data` which is created for the class `Instances`.
Code:	`if (data.classIndex() == -1)` `  data.setClassIndex(data.numAttributes() - 1);`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset

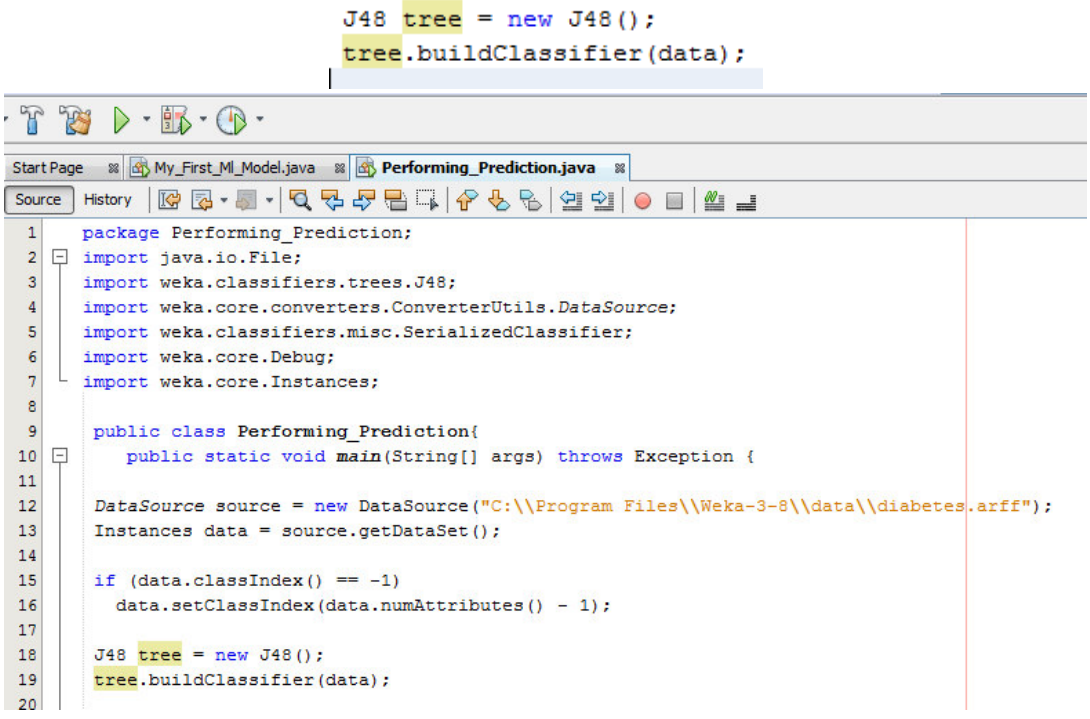
```
if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
```



Step V: Develop the Machine Learning Model from the dataset

Definition:	`J48` class for developing a pruned or unpruned C4.5 decision tree `buildClassifier` method generates the classifier (Machine learning model) Note: `tree` is the object of the class `J48` and the data is the object of the class `DataSource`
Code:	`J48 nb = new J48();` `nb.buildClassifier(data);`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

Snapshot of code to develop the Machine Learning Model from the dataset

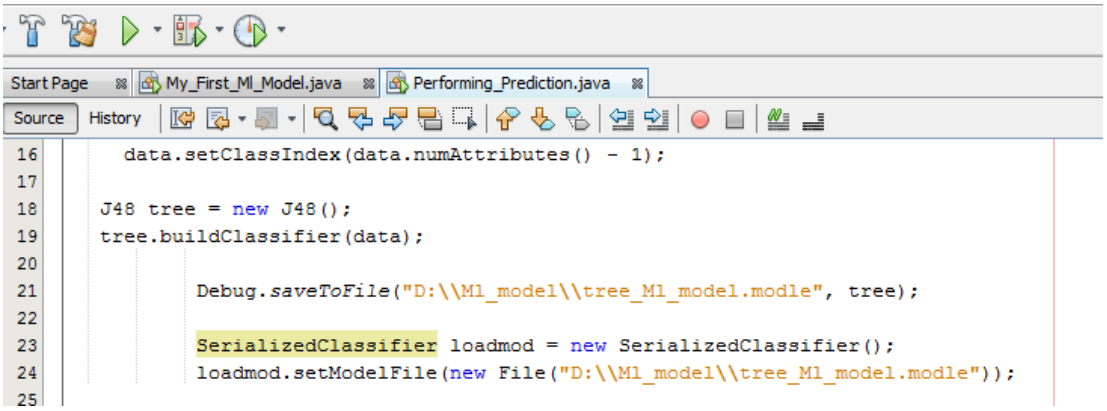


Step VI: Save the model

Definition:	`Debug` class is used for debug output, logging, clocking, etc. `saveToFile()` writes the serialized object to the speicified file
Code:	`Debug.saveToFile("D:\\Ml_model\\nb_Ml_model.modle", tree);`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);
```

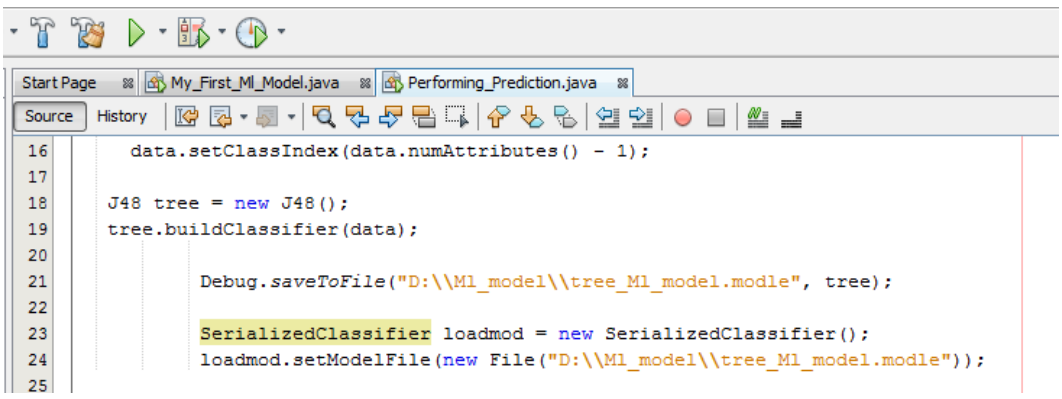


Step III: Load the model

Definition:	`SerializedClassifier` loads serialized models and uses it to make predictions. `setModelFile(java.io.File value)` sets the file containing the serialized model. `File(String pathname)` creates a new File instance by converting the given pathname string into an abstract pathname.
Code:	`SerializedClassifier loadmod = new SerializedClassifier();` `loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/misc/SerializedClassifier.html https://docs.oracle.com/javase/8/docs/api/java/io/File.html

Snapshot of code to load the full dataset

```
SerializedClassifier loadmod = new SerializedClassifier();
loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));
```



Step III: Load the unknown dataset from the file

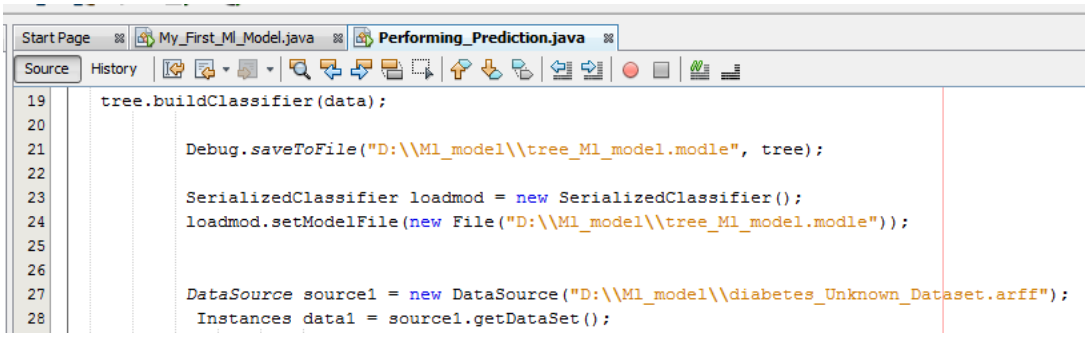
Definition:	`DataSource` class is used to loading data from files `getDataSet()` method returns the full dataset, can be null in case of an error. In this code, an object `source1` is created for the class `DataSource` and the object `source1` is used to invoke the method `getDataSet()` using dot operator Note: Provide the entire path of the dataset which should be loaded

Code:	`DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");`
	`Instances data1 = source1.getDataSet();`
	`if (data1.classIndex() == -1`
	`data1.setClassIndex(data1.numAttributes() - 1);`

Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
 Instances data1 = source1.getDataSet();
```

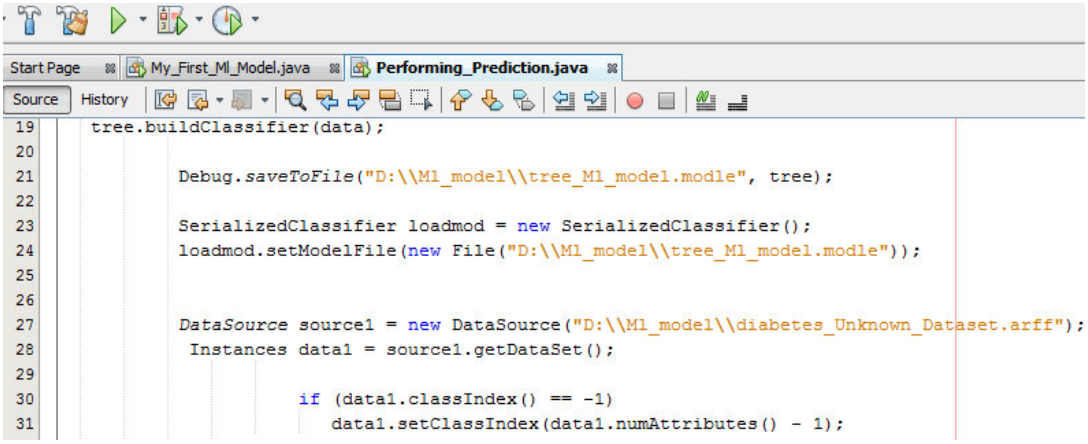


Step IV: Set the class attribute for the dataset

Definition:	`classIndex()` Returns the class attribute's index (default it is set -1). `setClassIndex(int classIndex)` Sets the class index of the dataset. `numAttributes()` Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object `data1` which is created for the class `Instances`.
Code:	`if (data1.classIndex() == -1)` `data1.setClassIndex(data1.numAttributes() - 1);`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
if (data1.classIndex() == -1)
   data1.setClassIndex(data1.numAttributes() - 1);
```

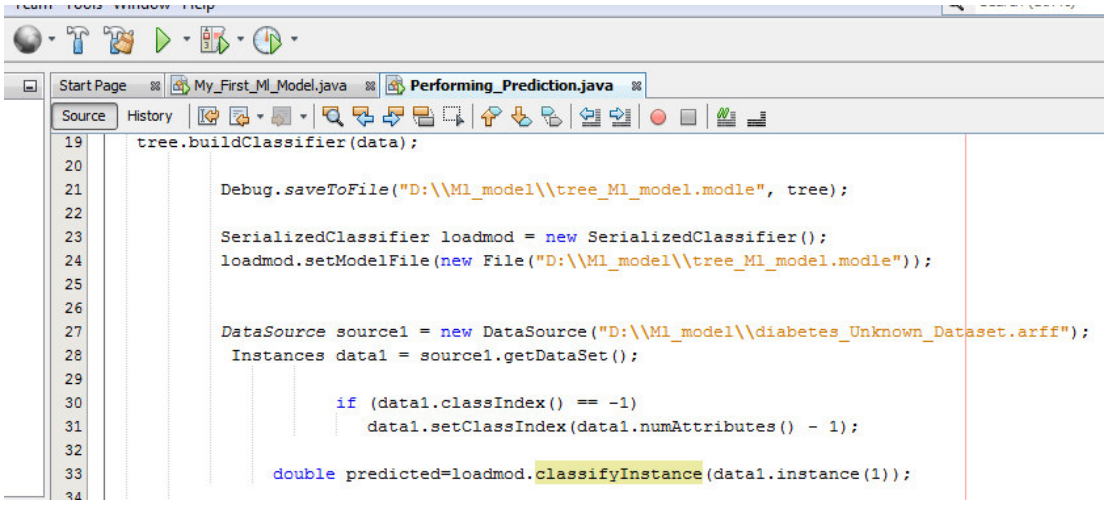


Step IV: Perform the prediction

Definition:	`classifyInstance(Instance instance)` classifies the given test instance. `instance(int index)` returns the instance at the given position.
Code:	`double predicted=loadmod.classifyInstance(data1.instance(1));`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Classifier.html http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

Snapshot of code to set the class attribute for the dataset

```
double predicted=loadmod.classifyInstance(data1.instance(1));
```

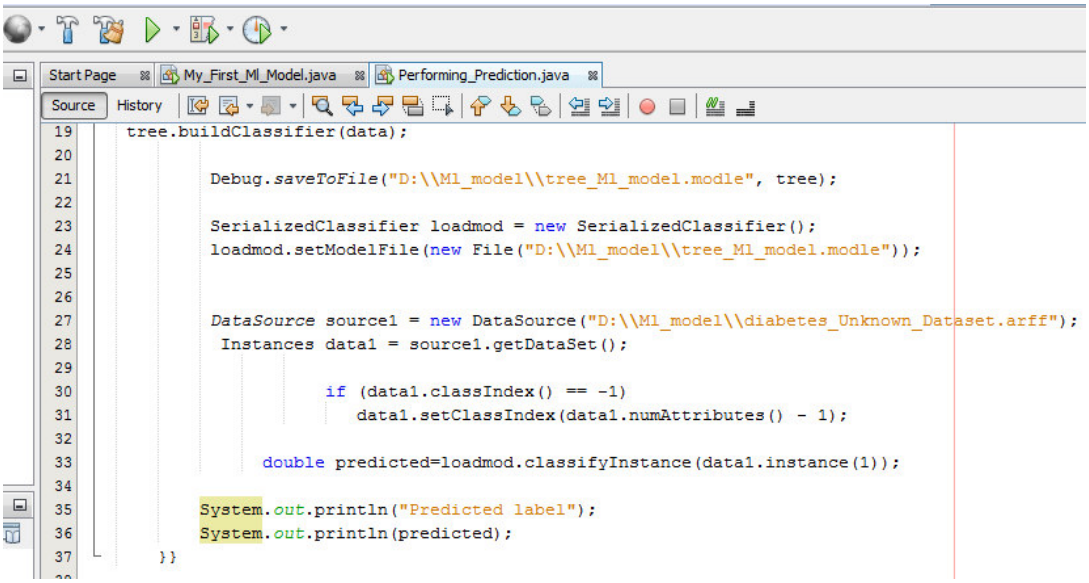


Step VI: Display the predicted label

Definition	`System.out.println(predicted)` is standard output stream that displays value of the variable predicted for output
Code:	`System.out.println("Predicted label");` `System.out.println(predicted);`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/core/Debug.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
System.out.println("Predicted label");
System.out.println(predicted);
```



Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

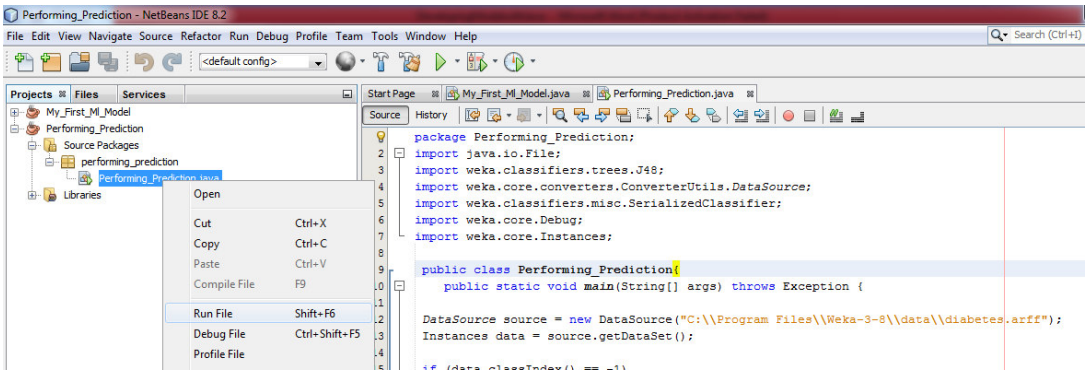
Snapshot of complete code to develop and evaluate the Machine Learning Model from the dataset

```
package Performing_Prediction;
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;

public class Performing_Prediction{
   public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();

if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);

J48 tree = new J48();
tree.buildClassifier(data);

        Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle", tree);

        SerializedClassifier loadmod = new SerializedClassifier();
        loadmod.setModelFile(new File("D:\\Ml_model\\tree_Ml_model.modle"));


        DataSource source1 = new DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
         Instances data1 = source1.getDataSet();

                if (data1.classIndex() == -1)
                   data1.setClassIndex(data1.numAttributes() - 1);

            double predicted=loadmod.classifyInstance(data1.instance(1));

        System.out.println("Predicted label");
        System.out.println(predicted);
    }}
```

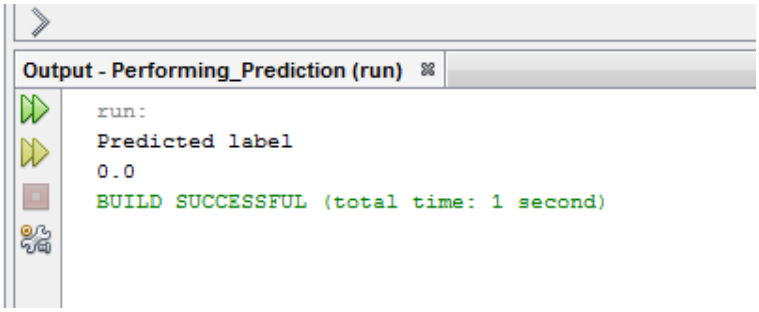
```
package Performing_Prediction;
import java.io.File;
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.misc.SerializedClassifier;
import weka.core.Debug;
import weka.core.Instances;

 public class Performing_Prediction{
    public static void main(String[] args) throws Exception {

 DataSource source = new DataSource("C:\\Program Files\\Weka-3-
8\\data\\diabetes.arff");
 Instances data = source.getDataSet();

 if (data.classIndex() == -1)
   data.setClassIndex(data.numAttributes() - 1);

 J48 tree = new J48();
 tree.buildClassifier(data);

         Debug.saveToFile("D:\\Ml_model\\tree_Ml_model.modle",
tree);

         SerializedClassifier loadmod = new SerializedClassifier();
         loadmod.setModelFile(new
File("D:\\Ml_model\\tree_Ml_model.modle"));

         DataSource source1 = new
DataSource("D:\\Ml_model\\diabetes_Unknown_Dataset.arff");
          Instances data1 = source1.getDataSet();

            if (data1.classIndex() == -1)
                 data1.setClassIndex(data1.numAttributes() - 1);

double predicted=loadmod.classifyInstance(data1.instance(1));

        System.out.println("Predicted label");
        System.out.println(predicted);
    }}
```

Step VII: Run the code and display the results

Do right-click on the Java class file named as “Performing_Prediction” one drop down menu is appeared and click on the “Run File” option to run the code and display the results as shown in the screenshot below. Or Simply Press the keys (Shift+F6) to run the code display the results.



Note: The results are appeared in the output window as shown in screenshot below



Part II
Additional Workouts

1. Develop a Multilayer Perceptron Ml Model on the dataset weather.nominal and perform the display the model.

Screenshot of code:

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\weather.nominal.arff");
Instances data = source.getDataSet();
   //set the class attribute
if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
//develop model
MultilayerPerceptron mlp = new MultilayerPerceptron();
mlp.buildClassifier(data);

 System.out.println(mlp);


   }}
```

Screenshot of output:

```
run:
Sigmoid Node 0
    Inputs    Weights
    Threshold    -4.597967080790812
    Node 2    2.433270074007242
    Node 3    2.0546443732203774
    Node 4    1.364159803860347
    Node 5    2.6974766889493536
    Node 6    3.9083227090643557
Sigmoid Node 1
    Inputs    Weights
    Threshold    4.60125196001115
    Node 2    -2.404522637307116
    Node 3    -2.053274495614413
    Node 4    -1.3799864297539497
    Node 5    -2.756274547604192
    Node 6    -3.877948258791869
Sigmoid Node 2
    Inputs    Weights
    Threshold    -0.15507980215013525
    Attrib outlook=sunny    -1.3234644779136855
    Attrib outlook=overcast    1.660267528039988
    Attrib outlook=rainy    -0.32078025528655973
    Attrib temperature=hot    -0.28731224569818364
```

Explanation:

Definition:	`MultilayerPerceptron` uses backpropagation to learn a multi-layer perceptron to classify instances `buildClassifier` method generates the classifier (Machine learning model) Note: `mlp` is the object of the class `MultilayerPerceptron` and the `data` is the object of the class `DataSource` `System.out.println(mlp);` displays value model MLP

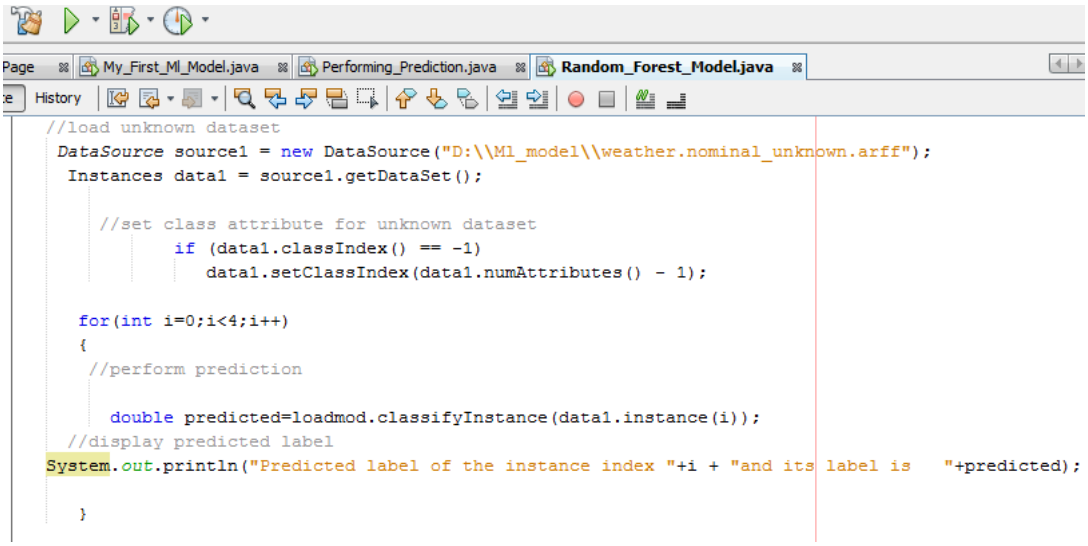
Code:	`MultilayerPerceptron mlp = new MultilayerPerceptron();`
	`mlp.buildClassifier(data);`
	`System.out.println(mlp);`

Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/functions/MultilayerPerceptron.html

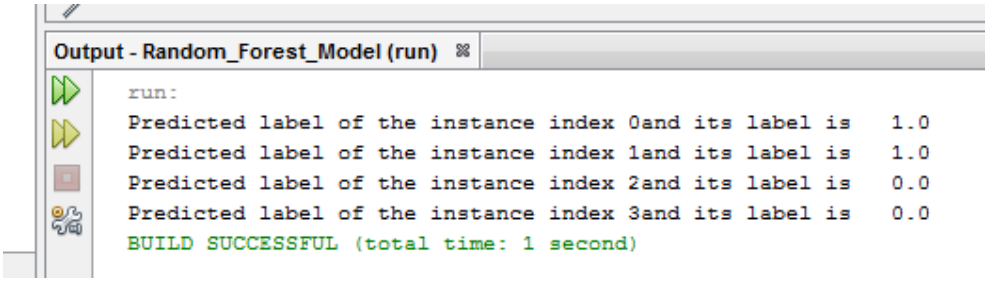
b. Develop a multilayer perceptron Ml Model on the dataset weather. nominal and perform prediction on the 5 unknown instances and display their class label.

```
@relation weather.symbolic

@attribute outlook {sunny, overcast, rainy}
@attribute temperature {hot, mild, cool}
@attribute humidity {high, normal}
@attribute windy {TRUE, FALSE}
@attribute play {yes, no}

@data
sunny,hot,high,FALSE,?
sunny,hot,high,TRUE,?
overcast,hot,high,FALSE,?
rainy,mild,high,FALSE,?
rainy,cool,normal,FALSE,?
```

Screenshot of code:



Screenshot of output:



Explanation:

Definition:	`for` statement provides a way to iterate over a range of values. `classifyInstance(Instance instance)` classifies the given unknown instance.

Code:	`for(int i=0;i<4;i++)`
	`{`
	`double predicted=loadmod.classifyInstance(data1.instance(i));`
	
	`System.out.println("Predicted label of the instance index "+i + "and its label is "+predicted);`

Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html https://docs.oracle.com/javase/tutorial/java/nutsandbolts/for.html

Chapter 4 Performing feature selection

Part I
Performing feature selection

Outline of the program

The steps to developed the java program is shown in the below Figure. Initially, the required packages are imported. The significant features are selected from the loaded training dataset. Machine learning model is developed from the selected feature from the training dataset. The performances are evaluated on the developed model and the results are displayed.

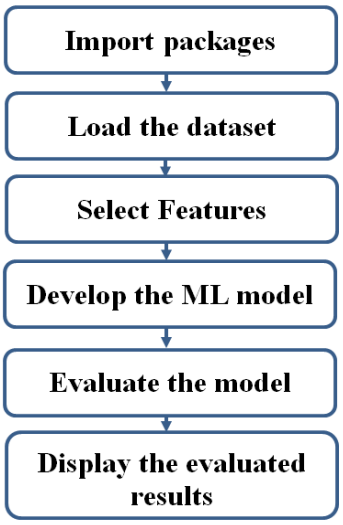


Figure: Steps to develop a Java code for performing feature selection

Step I: Develop the Java code to construct and predict label of the unknown data Ml models

1. Import the required packages:

a. Import Package to develop the Ml model (J48)

Definition:	`J48` class is used to generate decision tree-based machine learning model.
Code:	`import weka.classifiers.trees.J48;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html

b. Import package to access the dataset from a file

Definition:	`DataSource` class is used to load the data from files
Code:	`Import weka.core.converters.ConverterUtils.DataSource;`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

c. Import package to evaluating the performance of the Ml model

Definition	`Evaluation` class for evaluating machine learning models
Code:	`import weka.classifiers.Evaluation;`
Ref. URL	http://weka.sourceforge.net/doc.dev/weka/classifiers/Evaluation.html

d. Import package to generate the random number

Definition:	`Random` class is used to generate a stream of pseudorandom numbers
Code:	`import java.util.Random;`
Ref. URL:	https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

e. Import package to handle the instance of the dataset

Definition:	`Instances` class is used to handle the ordered set of weighted instances.
Code:	`import weka.core.Instances;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/core/Instances.html

f. Import package to select the correlated features

Definition:	`CfsSubsetEval` forms subsets of features that are highly correlated with the class while having low intercorrelation are preferred.
Code:	`import weka.attributeSelection.CfsSubsetEval;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CfsSubsetEval.html

g. Import package to handle the instance of the dataset

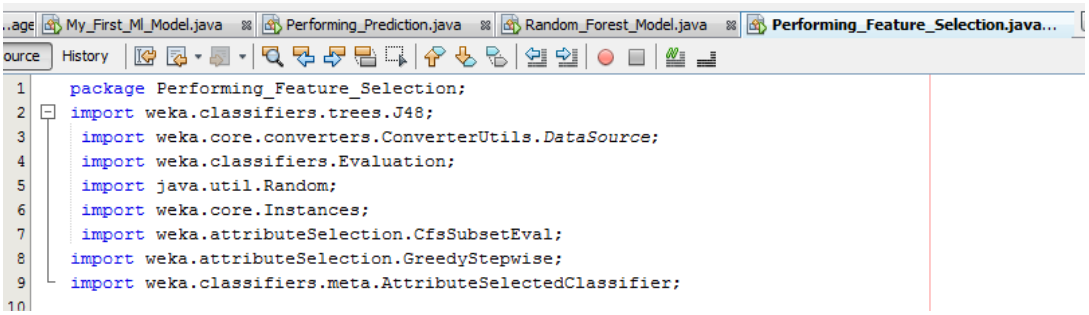
Definition:	`GreedyStepwise` performs a greedy forward or backward search through the space of attribute subsets
Code:	`import weka.attributeSelection.GreedyStepwise;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/attributeSelection/GreedyStepwise.html

h. Import package to select the correlated features

Definition:	`AttributeSelectedClassifier` used to reduce the dimensionality of training dataset by attribute selection before being passed on to a classifier.
Code:	`import weka.classifiers.meta.AttributeSelectedClassifier;`
Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AttributeSelectedClassifier.html

Snapshot of code to import required classes:

```
import weka.classifiers.trees.J48;
 import weka.core.converters.ConverterUtils.DataSource;
 import weka.classifiers.Evaluation;
 import java.util.Random;
 import weka.core.Instances;
 import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;
```



Step II: Define the Main class and the Main method

Note: Below a Java come is provided for displaying the string "HelloWorld"

```
public class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

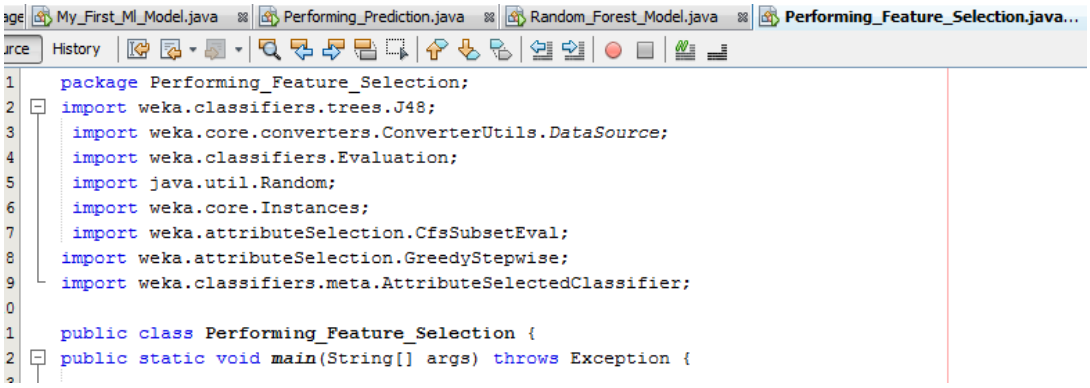
- public class: Any other class can access a public field or method.
- public method: This method is public and therefore available to anyone.
- static method: This method can be run without having to create an instance of the class MyClass.
- void method: This method does not return anything.
- (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray".

Ref. URL: https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Definition:	`public` class: Any other class can access a public field or method. `public` method: This method is public and therefore available to anyone. `static` method: This method can be run without having to create an instance of the class MyClass. `void` method: This method does not return anything. (String[] args): This method takes a String argument. Note that the argument args can be anything — it's common to use "args" but we could instead call it "stringArray". The throws keyword indicates what exception type may be thrown by a method.
Code:	`public class Performing_Prediction{` `public static void main(String[] args) throws Exception {`
Ref. URL:	https://docs.oracle.com/javase/tutorial/getStarted/application/index.html

Snapshot of code to define the Main class and the Main method

```
public class Performing_Prediction{
    public static void main(String[] args) throws Exception {
```

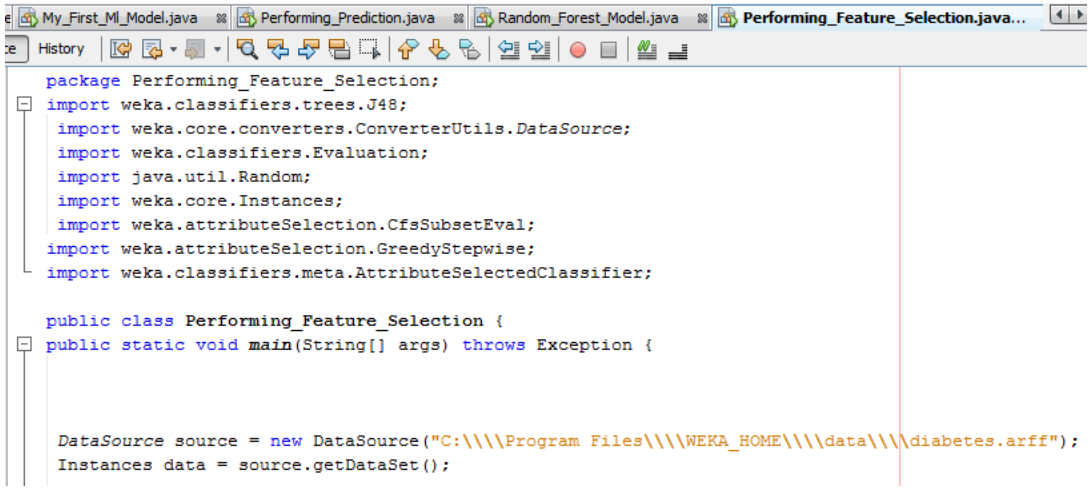


Step III: Load the dataset from the file

Definition:	`DataSource` class is used to loading data from files `getDataSet()` method returns the full dataset, can be null in case of an error. In this code, an object `source` is created for the class `DataSource` and the object `source` is used to invoke the method `getDataSet()` using dot operator Note: Provide the entire path of the dataset which should be loaded
Code:	`DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");` `Instances data = source.getDataSet();`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to load the full dataset

```
DataSource source = new DataSource("C:\\Program Files\\Weka-3-8\\data\\diabetes.arff");
Instances data = source.getDataSet();
```



Step IV: Set the class attribute for the dataset

Definition:	`classIndex()` Returns the class attribute's index (default it is set -1). `setClassIndex(int classIndex)` Sets the class index of the dataset.

	`numAttributes()` Returns the total number of attributes of the dataset Note: all the above methods are invoked by the object `data` which is created for the class `Instances`.
Code:	`if (data.classIndex() == -1)` `  data.setClassIndex(data.numAttributes() - 1);`
Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/core/converters/ConverterUtils.DataSource.html

Snapshot of code to set the class attribute for the dataset

```
if (data.classIndex() == -1)
  data.setClassIndex(data.numAttributes() - 1);
```

My_First_Ml_Model.java | Performing_Prediction.java | Random_Forest_Model.java | Performing_Feature_Selection.java...

History

```
package Performing_Feature_Selection;
import weka.classifiers.trees.J48;
 import weka.core.converters.ConverterUtils.DataSource;
 import weka.classifiers.Evaluation;
 import java.util.Random;
 import weka.core.Instances;
 import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {



 DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
 Instances data = source.getDataSet();

  if (data.classIndex() == -1)
   data.setClassIndex(data.numAttributes() - 1);
```

Step V: Perform feature selection

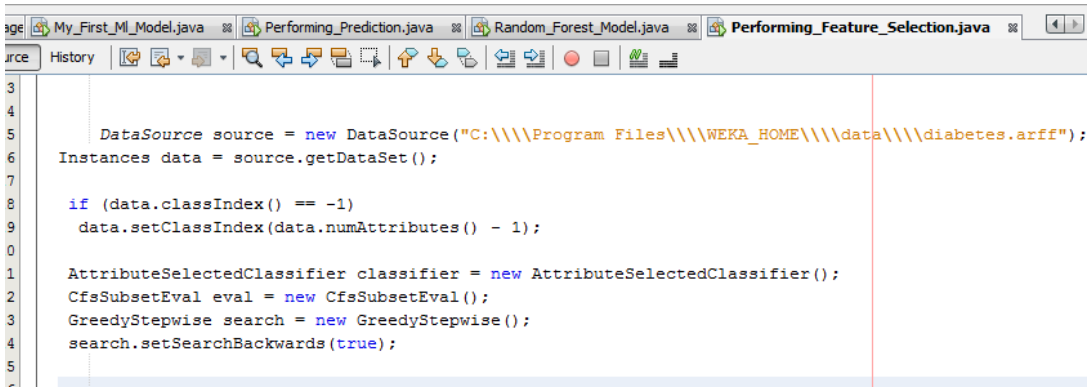
Definition:	`AttributeSelectedClassifier` used to reduce the dimensionality of training dataset by attribute selection before being passed on to a classifier. `CfsSubsetEval` forms subsets of features that are highly correlated with the class while having low intercorrelation are preferred. `GreedyStepwise` performs a greedy forward or backward search through the space of attribute subsets `setSearchBackwards(boolean back)` used to set whether to search backwards instead of forwards

Code:	`AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();`
	`CfsSubsetEval eval = new CfsSubsetEval();`
	`GreedyStepwise search = new GreedyStepwise();`
	`search.setSearchBackwards(true);`

Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/AttributeSelectedClassifier.html http://weka.sourceforge.net/doc.dev/weka/attributeSelection/CfsSubsetEval.htmlhttp://weka.sourceforge.net/doc.dev/weka/attributeSelection/GreedyStepwise.html

Snapshot of code to develop the Machine Learning Model from the dataset

```
AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
CfsSubsetEval eval = new CfsSubsetEval();
GreedyStepwise search = new GreedyStepwise();
search.setSearchBackwards(true);
```



Step V: Develop the machine learning model from the selected features dataset

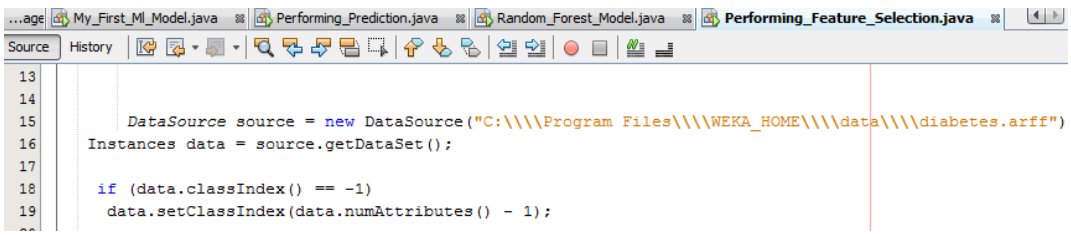
Definition:	`J48` class for developing a pruned or unpruned C4.5 decision tree `buildClassifier` method generates the classifier (Machine learning model) `j48` is the object of the class `J48` `setClassifier(Classifier newClassifier)` sets the base learner. `setEvaluator(ASEvaluation evaluator)` sets the attribute/subset evaluator `setSearch(ASSearch search)` sets the search method

Code:	`J48 j48 = new J48();`
	`classifier.setClassifier(j48);`
	`classifier.setEvaluator(eval);`
	`classifier.setSearch(search);`

Ref. URL:	http://weka.sourceforge.net/doc.dev/weka/classifiers/trees/J48.html http://weka.sourceforge.net/doc.dev/weka/classifiers/SingleClassifierEnhancer.html http://weka.sourceforge.net/doc.dev/weka/attributeSelection/AttributeSelection.html

Snapshot of code to develop the Machine Learning Model from the dataset

```
J48 j48 = new J48();
classifier.setClassifier(j48);
classifier.setEvaluator(eval);
classifier.setSearch(search);
```



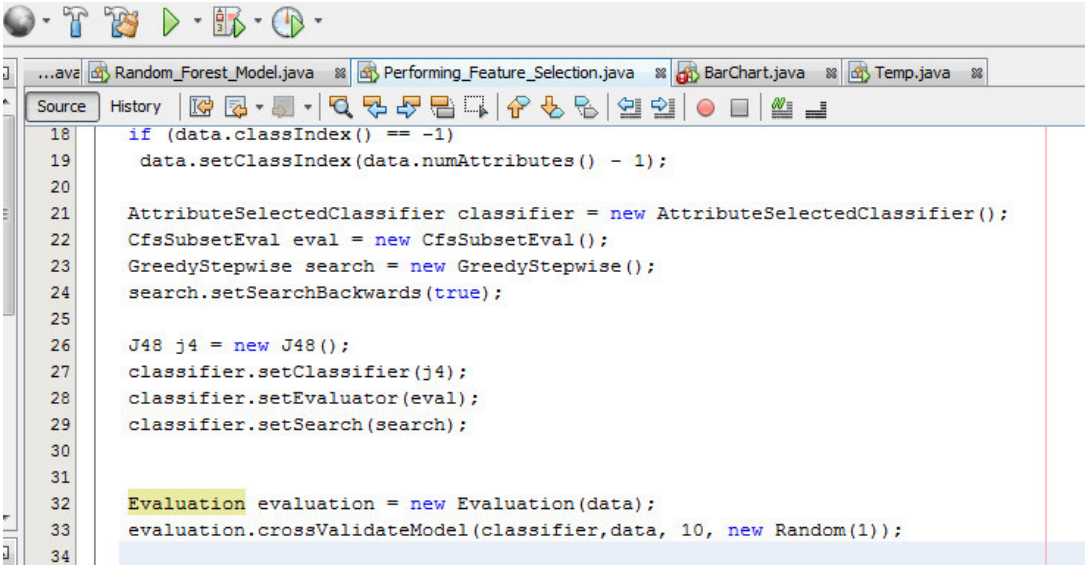
```
21    AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
22    CfsSubsetEval eval = new CfsSubsetEval();
23    GreedyStepwise search = new GreedyStepwise();
24    search.setSearchBackwards(true);
25
26     J48 j48 = new J48();
27    classifier.setClassifier(j48);
28    classifier.setEvaluator(eval);
29    classifier.setSearch(search);
```

Step VI: Evaluate the performance of machine learning model on the selected feature set

Definition	`Evaluation` class is used for evaluating machine learning models. `crossValidateModel` method Performs a (stratified if class is nominal) cross-validation for a classifier on a set of instances. `Random` class is used to generate a stream of pseudorandom numbers
Code:	`Evaluation evaluation = new Evaluation(data); evaluation.crossValidateModel(classifier,data, 10, new Random(1));`
Ref. URL	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/util/Random.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

```
Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier,data, 10, new Random(1));
```



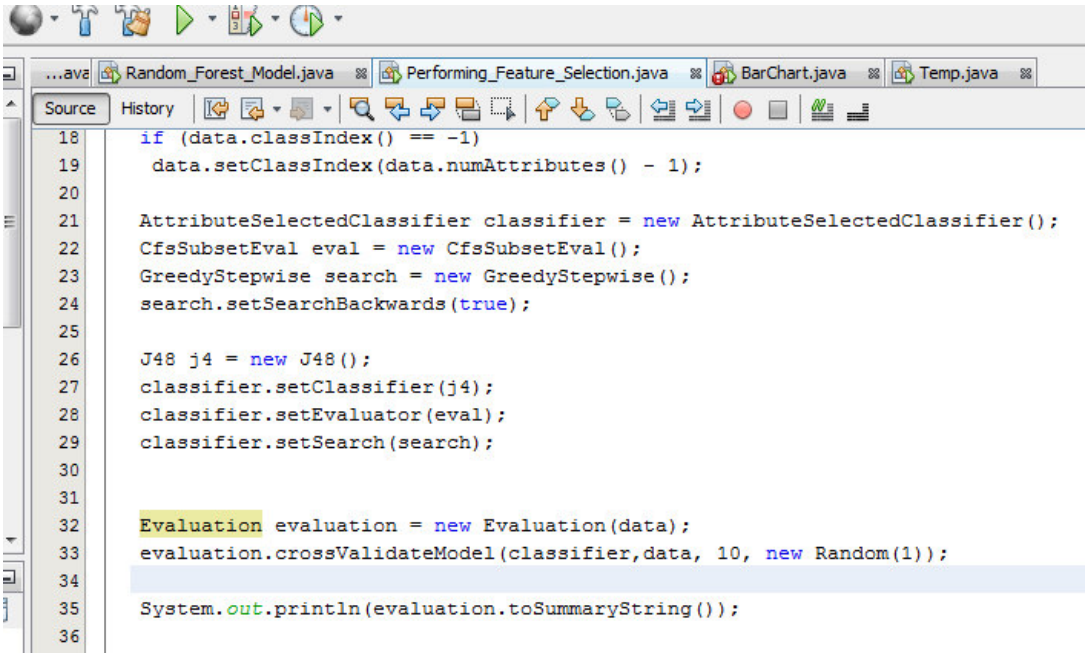
Step VI: Display the evaluated result

Definition:	`System.out.println(data)` is standard output stream that displays value of the data for output `toSummaryString` method outputs the performance statistics in summary form.
Code:	`System.out.println(evaluation.toSummaryString());`

Ref. URL:	http://weka.sourceforge.net/doc.stable/weka/classifiers/Evaluation.html https://docs.oracle.com/javase/8/docs/api/java/lang/System.html

Snapshot of code to evaluate the performance of Machine learning model and display the result in console window

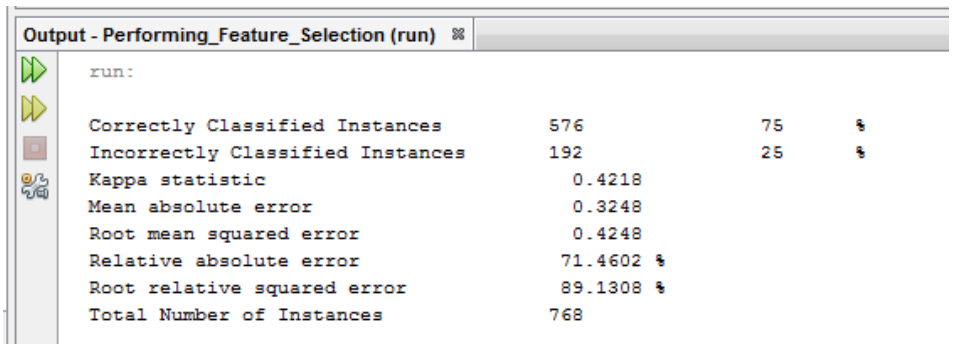
```
System.out.println(evaluation.toSummaryString());
```



Note: To complete the program use curly braces at the end of the program one for closing the main class and another one for closing the main method as shown in the below screenshot of complete code.

```
Evaluation evaluation = new Evaluation(data);
evaluation.crossValidateModel(classifier,data, 10, new Random(1));

System.out.println(evaluation.toSummaryString());


}
  }
```

Screenshot of output with feature selection:



Snapshot of complete code to perform the feature selection

```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {


    DataSource source = new DataSource("C:\\\\Program Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
 Instances data = source.getDataSet();

  if (data.classIndex() == -1)
   data.setClassIndex(data.numAttributes() - 1);

  AttributeSelectedClassifier classifier = new AttributeSelectedClassifier();
  CfsSubsetEval eval = new CfsSubsetEval();
  GreedyStepwise search = new GreedyStepwise();
  search.setSearchBackwards(true);

   J48 j48 = new J48();
  classifier.setClassifier(j48);
  classifier.setEvaluator(eval);
  classifier.setSearch(search);


  Evaluation evaluation = new Evaluation(data);
  evaluation.crossValidateModel(classifier, data, 10, new Random(1));

  System.out.println(evaluation.toSummaryString());

    }
}
```

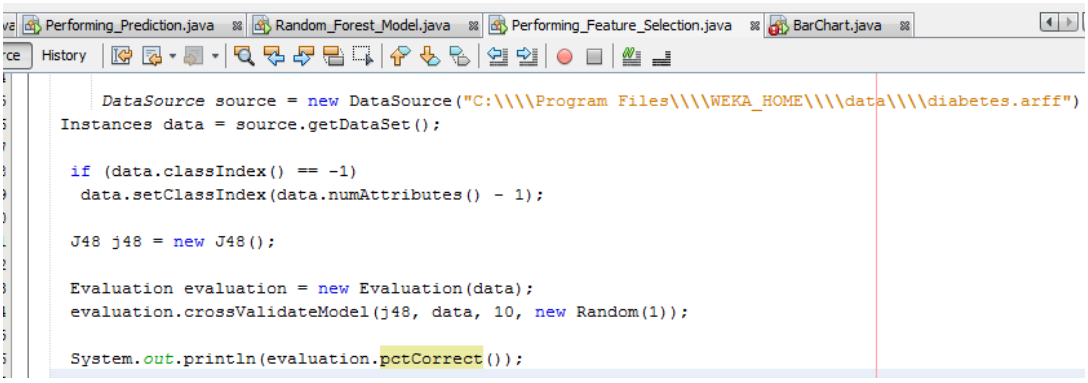
```
import weka.classifiers.trees.J48;
import weka.core.converters.ConverterUtils.DataSource;
import weka.classifiers.Evaluation;
import java.util.Random;
import weka.core.Instances;
import weka.attributeSelection.CfsSubsetEval;
import weka.attributeSelection.GreedyStepwise;
import weka.classifiers.meta.AttributeSelectedClassifier;

public class Performing_Feature_Selection {
public static void main(String[] args) throws Exception {

DataSource source = new DataSource("C:\\\\Program
Files\\\\WEKA_HOME\\\\data\\\\diabetes.arff");
 Instances data = source.getDataSet();

  if (data.classIndex() == -1)
   data.setClassIndex(data.numAttributes() - 1);

  AttributeSelectedClassifier classifier = new
AttributeSelectedClassifier();
  CfsSubsetEval eval = new CfsSubsetEval();
  GreedyStepwise search = new GreedyStepwise();
  search.setSearchBackwards(true);
```

```
    J48 j48 = new J48();
    classifier.setClassifier(j48);
    classifier.setEvaluator(eval);
    classifier.setSearch(search);

    Evaluation evaluation = new Evaluation(data);
    evaluation.crossValidateModel(classifier, data, 10, new
Random(1));

    System.out.println(evaluation.toSummaryString());

      }
}
```

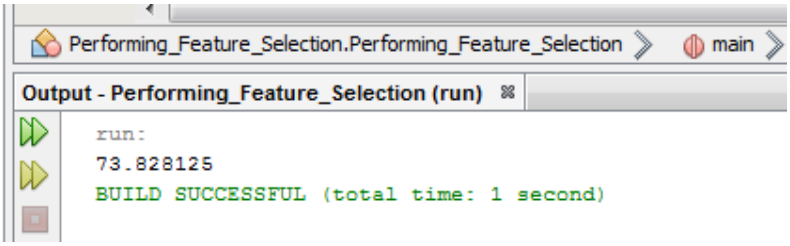
Part II
Additional Workouts

1. Develop a J48 model on diabetes dataset and compare the accuracy of the model with and without feature selection

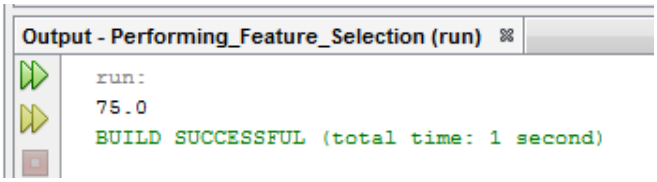
Screenshot of code without feature selection:



Screenshot of output without feature selection:



Screenshot of code with feature selection:



Comparison table for accuracy with and without feature selection

Accuracy in percentage without feature selection	Accuracy in percentage with feature selection
73.82	75.00

Gaining knowledge,
is the first step to wisdom.
Sharing it,
is the first step to humanity.