STROKE PREDICTION

FRANCESCO FAZIO	227758
DOMENICO SPAGNOLO	227805
YOSLENY LEAL ECHAVARRIA	224180

INTRODUCTION

According to World Health Organization (WHO), stroke are the second leading cause of death and the third leading cause of disability globally. Stroke is the sudden death of some brain cells due to lack of oxygen when the blood flow to the brain is lost by blockage or rupture of an artery to the brain.

Nearly 800,000 people in the United States suffer from a stroke each year, with about three in four being first-time strokes. 80% of the time these strokes can be prevented, so putting in place proper education on the signs of stroke is very important.

The objective of this study is to construct a prediction model for predicting stroke and to assess the accuracy of the model.

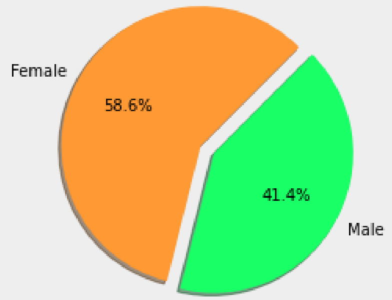
DATA UNDERSTANDING

A population of 5110 people are involved in this study. The dataset is extracted from Kaggle to predict whether a patient is likely to get stroke based on the following attribute information:

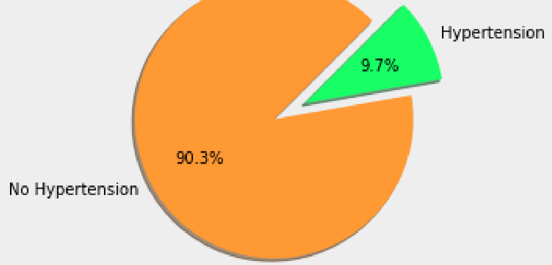
1. id	: unique identifier
2. gender	: "Male", "Female" or "Other"
3. age	: age of the patient
4. hypertension	: 0 if the patient doesn't have hypertension, 1 if the patient has hypertension
5. heart_disease	: 0 if the patient doesn't have any heart diseases, 1 if the patient has a heart disease
6. ever_married	: "No" or "Yes"
7. work_type	: "children", "Govt_jov", "Never_worked", "Private" or "Self-employed"
8. Residence_type	: "Rural" or "Urban"
9. avg_glucose_level	: average glucose level in blood
10. bmi	: body mass index
11. smoking_status	: "formerly smoked", "never smoked", "smokes" or "Unknown"
12. stroke	: 1 if the patient had a stroke, 0 the patient do not have a stroke

DATA UNDERSTANDING (2)

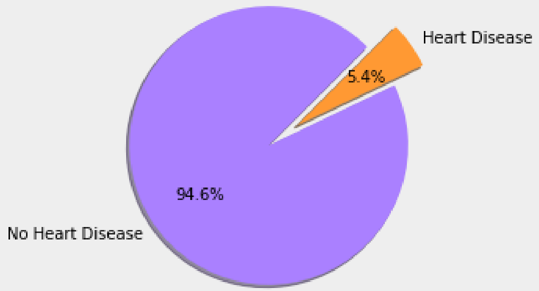
Gender



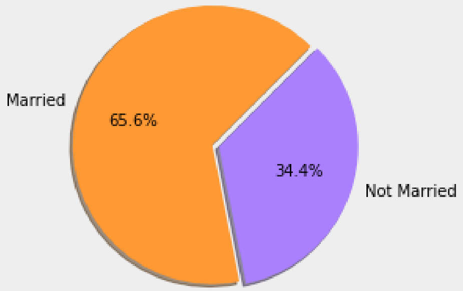
Hypertension



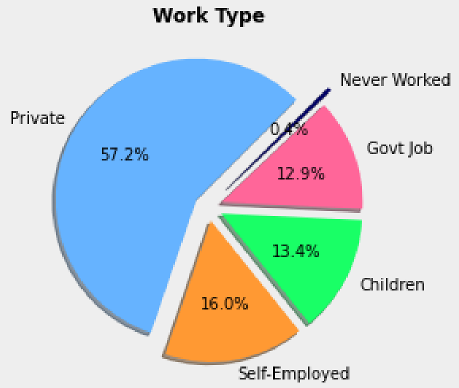
Heart disease

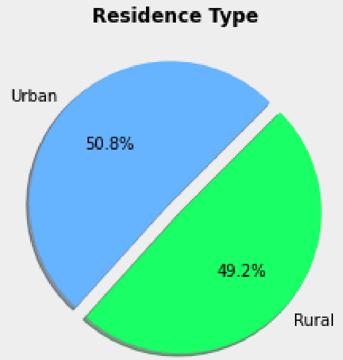


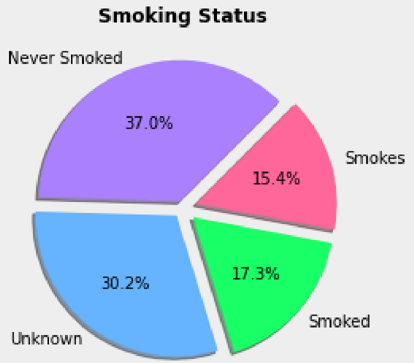
Marriage

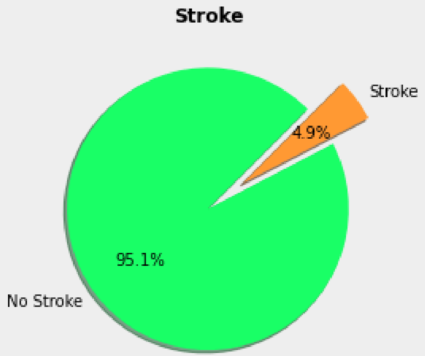


DATA UNDERSTANDING (3)



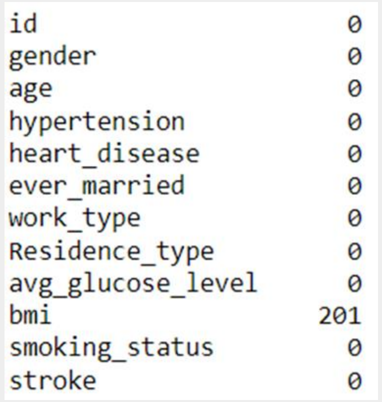




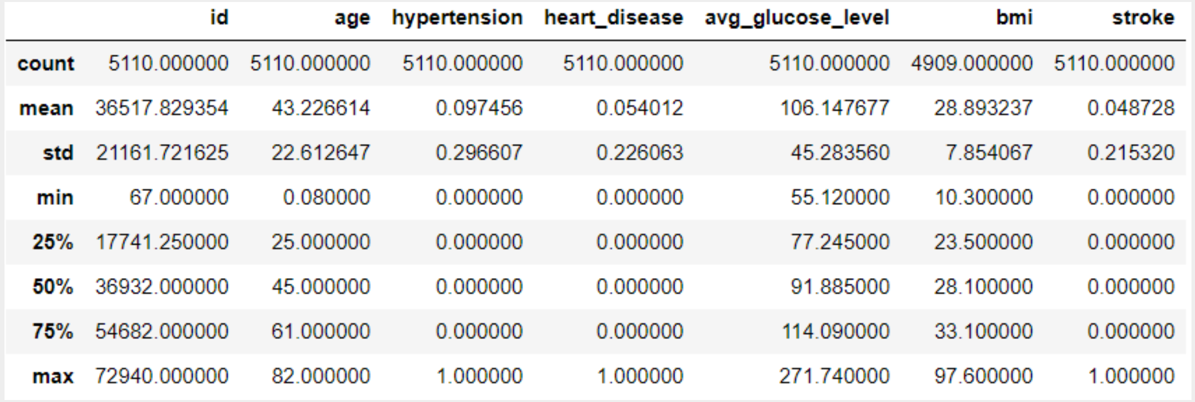


DATA CLEANING

There are 201 missing values in BMI feature. A simple way to dealing with the missing values is to remove the rows with null values however this may potentially remove data that aren’t null. Thus, we will substitute missing values with mean of BMI.

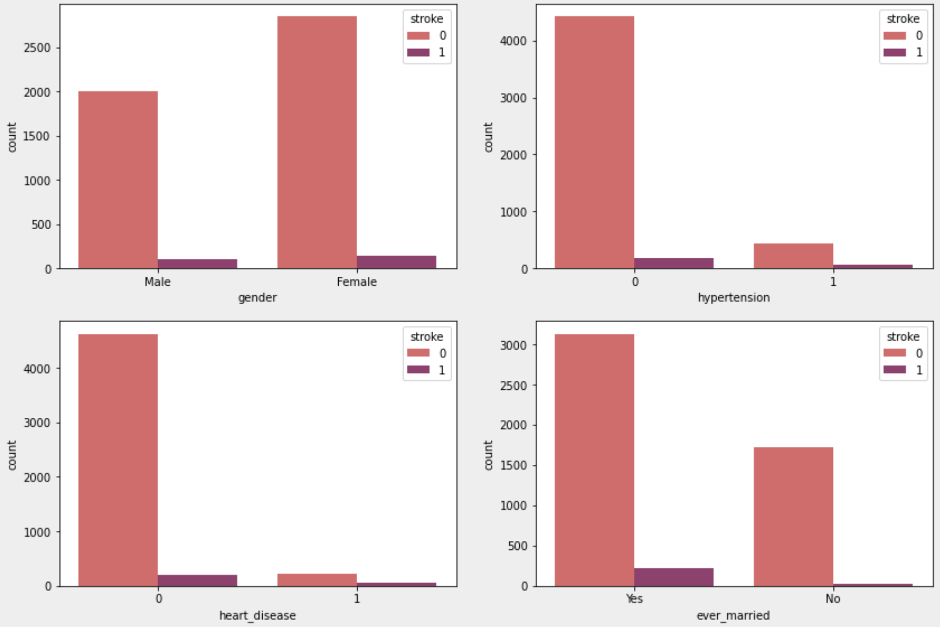


```
id                   0
gender               0
age                  0
hypertension         0
heart_disease        0
ever_married         0
work_type            0
Residence_type       0
avg_glucose_level    0
bmi                201
smoking_status       0
stroke               0
```



	id	age	hypertension	heart_disease	avg_glucose_level	bmi	stroke
count	5110.000000	5110.000000	5110.000000	5110.000000	5110.000000	4909.000000	5110.000000
mean	36517.829354	43.226614	0.097456	0.054012	106.147677	28.893237	0.048728
std	21161.721625	22.612647	0.296607	0.226063	45.283560	7.854067	0.215320
min	67.000000	0.080000	0.000000	0.000000	55.120000	10.300000	0.000000
25%	17741.250000	25.000000	0.000000	0.000000	77.245000	23.500000	0.000000
50%	36932.000000	45.000000	0.000000	0.000000	91.885000	28.100000	0.000000
75%	54682.000000	61.000000	0.000000	0.000000	114.090000	33.100000	0.000000
max	72940.000000	82.000000	1.000000	1.000000	271.740000	97.600000	1.000000

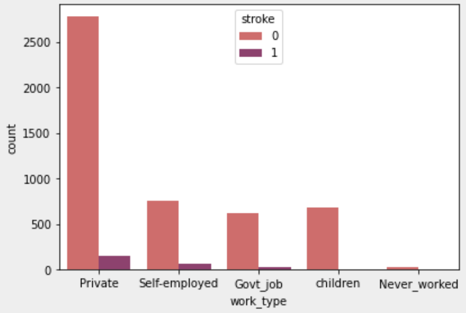
CATEGORICAL FEATURES ANALYSIS

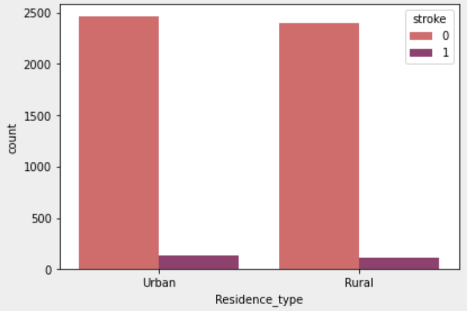


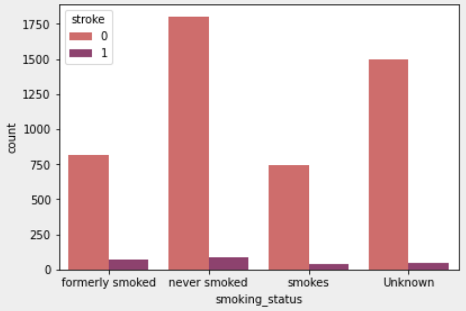
From the count plots, some observations can be drawn:

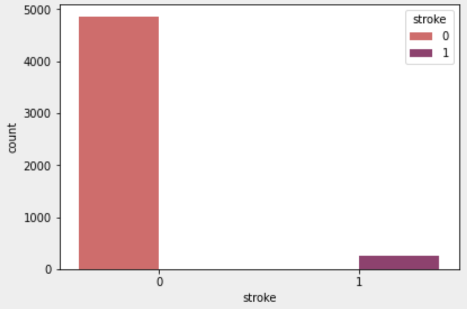
- **hypertension**: Subjects that previously diagnosed with hypertension have highly risk of having stroke.
- **heart_disease**: Subjects that previously diagnosed with heart disease have highly risk of having stroke.
- **ever_married**: Subjects that ever married have highly risk of having stroke.

CATEGORICAL FEATURES ANALYSIS (2)



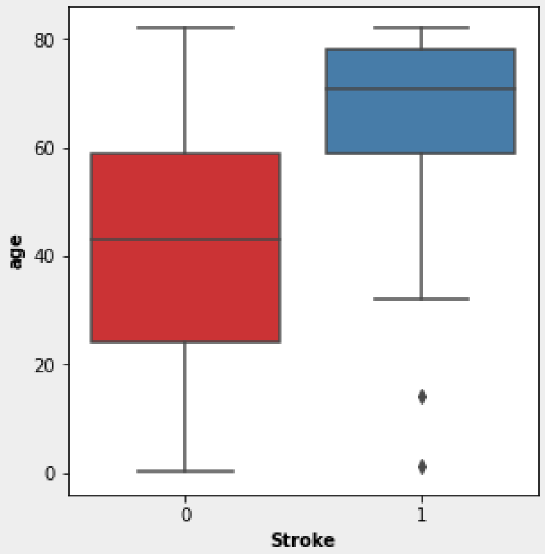


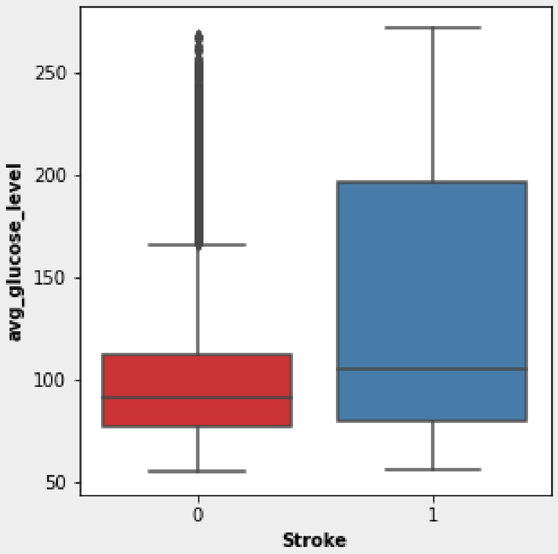


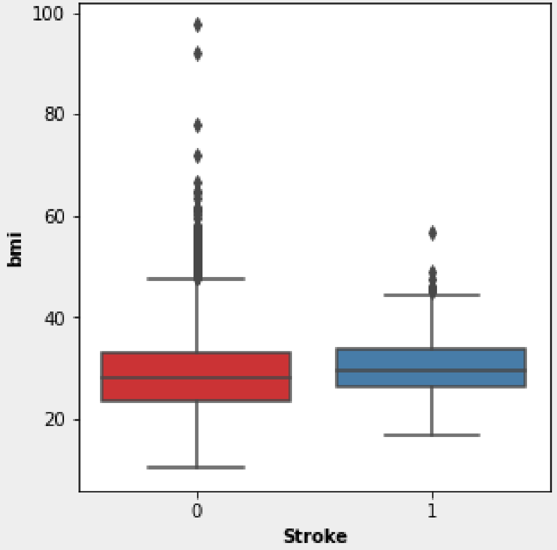


- **work_type**: Subjects that have any work experience and in government related work have highly risk of having stroke while those with no work experience barely experienced stroke.
- **Residence_type**: No obvious relationship with likelihood of experiencing stroke.
- **smoking_status**: Being a smoker or former smoker increases risk of having a stroke.

NUMERICAL FEATURES ANALYSIS







NUMERICAL FEATURES ANALYSIS (2)

From the boxplot, some observations can be drawn:

- **age**: Subjects with stroke tends to have higher mean age.
- **avg_glucose_level**: Subjects with stroke tends to have higher average glucose level.
- **bmi**: bmi index does not give much indication on the likelihood of experiencing stroke. bmi index for super obesity is 50. Outliers in this feature should be replaced to its highest limit (50). There are total 79 counts of outliers detected.

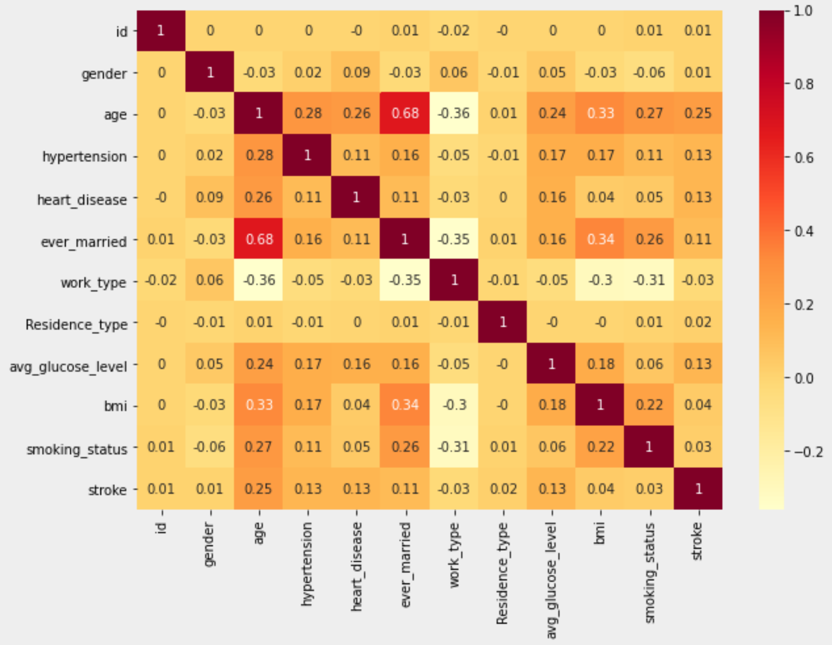
CORRELATION FEATURES ANALYSIS

Since correlation check only accept numerical variables, preprocessing the categorical variables becomes a necessary step, we need to convert these categorical variables to numbers encoded to 0 or 1.

1. gender
2. ever_married
3. work_type
4. Residence_type
5. smoking_status

CORRELATION FEATURES ANALYSIS (2)

From the correlation matrix, we can verify the presence of multicollinearity between some of the variables. For instance, the **ever_married** and **age** column has a correlation of 0.68. Between this two attributes, age contains more information on whether one is susceptible to stroke. Thus, we will drop the ever_married column.



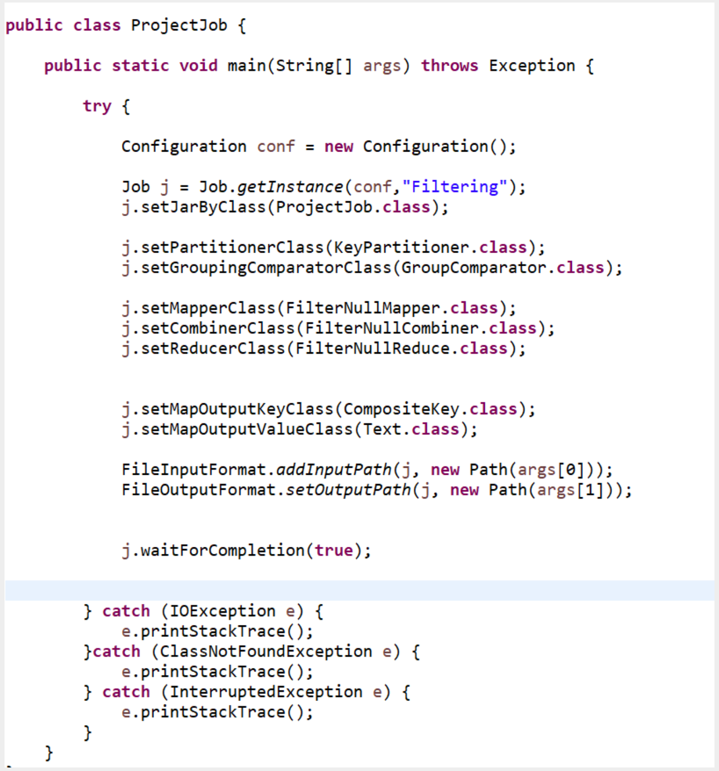
	id	gender	age	hypertension	heart_disease	ever_married	work_type	Residence_type	avg_glucose_level	bmi	smoking_status	stroke
id	1	0	0	0	-0	0.01	-0.02	-0	0	0	0.01	0.01
gender	0	1	-0.03	0.02	0.09	-0.03	0.06	-0.01	0.05	-0.03	-0.06	0.01
age	0	-0.03	1	0.28	0.26	0.68	-0.36	0.01	0.24	0.33	0.27	0.25
hypertension	0	0.02	0.28	1	0.11	0.16	-0.05	-0.01	0.17	0.17	0.11	0.13
heart_disease	-0	0.09	0.26	0.11	1	0.11	-0.03	0	0.16	0.04	0.05	0.13
ever_married	0.01	-0.03	0.68	0.16	0.11	1	-0.35	0.01	0.16	0.34	0.26	0.11
work_type	-0.02	0.06	-0.36	-0.05	-0.03	-0.35	1	-0.01	-0.05	-0.3	-0.31	-0.03
Residence_type	-0	-0.01	0.01	-0.01	0	0.01	-0.01	1	-0	-0	0.01	0.02
avg_glucose_level	0	0.05	0.24	0.17	0.16	0.16	-0.05	-0	1	0.18	0.06	0.13
bmi	0	-0.03	0.33	0.17	0.04	0.34	-0.3	-0	0.18	1	0.22	0.04
smoking_status	0.01	-0.06	0.27	0.11	0.05	0.26	-0.31	0.01	0.06	0.22	1	0.03
stroke	0.01	0.01	0.25	0.13	0.13	0.11	-0.03	0.02	0.13	0.04	0.03	1

STANDARDIZATION FEATURES

Variables that are measured at different scales do not contribute equally to model fitting and might end up creating a bias. Thus, to deal with this potential problem feature standardization is usually used prior to model fitting. We have applied standardization on **avg_glucose_level**, **bmi** and **age** columns.

MAP REDUCE: JOB CONF

Configuration of MapReduce job.



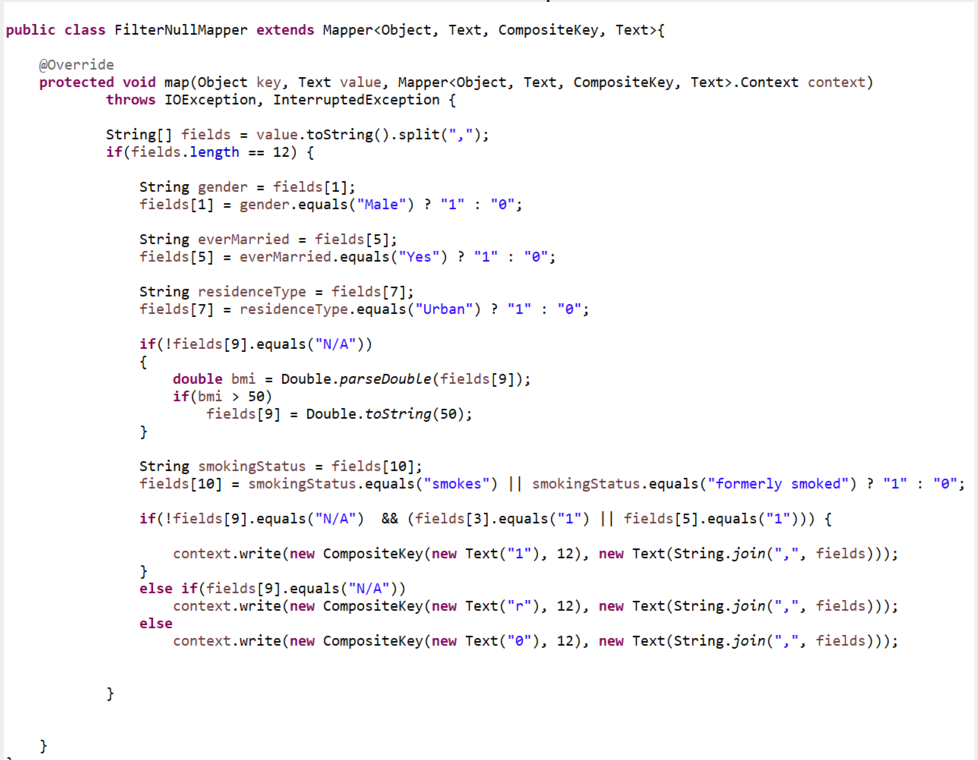
```
public class ProjectJob {

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

        try {

            Configuration conf = new Configuration();

            Job j = Job.getInstance(conf,"Filtering");
            j.setJarByClass(ProjectJob.class);

            j.setPartitionerClass(KeyPartitioner.class);
            j.setGroupingComparatorClass(GroupComparator.class);

            j.setMapperClass(FilterNullMapper.class);
            j.setCombinerClass(FilterNullCombiner.class);
            j.setReducerClass(FilterNullReduce.class);


            j.setMapOutputKeyClass(CompositeKey.class);
            j.setMapOutputValueClass(Text.class);

            FileInputFormat.addInputPath(j, new Path(args[0]));
            FileOutputFormat.setOutputPath(j, new Path(args[1]));


            j.waitForCompletion(true);


        } catch (IOException e) {
            e.printStackTrace();
        }catch (ClassNotFoundException e) {
            e.printStackTrace();
        } catch (InterruptedException e) {
            e.printStackTrace();
        }
    }
```

To execute the MapReduce job is required to send as first parameter the path for the csv file and the second parameter the path for the output

```
yarn jar map.jar BigData.ProjectJob /user/user/data.csv /user/user/project/map
```

MAP REDUCE: MAPPER

MapReduce (Mapper) was used also to transform categorical features into numerical features and to replace outliers in bmi column to its highest limit (50).

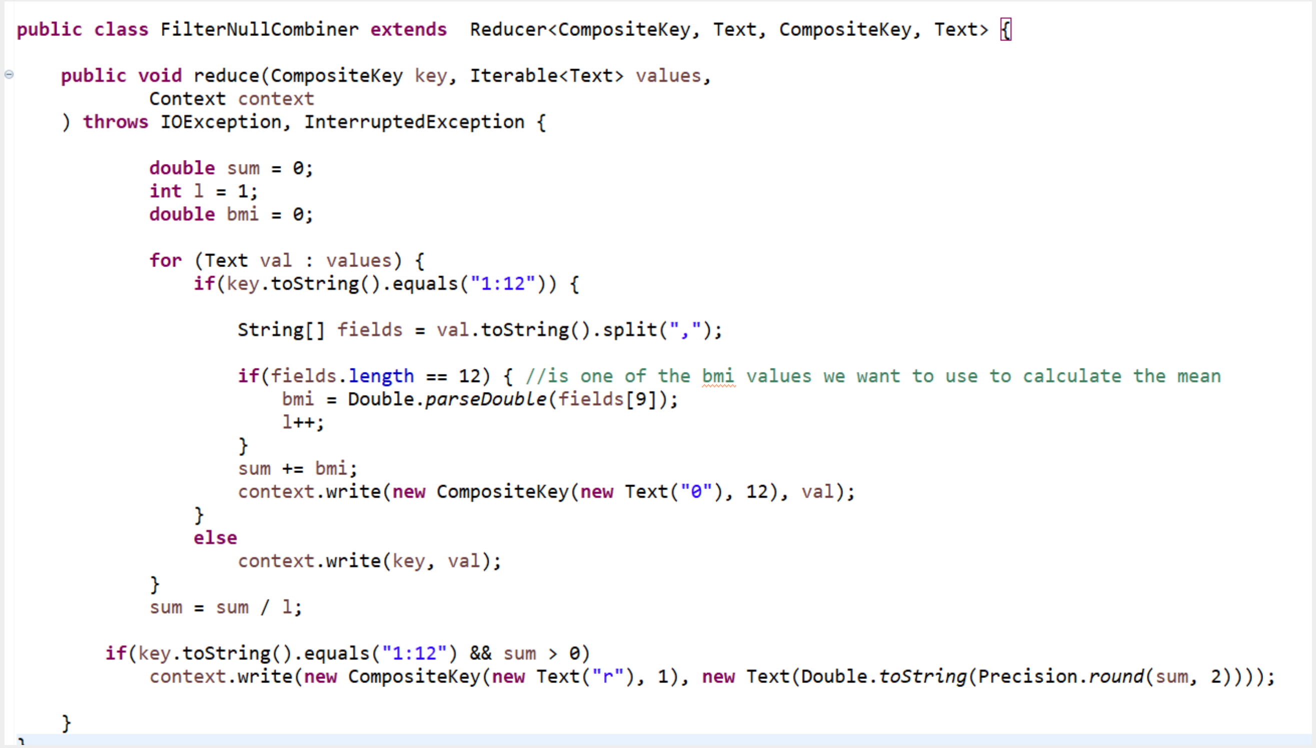


```java
public class FilterNullMapper extends Mapper<Object, Text, CompositeKey, Text>{

    @Override
    protected void map(Object key, Text value, Mapper<Object, Text, CompositeKey, Text>.Context context)
            throws IOException, InterruptedException {

            String[] fields = value.toString().split(",");
            if(fields.length == 12) {

                String gender = fields[1];
                fields[1] = gender.equals("Male") ? "1" : "0";

                String everMarried = fields[5];
                fields[5] = everMarried.equals("Yes") ? "1" : "0";

                String residenceType = fields[7];
                fields[7] = residenceType.equals("Urban") ? "1" : "0";

                if(!fields[9].equals("N/A"))
                {
                    double bmi = Double.parseDouble(fields[9]);
                    if(bmi > 50)
                        fields[9] = Double.toString(50);
                }

                String smokingStatus = fields[10];
                fields[10] = smokingStatus.equals("smokes") || smokingStatus.equals("formerly smoked") ? "1" : "0";

                if(!fields[9].equals("N/A")  && (fields[3].equals("1") || fields[5].equals("1"))) {

                    context.write(new CompositeKey(new Text("1"), 12), new Text(String.join(",", fields)));
                }
                else if(fields[9].equals("N/A"))
                    context.write(new CompositeKey(new Text("r"), 12), new Text(String.join(",", fields)));
                else
                    context.write(new CompositeKey(new Text("0"), 12), new Text(String.join(",", fields)));


            }


    }
```

We defined a composite-key to calculate the bmi mean to impute the missing values, using those bmi values present in the records where patients Suffer hypertension or were married considering how correlated they are with the column we want to predict.

MAP REDUCE: COMBINER

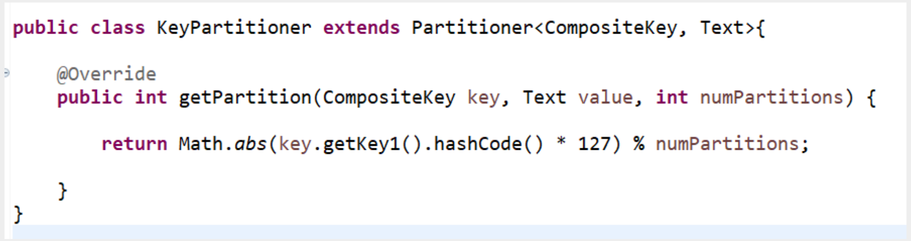
MapReduce (Combiner) was used to calculate the mean of the bmi column using the composite-key previously defined.



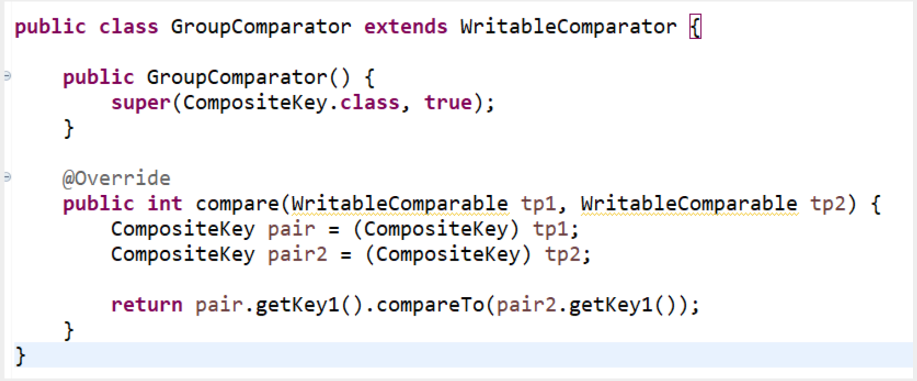
```
public class FilterNullCombiner extends  Reducer<CompositeKey, Text, CompositeKey, Text> {

    public void reduce(CompositeKey key, Iterable<Text> values,
            Context context
    ) throws IOException, InterruptedException {

            double sum = 0;
            int l = 1;
            double bmi = 0;

            for (Text val : values) {
                if(key.toString().equals("1:12")) {

                    String[] fields = val.toString().split(",");

                    if(fields.length == 12) { //is one of the bmi values we want to use to calculate the mean
                        bmi = Double.parseDouble(fields[9]);
                        l++;
                    }
                    sum += bmi;
                    context.write(new CompositeKey(new Text("0"), 12), val);
                }
                else
                    context.write(key, val);
            }
            sum = sum / l;

        if(key.toString().equals("1:12") && sum > 0)
            context.write(new CompositeKey(new Text("r"), 1), new Text(Double.toString(Precision.round(sum, 2))));

    }
```

MAP REDUCE: PARTITIONER, GROUPING

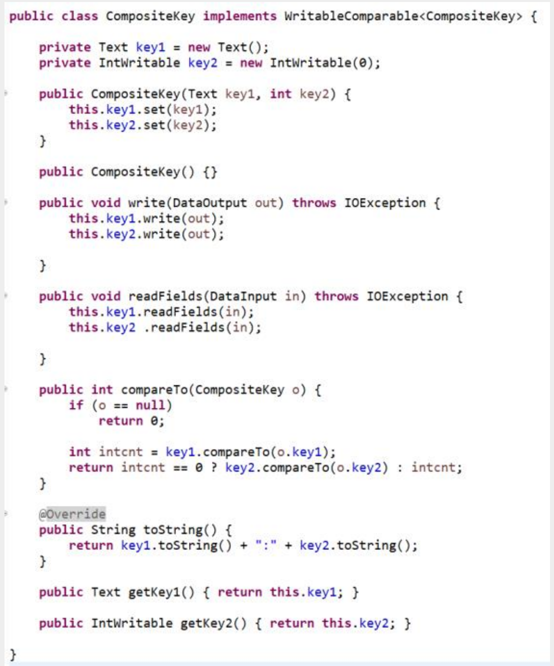
After the combiner is executed, there are only two keys: the one identifying rows that don’t need change(“0:12”) and those that need to impute the bmi column(“r:{#}”). The # can be **1** to indicate the row that contains the calculated mean for bmi and **12** for the rows where bmi column value is **N/A.** Using the partitioner, we will group by the natural key (“0” or “r”) and the group comparator will order those records by descending order on the secondary key.



```java
public class KeyPartitioner extends Partitioner<CompositeKey, Text>{

    @Override
    public int getPartition(CompositeKey key, Text value, int numPartitions) {

        return Math.abs(key.getKey1().hashCode() * 127) % numPartitions;

    }
}
```



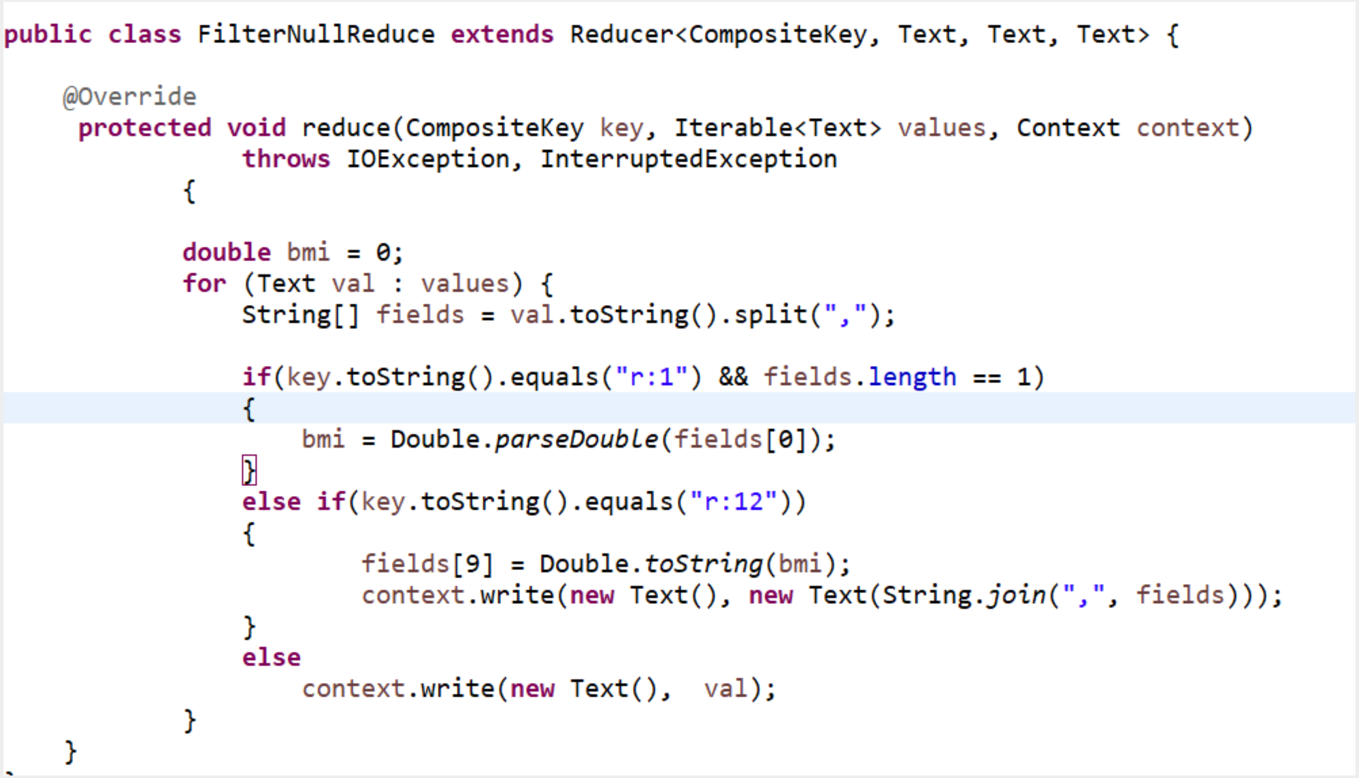
```java
public class GroupComparator extends WritableComparator {

    public GroupComparator() {
        super(CompositeKey.class, true);
    }

    @Override
    public int compare(WritableComparable tp1, WritableComparable tp2) {
        CompositeKey pair = (CompositeKey) tp1;
        CompositeKey pair2 = (CompositeKey) tp2;

        return pair.getKey1().compareTo(pair2.getKey1());
    }
}
```



```java
public class CompositeKey implements WritableComparable<CompositeKey> {

    private Text key1 = new Text();
    private IntWritable key2 = new IntWritable(0);

    public CompositeKey(Text key1, int key2) {
        this.key1.set(key1);
        this.key2.set(key2);
    }

    public CompositeKey() {}

    public void write(DataOutput out) throws IOException {
        this.key1.write(out);
        this.key2.write(out);

    }

    public void readFields(DataInput in) throws IOException {
        this.key1.readFields(in);
        this.key2 .readFields(in);

    }

    public int compareTo(CompositeKey o) {
        if (o == null)
            return 0;

        int intcnt = key1.compareTo(o.key1);
        return intcnt == 0 ? key2.compareTo(o.key2) : intcnt;
    }

    @Override
    public String toString() {
        return key1.toString() + ":" + key2.toString();
    }

    public Text getKey1() { return this.key1; }

    public IntWritable getKey2() { return this.key2; }

}
```

MAP REDUCE: REDUCE

MapReduce (Reducer) we impute the missing values for bmi column



```java
public class FilterNullReduce extends Reducer<CompositeKey, Text, Text, Text> {

    @Override
     protected void reduce(CompositeKey key, Iterable<Text> values, Context context)
                throws IOException, InterruptedException
            {

            double bmi = 0;
            for (Text val : values) {
                String[] fields = val.toString().split(",");

                if(key.toString().equals("r:1") && fields.length == 1)
                {
                    bmi = Double.parseDouble(fields[0]);
                }
                else if(key.toString().equals("r:12"))
                {
                        fields[9] = Double.toString(bmi);
                        context.write(new Text(), new Text(String.join(",", fields)));
                }
                else
                    context.write(new Text(),  val);
            }
    }
```

MySQL DATABASE

In order to import data into MySQL the csv file must be put into the folder: /var/lib/mysql-files

```
sudo cp stroke_cln.csv /var/lib/mysql-files
sudo chmod ugo=rwx /var/lib/mysql-files/stroke_cln.csv
```

```
CREATE TABLE stroke_data(id int primary key, gender int, age int,
hypertension int, heart_disease int, ever_married int, work_type
varchar(255), Residence_type int, avg_glucose_level double, bmi double,
smoking_status int, stroke int);
```

```
LOAD DATA INFILE '/var/lib/mysql-files/stroke_cln.csv' INTO TABLE
stroke_data FIELDS TERMINATED BY ','
```

SQOOP IMPORT

We imported table ‘stroke_data’ from MySQL database called ‘stroke’ into hive:

```
sqoop-import
        --connect jdbc:mysql://master/stroke
        --username hive –P
        --warehouse-dir stroke
        --table stroke_data
        --hive-import
        -m 1
```

SPARK

- Setup SparkSession:

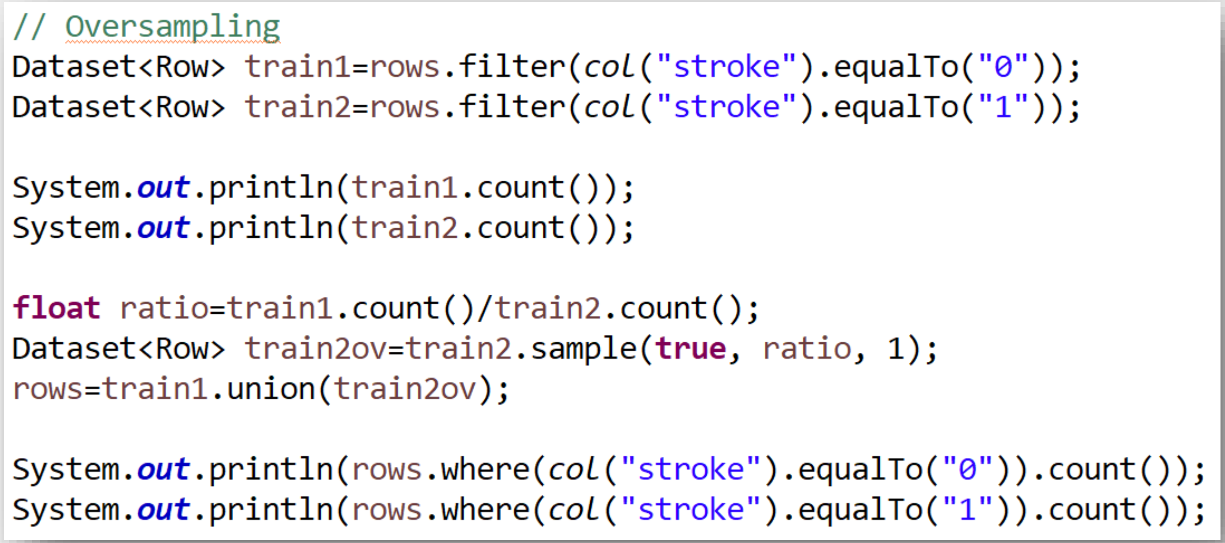
```
SparkSession spark = SparkSession.builder().appName("StrokePrediction").enableHiveSupport().getOrCreate();
spark.sparkContext().setLogLevel("WARN");
```

- Read data from Hive:

```
Dataset<Row> rows = spark.sql("SELECT gender,age,hypertension,heart_disease,work_type,Residence_type,avg_glucose_level,bmi,smoking_status,stroke from stroke_data where age>0");
```

SPARK: PRE-PROCESSING

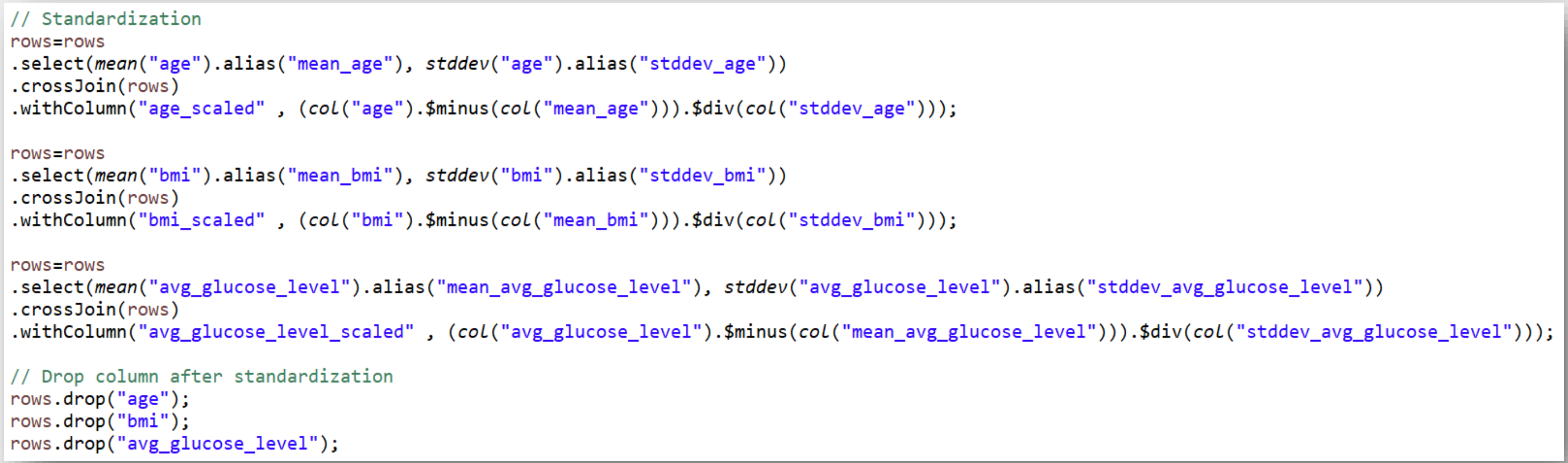
1. Oversampling of imbalanced dataset



```
// Oversampling
Dataset<Row> train1=rows.filter(col("stroke").equalTo("0"));
Dataset<Row> train2=rows.filter(col("stroke").equalTo("1"));

System.out.println(train1.count());
System.out.println(train2.count());

float ratio=train1.count()/train2.count();
Dataset<Row> train2ov=train2.sample(true, ratio, 1);
rows=train1.union(train2ov);

System.out.println(rows.where(col("stroke").equalTo("0")).count());
System.out.println(rows.where(col("stroke").equalTo("1")).count());
```

SPARK: PRE-PROCESSING (2)

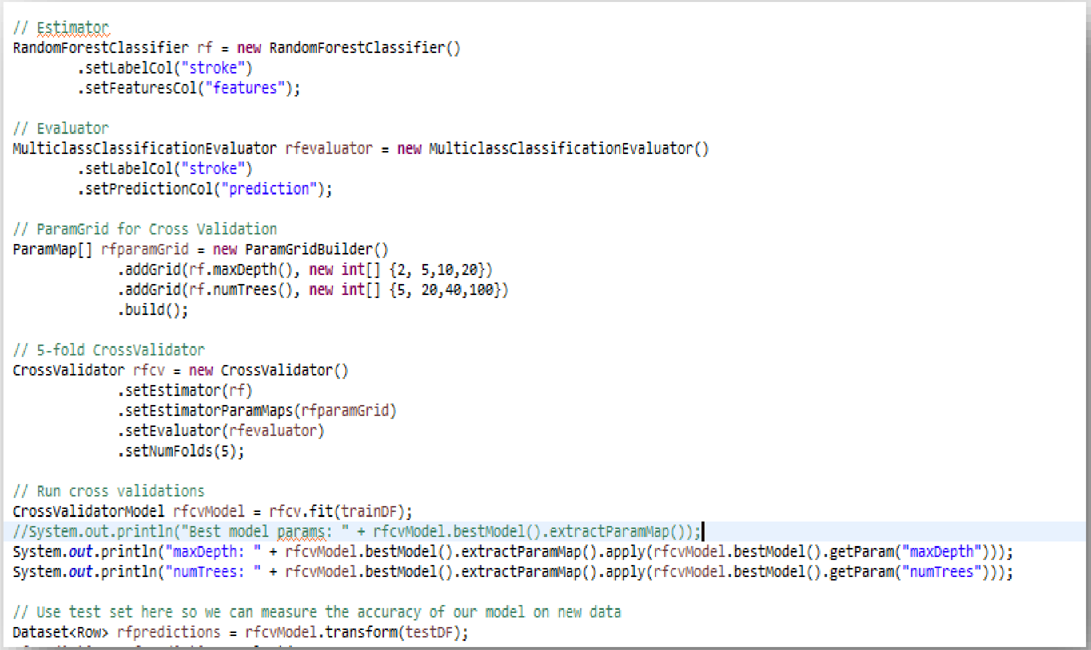
2. Standardization of some variables



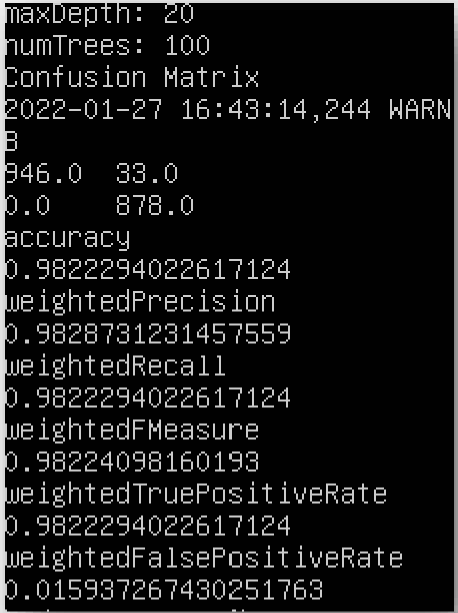
```
// Standardization
rows=rows
.select(mean("age").alias("mean_age"), stddev("age").alias("stddev_age"))
.crossJoin(rows)
.withColumn("age_scaled" , (col("age").$minus(col("mean_age"))).$div(col("stddev_age")));

rows=rows
.select(mean("bmi").alias("mean_bmi"), stddev("bmi").alias("stddev_bmi"))
.crossJoin(rows)
.withColumn("bmi_scaled" , (col("bmi").$minus(col("mean_bmi"))).$div(col("stddev_bmi")));

rows=rows
.select(mean("avg_glucose_level").alias("mean_avg_glucose_level"), stddev("avg_glucose_level").alias("stddev_avg_glucose_level"))
.crossJoin(rows)
.withColumn("avg_glucose_level_scaled" , (col("avg_glucose_level").$minus(col("mean_avg_glucose_level"))).$div(col("stddev_avg_glucose_level")));

// Drop column after standardization
rows.drop("age");
rows.drop("bmi");
rows.drop("avg_glucose_level");
```

SPARK: RANDOM FOREST

Random Forest: Model and evaluations.



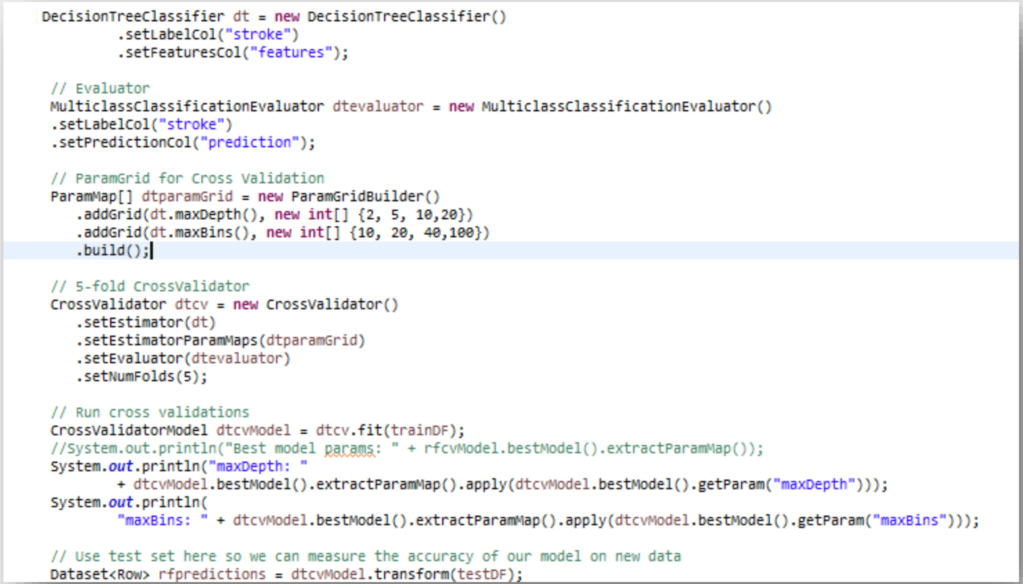
```java
// Estimator
RandomForestClassifier rf = new RandomForestClassifier()
        .setLabelCol("stroke")
        .setFeaturesCol("features");

// Evaluator
MulticlassClassificationEvaluator rfevaluator = new MulticlassClassificationEvaluator()
        .setLabelCol("stroke")
        .setPredictionCol("prediction");

// ParamGrid for Cross Validation
ParamMap[] rfparamGrid = new ParamGridBuilder()
             .addGrid(rf.maxDepth(), new int[] {2, 5,10,20})
             .addGrid(rf.numTrees(), new int[] {5, 20,40,100})
             .build();

// 5-fold CrossValidator
CrossValidator rfcv = new CrossValidator()
             .setEstimator(rf)
             .setEstimatorParamMaps(rfparamGrid)
             .setEvaluator(rfevaluator)
             .setNumFolds(5);

// Run cross validations
CrossValidatorModel rfcvModel = rfcv.fit(trainDF);
//System.out.println("Best model params: " + rfcvModel.bestModel().extractParamMap());
System.out.println("maxDepth: " + rfcvModel.bestModel().extractParamMap().apply(rfcvModel.bestModel().getParam("maxDepth")));
System.out.println("numTrees: " + rfcvModel.bestModel().extractParamMap().apply(rfcvModel.bestModel().getParam("numTrees")));

// Use test set here so we can measure the accuracy of our model on new data
Dataset<Row> rfpredictions = rfcvModel.transform(testDF);
```



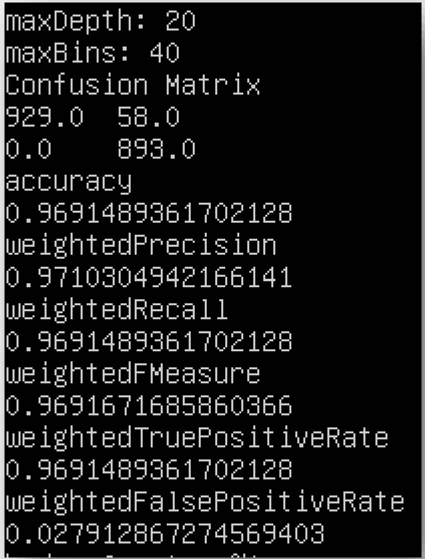
```
maxDepth: 20
numTrees: 100
Confusion Matrix
2022-01-27 16:43:14,244 WARN
B
946.0  33.0
0.0    878.0
accuracy
0.982229402617124
weightedPrecision
0.9828731231457559
weightedRecall
0.982229402617124
weightedFMeasure
0.98224098160193
weightedTruePositiveRate
0.982229402617124
weightedFalsePositiveRate
0.015937267430251763
```

SPARK: DECISION TREE

Decision Tree: Model and evaluations.



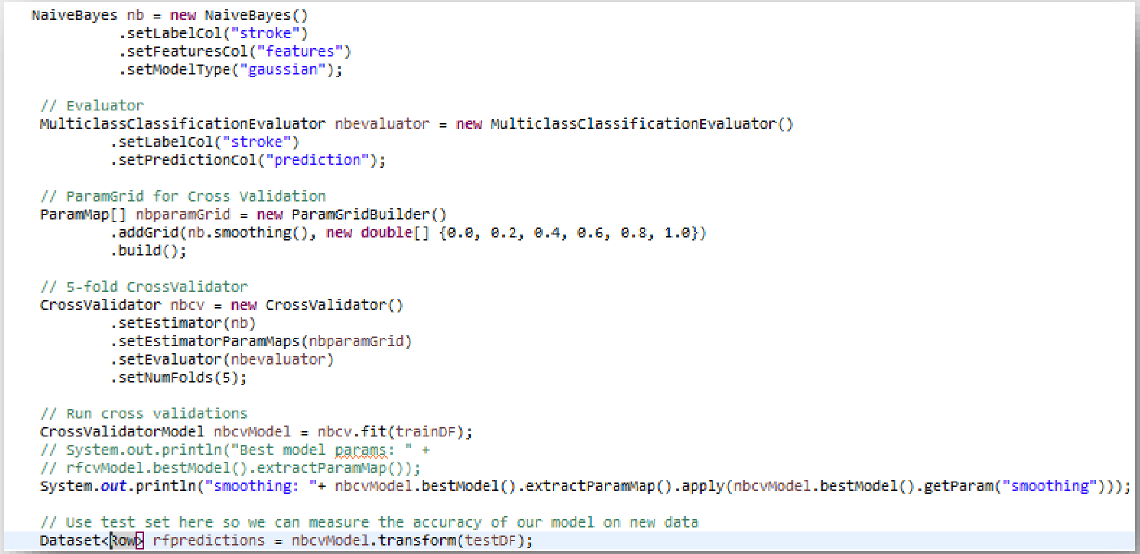
```java
    DecisionTreeClassifier dt = new DecisionTreeClassifier()
            .setLabelCol("stroke")
            .setFeaturesCol("features");

     // Evaluator
     MulticlassClassificationEvaluator dtevaluator = new MulticlassClassificationEvaluator()
     .setLabelCol("stroke")
     .setPredictionCol("prediction");

     // ParamGrid for Cross Validation
     ParamMap[] dtparamGrid = new ParamGridBuilder()
        .addGrid(dt.maxDepth(), new int[] {2, 5, 10,20})
        .addGrid(dt.maxBins(), new int[] {10, 20, 40,100})
        .build();

     // 5-fold CrossValidator
     CrossValidator dtcv = new CrossValidator()
        .setEstimator(dt)
        .setEstimatorParamMaps(dtparamGrid)
        .setEvaluator(dtevaluator)
        .setNumFolds(5);

     // Run cross validations
     CrossValidatorModel dtcvModel = dtcv.fit(trainDF);
     //System.out.println("Best model params: " + rfcvModel.bestModel().extractParamMap());
     System.out.println("maxDepth: "
            + dtcvModel.bestModel().extractParamMap().apply(dtcvModel.bestModel().getParam("maxDepth")));
     System.out.println(
            "maxBins: " + dtcvModel.bestModel().extractParamMap().apply(dtcvModel.bestModel().getParam("maxBins")));

     // Use test set here so we can measure the accuracy of our model on new data
     Dataset<Row> rfpredictions = dtcvModel.transform(testDF);
```



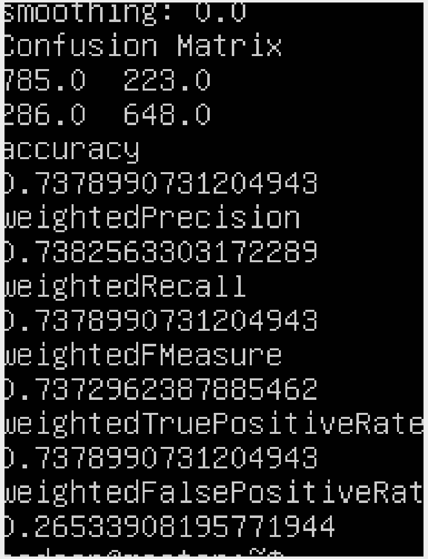
```
maxDepth: 20
maxBins: 40
Confusion Matrix
929.0  58.0
0.0    893.0
accuracy
0.9691489361702128
weightedPrecision
0.9710304942166141
weightedRecall
0.9691489361702128
weightedFMeasure
0.969167168586O366
weightedTruePositiveRate
0.9691489361702128
weightedFalsePositiveRate
0.02791286727456940З
```

SPARK: Naïve Bayes

Naïve Bayes: Model and evaluations.



```java
NaiveBayes nb = new NaiveBayes()
          .setLabelCol("stroke")
          .setFeaturesCol("features")
          .setModelType("gaussian");

 // Evaluator
 MulticlassClassificationEvaluator nbevaluator = new MulticlassClassificationEvaluator()
         .setLabelCol("stroke")
         .setPredictionCol("prediction");

 // ParamGrid for Cross Validation
 ParamMap[] nbparamGrid = new ParamGridBuilder()
         .addGrid(nb.smoothing(), new double[] {0.0, 0.2, 0.4, 0.6, 0.8, 1.0})
         .build();

 // 5-fold CrossValidator
 CrossValidator nbcv = new CrossValidator()
         .setEstimator(nb)
         .setEstimatorParamMaps(nbparamGrid)
         .setEvaluator(nbevaluator)
         .setNumFolds(5);

 // Run cross validations
 CrossValidatorModel nbcvModel = nbcv.fit(trainDF);
 // System.out.println("Best model params: " +
 // rfcvModel.bestModel().extractParamMap());
 System.out.println("smoothing: "+ nbcvModel.bestModel().extractParamMap().apply(nbcvModel.bestModel().getParam("smoothing")));

 // Use test set here so we can measure the accuracy of our model on new data
 Dataset<Row> rfpredictions = nbcvModel.transform(testDF);
```



```
smoothing: 0.0
Confusion Matrix
785.0  223.0
286.0  648.0
accuracy
0.7378990731204943
weightedPrecision
0.7382563303172289
weightedRecall
0.7378990731204943
weightedFMeasure
0.7372962387885462
weightedTruePositiveRate
0.7378990731204943
weightedFalsePositiveRat
0.2653390819577194
```

Best Results

After comparing the various metrics, we concluded that the Random Forest is the model with the highest performance compared to the other two.

The best model parameters are:

- maxDepth = 20
- numTrees = 100

THANKS FOR THE ATTENTION!