Vellore Institute of Technology
(Deemed to be University under section 3 of UGC Act, 1956)

School of Computer Science and Engineering

J Component report

Programme : **MTech Integrated Software Engineering**

Course Title : **BIG DATA ANALYTICS**

Course Code : **SWE2011**

Slot : **E2+TE2**

Title: Heart Stroke Prediction

Team Members: **Kshitiz Goyal | 19MIS1009**

Naman Jain | 19MIS1040

Srijan Kumar Airon | 19MIS1119

Faculty: Syed Ibrahim S P

Sign:

Date: 25-04-2022

AIM:

To predict the chances of a person getting a heart stroke on the basis of given parameters by implementing data mining algorithms in traditional and big data framework.

PROBLEM STATEMENT:

Heart disease can be effectively managed with a combination of lifestyle changes, medication, and surgery in some circumstances. The symptoms of heart disease can be lessened and the heart's function enhanced with the correct treatment. The projected outcomes can be utilized to prevent and thereby minimize the cost of surgery and other costly treatments. The overarching goal of my research will be to reliably predict the occurrence of heart stroke using only a few tests and features.

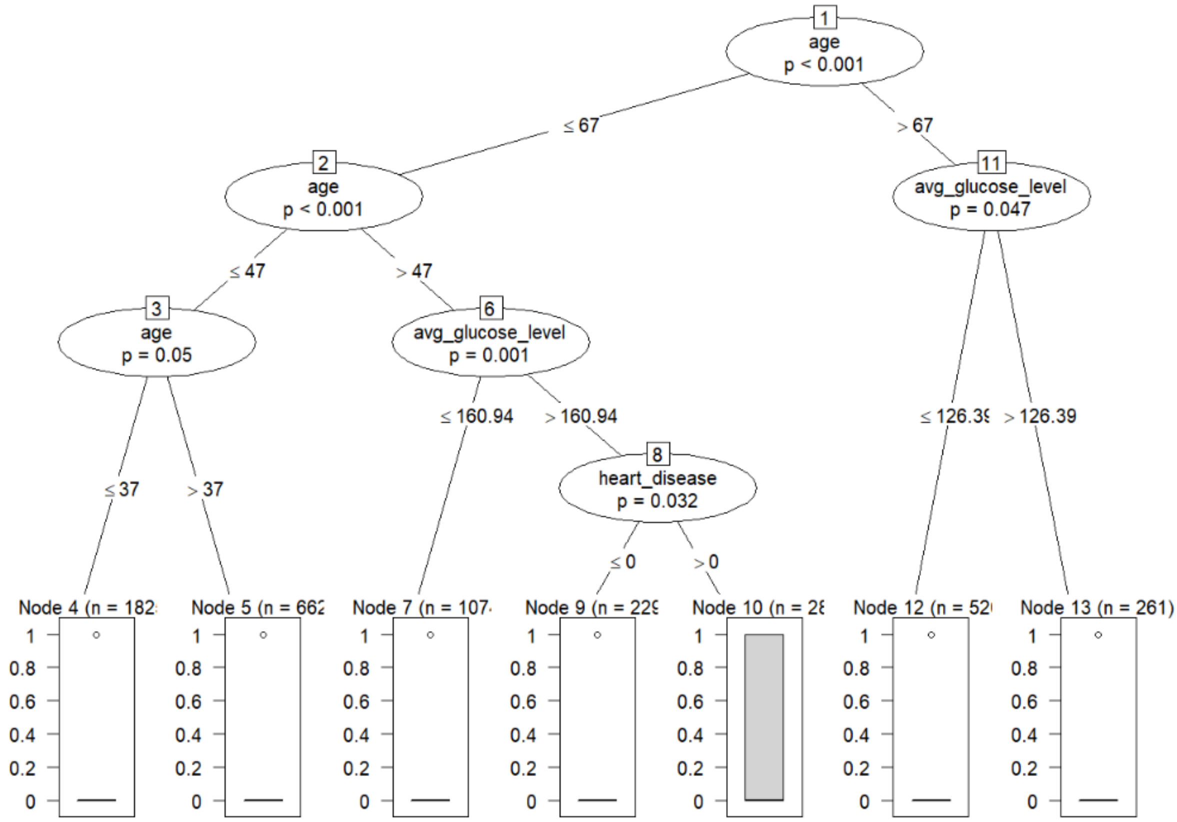
The attributes that are taken into account are the primary foundation for testing and, for the most part, provide accurate findings. Many more input attributes can be used, but our goal is to forecast the risk of heart disease with fewer and more efficient features. Rather than the knowledge-rich data hidden in the data set and databases, decisions are frequently made purely on doctors' intuition and expertise. This approach results in unintended biases, errors, and exorbitant medical costs, all of which have an impact on the quality of care offered to patients. The healthcare business may benefit greatly from data mining since it allows health organizations to systematically use data and analytics to find inefficiencies and practice guidelines that improve treatment and lower costs.

IMPLEMENTATION IN THE TRADITIONAL SYSTEM:

1. C4.5 ALGORITHM

```r
library(party)
data <- read.csv(file.choose(),header=T)
start_time <- Sys.time()
str(data)
set.seed (1234)
dt = sort(sample(nrow(data), nrow(data)*.9))
train<-data[dt,]
validate<-data[-dt,]
tree<-ctree(stroke ~.,data = train)
plot(tree)
predict(tree,validate,type="prob")
pred <- predict(tree,validate)
pred1 <- as.integer(pred)
count=0

for (x in 1:length(validate$stroke)) {
  if(pred1[x]==validate$stroke[x] )
  {
    count=count+1
  }
}
accuracy = count/length(validate$stroke)
accuracy
end_time <- Sys.time()
end_time-start_time
```

```
> pred <- predict(tree,validate)
> pred1 <- as.integer(pred)
> count=0
>
> for (x in 1:length(validate$stroke)) {
+   if(pred1[x]==validate$stroke[x] )
+   {
+     count=count+1
+   }
+ }
> accuracy = count/length(validate$stroke)
> accuracy
[1] 0
> end_time <- Sys.time()
> end_time-start_time
Time difference of 0.542336 secs
>
```

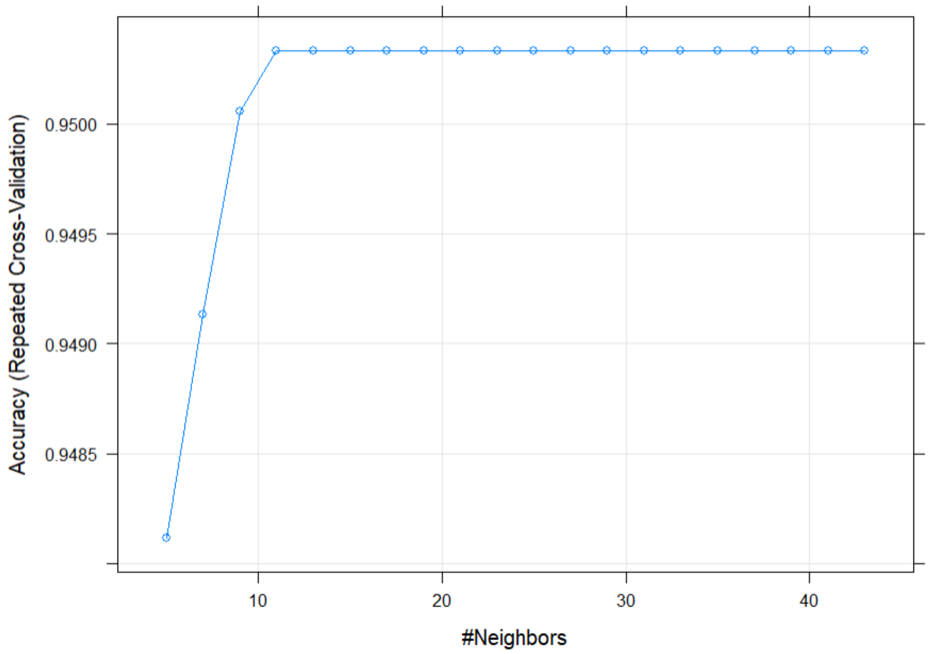


2. K-NEAREST NEIGHBOURS ALGORITHM

```
> library(mlbench)
> data <- read.csv(file.choose(),header=T)
> start_time <- Sys.time()
> str(data)
'data.frame':	5110 obs. of  11 variables:
 $ id               : int  9046 51676 31112 60182 1665 56669 53882 10434 27419 60491 ...
 $ gender           : int  1 0 1 0 0 1 1 0 0 0 ...
 $ age              : num  67 61 80 49 79 81 74 69 59 78 ...
 $ hypertension     : int  0 0 0 0 1 0 1 0 0 0 ...
 $ heart_disease    : int  1 0 1 0 0 0 1 0 0 0 ...
 $ ever_married     : int  1 1 1 1 1 1 1 0 1 1 ...
 $ work_type        : int  1 2 1 1 2 1 1 1 1 1 ...
 $ Residence_type   : int  1 2 2 1 2 1 2 1 2 1 ...
 $ avg_glucose_level: num  229 202 106 171 174 ...
 $ smoking_status   : int  1 0 0 2 0 1 0 0 3 3 ...
 $ stroke           : chr  "Yes" "Yes" "Yes" "Yes" ...
> data$stroke[data$stroke == 0] <- 'No'
> data$stroke[data$stroke == 1] <- 'Yes'
> data$stroke<-factor(data$stroke)
> set.seed(1234)
> ind <- sample(2, nrow(data), replace=T, prob=c(0.7,0.3))
> training <- data[ind==1,]
> test <- data[ind==2,]
>
> #KNN Model
> trControl <- trainControl(method="repeatedcv",
+                           number=10,
+                           repeats=3)
> set.seed(222)
> fit <- train(stroke ~.,
+              data=training,
+              method="knn",
+              tuneLength=20,
+              trControl=trControl,
+              preProc=c("center","scale"))
> fit
k-Nearest Neighbors

3604 samples
  10 predictor
   2 classes: 'No', 'Yes'
```

```
  k   Accuracy   Kappa
   5  0.9481150   0.0080622833
   7  0.9491320  -0.0022727919
   9  0.9500569  -0.0005276271
  11  0.9503339   0.0000000000
  13  0.9503339   0.0000000000
  15  0.9503339   0.0000000000
  17  0.9503339   0.0000000000
  19  0.9503339   0.0000000000
  21  0.9503339   0.0000000000
  23  0.9503339   0.0000000000
  25  0.9503339   0.0000000000
  27  0.9503339   0.0000000000
  29  0.9503339   0.0000000000
  31  0.9503339   0.0000000000
  33  0.9503339   0.0000000000
  35  0.9503339   0.0000000000
  37  0.9503339   0.0000000000
  39  0.9503339   0.0000000000
  41  0.9503339   0.0000000000
  43  0.9503339   0.0000000000

Accuracy was used to select the optimal model using the largest value
The final value used for the model was k = 43.
> plot(fit)
> varImp(fit)
ROC curve variable importance

                  Importance
age                 100.0000
ever_married         34.0686
avg_glucose_level    25.4637
hypertension         22.4141
heart_disease        18.6933
smoking_status       15.5226
Residence_type        8.9154
work_type             8.7970
id                    0.2145
gender                0.0000
> pred <- predict(fit,newdata=test)
> confusionMatrix(pred,test$stroke)
Confusion Matrix and Statistics
```

```
> confusionMatrix(pred,test$stroke)
Confusion Matrix and Statistics

          Reference
Prediction   No  Yes
       No  1436   70
       Yes    0    0

               Accuracy : 0.9535
                 95% CI : (0.9416, 0.9636)
    No Information Rate : 0.9535
    P-Value [Acc > NIR] : 0.5317

                  Kappa : 0

 Mcnemar's Test P-Value : <2e-16

            Sensitivity : 1.0000
            Specificity : 0.0000
         Pos Pred Value : 0.9535
         Neg Pred Value :    NaN
             Prevalence : 0.9535
         Detection Rate : 0.9535
   Detection Prevalence : 1.0000
      Balanced Accuracy : 0.5000

       'Positive' Class : No

> end_time <- Sys.time()
> end_time-start_time
Time difference of 20.29942 secs
>
```



IMPLEMENTATION IN THE BIG DATA FRAMEWORK:

1. K-MEANS ALGORITHM

```
In [31]: import findspark
         findspark.init()
         import time as t

In [32]: from pyspark.sql import SparkSession
         spark = SparkSession.builder.appName('patients').getOrCreate()

In [33]: from pyspark.ml.clustering import KMeans
         dataset = spark.read.csv("D:\COURSE PDFs\College Notes\SEMESTER VI\Big Data Analytics\Healthcare Stroke Dataset\data.csv",header=

In [34]: dataset.head(1)

Out[34]: [Row(id=9046, gender=1, age=67.0, hypertension=0, heart_disease=1, ever_married=1, work_type=1, Residence_type=1, avg_glucose_l
         evel=228.69, smoking_status=1, stroke=1)]

In [35]: dataset.printSchema()

         root
          |-- id: integer (nullable = true)
          |-- gender: integer (nullable = true)
          |-- age: double (nullable = true)
          |-- hypertension: integer (nullable = true)
          |-- heart_disease: integer (nullable = true)
          |-- ever_married: integer (nullable = true)
          |-- work_type: integer (nullable = true)
          |-- Residence_type: integer (nullable = true)
          |-- avg_glucose_level: double (nullable = true)
          |-- smoking_status: integer (nullable = true)
          |-- stroke: integer (nullable = true)
```

In [36]:
```
columns_to_drop = ['avg_glucose_level']
dataset = dataset.drop(*columns_to_drop)
dataset.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- gender: integer (nullable = true)
 |-- age: double (nullable = true)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- ever_married: integer (nullable = true)
 |-- work_type: integer (nullable = true)
 |-- Residence_type: integer (nullable = true)
 |-- smoking_status: integer (nullable = true)
 |-- stroke: integer (nullable = true)
```

In [37]:
```
dataset.columns
```

Out[37]:
```
['id',
 'gender',
 'age',
 'hypertension',
 'heart_disease',
 'ever_married',
 'work_type',
 'Residence_type',
 'smoking_status',
 'stroke']
```

In [38]:
```
start_time=t.time()
from pyspark.ml.linalg import Vectors
from pyspark.ml.feature import VectorAssembler
```

In [39]:
```
from pyspark.sql.types import IntegerType
dataset = dataset.withColumn("smoking_status", dataset["smoking_status"].cast(IntegerType()))
```

In [40]:
```
dataset.printSchema()
```

```
root
 |-- id: integer (nullable = true)
 |-- gender: integer (nullable = true)
 |-- age: double (nullable = true)
 |-- hypertension: integer (nullable = true)
 |-- heart_disease: integer (nullable = true)
 |-- ever_married: integer (nullable = true)
 |-- work_type: integer (nullable = true)
 |-- Residence_type: integer (nullable = true)
 |-- smoking_status: integer (nullable = true)
 |-- stroke: integer (nullable = true)
```

In [41]:
```
feat_cols = [
 'gender','age','hypertension','heart_disease','ever_married','work_type','Residence_type','smoking_status','stroke']
```

In [42]:
```
vec_assembler = VectorAssembler(inputCols = feat_cols, outputCol='features')
```

In [43]:
```
final_data = vec_assembler.transform(dataset)
```

In [44]:
```
from pyspark.ml.feature import StandardScaler
```

In [45]:
```
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)
```

In [45]:
```
scaler = StandardScaler(inputCol="features", outputCol="scaledFeatures", withStd=True, withMean=False)
```

In [46]:
```
final_data
```

Out[46]:
```
DataFrame[id: int, gender: int, age: double, hypertension: int, heart_disease: int, ever_married: int, work_type: int, Residenc
e_type: int, smoking_status: int, stroke: int, features: vector]
```

In [47]:
```
scalerModel = scaler.fit(final_data)
```

In [48]:
```
cluster_final_data = scalerModel.transform(final_data)
```

In [49]:
```
kmeans3 = KMeans(featuresCol='scaledFeatures',k=3)
kmeans2 = KMeans(featuresCol='scaledFeatures',k=2)
```

In [50]:
```
model3 = kmeans3.fit(cluster_final_data)
model2 = kmeans2.fit(cluster_final_data)
```

In [51]:
```
from pyspark.ml.clustering import KMeans
from pyspark.ml.evaluation import ClusteringEvaluator
```

In [52]:
```
predictions3 = model3.transform(cluster_final_data)
predictions2 = model2.transform(cluster_final_data)
```

In [53]:
```
evaluator = ClusteringEvaluator()
```

In [53]: evaluator = ClusteringEvaluator()

In [54]:
```
silhouette = evaluator.evaluate(predictions3)
print("With k=3 Silhouette with squared euclidean distance = " + str(silhouette))
silhouette = evaluator.evaluate(predictions2)
print("With k=2 Silhouette with squared euclidean distance = " + str(silhouette))
```

```
With k=3 Silhouette with squared euclidean distance = -0.051780492409706495
With k=2 Silhouette with squared euclidean distance = 0.08301564925690019
```

In [55]:
```
centers=model2.clusterCenters()
print("Cluster Centers:")
for center in centers:
    print(center)
```

```
Cluster Centers:
[8.30328948e-01 1.77326310e+00 0.00000000e+00 1.77700557e-01
 1.30316618e+00 1.68107417e+00 2.98497153e+00 1.14720772e+00
 2.09625525e-03]
[0.90508111 2.81444811 2.47274111 0.63844992 1.8911928  1.39416261
 2.97800074 0.81946711 1.68944126]
```

In [56]:
```
for k in range(2,5):
    kmeans = KMeans(featuresCol='scaledFeatures',k=k)
    model = kmeans.fit(cluster_final_data)
    predictions = model.transform(cluster_final_data)
    evaluator = ClusteringEvaluator()
    silhouette = evaluator.evaluate(predictions)
    print("With K={}".format(k))
    print("Silhouette with squared euclidean distance = " + str(silhouette))
    print('--'*30)
```

```
With K=2
```

```
In [56]: for k in range(2,5):
             kmeans = KMeans(featuresCol='scaledFeatures',k=k)
             model = kmeans.fit(cluster_final_data)
             predictions = model.transform(cluster_final_data)
             evaluator = ClusteringEvaluator()
             silhouette = evaluator.evaluate(predictions)
             print("With K={}".format(k))
             print("Silhouette with squared euclidean distance = " + str(silhouette))
             print('--'*30)
```

```
With K=2
Silhouette with squared euclidean distance = 0.08301564925690019
------------------------------------------------------------
With K=3
Silhouette with squared euclidean distance = -0.051780492409706495
------------------------------------------------------------
With K=4
Silhouette with squared euclidean distance = -0.08035770895238105
------------------------------------------------------------
```

```
In [57]: model3.transform(cluster_final_data).groupBy('prediction').count().show()
```

```
+----------+-----+
|prediction|count|
+----------+-----+
|         1|  432|
|         2|  249|
|         0| 4429|
+----------+-----+
```

```
In [58]: model2.transform(cluster_final_data).groupBy('prediction').count().show()
```


```
In [58]: model2.transform(cluster_final_data).groupBy('prediction').count().show()
```


```
+----------+-----+
|prediction|count|
+----------+-----+
|         1|  679|
|         0| 4431|
+----------+-----+
```

```
In [59]: end_time = t.time()
```

```
In [60]: end_time - start_time
Out[60]: 8.335537672042847
```

2. DECISION TREE ALGORITHM

In [1]:
```
from pyspark import SparkContext
sc = SparkContext(master = 'local')
```

In [2]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder \
          .appName("Python Spark SQL basic example") \
          .config("spark.some.config.option", "some-value") \
          .getOrCreate()
```

In [3]:
```
import time as t

cuse = spark.read.csv('D:\COURSE PDFs\College Notes\SEMESTER VI\Big Data Analytics\Healthcare Stroke Dataset\data.csv', header=Tr
cuse.show(20)
start_time=t.time()
```

```
+-----+------+----+------------+-------------+------------+---------+--------------+-----------------+--------------+------+
|   id|gender| age|hypertension|heart_disease|ever_married|work_type|Residence_type|avg_glucose_level|smoking_status|stroke|
+-----+------+----+------------+-------------+------------+---------+--------------+-----------------+--------------+------+
| 9046|     1|67.0|           0|            1|           1|        1|             1|           228.69|             1|     1|
|51676|     0|61.0|           0|            0|           1|        2|             2|           202.21|             0|     1|
|31112|     1|80.0|           0|            1|           1|        1|             2|           105.92|             0|     1|
|60182|     0|49.0|           0|            0|           1|        1|             1|           171.23|             2|     1|
| 1665|     0|79.0|           1|            0|           1|        2|             2|           174.12|             0|     1|
|56669|     1|81.0|           0|            0|           1|        1|             1|           186.21|             1|     1|
|53882|     1|74.0|           1|            1|           1|        1|             2|            70.09|             0|     1|
|10434|     0|69.0|           0|            0|           0|        1|             1|            94.39|             0|     1|
|27419|     0|59.0|           0|            0|           1|        1|             2|            76.15|             3|     1|
|60491|     0|78.0|           0|            0|           1|        1|             1|            58.57|             3|     1|
|12109|     0|81.0|           1|            0|           1|        1|             2|            80.43|             0|     1|
```

```
|56669|     1|81.0|           0|            0|           1|        1|             1|           186.21|             1|     1|
|53882|     1|74.0|           1|            1|           1|        1|             2|            70.09|             0|     1|
|10434|     0|69.0|           0|            0|           0|        1|             1|            94.39|             0|     1|
|27419|     0|59.0|           0|            0|           1|        1|             2|            76.15|             3|     1|
|60491|     0|78.0|           0|            0|           1|        1|             1|            58.57|             3|     1|
|12109|     0|81.0|           1|            0|           1|        1|             2|            80.43|             0|     1|
|12095|     0|61.0|           0|            1|           1|        3|             2|           120.46|             2|     1|
|12175|     0|54.0|           0|            0|           1|        1|             1|           104.51|             2|     1|
| 8213|     1|78.0|           0|            1|           1|        1|             1|           219.84|             3|     1|
| 5317|     0|79.0|           0|            1|           1|        1|             1|           214.09|             0|     1|
|58202|     0|50.0|           1|            0|           1|        2|             2|           167.41|             0|     1|
|56112|     1|64.0|           0|            1|           1|        1|             1|           191.61|             2|     1|
|34120|     1|75.0|           1|            0|           1|        1|             1|           221.29|             2|     1|
|27458|     0|60.0|           0|            0|           0|        1|             1|            89.22|             0|     1|
|25226|     1|57.0|           0|            1|           0|        3|             1|           217.08|             3|     1|
+-----+------+----+------------+-------------+------------+---------+--------------+-----------------+--------------+------+
only showing top 20 rows
```

In [4]:
```
from pyspark.ml.feature import StringIndexer, OneHotEncoder, VectorAssembler
from pyspark.ml import Pipeline

# categorical columns
categorical_columns = cuse.columns[0:3]
```

In [5]:
```
stringindexer_stages = [StringIndexer(inputCol=c, outputCol='strindexed_' + c) for c in categorical_columns]
stringindexer_stages += [StringIndexer(inputCol='stroke', outputCol='label')]
```

In [6]:
```
onehotencoder_stages = [OneHotEncoder(inputCol='strindexed_' + c, outputCol='onehot_' + c) for c in categorical_columns]
```

In [7]:
```
feature_columns = ['onehot_' + c for c in categorical_columns]
vectorassembler_stage = VectorAssembler(inputCols=feature_columns, outputCol='features')
```

In [7]:
```
feature_columns = ['onehot_' + c for c in categorical_columns]
vectorassembler_stage = VectorAssembler(inputCols=feature_columns, outputCol='features')
```

In [8]:
```
all_stages = stringindexer_stages + onehotencoder_stages + [vectorassembler_stage]
pipeline = Pipeline(stages=all_stages)
```

In [9]:
```
pipeline_model = pipeline.fit(cuse)
```

In [10]:
```
final_columns = feature_columns + ['features', 'label']
cuse_df = pipeline_model.transform(cuse).\
            select(final_columns)

cuse_df.show(10)
```

```
+-------------------+-------------+---------------+--------------------+-----+
|          onehot_id|onehot_gender|     onehot_age|            features|label|
+-------------------+-------------+---------------+--------------------+-----+
|(5109,[5053],[1.0])|(2,[1],[1.0])|(103,[63],[1.0])|(5214,[5053,5110,...|  1.0|
|(5109,[3218],[1.0])|(2,[0],[1.0])|(103,[15],[1.0])|(5214,[3218,5109,...|  1.0|
|(5109,[1593],[1.0])|(2,[1],[1.0])|(103,[28],[1.0])|(5214,[1593,5110,...|  1.0|
|(5109,[3907],[1.0])|(2,[0],[1.0])|(103,[12],[1.0])|(5214,[3907,5109,...|  1.0|
| (5109,[530],[1.0])|(2,[0],[1.0])| (103,[7],[1.0])|(5214,[530,5109,5...|  1.0|
|(5109,[3618],[1.0])|(2,[1],[1.0])|(103,[41],[1.0])|(5214,[3618,5110,...|  1.0|
|(5109,[3392],[1.0])|(2,[1],[1.0])|(103,[74],[1.0])|(5214,[3392,5110,...|  1.0|
|  (5109,[29],[1.0])|(2,[0],[1.0])|(103,[55],[1.0])|(5214,[29,5109,51...|  1.0|
|(5109,[1306],[1.0])|(2,[0],[1.0])|(103,[10],[1.0])|(5214,[1306,5109,...|  1.0|
|(5109,[3931],[1.0])|(2,[0],[1.0])| (103,[0],[1.0])|(5214,[3931,5109,...|  1.0|
+-------------------+-------------+---------------+--------------------+-----+
only showing top 10 rows
```

In [11]:
```
training, test = cuse_df.randomSplit([0.8, 0.2], seed=1234)
```

In [12]:
```
from pyspark.ml.regression import GeneralizedLinearRegression
from pyspark.ml.classification import LogisticRegression, DecisionTreeClassifier

dt = DecisionTreeClassifier(featuresCol='features', labelCol='label')
```

In [13]:
```
from pyspark.ml.tuning import ParamGridBuilder
param_grid = ParamGridBuilder().\
    addGrid(dt.maxDepth, [2,3,4,5]).\
    build()
```

In [14]:
```
from pyspark.ml.evaluation import BinaryClassificationEvaluator
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction", metricName="areaUnderROC")
```

In [15]:
```
from pyspark.ml.tuning import CrossValidator
cv = CrossValidator(estimator=dt, estimatorParamMaps=param_grid, evaluator=evaluator, numFolds=4)
```

In [16]:
```
cv_model = cv.fit(cuse_df)
```

In [17]:
```
show_columns = ['features', 'label', 'prediction', 'rawPrediction', 'probability']
```

In [18]:
```
pred_training_cv = cv_model.transform(training)
pred_training_cv.select(show_columns).show(10, truncate=False)
```

```
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
|features                           |label|prediction|rawPrediction |probability                              |
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
|(5214,[5110,5154],[1.0,1.0])       |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[1,5109,5125],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[2,5109,5125],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[4,5110,5146],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[5,5109,5125],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[6,5109,5132],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[7,5109,5114],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[8,5110,5132],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[9,5110,5151],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[10,5109,5176],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
only showing top 10 rows
```

```
In [19]: pred_test_cv = cv_model.transform(test)
         pred_test_cv.select(show_columns).show(10, truncate=False)
```

```
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
|features                           |label|prediction|rawPrediction |probability                              |
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
|(5214,[0,5110,5172],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[3,5110,5118],[1.0,1.0,1.0]) |0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[20,5109,5116],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[23,5110,5113],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[24,5109,5160],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[25,5109,5133],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[26,5109,5154],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[35,5110,5143],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[44,5110,5160],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
|(5214,[46,5109,5185],[1.0,1.0,1.0])|0.0  |0.0       |[4808.0,232.0]|[0.953968253968254,0.046031746031746035]|
+-----------------------------------+-----+----------+-------------+-----------------------------------------+
```

```
In [20]: end_time=t.time()
```

```
In [21]: end_time-start_time

Out[21]: 27.774269342422485
```

COMPARISON OF THE RESULTS:

Comparing the results of both the algorithms of big data framework, we can see that k-means algorithm is taking less time to run as compared to the decision tree algorithm.

Time taken to run K-Means algorithm in PySpark : 8.335 seconds

Time taken to run Decision Tree algorithm in PySpark : 27.774 seconds.

Time taken to run KNN algorithm in RStudio (Traditional) : 20.299 seconds.

Time taken to run C4.5 algorithm in RStudio (Traditional) : 0.542 seconds.

In K-Means algorithm, we have taken values of K = 2,3.

The Silhouette score is a metric used to calculate the goodness of a clustering technique. It ranges from -1 to 1.

1 : Means clusters are well apart from each other and clearly distinguished.

0 : Means clusters are indifferent, or we can say that the distance between clusters is not significant.

-1 : Means clusters are assigned in the wrong way.

The Silhouette score of K = 2 is 0.0830

The Silhouette score of K = 3 is -0.0517

CONCLUSION:

In this paper, we developed a number of algorithms combining both traditional and big data frameworks, including Decision Tree, KNN classifier, C4.5, and K-Means, which were compared and yielded encouraging results. We came to the conclusion that machine learning methods performed better in this study. Many academics have previously proposed that we should deploy machine learning when the dataset is small, which this work proves. Confusion matrix and precision, as well as the time difference, are utilized as comparison approaches. When data preprocessing was used, the K-Means algorithm performed better in the ML approach for these features that were in the dataset.

Additionally, the computation time was lowered, which is beneficial when deploying a model. It was also discovered that the dataset should be normalized; otherwise, the training model can become overfitted, resulting in limited accuracy when a model is evaluated for real-world data problems that differ greatly from the dataset on which the model was trained. It was also discovered that statistical analysis is crucial when analyzing a dataset. The problem here is that the dataset's sample size is relatively small. If a large dataset is present, the results can increase very much in deep learning and ML as well.