In [24]:
```
from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('ml-bank').getOrCreate()
df = spark.read.csv('diabetes.csv', header = True, inferSchema = True)

df.printSchema()
```

```
root
 |-- Pregnancies: integer (nullable = true)
 |-- Glucose: integer (nullable = true)
 |-- BloodPressure: integer (nullable = true)
 |-- SkinThickness: integer (nullable = true)
 |-- Insulin: integer (nullable = true)
 |-- BMI: double (nullable = true)
 |-- DiabetesPedigreeFunction: double (nullable = true)
 |-- Age: integer (nullable = true)
 |-- Outcome: integer (nullable = true)
```

In [25]:
```
df.describe().select("Summary","Pregnancies","Glucose","BloodPressure").show()
```

```
+-------+------------------+-----------------+------------------+
|Summary|       Pregnancies|          Glucose|     BloodPressure|
+-------+------------------+-----------------+------------------+
|  count|               768|              768|               768|
|   mean|3.8450520833333335|    120.89453125|       69.10546875|
| stddev|  3.36957806269887|31.97261819513622|19.355807170644777|
|    min|                 0|                0|                 0|
|    max|                17|              199|               122|
+-------+------------------+-----------------+------------------+
```

In [26]:
```
df.describe().select("Summary","SkinThickness","Insulin").show()
```

```
+-------+-----------------+------------------+
|Summary|     SkinThickness|           Insulin|
+-------+-----------------+------------------+
|  count|              768|               768|
|   mean|20.536458333333332| 79.79947916666667|
| stddev|15.952217567727642|115.24400235133803|
|    min|                0|                 0|
|    max|               99|               846|
+-------+-----------------+------------------+
```

In [27]:
```
cols=df.columns
cols.remove("Outcome")
# Let us import the vector assembler
from pyspark.ml.feature import VectorAssembler
assembler = VectorAssembler(inputCols=cols,outputCol="features")
# Now let us use the transform method to transform our dataset
df=assembler.transform(df)
df.select("features").show(truncate=False)
```

```
+-----------------------------------------+
|features                                 |
+-----------------------------------------+
|[6.0,148.0,72.0,35.0,0.0,33.6,0.627,50.0]  |
|[1.0,85.0,66.0,29.0,0.0,26.6,0.351,31.0]   |
|[8.0,183.0,64.0,0.0,0.0,23.3,0.672,32.0]   |
|[1.0,89.0,66.0,23.0,94.0,28.1,0.167,21.0]  |
|[0.0,137.0,40.0,35.0,168.0,43.1,2.288,33.0]|
|[5.0,116.0,74.0,0.0,0.0,25.6,0.201,30.0]   |
|[3.0,78.0,50.0,32.0,88.0,31.0,0.248,26.0]  |
|[10.0,115.0,0.0,0.0,0.0,35.3,0.134,29.0]   |
|[2.0,197.0,70.0,45.0,543.0,30.5,0.158,53.0]|
|[8.0,125.0,96.0,0.0,0.0,0.0,0.232,54.0]    |
|[4.0,110.0,92.0,0.0,0.0,37.6,0.191,30.0]   |
|[10.0,168.0,74.0,0.0,0.0,38.0,0.537,34.0]  |
|[10.0,139.0,80.0,0.0,0.0,27.1,1.441,57.0]  |
|[1.0,189.0,60.0,23.0,846.0,30.1,0.398,59.0]|
|[5.0,166.0,72.0,19.0,175.0,25.8,0.587,51.0]|
|[7.0,100.0,0.0,0.0,0.0,30.0,0.484,32.0]    |
|[0.0,118.0,84.0,47.0,230.0,45.8,0.551,31.0]|
|[7.0,107.0,74.0,0.0,0.0,29.6,0.254,31.0]   |
|[1.0,103.0,30.0,38.0,83.0,43.3,0.183,33.0] |
|[1.0,115.0,70.0,30.0,96.0,34.6,0.529,32.0] |
+-----------------------------------------+
only showing top 20 rows
```

In [31]:
```
df.select("features","Outcome").show()
```

```
+--------------------+-------+
|            features|Outcome|
+--------------------+-------+
|[6.0,148.0,72.0,3...|      1|
|[1.0,85.0,66.0,29...|      0|
|[8.0,183.0,64.0,0...|      1|
|[1.0,89.0,66.0,23...|      0|
|[0.0,137.0,40.0,3...|      1|
|[5.0,116.0,74.0,0...|      0|
|[3.0,78.0,50.0,32...|      1|
|[10.0,115.0,0.0,0...|      0|
|[2.0,197.0,70.0,4...|      1|
|[8.0,125.0,96.0,0...|      1|
|[4.0,110.0,92.0,0...|      0|
|[10.0,168.0,74.0,...|      1|
|[10.0,139.0,80.0,...|      0|
|[1.0,189.0,60.0,2...|      1|
|[5.0,166.0,72.0,1...|      1|
|[7.0,100.0,0.0,0....|      1|
|[0.0,118.0,84.0,4...|      1|
|[7.0,107.0,74.0,0...|      1|
|[1.0,103.0,30.0,3...|      0|
|[1.0,115.0,70.0,3...|      1|
+--------------------+-------+
only showing top 20 rows
```

In [35]:
```
train=df.select("features","Outcome")
test=df.select("features","Outcome")
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="Outcome", featuresCol="features")
model=lr.fit(train)
predict_train=model.transform(train)
predict_test=model.transform(test)
predict_test.show(10)
```

```
+--------------------+-------+--------------------+--------------------+-------
---+
|            features|Outcome|       rawPrediction|         probability|predict
ion|
+--------------------+-------+--------------------+--------------------+-------
---+
|[6.0,148.0,72.0,3...|      1|[-0.9530437928800...|[0.27827310281794...|
1.0|
|[1.0,85.0,66.0,29...|      0|[2.97341355586980...|[0.95135848467122...|
0.0|
|[8.0,183.0,64.0,0...|      1|[-1.3658090656129...|[0.20329779887119...|
1.0|
|[1.0,89.0,66.0,23...|      0|[3.13654488056892...|[0.95837526675900...|
0.0|
|[0.0,137.0,40.0,3...|      1|[-2.2217341923019...|[0.09781565894362...|
1.0|
|[5.0,116.0,74.0,0...|      0|[1.76126995738811...|[0.85336864205596...|
0.0|
|[3.0,78.0,50.0,32...|      1|[2.64049246425899...|[0.93342257508603...|
0.0|
|[10.0,115.0,0.0,0...|      0|[-0.5952553911576...|[0.35542993414812...|
1.0|
|[2.0,197.0,70.0,4...|      1|[-0.8922643314355...|[0.29064277002922...|
1.0|
|[8.0,125.0,96.0,0...|      1|[3.27794169861517...|[0.96366428004897...|
0.0|
+--------------------+-------+--------------------+--------------------+-------
---+
only showing top 10 rows
```

In [37]:
```
from pyspark.ml import Pipeline
from pyspark.ml.classification import LogisticRegression
from pyspark.ml.feature import HashingTF, Tokenizer

# Prepare training documents from a list of (id, text, label) tuples.
training = spark.createDataFrame([
    (0, "a b c d e spark", 1.0),
    (1, "b d", 0.0),
    (2, "spark f g h", 1.0),
    (3, "hadoop mapreduce", 0.0)
], ["id", "text", "label"])

# Configure an ML pipeline, which consists of three stages: tokenizer, hashingTF,
tokenizer = Tokenizer(inputCol="text", outputCol="words")
hashingTF = HashingTF(inputCol=tokenizer.getOutputCol(), outputCol="features")
lr = LogisticRegression(maxIter=10, regParam=0.001)
pipeline = Pipeline(stages=[tokenizer, hashingTF, lr])

# Fit the pipeline to training documents.
model = pipeline.fit(training)

# Prepare test documents, which are unlabeled (id, text) tuples.
test = spark.createDataFrame([
    (4, "spark i j k"),
    (2, "spark f g h"),
    (5, "l m n"),
    (6, "spark hadoop spark"),
    (7, "apache hadoop")
], ["id", "text"])

# Make predictions on test documents and print columns of interest.
prediction = model.transform(test)
selected = prediction.select("id", "text", "probability", "prediction")
for row in selected.collect():
    rid, text, prob, prediction = row
    print("(%d, %s) --> prob=%s, prediction=%f" % (rid, text, str(prob), predict
```

```
(4, spark i j k) --> prob=[0.1596407738787475,0.8403592261212525], prediction=
1.000000
(2, spark f g h) --> prob=[0.0014541569986709774,0.9985458430013291], predictio
n=1.000000
(5, l m n) --> prob=[0.8378325685476744,0.16216743145232562], prediction=0.0000
00
(6, spark hadoop spark) --> prob=[0.06926633132976037,0.9307336686702395], pred
iction=1.000000
(7, apache hadoop) --> prob=[0.9821575333444218,0.01784246665557808], predictio
n=0.000000
```

In [28]:
```
from pyspark.ml.feature import StandardScaler
standardscaler=StandardScaler().setInputCol("features").setOutputCol("Scaled_fea
df=standardscaler.fit(df).transform(df)
df.select("features","Scaled_features").show(5)
```

```
+--------------------+--------------------+
|            features|     Scaled_features|
+--------------------+--------------------+
|[6.0,148.0,72.0,3...|[1.78063837321943...|
|[1.0,85.0,66.0,29...|[0.29677306220323...|
|[8.0,183.0,64.0,0...|[2.37418449762590...|
|[1.0,89.0,66.0,23...|[0.29677306220323...|
|[0.0,137.0,40.0,3...|[0.0,4.2849165233...|
+--------------------+--------------------+
only showing top 5 rows
```

In [ ]:
```
from pyspark.ml.classification import LogisticRegression
lr = LogisticRegression(labelCol="Outcome", featuresCol="Aspect",weightCol="clas
model=lr.fit(train)
predict_train=model.transform(train)
predict_test=model.transform(test)
predict_test.select("Outcome","prediction").show(10)
```