```
#pip install pyspark

from pyspark.sql import SparkSession
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.classification import RandomForestClassifier,
        GBTClassifier
from pyspark.ml import Pipeline
from pyspark.ml.evaluation import BinaryClassificationEvaluator


from pyspark.sql import SparkSession
spark = SparkSession.builder.appName('Credit Analysis').getOrCreate()

data = spark.read.csv("Loan Data.csv", inferSchema=True, header=True)

data.printSchema()

root
 |-- loan_amnt: integer (nullable = true)
 |-- funded_amnt: integer (nullable = true)
 |-- funded_amnt_inv: integer (nullable = true)
 |-- term: integer (nullable = true)
 |-- int_rate: double (nullable = true)
 |-- emp_length: integer (nullable = true)
 |-- home_ownership: integer (nullable = true)
 |-- annual_inc: double (nullable = true)
 |-- verification_status: integer (nullable = true)
 |-- purpose: string (nullable = true)
 |-- 1/0: integer (nullable = true)


data.head(1)

[Row(loan_amnt=3600, funded_amnt=3600, funded_amnt_inv=3600, term=0,
int_rate=13.99, emp_length=10, home_ownership=4, annual_inc=55000.0,
verification_status=0, purpose='debt_consolidation', 1/0=1)]

from pyspark.ml.feature import VectorAssembler

data.columns

['loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
```

```
 'term',
 'int_rate',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 'purpose',
 '1/0']

assembler = VectorAssembler(inputCols=['loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'verification_status',
 '1/0'],
outputCol='features')


from pyspark.ml.feature import StringIndexer

from pyspark.ml.feature import StringIndexer

indexer = StringIndexer(inputCol=('purpose'),outputCol=
        ('purposeIndex'))

output = assembler.transform(data)

output_fixed = indexer.fit(output).transform(output)

output_fixed.printSchema()

root
 |-- loan_amnt: integer (nullable = true)
 |-- funded_amnt: integer (nullable = true)
 |-- funded_amnt_inv: integer (nullable = true)
 |-- term: integer (nullable = true)
 |-- int_rate: double (nullable = true)
 |-- emp_length: integer (nullable = true)
 |-- home_ownership: integer (nullable = true)
 |-- annual_inc: double (nullable = true)
 |-- verification_status: integer (nullable = true)
 |-- purpose: string (nullable = true)
```

```
 |-- 1/0: integer (nullable = true)
 |-- features: vector (nullable = true)
 |-- purposeIndex: double (nullable = false)
```

```
final_data = output_fixed.select('features','purposeIndex')
```

```
train_data,test_data = final_data.randomSplit([0.7,0.3])
```

```
from pyspark.ml.classification import (DecisionTreeClassifier,
                                       RandomForestClassifier,
                                      )
```

```
from pyspark.ml import Pipeline
```

```
dtc = DecisionTreeClassifier(labelCol='purposeIndex',
        featuresCol='features')
rfc = RandomForestClassifier(numTrees=150,labelCol='purposeIndex',
        featuresCol='features')
```

```
dtc_model=dtc.fit(train_data)
rfc_model=rfc.fit(train_data)
```

```
dtc_pred = dtc_model.transform(test_data)
rfc_pred = rfc_model.transform(test_data)
```

```
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
```

```
my_multi_eval =
        MulticlassClassificationEvaluator(labelCol='purposeIndex')
```

```
print('RFC')
print(my_multi_eval.evaluate(rfc_pred))
```

```
RFC
0.4202160395799258
```

```
print('DTC')
print(my_multi_eval.evaluate(dtc_pred))
```

```
DTC
0.44335634654276584
```

```
from pyspark.ml.classification import LogisticRegression
```

```
lr_churn = LogisticRegression(labelCol='purposeIndex')
```

```
fitted_churn_model = lr_churn.fit(train_data)
```

```
pred_and_labels = fitted_churn_model.evaluate(test_data)
```

```
churn_eval = MulticlassClassificationEvaluator(
                                                labelCol='purposeIndex')

auc = churn_eval.evaluate(pred_and_labels.predictions)

auc
```

```
0.4400728712776844
```

```
acc_eval = MulticlassClassificationEvaluator(labelCol='purposeIndex',
                                             metricName='accuracy')
rfc_acc = acc_eval.evaluate(rfc_pred)
rfc_acc
```

```
0.5690208667736758
```