databricks Machine_Learning

```
spark
```

```
Out[949]: <pyspark.sql.session.SparkSession at 0x7f8819619cd0>
```

```
sc
```

```
Out[950]: <SparkContext master=local[8] appName=Databricks Shell>
```

```
sqlContext
```

```
Out[951]: <pyspark.sql.context.HiveContext at 0x7f8819619d90>
```

Identifying Safe Loans with Decision Trees

Read in Loan data

```
loan_df = sqlContext.read.format("csv").options(header = "true", inferschema =
"true").load("/FileStore/tables/7oewn9wr1506610007136/lending_club_data-
fbee9.csv")
```

```
display(loan_df)
```

id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installmen
1077501	1296599	5000	5000	4975	36 months	10.65	162.87

Showing the first 1000 rows.

```
loan_df.head(5)
```

```
Out[954]:
[Row(id=1077501, member_id=1296599, loan_amnt=5000, funded_amnt=5000, funded_a
mnt_inv=4975, term=u' 36 months', int_rate=10.65, installment=162.87, grade=
u'B', sub_grade=u'B2', emp_title=None, emp_length=u'10+ years', home_ownership
=u'RENT', annual_inc=24000, is_inc_v=u'Verified', issue_d=u'20111201T000000',
 loan_status=u'Fully Paid', pymnt_plan=u'n', url=u'https://www.lendingclub.co
m/browse/loanDetail.action?loan_id=1077501', desc=u'  Borrower added on 12/22/
11 > I need to upgrade my business technologies.<br>', purpose=u'credit_card',
 title=u'Computer', zip_code=u'860xx', addr_state=u'AZ', dti=u'27.65', delinq_
2yrs=u'0', earliest_cr_line=u'19850101T000000', inq_last_6mths=u'1', mths_sinc
e_last_delinq=None, mths_since_last_record=None, open_acc=u'3', pub_rec=u'0',
 revol_bal=u'13648', revol_util=u'83.7', total_acc=u'9', initial_list_status=
u'f', out_prncp=u'0', out_prncp_inv=u'0', total_pymnt=u'5861.07', total_pymnt_
inv=u'5831.78', total_rec_prncp=u'5000', total_rec_int=u'861.07', total_rec_la
te_fee=u'0', recoveries=u'0', collection_recovery_fee=u'0', last_pymnt_d=u'201
50101T000000', last_pymnt_amnt=u'171.62', next_pymnt_d=None, last_credit_pull_
d=u'20150101T000000', collections_12_mths_ex_med=u'0', mths_since_last_major_d
erog=None, policy_code=u'1', not_compliant=u'0', status=u'Fully Paid', inactiv
e_loans=u'1', bad_loans=u'0', emp_length_num=u'11', grade_num=u'5', sub_grade_
num=u'0.4', delinq_2yrs_zero=u'1', pub_rec_zero=u'1', collections_12_mths_zero
=u'1', short_emp=u'0', payment_inc_ratio=u'8.1435', final_d=u'20141201T00000
```

Exploring Features

```
display(loan_df.describe())
```

summary	id	member_id	loan_amnt	funded_amnt	fu
count	122607	122607	122607	122607	12
mean	4728452.039141322	5493222.4608546	12809.73374277162	12736.12375312992	12
stddev	5938516.991060433	6604693.479637761	7932.313397523164	7887.167118457222	79

```
loan_df.columns
```

```
Out[956]:
['id',
 'member_id',
 'loan_amnt',
 'funded_amnt',
 'funded_amnt_inv',
 'term',
 'int_rate',
 'installment',
 'grade',
 'sub_grade',
 'emp_title',
 'emp_length',
 'home_ownership',
 'annual_inc',
 'is_inc_v',
 'issue_d',
 'loan_status',
 'pymnt_plan',
 'url',
 'desc',
```

```
loan_df.printSchema()
```

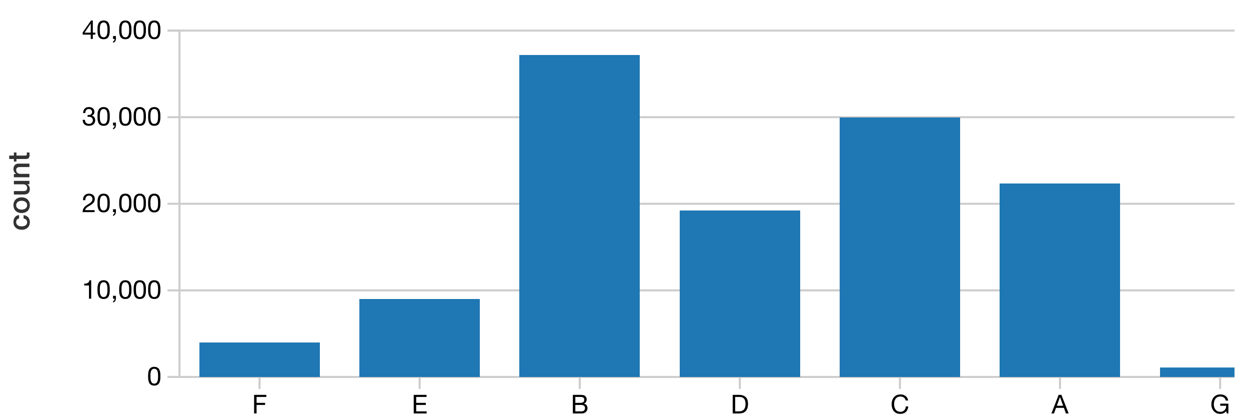
```
root
 |-- id: integer (nullable = true)
 |-- member_id: integer (nullable = true)
 |-- loan_amnt: integer (nullable = true)
 |-- funded_amnt: integer (nullable = true)
 |-- funded_amnt_inv: integer (nullable = true)
 |-- term: string (nullable = true)
 |-- int_rate: double (nullable = true)
 |-- installment: double (nullable = true)
 |-- grade: string (nullable = true)
 |-- sub_grade: string (nullable = true)
 |-- emp_title: string (nullable = true)
 |-- emp_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: integer (nullable = true)
 |-- is_inc_v: string (nullable = true)
 |-- issue_d: string (nullable = true)
 |-- loan_status: string (nullable = true)
 |-- pymnt_plan: string (nullable = true)
 |-- url: string (nullable = true)
 |-- desc: string (nullable = true)
```

```
loan_df.cache()
```

```
Out[958]: DataFrame[id: int, member_id: int, loan_amnt: int, funded_amnt: int,
 funded_amnt_inv: int, term: string, int_rate: double, installment: double, gra
de: string, sub_grade: string, emp_title: string, emp_length: string, home_owne
rship: string, annual_inc: int, is_inc_v: string, issue_d: string, loan_status:
 string, pymnt_plan: string, url: string, desc: string, purpose: string, title:
 string, zip_code: string, addr_state: string, dti: string, delinq_2yrs: strin
g, earliest_cr_line: string, inq_last_6mths: string, mths_since_last_delinq: st
ring, mths_since_last_record: string, open_acc: string, pub_rec: string, revol_
bal: string, revol_util: string, total_acc: string, initial_list_status: strin
g, out_prncp: string, out_prncp_inv: string, total_pymnt: string, total_pymnt_i
nv: string, total_rec_prncp: string, total_rec_int: string, total_rec_late_fee:
 string, recoveries: string, collection_recovery_fee: string, last_pymnt_d: str
ing, last_pymnt_amnt: string, next_pymnt_d: string, last_credit_pull_d: string,
 collections_12_mths_ex_med: string, mths_since_last_major_derog: string, polic
y_code: string, not_compliant: string, status: string, inactive_loans: string,
 bad_loans: string, emp_length_num: string, grade_num: string, sub_grade_num: s
tring, delinq_2yrs_zero: string, pub_rec_zero: string, collections_12_mths_zer
o: string, short_emp: string, payment_inc_ratio: string, final_d: string, last_
delinq_none: string, last_record_none: string, last_major_derog_none: string]
```

Look at distribution of grade data

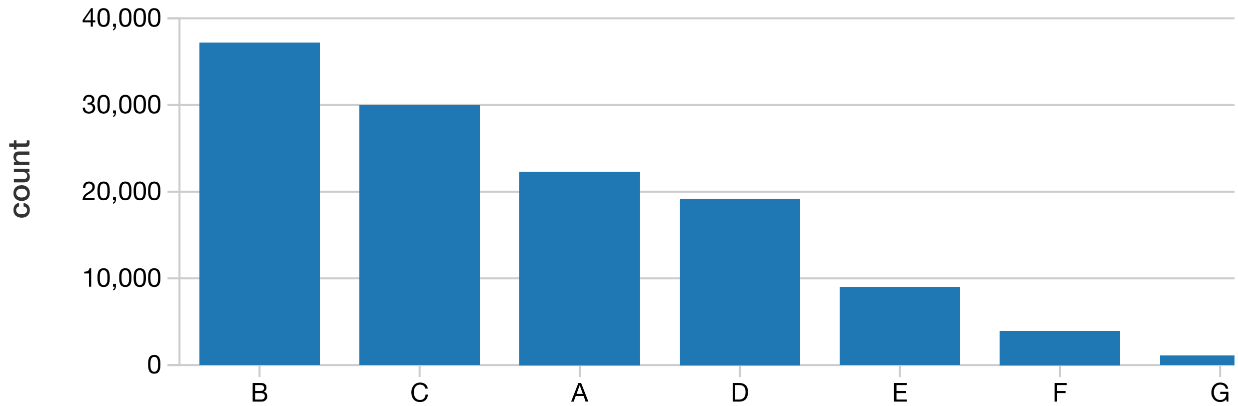
```
#import matplotlib as plt
#%matplotlib inline
#grade_distribution = loan_df.['grade'].value_counts()
grade_distribution = loan_df.groupBy('grade').count()
display(grade_distribution)
```



```
loan_df.createOrReplaceTempView('loan_tab')
```

Over half the loan grades are 'B or 'C'

```
display(spark.sql("select grade, count(*) as count from loan_tab group by grade
order by count(*) desc"))
```



Only a Small percentage of loanees own a home

```
#display(loan_df.groupBy('home_ownership').count().sort(desc("count")))
display(loan_df.groupBy('home_ownership').count().orderBy("count",
ascending=False))
```

```
from pyspark.sql import functions as f
#display(loan_df.groupBy('home_ownership').agg(f.count('*').alias('total')).sor
t(desc('total')))
display(loan_df.groupBy('home_ownership').agg(f.count('*').alias('total')).orde
rBy('total', ascending=False))
```

home_ownership
MORTGAGE
RENT
OWN
OTHER

Exploring Target Column 'Bad Loans'

Remapping it to be +1 and -1 as it is more intuitive.

- **+1 -> safe**
- **-1 -> risky (bad loan)**

```
from pyspark.sql.types import IntegerType
loan_df = loan_df.withColumn('bad_loans_int',
loan_df.bad_loans.cast(IntegerType()))
```

```
display(loan_df)
```

id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installmen
1077501	1296599	5000	5000	4975	36 months	10.65	162.87

1077430	1314167	2500	2500	2500	60 months	15.27	59.83

Showing the first 1000 rows.

```
from pyspark.sql.functions import udf, col
from pyspark.sql.types import IntegerType

fn = udf(lambda x: +1 if x==0 else -1, IntegerType())

loan_df = loan_df.withColumn('safe_loans', fn(col('bad_loans_int')))
```

```
display(loan_df)
```

id	member_id	loan_amnt	funded_amnt	funded_amnt_inv	term	int_rate	installmen
1077501	1296599	5000	5000	4975	36 months	10.65	162.87

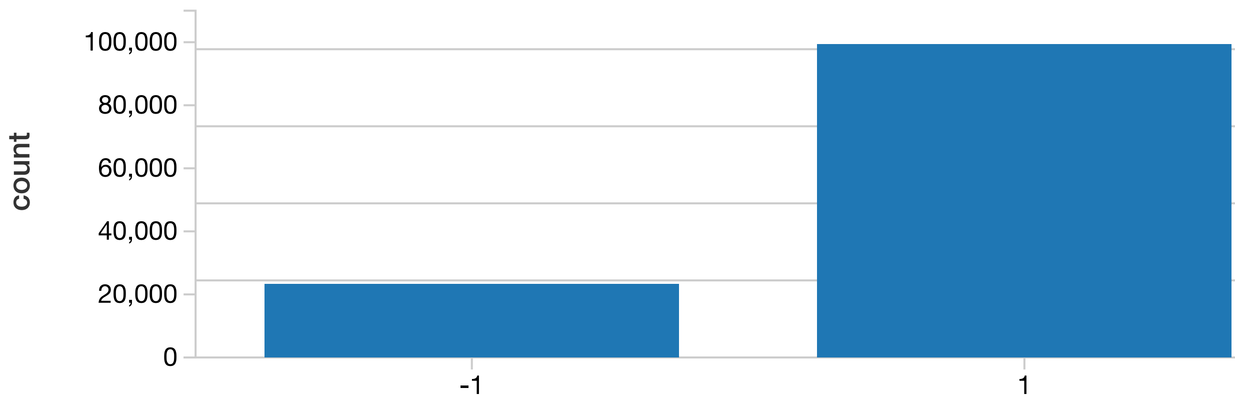
Showing the first 1000 rows.

```
display(loan_df.groupBy('safe_loans').count())
```

safe_loans
-1
1

Majority are safe loans

```
from pyspark.sql import functions as f
display(loan_df.groupBy('safe_loans').count())
```



Data is disproportionately full of safe loans. We can undersample the larger class until distribution is close to half and half

Analyzing data with a subset of the features

```
features = ['grade',                     # grade of the loan
            'sub_grade',                 # sub-grade of the loan
            'short_emp',                 # one year or less of employment
            'emp_length_num',            # number of years of employment
            'home_ownership',            # home_ownership status: own, mortgage
or rent
            'dti',                       # debt to income ratio
            'purpose',                   # the purpose of the loan
            'term',                      # the term of the loan
            'last_delinq_none',          # has borrower had a delinquincy
            'last_major_derog_none',     # has borrower had 90 day or worse
rating
            'revol_util',                # percent of available credit being
used
            'total_rec_late_fee'        # total late fees received to day'
           ]
```

```
target = 'safe_loans' # prediction target (y) (+1 means safe, -1 is risky)
```

```
loan_subset_df = loan_df.select(features + [target])
```

```
display(loan_subset_df)
```

grade	sub_grade	short_emp	emp_length_num	home_ownership	dti	purpose
B	B2	0	11	RENT	27.65	credit_card
C	C4	1	1	RENT	1	car
C	C5	0	11	RENT	8.72	small_business
C	C1	0	11	RENT	20	other
A	A4	0	4	RENT	11.2	wedding

Showing the first 1000 rows.

```
safe_loans_df = loan_subset_df[loan_subset_df[target] == +1]
display(safe_loans_df)
```

grade	sub_grade	short_emp	emp_length_num	home_ownership	dti	purpose
B	B2	0	11	RENT	27.65	credit_card
C	C5	0	11	RENT	8.72	small_business
C	C1	0	11	RENT	20	other
A	A4	0	4	RENT	11.2	wedding
E	E1	0	10	RENT	5.35	car

Showing the first 1000 rows.

```
risky_loans_df = loan_subset_df[loan_subset_df[target] == -1]
display(risky_loans_df)
```

grade	sub_grade	short_emp	emp_length_num	home_ownership	dti	purpose
C	C4	1	1	RENT	1	car
F	F2	0	5	OWN	5.55	small_business
B	B5	1	1	RENT	18.08	other
C	C1	1	1	RENT	10.08	debt_consolidation
B	B2	0	4	RENT	7.06	other

Showing the first 1000 rows.

find the ratio of the sizes (risky loan to safe loan) and use this percentage to undersample the safe loans

```
safe_loan_count = safe_loans_df.count()
risky_loan_count = risky_loans_df.count()
print(safe_loan_count, risky_loan_count)
print('Percentage of safe loans: ' , safe_loans_df.count()/
float(loan_subset_df.count()))
print('Percentage of risky loans: ' , risky_loans_df.count()/
float(loan_subset_df.count()))
risky_to_safe_ratio = risky_loan_count/float(safe_loan_count)
print('Ratio of risky loans to safe loans: ', risky_to_safe_ratio)
```

```
(99303, 23304)
('Percentage of safe loans: ', 0.8099292862560865)
('Percentage of risky loans: ', 0.1900707137439135)
('Ratio of risky loans to safe loans: ', 0.23467568955620677)
```

```
safe_loans_sample_df = safe_loans_df.sample(False, risky_to_safe_ratio, 42)
```

```
safe_loans_sample_df.count()
```

```
Out[977]: 23010
```

```
redist_loan_df = risky_loans_df.unionAll(safe_loans_sample_df)
```

```
redist_loan_df.count()
```

```
Out[979]: 46314
```

```
print('Percentage of safe loans in redistributed data: ' ,
safe_loans_sample_df.count()/ float(redist_loan_df.count()))
print('Percentage of risky loans in redistributed data: ' ,
risky_loans_df.count()/ float(redist_loan_df.count()))
```

```
('Percentage of safe loans in redistributed data: ', 0.49682601373234875)
('Percentage of risky loans in redistributed data: ', 0.5031739862676512)
```

```
redist_loan_df.printSchema()
```

```
root
 |-- grade: string (nullable = true)
 |-- sub_grade: string (nullable = true)
 |-- short_emp: string (nullable = true)
 |-- emp_length_num: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- dti: string (nullable = true)
 |-- purpose: string (nullable = true)
 |-- term: string (nullable = true)
```

```
 |-- last_delinq_none: string (nullable = true)
 |-- last_major_derog_none: string (nullable = true)
 |-- revol_util: string (nullable = true)
 |-- total_rec_late_fee: string (nullable = true)
 |-- safe_loans: integer (nullable = true)
```

```
display(redist_loan_df.describe())
```

summary	grade	sub_grade	short_emp	emp_length_num	home_ownership	dti
count	46314	46314	46312	46300	46314	463
mean	null	null	0.4402540011663319	8.003019132934236	null	16.
stddev	null	null	36.881054938236616	124.88494861517432	null	7.5
min	A	A1	0	0	MORTGAGE	K
max	G	G5	Fully Paid	Fully Paid	RENT	we

```
redist_loan_df = redist_loan_df.fillna("0")
```

```
display(redist_loan_df.describe())
```

summary	grade	sub_grade	short_emp	emp_length_num	home_ownership	dt
count	46314	46314	46314	46314	46314	46
mean	null	null	0.44023498412561285	8.00059842929675	null	16
stddev	null	null	36.88025847607735	124.86613711895251	null	7.
min	A	A1	0	0	MORTGAGE	
max	G	G5	Fully Paid	Fully Paid	RENT	we

```
numericCols = ["short_emp", "emp_length_num", "dti", "last_delinq_none",
"last_major_derog_none", "revol_util", "total_rec_late_fee"]
categoricalCols = ["grade", "sub_grade", "home_ownership", "purpose", "term"]
for c in numericCols:
  display(redist_loan_df.where(col(c).isNull()))
#for c in categoricalCols:
#  display(redist_loan_df.where(col(c).isNull()))
```

OK

```
cols = redist_loan_df.columns
```

```
redist_loan_df.head(5)
```

```
Out[987]:
[Row(grade=u'C', sub_grade=u'C4', short_emp=u'1', emp_length_num=u'1', home_own
ership=u'RENT', dti=u'1', purpose=u'car', term=u' 60 months', last_delinq_none=
u'1', last_major_derog_none=u'1', revol_util=u'9.4', total_rec_late_fee=u'0', s
afe_loans=-1),
 Row(grade=u'F', sub_grade=u'F2', short_emp=u'0', emp_length_num=u'5', home_own
ership=u'OWN', dti=u'5.55', purpose=u'small_business', term=u' 60 months', last
_delinq_none=u'1', last_major_derog_none=u'1', revol_util=u'32.6', total_rec_la
te_fee=u'0', safe_loans=-1),
 Row(grade=u'B', sub_grade=u'B5', short_emp=u'1', emp_length_num=u'1', home_own
ership=u'RENT', dti=u'18.08', purpose=u'other', term=u' 60 months', last_delinq
_none=u'1', last_major_derog_none=u'1', revol_util=u'36.5', total_rec_late_fee=
u'0', safe_loans=-1),
 Row(grade=u'C', sub_grade=u'C1', short_emp=u'1', emp_length_num=u'1', home_own
ership=u'RENT', dti=u'10.08', purpose=u'debt_consolidation', term=u' 36 month
s', last_delinq_none=u'1', last_major_derog_none=u'1', revol_util=u'91.7', tota
l_rec_late_fee=u'0', safe_loans=-1),
 Row(grade=u'B', sub_grade=u'B2', short_emp=u'0', emp_length_num=u'4', home_own
ership=u'RENT', dti=u'7.06', purpose=u'other', term=u' 36 months', last_delinq_
none=u'1', last_major_derog_none=u'1', revol_util=u'55.5', total_rec_late_fee=
u'0', safe_loans=-1)]
```

Decision Tree Model

Converting numeric columns to numeric type from string

```
from pyspark.sql.functions import col  # for indicating a column using a string
in the line below
numericCols = ["short_emp", "emp_length_num", "dti", "last_delinq_none",
"last_major_derog_none", "revol_util", "total_rec_late_fee"]

exprs = [col(c).cast("double") if c in numericCols else c for c in
redist_loan_df.columns]
redist_loan_df = redist_loan_df.select(*exprs)
redist_loan_df.printSchema()
```

```
root
 |-- grade: string (nullable = false)
 |-- sub_grade: string (nullable = false)
 |-- short_emp: double (nullable = true)
 |-- emp_length_num: double (nullable = true)
 |-- home_ownership: string (nullable = false)
 |-- dti: double (nullable = true)
 |-- purpose: string (nullable = false)
 |-- term: string (nullable = false)
 |-- last_delinq_none: double (nullable = true)
 |-- last_major_derog_none: double (nullable = true)
 |-- revol_util: double (nullable = true)
 |-- total_rec_late_fee: double (nullable = true)
 |-- safe_loans: integer (nullable = true)
```

```
redist_loan_df = redist_loan_df.fillna(0)
```

```
numericCols = ["short_emp", "emp_length_num", "dti", "last_delinq_none",
"last_major_derog_none", "revol_util", "total_rec_late_fee"]
categoricalCols = ["grade", "sub_grade", "home_ownership", "purpose", "term"]
for c in numericCols:
  display(redist_loan_df.where(col(c).isNull()))
#for c in categoricalCols:
#  display(redist_loan_df.where(col(c).isNull()))
```

OK

```
display(redist_loan_df)
```

grade	sub_grade	short_emp	emp_length_num	home_ownership	dti	purpose
C	C4	1	1	RENT	1	car
F	F2	0	5	OWN	5.55	small_business

B	B5	1	1	RENT	18.08	other
C	C1	1	1	RENT	10.08	debt_consolidation

Showing the first 1000 rows.

```
display(redist_loan_df.describe())
```

summary	grade	sub_grade	short_emp	emp_length_num	home_ownership	dt
count	46314	46314	46314	46314	46314	46
mean	null	null	0.44011141339551757	7.995588770134302	null	16
stddev	null	null	36.87508272948457	124.82719747415113	null	7.(
min	A	A1	0.0	0.0	MORTGAGE	0.(
max	G	G5	5916.62	12000.0	RENT	39

Converting Categorical variables to numeric with One-Hot Encoding

more than 1 stages of feature transformations. Using a Pipeline to tie the stages together

```
###One-Hot Encoding
from pyspark.ml import Pipeline
from pyspark.ml.feature import OneHotEncoder, StringIndexer, VectorAssembler

categoricalColumns = ["grade", "sub_grade", "home_ownership", "purpose",
"term"]
stages = [] # stages in our Pipeline
for categoricalCol in categoricalColumns:
  # Category Indexing with StringIndexer
  stringIndexer = StringIndexer(inputCol=categoricalCol,
outputCol=categoricalCol+"Index")
  # Use OneHotEncoder to convert categorical variables into binary
SparseVectors
  encoder = OneHotEncoder(inputCol=categoricalCol+"Index",
outputCol=categoricalCol+"classVec")
  # Add stages.  These are not run here, but will run all at once later on.
  stages += [stringIndexer, encoder]


# Convert label into label indices using the StringIndexer
label_stringIdx = StringIndexer(inputCol = "safe_loans", outputCol = "label")
stages += [label_stringIdx]


# Transform all features into a vector using VectorAssembler
numericCols = ["short_emp", "emp_length_num", "dti", "last_delinq_none",
"last_major_derog_none", "revol_util", "total_rec_late_fee"]
assemblerInputs = map(lambda c: c + "classVec", categoricalColumns) +
numericCols
assembler = VectorAssembler(inputCols=assemblerInputs, outputCol="features")
stages += [assembler]


# Create a Pipeline.
pipeline = Pipeline(stages=stages)
# Run the feature transformations.
#  - fit() computes feature statistics as needed.
#  - transform() actually transforms the features.
pipelineModel = pipeline.fit(redist_loan_df)
redist_loan_df = pipelineModel.transform(redist_loan_df)

# Keep relevant columns
selectedcols = ["label", "features"] + cols
#selectedcols = ["label", "features"]
redist_loan_df = redist_loan_df.select(selectedcols)
display(redist_loan_df)
```

label	features	grade	sub_grade	short_emp	emp_length_num
0	▸ [0,271, [1,13,41,49,264,265,266,267,268,269], [1,1,1,1,1,1,1,1,1,9.4]]	C	C4	1	1
0	▸ [0,271,[5,32,42,47,265,266,267,268,269], [1,1,1,1,5,5.55,1,1,32.6]]	F	F2	0	5
0	▸ [0,271, [0,9,41,45,264,265,266,267,268,269], [1,1,1,1,1,1,18.08,1,1,36.5]]	B	B5	1	1
0	▸ [0,271, [1,8,41,43,263,264,265,266,267,268,269],	C	C1	1	1

Showing the first 1000 rows.

Split data into training and validation sets

```
(train_data, validation_data) = redist_loan_df.randomSplit([0.7, 0.3], seed =
100)
print train_data.count()
print validation_data.count()
```

```
32423
13891
```

Using Decision Tree Model

```
from pyspark.ml.classification import DecisionTreeClassifier

# Create initial Decision Tree Model
dtree = DecisionTreeClassifier(labelCol="label", featuresCol="features",
maxDepth=3)

# Train model with Training Data
dtree = dtree.fit(train_data)


print "numNodes = ", dtree.numNodes
print "depth = ", dtree.depth
```

```
numNodes =  15
depth =  3
```

```
predictions = dtree.transform(validation_data)
```

```
predictions.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- grade: string (nullable = false)
 |-- sub_grade: string (nullable = false)
 |-- short_emp: double (nullable = false)
 |-- emp_length_num: double (nullable = false)
 |-- home_ownership: string (nullable = false)
 |-- dti: double (nullable = false)
 |-- purpose: string (nullable = false)
 |-- term: string (nullable = false)
 |-- last_delinq_none: double (nullable = false)
 |-- last_major_derog_none: double (nullable = false)
 |-- revol_util: double (nullable = false)
 |-- total_rec_late_fee: double (nullable = false)
 |-- safe_loans: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

```
selected = predictions.select("label", "prediction", "probability", "grade",
"home_ownership")
display(selected)
```

label	prediction	probability
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	0	▸ [1,2,[],[0.8292079207920792,0.1707920792079208]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]
0	1	▸ [1,2,[],[0.4025943942552699,0.5974056057447301]]

Showing the first 1000 rows.

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)
```

```
Out[1003]: 0.41768112249551803
```

```
evaluator.getMetricName()
```

```
Out[1004]: 'areaUnderROC'
```

Creating more complex Decision Tree Model with max depth 10

```
# Create initial Decision Tree Model
dtree = DecisionTreeClassifier(labelCol="label", featuresCol="features",
maxDepth=10)

# Train model with Training Data
dtree = dtree.fit(train_data)
```

```
print "numNodes = ", dtree.numNodes
print "depth = ", dtree.depth
```

```
numNodes =  903
depth =  10
```

```
predictions = dtree.transform(validation_data)
```

```
predictions.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- grade: string (nullable = false)
 |-- sub_grade: string (nullable = false)
 |-- short_emp: double (nullable = false)
 |-- emp_length_num: double (nullable = false)
 |-- home_ownership: string (nullable = false)
 |-- dti: double (nullable = false)
 |-- purpose: string (nullable = false)
 |-- term: string (nullable = false)
 |-- last_delinq_none: double (nullable = false)
 |-- last_major_derog_none: double (nullable = false)
```

```
 |-- revol_util: double (nullable = false)
 |-- total_rec_late_fee: double (nullable = false)
 |-- safe_loans: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

```
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Evaluate model
evaluator = BinaryClassificationEvaluator(rawPredictionCol="rawPrediction")
evaluator.evaluate(predictions)
```

```
Out[1009]: 0.6034347951553064
```

Using Random Forest Model that uses an ensemble of trees to improve accuracy

```
from pyspark.ml.classification import RandomForestClassifier

# Create an initial RandomForest model.
rf = RandomForestClassifier(labelCol="label", featuresCol="features")

# Train model with Training Data
rfModel = rf.fit(train_data)
```

```
predictions = rfModel.transform(validation_data)
```

```
predictions.printSchema()
```

```
root
 |-- label: double (nullable = false)
 |-- features: vector (nullable = true)
 |-- grade: string (nullable = false)
 |-- sub_grade: string (nullable = false)
 |-- short_emp: double (nullable = false)
 |-- emp_length_num: double (nullable = false)
 |-- home_ownership: string (nullable = false)
 |-- dti: double (nullable = false)
 |-- purpose: string (nullable = false)
 |-- term: string (nullable = false)
 |-- last_delinq_none: double (nullable = false)
```

```
 |-- last_major_derog_none: double (nullable = false)
 |-- revol_util: double (nullable = false)
 |-- total_rec_late_fee: double (nullable = false)
 |-- safe_loans: integer (nullable = true)
 |-- rawPrediction: vector (nullable = true)
 |-- probability: vector (nullable = true)
 |-- prediction: double (nullable = false)
```

```
selected = predictions.select("label", "prediction", "probability", "grade",
"home_ownership")
display(selected)
```

label	prediction	probability
0	1	▸ [1,2,[],[0.4747048047164014,0.5252951952835986]]
0	0	▸ [1,2,[],[0.5346007516441156,0.4653992483558843]]
0	0	▸ [1,2,[],[0.5188273997824944,0.4811726002175056]]
0	0	▸ [1,2,[],[0.5346201430789995,0.4653798569210004]]
0	0	▸ [1,2,[],[0.5600069951142711,0.4399930048857289]]
0	1	▸ [1,2,[],[0.4660683834818634,0.5339316165181366]]
0	1	▸ [1,2,[],[0.4783955919397572,0.5216044080602428]]
0	1	▸ [1,2,[],[0.46808074538202715,0.5319192546179728]]
0	1	▸ [1,2,[],[0.48877824348942056,0.5112217565105793]]

Showing the first 1000 rows.

```
Out[1014]: 0.6865432551258851
```