CISC-5950 Project Notebook

May 1, 2018

In [1]:
```
import findspark
findspark.init()

from pyspark import SparkConf
from pyspark.sql import SparkSession
from pyspark.sql import SQLContext
from pyspark.sql import functions as F
from pyspark.sql.functions import isnan, when, count, col, year, quarter, lit, to_date
from pyspark.sql.types import DateType, TimestampType
from pyspark import SparkContext
from pyspark import SparkConf
from pyspark.ml.feature import Imputer
from pyspark.sql import DataFrameStatFunctions as statFunc
from pyspark.ml.feature import StringIndexer
from pyspark.ml.feature import VectorAssembler
from pyspark.ml.feature import IndexToString
from pyspark.mllib.tree import RandomForest, RandomForestModel
from pyspark.ml.classification import GBTClassifier
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.classification import NaiveBayes
from pyspark.ml.classification import LinearSVC
from pyspark.ml.evaluation import BinaryClassificationEvaluator
from pyspark.mllib.evaluation import BinaryClassificationMetrics
from pyspark.ml.evaluation import MulticlassClassificationEvaluator
from pyspark.ml.feature import PCA


from pyspark.ml.classification import LogisticRegression
from pyspark.mllib.classification import LogisticRegressionWithLBFGS

from pyspark.ml import Pipeline
from pyspark.ml.evaluation import MulticlassClassificationEvaluator

from pyspark.mllib.classification import SVMWithSGD, SVMModel
from pyspark.mllib.regression import LabeledPoint

from pyspark.ml.tuning import CrossValidator, ParamGridBuilder
```

```
from sklearn.metrics import roc_curve, auc


#%matplotlib inline

import datetime
import numpy as np
import pandas as pd
from pandas import DataFrame as df
import matplotlib
# Force matplotlib to not use any Xwindows backend.
matplotlib.use('Agg')
import matplotlib.pyplot as plt
import seaborn as sns
sns.set(color_codes=True)
from scipy import stats

import plotly.plotly as py

import plotly.graph_objs as go
from plotly.offline import init_notebook_mode,iplot
init_notebook_mode(connected=True)

import os
memory = '4g'
pyspark_submit_args = ' --driver-memory ' + memory + ' pyspark-shell'
os.environ["PYSPARK_SUBMIT_ARGS"] = pyspark_submit_args

#sc = SparkContext()
SparkContext.setSystemProperty('spark.executor.memory', '4g')
SparkContext.setSystemProperty('spark.driver.memory', '4g')

spark_conf = SparkConf().setAll(pairs = [('spark.executor.memory', '4g'), ('spark.execu

spark = SparkSession.builder.master("local[*]").config(conf = spark_conf).appName("Lend
sqlContext = SQLContext(spark)

spark.sparkContext.setLogLevel('ERROR')

import warnings
warnings.filterwarnings('ignore')
```

```
C:\Anaconda3\lib\site-packages\ipykernel_launcher.py:50: UserWarning:
This call to matplotlib.use() has no effect because the backend has already
been chosen; matplotlib.use() must be called *before* pylab, matplotlib.pyplot,
or matplotlib.backends is imported for the first time.
```

```
The backend was *originally* set to 'module://ipykernel.pylab.backend_inline' by the following
  File "C:\Anaconda3\lib\runpy.py", line 193, in _run_module_as_main
    "__main__", mod_spec)
  File "C:\Anaconda3\lib\runpy.py", line 85, in _run_code
    exec(code, run_globals)
  File "C:\Anaconda3\lib\site-packages\ipykernel_launcher.py", line 16, in <module>
    app.launch_new_instance()
  File "C:\Anaconda3\lib\site-packages\traitlets\config\application.py", line 658, in launch_in
    app.start()
  File "C:\Anaconda3\lib\site-packages\ipykernel\kernelapp.py", line 477, in start
    ioloop.IOLoop.instance().start()
  File "C:\Anaconda3\lib\site-packages\zmq\eventloop\ioloop.py", line 177, in start
    super(ZMQIOLoop, self).start()
  File "C:\Anaconda3\lib\site-packages\tornado\ioloop.py", line 888, in start
    handler_func(fd_obj, events)
  File "C:\Anaconda3\lib\site-packages\tornado\stack_context.py", line 277, in null_wrapper
    return fn(*args, **kwargs)
  File "C:\Anaconda3\lib\site-packages\zmq\eventloop\zmqstream.py", line 440, in _handle_events
    self._handle_recv()
  File "C:\Anaconda3\lib\site-packages\zmq\eventloop\zmqstream.py", line 472, in _handle_recv
    self._run_callback(callback, msg)
  File "C:\Anaconda3\lib\site-packages\zmq\eventloop\zmqstream.py", line 414, in _run_callback
    callback(*args, **kwargs)
  File "C:\Anaconda3\lib\site-packages\tornado\stack_context.py", line 277, in null_wrapper
    return fn(*args, **kwargs)
  File "C:\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 283, in dispatcher
    return self.dispatch_shell(stream, msg)
  File "C:\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 235, in dispatch_shell
    handler(stream, idents, msg)
  File "C:\Anaconda3\lib\site-packages\ipykernel\kernelbase.py", line 399, in execute_request
    user_expressions, allow_stdin)
  File "C:\Anaconda3\lib\site-packages\ipykernel\ipkernel.py", line 196, in do_execute
    res = shell.run_cell(code, store_history=store_history, silent=silent)
  File "C:\Anaconda3\lib\site-packages\ipykernel\zmqshell.py", line 533, in run_cell
    return super(ZMQInteractiveShell, self).run_cell(*args, **kwargs)
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2698, in run_cel
    interactivity=interactivity, compiler=compiler, result=result)
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2802, in run_ast
    if self.run_code(code, result):
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2862, in run_cod
    exec(code_obj, self.user_global_ns, self.user_ns)
  File "<ipython-input-1-321d3b4cefa5>", line 42, in <module>
    get_ipython().magic('matplotlib inline')
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2146, in magic
    return self.run_line_magic(magic_name, magic_arg_s)
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2067, in run_lin
    result = fn(*args,**kwargs)
  File "<decorator-gen-108>", line 2, in matplotlib
```

```
  File "C:\Anaconda3\lib\site-packages\IPython\core\magic.py", line 187, in <lambda>
    call = lambda f, *a, **k: f(*a, **k)
  File "C:\Anaconda3\lib\site-packages\IPython\core\magics\pylab.py", line 99, in matplotlib
    gui, backend = self.shell.enable_matplotlib(args.gui)
  File "C:\Anaconda3\lib\site-packages\IPython\core\interactiveshell.py", line 2930, in enable_
    pt.activate_matplotlib(backend)
  File "C:\Anaconda3\lib\site-packages\IPython\core\pylabtools.py", line 307, in activate_matpl
    matplotlib.pyplot.switch_backend(backend)
  File "C:\Anaconda3\lib\site-packages\matplotlib\pyplot.py", line 229, in switch_backend
    matplotlib.use(newbackend, warn=False, force=True)
  File "C:\Anaconda3\lib\site-packages\matplotlib\__init__.py", line 1305, in use
    reload(sys.modules['matplotlib.backends'])
  File "C:\Anaconda3\lib\importlib\__init__.py", line 166, in reload
    _bootstrap._exec(spec, module)
  File "C:\Anaconda3\lib\site-packages\matplotlib\backends\__init__.py", line 14, in <module>
    line for line in traceback.format_stack()
```

0.0.1 Load Data to Spark DataFrame

```
In [2]: loanDF = spark.read.csv("loan.csv", header=True, mode="DROPMALFORMED")
        #loanDF = spark.read.csv("loan.csv", header=True, mode="DROPMALFORMED", inferSchema =T

        #loanDFRows = loanDF.count()

        loanDF.printSchema()

        # Loading it as pandasDF also, only for the comparison/testing.
        loanDF_Pandas = pd.read_csv("loan.csv", low_memory=False)
```

```
root
 |-- id: string (nullable = true)
 |-- member_id: string (nullable = true)
 |-- loan_amnt: string (nullable = true)
 |-- funded_amnt: string (nullable = true)
 |-- funded_amnt_inv: string (nullable = true)
 |-- term: string (nullable = true)
 |-- int_rate: string (nullable = true)
 |-- installment: string (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: string (nullable = true)
 |-- verification_status: 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)
|-- purpose: string (nullable = true)
|-- title: string (nullable = true)
|-- zip_code: string (nullable = true)
|-- addr_state: string (nullable = true)
|-- dti: string (nullable = true)
|-- delinq_2yrs: string (nullable = true)
|-- earliest_cr_line: string (nullable = true)
|-- inq_last_6mths: string (nullable = true)
|-- mths_since_last_delinq: string (nullable = true)
|-- mths_since_last_record: string (nullable = true)
|-- open_acc: string (nullable = true)
|-- pub_rec: string (nullable = true)
|-- revol_bal: string (nullable = true)
|-- revol_util: string (nullable = true)
|-- total_acc: string (nullable = true)
|-- initial_list_status: string (nullable = true)
|-- out_prncp: string (nullable = true)
|-- out_prncp_inv: string (nullable = true)
|-- total_pymnt: string (nullable = true)
|-- total_pymnt_inv: string (nullable = true)
|-- total_rec_prncp: string (nullable = true)
|-- total_rec_int: string (nullable = true)
|-- total_rec_late_fee: string (nullable = true)
|-- recoveries: string (nullable = true)
|-- collection_recovery_fee: string (nullable = true)
|-- last_pymnt_d: string (nullable = true)
|-- last_pymnt_amnt: string (nullable = true)
|-- next_pymnt_d: string (nullable = true)
|-- last_credit_pull_d: string (nullable = true)
|-- collections_12_mths_ex_med: string (nullable = true)
|-- mths_since_last_major_derog: string (nullable = true)
|-- policy_code: string (nullable = true)
|-- application_type: string (nullable = true)
|-- annual_inc_joint: string (nullable = true)
|-- dti_joint: string (nullable = true)
|-- verification_status_joint: string (nullable = true)
|-- acc_now_delinq: string (nullable = true)
|-- tot_coll_amt: string (nullable = true)
|-- tot_cur_bal: string (nullable = true)
|-- open_acc_6m: string (nullable = true)
|-- open_il_6m: string (nullable = true)
|-- open_il_12m: string (nullable = true)
|-- open_il_24m: string (nullable = true)
```

```
 |-- mths_since_rcnt_il: string (nullable = true)
 |-- total_bal_il: string (nullable = true)
 |-- il_util: string (nullable = true)
 |-- open_rv_12m: string (nullable = true)
 |-- open_rv_24m: string (nullable = true)
 |-- max_bal_bc: string (nullable = true)
 |-- all_util: string (nullable = true)
 |-- total_rev_hi_lim: string (nullable = true)
 |-- inq_fi: string (nullable = true)
 |-- total_cu_tl: string (nullable = true)
 |-- inq_last_12m: string (nullable = true)
```

0.0.2 Find the Feature columns which has more than 50% empty data

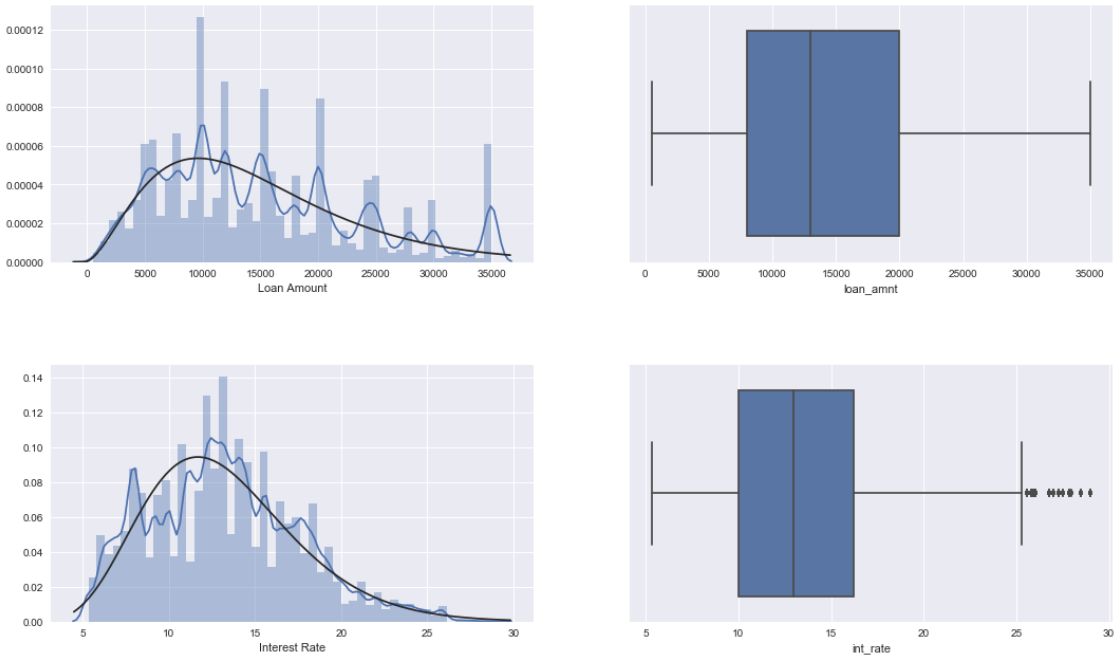
```
In [3]: # find list of columns which has more than 50% of data missing.
        def findMissingValueCols(df):
            #df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.column
            missingValueColumns = []
            for column in df.columns:
                nullRows = df.where(col(column).isNull()).count()
                print(column, "--", nullRows)
                if nullRows > loanDFRows*0.5 : # i.e. if ALL values are NULL
                    missingValueColumns.append(column)
            return missingValueColumns

        # columns names which has more than 50% data missing
        #missingValueColList = findMissingValueCols(loanDF)

        #print(missingValueColList)
```

Analyzing Loan amount and Interest rates

- Frequency distribution of loan amount with gamma distribution.
- Five number summary distribution of loan amount.
- Frequency distribution of interest rates with gamma distribution.
- Five number summary distribution of interest rates.

```
In [4]: loanDF = loanDF.withColumn("loan_amnt",loanDF["loan_amnt"].cast('float'))
        loanDF = loanDF.withColumn("int_rate",loanDF["int_rate"].cast('float'))

        tmp = loanDF.select("loan_amnt", "int_rate").toPandas()
        #loanDF.printSchema()

        fig, ax =plt.subplots(2,2, figsize=(18,12))
        plt.subplots_adjust(hspace = 0.4, top = 0.8)
```

```
# Loan amount distribution plots
sns.distplot(tmp.loan_amnt, fit=stats.gamma, axlabel="Loan Amount", label="Loan Amount
sns.boxplot(x=tmp.loan_amnt, ax=ax[0][1])

# Interest rates distribution plots
sns.distplot(tmp.int_rate, fit=stats.gamma, axlabel="Interest Rate", label="Interest Fi
sns.boxplot(x=tmp.int_rate, ax=ax[1][1])

fig.show()

fig.savefig("LoanDistribution.pdf")
```



0.0.3 Converting the issue_d as DateType.

In order to group by quarterly, adding a new column as "issue_year" which parse the date field and get year and quarter details.

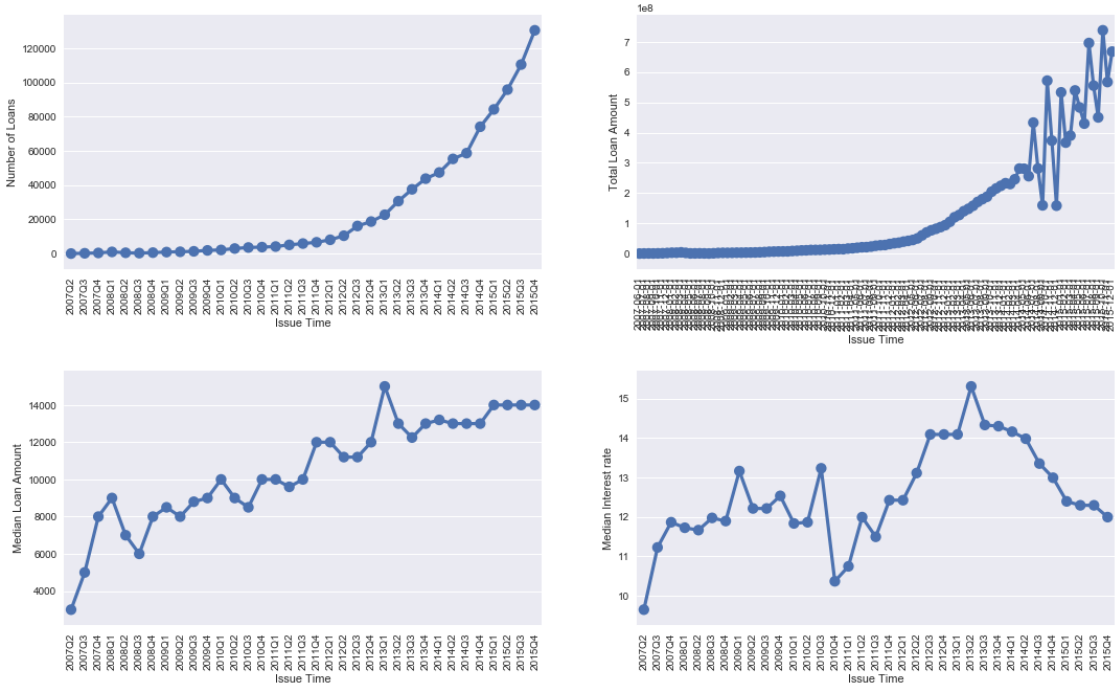
In [5]:
```
# Converting issue_d to datetype for the aggregation, Creating a new field for monthly
#loanDF.printSchema
#loanDF.select("issue_d").show(10)

loanDF = loanDF.withColumn("issue_d_dateType", to_date("issue_d", "MM/dd/yyyy"))

loanDF = loanDF.withColumn("issue_year", concat(year("issue_d_dateType"), lit("Q"), qua

#loanDF.select("issue_d", "issue_d_dateType", "issue_year").show(10)
```

```
#loanDF.printSchema

#loanDF.select("issue_year").distinct().show()
```

Analyzing Loans Interest rates over time

- Number of loan's growth over time
- Total loan book value growth over time
- Customers loan requirements over time (Median loan amount)
- Median interest rates over time

In [6]:
```
loanDF.registerTempTable("loanDFTable")

fig, ax =plt.subplots(2,2, figsize=(18,12))
plt.subplots_adjust(hspace = 0.4, top = 0.8)

# Number of loan over year
loansOverTime = loanDF.sort("issue_year").groupBy("issue_year").count().toPandas()
#print(loansOverTime)
loansOverTime.columns = ["Issue Time", "Number of Loans"]
s0=sns.pointplot(x=loansOverTime["Issue Time"], y=loansOverTime["Number of Loans"], ax=
s0.set_xticklabels(s0.get_xticklabels(),rotation=90)

# loan Amount over year
totalloansOverTime = loanDF.sort("issue_d_dateType").groupBy("issue_d_dateType").sum('
#print(totalloansOverTime)
totalloansOverTime.columns = ["Issue Time", "Total Loan Amount"]
s1 = sns.pointplot(x=totalloansOverTime['Issue Time'], y=totalloansOverTime["Total Loa
s1.set_xticklabels(s1.get_xticklabels(),rotation=90)


# Median Loan Amount, grouped by issue year
medianloanAmtOverTime = sqlContext.sql("select issue_year, percentile_approx(loan_amnt
#print(medianloanAmtOverTime)
medianloanAmtOverTime.columns = ["Issue Time", "Median Loan Amount"]
s2 = sns.pointplot(x=medianloanAmtOverTime['Issue Time'], y=medianloanAmtOverTime["Medi
s2.set_xticklabels(s2.get_xticklabels(),rotation=90)

# Median Interest rate, grouped by issue year
medianIntrateOverTime = sqlContext.sql("select issue_year, percentile_approx(int_rate,
#print(medianIntrateOverTime)
medianIntrateOverTime.columns = ["Issue Time", "Median Interest rate"]
s3 = sns.pointplot(x=medianIntrateOverTime['Issue Time'], y=medianIntrateOverTime["Medi
s3.set_xticklabels(s3.get_xticklabels(),rotation=90)

fig.show()

fig.savefig("LoanPatternOverTime.pdf")
```



0.0.4 Loan Counts group by loan_status

In [7]:
```
#loanDF.registerTempTable("loanDFTable")

loanDF.groupBy('loan_status').count().sort("count", ascending=False).show()
#print(loanDF.loan_status.value_counts())

#print((loanDF[loanDF.loan_status.isin(["Default"])]).verification_status.value_counts

#loanDF.loan_status.describe()
```

```
+--------------------+------+
|         loan_status| count|
+--------------------+------+
|             Current|601776|
|          Fully Paid|207533|
|         Charged Off| 45215|
|  Late (31-120 days)| 11591|
|              Issued|  8460|
|     In Grace Period|  6253|
|   Late (16-30 days)|  2357|
|Does not meet the...|  1969|
|             Default|  1219|
|Does not meet the...|   751|
+--------------------+------+
```

0.0.5 Default Loan Count group by varification_status

In [8]:
```
#DataFrame newdf = df.select(df.col("*")).filter("somecol <= 10")

        loanDF.filter("loan_status == 'Default'").groupBy('verification_status').count().show()
```

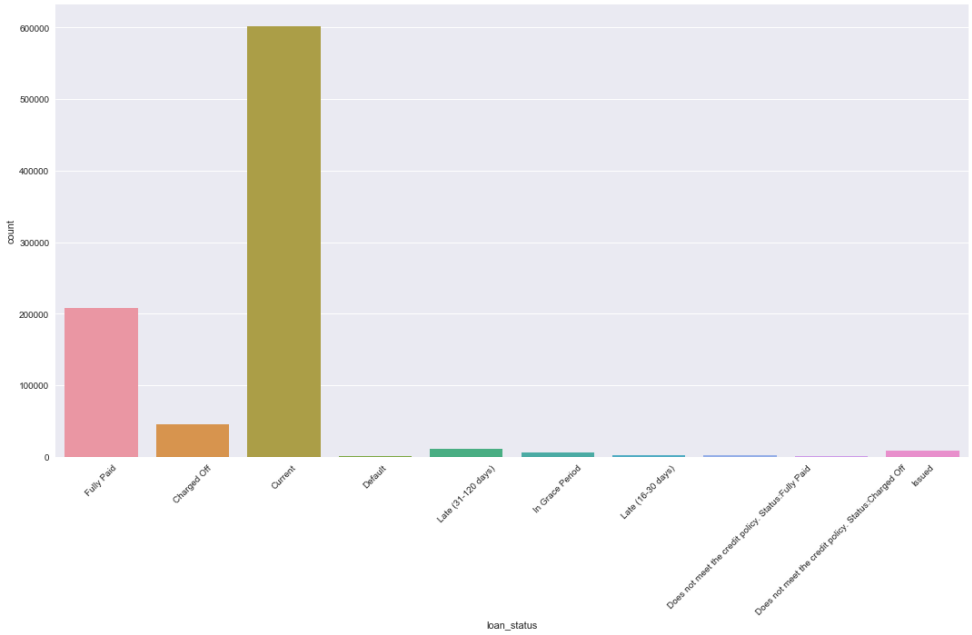
```
+-------------------+-----+
|verification_status|count|
+-------------------+-----+
|           Verified|  479|
|    Source Verified|  462|
|       Not Verified|  278|
+-------------------+-----+
```

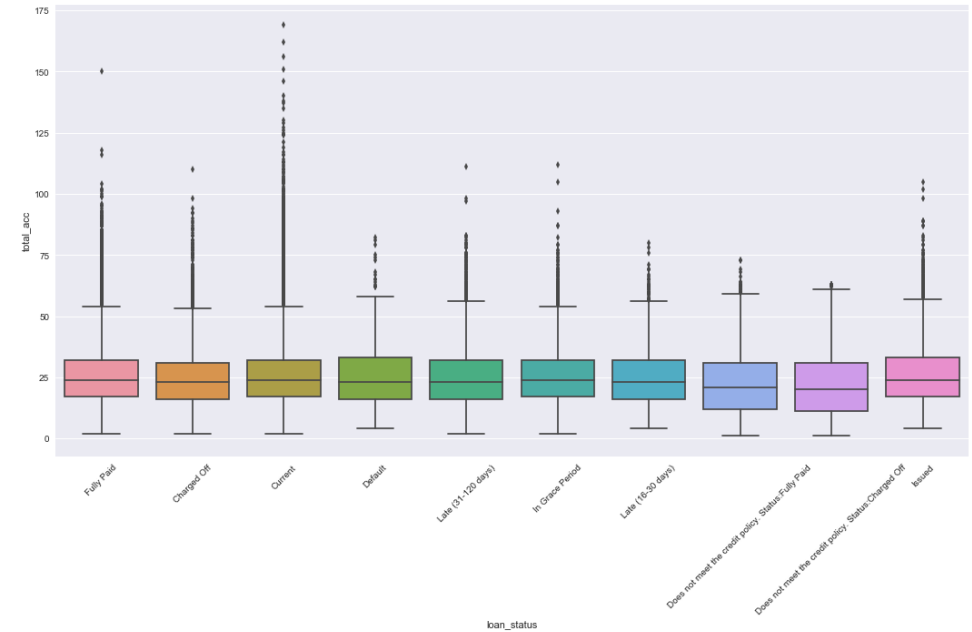
0.0.6 Alalyzing Loans over loan status.

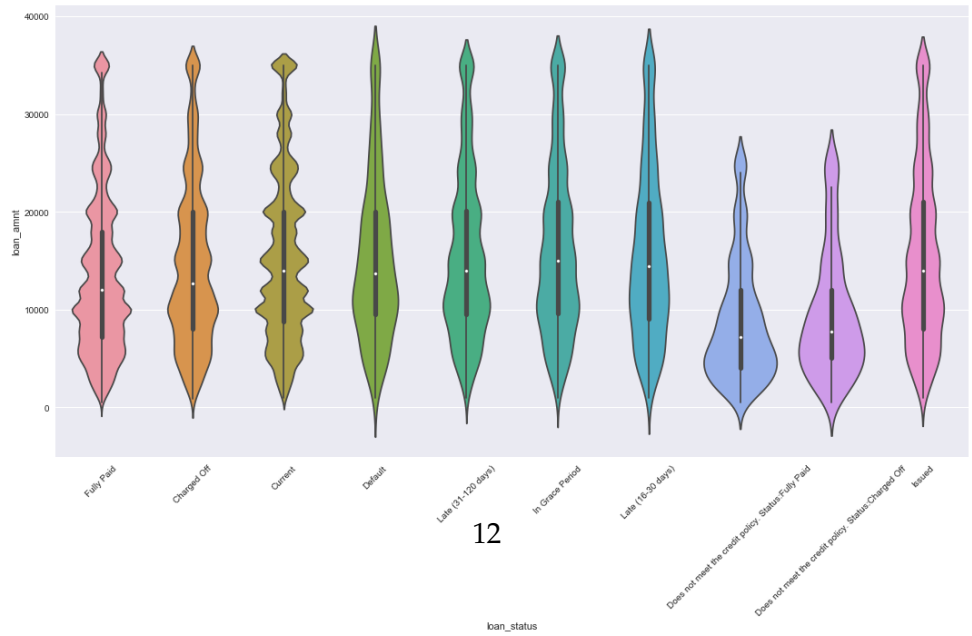
- Number of loans over for each loan status
- Distribution of total accounts for each status
- Distribution of loan amount with the probability density for each loan value over loan status

In [9]:
```
loanDF = loanDF.withColumn("total_acc",loanDF["total_acc"].cast('float'))

        tmp = loanDF.select("loan_status", "total_acc", "loan_amnt").toPandas()

        fig, ax =plt.subplots(3,1,figsize=(18,30))
        plt.subplots_adjust(hspace = 0.4, top = 1.0)

        s0 = sns.countplot(x="loan_status", data=tmp,ax=ax[0])
        s0.set_xticklabels(s0.get_xticklabels(),rotation=45)

        s1 = sns.boxplot(x="loan_status", y="total_acc", data=tmp,ax=ax[1])
        s1.set_xticklabels(s1.get_xticklabels(),rotation=45)

        # loan Amount over loan status
        #totalloansOverStatus = pd.DataFrame(loanDF.groupby(loanDF.loan_status)['loan_amnt'].s
        #totalloansOverStatus = loanDF.groupBy('loan_status').sum('loan_amnt').toPandas()
        #totalloansOverStatus.columns = ["Total Loan Amount"]
        #totalloansOverStatus.index.names = ["Loan Status"]

        s2 = sns.violinplot(x="loan_status", y="loan_amnt", data=tmp,ax=ax[2] , estimator=sum)
        #s2 = sns.violinplot(x=totalloansOverStatus.index, y=totalloansOverStatus["Total Loan
        s2.set_xticklabels(s2.get_xticklabels(),rotation=45)

        plt.show()
```

```
fig.savefig("LoanOveroanStatus.pdf")
```

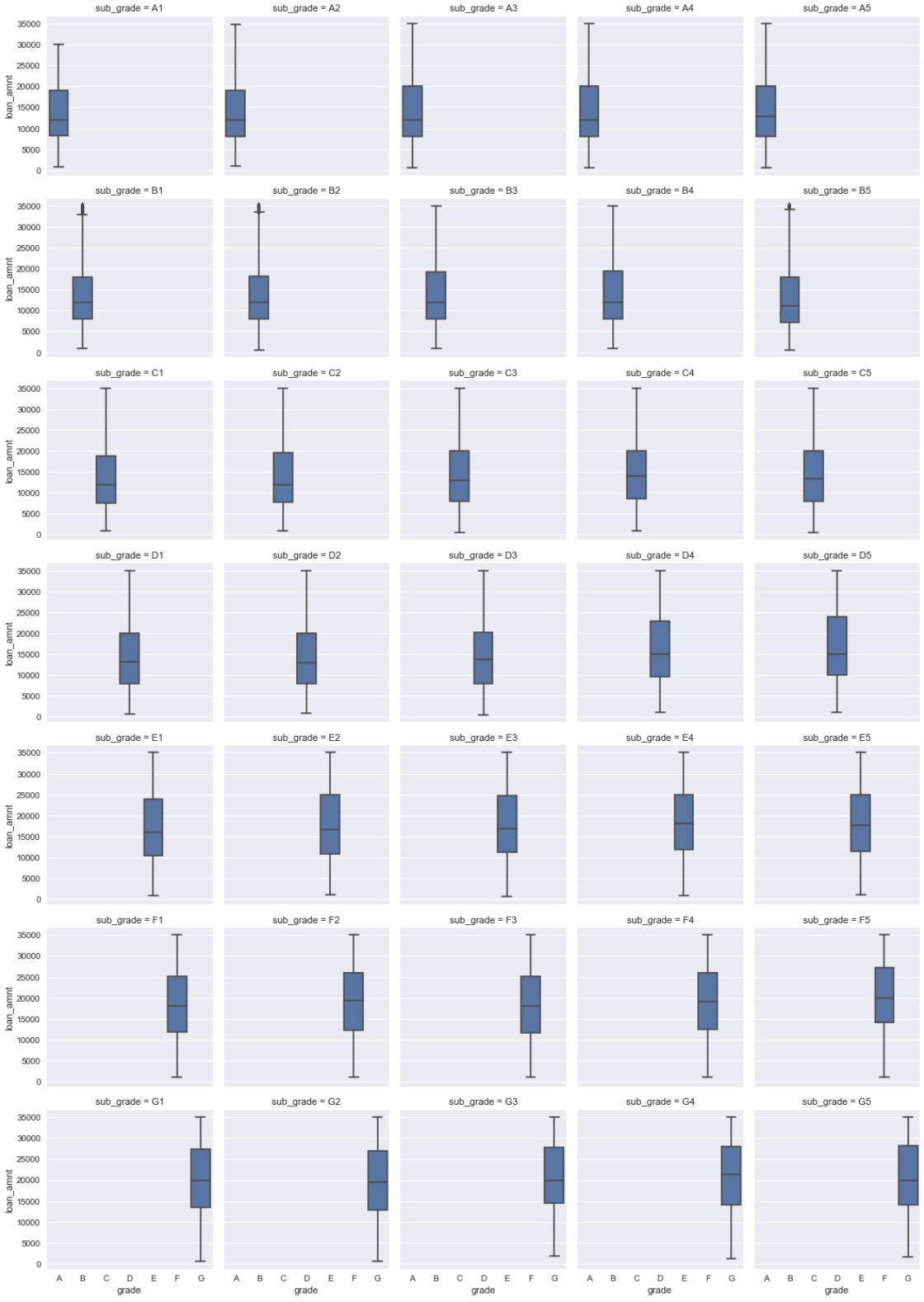

count
600000
500000
400000
300000
200000
100000
0
Fully Paid
Charged Off
Current
Default
Late (31-120 days)
In Grace Period
Late (16-30 days)
Does not meet the credit policy. Status:Fully Paid
Does not meet the credit policy. Status:Charged Off
Issued
loan_status


total_acc
175
150
125
100
75
50
25
0
Fully Paid
Charged Off
Current
Default
Late (31-120 days)
In Grace Period
Late (16-30 days)
Does not meet the credit policy. Status:Fully Paid
Does not meet the credit policy. Status:Charged Off
Issued
loan_status


loan_amnt
40000
30000
20000
10000
0
Fully Paid
Charged Off
Current
Default
Late (31-120 days)
In Grace Period
Late (16-30 days)
Does not meet the credit policy. Status:Fully Paid
Does not meet the credit policy. Status:Charged Off
Issued
loan_status

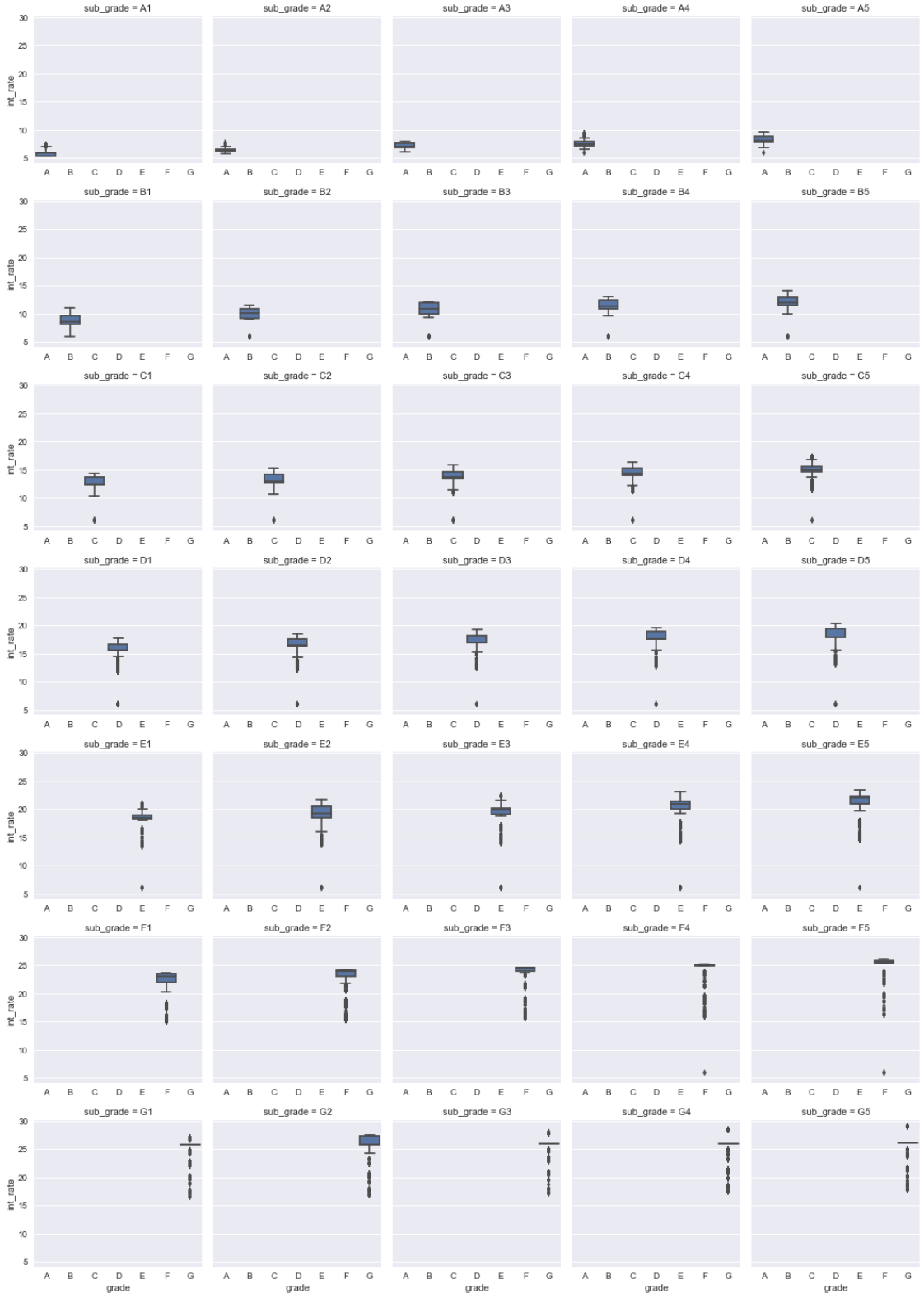
Analysing loan amount distribution for each grade, factored over sub grade. This shows a linear relationship between loan amount and customer credit ratings, Notice here that requested loan amount is slightly higher for the low rating customers.

In [10]:
```
#sns.boxplot(x="grade", y="loan_amnt", data=loanDF)
tmpDF = loanDF.select("sub_grade", "grade", "loan_amnt", "int_rate").toPandas()
tmpDF['grade'] = tmpDF['grade'].astype('category')
tmpDF['sub_grade'] = tmpDF['sub_grade'].astype('category')
#tmpDF['loan_amnt'] = tmpDF['loan_amnt'].astype('float')
#tmpDF['int_rate'] = tmpDF['int_rate'].astype('float')
#loanDF_Pandas['grade'] = loanDF_Pandas['grade'].astype('category')
#loanDF_Pandas['sub_grade'] = loanDF_Pandas['sub_grade'].astype('category')

#print(tmpDF)

g = sns.FacetGrid(tmpDF, col="sub_grade", sharex=True, col_wrap=5)
#g = sns.FacetGrid(loanDF.toPandas(), col="sub_grade", sharex=False, col_wrap=5)
g.map(sns.boxplot, 'grade', 'loan_amnt')

g.savefig("FacetGrid_LoanAmt.pdf")
#fig.savefig("FacetGrid_LoanAmt.pdf")
```


sub_grade = A1
sub_grade = A2
sub_grade = A3
sub_grade = A4
sub_grade = A5
sub_grade = B1
sub_grade = B2
sub_grade = B3
sub_grade = B4
sub_grade = B5
sub_grade = C1
sub_grade = C2
sub_grade = C3
sub_grade = C4
sub_grade = C5
sub_grade = D1
sub_grade = D2
sub_grade = D3
sub_grade = D4
sub_grade = D5
sub_grade = E1
sub_grade = E2
sub_grade = E3
sub_grade = E4
sub_grade = E5
sub_grade = F1
sub_grade = F2
sub_grade = F3
sub_grade = F4
sub_grade = F5
sub_grade = G1
sub_grade = G2
sub_grade = G3
sub_grade = G4
sub_grade = G5
loan_amnt
0
5000
10000
15000
20000
25000
30000
35000
A B C D E F G
grade

Analyzing interest rate distribution for each grade, factored over sub grade. This shows the interest rates goes high for low credit rating customers.

```
In [11]: g = sns.FacetGrid(tmpDF, col="sub_grade", sharex=False, col_wrap=5)
         #g = sns.FacetGrid(loanDF.toPandas(), col="sub_grade", sharex=False, col_wrap=5)
         g.map(sns.boxplot, 'grade', 'int_rate')

         g.savefig("FacetGrid_LoanInt.pdf")
         #figure.savefig("FacetGrid_LoanInt.pdf")
```


sub_grade = A1
sub_grade = A2
sub_grade = A3
sub_grade = A4
sub_grade = A5
sub_grade = B1
sub_grade = B2
sub_grade = B3
sub_grade = B4
sub_grade = B5
sub_grade = C1
sub_grade = C2
sub_grade = C3
sub_grade = C4
sub_grade = C5
sub_grade = D1
sub_grade = D2
sub_grade = D3
sub_grade = D4
sub_grade = D5
sub_grade = E1
sub_grade = E2
sub_grade = E3
sub_grade = E4
sub_grade = E5
sub_grade = F1
sub_grade = F2
sub_grade = F3
sub_grade = F4
sub_grade = F5
sub_grade = G1
sub_grade = G2
sub_grade = G3
sub_grade = G4
sub_grade = G5
int_rate
grade
A B C D E F G
5 10 15 20 25 30

0.0.7 US states map with the total loan amount

In [13]:
```
totalloansByState = loanDF.groupBy("addr_state").sum('loan_amnt').toPandas()
totalloansByState.columns = ["US-State", "Total Loan Amount"]

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],

data = dict(
        type='choropleth',
        colorscale = scl,
        #autocolorscale = False,
        locations = totalloansByState['US-State'],
        z = totalloansByState['Total Loan Amount'].astype(float),
        locationmode = 'USA-states',
        text = totalloansByState['US-State'],
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            )
        ),
        colorbar = dict(
            title = "Billions USD"
        ),
        #colorscale = 'Viridis',
        reversescale = True
    )

layout = dict(
        title = 'Total Loan Amount by US States',
        geo = dict(
            #scope='usa',
            projection=dict( type='albers usa' ),
            #showlakes = True,
            showframe=False
            #lakecolor = 'rgb(255, 255, 255)',
        ),
    )

#fig = dict(data=data, layout=layout)
#url = py.plot(fig, filename='d3-cloropleth-map')
choromap = go.Figure(data = [data],layout = layout)
iplot(choromap, validate=False, image = 'png', filename='StateMapLoanAmount')
```

```
<IPython.core.display.HTML object>
```

0.0.8 US states map with the median interest rates

Interest rates looks slightly higher since i am using state wise median interest rate (Avg doesnt make sence in this case). Uncomment the code in case of maximum/minimum interest rates.

In [14]:
```
loanDF.registerTempTable("loanDFTable")

# Median Interest rate - Statewise
medianIntRateByState = sqlContext.sql("select addr_state, percentile_approx(int_rate,
#print(medianIntRateByState)
medianIntRateByState.columns = ["US-State", "Median Interest Rate"]

# Max Interest rate - Statewise
#maxIntRateByState = sqlContext.sql("select addr_state, max(int_rate) as int_rate_max
#print(maxIntRateByState)
#maxIntRateByState.columns = ["US-State", "Max Interest Rate"]

# Min Interest rate - Statewise
#minIntRateByState = sqlContext.sql("select addr_state, min(int_rate) as int_rate_min
#print(minIntRateByState)
#minIntRateByState.columns = ["US-State", "Min Interest Rate"]

#print(totalloansByState)

scl = [[0.0, 'rgb(242,240,247)'],[0.2, 'rgb(218,218,235)'],[0.4, 'rgb(188,189,220)'],

data = dict(
        type='choropleth',
        colorscale = scl,
        #autocolorscale = False,
        locations = medianIntRateByState['US-State'],
        z = medianIntRateByState['Median Interest Rate'].astype(float),
        locationmode = 'USA-states',
        text = medianIntRateByState['US-State'],
        marker = dict(
            line = dict (
                color = 'rgb(255,255,255)',
                width = 2
            )
        ),
        colorbar = dict(
            title = "Interest %"
        ),
        #colorscale = 'Viridis',
        reversescale = True
    )

layout = dict(
        title = 'Median Interest Rates by US States',
```

```
                geo = dict(
                    #scope='usa',
                    projection=dict( type='albers usa' ),
                    #showlakes = True,
                    showframe=False
                    #lakecolor = 'rgb(255, 255, 255)',
                ),
            )

        #fig = dict(data=data, layout=layout)
        #url = py.plot(fig, filename='d3-cloropleth-map')
        choromap = go.Figure(data = [data],layout = layout)
        iplot(choromap,validate=False, image = 'png', filename='StateMapLoanInterest')
```

```
<IPython.core.display.HTML object>
```

0.0.9 Creating customer income range (Binning).

Creating 15 income range bins, used gaussian distribution technique to define the income range.

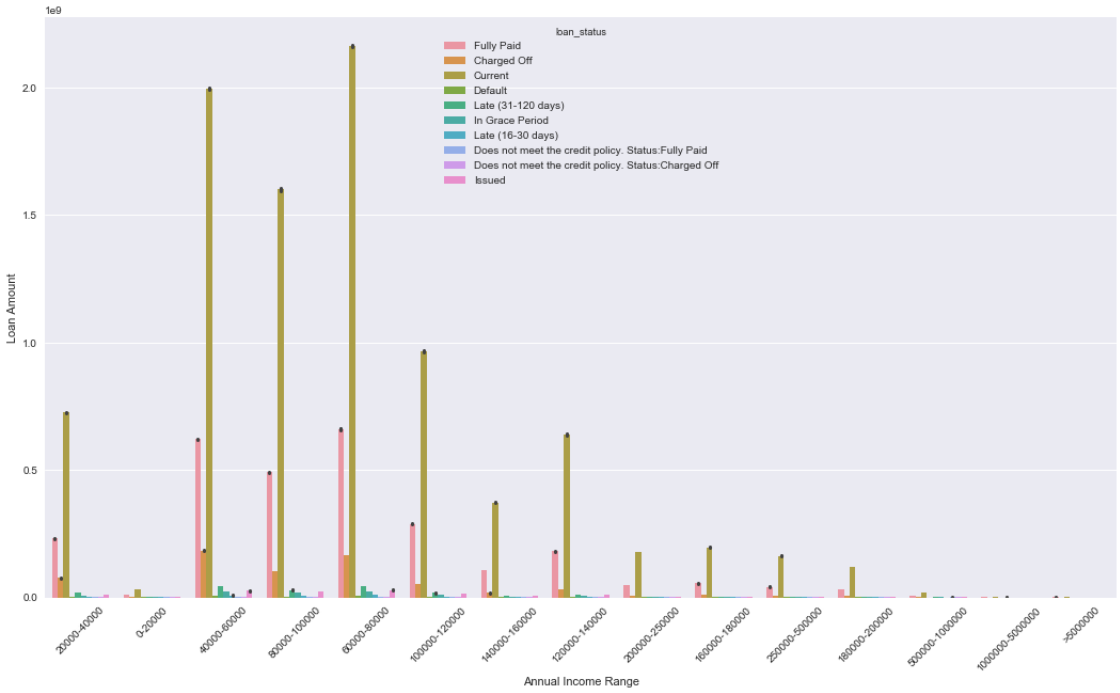
- Total number of loans grouped by income range and loan status
- Total loan amount by income range and loan status

In [16]:
```
loanDF = loanDF.withColumn("annual_inc",loanDF["annual_inc"].cast('float'))

        loanDF = loanDF.withColumn("annual_inc_range", when((col("annual_inc") >= -10000) & (
                                    .when((col("annual_inc") >= 20000) & (col("annual_inc") < 4
                                   .when((col("annual_inc") >= 40000) & (col("annual_inc") < 60
                                   .when((col("annual_inc") >= 60000) & (col("annual_inc") < 80
                                   .when((col("annual_inc") >= 80000) & (col("annual_inc") < 10
                                   .when((col("annual_inc") >= 100000) & (col("annual_inc") < 1
                                   .when((col("annual_inc") >= 120000) & (col("annual_inc") < 1
                                   .when((col("annual_inc") >= 140000) & (col("annual_inc") < 1
                                   .when((col("annual_inc") >= 160000) & (col("annual_inc") < 1
                                   .when((col("annual_inc") >= 180000) & (col("annual_inc") < 2
                                   .when((col("annual_inc") >= 200000) & (col("annual_inc") < 2
                                   .when((col("annual_inc") >= 250000) & (col("annual_inc") < 5
                                   .when((col("annual_inc") >= 500000) & (col("annual_inc") < 1
                                   .when((col("annual_inc") >= 1000000) & (col("annual_inc") <
                                   .otherwise(">5000000"))

        #loanDF.groupby("annual_inc_range", "loan_status").count().sort("annual_inc_range", "

        #loanDF.groupBy('loan_status').count().sort("count", ascending=False).show()
```

In [68]:
```
#loanDF.select("annual_inc", "annual_inc_range").show(50)
```

0.0.10 Total loan amount by income range and loan status

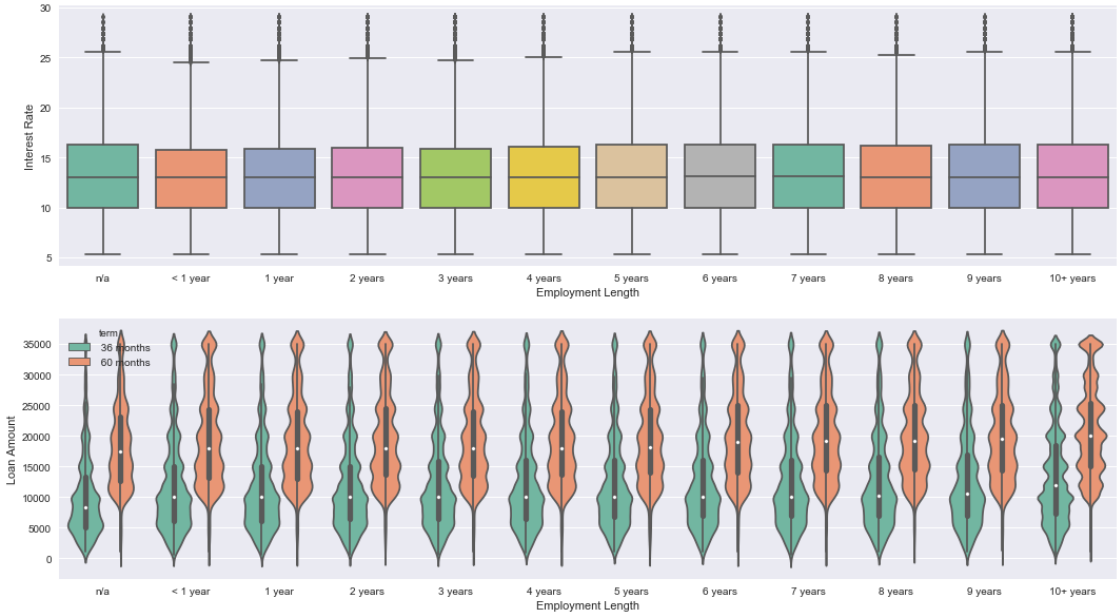
In [17]:
```
fig, ax =plt.subplots(1,1,figsize=(18,10))

    #s0 = sns.barplot(x="annual_inc_range", y="loan_amnt", hue="loan_status", data=loanDF
    s0 = sns.barplot(x="annual_inc_range", y="loan_amnt", hue="loan_status", data=loanDF.s
    s0.set(xlabel='Annual Income Range',ylabel='Loan Amount')
    s0.set_xticklabels(s0.get_xticklabels(),rotation=45)

    plt.show()

    fig.savefig("LoanByIncomeRange.pdf")
```



0.0.11 Analyzing Loan Amount and interest rate over customers employment length (With the loan term).

In [18]:
```
fig, ax = plt.subplots(2,1, figsize=(18,10))

    empLengthOrder = ["n/a",'< 1 year','1 year','2 years','3 years','4 years', '5 years',

    tmpLoanDF = loanDF.select("emp_length", "int_rate", "loan_amnt", "term", "purpose").to

    s0 = sns.boxplot(x="emp_length", y="int_rate", data=tmpLoanDF, palette="Set2", order=e
    s0.set(xlabel='Employment Length',ylabel='Interest Rate')
```

```
s1 = sns.violinplot(x="emp_length", y="loan_amnt",data=tmpLoanDF, hue="term", palette=
s1.set(xlabel='Employment Length',ylabel='Loan Amount')

plt.show()

fig.savefig("LoanAmountInt_OverEmploymentLength.pdf")
```

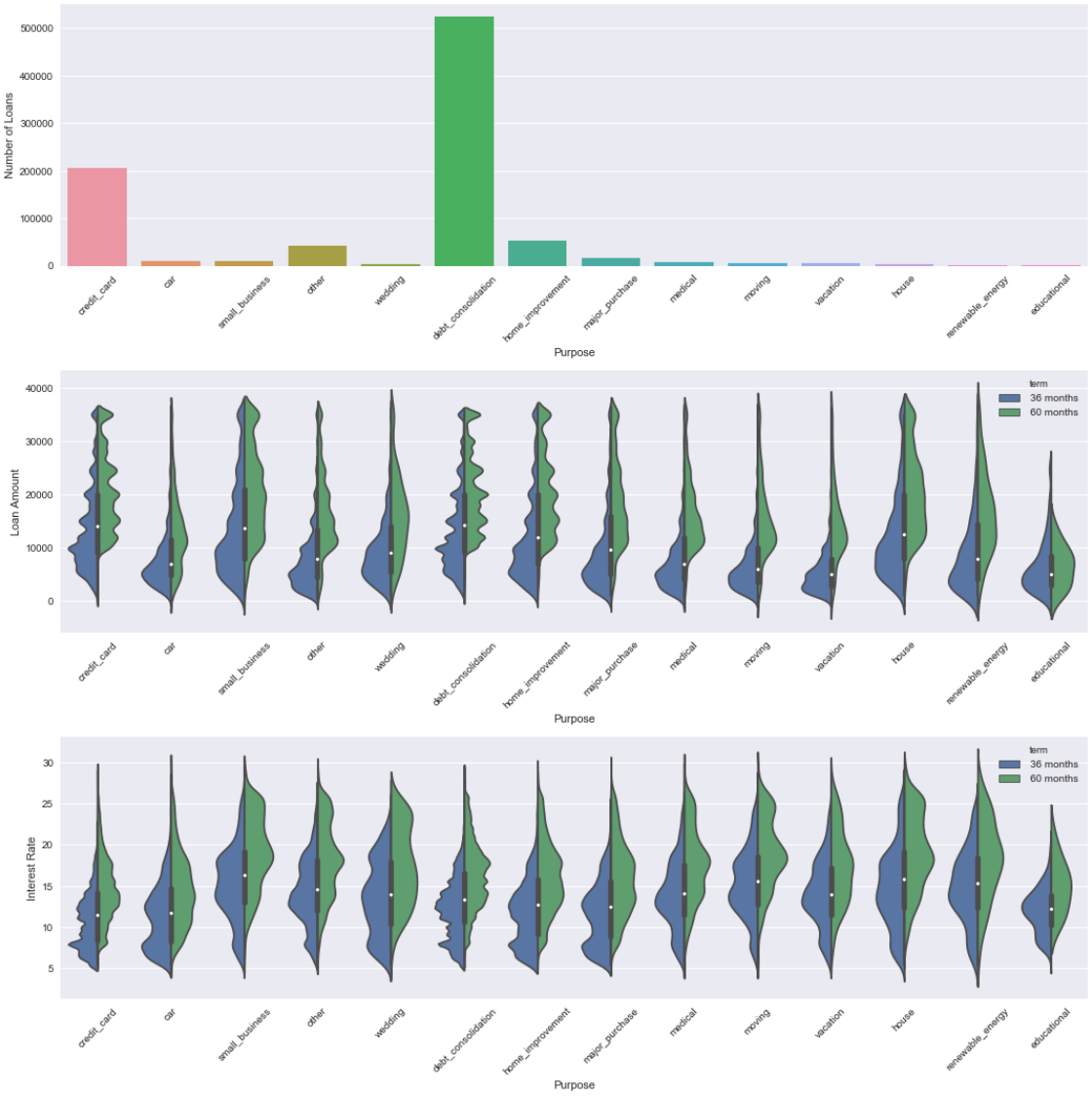


0.0.12 Alalyzing loans by its purpose

- Number of loans by its purpose
- Loan amount with its distribution pattern by purpose; hues by its term
- Interest rate with its distribution pattern by purpose; hues by its term

In [19]:
```
fig, ax=plt.subplots(3,1,figsize=(18,20))
plt.subplots_adjust(hspace = 0.4, top = 0.8)

#Already collected this in the above section
#tmpLoanDF = loanDF.select("purpose", "int_rate", "loan_amnt",).toPandas()

s0 = sns.countplot(x="purpose",data=tmpLoanDF,ax=ax[0])
s0.set(xlabel='Purpose',ylabel='Number of Loans')
s0.set_xticklabels(s0.get_xticklabels(),rotation=45)

s1 = sns.violinplot(x="purpose",y="loan_amnt",data=tmpLoanDF, ax=ax[1], hue="term", sp
s1.set(xlabel='Purpose',ylabel='Loan Amount')
s1.set_xticklabels(s1.get_xticklabels(),rotation=45)
```

```
s2 = sns.violinplot(x="purpose",y="int_rate",data=tmpLoanDF, ax=ax[2], hue="term", sp
s2.set(xlabel='Purpose',ylabel='Interest Rate')
s2.set_xticklabels(s2.get_xticklabels(),rotation=45)


plt.show()

fig.savefig("LoanByPurpose.pdf")
```



0.0.13 Creating interest range (Binning).

Creating interest range bins, to group by based on range

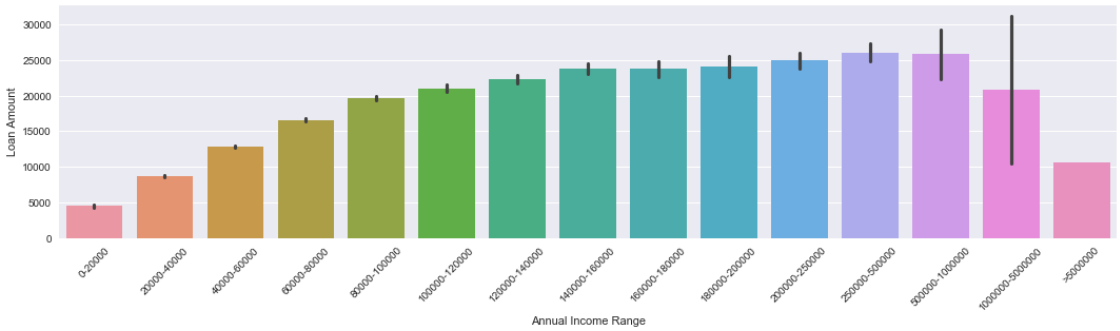
In [20]:
```
# Creating a int range columns
loanDF = loanDF.withColumn("int_rate_range", when((col("int_rate") >= 5) & (col("int_
                                .when((col("int_rate") >= 7) & (col("int_rate") < 9), "7-9%
                               .when((col("int_rate") >= 9) & (col("int_rate") < 11), "9-1
                               .when((col("int_rate") >= 11) & (col("int_rate") < 13), "11-
                               .when((col("int_rate") >= 13) & (col("int_rate") < 15), "13-
                               .when((col("int_rate") >= 15) & (col("int_rate") < 20), "15-
                               .when((col("int_rate") >= 20) & (col("int_rate") < 25), "20-
                               .when((col("int_rate") >= 25) & (col("int_rate") < 30), "25-
                               .otherwise(">30%"))

#loanDF.select("int_rate", "int_rate_range").show(50)
```

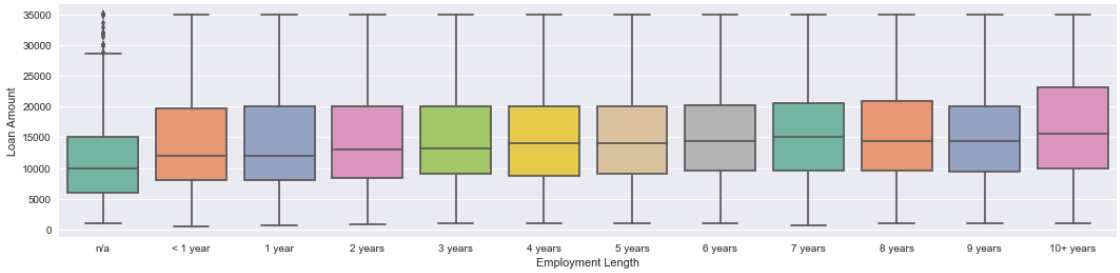
0.0.14 Analyzing Default loans

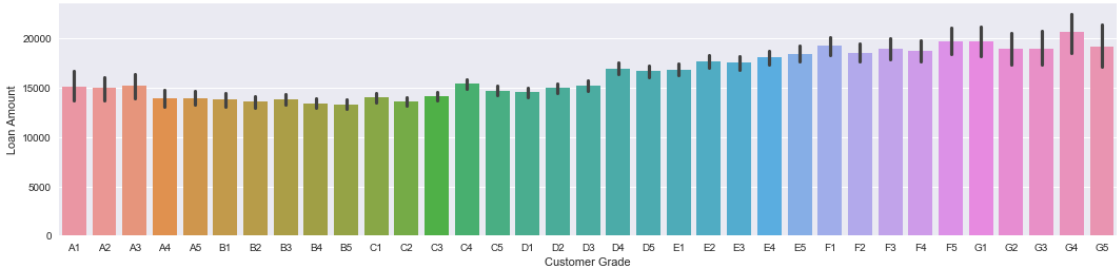
Loan status which are in following status will be considered as defaulted - - Default, Late (31-120 days), - In Grace Period, Late (16-30 days), - Does not meet the credit policy. - Status:Charged Off

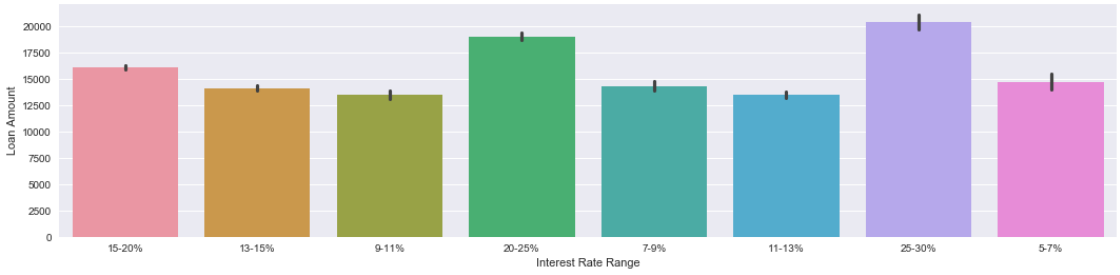
In [21]:
```
loanDF.registerTempTable("loanDFTable")

#mask = loanDF.loan_status.isin(["Default","Late (31-120 days)", "In Grace Period", "
                                #"Does not meet the credit policy. Status:Charged Of
#defaultLoanDF = loanDF.loc[mask]

defaultLoanDF = sqlContext.sql("select annual_inc_range, loan_amnt, emp_length, int_ra
from loanDFTable \
where loan_status in ('Default','Late (31-120 days)', 'In Grace Period', 'Late (16-30
'Does not meet the credit policy. Status:Charged Off')").toPandas()

#print(defaultLoanDF)

fig, ax = plt.subplots(5,1, figsize=(18,30))
plt.subplots_adjust(hspace = 0.4, top = 0.8)

incRangeOrder = ["0-20000","20000-40000","40000-60000","60000-80000", "80000-100000",
                 "140000-160000", "160000-180000", "180000-200000", "200000-250000", "
                 "1000000-5000000", ">5000000"]
s0 = sns.barplot(x="annual_inc_range", y="loan_amnt", data=defaultLoanDF, order=incRan
s0.set(xlabel='Annual Income Range',ylabel='Loan Amount')
s0.set_xticklabels(s0.get_xticklabels(),rotation=45)

empLengthOrder = ["n/a",'< 1 year','1 year','2 years','3 years','4 years', '5 years',

s1 = sns.boxplot(x="emp_length", y="loan_amnt", data=defaultLoanDF, palette="Set2", or
s1.set(xlabel='Employment Length',ylabel='Loan Amount')

s2 = sns.boxplot(x="emp_length", y="int_rate", data=defaultLoanDF, palette="Set2", ord
s2.set(xlabel='Employment Length',ylabel='Interest Rate')
```

```
#s0 = sns.countplot(x="purpose",data=loanDF,ax=ax[0])
#s0.set(xlabel='Purpose',ylabel='Number of Loans')
#s0.set_xticklabels(s0.get_xticklabels(),rotation=45)

#s1 = sns.violinplot(x="emp_length", y="loan_amnt",data=loanDF, hue="term", palette="
#s1.set(xlabel='Employment Length',ylabel='Loan Amount')

subgradeOrder = ["A1", "A2", "A3", "A4", "A5", "B1", "B2", "B3", "B4", "B5", "C1", "C2
                 "E1", "E2", "E3", "E4", "E5", "F1", "F2", "F3", "F4", "F5", "G1", "G2
s3 = sns.barplot(x="sub_grade", y="loan_amnt", data=defaultLoanDF, order = subgradeOrd
s3.set(xlabel='Customer Grade',ylabel='Loan Amount')

intRateRangeOrder = ["5-7%", "7-9%", "9-11%", "11-13%", "13-15%", "15-20%", "20-25%",
s4 = sns.barplot(x="int_rate_range", y="loan_amnt", data=defaultLoanDF, ax=ax[4])
s4.set(xlabel='Interest Rate Range',ylabel='Loan Amount')

plt.show()

fig.savefig("DefaultLoans.pdf")
```









In [22]: *## Preparing data for Learning model*

```
loanDF.groupby("loan_status").count().show()
```

```
+--------------------+------+
|         loan_status| count|
+--------------------+------+
|          Fully Paid|207533|
|             Default|  1219|
|     In Grace Period|  6253|
|Does not meet the...|  1969|
|         Charged Off| 45215|
|  Late (31-120 days)| 11591|
|              Issued|  8460|
|             Current|601776|
|Does not meet the...|   751|
|   Late (16-30 days)|  2357|
+--------------------+------+
```

0.0.15 Data Cleaning & Missing Data Imputation

In [23]:
```
# Cleaning up the data

######### 1. Removing all the features which has more than 50% of the data empty ####
# Temporary setting these hard coded values. (Above section takes lot of time to run)
missingValueColList = ['desc', 'mths_since_last_delinq', 'mths_since_last_record', 'm
loanDFForModel = loanDF.drop(*missingValueColList)

######### 2. Removing unique ID columns ##########
# Dropping ID & date columns (Unique id's, Don't help much in data analysis/modelling
loanDFForModel = loanDFForModel.drop("id", "member_id", "issue_d")
# Dropping the columns which were created for data analysis.
#loanDFForModel = loanDFForModel.drop("issue_d_dateType", "issue_year", "annual_inc_r

######### 3. Removing Other insignificant columns ##########
# application_type has only INDIVIDUAL, can be removed.
# pymnt_plan & initial_list_status has only one category "n" & "f". Keeping state fea
# removing date fileds as well. policy_code has only one category "1"
loanDFForModel = loanDFForModel.drop("emp_title", "url", "title", "zip_code", "earlie
                                     "next_pymnt_d", "last_credit_pull_d", "policy_co


######### 4. Missing data imputation for tot_cur_bal ##########
# 90% of the missing data in "tot_cur_bal", "tot_coll_amt" column can be filled with
loanDFForModel = loanDFForModel.withColumn("tot_cur_bal", when((col("tot_cur_bal").is
                                                    col("loan_status").isi
                                           .otherwise(col("tot_cur_bal")))
```

```
loanDFForModel = loanDFForModel.withColumn("tot_coll_amt", when((col("tot_coll_amt").
                                                        col("loan_status").isi
                                              .otherwise(col("tot_coll_amt")))

# Inputing mean value for "total_rev_hi_lim"
mean = int(loanDFForModel.select(avg("total_rev_hi_lim")).take(1)[0][0])
loanDFForModel = loanDFForModel.withColumn("total_rev_hi_lim", when(col("total_rev_hi
                                              .otherwise(col("total_rev_hi_lim")))

######### 5. Removing loan observations which still have missing data. (~ 0.8% record
#print("Total Loan Observations - ", loanDFForModel.count())
loanDFForModel = loanDFForModel.dropna(how="any")
#print("Loan Observations after dropna- ",loanDFForModel.count())


######### 6. Adding the lable column to dataframe. 1- defalut and 0-paid/current ####
loanDFForModel = loanDFForModel.withColumn("isDefault", when(col("loan_status").isin("
                                                        "Does not meet t
                                .otherwise(0))
#loanDFForModel.groupby("isDefault").count().show()


######### 7. Changing the feature datatype from string to numeric ##########
loanDFForModel = loanDFForModel.withColumn("loan_amnt",loanDFForModel["loan_amnt"].cas
loanDFForModel = loanDFForModel.withColumn("funded_amnt",loanDFForModel["funded_amnt"]
loanDFForModel = loanDFForModel.withColumn("funded_amnt_inv",loanDFForModel["funded_a
loanDFForModel = loanDFForModel.withColumn("int_rate",loanDFForModel["int_rate"].cast
loanDFForModel = loanDFForModel.withColumn("installment",loanDFForModel["installment"]
loanDFForModel = loanDFForModel.withColumn("annual_inc",loanDFForModel["annual_inc"].c
loanDFForModel = loanDFForModel.withColumn("dti",loanDFForModel["dti"].cast('float'))
loanDFForModel = loanDFForModel.withColumn("delinq_2yrs",loanDFForModel["delinq_2yrs"]
loanDFForModel = loanDFForModel.withColumn("inq_last_6mths",loanDFForModel["inq_last_6
loanDFForModel = loanDFForModel.withColumn("open_acc",loanDFForModel["open_acc"].cast
loanDFForModel = loanDFForModel.withColumn("pub_rec",loanDFForModel["pub_rec"].cast('i
loanDFForModel = loanDFForModel.withColumn("revol_bal",loanDFForModel["revol_bal"].cas
loanDFForModel = loanDFForModel.withColumn("revol_util",loanDFForModel["revol_util"].c
loanDFForModel = loanDFForModel.withColumn("total_acc",loanDFForModel["total_acc"].cas
#loanDFForModel = loanDFForModel.withColumn("out_prncp",loanDFForModel["out_prncp"].c
#loanDFForModel = loanDFForModel.withColumn("out_prncp_inv",loanDFForModel["out_prncp_
#loanDFForModel = loanDFForModel.withColumn("total_pymnt",loanDFForModel["total_pymnt
#loanDFForModel = loanDFForModel.withColumn("total_pymnt_inv",loanDFForModel["total_py
#loanDFForModel = loanDFForModel.withColumn("total_rec_prncp",loanDFForModel["total_re
#loanDFForModel = loanDFForModel.withColumn("total_rec_int",loanDFForModel["total_rec_
#loanDFForModel = loanDFForModel.withColumn("total_rec_late_fee",loanDFForModel["total
#loanDFForModel = loanDFForModel.withColumn("recoveries",loanDFForModel["recoveries"]
#loanDFForModel = loanDFForModel.withColumn("collection_recovery_fee",loanDFForModel[
#loanDFForModel = loanDFForModel.withColumn("last_pymnt_amnt",loanDFForModel["last_pym
```

```
#loanDFForModel = loanDFForModel.withColumn("collections_12_mths_ex_med",loanDFForMod
#loanDFForModel = loanDFForModel.withColumn("acc_now_delinq",loanDFForModel["acc_now_
#loanDFForModel = loanDFForModel.withColumn("tot_coll_amt",loanDFForModel["tot_coll_a
#loanDFForModel = loanDFForModel.withColumn("tot_cur_bal",loanDFForModel["tot_cur_bal
#loanDFForModel = loanDFForModel.withColumn("total_rev_hi_lim",loanDFForModel["total_
#loanDFForModel = loanDFForModel.withColumn("collection_recovery_fee",loanDFForModel[


######### 8. Finally removing loan_status ##########
#loan_status is used to create the class lable, removing it to avoid data leakage.
loanDFForModel = loanDFForModel.drop("loan_status")


######### 8. Removing the fileds which are related to the current loan ##########
loanDFForModel = loanDFForModel.drop("out_prncp", "out_prncp_inv", "total_pymnt", "to
                                     "total_rec_int", "total_rec_late_fee", "recoveri
                                    "last_pymnt_amnt", "collections_12_mths_ex_med",
                                    "tot_cur_bal", "total_rev_hi_lim")


#print(type(loanDFForModel))
#loanDFForModel.printSchema()

# Term, grade, sub_grade, emp_length, home_ownership, verification_status, pymnt_plan
# initial_list_status, application_type
```

0.0.16 Binary Encoding for Categorical Feature - "term", "initial_list_status", "application_type", "pymnt_plan"

These features have only two categories

In [24]:
```
#indexer = StringIndexer(inputCol="term", outputCol="termIndex")
#loanDFForModel = indexer.fit(loanDFForModel).transform(loanDFForModel)

indexer = StringIndexer(inputCol="term", outputCol="termIndex", handleInvalid="keep")
loanDFForModel = indexer.fit(loanDFForModel).transform(loanDFForModel)

indexer = StringIndexer(inputCol="initial_list_status", outputCol="initial_list_statu
loanDFForModel = indexer.fit(loanDFForModel).transform(loanDFForModel)

indexer = StringIndexer(inputCol="application_type", outputCol="application_typeIndex
loanDFForModel = indexer.fit(loanDFForModel).transform(loanDFForModel)

indexer = StringIndexer(inputCol="pymnt_plan", outputCol="pymnt_planIndex", handleInva
loanDFForModel = indexer.fit(loanDFForModel).transform(loanDFForModel)

#loanDFForModel.show()
```

0.0.17 One-Hot Encoding for Categorical Feature - "grade"

In [25]:
```
#categories = loanDFForModel.select("grade").distinct().rdd.flatMap(lambda x: x).coll
categories = loanDFForModel.select("grade").distinct().toPandas().grade.tolist()
#print(categories)
#print(type(categories))
exprs = [F.when(F.col("grade") == category, 1).otherwise(0).alias(category)
         for category in categories]

loanDFForModel = loanDFForModel.select("*", *exprs)

newCategories = []
for category in categories:
    loanDFForModel = loanDFForModel.withColumnRenamed(category, "grade_"+category)
    newCategories.append("grade_"+category)
print(newCategories)
```

```
['grade_F', 'grade_E', 'grade_B', 'grade_D', 'grade_C', 'grade_A', 'grade_G']
```

0.0.18 One-Hot Encoding for Categorical Feature - "sub_grade"

In [26]:
```
#loanDF.registerTempTable("loanDFTable")
#categories = sqlContext.sql("select distinct(sub_grade) from loanDFTable").collect()
categories = loanDFForModel.select("sub_grade").distinct().toPandas().sub_grade.tolis
#print(categories)
#print(type(categories))
exprs = [F.when(F.col("sub_grade") == category, 1).otherwise(0).alias(category)
         for category in categories]

loanDFForModel = loanDFForModel.select("*", *exprs)

newCategories = []
for category in categories:
    loanDFForModel = loanDFForModel.withColumnRenamed(category, "sub_grade_"+category
    newCategories.append("sub_grade_"+category)
print(newCategories)
```

```
['sub_grade_D5', 'sub_grade_F2', 'sub_grade_B4', 'sub_grade_A2', 'sub_grade_E4', 'sub_grade_B2
```

0.0.19 One-Hot Encoding for Categorical Feature - "home_ownership"

In [27]:
```
#categories = loanDFForModel.select("home_ownership").distinct().rdd.flatMap(lambda x
categories = loanDFForModel.select("home_ownership").distinct().toPandas().home_owners
#print(categories)
exprs = [F.when(F.col("home_ownership") == category, 1).otherwise(0).alias(category)
         for category in categories]
```

```
        loanDFForModel = loanDFForModel.select("*", *exprs)

        newCategories = []
        for category in categories:
            loanDFForModel = loanDFForModel.withColumnRenamed(category, "home_ownership_"+cate
            newCategories.append("home_ownership_"+category)
        print(newCategories)
```

```
['home_ownership_OWN', 'home_ownership_RENT', 'home_ownership_MORTGAGE', 'home_ownership_ANY',
```

0.0.20 One-Hot Encoding for Categorical Feature - "verification_status"

```
In [28]: #categories = loanDFForModel.select("verification_status").distinct().rdd.flatMap(lam
         categories = loanDFForModel.select("verification_status").distinct().toPandas().verif
         #print(categories)
         exprs = [F.when(F.col("verification_status") == category, 1).otherwise(0).alias(categ
                  for category in categories]

         loanDFForModel = loanDFForModel.select("*", *exprs)

         newCategories = []
         for category in categories:
             loanDFForModel = loanDFForModel.withColumnRenamed(category, "verification_status_
             newCategories.append("verification_status_"+category)
         print(newCategories)
```

```
['verification_status_Verified', 'verification_status_Source Verified', 'verification_status_N
```

0.0.21 One-Hot Encoding for Categorical Feature - "purpose"

```
In [29]: #categories = loanDFForModel.select("purpose").distinct().rdd.flatMap(lambda x: x).co
         categories = loanDFForModel.select("purpose").distinct().toPandas().purpose.tolist()
         #print(categories)
         exprs = [F.when(F.col("purpose") == category, 1).otherwise(0).alias(category)
                  for category in categories]

         loanDFForModel = loanDFForModel.select("*", *exprs)

         newCategories = []
         for category in categories:
             loanDFForModel = loanDFForModel.withColumnRenamed(category, "purpose_"+category)
             newCategories.append("purpose_"+category)
         print(newCategories)
```

```
['purpose_wedding', 'purpose_educational', 'purpose_other', 'purpose_small_business', 'purpose_
```

0.0.22 One-Hot Encoding for Categorical Feature - "addr_state"

In [30]:
```
#categories = loanDFForModel.select("addr_state").distinct().rdd.flatMap(lambda x: x)
categories = loanDFForModel.select("addr_state").distinct().toPandas().addr_state.tol
#print(categories)
exprs = [F.when(F.col("addr_state") == category, 1).otherwise(0).alias(category)
         for category in categories]

loanDFForModel = loanDFForModel.select("*", *exprs)

newCategories = []
for category in categories:
    loanDFForModel = loanDFForModel.withColumnRenamed(category, "addr_state_"+categor
    newCategories.append("addr_state_"+category)
print(newCategories)
```

```
['addr_state_AZ', 'addr_state_SC', 'addr_state_LA', 'addr_state_MN', 'addr_state_NJ', 'addr_sta
```

In [31]:
```
loanDFForModel.printSchema()

#loanDFForModel = loanDFForModel.withColumnRenamed("D5", "sub_grade_D5")

#loanDFForModel.printSchema()
```

```
root
 |-- loan_amnt: float (nullable = true)
 |-- funded_amnt: float (nullable = true)
 |-- funded_amnt_inv: float (nullable = true)
 |-- term: string (nullable = true)
 |-- int_rate: float (nullable = true)
 |-- installment: float (nullable = true)
 |-- grade: string (nullable = true)
 |-- sub_grade: string (nullable = true)
 |-- emp_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: float (nullable = true)
 |-- verification_status: string (nullable = true)
 |-- pymnt_plan: string (nullable = true)
 |-- purpose: string (nullable = true)
 |-- addr_state: string (nullable = true)
 |-- dti: float (nullable = true)
 |-- delinq_2yrs: float (nullable = true)
 |-- inq_last_6mths: float (nullable = true)
 |-- open_acc: float (nullable = true)
 |-- pub_rec: float (nullable = true)
 |-- revol_bal: float (nullable = true)
 |-- revol_util: float (nullable = true)
 |-- total_acc: float (nullable = true)
```

```
|-- initial_list_status: string (nullable = true)
|-- application_type: string (nullable = true)
|-- issue_d_dateType: date (nullable = true)
|-- issue_year: string (nullable = true)
|-- annual_inc_range: string (nullable = false)
|-- int_rate_range: string (nullable = false)
|-- isDefault: integer (nullable = false)
|-- termIndex: double (nullable = false)
|-- initial_list_statusIndex: double (nullable = false)
|-- application_typeIndex: double (nullable = false)
|-- pymnt_planIndex: double (nullable = false)
|-- grade_F: integer (nullable = false)
|-- grade_E: integer (nullable = false)
|-- grade_B: integer (nullable = false)
|-- grade_D: integer (nullable = false)
|-- grade_C: integer (nullable = false)
|-- grade_A: integer (nullable = false)
|-- grade_G: integer (nullable = false)
|-- sub_grade_D5: integer (nullable = false)
|-- sub_grade_F2: integer (nullable = false)
|-- sub_grade_B4: integer (nullable = false)
|-- sub_grade_A2: integer (nullable = false)
|-- sub_grade_E4: integer (nullable = false)
|-- sub_grade_B2: integer (nullable = false)
|-- sub_grade_C3: integer (nullable = false)
|-- sub_grade_D1: integer (nullable = false)
|-- sub_grade_C4: integer (nullable = false)
|-- sub_grade_F1: integer (nullable = false)
|-- sub_grade_D3: integer (nullable = false)
|-- sub_grade_F5: integer (nullable = false)
|-- sub_grade_G2: integer (nullable = false)
|-- sub_grade_B1: integer (nullable = false)
|-- sub_grade_B3: integer (nullable = false)
|-- sub_grade_E5: integer (nullable = false)
|-- sub_grade_C5: integer (nullable = false)
|-- sub_grade_G3: integer (nullable = false)
|-- sub_grade_A4: integer (nullable = false)
|-- sub_grade_F4: integer (nullable = false)
|-- sub_grade_B5: integer (nullable = false)
|-- sub_grade_E3: integer (nullable = false)
|-- sub_grade_G4: integer (nullable = false)
|-- sub_grade_D2: integer (nullable = false)
|-- sub_grade_C1: integer (nullable = false)
|-- sub_grade_F3: integer (nullable = false)
|-- sub_grade_E1: integer (nullable = false)
|-- sub_grade_A5: integer (nullable = false)
|-- sub_grade_C2: integer (nullable = false)
|-- sub_grade_D4: integer (nullable = false)
```

```
|-- sub_grade_E2: integer (nullable = false)
|-- sub_grade_A3: integer (nullable = false)
|-- sub_grade_G5: integer (nullable = false)
|-- sub_grade_G1: integer (nullable = false)
|-- sub_grade_A1: integer (nullable = false)
|-- home_ownership_OWN: integer (nullable = false)
|-- home_ownership_RENT: integer (nullable = false)
|-- home_ownership_MORTGAGE: integer (nullable = false)
|-- home_ownership_ANY: integer (nullable = false)
|-- home_ownership_OTHER: integer (nullable = false)
|-- home_ownership_NONE: integer (nullable = false)
|-- verification_status_Verified: integer (nullable = false)
|-- verification_status_Source Verified: integer (nullable = false)
|-- verification_status_Not Verified: integer (nullable = false)
|-- purpose_wedding: integer (nullable = false)
|-- purpose_educational: integer (nullable = false)
|-- purpose_other: integer (nullable = false)
|-- purpose_small_business: integer (nullable = false)
|-- purpose_debt_consolidation: integer (nullable = false)
|-- purpose_credit_card: integer (nullable = false)
|-- purpose_moving: integer (nullable = false)
|-- purpose_vacation: integer (nullable = false)
|-- purpose_renewable_energy: integer (nullable = false)
|-- purpose_house: integer (nullable = false)
|-- purpose_car: integer (nullable = false)
|-- purpose_major_purchase: integer (nullable = false)
|-- purpose_medical: integer (nullable = false)
|-- purpose_home_improvement: integer (nullable = false)
|-- addr_state_AZ: integer (nullable = false)
|-- addr_state_SC: integer (nullable = false)
|-- addr_state_LA: integer (nullable = false)
|-- addr_state_MN: integer (nullable = false)
|-- addr_state_NJ: integer (nullable = false)
|-- addr_state_DC: integer (nullable = false)
|-- addr_state_OR: integer (nullable = false)
|-- addr_state_VA: integer (nullable = false)
|-- addr_state_RI: integer (nullable = false)
|-- addr_state_KY: integer (nullable = false)
|-- addr_state_WY: integer (nullable = false)
|-- addr_state_NH: integer (nullable = false)
|-- addr_state_MI: integer (nullable = false)
|-- addr_state_NV: integer (nullable = false)
|-- addr_state_WI: integer (nullable = false)
|-- addr_state_ID: integer (nullable = false)
|-- addr_state_CA: integer (nullable = false)
|-- addr_state_CT: integer (nullable = false)
|-- addr_state_NE: integer (nullable = false)
|-- addr_state_MT: integer (nullable = false)
```

```
 |-- addr_state_NC: integer (nullable = false)
 |-- addr_state_VT: integer (nullable = false)
 |-- addr_state_MD: integer (nullable = false)
 |-- addr_state_DE: integer (nullable = false)
 |-- addr_state_MO: integer (nullable = false)
 |-- addr_state_IL: integer (nullable = false)
 |-- addr_state_ME: integer (nullable = false)
 |-- addr_state_WA: integer (nullable = false)
 |-- addr_state_ND: integer (nullable = false)
 |-- addr_state_MS: integer (nullable = false)
 |-- addr_state_AL: integer (nullable = false)
 |-- addr_state_IN: integer (nullable = false)
 |-- addr_state_OH: integer (nullable = false)
 |-- addr_state_TN: integer (nullable = false)
 |-- addr_state_NM: integer (nullable = false)
 |-- addr_state_IA: integer (nullable = false)
 |-- addr_state_PA: integer (nullable = false)
 |-- addr_state_SD: integer (nullable = false)
 |-- addr_state_NY: integer (nullable = false)
 |-- addr_state_TX: integer (nullable = false)
 |-- addr_state_WV: integer (nullable = false)
 |-- addr_state_GA: integer (nullable = false)
 |-- addr_state_MA: integer (nullable = false)
 |-- addr_state_KS: integer (nullable = false)
 |-- addr_state_FL: integer (nullable = false)
 |-- addr_state_CO: integer (nullable = false)
 |-- addr_state_AK: integer (nullable = false)
 |-- addr_state_AR: integer (nullable = false)
 |-- addr_state_OK: integer (nullable = false)
 |-- addr_state_UT: integer (nullable = false)
 |-- addr_state_HI: integer (nullable = false)
```

0.0.23 Converting Categorical feature "emp_lenght" to continous feature

```
In [32]: loanDFForModel = loanDFForModel.withColumn("emp_lengthIndex", when(col("emp_length") =
                                                    .when(col("emp_length") == "1 year", 1)
                                                    .when(col("emp_length") == "2 years", 2)
                                                    .when(col("emp_length") == "3 years", 3)
                                                    .when(col("emp_length") == "4 years", 4)
                                                    .when(col("emp_length") == "5 years", 5)
                                                    .when(col("emp_length") == "6 years", 6)
                                                    .when(col("emp_length") == "7 years", 7)
                                                    .when(col("emp_length") == "8 years", 8)
                                                    .when(col("emp_length") == "9 years", 9)
                                                    .when(col("emp_length") == "10+ years", 10)
                                                    .otherwise(0))
```

0.0.24 Dropping categorical features after One-Hot encoding

```
In [33]: # Remove the original categorical columns after encoding.
         loanDFForModel = loanDFForModel.drop("term", "initial_list_status", "application_type"
                                              "home_ownership", "verification_status", "purpose

         loanDFForModel = loanDFForModel.withColumn("ClassLable", col("isDefault"))
         loanDFForModel = loanDFForModel.drop("isDefault")

         loanDFForModel.printSchema()
```

```
root
 |-- loan_amnt: float (nullable = true)
 |-- funded_amnt: float (nullable = true)
 |-- funded_amnt_inv: float (nullable = true)
 |-- int_rate: float (nullable = true)
 |-- installment: float (nullable = true)
 |-- annual_inc: float (nullable = true)
 |-- dti: float (nullable = true)
 |-- delinq_2yrs: float (nullable = true)
 |-- inq_last_6mths: float (nullable = true)
 |-- open_acc: float (nullable = true)
 |-- pub_rec: float (nullable = true)
 |-- revol_bal: float (nullable = true)
 |-- revol_util: float (nullable = true)
 |-- total_acc: float (nullable = true)
 |-- issue_d_dateType: date (nullable = true)
 |-- issue_year: string (nullable = true)
 |-- annual_inc_range: string (nullable = false)
 |-- int_rate_range: string (nullable = false)
 |-- termIndex: double (nullable = false)
 |-- initial_list_statusIndex: double (nullable = false)
 |-- application_typeIndex: double (nullable = false)
 |-- pymnt_planIndex: double (nullable = false)
 |-- grade_F: integer (nullable = false)
 |-- grade_E: integer (nullable = false)
 |-- grade_B: integer (nullable = false)
 |-- grade_D: integer (nullable = false)
 |-- grade_C: integer (nullable = false)
 |-- grade_A: integer (nullable = false)
 |-- grade_G: integer (nullable = false)
 |-- sub_grade_D5: integer (nullable = false)
 |-- sub_grade_F2: integer (nullable = false)
 |-- sub_grade_B4: integer (nullable = false)
 |-- sub_grade_A2: integer (nullable = false)
 |-- sub_grade_E4: integer (nullable = false)
 |-- sub_grade_B2: integer (nullable = false)
 |-- sub_grade_C3: integer (nullable = false)
 |-- sub_grade_D1: integer (nullable = false)
```

```
|-- sub_grade_C4: integer (nullable = false)
|-- sub_grade_F1: integer (nullable = false)
|-- sub_grade_D3: integer (nullable = false)
|-- sub_grade_F5: integer (nullable = false)
|-- sub_grade_G2: integer (nullable = false)
|-- sub_grade_B1: integer (nullable = false)
|-- sub_grade_B3: integer (nullable = false)
|-- sub_grade_E5: integer (nullable = false)
|-- sub_grade_C5: integer (nullable = false)
|-- sub_grade_G3: integer (nullable = false)
|-- sub_grade_A4: integer (nullable = false)
|-- sub_grade_F4: integer (nullable = false)
|-- sub_grade_B5: integer (nullable = false)
|-- sub_grade_E3: integer (nullable = false)
|-- sub_grade_G4: integer (nullable = false)
|-- sub_grade_D2: integer (nullable = false)
|-- sub_grade_C1: integer (nullable = false)
|-- sub_grade_F3: integer (nullable = false)
|-- sub_grade_E1: integer (nullable = false)
|-- sub_grade_A5: integer (nullable = false)
|-- sub_grade_C2: integer (nullable = false)
|-- sub_grade_D4: integer (nullable = false)
|-- sub_grade_E2: integer (nullable = false)
|-- sub_grade_A3: integer (nullable = false)
|-- sub_grade_G5: integer (nullable = false)
|-- sub_grade_G1: integer (nullable = false)
|-- sub_grade_A1: integer (nullable = false)
|-- home_ownership_OWN: integer (nullable = false)
|-- home_ownership_RENT: integer (nullable = false)
|-- home_ownership_MORTGAGE: integer (nullable = false)
|-- home_ownership_ANY: integer (nullable = false)
|-- home_ownership_OTHER: integer (nullable = false)
|-- home_ownership_NONE: integer (nullable = false)
|-- verification_status_Verified: integer (nullable = false)
|-- verification_status_Source Verified: integer (nullable = false)
|-- verification_status_Not Verified: integer (nullable = false)
|-- purpose_wedding: integer (nullable = false)
|-- purpose_educational: integer (nullable = false)
|-- purpose_other: integer (nullable = false)
|-- purpose_small_business: integer (nullable = false)
|-- purpose_debt_consolidation: integer (nullable = false)
|-- purpose_credit_card: integer (nullable = false)
|-- purpose_moving: integer (nullable = false)
|-- purpose_vacation: integer (nullable = false)
|-- purpose_renewable_energy: integer (nullable = false)
|-- purpose_house: integer (nullable = false)
|-- purpose_car: integer (nullable = false)
|-- purpose_major_purchase: integer (nullable = false)
```

```
|-- purpose_medical: integer (nullable = false)
|-- purpose_home_improvement: integer (nullable = false)
|-- addr_state_AZ: integer (nullable = false)
|-- addr_state_SC: integer (nullable = false)
|-- addr_state_LA: integer (nullable = false)
|-- addr_state_MN: integer (nullable = false)
|-- addr_state_NJ: integer (nullable = false)
|-- addr_state_DC: integer (nullable = false)
|-- addr_state_OR: integer (nullable = false)
|-- addr_state_VA: integer (nullable = false)
|-- addr_state_RI: integer (nullable = false)
|-- addr_state_KY: integer (nullable = false)
|-- addr_state_WY: integer (nullable = false)
|-- addr_state_NH: integer (nullable = false)
|-- addr_state_MI: integer (nullable = false)
|-- addr_state_NV: integer (nullable = false)
|-- addr_state_WI: integer (nullable = false)
|-- addr_state_ID: integer (nullable = false)
|-- addr_state_CA: integer (nullable = false)
|-- addr_state_CT: integer (nullable = false)
|-- addr_state_NE: integer (nullable = false)
|-- addr_state_MT: integer (nullable = false)
|-- addr_state_NC: integer (nullable = false)
|-- addr_state_VT: integer (nullable = false)
|-- addr_state_MD: integer (nullable = false)
|-- addr_state_DE: integer (nullable = false)
|-- addr_state_MO: integer (nullable = false)
|-- addr_state_IL: integer (nullable = false)
|-- addr_state_ME: integer (nullable = false)
|-- addr_state_WA: integer (nullable = false)
|-- addr_state_ND: integer (nullable = false)
|-- addr_state_MS: integer (nullable = false)
|-- addr_state_AL: integer (nullable = false)
|-- addr_state_IN: integer (nullable = false)
|-- addr_state_OH: integer (nullable = false)
|-- addr_state_TN: integer (nullable = false)
|-- addr_state_NM: integer (nullable = false)
|-- addr_state_IA: integer (nullable = false)
|-- addr_state_PA: integer (nullable = false)
|-- addr_state_SD: integer (nullable = false)
|-- addr_state_NY: integer (nullable = false)
|-- addr_state_TX: integer (nullable = false)
|-- addr_state_WV: integer (nullable = false)
|-- addr_state_GA: integer (nullable = false)
|-- addr_state_MA: integer (nullable = false)
|-- addr_state_KS: integer (nullable = false)
|-- addr_state_FL: integer (nullable = false)
|-- addr_state_CO: integer (nullable = false)
```

```
 |-- addr_state_AK: integer (nullable = false)
 |-- addr_state_AR: integer (nullable = false)
 |-- addr_state_OK: integer (nullable = false)
 |-- addr_state_UT: integer (nullable = false)
 |-- addr_state_HI: integer (nullable = false)
 |-- emp_lengthIndex: integer (nullable = false)
 |-- ClassLable: integer (nullable = false)
```

```
In [34]: loanDFForModel.groupby("ClassLable").count().show()
```

```
+----------+------+
|ClassLable| count|
+----------+------+
|         1| 60153|
|         0|819919|
+----------+------+
```

0.0.25 Adding a Weight Column to handle class imbalancing.

```
In [35]: # Adding a weight columns to the dataset to handel class imbalance
         # Hardcoding it to save some execution time - (60153/819919) - (#Default Loans / #Tot
         balancingRatio  = 0.0735

         loanDFForModel = loanDFForModel.withColumn("weightColumn", when(col("ClassLable") == (
                                                    .otherwise((1-balancingRatio)))

         #loanDFForModel.groupby("weightColumn").count().show()
```

```
In [36]: loanDFForModel = loanDFForModel.drop("issue_d_dateType", "issue_year", "annual_inc_ra
```

```
In [37]: colList = loanDFForModel.columns
         print(colList)
         print(len(colList))
         colList.remove("ClassLable")
         print(colList)
         print(len(colList))
```

```
['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'int_rate', 'installment', 'annual_inc', 'dti'
137
['loan_amnt', 'funded_amnt', 'funded_amnt_inv', 'int_rate', 'installment', 'annual_inc', 'dti'
136
```

0.0.26 Vectorizing feature columns

In [38]:
```
#assembler = VectorAssembler(inputCols=[ *colList ], outputCol="features")

#transformed = assembler.transform(loanDFForModel)

#trainLablePoint = transformed.select(col("isDefault").alias("label"), col("features"

# set the input and output column names
assembler = VectorAssembler(inputCols=[ *colList ], outputCol="features")

loanDFTransformed = assembler.transform( loanDFForModel)

loanDFTransformed.show(5)
```

```
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
|loan_amnt|funded_amnt|funded_amnt_inv|int_rate|installment|annual_inc|  dti|delinq_2yrs|inq_la
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
|   5000.0|     5000.0|        4975.0|   10.65|     162.87|   24000.0|27.65|        0.0|
|   2500.0|     2500.0|        2500.0|   15.27|      59.83|   30000.0|  1.0|        0.0|
|   2400.0|     2400.0|        2400.0|   15.96|      84.33|   12252.0| 8.72|        0.0|
|  10000.0|    10000.0|       10000.0|   13.49|     339.31|   49200.0| 20.0|        0.0|
|   5000.0|     5000.0|        5000.0|    7.9|     156.46|   36000.0| 11.2|        0.0|
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
only showing top 5 rows
```

0.0.27 Creating LabelPoint, adding the class lable in transformed data

In [39]:
```
#abelIndexer = new StringIndexer().setInputCol("creditability").setOutputCol("label")
labelIndexer = StringIndexer(inputCol="ClassLable", outputCol="label", handleInvalid="
#val df3 = labelIndexer.fit(df2).transform(df2)
loanDFTransformed_2 = labelIndexer.fit(loanDFTransformed).transform(loanDFTransformed)

loanDFTransformed_2.show(5)

#indexer = StringIndexer(inputCol="term", outputCol="termIndex", handleInvalid="keep"
#loanDFTransformed = indexer.fit(loanDFForModel).transform(loanDFForModel)
```

```
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
|loan_amnt|funded_amnt|funded_amnt_inv|int_rate|installment|annual_inc|  dti|delinq_2yrs|inq_la
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
|   5000.0|     5000.0|        4975.0|   10.65|     162.87|   24000.0|27.65|        0.0|
|   2500.0|     2500.0|        2500.0|   15.27|      59.83|   30000.0|  1.0|        0.0|
|   2400.0|     2400.0|        2400.0|   15.96|      84.33|   12252.0| 8.72|        0.0|
|  10000.0|    10000.0|       10000.0|   13.49|     339.31|   49200.0| 20.0|        0.0|
|   5000.0|     5000.0|        5000.0|    7.9|     156.46|   36000.0| 11.2|        0.0|
+---------+-----------+--------------+--------+-----------+----------+-----+-----------+-----
```

```
only showing top 5 rows
```

0.0.28 PCA Test

```
In [40]: pca = PCA(k=3, inputCol="features", outputCol="pcaFeatures")
         pca_Model = pca.fit(loanDFTransformed_2)

         result = pca_Model.transform(loanDFTransformed_2)
         result.show(truncate=False)

         loanDFTransformed_2 = result
```

```
+---------+-----------+---------------+--------+-----------+----------+-----+-----------+-----
|loan_amnt|funded_amnt|funded_amnt_inv|int_rate|installment|annual_inc|dti  |delinq_2yrs|inq_la
+---------+-----------+---------------+--------+-----------+----------+-----+-----------+-----
|5000.0   |5000.0     |4975.0         |10.65   |162.87     |24000.0   |27.65|0.0        |1.0
|2500.0   |2500.0     |2500.0         |15.27   |59.83      |30000.0   |1.0  |0.0        |5.0
|2400.0   |2400.0     |2400.0         |15.96   |84.33      |12252.0   |8.72 |0.0        |2.0
|10000.0  |10000.0    |10000.0        |13.49   |339.31     |49200.0   |20.0 |0.0        |1.0
|5000.0   |5000.0     |5000.0         |7.9     |156.46     |36000.0   |11.2 |0.0        |3.0
|3000.0   |3000.0     |3000.0         |18.64   |109.43     |48000.0   |5.35 |0.0        |2.0
|5600.0   |5600.0     |5600.0         |21.28   |152.39     |40000.0   |5.55 |0.0        |2.0
|5375.0   |5375.0     |5350.0         |12.69   |121.45     |15000.0   |18.08|0.0        |0.0
|6500.0   |6500.0     |6500.0         |14.65   |153.45     |72000.0   |16.12|0.0        |2.0
|12000.0  |12000.0    |12000.0        |12.69   |402.54     |75000.0   |10.78|0.0        |0.0
|9000.0   |9000.0     |9000.0         |13.49   |305.38     |30000.0   |10.08|0.0        |1.0
|3000.0   |3000.0     |3000.0         |9.91    |96.68      |15000.0   |12.56|0.0        |2.0
|10000.0  |10000.0    |10000.0        |10.65   |325.74     |100000.0  |7.06 |0.0        |2.0
|1000.0   |1000.0     |1000.0         |16.29   |35.31      |28000.0   |20.31|0.0        |1.0
|10000.0  |10000.0    |10000.0        |15.27   |347.98     |42000.0   |18.6 |0.0        |2.0
|3600.0   |3600.0     |3600.0         |6.03    |109.57     |110000.0  |10.52|0.0        |0.0
|6000.0   |6000.0     |6000.0         |11.71   |198.46     |84000.0   |18.44|2.0        |0.0
|9200.0   |9200.0     |9200.0         |6.03    |280.01     |77385.19  |9.86 |0.0        |0.0
|20250.0  |20250.0    |19142.16       |15.27   |484.63     |43370.0   |26.53|0.0        |3.0
|21000.0  |21000.0    |21000.0        |12.42   |701.73     |105000.0  |13.22|0.0        |0.0
+---------+-----------+---------------+--------+-----------+----------+-----+-----------+-----
only showing top 20 rows
```

0.0.29 Train/Test split based on the hardcoded seed value (70 : 30)

```
In [41]: # Creating Training and Test set (70% , 30%)
         (trainingSetDF, testSetDF) = loanDFTransformed_2.randomSplit([0.7, 0.3], 1395)

         trainingSetDF.cache()
```

```
testSetDF.cache()

#print(transformedData.count())
#print(trainingSetDF.count())
#print(testSetDF.count())
```

Out[41]: DataFrame[loan_amnt: float, funded_amnt: float, funded_amnt_inv: float, int_rate: floa

0.0.30 Method to compute the model evaluation matrix

In [48]:
```
def getEvaluationMatrix(predicDF):
    lablePrediction = predicDF.select( "label", "prediction")
    lablePrediction.cache()
    totalCount = lablePrediction.count()
    correctCount = lablePrediction.filter(col("label") == col("prediction")).count()
    wrongCount = lablePrediction.filter(~(col("label") == col("prediction"))).count()
    trueP = lablePrediction.filter(col("label") == 0.0).filter(col("label") == col("pr
    trueN = lablePrediction.filter(col("label") == 1.0).filter(col("label") == col("pr
    falseN = lablePrediction.filter(col("label") == 1.0).filter(~(col("label") == col
    falseP = lablePrediction.filter(col("label") == 0.0).filter(~(col("label") == col

    ratioWrong = float(wrongCount) / float(totalCount)
    ratioCorrect = float(correctCount)/ float(totalCount)

    print("totalCount   - ", totalCount)
    print("correctCount - ", correctCount)
    print("wrongCount   - ", wrongCount)
    print("trueP        - ", trueP)
    print("trueN        - ", trueN)
    print("falseN       - ", falseN)
    print("falseP       - ", falseP)
    print("ratioWrong   - ", ratioWrong)
    print("ratioCorrect - ", ratioCorrect)

    precision = ((float(trueP) / (float(trueP) + float(falseP))) * 100 )
    recall = ((float(trueP) / (float(trueP) + float(falseN))) * 100 )
    print("Accuracy     - ", (trueP + trueN) / totalCount)
    print("Precision    - ", precision)
    print("Recall       - ", recall)
    print("F-1 Score    - ", ((2* ( (precision*recall) / (precision + recall))) ))
    print("Sensitivity  - ", ((float(trueP) / (float(trueP) + float(falseN))) * 100 ))
    print("Specificity  - ", ((float(trueN) / (float(trueN) + float(falseP))) * 100 ))

    createROC(predictions)
```

0.0.31 Method to compute the ROC Curve

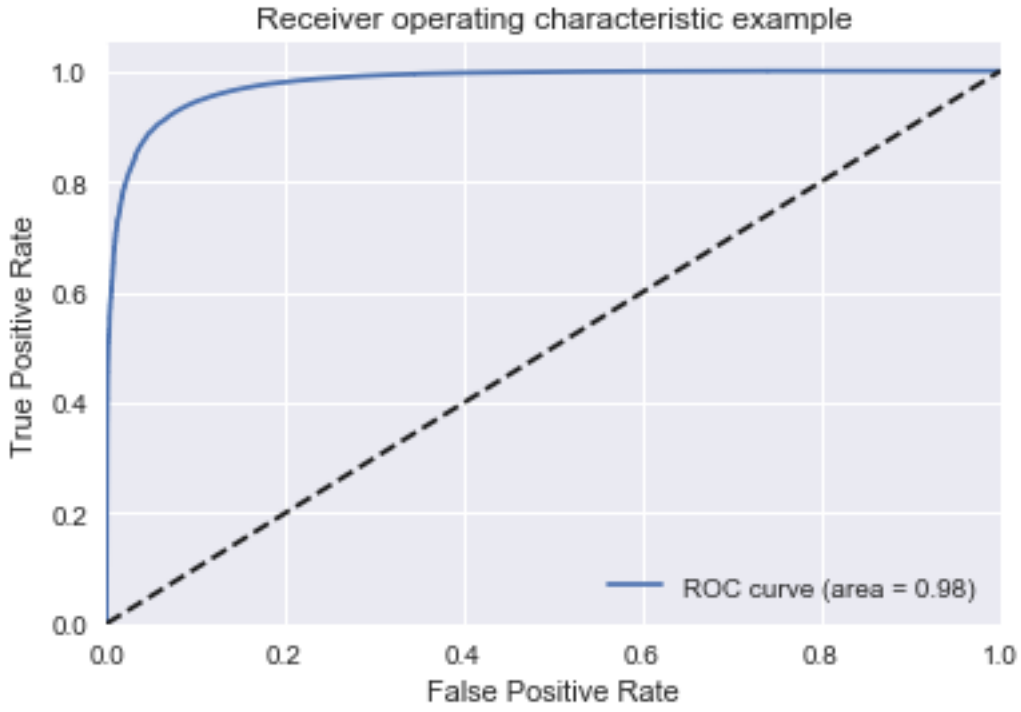
```
In [43]: def createROC(predictions):
             results = predictions.select(['probability', 'label'])

             ## prepare score-label set
             results_collect = results.collect()
             results_list = [(float(i[0][0]), 1.0-float(i[1])) for i in results_collect]
             scoreAndLabels = spark.sparkContext.parallelize(results_list)

             bcMetrics = BinaryClassificationMetrics(scoreAndLabels)
             print("ROC score is - ", bcMetrics.areaUnderROC)

             fpr = dict()
             tpr = dict()
             roc_auc = dict()

             y_test = [i[1] for i in results_list]
             y_score = [i[0] for i in results_list]

             fpr, tpr, _ = roc_curve(y_test, y_score)
             roc_auc = auc(fpr, tpr)

             plt.figure()
             plt.plot(fpr, tpr, label='ROC curve (area = %0.2f)' % roc_auc)
             plt.plot([0, 1], [0, 1], 'k--')
             plt.xlim([0.0, 1.0])
             plt.ylim([0.0, 1.05])
             plt.xlabel('False Positive Rate')
             plt.ylabel('True Positive Rate')
             plt.title('Receiver operating characteristic example')
             plt.legend(loc="lower right")
             plt.show()
```

0.0.32 Logistic Regression Classifier

```
In [49]: print("**** Running Logistic Regression Classifier with best parameter found using ML

         # Create initial LogisticRegression model
         lr_classifier = LogisticRegression(labelCol="label", featuresCol="features", maxIter=
         #lr_classifier = LogisticRegression(labelCol="label", featuresCol="pcaFeatures", maxI

         # Train model with Training Data
         lrModel = lr_classifier.fit(trainingSetDF)

         # Make predictions on test data using the transform() method.
         # LogisticRegression.transform() will only use the 'features' column.
         predictions = lrModel.transform(testSetDF)
```

```
        # Evaluate model
        evaluator = BinaryClassificationEvaluator( labelCol="label")
        lr_accuracy = evaluator.evaluate(predictions)

        #print(lr_accuracy)

        getEvaluationMatrix(predictions)
```

```
**** Running Logistic Regression Classifier with best parameter found using ML pipeline ****
totalCount    -  264025
correctCount  -  251938
wrongCount    -  12087
trueP         -  236174
trueN         -  15764
falseN        -  2375
falseP        -  9712
ratioWrong    -  0.045779755704952185
ratioCorrect  -  0.9542202442950478
Accuracy      -  0.9542202442950478
Precision     -  96.05020212618855
Recall        -  99.00439741939812
F-1 Score     -  97.50492842176969
Sensitivity   -  99.00439741939812
Specificity   -  61.87784581566965
ROC score is  -  0.9789662436081351
```



0.1 DO NOT RUN THIS, IT WILL TAKE 30-45 MINS TO RUN

0.1.1 Logistic Regression Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

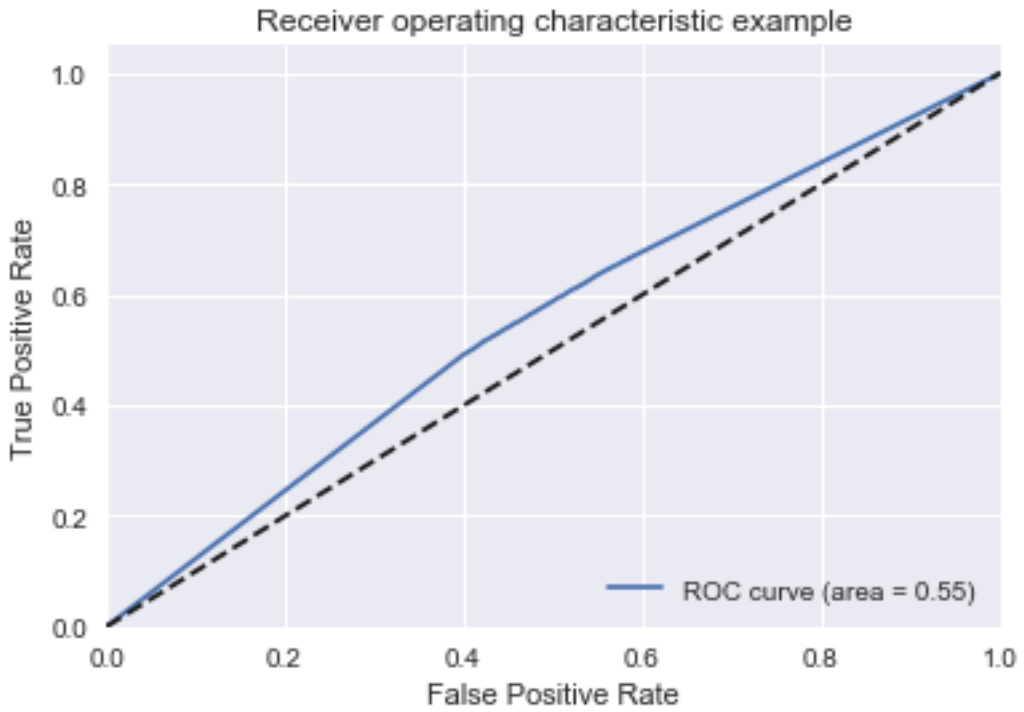
In [99]:
```
#paramGrid = ParamGridBuilder().addGrid(lr_classifier.regParam, [0.01, 0.1, 1.0]).add

#pipeline = Pipeline(stages=[ lr_classifier ])

#evaluator = BinaryClassificationEvaluator( labelCol = "label" )

#crossval_lr = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid,

# Run cross-validation, and choose the best set of parameters.
#cvModel_lr = crossval_lr.fit( trainingSetDF )

#cvLR_predictions = cvModel_lr.transform(testSetDF)
#cvLR_accuracy = evaluator.evaluate(cvLR_predictions)

#bestModel = cvModel_lr.bestModel
#print(cvModel_lr.avgMetrics)
#print(list(zip(cvModel_lr.avgMetrics, paramGrid)))

#print(bestModel.stages[0]._java_obj.getRegParam())
#print(bestModel.stages[0]._java_obj.getElasticNetParam())
#print(bestModel.stages[0]._java_obj.getMaxIter())

#print(cvLR_accuracy)

#getEvaluationMatrix(cvLR_predictions)
```

0.1.2 NaiveBayes Classifier

In [50]:
```
print("**** Running NaiveBayes Classifier with best parameter found using ML pipeline
# Create initial NaiveBayes model
nb_classifier = NaiveBayes(labelCol="label", featuresCol="features", smoothing=50, we
#nb_classifier = NaiveBayes(labelCol="label", featuresCol="pcaFeatures", smoothing=50

# Train model with Training Data
nbModel = nb_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# NaiveBayes.transform() will only use the 'features' column.
predictions = nbModel.transform(testSetDF)
```

```
        # Evaluate model
        evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="predi
        nb_accuracy = evaluator.evaluate(predictions)

        #print(nb_accuracy)

        getEvaluationMatrix(predictions)
```

```
**** Running NaiveBayes Classifier with best parameter found using ML pipeline ****
totalCount    -  264025
correctCount  -  133578
wrongCount    -  130447
trueP         -  122845
trueN         -  10733
falseN        -  7406
falseP        -  123041
ratioWrong    -  0.4940706372502604
ratioCorrect  -  0.5059293627497397
Accuracy      -  0.5059293627497397
Precision     -  49.96014413183345
Recall        -  94.31405517040177
F-1 Score     -  65.31928526042374
Sensitivity   -  94.31405517040177
Specificity   -  8.02323321422698
ROC score is  -  0.5515395015887257
```



0.2 DO NOT RUN THIS, IT WILL TAKE 30-45 MINS TO RUN

0.2.1 NaiveBayes Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

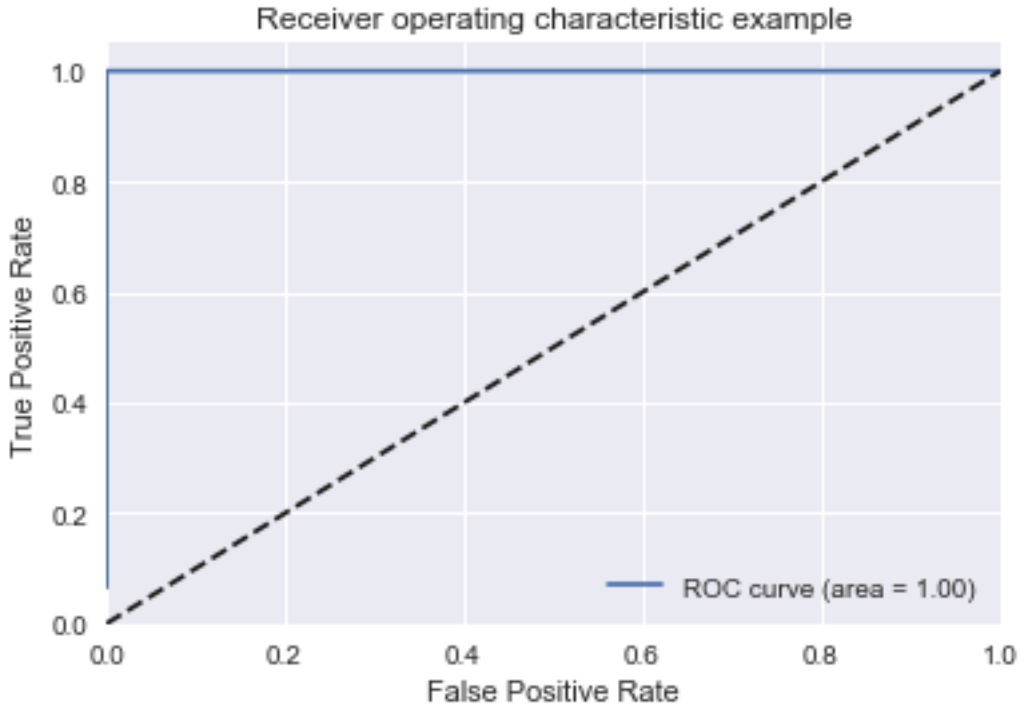
In [101]:
```
#paramGrid = ParamGridBuilder().addGrid(nb_classifier.smoothing, [1.0, 2.0, 3.0])).b

#pipeline = Pipeline(stages=[ nb_classifier ])

#evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="pre

#crossval_nb = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid,

# Run cross-validation, and choose the best set of parameters.
#cvModel_nb = crossval_nb.fit( trainingSetDF )

#cvNB_predictions = cvModel_nb.transform(testSetDF)
#cvNB_accuracy = evaluator.evaluate(cvNB_predictions)

#bestModel = cvModel_nb.bestModel
#print(cvModel_nb.avgMetrics)
#print(list(zip(cvModel_nb.avgMetrics, paramGrid)))

#print(bestModel.stages[0]._java_obj.getSmoothing())

#print(cvNB_accuracy)

#getEvaluationMatrix(cvNB_predictions)
```

0.2.2 Random Forest Classifier

In [51]:
```
# Create initial Random Forest Classifier model
print("**** Running Random Forest Classifier with best parameter found using ML pipeli
rf_classifier = RandomForestClassifier( impurity="gini", maxDepth=12, numTrees=10, fea

# Train model with Training Data
rf_model = rf_classifier.fit(trainingSetDF)

# Print the Forest tree rules.
#rf_model.toDebugString

# Make predictions on test data using the transform() method.
# RandomForest.transform() will only use the 'features' column.
predictions = rf_model.transform(testSetDF)
```

```
        #predictions.show(5)

        evaluator = BinaryClassificationEvaluator( labelCol = "label" )
        rf_accuracy = evaluator.evaluate(predictions)

        #print("accuracy - ", rf_accuracy)

        getEvaluationMatrix(predictions)
```

```
**** Running Random Forest Classifier with best parameter found using ML pipeline ****
totalCount    -  264025
correctCount  -  263981
wrongCount    -  44
trueP         -  245886
trueN         -  18095
falseN        -  44
falseP        -  0
ratioWrong    -  0.00016665088533282833
ratioCorrect  -  0.9998333491146671
Accuracy      -  0.9998333491146671
Precision     -  100.0
Recall        -  99.98210873012646
F-1 Score     -  99.99105356474779
Sensitivity   -  99.98210873012646
Specificity   -  100.0
ROC score is  -  1.0
```



0.3 DO NOT RUN THIS, IT WILL TAKE 30-45 MINS TO RUN

0.3.1 Random Forest Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

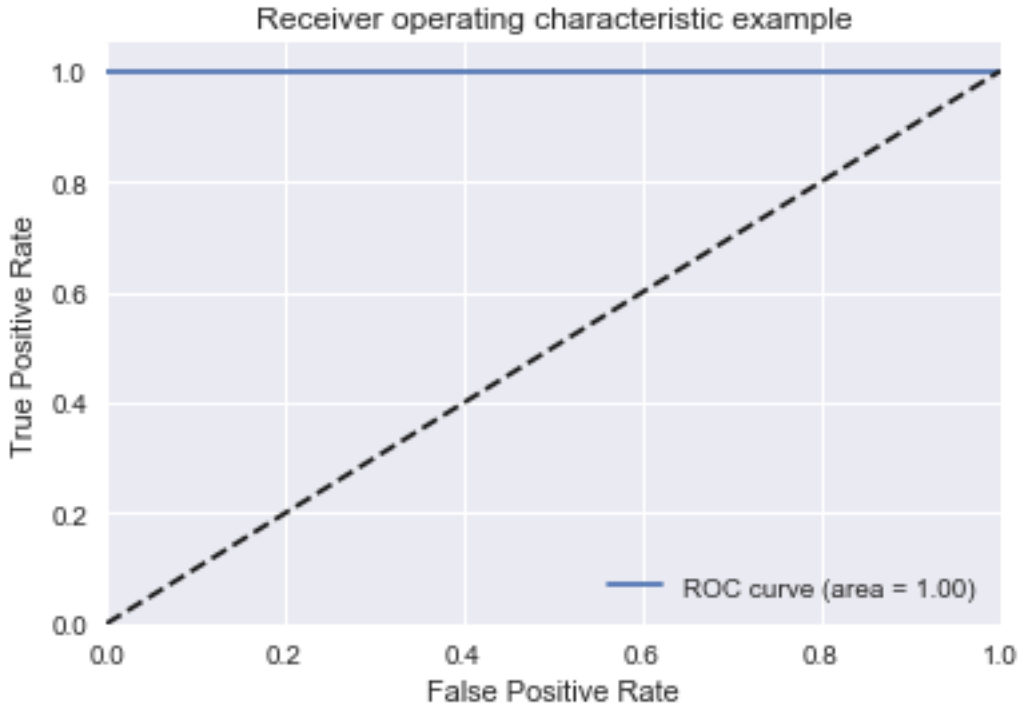
In [97]:
```
#paramGrid = ParamGridBuilder().addGrid(rf_classifier.maxBins, [25, 28, 31, 34]).addG

#pipeline = Pipeline(stages=[ rf_classifier ])

#evaluator = BinaryClassificationEvaluator( labelCol = "label" )

#crossval = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid, eva

# Run cross-validation, and choose the best set of parameters.
#cvModel = crossval.fit( trainingSetDF )

#cv_predictions = cvModel.transform(testSetDF)
#cv_accuracy = evaluator.evaluate(cv_predictions)

#bestModel = cvModel.bestModel
#print(cvModel.avgMetrics)
#print(list(zip(cvModel.avgMetrics, paramGrid)))

#print(bestModel.stages[0]._java_obj.getMaxBins())
#print(bestModel.stages[0]._java_obj.getMaxDepth())
#print(bestModel.stages[0]._java_obj.getImpurity())

#print(cv_accuracy)

#getEvaluationMatrix(cv_predictions)
```

0.3.2 Gradient Boosting Classifier

In [54]:
```
print("**** Running Gradient Boosting Classifier with best parameter found using ML p

# Create initial Gradient Boosting Classifier model
gb_classifier = GBTClassifier(labelCol="label", featuresCol="features", maxDepth=5, ma
#gb_classifier = GBTClassifier(labelCol="label", featuresCol="pcaFeatures", maxDepth=

# Train model with Training Data
gbModel = gb_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# NaiveBayes.transform() will only use the 'features' column.
predictions = gbModel.transform(testSetDF)
```

```
        # Evaluate model
        evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="predi
        gb_accuracy = evaluator.evaluate(predictions)

        #print(gb_accuracy)

        getEvaluationMatrix(predictions)
```

```
**** Running Gradient Boosting Classifier with best parameter found using ML pipeline ****
totalCount    -  264025
correctCount  -  264025
wrongCount    -  0
trueP         -  245886
trueN         -  18139
falseN        -  0
falseP        -  0
ratioWrong    -  0.0
ratioCorrect  -  1.0
Accuracy      -  1.0
Precision     -  100.0
Recall        -  100.0
F-1 Score     -  100.0
Sensitivity   -  100.0
Specificity   -  100.0
ROC score is  -  1.0
```



0.4 DO NOT RUN THIS, IT WILL TAKE 30-45 MIN TO RUN

0.4.1 Gradient Boosting Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

In [105]:
```
#paramGrid = ParamGridBuilder().addGrid(gb_classifier.maxDepth, [3, 5, 10]).addGrid(

#pipeline = Pipeline(stages=[ gb_classifier ])

#evaluator = MulticlassClassificationEvaluator( labelCol="label", predictionCol="pre

#crossval_gb = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid,

# Run cross-validation, and choose the best set of parameters.
#cvModel_gb = crossval_gb.fit( trainingSetDF )

#cvGB_predictions = cvModel_gb.transform(testSetDF)
#cvGB_accuracy = evaluator.evaluate(cvGB_predictions)

#bestModel = cvModel_gb.bestModel
#print(cvModel_gb.avgMetrics)
#print(list(zip(cvModel_gb.avgMetrics, paramGrid)))

#print(bestModel.stages[0]._java_obj.getMaxDepth())
#print(bestModel.stages[0]._java_obj.getMaxIter())
#print(bestModel.stages[0]._java_obj.getStepSize())

#print(cvGB_accuracy)

#getEvaluationMatrix(cvGB_predictions)
```

0.5 THIS NEEDS TO BE FIXED, ITS THROWING SOME EXCEPTION

0.5.1 SVM Classifier

In [102]:
```
# Create initial SVM model
print("**** Running SVM Classifier with best parameter found using ML pipeline **** "
#svm_classifier = LinearSVC(labelCol="label", featuresCol="features", maxIter=50, re
#svm_classifier = LinearSVC(labelCol="label", featuresCol="pcaFeatures", maxIter=50,

# Train model with Training Data
#svmModel = svm_classifier.fit(trainingSetDF)

# Make predictions on test data using the transform() method.
# LogisticRegression.transform() will only use the 'features' column.
#predictions = svmModel.transform(testSetDF)
```

```
# Evaluate model
#evaluator = BinaryClassificationEvaluator( labelCol="label")
#svm_accuracy = evaluator.evaluate(predictions)

#print(svm_accuracy)

#getEvaluationMatrix(predictions)
```

```
**** Running SVM Classifier with best parameter found using ML pipeline ****
```

0.6 DO NOT RUN THIS, IT WILL TAKE 30-45 MINS TO RUN

0.6.1 SVM Classifier with ML Pipeline to find the best hyper parameters Using Cross Validation

In [103]:
```
#paramGrid = ParamGridBuilder().addGrid(svm_classifier.regParam, [0.01, 0.1, 1.0]).a

#pipeline = Pipeline(stages=[ svm_classifier ])

#evaluator = BinaryClassificationEvaluator( labelCol = "label" )

#crossval_svm = CrossValidator( estimator = pipeline, estimatorParamMaps = paramGrid

# Run cross-validation, and choose the best set of parameters.
#cvModel_svm = crossval_svm.fit( trainingSetDF )

#cvSVM_predictions = cvModel_svm.transform(testSetDF)
#cvSVM_accuracy = evaluator.evaluate(cvSVM_predictions)

#bestModel = cvModel_svm.bestModel
#print(cvModel_svm.avgMetrics)
#print(list(zip(cvModel_svm.avgMetrics, paramGrid)))

#print(bestModel.stages[0]._java_obj.getRegParam())
#print(bestModel.stages[0]._java_obj.getMaxIter())
#print(bestModel.stages[0]._java_obj.getSmoothing())

#print(cvSVM_accuracy)

#getEvaluationMatrix(cvSVM_predictions)
```