2. Milestone 2: Data Preprocessing

- Objective of Project
  - Accurately determine which applicants to approve or not. Fully Paid(1) applicants are more likely to be approved than Charged Off(0). Lending Club provides historical data of applicants.
- Method of Analysis
  - Classification using Random Forest
- Dependent Variable
  - loan_status: current status of loan (Fully Paid = 1, Charged Off = 0)
- Independent Variables
  - loan_amnt:amount loaned
  - term: loan term in months
  - int_rate: interest rate
  - installment: monthly payments
  - grade: loan grade based on Lending Club
  - sub_grade: subgrade of loan
  - emp_title: job title
  - emp_length: how long applicant has been working on current job
  - home_ownership: home ownership status
  - annual_inc: applicant annual income
  - verification_status: if income was verified by Lending Club
  - issue_d: month loan was issued
  - purpose: free response reason why applicant wants a loan
  - title: free response title by applicant
  - zip_code: zipcode of applicant
  - dti: A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
  - earliest_cr_line: The month the borrower's earliest reported credit line was opened
  - open_acc: Number of open credit lines applicant currently has
  - pub_rec: Number of derogatory public records
  - revol_bal: Total credit revolving balance
  - revol_util: Revolving line utilization rate
  - total_acc: The total number of credit lines currently in the borrower's credit file
  - initial_list_status: The initial listing status of the loan. Possible values are – W, F (Whole or Fractional)
  - application_type: indicates whether the loan is an individual application or a joint application with two co-borrowers
  - mort_acc: Number of mortgage accounts
  - pub_rec_bankruptcies: Number of public record bankruptcies
  - address: street address of applicant
- These are the original variables from the dataset and we might remove some during preprocessing

Importing Packages

```
#!pip install pyspark # installs spark

#from pyspark.sql import SparkSession # import spark
# #spark = SparkSession.builder\
#         .master("local")\
#         .appName("Colab")\
#         .config('spark.ui.port', '4050')\
#         .getOrCreate()
# spark

# For Milestone 2
import matplotlib.pyplot as plt
import numpy as np
```

```
import pyspark.sql.functions as fn
from pyspark.sql.functions import udf  # user defined function
from pyspark.sql.types import *  # for replace string with Integer
from pyspark.ml.feature import Imputer  # Imputing Missing Data
from pyspark.sql.functions import to_date
from pyspark.sql.functions import from_unixtime, unix_timestamp, col
```

```
VBox()
Starting Spark application
```

ID	YARN Application ID	Kind	State	Spark UI	Driver log	Current session?
0	application_1627863988006_0001	pyspark	idle	Link	Link	✓

```
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
SparkSession available as 'spark'.
FloatProgress(value=0.0, bar_style='info', description='Progress:'
```

2.1 Read Original CSV

```
#!gdown --id 1ORlgCq9W1V2tjuxH03RchBB42VaqxWjn # Downloads original dataset

# see if we read the file as df

#spark_path = 's3://gba6430niki/Project/lending_club_loan_two.csv'
#stan_path = '/content/drive/MyDrive/Colab Notebooks/lending_club_loan_two.csv' # path on my google drive
#path = '/content/lending_club_loan_two.csv'

spark_path = 's3://gba6430niki/Project/lending_club_loan_two.csv'

loan_df = spark.read.csv(spark_path, header=True, inferSchema=True)
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
792060
```

- The number of row still seems to be wrong

```
loan_df.show(5)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+-----------+----------+--------+-----------+-----+---------+---------------+-----
|  loan_amnt|      term|int_rate|installment|grade|sub_grade|      emp_title|emp_l
+-----------+----------+--------+-----------+-----+---------+---------------+-----
|    10000.0| 36 months|   11.44|     329.48|    B|       B4|      Marketing| 10+
|Mendozaberg| OK 22690"|    null|       null| null|     null|           null|
|     8000.0| 36 months|   11.99|     265.68|    B|       B5|Credit analyst |   4
| Loganmouth| SD 05113"|    null|       null| null|     null|           null|
|    15600.0| 36 months|   10.49|     506.97|    B|       B3|   Statistician|  < 1
+-----------+----------+--------+-----------+-----+---------+---------------+-----
only showing top 5 rows
```

2.2 Changing Dependent Variable

```
# loan_status to be binary where 'Charged Off' = 0 and 'Fully Paid' = 1.

loan_df = loan_df.replace(['Fully Paid', 'Charged Off'], ['1', '0'],
                          'loan_status')

loan_df = loan_df.withColumn("loan_status", loan_df["loan_status"].cast(
    IntegerType()))  # Change string to integer
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

2.3 Fixing Duplicated Rows

- When we were trying to drop the missing values, we found that there are a lot of empty rows. And it's why the rows number is different than what it supposed to be.

```
(spark.createDataFrame(
    loan_df.rdd.map(lambda row: (row['address'], sum([c == None for c in row]))
                ).filter(lambda el: el[1] >= 1).collect(),
    ['address', 'CountMissing']).orderBy('CountMissing',
                                         ascending=False).show())
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+-------+------------+
|address|CountMissing|
+-------+------------+
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
|   null|          26|
+-------+------------+
only showing top 20 rows
```

- Upon checking the data through, there is a line break on the address column so it creates a extra row which places half of the address value in the first column loan_amnt.
- Then the rest of the columns of that extra row are null
- We will remove those extra rows

2.4 Drop the Rows that have 20 nulls or greater

```
#drop rows that have less than thresh NON-NULL values. 20-null
loan_df = loan_df.dropna(thresh=20)
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
396030
```

```
#check again
#the number of missing values has the maximum of 4

(spark.createDataFrame(
    loan_df.rdd.map(lambda row: (row['address'], sum([c == None for c in row]))
                ).filter(lambda el: el[1] >= 1).collect(),
    ['address', 'CountMissing']).orderBy('CountMissing',
                                         ascending=False).show())
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+--------------------+------------+
|             address|CountMissing|
+--------------------+------------+
|      788 Baker Wall|           4|
|20552 Wilson Junc...|           4|
|20591 Kerry Islan...|           4|
|41972 Amy Mission...|           4|
|    83789 Cook Haven|           4|
|0669 Rachel Union...|           3|
|60164 Daniels Cre...|           3|
|  03833 Benson Drive|           3|
|  76791 Emma Terrace|           3|
```

```
loan_df.show(10)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+---------+----------+--------+-----------+-----+---------+--------------------+--
|loan_amnt|      term|int_rate|installment|grade|sub_grade|           emp_title|em
+---------+----------+--------+-----------+-----+---------+--------------------+--
|  10000.0| 36 months|   11.44|     329.48|    B|       B4|           Marketing| 1
|   8000.0| 36 months|   11.99|     265.68|    B|       B5|     Credit analyst |
|  15600.0| 36 months|   10.49|     506.97|    B|       B3|        Statistician|
|   7200.0| 36 months|    6.49|     220.65|    A|       A2|     Client Advocate|
|  24375.0| 60 months|   17.27|     609.33|    C|       C5|Destiny Managemen...|
|  20000.0| 36 months|   13.33|     677.07|    C|       C3|       HR Specialist| 1
|  18000.0| 36 months|    5.32|     542.07|    A|       A1|Software Developm...|
|  13000.0| 36 months|   11.14|     426.47|    B|       B2|        Office Depot| 1
|  18900.0| 60 months|   10.99|     410.84|    B|       B3|Application Archi...| 1
|  26300.0| 36 months|   16.29|      928.4|    C|       C5|  Regado Biosciences|
+---------+----------+--------+-----------+-----+---------+--------------------+--
only showing top 10 rows
```

- Now we have the correct dataset.

2.5 Dropping Duplicates Rows

```
loan_df = loan_df.dropDuplicates()
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
396030
```

- No Duplicates to be found

2.6 Dropping Columns

Dropping these columns first since we wont use it for prediction

```
loan_df.select('emp_title').distinct().count()  # too many classes
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
173106
```

```
loan_df.select('purpose',
               'title').show(10)  # similar column so just drop one of them
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+------------------+------------------+
|           purpose|             title|
+------------------+------------------+
```

```
loan_df = loan_df.drop("emp_title")  # too many classes
loan_df = loan_df.drop("title")  # redudant column
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

2.7 Adding a New Variable, age_of_credit

```
only showing top 10 rows
```

```
# (age_of_credit = issue_d - earliest_cr_line)
split_issue_d = fn.split(loan_df["issue_d"], '-')
loan_df = loan_df.withColumn("issue_year", split_issue_d.getItem(1))

split_earliest_cr_line = fn.split(loan_df["earliest_cr_line"], '-')
loan_df = loan_df.withColumn("earliest_cr_line_year",
                             split_earliest_cr_line.getItem(1))

loan_df = loan_df.withColumn(
    'age_of_credit', (loan_df['issue_year'] - loan_df['earliest_cr_line_year']))

loan_df = loan_df.drop("issue_d")
loan_df = loan_df.drop("issue_year")
loan_df = loan_df.drop("earliest_cr_line")
loan_df = loan_df.drop("earliest_cr_line_year")
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
loan_df.show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+---------+----------+--------+-----------+-----+---------+----------+-----------
|loan_amnt|      term|int_rate|installment|grade|sub_grade|emp_length|home_ownersh
+---------+----------+--------+-----------+-----+---------+----------+-----------
|  18000.0| 36 months|   20.99|     678.06|    E|       E3|   2 years|          RE
|   5000.0| 36 months|     8.9|     158.77|    A|       A5|   2 years|          RE
|  28000.0| 60 months|    8.18|     570.16|    B|       B1|   5 years|      MORTGA
|   8400.0| 36 months|   14.65|     289.76|    C|       C5|   6 years|           O
|   6000.0| 36 months|   13.11|     202.49|    B|       B4|      null|      MORTGA
|   5000.0| 36 months|   14.65|     172.48|    C|       C5| 10+ years|          RE
|   6000.0| 36 months|   13.35|     203.18|    C|       C2| 10+ years|           O
|   5000.0| 36 months|   15.61|     174.83|    D|       D1| 10+ years|      MORTGA
|  19150.0| 36 months|   19.24|     704.29|    E|       E2|    1 year|      MORTGA
|  15000.0| 36 months|   10.99|     491.01|    B|       B4| 10+ years|           O
|  20400.0| 60 months|   10.49|     438.38|    B|       B3|   7 years|          RE
|   7000.0| 36 months|   14.49|     240.92|    C|       C4|   8 years|          RE
|  13200.0| 36 months|   10.64|     429.91|    B|       B2|   5 years|          RE
|   1000.0| 36 months|   13.66|      34.02|    C|       C3|   4 years|          RE
|   7100.0| 36 months|   19.19|     260.95|    E|       E3|  < 1 year|           O
|  10000.0| 36 months|     8.6|     316.14|    A|       A4|  < 1 year|      MORTGA
|  20000.0| 36 months|   12.85|     672.44|    B|       B4|   2 years|          RE
|  16000.0| 60 months|   23.63|     456.86|    F|       F3| 10+ years|      MORTGA
|  10075.0| 36 months|   11.14|     330.52|    B|       B2| 10+ years|          RE
|  20000.0| 36 months|   10.99|     654.68|    B|       B2|   8 years|           O
+---------+----------+--------+-----------+-----+---------+----------+-----------
only showing top 20 rows
```

- age_of_credit is the number of years from the applicants earliest credit line to when the applicant submitted the loan application.

Double-click (or enter) to edit

2.8 Converting Schema

```
loan_df.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
root
 |-- loan_amnt: string (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_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: double (nullable = true)
 |-- verification_status: string (nullable = true)
 |-- loan_status: integer (nullable = true)
 |-- purpose: string (nullable = true)
 |-- dti: string (nullable = true)
 |-- open_acc: string (nullable = true)
 |-- pub_rec: double (nullable = true)
 |-- revol_bal: double (nullable = true)
 |-- revol_util: double (nullable = true)
 |-- total_acc: double (nullable = true)
 |-- initial_list_status: string (nullable = true)
 |-- application_type: string (nullable = true)
 |-- mort_acc: string (nullable = true)
 |-- pub_rec_bankruptcies: double (nullable = true)
 |-- address: string (nullable = true)
 |-- age_of_credit: double (nullable = true)
```

```
loan_df = loan_df.withColumn("loan_amnt", loan_df["loan_amnt"].cast(
    IntegerType()))  # string to integer
loan_df = loan_df.withColumn(
    "open_acc", loan_df["open_acc"].cast("double"))  # string to integer
loan_df = loan_df.withColumn("dti",
                             loan_df["dti"].cast("double"))  # string to double
loan_df = loan_df.withColumn(
    "mort_acc", loan_df["mort_acc"].cast("double"))  # string to double
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
loan_df.printSchema()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
root
 |-- loan_amnt: 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_length: string (nullable = true)
 |-- home_ownership: string (nullable = true)
 |-- annual_inc: double (nullable = true)
 |-- verification_status: string (nullable = true)
 |-- loan_status: integer (nullable = true)
 |-- purpose: string (nullable = true)
 |-- dti: double (nullable = true)
 |-- open_acc: double (nullable = true)
 |-- pub_rec: double (nullable = true)
 |-- revol_bal: double (nullable = true)
 |-- revol_util: double (nullable = true)
 |-- total_acc: double (nullable = true)
 |-- initial_list_status: string (nullable = true)
 |-- application_type: string (nullable = true)
 |-- mort_acc: double (nullable = true)
 |-- pub_rec_bankruptcies: double (nullable = true)
 |-- address: string (nullable = true)
 |-- age_of_credit: double (nullable = true)
```

2.9 Dealing with Missing Values

```
(spark.createDataFrame(
    loan_df.rdd.map(lambda row:
                    (row['address'], sum([c == None for c in row])
                    )  #set to address because it's the only unique col.
                   ).filter(lambda el: el[1] >= 1).collect(),
```

```
    ['address', 'CountMissing']).orderBy('CountMissing',
                                         ascending=False).show())
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+--------------------+------------+
|             address|CountMissing|
+--------------------+------------+
|                 0.0|           5|
|20552 Wilson Junc...|           3|
|0644 Harris Ways ...|           3|
|20591 Kerry Islan...|           3|
|      788 Baker Wall|           3|
|    83789 Cook Haven|           3|
|     747 Allen Creek|           2|
|0455 Hernandez Me...|           2|
| 826 Kimberly Greens|           2|
|   468 Martin Valley|           2|
|16119 Kevin Circl...|           2|
|539 Jade Centers ...|           2|
|     538 Ricky Knoll|           2|
|9622 Dominguez St...|           2|
|    13374 Stacy Mews|           2|
|59016 Joshua Expr...|           2|
|10635 Christopher...|           2|
|579 Ramirez Ramp ...|           2|
|       827 Shaw Mill|           2|
|     4699 Cathy Camp|           2|
+--------------------+------------+
only showing top 20 rows
```

```
#let's see how many rows have missing value of 2
loan_df.rdd.map(
           lambda row: (
                 row['address']
               , sum([c == None for c in row])
           )
        )\
        .filter(lambda el: el[1] == 2)\
        .count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
1642
```

```
#check missing values by columns
for k, v in sorted(
        loan_df.agg(*[(1 - (fn.count(c) / fn.count('*'))).alias(c + '_miss')
                      for c in loan_df.columns]).collect()[0].asDict().items(),
        key=lambda el: el[1],
        reverse=True):
    print(k, v)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
mort_acc_miss 0.09543468929121535
emp_length_miss 0.04621114562028128
pub_rec_bankruptcies_miss 0.001353432820745848
revol_util_miss 0.0006969169002348785
dti_miss 2.5250612327720745e-06
open_acc_miss 2.5250612327720745e-06
age_of_credit_miss 2.5250612327720745e-06
loan_amnt_miss 0.0
term_miss 0.0
int_rate_miss 0.0
installment_miss 0.0
grade_miss 0.0
sub_grade_miss 0.0
home_ownership_miss 0.0
annual_inc_miss 0.0
verification_status_miss 0.0
loan_status_miss 0.0
purpose_miss 0.0
pub_rec_miss 0.0
revol_bal_miss 0.0
total_acc_miss 0.0
initial_list_status_miss 0.0
application_type_miss 0.0
address_miss 0.0
```

```
# Finding the Mode
loan_df.groupBy("emp_length").count().show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+----------+------+
|emp_length| count|
+----------+------+
|      null| 18301|
|   3 years| 31665|
|   8 years| 19168|
| 10+ years|126041|
|  < 1 year| 31725|
|   5 years| 26495|
|   4 years| 23952|
|   6 years| 20841|
|    1 year| 25882|
|   9 years| 15314|
|   2 years| 35827|
|   7 years| 20819|
+----------+------+
```

```
# Replace emp_length with mode
loan_df = loan_df.fillna('10+ years', subset='emp_length')
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
loan_df.show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+---------+----------+--------+-----------+-----+---------+----------+------------
|loan_amnt|      term|int_rate|installment|grade|sub_grade|emp_length|home_ownersh
+---------+----------+--------+-----------+-----+---------+----------+------------
|    15000| 36 months|   13.35|     507.95|    C|       C2|  < 1 year|          RE
|    12000| 36 months|   10.16|     388.11|    B|       B1|   2 years|          RE
|     3200| 36 months|   14.33|     109.89|    C|       C1|   3 years|          RE
|    25000| 36 months|    7.62|     779.04|    A|       A3|   7 years|      MORTGA
|     7200| 36 months|   12.69|     241.53|    C|       C2| 10+ years|      MORTGA
|     8825| 36 months|   15.61|     308.57|    D|       D1|    1 year|          RE
|    22250| 36 months|   13.11|     750.87|    B|       B4|   5 years|      MORTGA
|    20000| 36 months|   12.69|      670.9|    C|       C2| 10+ years|      MORTGA
|    19850| 60 months|   18.25|     506.77|    D|       D3| 10+ years|          RE
|     6000| 36 months|   13.98|     205.01|    C|       C3|   7 years|          RE
|    13325| 60 months|    17.1|     331.88|    C|       C5|  < 1 year|      MORTGA
|     6000| 36 months|   12.99|     202.14|    C|       C1| 10+ years|          RE
|    25000| 36 months|     8.9|     793.84|    A|       A5| 10+ years|      MORTGA
|    20000| 36 months|   11.36|     658.23|    B|       B5|    1 year|          RE
|    30000| 36 months|    7.89|     938.57|    A|       A5|   3 years|           O
|     1000| 36 months|    19.2|      36.76|    D|       D3|   3 years|          RE
|     4500| 36 months|   12.12|     149.73|    B|       B3|   4 years|          RE
|    10975| 60 months|   17.76|     277.27|    D|       D1|   3 years|           O
|    13200| 36 months|   10.99|     432.09|    B|       B4| 10+ years|           O
|     9000| 36 months|   14.47|     309.66|    C|       C2| 10+ years|      MORTGA
+---------+----------+--------+-----------+-----+---------+----------+------------
only showing top 20 rows
```

```
# Replace missing data in numerical columns with median https://stackoverflow.com/questions/51680462/replace-null-values-with-median-in-pyspa
# Imputer function from spark can replace numerical columns with mean,median, and mode but only deals with double schema.

# Select missing data numerical columns and set strategy to median
imputer = Imputer(inputCols=[
    'mort_acc', 'pub_rec_bankruptcies', 'revol_util', 'age_of_credit', 'dti',
    'open_acc'
],
                  outputCols=[
                      'mort_acc', 'pub_rec_bankruptcies', 'revol_util',
                      'age_of_credit', 'dti', 'open_acc'
                  ]).setStrategy("median")

# Impute the missing values
loan_df = imputer.fit(loan_df).transform(loan_df)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
# Check missing values after Imputation
for k, v in sorted(
        loan_df.agg(*[(1 - (fn.count(c) / fn.count('*'))).alias(c + '_miss')
                      for c in loan_df.columns]).collect()[0].asDict().items(),
        key=lambda el: el[1],
        reverse=True):
    print(k, v)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
loan_amnt_miss 0.0
term_miss 0.0
int_rate_miss 0.0
installment_miss 0.0
grade_miss 0.0
sub_grade_miss 0.0
emp_length_miss 0.0
home_ownership_miss 0.0
annual_inc_miss 0.0
verification_status_miss 0.0
loan_status_miss 0.0
purpose_miss 0.0
dti_miss 0.0
open_acc_miss 0.0
pub_rec_miss 0.0
revol_bal_miss 0.0
revol_util_miss 0.0
total_acc_miss 0.0
initial_list_status_miss 0.0
application_type_miss 0.0
mort_acc_miss 0.0
pub_rec_bankruptcies_miss 0.0
address_miss 0.0
age_of_credit_miss 0.0
```

- All the missing values have been imputed.

2.10 1st Descriptive Statistics

```
#make a list that contain all the numeric variables
num_features = []

for x, t in loan_df.dtypes:
    if t != "string":
        num_features.append(x)

print(num_features)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```

```
loan_df.describe(num_features).show()  #just to make it easier to view
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+-------+-----------------+-----------------+-----------------+----------------
|summary|         loan_amnt|          int_rate|       installment|       annual_in
+-------+-----------------+-----------------+-----------------+----------------
|  count|           396030|           396030|           396030|           39603
|   mean|14113.888089286165|13.639400045450396|431.84969802792745|74203.1757977173
| stddev| 8357.44134110476| 4.472157381531261|250.72778950372276|61637.6211580923
|    min|              500|             5.32|            16.08|             0.
|    max|            40000|            30.99|           1533.81|        8706582.
+-------+-----------------+-----------------+-----------------+----------------
```

- Obviously there are some outliers

2.11 Correlation Table (need interpretations for 3.2)

```
n_features = len(num_features)
corr = []
```

```
for i in range(0, n_features):
    temp = [None] * i

    for j in range(i, n_features):
        temp.append(loan_df.corr(num_features[i], num_features[j]))
    corr.append([num_features[i]] + temp)

correlations = spark.createDataFrame(corr, ['Column'] + num_features)

correlations.show()

# it sometime caused dead kernel, may just run the next cell
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+--------------------+---------+--------------------+--------------------+----------
|              Column|loan_amnt|           int_rate|        installment|
+--------------------+---------+--------------------+--------------------+----------
|           loan_amnt|      1.0|0.16892108560054628| 0.9539289082616202|  0.336887
|            int_rate|     null|                 1.0|0.16275842448581485|-0.0567707
|         installment|     null|                null|                 1.0| 0.3303808
|          annual_inc|     null|                null|                null|
|         loan_status|     null|                null|                null|
|                 dti|     null|                null|                null|
|            open_acc|     null|                null|                null|
|             pub_rec|     null|                null|                null|
|           revol_bal|     null|                null|                null|
|          revol_util|     null|                null|                null|
|           total_acc|     null|                null|                null|
|            mort_acc|     null|                null|                null|
|pub_rec_bankruptcies|     null|                null|                null|
|       age_of_credit|     null|                null|                null|
+--------------------+---------+--------------------+--------------------+----------
```

```
loan_df.corr('loan_amnt', 'installment')
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
0.9539289082616202
```

```
loan_df.corr('pub_rec', 'pub_rec_bankruptcies')
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
0.6990611182813461
```

```
loan_df.corr('open_acc', 'total_acc') #less than 0.8 is fine
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
0.6807283567887777
```

```
# Drop the Column that is highly correlated
loan_df = loan_df.drop("installment")
num_features.remove("installment")

# Remove Address Column as it is just unique id
loan_df = loan_df.drop("address")
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

2.12 Visualizations

a. Bar Plots

```
# %%spark -o loan_df
from pyspark.sql.functions import asc
import matplotlib
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:'
```

i. Let's check the distribution of the dataset first - the dependent variable (loan_status)

```
loan_status = loan_df.groupBy("loan_status").count()
loan_status.toPandas().set_index('loan_status').plot(kind='bar', figsize=(6, 3))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='loan_status'>
```

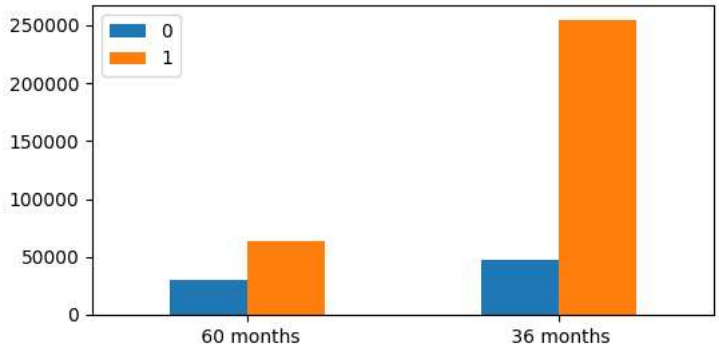
- The barplot indicates that it's an imbalance dataset, therefore, we might do tree classification which is better deal with imbalance data instead of logistic regression.

ii. term vs. loan_status

```
loan_df_term = loan_df \
.groupBy("term") \
.pivot("loan_status") \
.count()

loan_df_term.toPandas().set_index('term').plot(kind='bar', figsize=(6, 3))
plt.xticks(rotation=0)
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```




```
loan_df_term2 = loan_df.groupBy("term").agg(fn.mean('loan_amnt'))
loan_df_term2.toPandas().set_index('term').plot(kind='bar', figsize=(4, 2))
plt.xticks(rotation=0)
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
(array([0, 1]), [Text(0, 0, ' 60 months'), Text(1, 0, ' 36 months')])
```

- These 2 plots show that higher term (60m) has the higher loan amount and default rate.

iii. home_ownership vs. loan_status

```
loan_df_home_ownership = loan_df \
.groupBy("home_ownership") \
.pivot("loan_status") \
.count()

loan_df_home_ownership.toPandas().set_index('home_ownership').plot(kind='bar',
                                                                   figsize=(6,
                                                                            3))
plt.xticks(rotation=0)
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:'
```

- Client who rent house has higher default rate.

iv. verification_status vs. loan_status

```
loan_df_verification_status = loan_df \
.groupBy("verification_status") \
.pivot("loan_status") \
.count()

loan_df_verification_status.toPandas().set_index('verification_status').plot(
    kind='bar', figsize=(6, 3))
plt.xticks(rotation=0)
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

- Seems like having income verified doesnt mean that they have higher chance to pay off their loan. (Q: drop?)

v. purpose vs loan_status

```
loan_df_purpose = loan_df.groupBy("purpose").pivot("loan_status").count()
loan_df_purpose.toPandas().set_index('purpose').plot(kind='barh',
                                                     figsize=(8, 4))
# %matplot plt
```

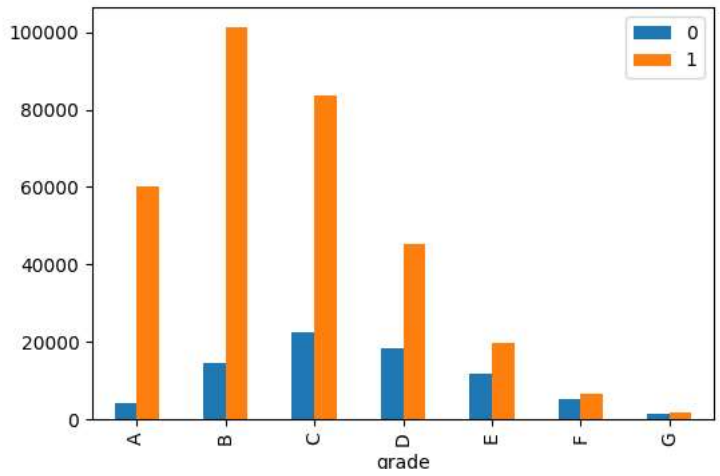
```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:ylabel='purpose'>
```

- Most of the cases are for debt consolidation

vi. Grade vs subGrade vs loan_status

```
loan_df_grade = loan_df.sort("grade").groupBy("grade").pivot(
    "loan_status").count()
loan_df_grade.toPandas().set_index('grade').plot(kind='bar', figsize=(6, 4))
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```



```
loan_df_sub_grade = loan_df.sort("sub_grade").groupBy("sub_grade").pivot("loan_status").count()
loan_df_sub_grade.toPandas().set_index('sub_grade').plot(kind='bar',figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='sub_grade'>
```

```
# drop sub_grade because too many classes and is similar to grade
loan_df = loan_df.drop("sub_grade")
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

Result:

- Grade (A to G) - A has the highest rate to paid off
- Sub_grade (1 to5) - 1 has the highest rate to paid off

Note: classifying the grade is important for the company

vii. emp_length vs loan_status

```
loan_df_emp_length = loan_df.groupBy("emp_length").pivot("loan_status").count()
loan_df_emp_length.toPandas().set_index('emp_length').plot(kind='barh',
                                                           figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='emp_length'>
```

- Seems like the classes have the similar rate across the loan_status (Q: drop?)

viii. intial_list_status vs. loan_status

```
loan_df_initial_list_status = loan_df.groupBy("initial_list_status").pivot(
    "loan_status").count()
loan_df_initial_list_status.toPandas().set_index('initial_list_status').plot(
    kind='bar', figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='initial_list_status'>
```

ix. application_type vs. loan_status

```
loan_df_application_type = loan_df.groupBy("application_type").pivot(
    "loan_status").count()
loan_df_application_type.toPandas().set_index('application_type').plot(
    kind='barh', figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='application_type'>
```

b. Line Plots

i. grade vs. int_rate

```
loan_df_grade2 = loan_df.sort("grade").groupBy("grade").agg(fn.mean('int_rate'))
loan_df_grade2.toPandas().set_index('grade').plot(kind='line', figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```

- GradeA has the lowest interent rate and that might be the reason why it has the highest paid off rate.

ii. grade vs. loan_amnt

```
loan_df_grade3 = loan_df.sort("grade").groupBy("grade").agg(
    fn.mean('loan_amnt'))
loan_df_grade3.toPandas().set_index('grade').plot(kind='line', figsize=(6, 4))
# %matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
<AxesSubplot:xlabel='grade'>
```

- grade and loan_amnt has linear relationship.

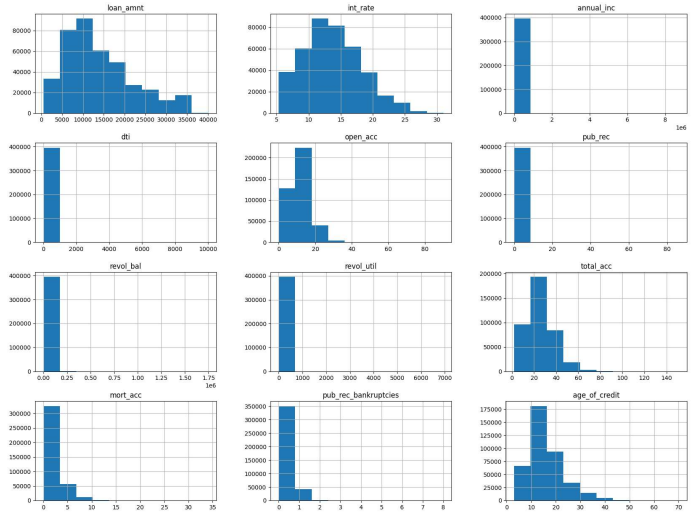
2.13 Remove Outlier (histograms & Scatter)

```
num_features.remove(
    "loan_status")  # removing loan status column before creating historgrams
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
loan_df_numeric = loan_df.select(num_features)
loan_df_numeric.toPandas().hist(figsize=(20, 15))
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```



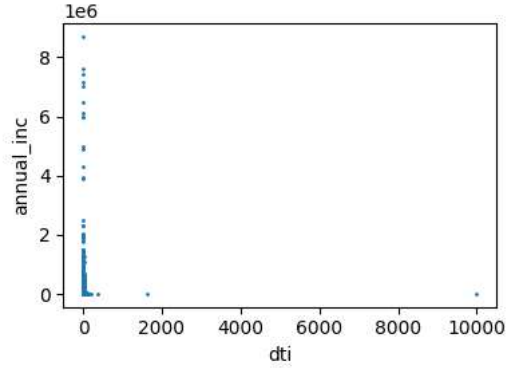
Take a deeper look with scatter Plots

annual_inc & dti

Dti: A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.

```
plt.clf()
scatter1 = loan_df.select('dti', 'annual_inc').toPandas()
plt.figure(figsize=(4, 3))
plt.scatter(scatter1["dti"], scatter1["annual_inc"],s=1)
plt.xlabel("dti")
plt.ylabel("annual_inc")
plt.tight_layout()
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```




```
loan_df = loan_df.filter(fn.col("annual_inc").between(0,750000))

#Investopedia: As a general guideline, 43% is the highest DTI ratio a borrower can have and still get qualified for a mortgage.
loan_df = loan_df.filter(fn.col("dti").between(0,55))
```

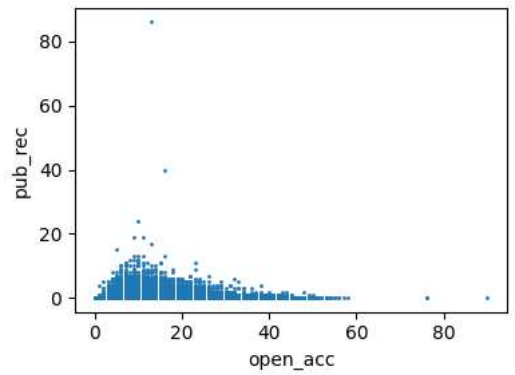
```
loan_df.count()
#396030->395852
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
395852
```

open_acc & pub_rec

```
plt.clf()
scatter2 = loan_df.select('open_acc', 'pub_rec').toPandas()
plt.figure(figsize=(4, 3))
plt.scatter(scatter2["open_acc"], scatter2["pub_rec"],s=1)
plt.xlabel("open_acc")
plt.ylabel("pub_rec")
plt.tight_layout()
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```




```
loan_df = loan_df.filter(fn.col("pub_rec").between(0,8))
loan_df = loan_df.filter(fn.col("open_acc").between(0,60))
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
395802
```

revol_bal & revol_util

revolving line utilization rate: how much you currently owe divided by your credit limit.

```
plt.clf()
scatter3 = loan_df.select('revol_bal', 'revol_util').toPandas()
plt.figure(figsize=(4, 3))
plt.scatter(scatter3["revol_bal"], scatter3["revol_util"],s=1)
plt.xlabel("revol_bal")
plt.ylabel("revol_util")
plt.tight_layout()
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```

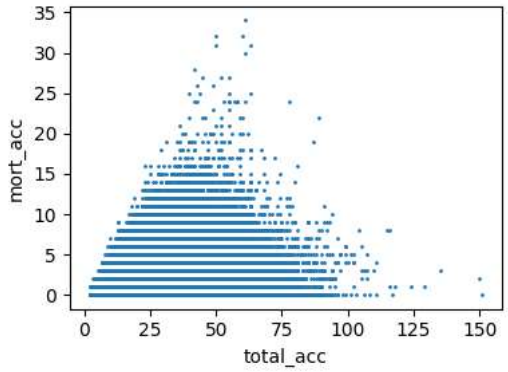
```
#if revol_util > 30% == unhealthy credit score
#set it to 120, because it wont go too much beyond the credit line usually.
loan_df = loan_df.filter(fn.col("revol_util").between(0,120))
loan_df = loan_df.filter(fn.col("revol_bal").between(0,300000))
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
395568
```

total_acc & mort_acc

```
plt.clf()
scatter4 = loan_df.select('total_acc', 'mort_acc').toPandas()
plt.figure(figsize=(4, 3))
plt.scatter(scatter4["total_acc"], scatter4["mort_acc"],s=1)
plt.xlabel("total_acc")
plt.ylabel("mort_acc")
plt.tight_layout()
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```




```
loan_df = loan_df.filter(fn.col("total_acc").between(0,100))
loan_df = loan_df.filter(fn.col("mort_acc").between(0,20))
loan_df.count()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
395499
```

pub_rec_bankruptcies

it is public record bankruptcies

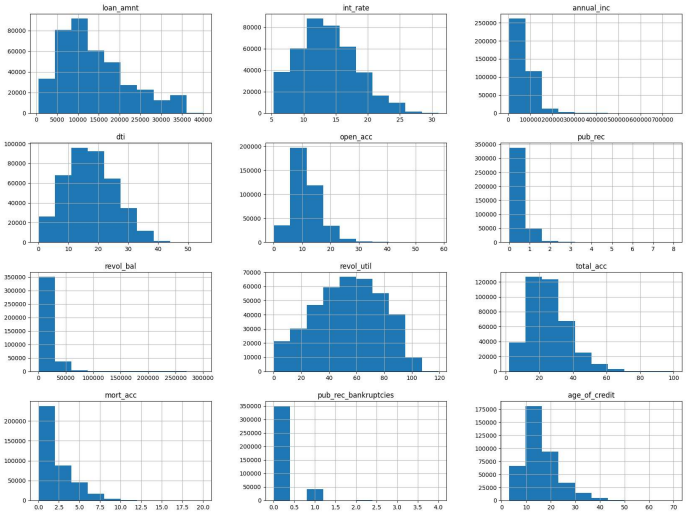
```
loan_df = loan_df.filter(fn.col("pub_rec_bankruptcies").between(0,4))
loan_df.count()
#396020 -> 395456 (dropped 0.14% records)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
395456
```

2.14 Recreate Histograms

```
loan_df_numeric = loan_df.select(num_features)
loan_df_numeric.toPandas().hist(figsize=(20, 15))
%matplot plt
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```



2.15 2nd Descriptive Statistics after removing Outliers

```
loan_df.describe(num_features).show()
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
+-------+-----------------+-----------------+-----------------+------------------
|summary|         loan_amnt|         int_rate|       annual_inc|               dti
+-------+-----------------+-----------------+-----------------+------------------
|  count|           395456|           395456|           395456|            395456
|   mean|14103.801118202784|13.63959947503572| 73526.0503031437|17.350212969331608
| stddev| 8350.090805901567|4.471577758643289|45172.74331883477| 8.117122899275918
|    min|              500|             5.32|           4000.0|               0.0
|    max|            40000|            30.99|         750000.0|             54.96
+-------+-----------------+-----------------+-----------------+------------------
```

2.16 Creating loan_df CSV

```
#loan_df.write.option("header", True).csv(
#    "/Project/loan_df")

loan_df.coalesce(1).write.option("header", True).csv("s3://gba6430niki/Project/loan_dfffffff")

# (Collab Path) Convert to csv so no need to keep re-cleaning the data everytime
# loan_df.write.option("header",True).csv("s3://gba6430stanley/Project/loan_df") # S3 Path
# !zip -r loan_df.zip /content/loan_df #zips folder to be downloaded
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

- The loan_df csv contains all original rows
  - Missing Values Imputed with mode for categorical and median for numerical
  - Installment Column removed as it is highly correlated with loan_amnt
  - New variable age_of_credit created
  - No Outliers have been removed yet
  - No Dummies have been created yet

3. Milestone 3: Modeling

- Modeling Plan
  - Create Numerical Vector using Vector Assembler, Categorical Vector using OneHotEncoder, Chisquared selected Categorical Vector.
  - Merge Numerical Vector and Chisquare selected Categorical Vector
  - Pass vectors through pipeline
  - Pipeline 1 includes:
    - Random Forest
    - Evaluate Model
  - Pipeline 2 includes:
    - Logistic Regression
    - Evaluate Model
  - Grid Search

Importing Packages

```
#!pip install pyspark # installs spark

# from pyspark.sql import SparkSession # import spark
# spark = SparkSession.builder\
#         .master("local")\
#         .appName("Colab")\
#         .config('spark.ui.port', '4050')\
#         .getOrCreate()
# spark
```

```
import pyspark.sql.functions as F
import pyspark.ml.feature as feat # rest of estimators and transformers
import numpy as np
from pyspark.ml import Pipeline # Pipline function
from pyspark.ml.classification import LogisticRegression # Logistic Regression function
from pyspark.ml.evaluation import BinaryClassificationEvaluator # Evaluate Logistc Regression
from pyspark.ml.tuning import ParamGridBuilder, CrossValidator # Tuning
from pyspark.ml.classification import RandomForestClassifier
from sklearn.metrics import classification_report, confusion_matrix
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

3.1 Reading loan_df CSV

```
# !zip -r loan_df.zip /content/loan_df #zips folder to be downloaded
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

```
#!gdown --id 1xirViZA8rjj9U2TwamUFBp6Hum9oZOWV # Downloads cleaned dataset
#!unzip /content/loan_df.zip # unzips the cleand dataset

# s3_path = 's3://gba6430stanley/Project/loan_df'

#loan_df = spark.read.csv('/content/content/loan_df', header=True, inferSchema=True)
loan_df = spark.read.csv('s3://gba6430niki/Project/loan_dfffffff/loan_df.csv', header=True, inferSchema=True)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

3.2 Correlations and their interpretations

- Done in Section 2.11

3.3 Data transformation

Modeling

a. Logisitc Regression

undersampling

```
# Undersample majority class to have balanced dataset

from pyspark.sql.functions import col

major_df = loan_df.filter(col("loan_status") == 1)
minor_df = loan_df.filter(col("loan_status") == 0)

ratio = int(major_df.count() /minor_df.count())
# Ratio is 4, 4 of class 1 to class 0

sampled_majority_df = major_df.sample(withReplacement=False, fraction=1 /ratio)
# Reduce Class 1 by 1/4

loan_df_log = sampled_majority_df.unionAll(minor_df)
# Now data is about 70K Class 1 and 70K Class 0

#loan_df_log = loan_df_log.drop("sub_grade")
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

```
#All Catergorical Features
cat_features = []

for x, t in loan_df_log.dtypes:
    if t == "string" and x != 'loan_status':
        cat_features.append(x)

print(cat_features)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

```
#All Numerical Features
num_features = []

for x, t in loan_df_log.dtypes:
    if t != "string" and x != 'loan_status':
        num_features.append(x)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

Transformer

```
#stages for pipeline
stages = []

for col in cat_features:
    stringIndexer = feat.StringIndexer(inputCol = col, outputCol = col + '_index')
    encoder = feat.OneHotEncoderEstimator(inputCols=[stringIndexer.getOutputCol()], outputCols=[col + "_encoded"])

    stages += [stringIndexer, encoder]

#label_stringIdx = feat.StringIndexer(inputCol = 'loan_status', outputCol = 'label')
#stages += [label_stringIdx]

assemblerInputs = [c + "_encoded" for c in cat_features] + num_features
assembler = feat.VectorAssembler(inputCols=assemblerInputs, outputCol="features")

norm = feat.StandardScaler(
    inputCol=assembler.getOutputCol(), outputCol='signal_norm', withMean=True, withStd=True)

stages += [assembler, norm]
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

pipeline

```
pipeline = Pipeline(stages=stages)

pModel = pipeline.fit(loan_df_log)
data = pModel.transform(loan_df_log)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

split (80% train, 20% test) -> put it into lr model

```
train, test = data.randomSplit([0.8, 0.2], seed=666)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'))
```

```
lr = LogisticRegression(featuresCol = 'features', labelCol = 'loan_status', maxIter=10)
```

```
lrModel = lr.fit(train)
predictions_LR = lrModel.transform(test)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
#see performance
lr_evaluator = BinaryClassificationEvaluator(labelCol='loan_status')
print('AUC: ', lr_evaluator.evaluate(predictions_LR))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
AUC:  0.7087513850113685
```

```
y_true = predictions_LR.select(['loan_status']).collect()
y_pred = predictions_LR.select(['prediction']).collect()
```

```
print(classification_report(y_true, y_pred))
#print(confusion_matrix(y_true, y_pred))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
              precision    recall  f1-score   support

           0       0.64      0.66      0.65     15495
           1       0.66      0.65      0.66     16067

    accuracy                           0.65     31562
   macro avg       0.65      0.65      0.65     31562
weighted avg       0.65      0.65      0.65     31562
```

```
print(confusion_matrix(y_true, y_pred))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
[[10220  5275]
 [ 5664 10201]]
```

i. Grid Search for Logistic Regression

```
loan_df_log_grid = loan_df.sample(fraction=0.1, seed=3)
train, test = loan_df_log_grid.randomSplit([0.8, 0.2], seed=666)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
import pyspark.ml.tuning as tune

logReg_grid = (
    tune.ParamGridBuilder()
    .addGrid(lr.regParam, [0, 0.01, 0.02, 0.03])
    #.addGrid(lr.elasticNetParam, [0.2, 0.25, 0.3])
    .build())

cross_v = tune.CrossValidator(estimator=lr,
                              estimatorParamMaps=logReg_grid,
                              evaluator=lr_evaluator)

pipeline = Pipeline(stages=stages)
data_trans = pipeline.fit(train)

lr_modelTest = cross_v.fit(data_trans.transform(train))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

- elasticNetParam was 0 after .addGrid(logReg_obj.elasticNetParam was added
- .addGrid(logReg_obj.regParam, [0.0, .33, 0.66, 1.0]) = Best params regParam: 1.0, elasticNetParam: 0.0
- .addGrid(lr.regParam, [0.0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.0] = Best params - regParam: 0.8
- .addGrid(lr.regParam, [0.72,0.74,0.76,0.78,0.8,0.82,0.84,0.86] = Best params - regParam: 0.82
- .addGrid(lr.regParam, [0.81,0.82,0.83,0.84] = Best params - regParam: 0.81

Niki

- elasticNetParam was 0 after .addGrid(logReg_obj.elasticNetParam was added (elasticNetParam: 0.33)
- lr.regParam, [0.0, 0.1, 0.2, 0.3] --> bestparams: 0.0
- lr.regParam, [0.0, 0.01, 0.02, 0.03] --> bestparams: 0.02

```
print('Best params - regParam: {}, elasticNetParam: {}'.format(
    lr_modelTest.bestModel._java_obj.getRegParam(),
    lr_modelTest.bestModel._java_obj.getElasticNetParam()))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Best params - regParam: 0.02, elasticNetParam: 0.0
```

```
# measure performance of best model
data_trans_test = data_trans.transform(test)
predictions = lr_modelTest.transform(data_trans_test)

print('AUC:', lr_evaluator.evaluate(predictions))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
AUC: 0.7046105463474336
```

```
# accuracy = predictions.filter(
#     predictions.loan_status == predictions.prediction).count() / float(
#         predictions.count())
# print("Accuracy : ", accuracy)
```

b. Random Forest

```
loan_df_rf = loan_df.sample(fraction=0.1, seed=3)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```

```
#same transformer, same stages
pipeline2 = Pipeline(stages=stages)

loan_df_rf_pip = pipeline2.fit(loan_df_rf).transform(loan_df_rf)

train, test = loan_df_rf_pip.randomSplit([0.8, 0.2], seed=1997)

rf = RandomForestClassifier(labelCol='loan_status',
                            featuresCol='features',
                            minInstancesPerNode=10,
                            numTrees=10)

rfModel = rf.fit(train)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
```

```
#apply the model to test set
predictions = rfModel.transform(test)
rf_ev = BinaryClassificationEvaluator(labelCol='loan_status')

print('AUC:', rf_ev.evaluate(predictions))
#predictions.select('loan_status', 'prediction', 'probability').show(5)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Test Area Under ROC: 0.7111214417803755
```

```
accuracy = predictions.filter(
    predictions.loan_status == predictions.prediction).count() / float(predictions.count())
print("Accuracy : ", accuracy)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Accuracy :  0.7995382839553674
```

i. Grid Search for Random Forest

```
loan_df_rf_grid = loan_df.sample(fraction=0.1, seed=3)
train, test = loan_df_rf_grid.randomSplit([0.8, 0.2], seed=1997)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%')
```

```
import pyspark.ml.tuning as tune

rf = RandomForestClassifier(labelCol='loan_status', featuresCol='features', minInstancesPerNode=10, numTrees=10)

#use ParamGridBuilder to build a grid of parameters
rf_grid = (
    tune.ParamGridBuilder()
    .addGrid(rf.minInstancesPerNode
            , [50, 55, 60])
    .addGrid(rf.maxDepth
            , [15, 16, 17])
    .build()
)
rf_ev = BinaryClassificationEvaluator(labelCol='loan_status')

cross_v = tune.CrossValidator(
    estimator=rf
    , estimatorParamMaps=rf_grid
    , evaluator=rf_ev,
    numFolds = 10
)

pipeline = Pipeline(stages=stages)
data_trans = pipeline.fit(train)

rf_modelTest = cross_v.fit(
    data_trans.transform(train)
)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Exception in thread cell_monitor-26:
Traceback (most recent call last):
  File "/emr/notebook-env/lib/python3.7/threading.py", line 926, in _bootstrap_inn
    self.run()
  File "/emr/notebook-env/lib/python3.7/threading.py", line 870, in run
    self._target(*self._args, **self._kwargs)
  File "/emr/notebook-env/lib/python3.7/site-packages/awseditorssparkmonitoringwid
    job_binned_stages[job_id][stage_id] = all_stages[stage_id]
KeyError: 8971

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-26-a7bf05763c3b> in <module>
----> 1 get_ipython().run_cell_magic('spark', '', "import pyspark.ml.tuning as
tune\n\n\nrf = RandomForestClassifier(labelCol='loan_status',
featuresCol='features', minInstancesPerNode=10, numTrees=10)\n\n#use
ParamGridBuilder to build a grid of parameters\nrf_grid = (\n
tune.ParamGridBuilder()\n    .addGrid(rf.minInstancesPerNode\n            , [50,
55, 60])\n    .addGrid(rf.maxDepth\n            , [15, 16, 17])\n
.build()\n)\nrf_ev =
BinaryClassificationEvaluator(labelCol='loan_status')\n\ncross_v =
tune.CrossValidator(\n    estimator=rf\n    , estimatorParamMaps=rf_grid\n    ,
evaluator=rf_ev,\n    numFolds = 10\n)\n\npipeline =
Pipeline(stages=stages)\ndata_trans = pipeline.fit(train)\n\nrf_modelTest =
cross_v.fit(\n    data_trans.transform(train)\n)\n")

24 frames
<decorator-gen-125> in spark(self, *args, **kwargs)

/emr/notebook-env/lib/python3.7/socket.py in readinto(self, b)
    587         while True:
```

- minInstancesPerNode, [18 ,20, 22, 25, 40]) and maxDepth, [4, 6, 8, 10] = Best params - MaxDepth: 10, MinInstancesPerNode: 20

- Niki

- MaxDepth: [4, 6, 8, 10], MinInstancesPerNode: [5, 20, 40, 60] = Best: 10, 40
- MaxDepth: [10,12,14,16], MinInstancesPerNode: [30, 40, 50]= Best: 14, 50
- MaxDepth: [15,16,17], MinInstancesPerNode: [50, 55, 60]= Best: 16, 55

```
print('Best params - MaxDepth: {}, MinInstancesPerNode: {}'.format(
    rf_modelTest.bestModel._java_obj.getMaxDepth(),
    rf_modelTest.bestModel._java_obj.getMinInstancesPerNode()))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Best params - MaxDepth: 16, MinInstancesPerNode: 55
```

```
# measure performance of best model
data_trans_test = data_trans.transform(test)
predictions = rf_modelTest.transform(data_trans_test)
print('AUC:', rf_ev.evaluate(predictions))
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
AUC: 0.7143831495204369
```

```
accuracy = predictions.filter(
    predictions.loan_status == predictions.prediction).count() / float(
        predictions.count())
print("Accuracy : ", accuracy)
```

```
VBox()
FloatProgress(value=0.0, bar_style='info', description='Progress:',
layout=Layout(height='25px', width='50%'),…
Accuracy :  0.8006925740669488
```