

Description:

- 1. In this project I have used the lending club dataset. Lending club is a peer to peer lending company. They will be providing the loans to companies, peoples. Sometimes peoples will not have good credit history but they want immediately want loan will come to lending club and ask for loan.
- 2. There will be set of peoples who will be seeing the people's profile
 - a. If the applicant is likely to repay the loan, then not approving the loan results in a loss of business to the company
 - b. If the applicant is not likely to repay the loan, i.e. he/she is likely to default, then approving the loan may lead to a financial loss for the company
- 3. In this the interest rate are higher than the other banks.
- 4. Above is the overview of the lending club and I have used the the 2018 Q4 dataset.

Dataset:

The size of the dataset is 70mb which contains 128K records & 125 columns.



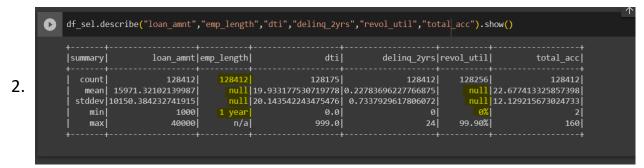
Problem:

The values present in the dataset is uncleaned(Some columns like term contains the datatypes as string('36 months') but in the business use we need to convert it into int(36).

If we downstream this data to machine learning models we have to make it to numeric.

Data Cleaning overview:

1. Since it contains 145 coulmns need to limit it into another df.



From the above image we can see in the 'emp_length' for the min it is showing as '1 year', mean & stddev it has treated as string but there is an integer value. Also we are having a good outliers for 'dti' for mean & stddev. We need to clean the above highlighed so that we can get the proper statistics.

From the 'emp_length' we could see 4 different types of values present in it, we need only the values in **numeric** type.

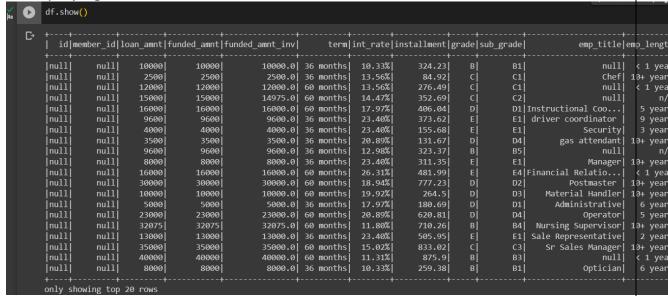
- 4. Clean the 'term' column
- 5. Clean the 'revol util' column.
- 6. Clean the 'dti' column -> Null values
- 7. Clean the 'loan_status'

Data Aggregation:

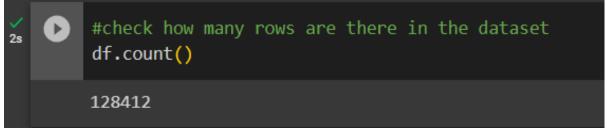
1.Schema of the dataset.

```
df.printSchema()
root
 |-- id: string (nullable = true)
 -- member id: string (nullable = true)
 -- loan amnt: integer (nullable = true)
 -- funded amnt: integer (nullable = true)
 -- funded amnt inv: double (nullable = true)
 -- term: string (nullable = true)
 -- int rate: string (nullable = true)
 -- installment: double (nullable = true)
 -- grade: string (nullable = true)
 -- sub grade: string (nullable = true)
 -- emp title: string (nullable = true)
 -- emp length: string (nullable = true)
 -- home ownership: string (nullable = true)
  -- annual inc: double (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: double (nullable = true)
  -- deling 2yrs: integer (nullable = true)
 -- earliest cr line: string (nullable = true)
 -- inq_last_6mths: integer (nullable = true)
 -- mths since last deling: integer (nullable = true)
```

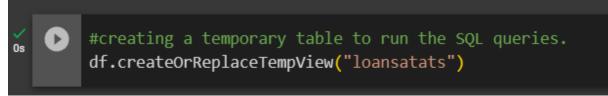
2. Displaying the records



3. How many row's are in the dataset.

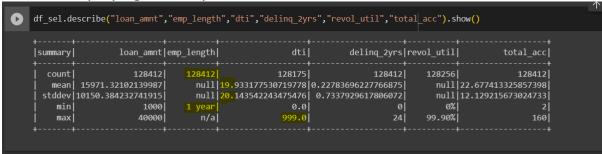


4.Creating a temporary variable to run the sql queries



5. Since there are more than 145 column limiting only the selected column

6. When displaying the output we can see the below error

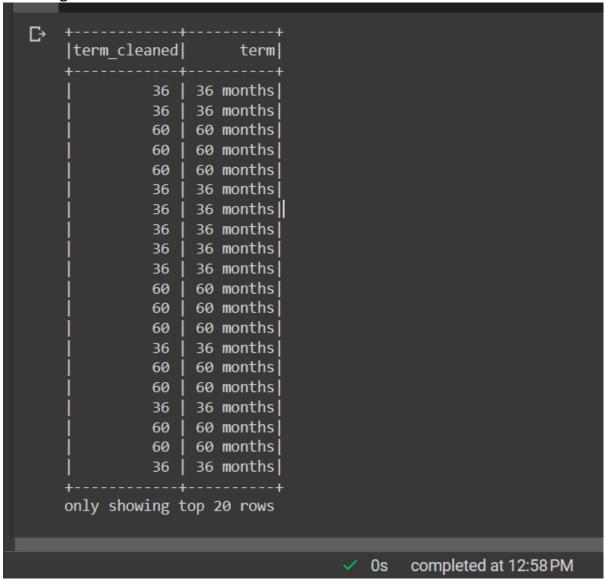


7. Checking the 'emp_length'

It contains 4 different types of values present in it but we need only the numeric values.

Data after cleaned:

Cleaning the 'term' coumn from '36 months' to 36



In the above one still it is not assigned to df. After assigning it to df,

```
df_sel.select("term","term_cleaned","emp_length","emplen_cleaned").show()
₽
          term|term_cleaned|emp_length|emplen_cleaned|
     36 months
                        36 I
                             < 1 year
                                                   1|
                             10+ years
     36 months
                        36
                                                  10
     60 months
                        60
                             < 1 year
                                                   1|
     60 months
                        60
                                  n/a
     60 months
                        60
                             5 years
     36 months
                        36 I
                             9 years
                                                   9
                              3 years
     36 months
                        36
                                                   3
      36 months
                           | 10+ years|
                                                  10
      36 months|
                                   n/a|
      36 months
                                                  10
                             10+ years
     60 months
                        60
                                                   1
                             < 1 year
     60 months
                             10+ years
                        60
                                                  10
     60 months
                             10+ years|
                        60
                                                  10
      36 months
                        36
                               6 years
                                                   6
                                                   5
     60 months
                               5 years
     60 months
                        60
                             10+ years
                                                  10
     36 months
                        36
                              2 years
                                                   2
     60 months
                        60 | 10+ years|
                                                  10
                              < 1 year
                                                   1|
      60 months
                        60
     36 months
                        36
                               6 years
                                                   6
    only showing top 20 rows
```

8. Print the schema

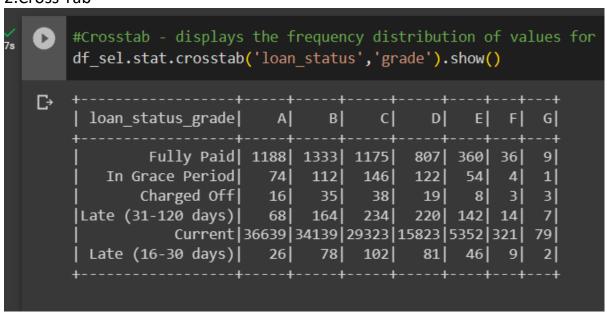
```
df sel.printSchema()
root
 |-- term: string (nullable = true)
 -- home ownership: string (nullable = true)
  -- grade: string (nullable = true)
  -- purpose: string (nullable = true)
  -- int rate: string (nullable = true)
  -- addr state: string (nullable = true)
  -- loan status: string (nullable = true)
 -- application type: string (nullable = true)
 -- loan amnt: integer (nullable = true)
 -- emp length: string (nullable = true)
 -- annual inc: double (nullable = true)
  -- dti: double (nullable = true)
 -- deling 2yrs: integer (nullable = true)
  -- revol util: string (nullable = true)
  -- total acc: integer (nullable = true)
  -- num tl 90g dpd 24m: integer (nullable = true)
  -- dti joint: double (nullable = true)
  -- term cleaned: string (nullable = true)
 -- emplen cleaned: string (nullable = true)
```

Even though the the column is cleaned and it has only the numerical values spark takes this as 'string' to convert the datatype into 'int'we need to do it manually(Redefine the schema).

Correlation Matrix:

1. Checking for Correlation.

2.Cross Tab



3.Frequency

```
freq=df_sel.stat.freqItems(['purpose','grade'],0.3)
freq.collect()

[Row(purpose_freqItems=['debt_consolidation', 'credit_card', 'other'], grade_freqItems=['A', 'B', 'C'])]
```

Aggregate Functions:

1. Checking how many null values are present in the column



Cleaning the 'revol_utill' column



Analysis:

```
#DTI Column
df_sel.describe("dti", "revol_util").show()
                         dti|revol util|
summary
                                  128256
   countl
                      128175
    mean | 19.933177530719778 |
                                    null|
  stddev 20.143542243475476
                                    null
     min|
                         0.0
                       999.0
                                 99.90%
     max
```

In the above image it shows the max value as 99 but in depth when analyzed,

It shows the max value as 184 but we can get only as 99. The reason behind this is it takes this as a **String[So that 9>1].** Now we need to convert it into **int**.

After cleaning we got the below output

'revol_util' is the column before cleaned. Once cleaned we can see the 'mean&stddev' values in it.

But still the max value is 99 since we haven't converted it into 'double'/'int'. Filling the null vales with average values present in it.

```
#Defining a function since the columns may conatin a lot of null values in it def fill_avg(df,colname):
    return df.select(colname).agg(avg(colname))

rev_avg=fill_avg(df_sel,'revolutil_cleaned')
```

Using Coalesce function.

```
#coalesce

It will take two variables

-if the first is not null it will take it.

-if the first is null it will take the second variable

from pyspark.sql.functions import coalesce

df_sel=df_sel.withcolumn('revolutil_cleaned',coalesce(col('revolutil_cleaned'),col('rev_avg')))

In the above one['revolutil_cleaned'] it will take all the not null values and whereever there is null it will use the 'rev_avg'.

So where ever there is a null it will substitute with the average(will convert the two columns into single columns)

"""
```

Converting the 'revolutil_cleaned' column to 'double' and displaying the describe function.

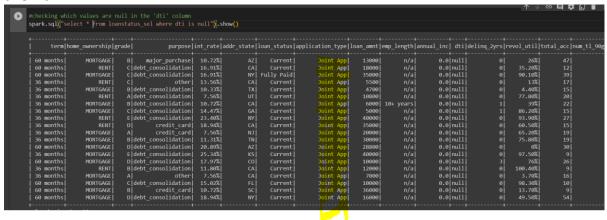
```
[38] #converting the datatype into double for the column 'revolutil cleaned'
     df_sel=df_sel.withColumn('revolutil_cleaned',df_sel['revolutil_cleaned'].cast('double'))
     df_sel.describe('revol_util','revolutil_cleaned').show()
     |summary|revol_util| revolutil_cleaned|
               128256
       count
                                    128412
                  null|43.762069610778525
        meanl
       stddev
                   null|24.786779696453955
                     0%|
          min|
                                       0.0
                  99.90%
                                      183.0
```

From the above picture we could able to see now **max** is showing the correct value 183.0 where the old value is 99.90%.

DTI Column:

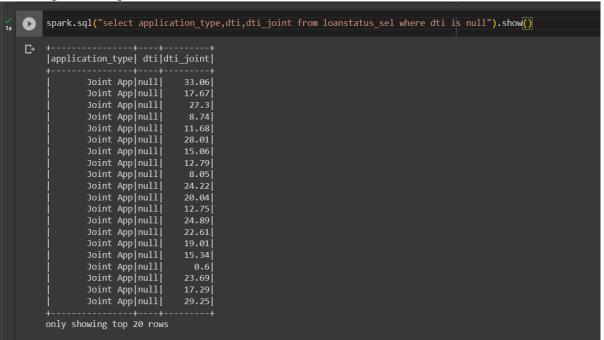


Still DTI column is showing it has 237 null values, Lets check which values are null.



When check we can see wherever the **application_type='Joint App'** the dti value is null.

Looking for Insights:



From the above insights,

we are going to create a new column name 'dti_cleaned' in that with the help of coalesce function, 'dti' records null values will be filled with the values of 'dti_joint'.



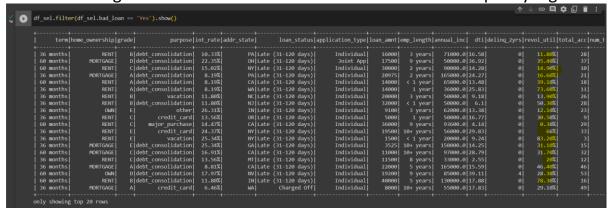
Cleaning 'loan_status'

From this we can understand only 'Fully Paid' alone shows that it has full paid, other values are showing like the payment is delayed then **Bad loan**.



From this we can see majority of loan is good loan.

When checking on the bad loan we can see the intrest rate are pretty high.



Dropping the column:

```
(19] #Dropping the unnecessary columns
        df sel final=df sel.drop('revol util','dti','dti joint','bad loan')
       df sel final.printSchema()
        root
         |-- term: string (nullable = true)
          -- home_ownership: string (nullable = true)
         -- grade: string (nullable = true)
          -- purpose: string (nullable = true)
          -- int rate: string (nullable = true)
          -- addr_state: string (nullable = true)
          -- loan status: string (nullable = true)
          -- application type: string (nullable = true)
          -- loan_amnt: integer (nullable = true)
          -- emp_length: string (nullable = true)
          -- annual inc: double (nullable = true)
          -- delinq_2yrs: integer (nullable = true)
          -- total acc: integer (nullable = true)
          -- num tl 90g dpd 24m: integer (nullable = true)
         -- bad_loan: string (nullable = false)
```

Storing data in a permanent table

```
#permanent table
#saving table in the parquet foramt
permanent_table='lc_loan_data'
df_sel.write.format("parquet").saveAsTable(permanent_table)
```

Notes:

- 1. Data frames in spark works in a distributed format like they will divide into chunks of parts and they assign it to worker node.
- 2. Data frames in Pandas is mostly dependent on Memory.
- 3. Data frames in spark has more function and used well for analyzing than SQL.
- 4. Spark is an lazy evaluation framework creates a DAG and then an action command comes and executes the DAG. So that it optimizes the DAG in the distributed way.
- 5. Cache()-Stores the df in the memory , subsequent operation can be performed faster without having to recompute the dataframe from source. If we cache too many dataframes will lead to excessive memory consumption and cause memory related issue.