Credit risk modeling

Introduction

Credit risk modeling is an essential component of the lending process for banks and other financial institutions, as it helps to determine the creditworthiness of borrowers, the risk of default, and the appropriate level of interest rates to charge. Credit risk modeling is the process of assessing the likelihood of a borrower defaulting on a loan or failing to repay debt.

In recent years, there has been a growing interest in using machine learning and artificial intelligence (AI) techniques to improve credit risk modeling. These techniques can analyze large datasets and identify patterns that may not be evident in traditional statistical models, leading to more accurate risk assessments and better lending decisions.

Overall, credit risk modeling plays a crucial role in the financial industry, helping lenders to assess the risk of default and make informed decisions about lending to specific borrowers. As the financial industry continues to evolve and new technologies emerge, credit risk modeling will continue to be an essential component of the lending process.

The main goal of this notebook is to show how to calculate the expected loss (EL)

Quick review

Expected Loss (EL) is the amount of money a lender can expect to lose on average over the life of a loan due to default. It takes into account the probability of default, the exposure at default, and the loss given default. Here's how to calculate the Expected Loss:

`EL = PD x LGD x EAD`

Where:

- PD = Probability of Default: the likelihood that the borrower will default on the loan during the life of the loan.
  - `Machine Learning model (classification problem) with a PD of 10%`
- LGD = Loss Given Default: the amount of money the lender expects to lose if the borrower defaults on the loan.
  - `LDG = (Total exposure - Recoveries) / Total exposure = (USD 100,000 - USD 20,000) / USD 100,000 = 80\%`
- EAD = Exposure at Default: the amount of money the lender is exposed to when the borrower defaults on the loan.
  - `EAD = Total exposure x (1 - Recovery rate) = USD 100,000 x (1 - 0.20) = USD 80,000`
  - The recovery rate of current loan is going to be calculated with GBM model - recovery rate (regression problem)

To calculate the Expected Loss, you need to estimate each of these components based on historical data.

For example, suppose a lender has a USD 100,000 loan to a borrower with a probability of default of 10\%, a loss given default of 80\%, and an exposure at default of USD 80,000. The Expected Loss with formulas and number above, would be:

`EL (result) = 10% x 80% x $80,000 = $6,400`

This means that the lender can expect to lose $6,400 on average over the life of the loan due to default. The Expected Loss is an important metric for lenders because it helps them estimate the amount of risk they are taking on and set appropriate loan pricing and risk management strategies.

This Notebook is going to use Spark framework and H2O cluster to run the entire process

- Spark 3.3.1
- H2O 3.38.0.4

In [1]:
```python
import os
import h2o
from pyspark.sql import SparkSession
import pandas as pd
# from deltalake import DeltaTable

## Metrics evaluation
from pyspark.ml.evaluation import RegressionEvaluator

## Sklearn Metrics
from sklearn.metrics import (confusion_matrix, classification_report, accuracy_score,
                             roc_auc_score, recall_score, roc_auc_score)

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

In [2]:
```python
def fshape(dataframe1):
    print('Shape : ', dataframe1.count(), len(dataframe1.columns))

def fhead(dataframe1, num_records=3):
    pd.options.display.max_columns = None
    return dataframe1.limit(num_records).toPandas()

def fsummary(dataframe1):
    return dataframe1.summary().toPandas()

## default Spark appName - se preferir
spark = SparkSession.builder.appName('Spark3-ML-quick-app').master('local[*]').getOrCreate()
```

```
sc = spark.sparkContext
spark
```

Out[2]: **SparkSession - in-memory**

SparkContext

Spark UI

Version	`v3.3.1`
Master	`local[*]`
AppName	`Spark3-ML-quick-app`

In [3]:
```
h2o.connect(ip='172.25.238.198')
h2o.remove_all()
```

```
Connecting to H2O server at http://172.25.238.198:54321 ... successful.
Warning: Your H2O cluster version is too old (3 months and 24 days)!Please download and install the latest version from http://h
2o.ai/download/
```

H2O_cluster_uptime:	2 hours 42 mins
H2O_cluster_timezone:	America/Sao_Paulo
H2O_data_parsing_timezone:	UTC
H2O_cluster_version:	3.38.0.4
H2O_cluster_version_age:	3 months and 24 days !!!
H2O_cluster_name:	userds1
H2O_cluster_total_nodes:	1
H2O_cluster_free_memory:	5.009 Gb
H2O_cluster_total_cores:	12
H2O_cluster_allowed_cores:	12
H2O_cluster_status:	locked, healthy
H2O_connection_url:	http://172.25.238.198:54321
H2O_connection_proxy:	null
H2O_internal_security:	False
Python_version:	3.9.13 final

In [4]:
```
data_dir = '/tmp/Credit_Risk_Modeling/dat1_raw/'
sdf1_loan = spark.read.parquet(data_dir + 'dat1_loan.2M__1.2GB.FULL_FILE_WITH_CONTRACT.parquet/')
fshape(sdf1_loan)
# sdf1_Loan.printSchema()
fhead(sdf1_loan)
```

```
Shape :  2260668 146
```

Out[4]:

	contract_id	acc_now_delinq	acc_open_past_24mths	addr_state	all_util	annual_inc	annual_inc_joint	application_type	avg_cur_bal	bc_open_to_buy
0	42949672960	0	9	NY	28	55000.0	NaN	Individual	1878	34360
1	42949672961	0	10	LA	57	90000.0	NaN	Individual	24763	13761
2	42949672962	0	4	MI	35	59280.0	NaN	Individual	18383	13800

One small sample of raw data will be provided at data_s3/ credit_risk_modeling_github_sample.parquet

- Loan_Lending_Club_profile_report_eda.html is also provided (full parquet file)
- Public dataset provided by Lending Club. Just google for kaggle lending club and download it

Start data engineering with Spark

In [5]:
```
## RUN LEGACY TO_DATE()
sdf1_loan.createOrReplaceTempView('TBP_LOAN_RAW')

# set Legacy timestamp police
spark.conf.set("spark.sql.legacy.timeParserPolicy", "LEGACY")

sql_tbp_loan_raw_to_silver = """
WITH TBP_SILVER (
    SELECT
        contract_id,
        acc_now_delinq,
        addr_state,
        annual_inc,
        delinq_2yrs,
        CASE WHEN dti < 0 THEN 0
```

```
          WHEN dti IS NULL THEN 0
          ELSE dti
         END dti,
         earliest_cr_line,
         TO_DATE(earliest_cr_line, 'MMMM-yyyy') AS earliest_cr_line_DT,
         CAST (SUBSTRING(earliest_cr_line, length(earliest_cr_line) - 3, 4) AS INT) earliest_cr_line_year,
         emp_length,
         CAST ( REPLACE ( REPLACE ( REPLACE (REPLACE( REPLACE ( REPLACE(emp_length, '+ years', ''), 'years', '')
                  , '< 1 year', '0') , 'year', '') , ' ', ''), 'n/a', '0') AS INT) emp_length_int,
         funded_amnt,
         funded_amnt_inv,
         grade,
         home_ownership,
         initial_list_status,
         inq_last_6mths,
         installment,
         int_rate,
         issue_d,
         TO_DATE(issue_d, 'MMMM-yyyy') AS issue_d_DT,
         CAST (SUBSTRING(issue_d, length(issue_d) - 3, 4) AS INT) issue_d_year,
         loan_amnt,
         loan_status,
         CASE WHEN loan_status IN ('Charged Off', 'Default', 'Late (31-120 days)',
               'Does not meet the credit policy. Status:Charged Off' )
               THEN '0'
               ELSE '1'
         END AS loan_status_good_vs_bad,
         CASE
          WHEN mths_since_last_delinq IS NOT NULL THEN mths_since_last_delinq
          ELSE 0
         END mths_since_last_delinq,
         CASE
          WHEN mths_since_last_record IS NOT NULL THEN mths_since_last_record
          ELSE 0
         END mths_since_last_record,
         purpose,
         recoveries,
         term,
         CAST(REPLACE(term, 'months', '') AS INT) term_int,
         verification_status,
         zip_code,

         -- REPORT ONLY
         emp_title,
         chargeoff_within_12_mths,
         last_pymnt_amnt,
         last_pymnt_d,
         next_pymnt_d,
         title,
         total_acc,

         -- ML AND SCORECARD
         -- contract_id,
         total_pymnt,
         total_rec_prncp,
         ROUND(recoveries / funded_amnt, 3) as recovery_rate,
         (funded_amnt - total_rec_prncp) / funded_amnt as credit_conversion_factor_CCF

         -- COMPLEMENT COLS
         ,sub_grade
         ,open_acc
         ,pub_rec
         ,total_acc
         ,total_rev_hi_lim
     FROM TBP_LOAN_RAW
     WHERE 1 = 1
     --    AND issue_d LIKE '%2014'
)
SELECT CASE
    WHEN recovery_rate > 1 THEN 1
    WHEN recovery_rate < 0 THEN 0
    ELSE recovery_rate
   END as recovery_rate_pct
, ROUND( months_between(TO_DATE('2019-03-01', 'yyyy-MM-dd'), issue_d_DT), 1)  as mths_since_issue_d
, ROUND(months_between(TO_DATE('2019-03-01', 'yyyy-MM-dd'), earliest_cr_line_DT), 1)  as mths_since_earliest_credit_line
, TBP_SILVER.*
FROM TBP_SILVER
WHERE 1 = 1
AND issue_d_year in (2015, 2016, 2017, 2018)
"""

sdf2_silver = spark.sql(sql_tbp_loan_raw_to_silver)
# sdf2_silver.printSchema()
```

In [6]:

```
cols_sorted = sorted(set(sdf2_silver.columns))
initial_columns = ['contract_id', 'loan_status']
initial_columns.reverse()
cols_sorted = [col for col in cols_sorted if col not in initial_columns]
for col_idx in initial_columns:
    cols_sorted.insert(0, col_idx)
cols_sorted
```

```python
## Spark dataframe with columns sort
sdf2_silver = sdf2_silver[cols_sorted]
# sdf2_silver.printSchema()
```

In [7]:

```python
sdf2_silver.groupBy('loan_status').count().show()
```

```
+------------------+------+
|       loan_status| count|
+------------------+------+
|        Fully Paid|668930|
|           Default|    31|
|   In Grace Period|  8716|
|       Charged Off|185263|
|Late (31-120 days)| 21537|
|           Current|906193|
| Late (16-30 days)|  3653|
+------------------+------+
```

In [8]:

```python
sdf2_hdf = sdf2_silver.where(" loan_status = 'Current' ")
fshape(sdf2_hdf)
fhead(sdf2_hdf)
```

```
Shape :  906193 51
```

Out[8]:

	contract_id	loan_status	acc_now_delinq	addr_state	annual_inc	chargeoff_within_12_mths	credit_conversion_factor_CCF	delinq_2yrs	dti	earliest_c
0	42949672960	Current	0	NY	55000.0	0	0.954408	0	18.24	Apı
1	42949672961	Current	0	LA	90000.0	0	0.979592	0	26.52	Jur
2	42949672962	Current	0	MI	59280.0	0	0.957442	0	10.51	Apı

In [9]:

```python
## Used table for prediction
sdf2_hdf.createOrReplaceTempView('TBP_CREDIT_RISK_MODELING')
```

Wrapper function for classification and regression metrics evaluation

A wrapper function for classification metrics evaluation is a function that simplifies the process of evaluating the performance of a classification model. It provides a unified interface for calculating various metrics such as accuracy, precision, recall, and F1 score, which are commonly used to measure the effectiveness of a classification algorithm.

Python - Sklearn sample

In [10]:

```python
## Function to print Confusion Matrix and metrics
def rpt_metrics_report_CM(y_true, y_pred, msg_model=' model name ... ', rpt_confusion_matrix=False):
    """Print metrics """

    accuracy_score_rpt = accuracy_score(y_true, y_pred)
    recall_score_rpt = recall_score(y_true, y_pred)
    auc_rpt = roc_auc_score(y_true, y_pred)

    print('Model: ', msg_model)
    print('--   Accuracy: ', accuracy_score_rpt)
    print('--   AUC      : ', auc_rpt)
    print('--   Recall   : ', recall_score_rpt)
    print('')
    if rpt_confusion_matrix:
        report = classification_report(y_true, y_pred)
        confusion_matrix_rpt = confusion_matrix(y_true, y_pred)
        print('--   Confusion Matrix')
        print('0   FP')
        print('FN  1')
        print('')
        print(confusion_matrix_rpt)
        print('')
        print('')
        print('--   Metrics report')
        print(report)
        print('')
```

Spark Sample - metrics evaluation

- Classification and Regression

In [11]:

```python
def print_evaluation_fnc_classification_metrics(dataframe1, label='label', prediction='prediction'):
    from pyspark.ml.evaluation import BinaryClassificationEvaluator, MulticlassClassificationEvaluator
    from pyspark.sql.functions import expr, col

    ## obs. as colunas precisam ter o nome label e prediction
    df = dataframe1.select(label, prediction)
    cols = ['label', 'prediction']
    df = df.toDF(*cols)
```

```python
    # cast label column to Double
    df = df.withColumn("label", df["label"].cast("Double"))
    df = df.withColumn("prediction", df["prediction"].cast("Double"))

    # assuming your DataFrame has the following column names: "label" and "prediction"
    predictionsAndLabels = df.select("label", "prediction")

    # create BinaryClassificationEvaluator object
    binary_evaluator = BinaryClassificationEvaluator(labelCol="label", rawPredictionCol="prediction", metricName="areaUnderROC")

    # create MulticlassClassificationEvaluator object
    multiclass_evaluator = MulticlassClassificationEvaluator(labelCol="label", predictionCol="prediction")

    # compute classification metrics for binary classification
    areaUnderROC = binary_evaluator.evaluate(predictionsAndLabels)
    areaUnderPR = binary_evaluator.setMetricName("areaUnderPR").evaluate(predictionsAndLabels)
    # f1Score = binary_evaluator.setMetricName("f1").evaluate(predictionsAndLabels)

    # compute classification metrics for multiclass classification
    accuracy = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "accuracy"})
    precision = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "weightedPrecision"})
    recall = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "weightedRecall"})
    f1Score = multiclass_evaluator.evaluate(predictionsAndLabels, {multiclass_evaluator.metricName: "f1"})
    confusionMatrix = predictionsAndLabels.groupBy("label", "prediction").count().orderBy("label", "prediction").toPandas()

    # print classification metrics
    print("")
    print("Multiclass Classification Metrics:")
    print("")
    print("Accuracy = %s" % accuracy)
    print("Precision = %s" % precision)
    print("Recall = %s" % recall)
    print("F1 Score = %s" % f1Score)
    print("")
    print("")
    print("Confusion Matrix:")
    print(confusionMatrix)

    print("")
    print("\nBinary Classification Metrics:")
    print("")
#     print("Area Under ROC = %s" % areaUnderROC)
    print("Area Under PR = %s" % areaUnderPR)
    print("F1 Score = %s" % f1Score)
#     print("Confusion Matrix:")
#     print(binary_evaluator.evaluate(predictionsAndLabels, {binary_evaluator.metricName: "confusionMatrix"}))

# print_evaluation_fnc_classification_metrics(sdf_credit_score,label='loan_status_good_vs_bad', prediction='predict')

def print_evaluation_regression_metrics(pred_dataframeSpark_1, label_col_1='label', prediction_col_1='prediction'):


    print('---------------  Regression Metrics')
    print()
    evaluator = RegressionEvaluator(labelCol=label_col_1, predictionCol=prediction_col_1, metricName="r2")
    r2 = evaluator.evaluate(pred_dataframeSpark_1)
    print("R2 - coeficient of determination on test data = %g" % r2)
    print()

    # Select (prediction, true label) and compute test error
    evaluator = RegressionEvaluator(labelCol=label_col_1, predictionCol=prediction_col_1, metricName="rmse")
    rmse = evaluator.evaluate(pred_dataframeSpark_1)
    print("Root Mean Squared Error (RMSE) on test data = %g" % rmse)
    print()

    evaluator = RegressionEvaluator(labelCol=label_col_1, predictionCol=prediction_col_1, metricName="mae")
    mae = evaluator.evaluate(pred_dataframeSpark_1)
    print("Mean Absolute Error (MAE) on test data = %g" % mae)
    print()

# print_evaluation_regression_metrics(predictions_boston_house, 'MEDV', 'ZSCORE0')
```

H2O - Load machine learning models

- GLM - predict PD : Probability of default - Classification model with 78.349% of accuracy
- GBM - predict recovery rate percent for current loan transactions - regression model with MAE of 0.062

In [12]:
```python
h2o_model_dir = '/tmp/Credit_Risk_Modeling/h2o_glm_gbm_model/'
glm_model = h2o.load_model('/mnt/d/'+h2o_model_dir +'fit_glm_2015_2017.model')
```

In [13]:
```python
def fnc_percent_print(metric):
    return round(metric * 100 , 4)

# glm_model.model_performance()
print(' GLM model accuracy - ', round(glm_model.accuracy()[0][1] * 100 , 4), ' % ')

print(' --- Max precision of ' , fnc_percent_print(glm_model.find_threshold_by_max_metric(metric='precision')),
      ' % with threshold adjustment ')
```

```
GLM model accuracy -  78.3499  %
--- Max precision of  96.2257  % with threshold adjustment
```

In [14]:
```python
## gbm model - Predict recovery rate pct for Current Loan
gbm_model = h2o.load_model('/mnt/d/'+h2o_model_dir +'gbm_model_recovery_rate_pct_EL_calculation.model')
```

In [15]:
```python
gbm_model.model_performance()
```

Out[15]:
```
ModelMetricsRegression: gbm
** Reported on train data. **

MSE: 0.00855472323108516
RMSE: 0.09249174682686645
MAE: 0.062329840365562966
RMSLE: 0.0782266478276316
Mean Residual Deviance: 0.00855472323108516
```

Note about integatrion with Spark and H2O

- The frameworks and running in separate enviroments so the integration will be done through file exchange.
  This approach can process massive amount of data also

Spark - export data in parquet format

In [16]:
```python
# ## Example to export with partition data by Year
# pq_spark_h2o_integration = '/tmp/zdata_s3/credit_risk_modeling/data_s3_credit_risk_modeling_2018.parquet'
# sdf2_hdf.write.format('parquet').mode('overwrite').partitionBy('issue_d_year').save(pq_spark_h2o_integration)
```

In [17]:
```python
## Export as only one file sample
pq_spark_h2o_integration_workaround = '/tmp/zdata_s3/credit_risk_modeling/data_zs3_credit_risk_modeling_2018_01.parquet'
sdf2_hdf.coalesce(1).write.format('parquet').mode('overwrite').save(pq_spark_h2o_integration_workaround +'.one_file.parquet')
```

Read parquet file with H2O Cluster

In [18]:
```python
dir_path = pq_spark_h2o_integration_workaround+'.one_file.parquet' + '/'
parquet_files = []
# Get a list of all the Parquet files in the directory
parquet_files = [os.path.join(dir_path, f) for f in os.listdir(dir_path) if f.endswith(".parquet")]
# parquet_files[0]
h2o_file = parquet_files[0]
```

In [19]:
```python
hdf_sdf2 = h2o.upload_file(h2o_file, destination_frame='hdf_loan_credit_risk_2018.hex')
```

```
Parse progress: |████████████████████████████████████████████████████████████| (done) 100%
```

In [20]:
```python
hdf_sdf2.head(3)
```

Out[20]:

contract_id	loan_status	acc_now_delinq	addr_state	annual_inc	chargeoff_within_12_mths	credit_conversion_factor_CCF	delinq_2yrs	dti	earliest_cr_li
4.29497e+10	Current	0	NY	55000	0	0.954408	0	18.24	2001-04- 00:00:
4.29497e+10	Current	0	LA	90000	0	0.979592	0	26.52	1987-06- 00:00:
4.29497e+10	Current	0	MI	59280	0	0.957442	0	10.51	2011-04- 00:00:

```
[3 rows x 51 columns]
```

H2O - GLM prediction - calcualte PD (p0)

In [21]:
```python
hdf_glm_predict = glm_model.predict(hdf_sdf2)
```

```
glm prediction progress: |████████████████████████████████████████████████████| (done) 100%
```

In [22]:
```python
hdf_glm_predict.head(3)
```

Out[22]:

predict	p0	p1
1	0.233671	0.766329
1	0.446381	0.553619
1	0.208331	0.791669

```
[3 rows x 3 columns]
```

H2O - GBM prediction - recovery rate percent (predict column)

In [23]:
```python
hdf_gbm = gbm_model.predict(hdf_sdf2)
```

```
gbm prediction progress: |████████████████████████████████████████████████| (done) 100%
```

In [24]:
```
hdf_gbm.head(3)
```

Out[24]:

predict
0.0709352
0.079283
0.0781656

[3 rows x 1 column]

DEEP COPY - full backup for quick restore if necessary

- Manipulation of h2o frames

In [25]:
```
# teste
hdf_sdf3_gold = h2o.deep_copy(hdf_sdf2['contract_id'], xid='hdf_loan_credit_risk_2018_EL.hex')
```

In [26]:
```
## Recovery rate with GBM prediction
hdf_sdf3_gold['recovery_rate_pct_predict_gbm'] = hdf_gbm['predict']
hdf_sdf3_gold.head(3)
```

Out[26]:

contract_id	recovery_rate_pct_predict_gbm
4.29497e+10	0.0709352
4.29497e+10	0.079283
4.29497e+10	0.0781656

[3 rows x 2 columns]

In [27]:
```
## Concatenate h2o frames with prediction - GLM model prediction
hdf_sdf3_gold = hdf_sdf3_gold.concat(hdf_glm_predict, axis=1)
hdf_sdf3_gold.head(3)
```

Out[27]:

contract_id	recovery_rate_pct_predict_gbm	predict	p0	p1
4.29497e+10	0.0709352	1	0.233671	0.766329
4.29497e+10	0.079283	1	0.446381	0.553619
4.29497e+10	0.0781656	1	0.208331	0.791669

[3 rows x 5 columns]

In [28]:
```
## Export file
h2o.export_file(frame=hdf_sdf3_gold, path='/tmp/credit_risk_modeling_h2o_2018.csv.gz', compression='gzip', force=True)
```

```
Export File progress: |██████████████████████████████████████████████████| (done) 100%
```

Read data with Spark

- Note that the process of exporting and importing data between Spark and H2O is related to situations where the Spark cluster and H2O cluster are running in different environments.
- The data is also stored using Data Lake architecture and use tools such as AWS Glue, AWS S3 and delta tables
- This notebook also simulate the execution with EC2 or EMR as an example

In [29]:
```
sdf_glm_gbm = spark.read.csv('/tmp/credit_risk_modeling_h2o_2018.csv.gz', inferSchema=True, header=True)
sdf_glm_gbm.printSchema()
```

```
root
 |-- contract_id: long (nullable = true)
 |-- recovery_rate_pct_predict_gbm: double (nullable = true)
 |-- predict: integer (nullable = true)
 |-- p0: double (nullable = true)
 |-- p1: double (nullable = true)
```

In [30]:
```
fhead(sdf_glm_gbm)
```

Out[30]:

	contract_id	recovery_rate_pct_predict_gbm	predict	p0	p1
0	42949672960	0.070935	1	0.233671	0.766329
1	42949672961	0.079283	1	0.446381	0.553619
2	42949672962	0.078166	1	0.208331	0.791669

In [31]:
```
# fshape(sdf_glm_gbm)

# fshape(sdf2_hdf)
```

In [32]:
```
fhead(sdf2_hdf)
```

Out[32]:

	contract_id	loan_status	acc_now_delinq	addr_state	annual_inc	chargeoff_within_12_mths	credit_conversion_factor_CCF	delinq_2yrs	dti	earliest_c
0	42949672960	Current	0	NY	55000.0	0	0.954408	0	18.24	Apı
1	42949672961	Current	0	LA	90000.0	0	0.979592	0	26.52	Jur
2	42949672962	Current	0	MI	59280.0	0	0.957442	0	10.51	Apı

In [33]:

```
## Create table to join all information
sdf_glm_gbm.createOrReplaceTempView('TB_CREDIT_RISK_PD_GLM_GBM')

sql_pd_glm_gbm = """
    SELECT contract_id
        , CASE
            WHEN recovery_rate_pct_predict_gbm < 0 THEN 0
            WHEN recovery_rate_pct_predict_gbm > 1 THEN 1
            ELSE recovery_rate_pct_predict_gbm
          END as recovery_rate_pct_predict_gbm
        , CASE
            WHEN predict = 1 THEN 'Full Payment'
            WHEN predict = 0 THEN 'Default'
            ELSE 'NOT MAPPED'
          END loan_prediction_str
        , predict as loan_prediction
        , p0 as p0_PD
        , p1 as p1
    FROM TB_CREDIT_RISK_PD_GLM_GBM
    WHERE 1 = 1
"""
sdf3_pd_glm_gbm = spark.sql(sql_pd_glm_gbm)
sdf3_pd_glm_gbm.createOrReplaceTempView('TB_CREDIT_RISK_PD_GLM_GBM_V2')
sdf3_pd_glm_gbm.printSchema()
```

```
root
 |-- contract_id: long (nullable = true)
 |-- recovery_rate_pct_predict_gbm: double (nullable = true)
 |-- loan_prediction_str: string (nullable = false)
 |-- loan_prediction: integer (nullable = true)
 |-- p0_PD: double (nullable = true)
 |-- p1: double (nullable = true)
```

In [34]:

```
# fhead(sdf3_pd_glm_gbm)

fsummary(sdf3_pd_glm_gbm)
```

Out[34]:

	summary	contract_id	recovery_rate_pct_predict_gbm	loan_prediction_str	loan_prediction	p0_PD	p1
0	count	906193	906193	906193	906193	906193	906193
1	mean	4.295048005718689E10	0.06782696133337138	None	0.9839294719778237	0.21968642917867556	0.7803135708213307
2	stddev	679831.7950687075	0.012169656607159591	None	0.1257469029455572	0.1360281936979018	0.13602819369790117
3	min	42949672960	0.0	Default	0	0.0	5.687117625267055E-6
4	25%	42949912885	0.06054192915197007	None	1	0.1096859414454423	0.6992008950554914
5	50%	42950207714	0.06833222666462632	None	1	0.19491164595872357	0.805032687793979
6	75%	42951110500	0.07558022842620801	None	1	0.3007592422127676	0.8903088404354708
7	max	42951933627	0.1607236757877962	Full Payment	1	0.9999943128823747	1.0

EL - Expected Loss calculation

- Credit Risk Modeling

In [35]:

```
sql_credit_modeling = """
WITH TBP_CREDIT_MODELING_V2
(
    SELECT TBP_CREDIT_RISK_MODELING.*
        , recovery_rate_pct_predict_gbm
        , loan_prediction
        , loan_prediction_str
        , tb_credit_risk_pd_glm_gbm.p0_PD as PD
        , TBP_CREDIT_RISK_MODELING.credit_conversion_factor_CCF as LGD
        , ( (TBP_CREDIT_RISK_MODELING.funded_amnt - TBP_CREDIT_RISK_MODELING.total_rec_prncp ) *
            ( 1 -   tb_credit_risk_pd_glm_gbm.recovery_rate_pct_predict_gbm)
         ) as EAD
    FROM TBP_CREDIT_RISK_MODELING,
        TB_CREDIT_RISK_PD_GLM_GBM_V2 as TB_CREDIT_RISK_PD_GLM_GBM
    WHERE 1 = 1
        AND TBP_CREDIT_RISK_MODELING.contract_id = tb_credit_risk_pd_glm_gbm.contract_id
)
SELECT TBP_CREDIT_MODELING_V2.*
```

```
    , (PD * LGD * EAD ) as EL
FROM TBP_CREDIT_MODELING_V2
"""

sdf3_credit_risk_modeling = spark.sql(sql_credit_modeling)
fshape(sdf3_credit_risk_modeling)
```

Shape : 906193 58

In [36]:
```
fhead(sdf3_credit_risk_modeling)
```

Out[36]:

	contract_id	loan_status	acc_now_delinq	addr_state	annual_inc	chargeoff_within_12_mths	credit_conversion_factor_CCF	delinq_2yrs	dti	earliest_c
0	42949672966	Current	0	IL	51000.0	0	0.957355	0	2.40	Nov
1	42949672976	Current	0	OH	102500.0	0	0.954014	0	15.20	Dec
2	42949673064	Current	0	CA	65000.0	0	0.979040	0	8.12	Apr

In [37]:
```
# ## Evaluate null values
# from pyspark.sql.functions import isnan, when, count, col
# def fnc_count_null_values(dataframe1):
#     dataframe1.select([count(when(isnan(c) | col(c).isNull(), c  )).alias(c) for c in dataframe1.columns if c not in [
#         'home.dest', 'issue_d_DT', 'earliest_cr_line_DT'
#     ]]
#                         ).show(vertical=True)

# fnc_count_null_values(sdf3_credit_risk_modeling)
```

Export results for dashboarding

In [38]:
```
## backup - v2
sdf3_credit_risk_modeling.coalesce(1).write.mode('overwrite').format('parquet').save(
    '/tmp/zdata_s3/credit_risk_modeling_full_1M_PBI.parquet'
)
```

In [39]:
```
# ## export sample with partition key for GitHub - small sample of data
sdf3_credit_risk_modeling.where(' issue_d_year = 2015 ').write.mode('overwrite').format(
    'parquet').partitionBy('issue_d_year').save('/tmp/zdata_s3/credit_risk_modeling_github_sample.parquet')
```

In [40]:
```
sdf3_credit_risk_modeling.createOrReplaceTempView('TBP_RPT__PBI__CREDIT_RISK')

sql_qry = """
WITH TB_EL_RPT AS
(
    SELECT loan_status
        , loan_prediction_str
        ,sum(funded_amnt) as sum_funded_amnt
        ,sum( round( EL, 4)) as sum_EL
    FROM TBP_RPT__PBI__CREDIT_RISK
    WHERE 1 = 1
    GROUP BY 1, 2

)
SELECT loan_status
    , loan_prediction_str
    , sum_funded_amnt
, ( sum_EL / sum_funded_amnt ) * 100 EL_pct
FROM TB_EL_RPT
WHERE 1 = 1
    AND 1 = 1
ORDER BY 3 DESC
"""
sql_sdf_current = spark.sql(sql_qry)
# sql_sdf_current.printSchema()
fhead(sql_sdf_current)
```

Out[40]:

	loan_status	loan_prediction_str	sum_funded_amnt	EL_pct
0	Current	Full Payment	14044302400	11.090311
1	Current	Default	295299225	41.932573

Spark warehouse tables

In [41]:
```
### JOIN DEFAULT Spark with all Predictions
# TBP_CREDIT_RISK_MODELING AND TB_CREDIT_RISK_PD_GLM_GBM
print(' ----------  data pipeline from raw to silver and bronze table ... ')
spark.sql(' SHOW TABLES ').show(truncate=False)
```

```
----------  data pipeline from raw to silver and bronze table ...
+---------+--------------------------+-----------+
|namespace|tableName                 |isTemporary|
+---------+--------------------------+-----------+
|         |tb_credit_risk_pd_glm_gbm |true       |
|         |tb_credit_risk_pd_glm_gbm_v2|true     |
|         |tbp_credit_risk_modeling  |true       |
|         |tbp_loan_raw              |true       |
|         |tbp_rpt__pbi__credit_risk |true       |
+---------+--------------------------+-----------+
```

EL - Expected Loss report

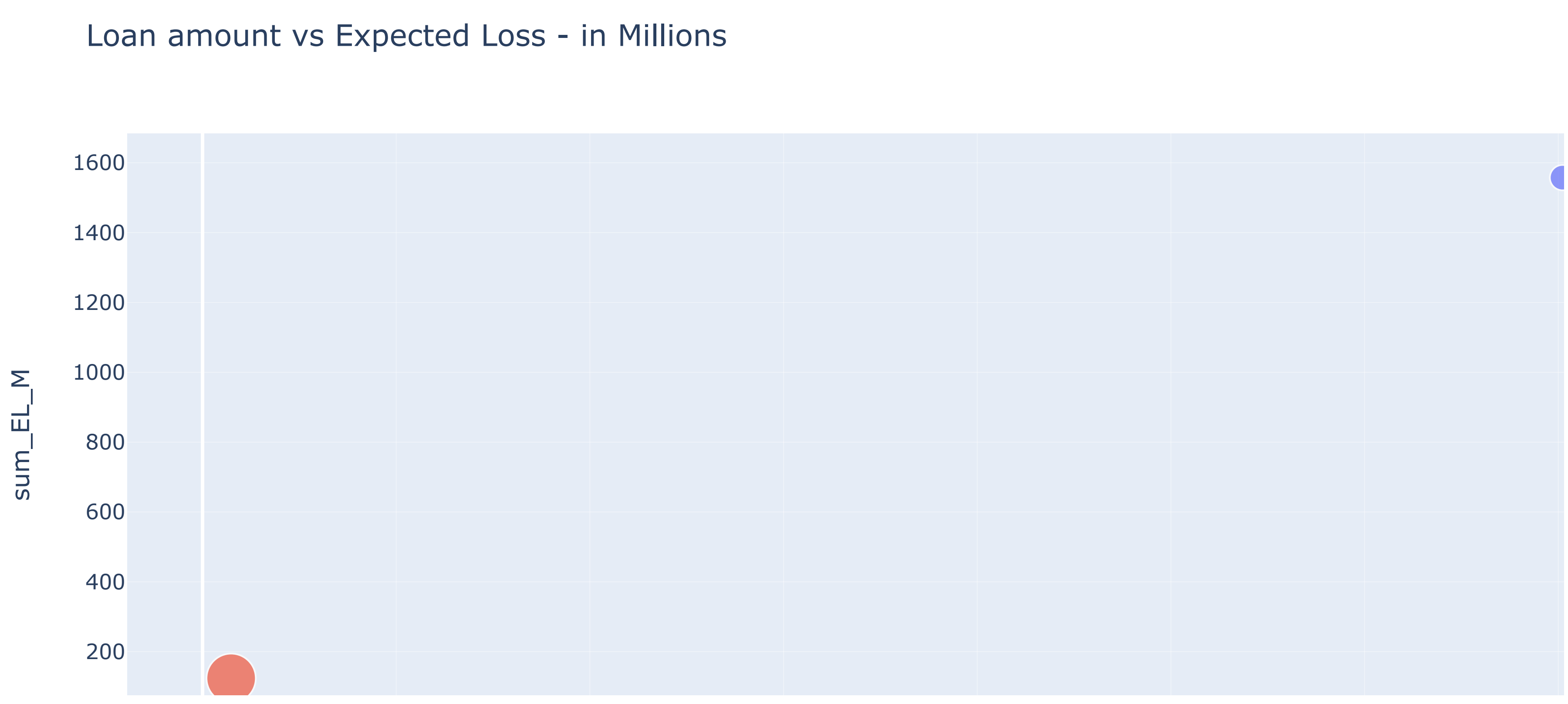
In [42]:
```
sql_rpt = """
WITH TB_EL_RPT AS
(
    SELECT loan_status
        , loan_prediction_str
        ,sum(funded_amnt) as sum_funded_amnt
        ,sum( round( EL, 4)) as sum_EL
    FROM TBP_RPT__PBI__CREDIT_RISK
    WHERE 1 = 1
    GROUP BY 1, 2

)
SELECT loan_status
    , loan_prediction_str
    , sum_funded_amnt / 1000 / 1000 as sum_funded_amnt_M
    , sum_EL / 1000 / 1000 as sum_EL_M
, ( sum_EL / sum_funded_amnt ) * 100 EL_pct
FROM TB_EL_RPT
WHERE 1 = 1
    AND 1 = 1
ORDER BY 3 DESC
"""
sdf_rpt = spark.sql(sql_rpt)
```

In [43]:
```
fhead(sdf_rpt)
```

Out[43]:

	loan_status	loan_prediction_str	sum_funded_amnt_M	sum_EL_M	EL_pct
0	Current	Full Payment	14044.302400	1557.556806	11.090311
1	Current	Default	295.299225	123.826564	41.932573

In [44]:
```
## Scatter with BUBLE SIZE
sdf_rpt.pandas_api().plot.scatter(x='sum_funded_amnt_M', y='sum_EL_M',
                              color='loan_prediction_str', size='EL_pct', title=' Loan amount vs Expected Loss - in Millions')
```



In [45]:
```
!jupyter nbconvert --to html Credit_risk_modeling__Expected_Loss__EL__PD_LGD_EAD.ipynb
```

In [ ]: