# Credit Risk Scoring

- **Objective 2.1: Data Cleaning**
- **Objective 2.2: Feature Scaling & Hyper-Parameter Tuning**
- **Objective 2.3 Score-carding *with* Machine Learning**

# Problem and Motivation

- **Environmental Scope:**
  - One of the Company's services ("Loan Plan" Service), provided to new eligible customers seeking a monetary loan.
- **Problems:**
  - Before Granting such Loans, the Company would First Need to Evaluate All new Applicants based on their Submitted Loan Registration Details:
    - Loan History
    - Assets Currently Available (e.g. Owned/Rented Properties)
    - and more
  - These details allow the Company to Tabulate the Probability of these applicants defaulting on their Desired Loans through referencing the Company's Historical Loan Lending Data of Previous Customers to date with their Loan Outcomes (i.e. whether they have Defaulted or Not).
  - From there, the Company can then Grant Customers displaying Lower Probabilities of Defaulting, on their Requested Loans, while Considering the Company's Currently Available Funding for provisioning the New Loans.
    - This is known as **<u>Credit Scoring</u>**.

# Problem and Motivation

- **Motivation:**
  - Digitalizing and Automating an efficient Credit Scoring System
    - To reduce any unnecessary Operating Expenditure(s) caused by Human intervention (i.e. Credit Scoring Mistakes, Slow Score Tabulations, and other Human-Related Errors).
  - Despite it being possible to curate a Algorithmic System (Non-Machine Learning) for such a purpose, the Company has personally Requested the Use of Machine Learning, particularly:
    - <u>Logistic Regression (LR) algorithm</u>
    - With a <u>Measurable Scorecard</u> for more Transparent Credit Scoring Interpretations.

# **GAPS** Capstone Tackles

**Studies Lack of:**

- **Feature Scaling** *(Gap 1)*
  - Nikolic et al [15] : **No Scaling indications** in Study
  - Li and Chen [12] : **No Scaling indications** in Study
  - Bensic et al [13] : **No Scaling indications** in Study
- **Hyper-Parameter Tuning of Logistic Regression Model** *(Gap 2)*
  - Bensic et al [13] : Tuned all Models **Except** for **Logistic Regression (LR)** Using Credit Scoring Data, experimented using **LR**, <u>4</u> Neural Networks, Decision Trees. Study's Justification for **Not** tuning **LR-** <u>Small Dataset</u>
  - Li and Chen [12] : Tuning Done on Random Forest, except **for LR** (**No Tuning Indications** in Study)
    (Aim of Study- Use 10 Models Classifiers, Evaluated using AUC/Accuracy/Kolmogorov-Smirnov Performance Metrics)

**As such, the Capstone's solution would include the use of both <u>Feature Scaling</u> and <u>Hyper-Parameter Tuning</u> to Improve Credit Scoring Performance in Continuation to the Studies.**

# Objective 2.1

**Clean and Transform the Dataset**

# Raw (Loan) Data

- Dataset from **Lending Club**™ (peer-to-peer Lending Company based in the US).
  - Loans Data from Year 2007 to 2018 (4th Quarter).
    - 2 Million Observations (Rows) and 145 Features (Variables).

| | id | member_id | loan_amnt | funded_amnt | funded_amnt_inv | term | int_rate | installment | grade | sub_grade | emp_title | emp_length | home_ownership | annu |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | NaN | NaN | 2500 | 2500 | 2500.0 | 36 months | 13.56 | 84.92 | C | C1 | Chef | 10+ years | RENT | 5 |
| **1** | NaN | NaN | 30000 | 30000 | 30000.0 | 60 months | 18.94 | 777.23 | D | D2 | Postmaster | 10+ years | MORTGAGE | 9 |
| **2** | NaN | NaN | 5000 | 5000 | 5000.0 | 36 months | 17.97 | 180.69 | D | D1 | Administrative | 6 years | MORTGAGE | 5 |
| **3** | NaN | NaN | 4000 | 4000 | 4000.0 | 36 months | 18.94 | 146.51 | D | D2 | IT Supervisor | 10+ years | MORTGAGE | 9 |
| **4** | NaN | NaN | 30000 | 30000 | 30000.0 | 60 months | 16.14 | 731.78 | C | C4 | Mechanic | 10+ years | MORTGAGE | 5 |

# Columns Kept (after Dropping 80% Null Columns)

**Columns Kept ( 97 columns):**

| # | Column | Dtype |
|---|---|---|
| 0 | loan_amnt | int64 |
| 1 | funded_amnt | int64 |
| 2 | funded_amnt_inv | float64 |
| 3 | term | object |
| 4 | int_rate | float64 |
| 5 | installment | float64 |
| 6 | grade | object |
| 7 | emp_length | object |
| 8 | home_ownership | object |
| 9 | annual_inc | float64 |
| 10 | verification_status | object |
| 11 | issue_d | object |
| 12 | loan_status | object |
| 13 | pymnt_plan | object |
| 14 | purpose | object |
| 15 | addr_state | object |
| 16 | dti | float64 |
| 17 | delinq_2yrs | float64 |
| 18 | earliest_cr_line | object |
| 19 | inq_last_6mths | float64 |
| 20 | mths_since_last_delinq | float64 |
| 21 | open_acc | float64 |
| 22 | pub_rec | float64 |
| 23 | revol_bal | int64 |
| 24 | revol_util | float64 |
| 25 | total_acc | float64 |
| 26 | initial_list_status | object |
| 27 | out_prncp | float64 |
| 28 | out_prncp_inv | float64 |
| 29 | total_pymnt | float64 |
| 30 | total_pymnt_inv | float64 |
| 31 | total_rec_int | float64 |
| 32 | last_pymnt_d | object |
| 33 | last_pymnt_amnt | float64 |
| 34 | last_credit_pull_d | object |
| 35 | collections_12_mths_ex_med | float64 |
| 36 | mths_since_last_major_derog | float64 |
| 37 | policy_code | int64 |
| 38 | application_type | object |
| 39 | acc_now_delinq | float64 |
| 40 | tot_coll_amt | float64 |
| 41 | tot_cur_bal | float64 |
| 42 | open_acc_6m | float64 |
| 43 | open_act_il | float64 |
| 44 | open_il_12m | float64 |
| 45 | open_il_24m | float64 |
| 46 | mths_since_rcnt_il | float64 |
| 47 | total_bal_il | float64 |
| 48 | il_util | float64 |
| 49 | open_rv_12m | float64 |
| 50 | open_rv_24m | float64 |
| 51 | max_bal_bc | float64 |
| 52 | all_util | float64 |
| 53 | total_rev_hi_lim | float64 |
| 54 | inq_fi | float64 |
| 55 | total_cu_tl | float64 |
| 56 | inq_last_12m | float64 |
| 57 | acc_open_past_24mths | float64 |
| 58 | avg_cur_bal | float64 |
| 59 | bc_open_to_buy | float64 |
| 60 | bc_util | float64 |
| 61 | chargeoff_within_12_mths | float64 |
| 62 | delinq_amnt | float64 |
| 63 | mo_sin_old_il_acct | float64 |
| 64 | mo_sin_old_rev_tl_op | float64 |
| 65 | mo_sin_rcnt_rev_tl_op | float64 |
| 66 | mo_sin_rcnt_tl | float64 |
| 67 | mort_acc | float64 |
| 68 | mths_since_recent_bc | float64 |
| 69 | mths_since_recent_bc_dlq | float64 |
| 70 | mths_since_recent_inq | float64 |
| 71 | mths_since_recent_revol_delinq | float64 |
| 72 | num_accts_ever_120_pd | float64 |
| 73 | num_actv_bc_tl | float64 |
| 74 | num_actv_rev_tl | float64 |
| 75 | num_bc_sats | float64 |
| 76 | num_bc_tl | float64 |
| 77 | num_il_tl | float64 |
| 78 | num_op_rev_tl | float64 |
| 79 | num_rev_accts | float64 |
| 80 | num_rev_tl_bal_gt_0 | float64 |
| 81 | num_sats | float64 |
| 82 | num_tl_120dpd_2m | float64 |
| 83 | num_tl_30dpd | float64 |
| 84 | num_tl_90g_dpd_24m | float64 |
| 85 | num_tl_op_past_12m | float64 |
| 86 | pct_tl_nvr_dlq | float64 |
| 87 | percent_bc_gt_75 | float64 |
| 88 | pub_rec_bankruptcies | float64 |
| 89 | tax_liens | float64 |
| 90 | tot_hi_cred_lim | float64 |
| 91 | total_bal_ex_mort | float64 |
| 92 | total_bc_limit | float64 |
| 93 | total_il_high_credit_limit | float64 |
| 94 | hardship_flag | object |
| 95 | disbursement_method | object |
| 96 | debt_settlement_flag | object |

# Utilize Stratification Split (Reduce Imbalance Classes)

**Split data**

```
# Split the data using an 80/20 split
X = loan_data.drop('good_bad', axis = 1)
y = loan_data['good_bad']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42, stratify = y)

X_train, X_test = X_train.copy(), X_test.copy()
```

- **Training:Test Split** (80:20) to **Prevent Overfitting** (as compared to using 100% Dataset for Testing)
- **Random Stratified Sampling** also Assists in **Reducing Class imbalances** during Data Sampling.

Vannoy et al [75]

Wall and Fontenot [77]

# Data cleaning (Data-Specific)

- String Replacement (emp_length, **String to Numbers** only)

```
array([ 7., 10.,  3.,  4.,  2.,  0.,  1.,  6.,  5.,  8.,  9.])
```

*E.g. emp_length String to Integers type*

- **Date Conversions to Integer** (Number of Days from specific Date- e.g. Today) (earliest_cr_line, issue_d, last_pymnt_d, last_credit_pull_d)

# Feature Selection

- A p-value **less than 0.05 (typically ≤ 0.05) is statistically significant**.
  - It indicates Strong Evidence against the Null Hypothesis (No **Statistical Relationship**).

Panesar [53]

- P value **greater than 0.05** means (**Insignificant**) that no effect was observed.
  - Remove such Insignificant Features (≥ **0.05**)

Cai et al [21], Panesar [53], Doan et al [55]

# Chi-Squared

- Gauges the Degree of Independency between Categorical Variables.
  - i.e.Identifies Highly Related/Dependent Variables (against One Another)
- Also gives the Option of Determining which Categories are Significant.
  - Being the "most useful tools in the researcher's array of available analysis tools"
- Limited to **only** Discrete/Categorical Variables.
- **Action: Select top 5**

| | Feature | p-value |
|---|---|---|
| 0 | grade | 0.0 |
| 1 | home_ownership | 0.0 |
| 2 | verification_status | 0.0 |
| 3 | pymnt_plan | 0.0 |
| 4 | purpose | 0.0 |
| 5 | addr_state | 0.0 |
| 6 | initial_list_status | 0.0 |
| 7 | application_type | 0.0 |
| 8 | hardship_flag | 0.0 |
| 9 | disbursement_method | 0.0 |
| 10 | debt_settlement_flag | 0.0 |

Kamalov and Thabtah [44]

McHugh [78]

Urbanowicz [54]

# ANOVA F-Score

- Determines Strength of Relationship between Features and Labels
- Able to handle <u>Discrete and Continuous Variables</u>.
  - Replaces Chi-Squared for Continuous Variables.
- **Action: Select top 20**

| | Numerical_Feature | F-Score | p values |
|---|---|---|---|
| 0 | mths_since_last_pymnt_d | 101279.076110 | 0.0 |
| 1 | int_rate | 79891.093938 | 0.0 |
| 2 | mths_since_last_credit_pull_d | 64883.628239 | 0.0 |
| 3 | last_pymnt_amnt | 63017.806787 | 0.0 |
| 4 | out_prncp | 60841.859961 | 0.0 |
| 5 | out_prncp_inv | 60830.025704 | 0.0 |
| 6 | mths_since_issue_d | 36090.598122 | 0.0 |
| 7 | total_pymnt_inv | 34518.813024 | 0.0 |
| 8 | total_pymnt | 34484.620171 | 0.0 |
| 9 | acc_open_past_24mths | 15827.049134 | 0.0 |
| 10 | term | 14166.382950 | 0.0 |
| 11 | inq_last_6mths | 13784.078774 | 0.0 |
| 12 | num_tl_op_past_12m | 12395.693952 | 0.0 |
| 13 | bc_open_to_buy | 11932.905087 | 0.0 |
| 14 | total_bc_limit | 10038.049778 | 0.0 |
| 15 | percent_bc_gt_75 | 9830.099516 | 0.0 |
| 16 | bc_util | 9186.155895 | 0.0 |
| 17 | revol_util | 7627.581526 | 0.0 |
| 18 | tot_hi_cred_lim | 7152.718716 | 0.0 |
| 19 | mths_since_recent_inq | 6168.541817 | 0.0 |

Yange et al [79]

Urbanowicz et al [54]

# One Hot Encoding

- One-Hot Encoding was used as opposed to Label Encoding.
  - Label Encoding (uses numerical values eg. 5 , 7) to represent category labels
    - ML algorithms may perceive such labels as a form of numerical relation (e.g. hierarchy/order), leading potentially bias predictions.
  - One-Hot Encoding (uses binary values e.g. 1, 0) and separate columns instead
    - Eliminates undesirable numerical relation biases

| grade:A | grade:B | grade:C | grade:D | grade:E | grade:F | grade:G | home_ownership:ANY | home_ownership:MORTGAGE | home_ownership:NONE | home_ownership:OTHEI |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | [illegible] |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | [illegible] |
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | [illegible] |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | [illegible] |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | [illegible] |

Miron et al [71]

# Types of Features

- Discrete (**Categorical**) Features ->

| grade | home_ownership |
|---|---|
| C | MORTGAGE |
| C | MORTGAGE |
| D | RENT |
| C | MORTGAGE |
| C | RENT |

- Continuous (**Numerical**) Features ->

| inq_last_6mths | revol_util |
|---|---|
| 2.0 | 36.4 |
| 0.0 | 90.7 |
| 0.0 | 44.0 |
| 0.0 | 39.3 |
| 0.0 | 39.0 |

# Weight of Evidence (WOE)

- Estimates the **Predictive Power** (I.e. Weight) **of any Given Bin,** within a Feature.
  - Later Transformed **to Determine Information Value (IV)** - Main Goal.

Siddiqi [57] , Zeng [59]

$$\boxed{WoE = \ln\left(goods_i / \sum_{i=1}^{n} goods_i\right) - \ln\left(bads_i / \sum_{i=1}^{n} bads_i\right)}$$

Nikolic et al's study [3]

*Distribution of Non-Defaulting Cases* ***DIVIDED BY*** *Defaulting Cases*

- **Inner Workings of WOE:**
  - Contribution of Each Feature in Relation to the Actual Outcome (Predictive Power)
  - Rank-ability of Features' Bins according to their Predictive Power
  - Accept and Handle Missing Values without the Need of any Imputative Actions, on the Dataset.

Siddiqi [57] and Baesens et al's [58]

# Information Value (IV)

- Determines the Feature's uncertainty of predicting the outcome based on its

Zhang et al [81]

Sum of the (% of non-defaulting customers - % of defaulting customers)

*

**WOE**

*(WOE)*

$$IV = \sum_{i=1}^{n} (\text{Distr Good}_i - \text{Distr Bad}_i) * \ln\left(\frac{\text{Distr Good}_i}{\text{Distr Bad}_i}\right)$$

Nikolic et al [15]

# Determining Information Value

Guideline for Information Value (IV):

- IV <0.02 == not useful for prediction (Action: Ignore Feature)
- IV 0.02 to 0.1 == weak predictive power **(Action: Keep Feature)**
- IV 0.1 to 0.3 == medium predictive power **(Action: Keep Feature)**
- IV 0.3 to 0.5 == strong predictive power **(Action: Keep Feature)**
- IV >0.5 suspicious predictive power (Action: Ignore Feature)

# WOE (Pseudo Code 1)

- Pseudo-Code of ***Number of Observations ['n_obs' column] <u>AND</u> Proportion of Non-Defaulting ['prop_good' column]***

- For Each Feature (eg. 'Grade' Feature):

  - For Each Feature's Unique <u>Value/Value-Range</u> (eg. Categorical Value of *'A'/'B'/'C'/'D'* **<u>OR</u>** Continuous Value-Range of *(5.284, 6.594] / (6.594, 7.878]* )

    - Calculate the **Number of Observations (**i.e. ***'n_obs' column***)
      Get the **Number of Non-Defaulting Records** that Uses that Feature's <u>Value/Value-Range</u> (eg. Total of 12,000 Defaulting Cases)

    - Calculate the **Proportion of Non-Defaulting (**i.e. **'prop_good' column**)
      Get the **Proportion of the Number of Non-Defaulting Records** that uses that Feature's <u>Categorical/Continuous Value/Value-Range</u>
      (eg. Proportion of 0.6 for Grade 'B' means that 60% of All Records that uses the Feature's <u>Categorical/Continuous Value/Value-Range</u> of Grade 'B' are non-Defaulting Outcomes)

# WOE (Pseudo Code 2)

- Pseudo-Code of ***WOE*** *(i.e. **'WoE' column** in Section 3.6)***:**
- For Each Feature (eg. 'Grade' Feature):
  - For Each Feature's Unique Value/Value-Range (eg. Categorical Value of *'A'/'B'/'C'/'D'* **OR** Continuous Value-Range of *(5.284, 6.594] / (6.594, 7.878]* )
  - Calculate **Number of Observations**
  - Calculate **Proportion of Non-Defaulting Records**
  - Calculate **Proportion of Defaulting Records** (i.e. Same Procedure as **Proportion of Non-Defaulting Records** but by using Defaulting Outcomes**)**
  - Calculate the **Number of Non-Defaulting Records**
    - (i.e. '**n_good' column**: **Proportion of Non-Defaulting Records MULTIPLY Number of Observations**)
  - Calculate the **Number of Defaulting Records**
    - (i.e. '**n_bad' column**: [1 **MINUS Proportion of Defaulting Records**] ***MULTIPLY*** **Number of Observations**)
  - Calculate the **Proportion of Non-Defaulting Records**
    - (i.e. '**prop_n_good' column**: **Number of Non-Defaulting Records**
      ***DIVIDED BY*** **Summation of All Non-Defaulting Records in Feature**)
  - Calculate the **Proportion of Defaulting Records**
    - (i.e. '**prop_n_bad' column**: **Number of Defaulting Records**
      ***DIVIDED BY*** **Summation of All Defaulting Records in Feature**)
  - Calculate **WOE** of Each Feature's Categorical/Continuous Value/Value-Range
    (**Proportion of Non-Defaulting Records DIVIDED BY Proportion of Defaulting Records)**

# Example 1: Information Value

- IV <0.02 == not useful for prediction (Action: Ignore Feature)
- IV 0.02 to 0.1 == weak predictive power **(Action: Keep Feature)**
- IV 0.1 to 0.3 == medium predictive power **(Action: Keep Feature)**
- IV 0.3 to 0.5 == strong predictive power **(Action: Keep Feature)**
- IV >0.5 suspicious predictive power (Action: Ignore Feature)

Out[ ]:

| | purpose | n_obs | prop_good | prop_n_obs | n_good | n_bad | prop_n_good | prop_n_bad | WoE | diff_prop_good | diff_WoE | IV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | small_business | 5605 | 0.787333 | 0.015026 | 4413.0 | 1192.0 | 0.013282 | 0.029234 | -0.788933 | NaN | NaN | 0.036668 |
| 1 | educational | 351 | 0.792023 | 0.000941 | 278.0 | 73.0 | 0.000837 | 0.001790 | -0.760693 | 0.004690 | 0.028240 | 0.036668 |
| 2 | renewable_energy | 295 | 0.837288 | 0.000791 | 247.0 | 48.0 | 0.000743 | 0.001177 | -0.459668 | 0.045265 | 0.301026 | 0.036668 |
| 3 | moving | 2397 | 0.848561 | 0.006426 | 2034.0 | 363.0 | 0.006122 | 0.008903 | -0.374498 | 0.011273 | 0.085169 | 0.036668 |
| 4 | house | 1824 | 0.861294 | 0.004890 | 1571.0 | 253.0 | 0.004728 | 0.006205 | -0.271777 | 0.012733 | 0.102721 | 0.036668 |
| 5 | other | 19006 | 0.861675 | 0.050951 | 16377.0 | 2629.0 | 0.049291 | 0.064477 | -0.268581 | 0.000381 | 0.003196 | 0.036668 |
| 6 | medical | 3750 | 0.863467 | 0.010053 | 3238.0 | 512.0 | 0.009746 | 0.012557 | -0.253469 | 0.001791 | 0.015112 | 0.036668 |

# Example 2: Information Value

- IV <0.02 == not useful for prediction **(Action: Ignore Feature)**
- IV 0.02 to 0.1 == weak predictive power **(Action: Keep Feature)**
- IV 0.1 to 0.3 == medium predictive power **(Action: Keep Feature)**
- IV 0.3 to 0.5 == strong predictive power **(Action: Keep Feature)**
- IV >0.5 suspicious predictive power (Action: Ignore Feature)

Out[ ]:

| | emp_length | n_obs | prop_good | prop_n_obs | n_good | n_bad | prop_n_good | prop_n_bad | WoE | diff_prop_good | diff_WoE | IV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 0.0 | 45764 | 0.876497 | 0.122682 | 40112.0 | 5652.0 | 0.120727 | 0.138618 | -0.138189 | NaN | NaN | 0.006408 |
| 1 | 1.0 | 23582 | 0.888262 | 0.063218 | 20947.0 | 2635.0 | 0.063045 | 0.064625 | -0.024743 | 0.011765 | 0.113446 | 0.006408 |
| 2 | 2.0 | 33123 | 0.889895 | 0.088795 | 29476.0 | 3647.0 | 0.088715 | 0.089444 | -0.008184 | 0.001633 | 0.016559 | 0.006408 |
| 3 | 3.0 | 29301 | 0.890925 | 0.078549 | 26105.0 | 3196.0 | 0.078569 | 0.078383 | 0.002372 | 0.001030 | 0.010555 | 0.006408 |
| 4 | 4.0 | 22482 | 0.891602 | 0.060269 | 20045.0 | 2437.0 | 0.060330 | 0.059768 | 0.009357 | 0.000677 | 0.006985 | 0.006408 |
| 5 | 5.0 | 24654 | 0.885374 | 0.066092 | 21828.0 | 2826.0 | 0.065697 | 0.069309 | -0.053524 | 0.006229 | 0.062881 | 0.006408 |
| 6 | 6.0 | 21057 | 0.883079 | 0.056449 | 18595.0 | 2462.0 | 0.055966 | 0.060382 | -0.075936 | 0.002294 | 0.022413 | 0.006408 |

# Steps for WOE Binning

Only if the Feature's **IV is within the Acceptable Range**, then it'll be allowed to be WOE Binned:

- Each bin should contain 5% of the observations (*prop_n_obs*)
  - Fewer the bins increases 'smoothing' allowing the capture of important data while leaving out noise.
- The Current Approach taken: 10/20 bins
  - Too Few Bins, Reduce Data Distribution leading to Model Instability)
  - Too Many Bins, causes Model Overfitting
- Each Category (Bin) must be Non-Zero for both events and non-events (e.g. default and non-default)
- WOE Values Must be Distinct for each Category
  - Similar Groups are Binned Together (I.e. Bins with Similar WOE values have Same Proportion of Defaulting/Not-Defaulting on loans -> Same predictive power)
- Missing Values are Binned Separately

Siddiqi [57] and Nikolic et al [15]

# Eg: Plan Weight Of Evidence (**Continuous** Features)

- Each Bin
  **5% of Observations**

| | int_rate_factor | n_obs | Bin 1 | prop_n_obs | n_good | n_bad | prop_n_good | prop_n_bad | WoE | diff_prop_good | diff_WoE | IV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | (5.284, 6.594] | 99983 | Bin 2 | 0.055284 | 97848.0 | 2135.0 | 0.061888 | 0.009386 | 1.886139 | NaN | NaN | 0.449621 |
| 1 | (6.594, 7.878] | 145963 | Bin 3 | 0.080708 | 141215.0 | 4748.0 | 0.089317 | 0.020873 | 1.453750 | 0.011175 | 0.432389 | 0.449621 |
| 2 | (7.878, 9.162] | 154733 | Bin 4 | 0.085557 | 144831.0 | 9902.0 | 0.091604 | 0.043530 | 0.744021 | 0.031465 | 0.709729 | 0.449621 |
| 3 | (9.162, 10.446] | 166225 | Bin 5 | 0.091911 | 154837.0 | 11388.0 | 0.097932 | 0.050063 | 0.671003 | 0.004515 | 0.073018 | 0.449621 |
| 4 | (10.446, 11.73] | 217977 | Bin 6 | 0.120527 | 197517.0 | 20460.0 | 0.124927 | 0.089944 | 0.328543 | 0.025354 | 0.342460 | 0.449621 |
| 5 | (11.73, 13.014] | 201025 | Bin 7 | 0.111154 | 176975.0 | 24050.0 | 0.111934 | 0.105726 | 0.057064 | 0.025774 | 0.271479 | 0.449621 |
| 6 | (13.014, 14.298] | 187860 | Bin 8 | 0.103874 | 161992.0 | 25868.0 | 0.102458 | 0.113718 | -0.104270 | 0.018061 | 0.161333 | 0.449621 |
| 7 | (14.298, 15.582] | 142739 | Bin 9 | 0.078925 | 121497.0 | 21242.0 | 0.076845 | 0.093382 | -0.194901 | 0.011119 | 0.090631 | 0.449621 |
| 8 | (15.582, 16.866] | 118746 | Bin 10 | 0.065659 | 97265.0 | 21481.0 | 0.061519 | 0.094432 | -0.428539 | 0.032082 | 0.233639 | 0.449621 |
| 9 | (16.866, 18.15] | 113345 | Bin 11 | 0.062672 | 91132.0 | 22213.0 | 0.057640 | 0.097650 | -0.527179 | 0.015078 | 0.098639 | 0.449621 |
| 10 | (18.15, 19.434] | 79641 | Bin 12 | 0.044036 | 61614.0 | 18027.0 | 0.038970 | 0.079248 | -0.709791 | 0.030376 | 0.182613 | 0.449621 |
| 11 | (19.434, 20.718] | 48931 | | 0.027056 | 37599.0 | 11332.0 | 0.023781 | 0.049816 | -0.739463 | 0.005238 | 0.029672 | 0.449621 |
| 12 | (20.718, 22.002] | 39421 | | 0.021797 | 29906.0 | 9515.0 | 0.018915 | 0.041829 | -0.793620 | 0.009777 | 0.054157 | 0.449621 |
| 13 | (22.002, 23.286] | 21935 | Bin 13 | 0.012129 | 16163.0 | 5772.0 | 0.010223 | 0.025374 | -0.909104 | 0.021772 | 0.115484 | 0.449621 |
| 14 | (23.286, 24.57] | 21226 | | 0.011737 | 15596.0 | 5630.0 | 0.009864 | 0.024750 | -0.919905 | 0.002100 | 0.010801 | 0.449621 |
| 15 | (24.57, 25.854] | 20624 | | 0.011404 | 14780.0 | 5844.0 | 0.009348 | 0.025691 | -1.010950 | 0.018118 | 0.091046 | 0.449621 |
| 16 | (25.854, 27.138] | 10973 | | 0.006067 | 7950.0 | 3023.0 | 0.005028 | 0.013289 | -0.971888 | 0.007865 | 0.039063 | 0.449621 |
| 17 | (27.138, 28.422] | 4060 | | 0.002245 | 3060.0 | 1000.0 | 0.001935 | 0.004396 | -0.820395 | 0.029189 | 0.151493 | 0.449621 |
| 18 | (28.422, 29.706] | 5406 | | 0.002989 | 3896.0 | 1510.0 | 0.002464 | 0.006638 | -0.990969 | 0.033014 | 0.170574 | 0.449621 |
| 19 | (29.706, 30.99] | 7721 | | 0.004269 | 5386.0 | 2335.0 | 0.003407 | 0.010265 | -1.103019 | 0.023103 | 0.112050 | 0.449621 |

# Eg: Plan Weight Of Evidence (**Discrete** Features)

- Each Bin **5% of Observations**

| | grade | n_obs | prop_good | prop_n_obs | n_good | n_bad | prop_n_good | prop_n_bad | WoE | diff_prop_good | diff_WoE | IV |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | G | 9697 | Bin 1 | 0.005362 | 5845.0 | 3852.0 | 0.003697 | 0.016934 | -1.521816 | NaN | NaN | 0.483175 |
| **1** | F | 33507 | | 0.018527 | 21455.0 | 12052.0 | 0.013570 | 0.052982 | -1.362083 | 0.037550 | 0.159733 | 0.483175 |
| **2** | E | 108512 | | 0.060000 | 78263.0 | 30249.0 | 0.049500 | 0.132977 | -0.988198 | 0.080924 | 0.373885 | 0.483175 |
| **3** | D | 259341 | Bin 2 | 0.143398 | 207569.0 | 51772.0 | 0.131285 | 0.227594 | -0.550195 | 0.079133 | 0.438003 | 0.483175 |
| **4** | C | 519664 | Bin 3 | 0.287340 | 446903.0 | 72761.0 | 0.282661 | 0.319864 | -0.123648 | 0.059614 | 0.426547 | 0.483175 |
| **5** | B | 531178 | Bin 4 | 0.293706 | 486447.0 | 44731.0 | 0.307672 | 0.196641 | 0.447651 | 0.055805 | 0.571300 | 0.483175 |
| **6** | A | 346635 | Bin 5 | 0.191666 | 334577.0 | 12058.0 | 0.211616 | 0.053008 | 1.384329 | 0.049425 | 0.936678 | 0.483175 |

# Objective 2.2

**Feature Scaling & Hyper-Parameter Tuning**
**(Tackle Gaps here)**

# Machine Learning Pipelines (under Evaluation)

**4 Pipelines:**

- Pipeline 1: WOE Binning > Logistic Regression (**Untuned**)
- Pipeline 2: WOE Binning > Logistic Regression (**Tuned**)
- Pipeline 3: WOE Binning > **StandardScaler** > Logistic Regression (**Untuned**)
- Pipeline 4: WOE Binning > **StandardScaler** > Logistic Regression (**Tuned**)

Aim of Pipelines: <u>Tackle (two) Gaps</u> in Studies

*(**Tuned**) means that Logistic Regression Model being Trained by GridSearch Hyper-Parameter Optimization

** **StandardScaler** means that Dataset is Normalized through Feature Scaling

# Logistic Regression (Tuning and Parameters)

- **Brute Force approach** (discovery of 14 million Models) by W. Ksią ̇zek et al's study [11]
  - **GridSearch** Hyper-Parameter Optimization used (for Experiment)
- A Few **Parameters adapted** from W. Ksią ̇zek et al's study [11]:
  - **C**: 1-100
    ("Higher values generalize the model")
  - **Max Iteration** : 1000
    ("Maximum iterations to converge the model")
  - **Penalty**: l2
    "Adds bias to Model when it is suffering from high variance"
  - **Solver**: lbfgs, liblinear, saga
    ("Type of an Algorithm Solver")

# Pipeline (Model Evaluation)

Evaluate Models via **<u>Performance Metrics</u>**:

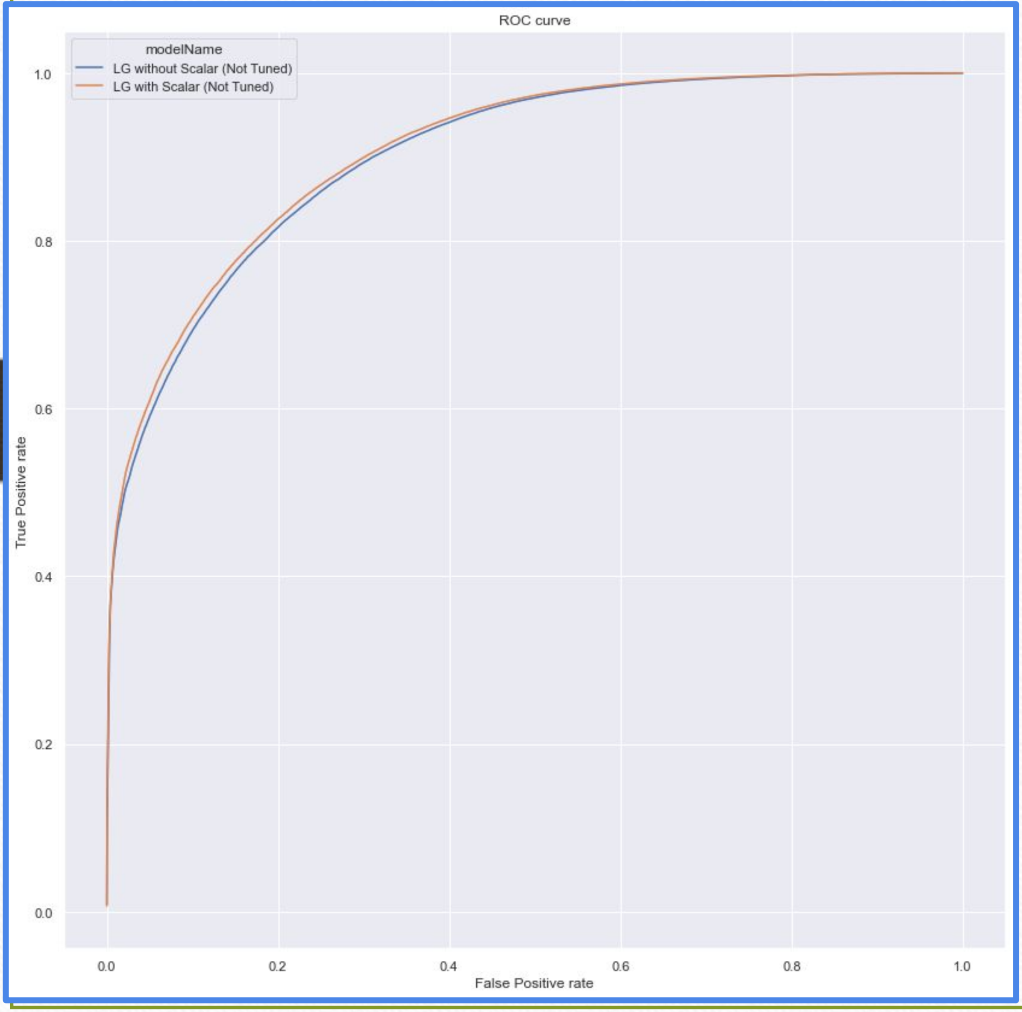
- Reason: Attain **Numerical** TP, TN, FP, FN.

- **ROC Curve** (Graphical Plot): *Relation between TPR/FPR*
  - Reason: **Visually Compare** 4 Pipelines' TPR and FPR.
- **AUROC** (Performance Score): *Area Under ROC Curve,* closer to 1 means better performance
  - Reason: Should TPR/FPR **Results be Too Small** for ROC Curve Inspection, a Score of 0 to 1 is Given for Easier Referencing (Based on TPR/FPR Values).
- **GINI** (Performance Score): *Twice of Area Under ROC*
  - **Reaffirm** AUROC score (Based on TPR/FPR Values).

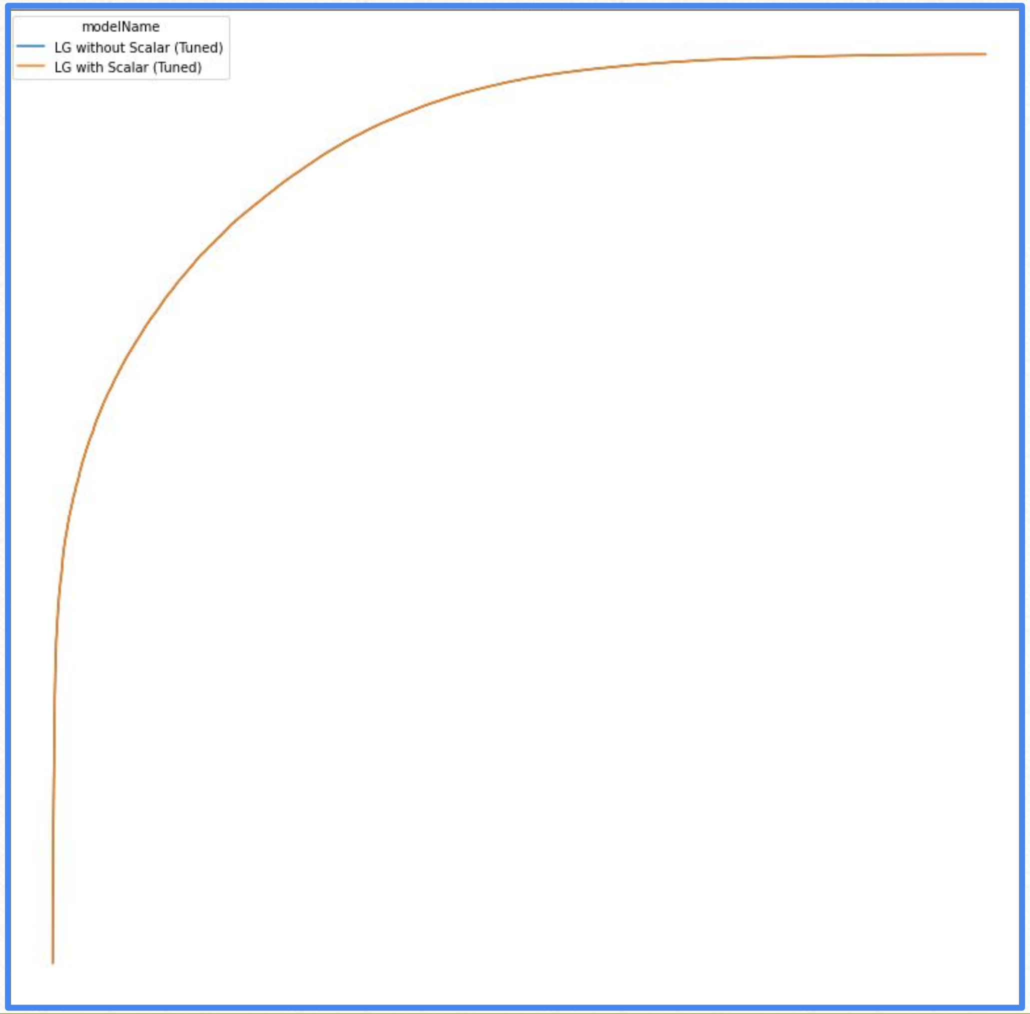
# Results

| Pipeline No. | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| LR Tuned? | No | Yes | No | Yes |
| Feature Scaling? | No | No | Yes | Yes |
| GINI Score | 0.8158713758013114 | 0.8158900854661821 | 0.8158908685747437 | 0.8158908685747437 |
| AUROC Score | 0.9079356879006557 | 0.9079450427330911 | 0.9079454342873718 | 0.9079454342873718 |
| True Positive (TP) | 26107 | 26121 | 26120 | 26122 |
| True Negative (TN) | 387327 | 387327 | 387328 | 387328 |
| False Positive (FP) | 30762 | 30748 | 30749 | 30747 |
| False Negative (FN) | 7938 | 7938 | 7937 | 7937 |
| Confusion Matrix Original Plot Found in | Figure 6 (Appendix A) | Figure 7 (Appendix A) | Figure 6 (Appendix A) | Figure 7 (Appendix A) |
| Confusion Matrix Plot Name (in Figure) | LG (No Scalar) | LG (No Scalar) Tuned | LG & Scalar | LG & Scalar Tuned |
| ROC Original Plot Found in | Figure 29 LG without Scalar (Not Tuned) | Figure 30 LG without Scalar (Tuned) | Figure 29 LG with Scalar (Not Tuned) | Figure 30 LG with Scalar (Tuned) |

# ROC Curves (Visual but too small Value Variances)

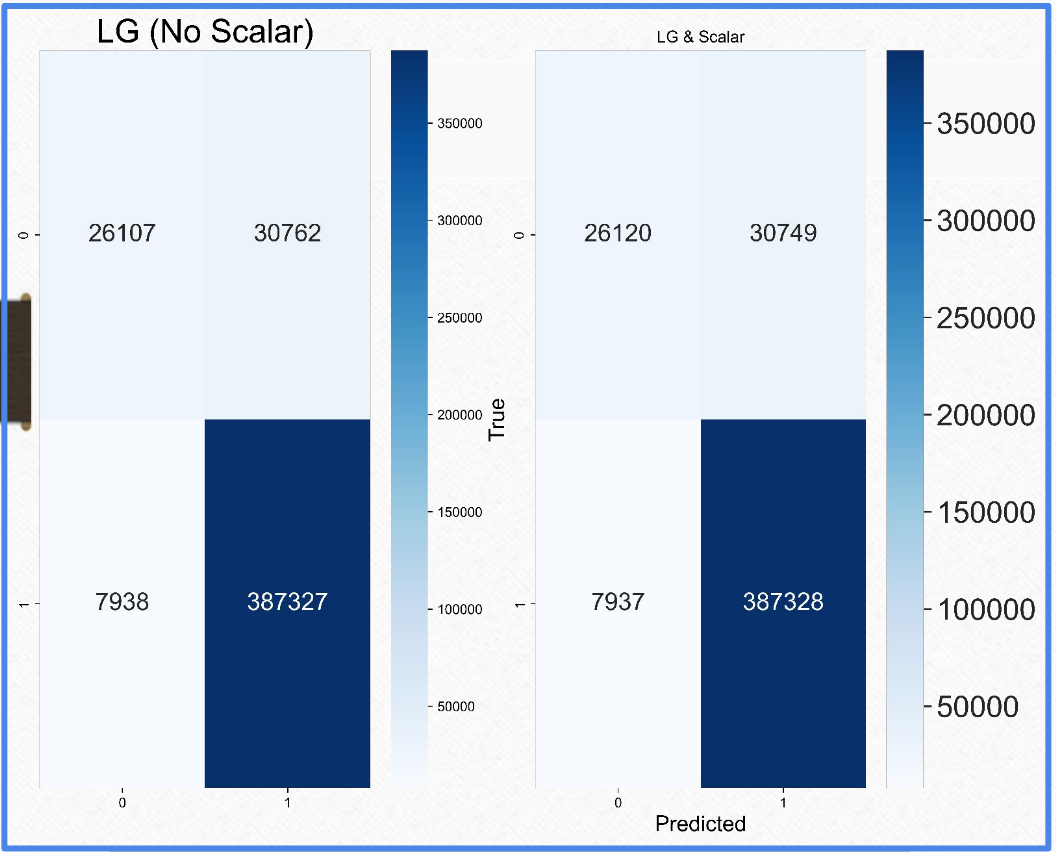



*ROC Curve for **Pipeline 1 and 3**, Without and With Feature Scaling Respectively*

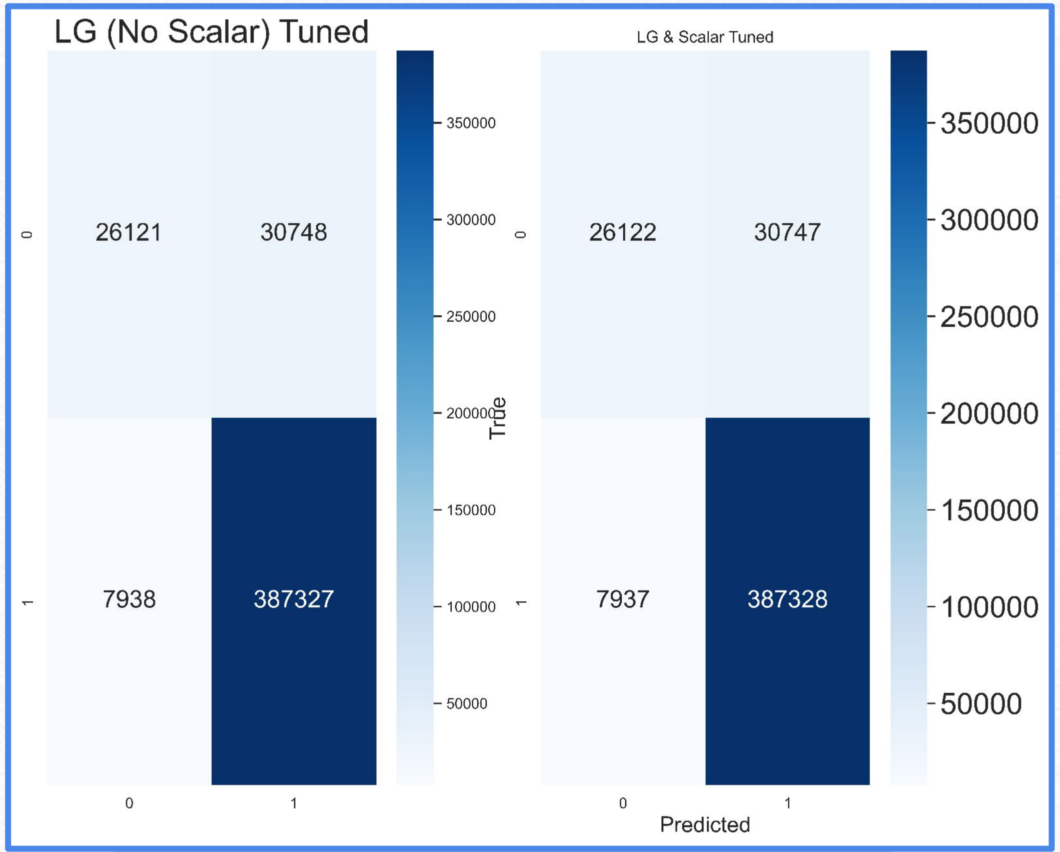



*ROC Curve for **Pipeline 2 and 4**, Without and With Feature Scaling Respectively*

# ROC Curves (Visual but too small Value Variances)




*Confusion Matrixes for* ***Untuned*** *Pipeline 1 and 3,*
***Without and With Feature Scaling*** *Respectively*




*Confusion Matrixes for* ***Tuned*** *Pipeline 2 and 4,*
***Without and With Feature Scaling*** *Respectively*

# Discussion: **Feature Scaling** (GAP)

Comparing Both **Untuned** LR Models (Pipeline 1 and 3):

- Pipeline 3 (Feature **Scaled** and **Without Tuning**) had **13 TP and 1 TN** (how the Model "got it right") Values more than Pipeline 1 (**No** Feature Scaling and **Without Tuning**).
- Pipeline 3 had **13 FP and 1 FN** (how the Model "got it wrong") Values lesser than Pipeline 1.
- Pipeline 3 has a **Higher** AUROC Score when compared to Pipeline 1.
- Pipeline 3 has a **Higher** GINI Score when compared to Pipeline 1.

Comparing Both **Tuned** LR Models (Pipeline 2 and 4):

- Pipeline 4 (Feature **Scaled** and **With Tuning**) also had **1 TP and 1** TN Values more than the Pipeline 2 (**No** Feature Scaling and **With Tuning**).
- Pipeline 4 had **1 FP and 1 FN** Values lesser than Pipeline 2.
- Pipeline 4 has a **Higher** AUROC Score when compared to Pipeline 2.
- Pipeline 4 has a **Higher** GINI Score when compared to Pipeline 2.

# Discussion: **Hyper-Parameter Tuning** (GAP)

Comparing Both (Feature) **Unscaled** LR Models (Pipeline 1 and 2):

- Pipeline 2 (**Tuning** and **Without Feature Scaling**) had **14 TP** (how the Model "got it right") Values more than Pipeline 1 (**<u>No</u>** Tuning and **Without Feature Scaling**).
- Pipeline 2 had **14 FP** (how the Model "got it wrong") Values lesser than Pipeline 1.
- Pipeline 2 has a **Higher** AUROC Score when compared to Pipeline 1.
- Pipeline 2 has a **Higher** GINI Score when compared to Pipeline 1.

Comparing Both (Feature) **Scaled** LR Models (Pipeline 3 and 4):

- Pipeline 4 (Feature **Scaled** and **With Tuning**) also had **2 TP** values more than the Pipeline 3 (Feature **Scaled** and **Without Tuning**).
- Pipeline 4 had **2 FP** Values lesser than Pipeline 3.
- Pipeline 4 has the **Same** AUROC Score as Pipeline 3.
- Pipeline 4 has a **Same** GINI Score as Pipeline 3.
  (AUROC and GINI based on TPR/FPR, Confusion Matrices Show Small Positive/Negative Value Improvements)

<u>About Same AUROC/GINI:</u>
1) Feature Scaling Bridged/Normalized "features with larger numerical values" **<u>[7]</u>** giving GridSearch Optimizer Similar Transformed Datasets to Work With; and/or
2) Pipelines' Relatively Small **TP** and **FP** Hard to Notice.

However, despite the Same AUROC/GINI Scores (I.e. Dependent on TPR/FPR Values), their Derived **TP/FP/TN/FN Values** can be Taken from the Confusion Matrix.

# In Summary

- **This Capstone was able to Fulfil the Company's need for :**
  - An **Interpretable** Scorecard
  - Complex Enough to be Trained via the Logistic Regression (LR) Model with Grid Search Optimization to **Maximize the Predicted Credit Scores**' Accuracy

**Deliverables:**

- The **Scorecard** and **Trained Model's Pipeline** was then **Conformed** to the FICO™ Credit Scoring Range (300 to 850), **allowing the Company to Analyze** Further if the Predicted Scores are still Relevant/Usable (I.e. by being within the FICO™ Range)
- The Capstone's Workflow Design included Initial Data Pre-Processing of **145 Features** which includes:
  - **Removing** Empty Features
  - **Preparing the Model Labels** for Each Record's Default Outcome
  - **Ensuring** High Variance (reduced biases) through Dataset Splitting
  - **Transforming** the Train/Test Datasets' Data into Numerical Values for Compatibility with the Machine Learning Model later
  - **Feature Selection** (i.e. ANOVA F-Test and Chi-Squared) to Determine the Most Significant Features within the Dataset
    - Top 4 **Categorical** and 20 **Continuous Features** were Selected **for WOE Binning**
    - One-Hot Encoded **Eliminated Relational Value Biases** within Each Feature
    - Dataset was then **Analyzed** based on its Weight of Evidence (WOE) and Information Values prior to being WOE Binned, respectively.

# In Summary

**Deliverables:**

- The Capstone's **Main Aim in Resolving the Gaps**, such as How Model Tuning and Feature Scaling could **Improve ML Model Performance**, was Identified in Previous Credit Scoring Studies [[3],[4],[5]].
  - These Resolutions were accomplished by Employing Grid Search Optimization (i.e. **Model Tuning**) on the LR Model with the Inclusion of Standard Scaler (i.e. **Feature Scaling**).
- **Accuracy** of the four differing ML Experimental Pipeline Configurations was **Evaluated** through Metrics like:
  - ROC Curves
  - AUROC
  - GINI Coefficient
  - TPR/FPR Confusion Matrixes
- **Results**
  - Pipeline with the Highest Accuracy was the **combined use of Grid Search** and **Standard Scalar**, together **with the standard WOE Binning**
  - This Pipeline Attained the **Most True-Positives and True-Negatives**, with the **Lowest False-Negatives and False-Positives**.
- **Creation of a Credit Scorecard** with the Chosen Pipeline producing Credit Scores based on its Customers' Loan Application Data
- **Future Improvement**
  - Potential use of the Model Tuning **Bayesian Search Optimizer** (with ROC evaluation for TPR/FPR Values)
    - Could Replace Grid Search and **Reduce Overall Time Complexity** and **Computation Overhead** while still providing Parameters with Highest Model Accuracy.

# Objective 2.3

Score-carding with Machine Learning

# Sample of Scorecard

Out[38]:

| | index | Feature name | Coefficients | Original feature name | Score - Calculation | Score - Preliminary |
|---|---|---|---|---|---|---|
| **0** | 0 | Intercept | 3.928058 | Intercept | 599.362503 | 599.0 |
| **1** | 1 | grade:A | 0.393112 | grade | 25.982462 | 26.0 |
| **2** | 2 | grade:B | 0.340824 | grade | 22.526535 | 23.0 |
| **3** | 3 | grade:C | 0.232294 | grade | 15.353307 | 15.0 |
| **4** | 4 | grade:D | 0.168589 | grade | 11.142754 | 11.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **103** | 11 | tot_hi_cred_lim:>499999.95 | 0.000000 | tot_hi_cred_lim | 0.000000 | 0.0 |
| **104** | 12 | mths_since_last_credit_pull_d:>56.3 | 0.000000 | mths_since_last_credit_pull_d | 0.000000 | 0.0 |
| **105** | 13 | mths_since_issue_d:>93.2 | 0.000000 | mths_since_issue_d | 0.000000 | 0.0 |
| **106** | 14 | mths_since_recent_inq:>18.75 | 0.000000 | mths_since_recent_inq | 0.000000 | 0.0 |
| **107** | 15 | grade:E_F_G | 0.000000 | grade | 0.000000 | 0.0 |

- Based on Coefficients of LR Model
- Fitted to FICO's Credit Scoring Range (300 to 850)

# Creating a Scorecard

| | Feature name | Coefficients |
|---|---|---|
| 0 | Intercept | 3.928058 |
| 1 | grade:A | 0.393112 |
| 2 | grade:B | 0.340824 |
| 3 | grade:C | 0.232294 |
| 4 | grade:D | 0.168589 |
| 5 | home_ownership:OTHER_NONE_RENT | 0.006512 |
| 6 | home_ownership:OWN | 0.015960 |
| 7 | home_ownership:MORTGAGE_ANY | -0.016434 |
| 8 | int_rate:missing | 0.000000 |
| 9 | int_rate:<6.594 | 0.510612 |
| 10 | int_rate:6.594-7.878 | 0.464779 |

- Load WOE Binned Feature Names and Coefficients from Logistic Regression Model into DataFrame

# Creating a Scorecard

```
# create a list of all the reference categories, i.e. one category from each of the global features
ref_categories = ['int_rate:>20.718','last_pymnt_amnt:>12657.615','total_pymnt:>28483.595','acc_open_past_24mths:>9.6',
                  'inq_last_6mths:>1.6','num_tl_op_past_12m:>4.8','bc_open_to_buy:>35557.0','total_bc_limit:>55275.0',
                  'bc_util:>84.9','tot_hi_cred_lim:>499999.95','mths_since_last_credit_pull_d:>56.3',
                  'mths_since_issue_d:>93.2','mths_since_recent_inq:>18.75']
```

*Adapted from WOE Binning Python Method*

- Reference Column (Outliers- Max Values Outside Current Range) to be Dropped during WOE Binning

# Creating a Scorecard

| | index | Feature name | Coefficients |
|---|---|---|---|
| 0 | 0 | Intercept | 3.928058 |
| 1 | 1 | grade:A | 0.393112 |
| 2 | 2 | grade:B | 0.340824 |
| 3 | 3 | grade:C | 0.232294 |
| 4 | 4 | grade:D | 0.168589 |
| ... | ... | ... | ... |
| 103 | 11 | tot_hi_cred_lim:>499999.95 | 0.000000 |
| 104 | 12 | mths_since_last_credit_pull_d:>56.3 | 0.000000 |
| 105 | 13 | mths_since_issue_d:>93.2 | 0.000000 |
| 106 | 14 | mths_since_recent_inq:>18.75 | 0.000000 |
| 107 | 15 | grade:E_F_G | 0.000000 |

108 rows × 3 columns

- Contain Indexes, Coefficients and Binned Feature Names of the **WOE Binned Features** and *optional* **Reference Categories** (have 0 Value Score Contribution).

# Creating a Scorecard

Out[38]:

| | index | Feature name | Coefficients | Original feature name | Score - Calculation | Score - Preliminary |
|---|---|---|---|---|---|---|
| **0** | 0 | Intercept | 3.928058 | Intercept | 599.362503 | 599.0 |
| **1** | 1 | grade:A | 0.393112 | grade | 25.982462 | 26.0 |
| **2** | 2 | grade:B | 0.340824 | grade | 22.526535 | 23.0 |
| **3** | 3 | grade:C | 0.232294 | grade | 15.353307 | 15.0 |
| **4** | 4 | grade:D | 0.168589 | grade | 11.142754 | 11.0 |
| **...** | ... | ... | ... | ... | ... | ... |
| **103** | 11 | tot_hi_cred_lim:>499999.95 | 0.000000 | tot_hi_cred_lim | 0.000000 | 0.0 |
| **104** | 12 | mths_since_last_credit_pull_d:>56.3 | 0.000000 | mths_since_last_credit_pull_d | 0.000000 | 0.0 |
| **105** | 13 | mths_since_issue_d:>93.2 | 0.000000 | mths_since_issue_d | 0.000000 | 0.0 |
| **106** | 14 | mths_since_recent_inq:>18.75 | 0.000000 | mths_since_recent_inq | 0.000000 | 0.0 |
| **107** | 15 | grade:E_F_G | 0.000000 | grade | 0.000000 | 0.0 |

- Extract Original Feature by Splitting Feature Name by **:** symbol
- **Score – Calculation**: Calculate the Credit Score (resulting in **Continuous** Data Type)
- **Score – Preliminary**: Round Score – Calculation to **Round** Whole Numbers

# Creating a Scorecard

$$Score = Offset + Factor * \ln(Odds\ \ )$$

***Figure 1****: Score Calculation Equation of each 'Binned' Feature, adapted from Saddiq [20]*

$$Factor = \frac{Max\ FICO\ Score - Min\ FICO\ Score}{\sum(Max\ Coef\ of\ Main\ Feature) - \sum(Min\ Coef\ of\ Main\ Feature)}$$

***Figure 2:*** *Equation for Factor variable*

- The Score of Each WOE 'Binned' Feature can be calculated by the equation in **Figure 1**.
  - The **'Odds'** Variable for Each Feature is derived from the Probability of an Attribute being "good or bad" (non-default or default) [20] i.e Coefficient of Each WOE 'Binned' Feature in Trained LR Model
  - The **'Factor'** Variable for Each 'Binned' Feature "depend[s] on the approval rate" [20] derived by the Equation in **Figure 2** Divides the Value Difference of the **Maximum and Minimum** FICO scores (300 to 850) with the **value difference of the Summation of the Maximum and Minimum Coefficients** of Each WOE Binned Feature (e.g. 'grade:A' / 'grade:B' / 'grade:C') based off the Main Feature (e.g. 'grade').
  - In **Figure 1**, the **'Offset'** variable is *optional* and comes into play when the Maximum and Minimum Scores (Multiplication of Factor and Odds) are Found to be Outside the FICO range (300 to 850).
    - When Implemented, the **'Offset'** would be **added/subtracted** from both the Multiplied Values (Maximum and Minimum Scores of the Model), resulting in a **Corrected Score** Predicted for Each 'Binned' Feature.
      The Customer's (FICO-based) Credit Score is Tabulated by Adding each Feature's Score together.

# Creating a Scorecard

```
In [39]: # check the min and max possible scores of our scorecard
         min_sum_score_prel = df_scorecard.groupby('Original feature name')['Score - Preliminary'].min().sum()
         max_sum_score_prel = df_scorecard.groupby('Original feature name')['Score - Preliminary'].max().sum()
         print(min_sum_score_prel)
         print(max_sum_score_prel)
         #Guideline from FICO: from 300 to 850

         301.0
         849.0
```

- Check if the Scorecard Values' Range is within FICO™ Range (300 to 850)

# Creating a Scorecard

| | Intercept | grade:A | grade:B | grade:C | grade:D | home_ownership:OTHER_NONE_RENT | home_ownership:OWN | home_ownership:MORTGAGE_ANY | int_rate:missing |
|---|---|---|---|---|---|---|---|---|---|
| **0** | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| **1** | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| **2** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| **3** | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| **4** | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

5 rows × 92 columns

- Customers' Data Collected During the Company's Loan Application Process would then need to be Transformed with WOE Binning (in Picture) Found in the Model Pipeline.

# Creating a Scorecard

```
df_scorecard['Score - Preliminary'] = df_scorecard['Score - Calculation'].round()
df_scorecard
```

Out[38]:

| | index | Feature name | Coefficients | Original feature name | Score - Calculation | Score - Preliminary |
|---|---|---|---|---|---|---|
| 0 | 0 | Intercept | 3.928058 | Intercept | 599.362503 | 599.0 |
| 1 | 1 | grade:A | 0.393112 | grade | 25.982462 | 26.0 |
| 2 | 2 | grade:B | 0.340824 | grade | 22.526535 | 23.0 |
| 3 | 3 | grade:C | 0.232294 | grade | 15.353307 | 15.0 |
| 4 | 4 | grade:D | 0.168589 | grade | 11.142754 | 11.0 |
| ... | ... | ... | ... | ... | ... | ... |
| 103 | 11 | tot_hi_cred_lim:>499999.95 | 0.000000 | tot_hi_cred_lim | 0.000000 | 0.0 |
| 104 | 12 | mths_since_last_credit_pull_d:>56.3 | 0.000000 | mths_since_last_credit_pull_d | 0.000000 | 0.0 |
| 105 | 13 | mths_since_issue_d:>93.2 | 0.000000 | mths_since_issue_d | 0.000000 | 0.0 |
| 106 | 14 | mths_since_recent_inq:>18.75 | 0.000000 | mths_since_recent_inq | 0.000000 | 0.0 |
| 107 | 15 | grade:E_F_G | 0.000000 | grade | 0.000000 | 0.0 |

| | Intercept | grade:A | grade:B | grade:C | grade:D | home_ownership:OTHER_NONE_RENT | home_ownership:OWN | home_ownership:MORTGAGE_ANY | int_rate:missing |
|---|---|---|---|---|---|---|---|---|---|
| 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 1 | 0 |
| 2 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 3 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 4 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 |

5 rows × 92 columns

- Reshape Customers' Dataset from (…, **92**) to (**108**, …) to Perform **Dot Matrix Multiplication**

# Creating a Scorecard

```
In [46]: # matrix dot multiplication of test set with scorecard scores
         y_scores = X_test_woe_transformed.dot(scorecard_scores)
         y_scores.head()
```

Out[46]:

| | 0 |
|---|---|
| 0 | 623.0 |
| 1 | 421.0 |
| 2 | 414.0 |
| 3 | 643.0 |
| 4 | 506.0 |

- Perform Dot Matrix Multiplication to get **Final Credit Scores**

# Thank You

# Q&A