Research Square

Preprints are preliminary reports that have not undergone peer review.
They should not be considered conclusive, used to inform clinical practice,
or referenced by the media as validated information.

Gini calculation and rule performance interpretation

Meenal Badki

meenal312b@gmail.com

Method Article

Keywords: model metrics, rule performance, Gini index, accuracy ratio

Posted Date: July 30th, 2024

DOI: https://doi.org/10.21203/rs.3.rs-4822777/v1

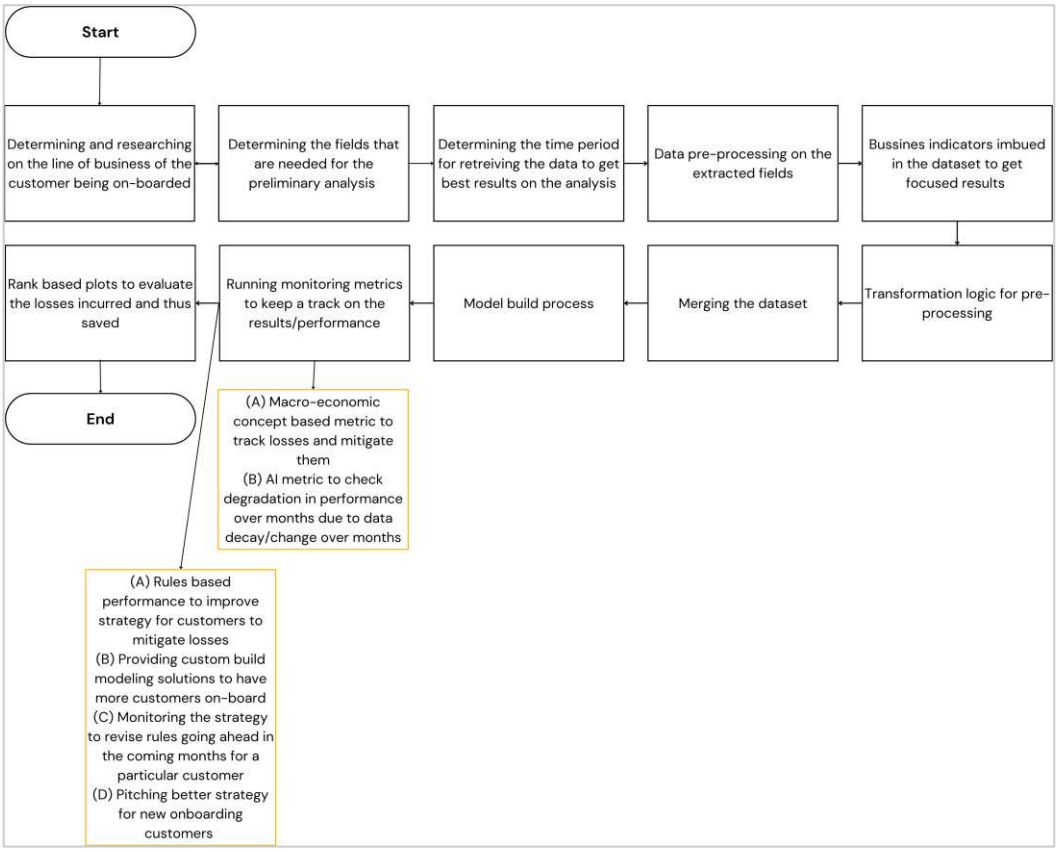
License: This work is licensed under a Creative Commons Attribution 4.0 International License.
Read Full License

Additional Declarations: The authors declare no competing interests.

Gini calculation and rule performance interpretation

Abstract:
The Gini coefficient is a widely used measure of income inequality within a population. This paper investigates the application of the Gini coefficient concept to calculate fraud risk based transaction amount losses. The paper is essentially divided into two parts: (1) Interpretation and implementation of Lorenz curve in calculating fraud risk in a real-time scenario for online e-commerce merchants/customers for various industries. Implementing Gini is useful for verifying fraud-based metrics or saving fraud losses after a machine learning model has been implemented and deployed in production. (2) The rules automation system is helpful in optimizing and improving strategies required for deployment on a real-time transaction based platform in the financial services world where the real-time decisioning system for a customer on board plays an important role. The study concludes that Gini coefficient remains a robust tool for evaluating risk based losses, thus helping financial institutions make informed strategy-driven decisions, and further also explores the business rule sets that help to optimize the strategic decisions.
Keywords: model metrics, rule performance, Gini index, accuracy ratio

Introduction:
The below diagram helps to understand the preliminary implementation to build a process to get Gini results in fraud risk domain as well as rules optimization + strategy to save losses for a financial institution. As such, Gini is a widely adopted macroeconomic phenomenon that helps to compare the economic status and income inequality amongst different states in a country, but its application on risk management or risk management operations to help save losses hasn't been explored.
Fraud risk and its fundamentals (a significant concern for organizations of all sizes and types):
Fraud risk puts all kinds of organizations at *financial* risk– financial institutions, legal and tax firms, nonprofits, government benefits agencies (such as those that manage Medicaid), large corporations, and small businesses. Types of fraud can be in accounting, mail fraud, check fraud, payroll fraud or identify theft, bank fraud, healthcare fraud. In case of institutions that handle frauds pertaining to e-commerce domain transactions (fraud spans over chargebacks management, account takeover, etc), we try to determine whether a fraud by a customer is a friendly fraud or a hostile fraud.
Benefits of fraud-risk management are : reduced financial losses due to undetected fraud, reduced costs of responding to fraud, more effective corporate governance, effective custom modeling solutions based strategy etc.
The steps mentioned in the figure are self-explanatory, but the area of focus remains the step where we run monitoring metrics to keep a track on the results/performance as well as what we do after we have monitored the results (as highlighted in orange).



RFigure [0]: Steps for a business process

1. Idea behind Gini: A measure of distribution or statistical dispersion used to represent income across a population is the Gini coefficient, also known as the Gini ratio. The dispersion of wealth or income among population members is measured by this coefficient. One of the most widely used metrics for assessing economic inequality is the Gini coefficient. A population's income and wealth distribution are perfectly equal when the coefficient is zero. When one member of the population receives all income and the others receive nothing, this is referred to as perfect inequality and is represented by a coefficient of one.

1.1 How to define Gini: There are two main methods of calculating Gini coefficient.
The first method can be demonstrated with an example where two people randomly bump into each other on the street. By comparing their incomes, we can determine who is wealthier or we can analyze the expected gap between their incomes. This expected gap is what the Gini coefficient measures, expressing the anticipated absolute difference in income between individuals relative to the population's average income. The coefficient is calculated as the expected gap in relation to twice the mean income, with perfect inequality resulting in a coefficient of 1, indicating one person having all the income **[2]**. Conversely, perfect equality results in a coefficient of 0, signifying that everyone earns the same and there are no income disparities between individuals.
The second method is a visual definition of Gini coefficient.
The income share that each fifth of a fictitious population receives is displayed in the left panel. This data is plotted cumulatively in the right panel. This is referred to as a "Lorenz curve". If everyone in a population had the same amount of money, the Lorenz curve would be a straight diagonal line, with 10% of the population earning 10% of the total income, 20% earning 20% of the total income, and so on. The "line of equality" in the chart represents this. However, incomes are not distributed equally in the hypothetical population depicted in the chart. Thirty percent goes toward the bottom sixty percent of the population.
The Gini coefficient captures how far the Lorenz curve falls from the 'line of equality' by comparing the areas A and B, as calculated in the following way:
Gini coefficient = A/(A + B)
The Lorenz curve is simply the "line of equality" where incomes are distributed exactly equally. Since

the area A is zero, the Gini coefficient is also zero. The Lorenz curve will run along the bottom axis of the chart in the scenario where one person receives all income and everyone else receives none; up until the very last person, the cumulative share of income is zero. The Gini coefficient will be equal to 1, and area B will be zero. **Figure [1]** could be used to understand this.

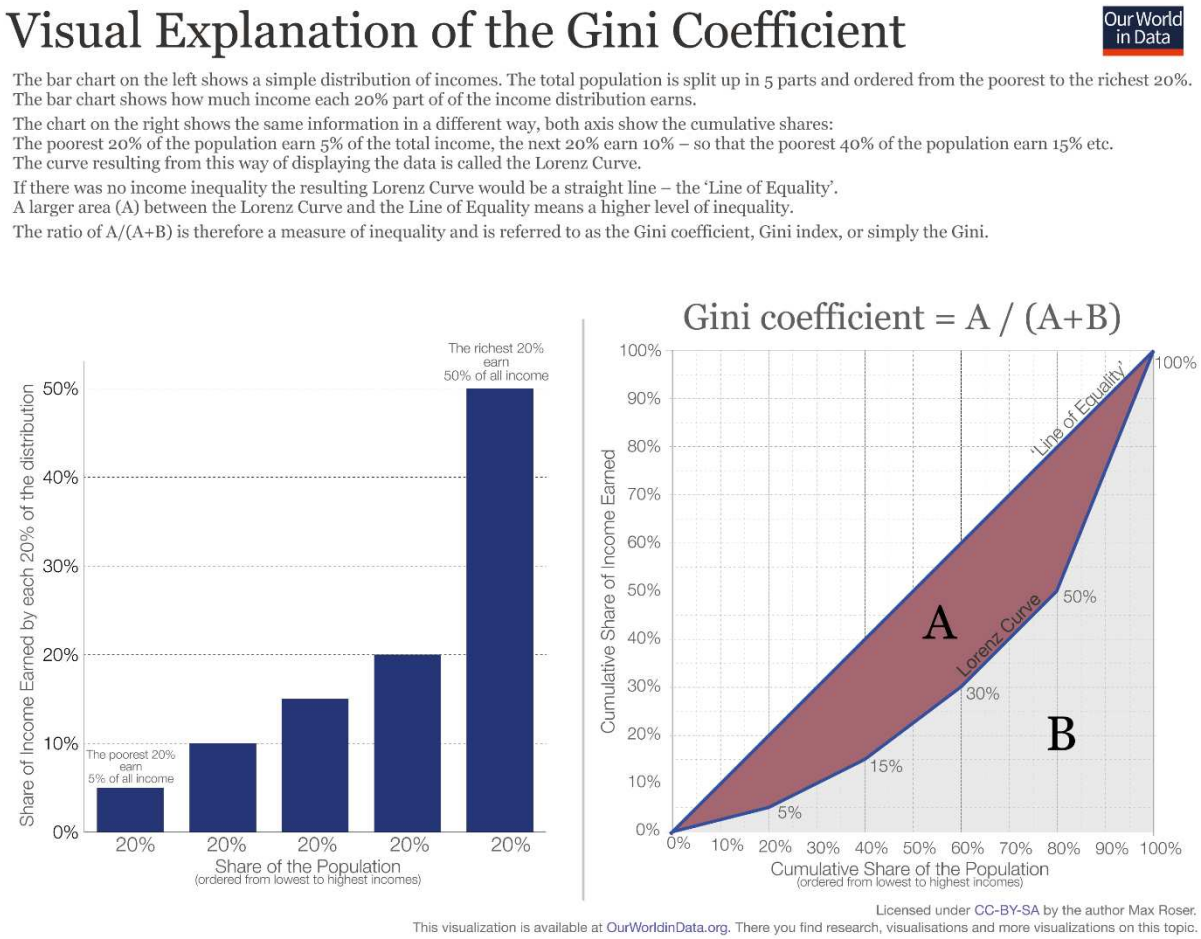


Figure [1]: self-explanatory figure for gini Lorenz curve, more references in **[1]**

Such summary measures implicitly make decisions about which metrics should be prioritized when assessing inequality **[3]**. Consider comparing two fictitious populations. In one, the incomes of the rich are significantly higher than those of the middle class, but the poorer class's incomes are only marginally lower. In another, things are completely reversed: the rich earn slightly more than the middle class, but the poor are significantly impoverished. Your response will depend on your personal assessments of how these various gaps, in various parts of the distribution, compare in terms of the contribution they make to overall inequality. In which population would you say inequality is highest? Such value judgments are implicitly included in the definition of an inequality measure in mathematics. This holds true for all measures of inequality, including the Gini coefficient. One characteristic of the Gini is its sensitivity to changes around the middle of the distribution rather than at the top and bottom, which sets it apart from some other metrics of inequality. **[4]**
This is demonstrated by the examples in the chart below. It displays the comparison between the Gini and two other measures of inequality for four countries: the income share received by the richest 10% and 1%, respectively. The chart displays each measure's relative change over time to enable more straightforward comparisons. We observe that relatively small changes in the Gini coefficient can be accompanied by significantly larger relative changes in the income share received by the wealthiest 1 percent. That holds true in both situations—where inequality is declining in Uruguay and increasing in the United States. The top 10 percent share of income, however, is tracked far more closely by the Gini coefficient. For the very top of the distribution, the Gini's lack of sensitivity to changes in that region is primarily concerning.

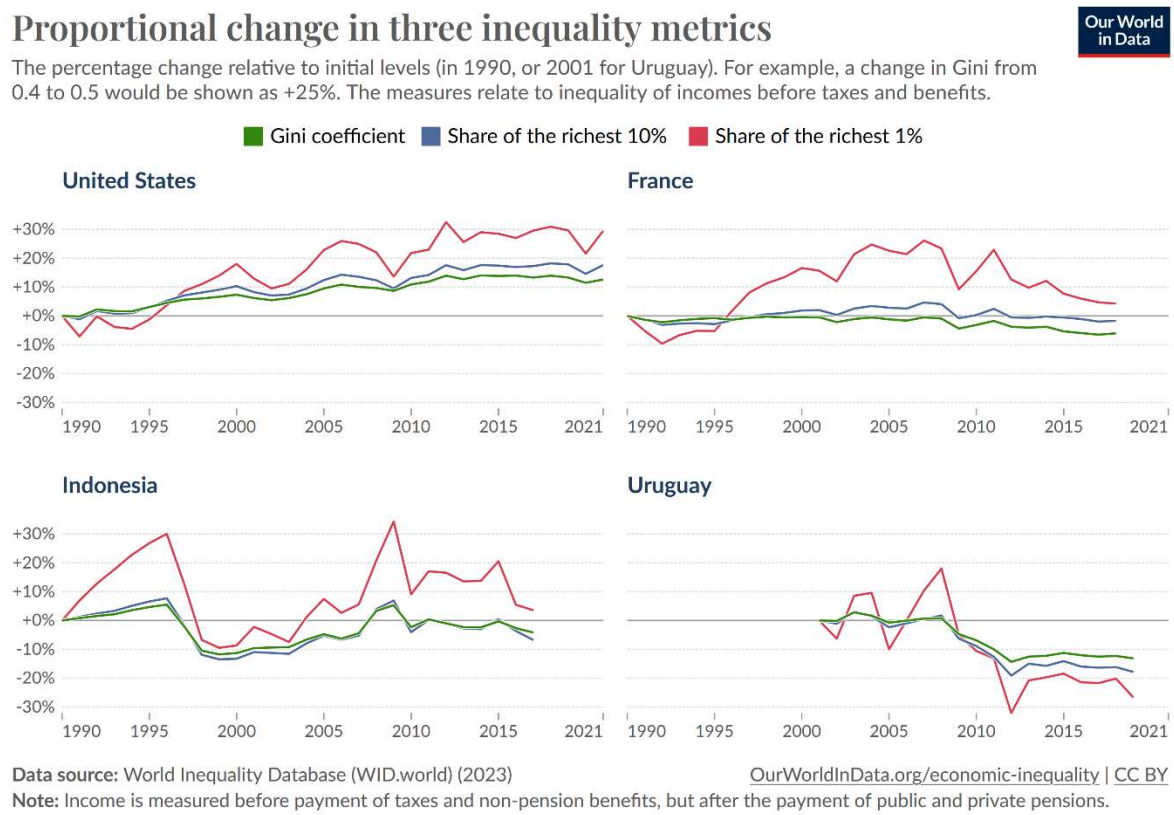


Figure [2]: Comparison between interpretations of definition of Gini (as explained above)

Methods:

1.2 Two approaches to calculate Gini metrics: First approach is calculating Gini using Area under curve concept, second approach is calculating capture table metrics.

Steps followed before Gini calculation: In financial transactions processing, we follow MLOps steps to arrive to model probabilities being assigned to every transaction.

We want to compare the Gini for old model vs Gini for new model.

Transactio	Card_men	Transactio	Weights	Transactio	Fraud_ind	model_pro
T001	aaa	00:00	1	1000	0	0.0005
T002	bbb	00:05	50	2000	1	0.99999
T003	ccc	00:10	1	3000	0	0.00001
T004	ddd	00:15	50	4000	1	0.99888

Table [1]: This table describes Transaction_id, cardmember_name, transaction_timestamp, weights for fraud + non_fraud transactions, transaction_amounts, fraud_ind, model_probability_score (depending upon the frauds + non_frauds)

Intepretation of frauds + non_frauds from this **Table [1]** : more weights are assigned to transactions which have lesser model probability score and fraud_ind = 1 (being a fraud transaction) while lesser weights are assigned to transactions which have more model probability score and fraud_ind = 0 (being a non_fraud transaction or a good transaction). For the task of fraud risk prevention in financial transactions processing, we would have more non_fraud transactions/good transactions/transactions with lesser model probability score values/transactions with fraud_ind = 0 hence we would assigned them lesser weights as compared to fraud transactions to maintain sample efficiency/proper weightage assigned to all the transactions alike – weights assigned to transactions play a major part in determining gini-coefficient metric values as well as capture table metrics we calculate.

transact	card	transactio	weig	frau	transac	model_p	weighted	cumul	cun	cumulati	cumulative
T_0001	aaa	00:00:00	50	1	1000	0.99832	50000	50	1	1000	500000
T_0002	bbb	00:00:01	50	1	2000	0.99994	100000	100	2	3000	600000
T_0003	ccc	00:00:02	1	0	3000	0.00005	3000	101	2	6000	603000
T_0004	ddd	00:00:03	1	0	4000	0.0004	4000	102	2	10000	607000
T_0005	eee	00:00:04	1	0	5000	5E-06	5000	102	2	15000	612000
T_0006	fff	00:00:05	50	1	6000	0.99993	300000	103	3	21000	912000

Table [2]: Cumulative column calculations

Interpretations of the **Table [2]**: This table consists of transaction_id, cardmember_name (for some cases we have cardmember_name removed from our dataset since its not a personally identifiable dataset), transaction_timestamp (helpful in cases where we want to fetch exact day or exact one week/one month data for multiple ecommerce-clients on the go), weights, fraud_ind (involves many parameters before we assign fraud_ind = 0 or 1), transaction_amount (either USD or EUR or depending on whichever area of interest we are working in), model_probability_scores, weighted_transaction_amount (weights*transaction_amount), cumulative_weights, cumulative_frauds, cumulative_transaction_amount, cumulative_weighted_transaction_amount (all the cumulative values are derived fields).

How financial transactions look like?

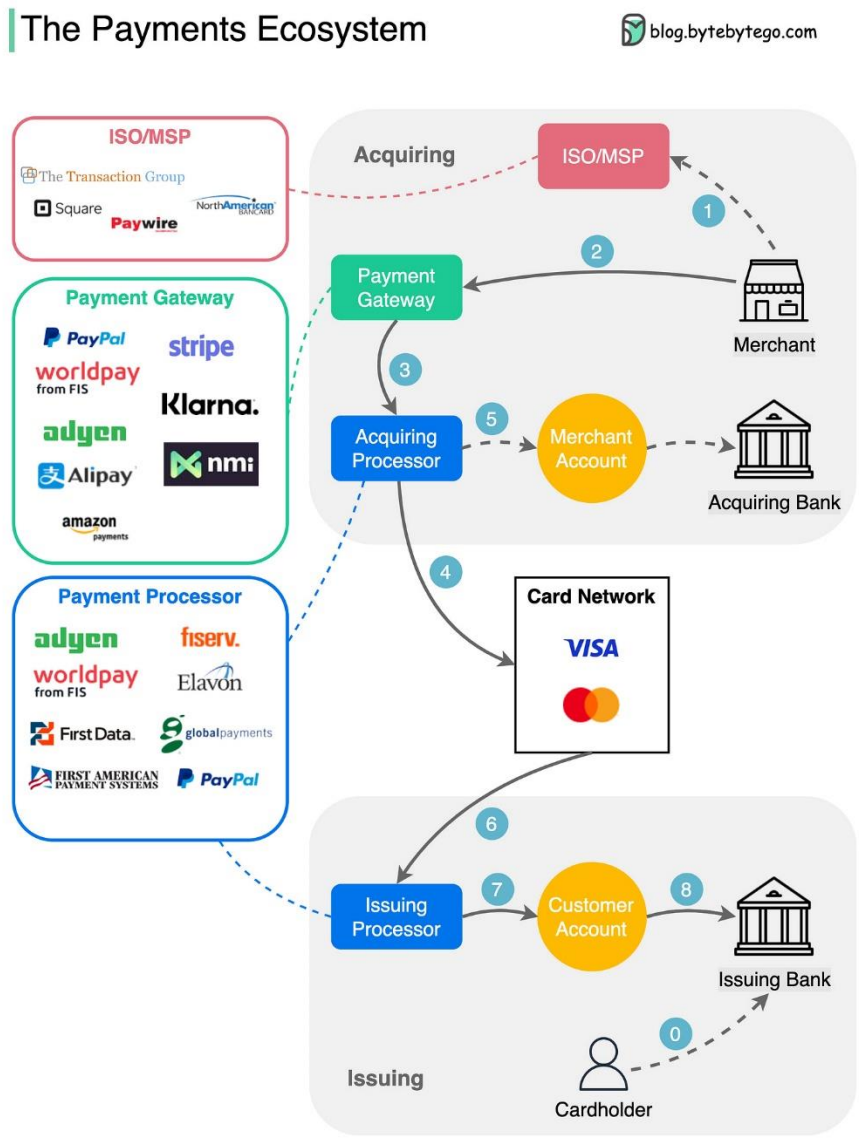


Figure [3]: In the above referenced **Figure [3]** is included in this paper to get hold of the data fields that ideally goes into real-time transaction database. A simple understanding of steps-
Steps 0-1: The cardholder opens an account in the issuing bank and gets hold of debit/credit card. The merchant registers with ISO (Independent Sales Organisation) or MSP (Member Service Provider) for in-store sales. ISO/MSP partners with payment processors to open merchant accounts.
Steps 2-5: The acquiring process - The payment gateway accepts the purchase transaction and collects payment information. It is then sent to a payment processer, which uses customer information to collect payments. The acquiring processer sends the transaction to the card network. It also owns and operates the merchant's account during settlement, which doesn't happen in real-time.
Steps 6-8: The issuing process - The issuing processor talks to the card network on the issuing bank's behalf. It validates and operates the customer's account.
To specifically consider risk scenarios happening in ecommerce based transactions systems we would consider online payment on online ecommerce websites or POS terminal payments/card-not-present(UPI), card present (ATM) based payments, these are the fields that we could check on payment receipts (which are of significance in finance data management) – CIN (corporate identification number : this could be viewed as close to what ISO/MSP identify with), Billing location, Mode of payment, Txn/Auth/Order ID (order placed consisting average items in the bucket), Date and Time (date and time of the transaction taking place), Coupon Number/Coupon Redeem (if

there are coupon discounts valid/applicable for that item/product purchased). For QR Code/UPI transaction based accounts, these are the fields of importance: Date + Time (date and time of transaction taking place from QR scanner or UPI mode), TID (terminal identification number – POS terminal or any other terminal), MID (Merchant ID), Client ID, Batch ID, Bill Num, Payment mode, Txn ID (numeric value specifying a specific transaction), Host Txn ID (alphanumeric value belonging to a certain network ID on which a particular transaction with Txn ID has been made/transacted), Merchant UPI ID (UPI ID on which acquiring bank is hosted), Client/customer UPI ID (UPI ID on which issuing bank is hosted).

Let's say the merchant is Walmart having certain CID, MID and the merchant's Acquiring bank is ABC Bank and customer's issuing bank is DEF. ABC Bank would have merchant UPI ID (which is different from MID), and DEF Bank would have client/customer ID (which is different from Client ID). Hence, once the transaction is completed, we would get the aforementioned fields printed on payment receipts according to mode of payment. From a transaction perspective, we can look at it from a global/local/client scope. For card present based payments/ATM – EMV dipping happens (we have similar fields to be noted).

1.2.1 Steps to follow preceding GINI calculation:

Before we deep dive in how transactions are processed to get GINI results, we follow these steps (to get model probability score values):

1) We perform extraction/transformation/load process for intended months for intended ecommerce merchant clients by mapping input columns according to the ones ingested in the database.

```
def MapR_string_to_type(self, MapR_Mapping):
    retyped_dict = {}
    for column_name, column_attr in MapR_Mapping.items():
        retyped_dict[column_name] = {}
        retyped_dict[column_name]['MapR_Name'] = column_attr['MapR_Name']
        if column_attr['MapR_Type'] == 'StringType':
            retyped_dict[column_name]['MapR_Type'] = StringType()
        elif column_attr['MapR_Type'] == 'LongType':
            retyped_dict[column_name]['MapR_Type'] = LongType()
        elif column_attr['MapR_Type'] == 'TimestampType':
            retyped_dict[column_name]['MapR_Type'] = TimestampType()
        elif column_attr['MapR_Type'] == 'DecimalType_29_15':
            retyped_dict[column_name]['MapR_Type'] = DecimalType(29, 15)
        elif column_attr['MapR_Type'] == 'DecimalType_28_5':
            retyped_dict[column_name]['MapR_Type'] = DecimalType(28, 5)
        else:
            retyped_dict[column_name]['MapR_Type'] = StringType()
    return retyped_dict

coreP_MapR_Mappings = MapR_string_to_type(yaml.load(open('CoreP_MapR_Mappings.yaml','r'))) #"""CHECK HOW TO INCLUDE YAML FILE FROM SCRIPT/MAPR LOCATION"""
extract_schema = StructType([StructField(coreP_MapR_Mappings[colX]['MapR_Name'],coreP_MapR_Mappings[colX]['MapR_Type']) for colX in extract_columns])

extract_mapr_cols = [F.col(coreP_MapR_Mappings[col_nm]['MapR_Name']).alias(col_nm) for col_nm in extract_columns]
```

Figure[4]: Mapping of input variables stored in development/production mode with different fields consisting of different data types. This is one of the crucial steps we perform in ETL processes.

As illustrated in **Figure[4]**, we have performed mapping of the variables.

2) We perform univariate/numerical/categorical variables quality checks:

```
def univariate(spdf, var_list, out_path):

    def univar(sp_df, c):

        df = sp_df.groupby(F.lit(c)).agg(F.lit(c).alias('Variable'),\
                    sum(col(c).isNotNull().cast("int")).alias("N"),\
                    sum(col(c).isNull().cast("int")).alias("N_MISS"),\
                    ceil(avg(col(c))).alias("MEAN"),\
                    min(col(c)).alias("MIN"),\
                    max(col(c)).alias("MAX"),\
                    round(stddev(col(c)), 2).alias("STDDEV"),\
                    F.expr("""percentile_approx("""+c+""", 0.01)""").alias("P_1"),\
                    F.expr("""percentile_approx("""+c+""", 0.05)""").alias("P_5"),\
                    F.expr("""percentile_approx("""+c+""", 0.10)""").alias("P_10"),\
                    F.expr("""percentile_approx("""+c+""", 0.25)""").alias("P_25"),\
                    F.expr("""percentile_approx("""+c+""", 0.50)""").alias("P_50"),\
                    F.expr("""percentile_approx("""+c+""", 0.75)""").alias("P_75"),\
                    F.expr("""percentile_approx("""+c+""", 0.90)""").alias("P_90"),\
                    F.expr("""percentile_approx("""+c+""", 0.99)""").alias("P_99")).select('Variable', 'N', 'N_MISS', 'MEAN',
                                'STDDEV', 'MIN', 'P_1', 'P_5', 'P_10', 'P_25', 'P_50', 'P_75', 'P_90', 'P_99','MAX').\
                    withColumn('is_single_valued', F.when(col('MIN') == col('MAX'), "Yes").otherwise("No"))

        return df

    i = 1
    for c in var_list:
        sdf = spdf.select(c)
        sdf.persist()
        df = univar(sdf, c)
        sdf.unpersist()
        df.write.option('header', 'true').parquet(out_path + ".temp/" + "qc_var_{}".format(i))
        i = i + 1

    for j in range(1,len(var_list)+1):
        df = spark.read.option('header', 'true').parquet(out_path + ".temp/" + "qc_var_{}".format(j))
        if j==1:
            rdf = df
        else:
            rdf=rdf.union(df)

    rdf.coalesce(1).write.mode('overwrite').option('header', 'true').csv(out_path + "univariate_fn")

    fs = (sc._jvm.org.apache.hadoop.fs.FileSystem.get(sc._jsc.hadoopConfiguration()))
    path = out_path + ".temp"
    fs.delete(sc._jvm.org.apache.hadoop.fs.Path(path), True)
    fs.close()
```

Figure[5]: We perform univariate analysis on a single input independent variable to check the distribution of that variable. Here, we are checking the distribution of the data values of that variable from 1 percentile to 99 percentile, mean, mode, standard deviation, minimum value, maximum value, total number of values, missing values, whether the variable is single valued or not, and distinct count of all values contained in that variable.

3) We sample the data based on its total number of fraud transactions or non-fraud transactions

4) Then we create exclusion + tagging indicators based on fields like hierarchy_id, transaction_amount, recommendation_code, scoring7_disposition, scoring7_rules_tripped for negative exclusions pertaining to negative history, scoring7_rules_tripped for positive exclusions pertaining to positive history, test exclusions, friendly fraud exclusions, cancel exclusions, auto-reject exclusions, other exclusions, client based exclusions and so on and forth, these would vary depending on the client.

5) These exclusion and tagging indicators contribute to creating a final exclusion indicator (xc_fin)

6) We create normal variables/base variables to preprocess the input tagged data or it could be called feature transformation step. There are various set of function calls defined to perform preprocessing of the data before we create velocity variables and fetch DRV variables.

7) We create velocity variables which is defined as how many times a person has having a base_var (example: billing_email) made transactions with a certain transaction amount across a span of 1 hour or 5 hours or 24 hours or 120 hours.

or how many times a person has having a base_var (example: cc_number) made transactions with billing_phone_number across a span of 1 hour or 5 hour or 24 hours or 120 hours and so on and so forth combinations. Here, count of distinct base_var is calculated against number of times a count_var occurs. We follow this simple algorithm to calculate the value of velocity variables and inputs of base_var and count_var and time_periods are also explained.

```
def velocity(data, lookup, result, key, base_vars, count_vars, ts_var, time_periods):
      data = data.repartition(1000, base_vars)

      data.createOrReplaceTempView("data")
      lookup.createOrReplaceTempView("lookup")

      base_vars = base_vars.split('/')
      count_vars = count_vars.split('/')
      time_periods = time_periods('/')

      for base_var in base_vars:
            for time_period in time_periods:
                  i = 1
                  count_var_string = ""
                  count_final_string = ""
                  for count in count_vars:
                        velocity_var = "fn_vl1_" + base_var + "_" + count_var + time_period
                        count_var_string = count_var_string + "l." + count_var + " as count_var_" + str(i) + ","
                        count_final_string = count_final_string + "count(distinct count_var_" + str(i) + ") as " + velocity_var+","
                        i = i + 1
            count_var_string = count_var_string[:-1]
            count_final_string = count_final_string[:-1]
            query = """
            Select """ + key + """,""" + count_final_string + """
            from
            (
            Select d.""" + key + """,""" + count_var_string + """
            from d left join lookup l
            on d.""" + base_var + """ >= l.""" + base_var + """
            and d.""" + ts_var + """ >= l.""" + ts_var + """
            and l.""" + ts_var + """ >= (d.""" + ts_var + """ - interval '""" + time_period + """' hour)
            )
            group by """ + key
            velocity_df = spark.sql(query)
            velocity_df = velocity_df.repartition(2048, key)
            result = result.join(velocity_df, key, "left")
            result.repartition(100).write.mode("overwrite").parquet(result_path)
            result = spark.read.parquet(result_path)

      return result

velocity(data, lookup, result, key,base_vars,count_vars, ts_var,time_periods)
```

Figure[6]: Algorithm for creation of velocity variable for different combination of base_var and count_var. As explained we have data and lookup as input parameters. We parse through every base_var slots as well as every time_period slots.

Lets say we have count_var = r_txn_id (transaction_id)

ts_var = r_txn_ts (transaction timestamp)

base_var = cc_numbr (credit card number)

velocity_var = “fn_vl1_cc_numbr_r_txn_id1” (derived field)

count_var_string = “l.r_txn_id as count_var_1,”

count_final_tring = “count(distinct count_var_1) as fn_vl1_cc_numbr_r_txn_id1,”

query = Select transaction_id, count(distinct count_var_1)

from

(

Select d.rn_txn_id, count_var_1

from d left join lookup l

on d.cc_numbr >= l.cc_numbr

and d.r_txn_ts >= l.r_txn_ts

and l.r_txn_ts >= (d.r_txn_ts – interval 1 hour)

)

group by r_txn_id

velocity_df = spark.sql(query) #we execute the query mentioned above

result = result.join(velocity_df, “r_txn_id”, “left”) #we perform a join on all those transactions which are already in result dataframe with the newly created velocity dataframe

result = spark.read.parquet(result_path) #we read the result data from the intended result_path

How do we define data in the input function call as mentioned in the **Figure[6]**: input data which we get as an output of tagging data, we generally consider month by month or week by week or day by day dataset against larger lookup dataset.

How do we define lookup data in the input function call as mentioned in the **Figure[6]**: this is the entire dataset consisting of all months (it is used as a lookup dataset to check the count of velocities), also we exclude test hierarchies from this dataset while creating velocity variables

How do we define result data in the input function call as mentioned in the **Figure[6]**: this is just selecting transaction_id/id values
How do we define other fields in the input function call as mentioned in the **Figure[6]**
key = “txn_id” (transaction_id value to perform certain joins)
base_vars = deviceid/ ipaddress/ cc_number/ bill_eml/ bill_phn
count_vars = deviceid, ipaddress, cc_number, bill_eml, bill_phn/r_txn_id
velocity(data, lookup, result, key, base_vars = base_var, count_vars = count_var, ts_var = “txn_ts”, time_periods = “1/5/24/120”)
7) we create sum velocity variable, gift velocity variables
base_vars = deviceid/ ipaddress/ cc_numbr/ bill_eml/ bill_phn/r_txn_id
count_vars = txn_amount
sum_velocity(data, lookup, result, key, base_vars, count_vars, ts_var, time_periods = “1/5/24/120”).
We check count of sum of base_var that is calculated against number of times a count_var occurs. We follow this simple algorithm to calculate the value of velocity variables and inputs of base_var and count_var and time_periods are also explained.

```
def sum_velocity(data, lookup, result, key, base_vars, count_vars, ts_var, time_periods):
      data = data.repartition(1000, base_vars)

      data.createOrReplaceTempView("data")
      lookup.createOrReplaceTempView("lookup")

      base_vars = base_vars.split('/')
      count_vars = count_vars.split('/')
      time_periods = time_periods('/')

      for base_var in base_vars:
            for time_period in time_periods:
                  i = 1
                  count_var_string = ""
                  count_final_string = ""
                  for count in count_vars:
                        velocity_var = "fn_vl1_" + base_var + "_" + count_var + time_period
                        count_var_string = count_var_string + "l." + count_var + " as count_var_" + str(i) + ","
                        count_final_string = count_final_string + "sum(count_var_" + str(i) + ") as " + velocity_var+","
                        i = i + 1
             count_var_string = count_var_string[:-1]
             count_final_string = count_final_string[:-1]
             query = """
             Select """ + key + """,""" + count_final_string + """
             from
             (
             Select d.""" + key + """,""" + count_var_string + """
             from d left join lookup l
             on d.""" + base_var + """ >= l.""" + base_var + """
             and d.""" + ts_var + """ >= l.""" + ts_var + """
             and l.""" + ts_var + """ >= (d.""" + ts_var + """ - interval '""" + time_period + """' hour)
             )
             group by """ + key
             velocity_df = spark.sql(query)
             velocity_df = velocity_df.repartition(2048, key)
             result = result.join(velocity_df, key, "left")
             result.repartition(100).write.mode("overwrite").parquet(result_path)
             result = spark.read.parquet(result_path)

      return result

sum_velocity(data, lookup, result, key,base_vars,count_vars, ts_var,time_periods)
```

Figure[7]: Algorithm for creation of sum velocity variable for different combination of base_var and count_var. As explained we have data and lookup as input parameters. We parse through every base_var slots as well as every time_period slots.
8) We create gift card velocity variables to calculate variables which consists of gift card data
base_vars = ipadress/ bill_eml/ deviceid/ cc_numbr/ bill_phn
count_vars = transaction_id
input data = production pulled data which has been pushed to production environment, while pulling it with fc_ind_bought_gc == 1(which mentions that we are pulling data with just gift card variables), lookup data = tagged preprocessed data.
Gift card velocity variables are created by referencing **Figure[7]** algorithm.
DRV variables are extracted for the intended set of clients for intended months from another process which comes from another data source and requires another set of

optimizations/computations/preprocessing capabilities. These are called as dynamic resource vector variables which are FTG, chargeback, history based variables (negative/positive history), DFFS (direct fraud first seen), DFLS (direct fraud last seen) where the scope of these variables are C (client based DRV variables), G (global level DRV variables), M (merchant level DRV variables)
9) We perform production pull of intended input columns of data for intended client in the time span of intended months
10) We perform data merge process of the initially pulled data and production data to get the most updated dataset to perform our rule decisioning process that helps in monarch decision-making.
11) Post implementation testing: We compare base variables created offline with the same set of base variables pushed to production. This would ensure that we have uniformity across the platforms in production and development modes.
12) Groovy automation process:
This the pseudo code that was developed to extract

```
fn_vars = []
raw_vars = []
vars_ = []
set_as_miss_fn = []

for ln in lines:
    line = re.sub(r'^.*?\.','',ln)
    if '(' not in line:
        out1.writelines(line)
        continue
    between_braces = re.findall(r'\((.*?)\)\s*$', line)[0]
    if len(between_braces) < 1:
        out1.writelines(line)
        continue
    for ele in between_braces.split(','):
        ele = ele.strip()
        if ele.startswith('var_def'):
            continue
        if ele.startswith("var_name") or ele.startswith("var_cnm") or ele.startswith("var_csp") or ele.startswith("var_csv") or \
           ele.startswith("var_rvo"):
           var = re.findall(r'=\s*\"(.*)\"',ele)[0]
           var = var.strip()
           if(var.startswith('fntdf_')) or (var.startswith('fnrto_')) or (var.startswith('fc_')) or (var.startswith('fn_')) or (var.startswith('fname')) or (var.startswith('clien')):
                line = ' /*Fin*/' + line
                if var not in fn_vars:
                    fn_vars.append(var)
                else:
                    out3.write(var + " already created once\n\n")
           else:
                if var not in vars_:
                    vars_.append(var)
        elif ele.startswith("phy_name"):
            var = re.findall(r'=\s*\"(.*)\"', ele)[0]
            var = var.strip()
            key = str(var)
            if var not in raw_vars:
                raw_vars.append(var)
            else:
                out3.write(var +"already used as a raw_var before")

            elif ele.startswith("test_var"):
                var = re.findall(r'=\s*\"(.*)\"',ele)[0]
                var = var.strip()
                if (var not in vars_) and (var not in raw_vars):
                    out3.write(var + "undefined usage")
                if var in fn_vars:
                    out3.write(var+" fin var used as input variable")
          elif ele.startswith("lat"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")

          elif ele.startswith("long"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")

          elif ele.startswith("name"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")

          elif ele.startswith("str"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")

          elif ele.startswith("amount"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")

          elif ele.startswith("country"):
              var = re.findall(r'=\s*\"(.*)\"',ele)[0]
              var = var.strip()
              if (var not in vars_) and (var not in raw_vars):
                  out3.write(var + " undefined usage\n\n")
              if var in fn_vars:
                  out3.write(var+" fin var used as input variable")
```

```
elif ele.startswith("cc_exp_var"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("order_time"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("time_pre"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("time_post"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("ts_dt"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("email"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")
elif ele.startswith("zip_input"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("phone"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("ts_var"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("cnt"):
    var = re.findall(r'=\s*\"(.*)\"',ele)[0]
    var = var.strip()
    if (var not in vars_) and (var not in raw_vars):
        out3.write(var + " undefined usage\n\n")
    if var in fn_vars:
        out3.write(var+" fin var used as input variable")

elif ele.startswith("base_var"):
    inp = re.findall(r'=\s*\"(.*)\"',ele)[0]
    if '/' in inp:
        for v in inp_list:
            v = v.strip()
            if len(v) >0:
                if (v not in vars_) and (v not in raw_vars):
                    out3.write(v + "undefined usage")
            if v in fn_vars:
                    out3.write(v+" fin var used as input var")
    else:
        inp = inp.strip()
        if (inp not in vars_) and (inp not in raw_vars):
            out3.write(inp+ " undefined usage")
        if inp in fn_vars:
```

```
                    out3.write(inp+ "fin var used as input var")

    if bool(re.search(r"phy_name", line) ==True):
       pat = r'phy_name\s*=\s*\"(.*?)\"'
       key = re.findall(pat.line)[0]
       res = re.sub(r'phy_name\s*=\s*\"(.*?)\"', "phy_name="+value,line)
       out1.writelines(res+"\n")

       elif len(re.findall("concat", line))>0:
           inp = re.findall(r'inputs\s*=\s*\"(.*?)\"',line[0])

           if '/' in inp:
               inp_list = inp.split('/')
               value = ""
               for i, key in enumerate(inp_list):
                   key = key.strip()
                   if len(key) >0:
                       if key not in map_2_way_dict.keys():
                           out2.write("Key not found in mapping:"+key + "\n\n")
                           out1.writelines(line + '\n')
                           continue
                        if map_2_way_dict[key] == None:
                           out2.write("For "+ key + " NoneType value found in mapping\n\n")
                           continue

                        value_alt = '${{' + map_2_way_dict[key] + '}}'

                        if i = 0:
                           value = value_alt + '/'
                           res = line.replace(key,value,1)

                        else:
                          if i == len(inp_list)- 1:
                              value = value+value_alt
                          else:
                              value= value + value_alt + '/'
                              res = res.replace(key, value, 1)

               res = re.sub(r'inputs\s*=\s*\"(.*?)\"',"inputs="+value, line)

            else:
               inp = inp.strip()
               if inp not in map_2_way_dict.keys():
                   out2.write("Key not found in mapping:" + inp + "\n")
                   out1.writelines(line + '\n')
                   continue
               if map_2_way_dict[inp]==None:
                        out2.write("Key not found in mapping:" + inp + "\n")
                        out1.writelines(line + '\n')
                        continue
                    if map_2_way_dict[inp]==None:
                        out2.write("For " + inp + " Nonetype value found in mapping")
                        continue
                    value = '${{' + map_2_way_dict[inp] + '}}'
                    res = line.replace(inp, value,1)

                out1.writelines(res+'\n')


           elif len(re.findall("categorical", line))>0:
               pat = 'transformatio\s*=\s*\"(.*?)\"'
               key = re.findall(pat,line)[0]
               value = str(key)
               pat1 = r'\'(.+?)\''
               l = re.findall(pat1, value)
               for c, ele in enumerate(l):
                   value = value.replace(ele, "placeholder{}".format(c), l)
               for c, ele in enumerate(l):
                   value = value.replace("placeholder{}".format(c),l)
               res = line.replace(value)
               out1.writelines(res+'\n')

           elif len(re.findall("set_as_missing", line)) >0:
               pat = 'var_name\s*=\s*\"(.*?)\"'
               var_ = re.findall(pat, line)[0]
               set_as_miss_fn.append(var_)
               out1.writelines(line+'\n')

           else:
               out1.writelines(line+'\n')

fin_miss = [x for x in final_vars if x not in fn_vars]
if len(fin_miss) > 0:
    out3.write("xxxxxxxxxxx Not present in current list of vars")
    out3.write("\n\n".join(fin_miss))

out4.write("\n\n".join(set_as_miss_fn))
out1.close()
out2.close()
out3.close()
out4.close()
```

Figure[8]: This is the automation script developed to check the presence of fn_vars, raw_vars, vars_, set_as_miss_fn variables. This script returns cleansed version of code which is further referenced to

be pushed to production.
13) Running model build and model test scripts: We run GBDT (gradient boosting decision tree model) to get the model probability scores assigned to every transaction.
14) Gini calculation/capture table calculation/monitoring model performance using performance stability index or jenson-shannon divergence metrics are all part of the last step performance checks and quality checks as a part of data governance.
15) We filter the data according to time limit (start_ts and end_ts) for current/live and history based time fields and pass this as a parameter to strategy performance and strategy threshold creation. (This step is not in conjugation of data creation for modeling but helps in creating rules based strategy performance).

Steps for modeling process (Modules)	**Checkpoints**	**Implementation notes**
Production datapull	removing duplicates using windowing functions based on latest transaction id and latest transaction timestamp for that particular id, there are inbuilt pyspark functions like dropDuplicates but they are not correct representation of dropping duplicate records	1)To drop duplicates- removing duplicates before stacking data for all clients Exception: for larger clients before we pass the data into any of the aggregation operations like filter(sampling data - xc_fin == 0), join(base+velocity var), union(stacking)
		2)time zone issue: for postgrespull (converting execution_ts_utc to America/Phoenix date-time format) 3)for corep pull (converting import_ts to America/Chicago date-time format) - need confirmation implementation till this date: we are not using America/Phoenix for postgrespull, and for corep_pull we are using utc.localize (using pytz python library) which helps in comparing datetimes against input string values side_note - for global model datapulls, we have run corep pulls from jupyterhub using utc.localize and for postgres from hub too but haven't used America/Chicago or America/Phoenix settings
QC reports/checks: numerical QC/categorical QC	1)filter out -999999 (imputation) values if required, 2)multiple star referencing values in select (eg: *client_vars, *outputs didn't work on terminal, it works fine on hub so we can pass *client_vars, ["N", "NMISS", …] or vice versa)	separate function for ignoring imputation values

Development datapull	1)removing duplicates using windowing functions 2)for velocity var creation, pull extra days depending on the requirement of velocity computation - remove id pipe from transaction_id using substring (rather than doing it in the steps after velocity+base var join)	time zone issue: for postgrespull (converting execution_ts_utc to America/Phoenix date-time format) for corep pull (converting import_ts to America/Chicago date-time format) - need confirmation implementation till this date: we are not using America/Phoenix for postgrespull, and for corep_pull we are using utc.localize (using pytz python library) which helps in comparing datetimes against input string values
Initial qc, numeric qc, categorical qc	filter out -999999 (imputation) values if required	need to include initial qc functions to accertifyutils and then use as modularised part
Global model exclusion fraud tags	cross-checking conditions wrt to tagging values provided by modeling team	we have included all codes in a single module
sampling of data	sampling rates according to values provided by modeling team, whether to ensure to use double the sampling rate or same sampling rate **major checkpoint- to ensure the downsampled goods value matches with the sampling_rate_goods is used as input**	for larger sized clients in case of global model (walmart, target, etsy) we had done sampling three times
base var creation	we are ensuring that we have downsampled data wrt to goods and 100% frauds **major checkpoint - generating fraud_rate_report to check the accuracy of data (wrt fraud_ind, month_year, client_name), taking counts()**	for walmart, target, etsy base var creation was done separately per month
velocity var creation	xc_hier, xc_test, null analysis for checking/correcting the skewness in the data	1)a single cc_numbr value had major chunk of values that added skewness to data in dev data for etsy, the same value got reiterated in val data for etsy 2) make similar checks for ipaddres 3) walmart had bill_eml had a lot of duplicate values
base var + velocity var merge	dropping duplicates after merge -used substring for removing id-pipe since postgres transaction ids don't have id pipe (this step could be done right after corep pull) -filtering out data from this merge if velocities are not present (for validation we	**join operation: left join with respect to non-duplicate dataframe**

	removed jan 1 data and for dev we removed sept 15 data)	
corep filtered/cleansed data	capping of velocity variables, month_year to month (to make changes in tagging code rather than including it at this step), customised variables for retail/amusement park industries, handling of null corep cols	
postgres cleansed data	capping of drv variables, drv variables are renamed, fc_del_meth renamed and setting fc_del_meth to imputation, fc_booking source based changes	
corep+postgres cleansed data merge	ensuring drop duplicates using windowing functions, union of postgres data for the intended months (sept-dec for dev data, jan-feb for val data) before we do corep+postgres merge	**join operation: left join with respect to non-duplicate dataframe**
client specific changes after merging final corep+postgres data	industry_name, fc_industry, fn_avg_amt, fc_sub_industry, fn_client_ind, fn_frd_rate, fn_cntry and any other client specific variables related changes are added here	
qc on stacked data, fraud rate report	to compute qc on global model header columns only to save computation time, groupby on client_name, fraud_ind, month_year	
modeling dataset prep (stacking dev data, val data wasn't stacked)	to confirm downsampling of goods per client from modeling team	

Table[3]: The table illustrated above explains steps as explained before this table but with more detailed notes to create datasets and implement Gini

1.3 Fraud_ind calculation and model probability score values calculated?

For fraud_ind to be 0 or 1 we need to check model_prob_score values. If they are closer to 1 then we get fraud_ind = 1 (fraud transaction) and if they are closer to 0 then we get fraud_ind = 0 (non-fraud transaction). Most of the fraud risk or credit risk modeling processes have machine learning modeling algorithms like GBDT, decision tree or others in place to score every transaction and generate model probability score accordingly.

How are weights assigned for every transaction?

Weights are assigned based on number of fraud and non-fraud transactions, if non-fraud transactions are comparatively in large number as compared to fraud transactions we would assign higher value to

them as compared to non-fraud transaction.
What is Lorenz curve and what is gini-coefficient?
Gini-coefficient = A/(A+B)
lorenz curve is the area between A and B regions as mentioned in the graph
what are the values we get on x-axis and what are the values we get on y-axis?

1.4 Idea/intuition behind Gini calculation (without transaction_id):

score_var	amt_va	fraud	weigh	weighte	weighted	cum	cumula	cumula	cumula	F.grea	F.lea	wiz	bucket_w	F.lag(cu	lag_ca	gini_mode	lag_wizard	gini_wizar	(gini_mod	gini_value	gini_value
0.99997	1000	1	50	49.999	50000	1	50	0.25	12.5	0	1	1	0.248756	0.25	0.25	0.062189	0.062189	0.132113	-0.43781	-0.36789	1.01533
0.99987	2000	1	50	49.994	100000	2	100	0.5	25	0	1	1	0.248756	0.25	0.25	0.093284	0.093284	0.135981	-0.40672	-0.36402	
0.99936	3000	1	50	49.968	150000	3	150	0.75	37.5	0	1	1	0.248756	0.5	0.5	0.155473	0.155473	0.143716	-0.34453	-0.35628	
0.83483	5000	1	50	41.742	250000	4	200	1	50	0	1	1	0.248756	0.75	0.75	0.217662	0.217662	0.15145	-0.28234	-0.34855	
0.34993	4000	0	1	0.3499	4000	4	201	1	50.25	1	0	0	0.004975	1	1	0.004975	0.004975	1.24E-05	-0.49502	-0.49999	

4	201
sum(	sum(weights)

Table [4]: Comprehensive calculation of fields leading to gini_new (being calculated on new model with new model probability values)

Interpretations of **Table [4]** as explained below:

score_var	model_prob_score_var
amount_var	transaction_amount
fraud_ind	fraud indicator flag (created using business logic), weights (these are the assigned weights for fraud + non_fraud transactions where we check transactions with more weights are essentially frauds and transactions with less weights are non_frauds,
weights	weights assigned to every transaction
weighted_score	weights*model_prob_score
weighted_amount	weights*amount
cumulative_frauds	cumulative value of fraud_ind column; cumulative_frds = we calculate cumulative_frds over a window function by arranging model prob values in descending order (arranging transactions with highest model probability values which are given the highest ranking whereas transactions with lower model probability values are arranged afterwards).
cumulative_weights	cumulative value of weights
cumulative_capture	cumulative frauds/total_frauds
wizard_line	F.greatest(F.least(cumulative_wgt/sum_tot_frds))
bucket_width	sum_wgt_new/sum_tot_wgt
lag_capture	coalesce(lag(cumulative_capture).over(w)
weighted_score	weights*model_prob_score
weighted_amount	weights*amount
gini_model	0.5*(cumulative_capture + lag_capture)*bucket_width
lag_wizard	coalesce(lag(wizard_line)).over(w)
gini_wizard	0.5*(wizard_line + lag_wizard)*bucket_width
num	(gini_model – 0.5)
denom	(gini_wizard – 0.5)
gini_value	num/demon

Table [5]: Briefly mentioned fields that are used in Gini metric calculation

Similarly we do it for model with old model probabilities, to ensure we get gini_old values too (as referenced in **Table [5]**).

fraud_ind	weight	model_pr	all_txns_	all_txns_	weights*	sum_weig	avg_scor	transacti	wgt*total	wgt_total	unwgt_to	frd_amou	wgts*frd	wgt_frd_	wgt_dol_	all_frd_tx	unweight	weighted	frd_amt	weighted	weighted	wgt_frd_	wgt_total_amout
1	50	0.999933	303	1	49.99665	284.1614	0.72102	1000	50000	1221000	45000	1000	24000	0.019802	1.62E-08	6	0.666667	0.019802	1000	50000	49.99665	50000	50000
1	50	0.999334		1	49.9667			2000	100000			2000							2000	100000	49.9667	100000	100000
1	50	0.993853		1	49.69265			3000	150000			3000							3000	150000	49.69265	150000	150000
1	50	0.945582		1	47.2791			4000	200000			4000							4000	200000	47.2791	200000	200000
1	50	0.833443		1	41.67217			5000	250000			5000							5000	250000	41.67217	250000	250000
0	1	0.000005		1	0.000005			6000	6000			0							0	6000	0.000005	0	6000
0	1	0.340006		1	0.340006			7000	7000			0							0	7000	0.340006	0	7000
0	1	0.482392		1	0.482392			8000	8000			0							0	8000	0.482392	0	8000
1	50	0.894634		1	44.7317			9000	450000			9000							9000	450000	44.7317	450000	450000

Table [6]: Intermediate fields that we refer to for calculating Gini metrics.

Interpretations of **Table [6]** as explained below:

fraud_ind	If the real time decisioning system refers to having transactions with model probability score being less than the transaction is marked as non-fraud otherwise fraud
weights	we assign weights to non-fraud + fraud transactions to maintain equanimity between them.
model probability scores	we assign model probability scores to them if the transactions are grouped as fraud or non_fraud
all_txns_weighted	sum of all weights of all transactions taken into consideration to build the model.
all_txns_unweighted	sum of all transactions (and not the weights considered against these transactions)
sum_weighted_score	sum(weights*model_probability_score)
avg_score	average model probability score assigned to all transactions
transaction_amount	amount assigned to all transactions
wgt_total_amount	wgt*total_amount (total_amount = transaction_amount)
unwgt_total_amount	sum(total_amount)
fraud_amount	fraud_ind * transaction_amount
wgt_fraud_rate	total_frauds/all_txns_weighted
wgt_dol_rate	wgt_frd_amount/wgt_total_Amount
unweighted_frd_rate	all_frd_txns/sum(all_txns_unweighted)
weighted_frd_rate	all_frd_txns/sum(all_txns_weighted)
frd_amount	frd_ind * total_amount
weighted_amount	weights*total_amount
weighted_score	weights*model_prob_score
wgt_frd_amount	weights*frd_ind*total_amount
wgt_total_amount	weights*total_amount
row	we create this column to rank order all transactions with respect to model_probability_score
part	we create this column to have (row*num_parts)/total_events(total_transactions) : this shall give the fraction of the part that transaction would like respective to the rank assigned
num_parts	1000 or 10000 depending on the total number of buckets needed to assign capture table metrics values.

part < num_parts	defining condition to assign part values according to num_parts, this shall ensure that we are defining all parts pertaining to every bucket and the transaction count for it
left join	we perform left join of every part created as a result of that fraction
part >= part2	filtering out all the required transactions/records
groupby = criteria + tot_events + total_frauds	

Table [7]: Description of intermediate fields in Gini calculation are explained in this table

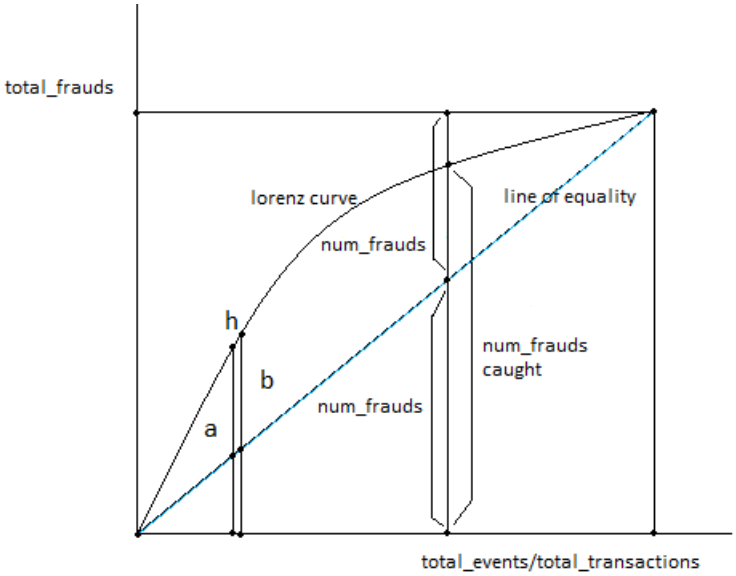


Figure [9] : Overall gini value calculation

On x-axis we have total_events/total_transactions coming from a merchant in real time transaction processing, on y-axis we have total_frauds coming from a merchant in real transaction processing the line drawn at 45% inclination is line of equality where if we plot a point on this line, it would be interpreted as total_frauds = total_events/total_transactions or num_frauds = num_frauds (above and below the line of equality)

Explanation with illustration:

If we consider a, b and h values in the trapezoid and h as the height:

a or b = num_frauds_caught – num_frauds

area of trapezoid = (0.5)*h*(a+b) = (0.5)*num_events*((2*num_frauds_caught) - num_frauds)

ideally here (a+b) = 2*(num_frauds_caught – num_frauds), but since a, b are almost similar and overlapping of each other we consider 2*num_frauds_caught

so, gini_A : we calculate area of trapezoid (0.5*(a+b)*h)

gini_B : area of triangle = (0.5)*b*h = (0.5)*total_events*total_frauds (area of the entire triangle)

gini_C: area of triangle = (0.5)*b*h = (0.5)*total_events*(total_events - total_frauds)

Here, B is area of the entire triangle (area of A + area of C)

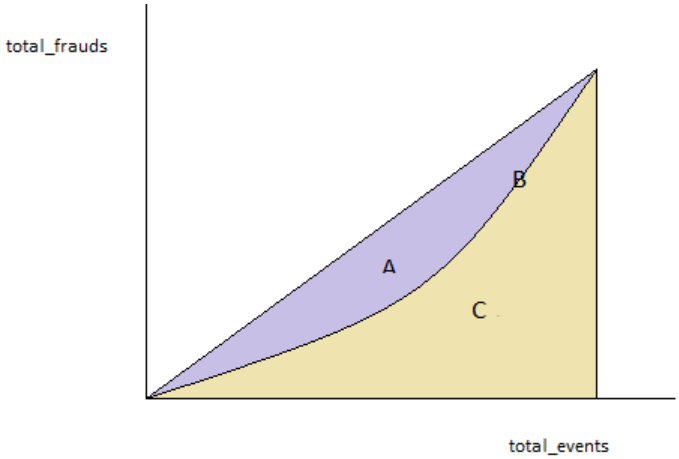


Figure [10]: violet shaded area is tagged as A, yellow shaded area is tagged as C, A+C = area covered under Lorenz curve + area outside of Lorenz curve – this figure is helpful to refer in case we want to understand the ratio of areas taken.
gini_new = A/(A+B) (as can be referred to **Figure[1]**: where B is area between Lorenz curve on x-axis, but in our calculations B is the area of the entire triangle)
so, as per our calculations : gini_new = (A-B)/C
Similarly, we follow the calculations for other models (having model probability values), like gini_old.

```
dfl = dfl.select([criteria[0],score_var1, frd_ind, wgt, amt_var])
dfl = dfl.withColumn("weighted_score_new", dfl[wgt]*dfl[score_var1])
dfl = dfl.withColumn("weighted_amount_new", dfl[wgt]*dfl[amt_var])
wgt_total_amt = dfl.agg({'weighted_amount_new':'sum'}).collect()[0][0]
sum_wgt_score_new = dfl.agg({"weighted_score_new":"sum"}).collect()[0][0]
dfl = dfl.drop('weighted_score_new','weighted_amount_new')
dfl = dfl.toDF(*[criteria[0], 'score','frd_ind','wgt_new','amt_var_new'])
dfl = dfl.withColumn('score', dfl.score.cast(DoubleType()))
dfl = dfl.groupby(criteria[0], 'score').agg({'frd_ind':'sum','wgt_new':'sum', 'amt_var_new':'sum'})
dfl = dfl.select(criteria[0],'score', 'sum(frd_ind)', 'sum(wgt_new)','sum(amt_var_new)')
dfl = dfl.toDF(*[criteria[0], 'score','tot_frds','sum_wgt_new','sum_amt_var_new'])
windowval = (Window.orderBy(dfl.score.desc()).rangeBetween(Window.unboundedPreceding, 0))
dfl = dfl.withColumn("cumulative_frds", F.sum("tot_frds").over(windowval))
sum_tot_frds = dfl.agg({"tot_frds":"sum"}).collect()[0][0]
sum_tot_wgt = dfl.agg({"sum_wgt_new":"sum"}).collect()[0][0]

dfl = dfl.withColumn("cumulative_capture", dfl.cumulative_frds/sum_tot_frds)
dfl = dfl.withColumn("cumulative_wgt", F.sum("sum_wgt_new").over(windowval))
dfl = dfl.withColumn("wizard_line", F.least(F.greatest((dfl.cumulative_wgt/sum_tot_frds), F.lit(0)), F.lit(1)))
dfl = dfl.withColumn('bucket_width', dfl['sum_wgt_new']/sum_tot_wgt)
w = Window().orderBy(dfl.score.desc())
dfl = dfl.withColumn("lag_capture", F.coalesce(F.lag("cumulative_capture").over(w), F.lit(0)))
dfl = dfl.withColumn("gini_model", F.lit(0.5)*(dfl.cumulative_capture+dfl.lag_capture)*dfl.bucket_width)
dfl = dfl.withColumn("lag_wizard", F.coalesce(F.lag("wizard_line").over(w), F.lit(0)))
dfl = dfl.withColumn("gini_wizard", F.lit(0.5)*(dfl.wizard_line+dfl.lag_wizard)*dfl.bucket_width)
num = (dfl.agg({"gini_model":"sum"}).collect()[0][0] - F.lit(0.5))
denom = (dfl.agg({'gini_wizard':'sum'}).collect()[0][0]-F.lit(0.5))
dfl = dfl.withColumn("gini_new", num/denom)

all_txns_unweighted = dfl.count()
dfl = dfl.withColumn("all_txns_unweighted", F.lit(all_txns_unweighted))
dfl = dfl.withColumn("all_txns_weighted", F.col('sum_wgt_new'))
dfl = dfl.withColumn("sum_weighted_score_new", F.lit(sum_wgt_score_new))
dfl = dfl.withColumn("avg_scr_new", dfl['sum_weighted_score_new']/dfl['all_txns_weighted'])
dfl = dfl.withColumn('wgt_total_amt', F.lit(wgt_total_amt))
dfl = dfl.withColumn('unwgt_total_amt', F.col('sum_amt_var_new'))

dfl_ = input_df
dfl_ = dfl_.withColumn("weighted_score_new", dfl_[wgt]*dfl_[score_var1])
dfl_ = dfl_.withColumn("weighted_amount_new", dfl_[wgt]*dfl_[amt_var])
wgt_total_amt = dfl_.agg({'weighted_amount_new':'sum'}).collect()[0][0]
sum_wgt_score_new = dfl_.agg({"weighted_score_new":"sum"}).collect()[0][0]
sum_amt_var_new = dfl_.agg({amt_var:"sum"}).collect()[0][0]
dfl_ = dfl_.withColumn("unwgt_total_amt", F.lit(sum_amt_var_new))
wgt_frd_amt = dfl_.filter(dfl_.fraud_ind==F.lit(1)).agg({amt_var:"sum"}).collect()[0][0]
dfl_ = dfl_.withColumn("wgt_frd_amt", F.lit(wgt_frd_amt))
dfl_ = dfl_.withColumn("wgt_total_amt", F.lit(wgt_total_amt))
wgt_dol_rate = dfl_["wgt_frd_amt"]/dfl_["wgt_total_amt"]
dfl_ = dfl_.withColumn("wgt_dol_rate", F.lit(wgt_dol_rate))
unwgt_dol_rate = dfl_["wgt_frd_amt"]/dfl_["unwgt_total_amt"]
dfl_ = dfl_.withColumn("unwgt_dol_rate", F.lit(unwgt_dol_rate))
all_frd_txns = dfl_.agg({'fraud_ind':'sum'}).collect()[0][0]
dfl_ = dfl_.withColumn('all_frd_txns', F.lit(all_frd_txns))
dfl_ = dfl_.drop("unwgt_total_amt","wgt_total_amt")
df_join = dfl.join(dfl_,criteria[0], "inner")
df_join = df_join.withColumn('unweighted_frd_rate', F.col('all_frd_txns')/F.col('all_txns_unweighted'))
df_join = df_join.withColumn('weighted_frd_rate', F.col('all_frd_txns')/F.col('all_txns_weighted'))

df_join.repartition(20).write.mode("overwrite").parquet(outpath+ "new_model_prob/new_model_prob_values/")
```

Figure[11]: Gini metrics calculation of model with old or new model probabilities where we don't have values for transaction id

```
columns = criteria + [score_var1, amt_var, frd_ind, wgt]
new_columns = criteria + ["score", "amt", "frd_ind","wgt_new"]
df1 = df1.select(columns).toDF(*new_columns)

df1 = df1.withColumn("frd_amt", df1.frd_ind * df1.amt)
df1 = df1.withColumn("weighted_amount", df1.wgt_new * df1.amt) #weighted_Amount
df1 = df1.withColumn("weighted_score_new", df1.wgt_new * df1.score)

df1 = df1.withColumn("row", row_number().over(Window.partitionBy(criteria).orderBy(col("score").desc())))

df1_new = df1.groupby(criteria).agg(count("frd_ind").alias("tot_evts"),\
                                    sum("frd_ind").cast("int").alias("tot_frds"),\
                                    sum("amt").cast("float").alias("tot_amt"),\
                                    sum("frd_amt").cast("float").alias("wgt_frd_amt"),\
                                    sum("wgt_new").alias("all_txns_weighted"),\
                                    sum("weighted_amount").cast("float").alias("wgt_total_amt"),\
                                    sum("weighted_score_new").cast("float").alias("sum_weighted_score_new"))

df1_new = df1_new.withColumn("fraud_rate", F.col("tot_frds")/F.col("tot_evts"))
df1_new = df1_new.withColumn("weighted_frd_rate", F.col("tot_frds")/F.col("all_txns_weighted"))
df1_new = df1_new.withColumn("wgt_dol_rate", F.col("wgt_frd_amt")/F.col("wgt_total_amt"))
df1_new = df1_new.withColumn("avg_scr_new", F.col("sum_weighted_score_new")/F.col("all_txns_weighted"))

df1_new.repartition(20).write.mode("overwrite").parquet(outpath + "new_model_prob/df1_new/")
df1.repartition(20).write.mode("overwrite").parquet(outpath + "new_model_prob/df1n/")

df1 = df1.join(df1_new, criteria, "left")
df1 = df1.withColumn("part", ((df1["row"]*num_parts) / df1["tot_evts"]).cast("int")) #0*1000/397,250

df1 = df1.filter(df1["part"] < num_parts)

df1 = df1.groupby(criteria + ["part","tot_evts","tot_frds","tot_amt","wgt_frd_amt"]).agg(\
                  count("frd_ind").alias("num_evts"),\
                  sum("frd_ind").cast("int").alias("num_frds"),\
                  sum("amt").cast("float").alias("dlr_amt"),\
                  sum("frd_amt").cast("float").alias("frd_amt"),\
                  min("score").alias("min_score"),\
                  min("score").alias("max_score"),\
                  avg("score").alias("avg_score"))

df2 = df1.select(criteria + ["part","num_evts","num_frds","dlr_amt","frd_amt"])

new_columns = criteria + ["part2","num_evts_2","num_frds_2","dlr_amt_2","frd_amt_2"]

df2 = df2.toDF(*new_columns)

df1 = df1.join(df2, criteria, "left")

df1 = df1.filter(df1["part"] >= df1["part2"])

df1 = df1.groupby(criteria + ["part", "tot_evts", "tot_frds", "tot_amt", "wgt_frd_amt", "num_evts", "num_frds","dlr_amt", "frd_amt", "min_score", "max_score", "avg_score"]).agg(\
                sum("num_evts_2").cast("int").alias("num_evts_caught"),\
                sum("num_frds_2").cast("int").alias("num_frds_caught"),\
                sum("dlr_amt_2").cast("float").alias("dlr_amt_caught"),\
                sum("frd_amt_2").cast("float").alias("frd_amt_caught"))

df1 = df1.withColumn("gini_A", 0.5 * df1["num_evts"] * ((2 * df1["num_frds_caught"]) - df1["num_frds"]))
df_gini_newprob = df1.groupby(criteria + ["tot_evts", "tot_frds"]).agg(sum("gini_A").alias("gini_A")).orderBy(criteria)
df_gini_newprob = df_gini_newprob.withColumn("gini_B", 0.5 * df_gini_newprob["tot_frds"] * df_gini_newprob["tot_evts"])
df_gini_newprob = df_gini_newprob.withColumn("gini_C", 0.5 * df_gini_newprob["tot_frds"] * (df_gini_newprob["tot_evts"] - df_gini_newprob["tot_frds"]))
df_gini_newprob = df_gini_newprob.withColumn("gini_new", round((df_gini_newprob["gini_A"] - df_gini_newprob["gini_B"]) * 100.0 / df_gini_newprob["gini_C"], 2))
df_gini_newprob.repartition(20).write.mode("overwrite").parquet(outpath + "/new_model_prob/new_model_prob_values/")
```

Figure[12]: Gini metrics calculation of model with old or new probabilities where we have values for transaction id

1.5 Accuracy values for old model prob + accuracy values for new model prob:

Accuracy is defined as correct predictions out of total predictions in machine learning use cases. But accuracy(for fraud risk model) = ((average_score for model consisting of model probabilities of old model or new model/model scores) – (fraud rate of model consisting of model probabilities of old model or new model))/(fraud rate of model consisting of model probabilities of old model or new model). As has been observed in the **Figure[12]** below Accuracy value can shoot up till 108% (more than 100%), so it renders to be a measure/metric which we don't tend to resort immediately. Also accuracy is suitable for machine learning use-cases where dataset is balanced (total positives > total negatives or total negatives < total positives) where the cost function of false positives is equal to cost function of false negatives. In ideal scenario/world, no model would consist of balanced dataset. Other definition is (TP+TN)/(TP+TN+FP+FN) = true_values/all_values.

model_pr	wgt	frd_ind	wgt_score	cumulativ	bucket_si	bkt_values	val	bkt_va	sum(wgt_	sum(fraud	total_txns	avg_score	fraud_rate	abs_diff	accuracy_deviation
0.999934	1	1	0.999934	1	1000	cumulative_wgt <= val1 then 1	10.8	1	10	4	108	0.092593	0.037037	0.055556	1.5
0.934342	1	1	0.934342	2		cumulative_wgt1 > {val1} and cumulative_wgt1 <= {val2} then 2	21.6	1	10	4					150
0.986323	1	1	0.986323	3		cumulative_wgt1 > {val2} and cumulative_wgt1 <= {val3} then 3	32.4	1	10	4					
0.933232	1	1	0.933232	4		cumulative_wgt1 > {val3} and cumulative_wgt1 <= {val4} then 4	43.2	1	10	4					
0.343332	50	0	17.1666	54		cumulative_wgt1 > {val4} and cumulative_wgt1 <= {val5} then 5	54	5	109	1					
0.672323	1	1	0.672323	55		cumulative_wgt1 > {val5} and cumulative_wgt1 <= {val6} then 6	64.8	5	109	1					
0.232312	50	0	11.6156	105		cumulative_wgt1 > {val6} and cumulative_wgt1 <= {val7} then 7	75.6	9	426	3					
0.822433	1	1	0.822433	106		cumulative_wgt1 > {val7} and cumulative_wgt1 <= {val8} then 8	86.4	9	426	3					
0.999323	1	1	0.999323	107		cumulative_wgt1 > {val8} and cumulative_wgt1 <= {val9} then 9	97.2	9	426	3					
0.734342	1	1	0.734342	108		cumulative_wgt1 > {val9} then 10	108	9	426	3					
		n_rows = max(cumulative_wgt1) = 108													
		bucket_size = 108/10		10.8											

Table [8]: In order to calculate accuracy values we take into consideration following fields (as mentioned in the **Figure[12]** above)

Interpretations of **Table [8]** as explained below:

model_prob_var1	model_probability values for old model or new model
wgt	weights assigned to every transaction depending on fraud_ind = 0 or fraud_ind = 1
frd_ind = 1	for all frauds
frd_ind = 0	for all goods
wgt_scr_new	weights_score assigned to every transaction depending on new model probability
cumulative_wgt1	derived field to calculate cumulative values for old or new model depending on model probability values
nrows	max(cumulative_wgt1)
bucket_size	nrows/10
condition for bucket_values is defined as:	
bkt_values	case when cumulative_wgt1 <= {val1} then 1 when cumulative_wgt1 > {val1} and cumulative_wgt1 <= {val2} then 2 when cumulative_wgt1 > {val2} and cumulative_wgt1 <= {val3} then 3 when cumulative_wgt1 > {val3} and cumulative_wgt1 <= {val4} then 4 WHEN cumulative_wgt1 > {val4} and cumulative_wgt1 <= {val5} then 5 WHEN cumulative_wgt1 > {val5} and cumulative_wgt1 <= {val6} then 6 WHEN cumulative_wgt1 > {val6} and cumulative_wgt1 <= {val7} then 7 WHEN cumulative_wgt1 > {val7} and cumulative_wgt1 <= {val8} then 8 WHEN cumulative_wgt1 > {val8} and cumulative_wgt1 <= {val9} then 9 WHEN cumulative_wgt1 > {val9} then 10 end
val (as mentioned in above query as val1, val2, val3 and so on) are computed as:	
val1	bucket_size * 1
val2	bucket_size * 2
val3	bucket_size * 3
val4	bucket_size * 4
val5	bucket_size * 5
val6	bucket_size * 6
val7	bucket_size * 7
val8	bucket_size * 8
val9	bucket_size * 9
bkt_values_assigned_as_per_condition	we groupby as per cumulative_wgts to get the bkt_values
sum(wgt_score_new)	we groupby as per wgt_score_new to get these values
sum(fraud_ind)	we groupby as per fraud_ind to get these values
total_txns	total number of transactions (unweighted)
avg_score	sum(wgt_score)/total_txns
fraud_rate	sum(fraud_ind)/total_txns
abs_diff	(avg_score – fraud_rate)/fraud_rate
accuracy_deviation	(abs_diff/avg_fraud_rate)*100

	as can be seen from the table, we get accuracy_deviation values as 1.5 or 150% (values for accuracy can shoot beyond 1 or 100% and don't contribute much to some scenarios in fraud risk world for determining if the transactions are being correctly tagged as fraud or non_fraud or whether they are correctly getting populated or no), but for some other use-cases/scenarios: this metric becomes an essential part to understand.

Table[9]: Detailed explanation of fields that are mentioned in **Table [8]**

1.6 Critical bucket calculation values:

Why is critical bucket calculation needed and what is the significance?

We calculate gini lift/fraud dollar lift/fraud transactions lift/total transactions lift values at critical bucket as well as at a standard rate of bucket value.

What is the deciding factor for critical bucket value evaluation and calculation?

The deciding factor for critical bucket value evaluation and calculation has been mentioned in the below table. The **Figure[13]** after this table is the main code for calculating critical bucket value.

df	input data on which capture table metrics are supposed to be calculated
model_prob_var1	input model probability score/values which are arranged in descending order
columns	criteria + [score_var, frd_ind] where criteria is month, client_name, transaction_id, industry
new_columns	criteria + [score, frd_ind]
score	model probability score value
df.groupby(criteria + [score].agg(count(frd_ind).alias(num_events), sum(frd_ind).alias(num_frds))	Groupby condition
columns1	criteria + [cutoff_score, num_events, num_frds]
columns2	criteria + [model_score, num_events2, num_frds2]
cutoff_score < model_score	This is to secure all values with respect to model probability score
events caught	number of events/transactions caught out of all the transactions
frauds caught	number of frauds caught out of all transactions
total events	total value of transactions count
total frauds	total value of fraud transactions
bucket fraud rate	number_of_frauds/num_events
events caught >= total frauds	conditional statement to deduce the critical bucket calculation
row	number of transactions/number of fraud transactions/number of transactions caught/number of fraud transactions caught
df_crit == 1	filtering out all the transactions where critical bucket transaction criteria is met
events_caught = transactions_caught_old	for old model, events_caught/transactions_caught > we would have the similar criteria for new model with new model probability values

cutoff_score = model_prob_cutoff_score_old	cutoff_score for old model with old model probability score values > we would have the similar criteria for old model with old model probability values

Table[10]: Detailed explanation of fields considered and calculated for critical bucket

```
df = input_df
score_var = model_prob_var1
columns = criteria + [score_var, frd_ind]
new_columns = criteria + ["score", "frd_ind"]
df = df.select(columns).toDF(*new_columns)
df = df.withColumn("score", round(df.score * 100, 0))
df_1 = df.groupby(criteria + ["score"]).agg(count("frd_ind").alias("num_evts"),sum("frd_ind").alias("num_frds"))
columns1 = criteria + ["Cutoff_Score", "Num_Evts", "Num_Frds"]
columns2 = criteria + ["model_score", "num_evts2", "num_frds2"]
df_1 = df_1.toDF(*columns1)
df_2 = df_1.toDF(*columns2)
df_3 = df_1.join(df_2, criteria, "left")
df_3 = df_3.filter(df_3.Cutoff_Score <= df_3.model_score)
df_4 = df_3.groupby(columns1).agg(sum("num_evts2").alias("Evts_Caught"),sum("num_frds2").alias("Frds_Caught"))
df_5 = df_1.groupby(criteria).agg(sum("Num_Evts").alias("Tot_Evts"),sum("Num_Frds").alias("Tot_Frds"))
fin_df = df_4.join(df_5, criteria, "left")
fin_df = fin_df.orderBy(criteria + ["Cutoff_Score"])
fin_df = fin_df.withColumn("Bckt_Frd_Rate", (fin_df.Num_Frds / fin_df.Num_Evts))
df_crit = fin_df.filter(fin_df.Evts_Caught >= fin_df.Tot_Frds)
df_crit = df_crit.withColumn("row", row_number().over(Window.partitionBy(criteria).orderBy(col("Evts_Caught"))))
df_crit = df_crit.filter(df_crit.row == 1).drop("row").orderBy(criteria)
df_crit = df_crit.select("Evts_Caught","Cutoff_Score",*criteria)
df_crit = df_crit.withColumnRenamed("Evts_Caught", "Txns_Caught_new")
df_crit = df_crit.withColumnRenamed("Cutoff_Score", "model_prob_cutoff_Score_new")
df_crit.coalesce(1).write.mode("overwrite").option("header",True).format("csv").save(outpath +"ckt_calc_new/")
df_join1 = spark.read.parquet(outpath + "/output/")
df_final = df_join1.join(df_crit, criteria, "inner")
```

Figure[13]: Critical bucket calculation code.

1.7 Capture table metrics calculation:

What is capture table metrics? How are we calculating capture table metrics? What is the relevance of capture table metrics in today's day and age?

We have Gini coefficient calculation/computation where we map cumulative share of population on x-axis and cumulative share of income earned on y-axis giving us a picture of Gini coefficient metrics calculation. For risk prevention in case of fraud risk or credit risk, we tend to have number of total transactions reviewed or received in real-time data processing plotted on x-axis and number of frauds captured or reviewed in real-time data processing on y-axis. Total area without considering cumulative values goes in the Gini coefficient calculation whereas for capture table metrics we need to consider cumulative values one by one. As it is, we have 10% transactions, 20% transactions and so on and so forth in increasing order, the total number of transactions going more and more towards an upward hill, and finally reaching 100%. Fields that are supposed to be defined to calculate capture table metrics:

rank_bps (rank ordering the buckets according to decreasing model probability values):	as mentioned earlier, the model probability score with highest values lies on the top, whereas model probability with lowest values lies on the bottom. In this manner, we are able to determine all the metrics related to frauds caught, fraud dollars saved from the initial bucket values
score_var1	model probability value for old model
score_var2	model probability value for new model
amt_var	transaction amount variable
frd_ind	fraud indicator
wgt_amt	weighted amount
frd_amt	fraud amount
wgt_score_old	weighted score for old model

wgt_score_new	weighted score for new model
total_buckets	number of parts (the value generally varies from 1 to 1000 or 10 to 10000)
bucket_size	total transactions that could be contained within one bucket = 1000/total_buckets
max_bucket	print_buckets*bucket_size
bps_list	similar to total buckets
all_txns_weighted	sum of all weighted transactions
all_txns_unweighted	sum of all transactions/total transaction count
cumulative_wgt1	cumulative values of weights assigned for every transaction/as a rule of thumb we assigned higher weight for fraud transaction and lower weight for non-fraud/good transaction which would ensure that we have equal representation of frauds vs non-frauds
cumulative_frd	cumulative values of frauds
cumulative_wgt_amt	cumulative values of weighted amount
cumulative_frd_amt	cumulative values of fraud amount
cumulative_wgt_score	cumulative values of weighted scores
bps_to_index	the total count of all parameters contained within that bucket, this index is calculated as such; first bucket = 0.001*all_txns_weighted, second bucket = 2*0.001*all_txns_weighted and so on and so forth
all_frd_txns	total count of fraud transactions
indexes for model with old and new model probabilities/scores	similar to bps_to_index values, we would have first bucket = 0.001*weighted_score, second bucket = 0.001*2*weighted_score
cTxn_new	total transaction count for new model = cumulative weighted transaction value
cFTxn_new	total fraud transaction count for new model = cumulative weighted fraud transaction value
cTxn_new	total transaction count for new model
cFTxn_new	total fraud transaction count for new model
fTCov_new	fraud transaction coverage for new model = cFTxn_new/all_frd_txns
cDol_new	count of dollar amount for new model
cFDol_new	count of fraud dollar amount for new model
dolRate_new	dollar amount rate for new model = cFDol_new/cDol_new
actual_new	ratio of fraud transactions against total transactions for new model=cFTxn_new/cTxn_new
score_new_sum	index value at that cumulative_wgt_score for new model with new model probability score values
pred_new	predicted value for new model=score_new_sum/cTxn_new
cTxn_old	count of transactions for old model
cFTxn_old	count of fraud transactions for new model
fTCov_old	fraud transaction coverage for old model = CFTxn_old/all_frd_txns
cDol_old	count of dollar amount for old model

cFDol_old	count of fraud dollar amount for old model
dolRate_old	dollar amount rate for old model = CFDol_old/cDol_old
actual_old	ratio of fraud transactions against total transactions for old model = CFTxn_old/cTxn_old
score_old_sum	index value at that cumulative_wgt_score for old model
pred_old	predicted value for old model = score_old_sum/cTxn_old
min_pred_old	minimum value of model probability score values at that particular bucket for old model with old model probability score values
min_pred_new	minimum value of model probability score values at that particular bucket for old model with old model probability score values
max_pred_new	minimum value of model probability score values at that particular bucket for new model with old model probability score values
max_pred_old	minimum value of model probability score values at that particular bucket for new model with old model probability score values
fTCovLift	fraud dollar transactions coverage lift value which checks how has the new model performed as compared to old model = (fTCov_new-fTCov_old)/fTCov_old
cDolLift	total dollar amount lift = (cDol_new-cDol_old)/cDol_old
cFDolLift	fraud dollar amount lift = (cFDol_new-cFDol_old)/cFDol_old
dolRateLift	fraud dollar rate lift = (dolRate_new-dolRate_old)/dolRate_old

Table[11]: Detailed explanation of capture table metrics

Table[11] fields look like the below **figure[14]**:

We consider reviewing and computing following fields in capture table metrics calculation:

	all_frd_txn	2502500		cumulative_wgt_amount				cumulative_wgt_score				cumulative	cumulative_frd_amt			cumulative_wgt_score						(fTCov_ne	(cDol_new	(cFDol_ne	(dolRate_ne
rank_bp	cTxn_ne	cFTxn_n	fTCov_new	cDol_ne	cFDol_ne	dolRate_	actual_n	score_ne	pred_ne	cTxn_ol	cFTxn_o	fTCov_old	cDol_old	cFDol_ol	dolRate_	actual_o	score_ol	pred_old	min_pre	min_pre	max_pre	max_pre	fTCovLif	cDolLift	cFDolLif	dolRateLif
1	1000	5	1.998E-06	5000	250	0.05	0.005	2500	2.5	1100	6	2.3976E-06	6000	300	0.05	0.005455	3000	2.727273	0.892323	0.872322	0.999323	0.998233	-0.16667	-0.16667	-0.16667	0
2	2000	10	3.996E-06	10000	500	0.05	0.005	3000	1.5	2100	12	4.7952E-06	12000	600	0.05	0.005714	3500	1.666667	0.898323	0.868322	0.995323	0.994233	-0.16667	-0.16667	-0.16667	0
3	3000	15	5.9940TE-06	15000	750	0.05	0.005	3500	1.166667	3100	18	7.1928E-06	18000	900	0.05	0.005806	4000	1.290323	0.884323	0.864322	0.991323	0.990233	-0.16667	-0.16667	-0.16667	0
4	4000	20	7.9920TE-06	20000	1000	0.05	0.005	4000	1	4100	24	9.5904E-06	24000	1200	0.05	0.005854	4500	1.097561	0.880323	0.860322	0.987323	0.986233	-0.16667	-0.16667	-0.16667	0
5	5000	25	9.9900TE-06	25000	1250	0.05	0.005	4500	0.9	5100	30	1.1988E-05	30000	1500	0.05	0.005882	5000	0.980392	0.876323	0.856322	0.983323	0.982233	-0.16667	-0.16667	-0.16667	0
6	6000	30	1.1988E-05	30000	1500	0.05	0.005	5000	0.833333	6100	36	1.4386E-05	36000	1800	0.05	0.005902	5500	0.901639	0.872323	0.852322	0.979323	0.978233	-0.16667	-0.16667	-0.16667	0
7	7000	35	1.3986E-05	35000	1750	0.05	0.005	5500	0.785714	7100	42	1.6783E-05	42000	2100	0.05	0.005915	6000	0.84507	0.868323	0.848322	0.975323	0.974233	-0.16667	-0.16667	-0.16667	0
8	8000	40	1.5984E-05	40000	2000	0.05	0.005	6000	0.75	8100	48	1.9181E-05	48000	2400	0.05	0.005926	6500	0.802469	0.864323	0.844322	0.971323	0.970233	-0.16667	-0.16667	-0.16667	0
9	9000	45	1.7982E-05	45000	2250	0.05	0.005	6500	0.722222	9100	54	2.1578E-05	54000	2700	0.05	0.005934	7000	0.769231	0.860323	0.840322	0.967323	0.966233	-0.16667	-0.16667	-0.16667	0
10	10000	50	1.998E-05	50000	2500	0.05	0.005	7000	0.7	10100	60	2.3976E-05	60000	3000	0.05	0.005941	7500	0.742574	0.856323	0.836322	0.963323	0.962233	-0.16667	-0.16667	-0.16667	0
11	11000	55	2.1978E-05	55000	2750	0.05	0.005	7500	0.681818	11100	66	2.6374E-05	66000	3300	0.05	0.005946	8000	0.720721	0.852323	0.832322	0.959323	0.958233	-0.16667	-0.16667	-0.16667	0
12	12000	60	2.3976E-05	60000	3000	0.05	0.005	8000	0.666667	12100	72	2.8771E-05	72000	3600	0.05	0.00595	8500	0.702479	0.848323	0.828322	0.955323	0.954233	-0.16667	-0.16667	-0.16667	0
13	13000	65	2.5974E-05	65000	3250	0.05	0.005	8500	0.653846	13100	78	3.1769E-05	78000	3900	0.05	0.005954	9000	0.687023	0.844323	0.824322	0.951323	0.950233	-0.16667	-0.16667	-0.16667	0
14	14000	70	2.7972E-05	70000	3500	0.05	0.005	9000	0.642857	14100	84	3.3566E-05	84000	4200	0.05	0.005957	9500	0.673759	0.840323	0.820322	0.947323	0.946233	-0.16667	-0.16667	-0.16667	0
15	15000	75	2.997E-05	75000	3750	0.05	0.005	9500	0.633333	15100	90	3.5964E-05	90000	4500	0.05	0.00596	10000	0.662252	0.836323	0.816322	0.943323	0.942233	-0.16667	-0.16667	-0.16667	0
16	16000	80	3.1968E-05	80000	4000	0.05	0.005	10000	0.625	16100	96	3.8362E-05	96000	4800	0.05	0.005963	10500	0.652174	0.832323	0.812322	0.939323	0.938233	-0.16667	-0.16667	-0.16667	0
17	17000	85	3.3966E-05	85000	4250	0.05	0.005	10500	0.617647	17100	102	4.0759E-05	102000	5100	0.05	0.005965	11000	0.643275	0.828323	0.808322	0.935323	0.934233	-0.16667	-0.16667	-0.16667	0
18	18000	90	3.5964E-05	90000	4500	0.05	0.005	11000	0.611111	18100	108	4.3157E-05	108000	5400	0.05	0.005967	11500	0.635359	0.824323	0.804322	0.931323	0.930233	-0.16667	-0.16667	-0.16667	0
19	19000	95	3.7962E-05	95000	4750	0.05	0.005	11500	0.605263	19100	114	4.5554E-05	114000	5700	0.05	0.005969	12000	0.628272	0.820323	0.800322	0.927323	0.926233	-0.16667	-0.16667	-0.16667	0
20	20000	100	3.996E-05	100000	5000	0.05	0.005	12000	0.6	20100	120	4.7952E-05	120000	6000	0.05	0.00597	12500	0.621891	0.816323	0.796322	0.923323	0.922233	-0.16667	-0.16667	-0.16667	0
21	21000	105	4.1958E-05	105000	5250	0.05	0.005	12500	0.595238	21100	126	5.035E-05	126000	6300	0.05	0.005972	13000	0.616114	0.812323	0.792322	0.919323	0.918233	-0.16667	-0.16667	-0.16667	0
22	22000	110	4.3956E-05	110000	5500	0.05	0.005	13000	0.590909	22100	132	5.2747E-05	132000	6600	0.05	0.005973	13500	0.61086	0.808323	0.788322	0.915323	0.914233	-0.16667	-0.16667	-0.16667	0
23	23000	115	4.5954E-05	115000	5750	0.05	0.005	13500	0.586957	23100	138	5.5145E-05	138000	6900	0.05	0.005974	14000	0.606061	0.804323	0.784322	0.911323	0.910233	-0.16667	-0.16667	-0.16667	0
24	24000	120	4.7952E-05	120000	6000	0.05	0.005	14000	0.583333	24100	144	5.7542E-05	144000	7200	0.05	0.005975	14500	0.60166	0.800323	0.780322	0.907323	0.906233	-0.16667	-0.16667	-0.16667	0
25	25000	125	4.995E-05	125000	6250	0.05	0.005	14500	0.58	25100	150	5.994E-05	150000	7500	0.05	0.005976	15000	0.59761	0.796323	0.776322	0.903323	0.902233	-0.16667	-0.16667	-0.16667	0
26	26000	130	5.1948TE-05	130000	6500	0.05	0.005	15000	0.576923	26100	156	6.2338E-05	156000	7800	0.05	0.005977	15500	0.59387	0.792323	0.772322	0.899323	0.898233	-0.16667	-0.16667	-0.16667	0
27	27000	135	5.3946TE-05	135000	6750	0.05	0.005	15500	0.574074	27100	162	6.4735E-05	162000	8100	0.05	0.005978	16000	0.590406	0.788323	0.768322	0.895323	0.894233	-0.16667	-0.16667	-0.16667	0
28	28000	140	5.5944TE-05	140000	7000	0.05	0.005	16000	0.571429	28100	168	6.7133E-05	168000	8400	0.05	0.005979	16500	0.587189	0.784323	0.764322	0.891323	0.890233	-0.16667	-0.16667	-0.16667	0
29	29000	145	5.7942TE-05	145000	7250	0.05	0.005	16500	0.568966	29100	174	6.953E-05	174000	8700	0.05	0.005979	17000	0.584192	0.780323	0.760322	0.887323	0.886233	-0.16667	-0.16667	-0.16667	0
30	30000	150	5.9940TE-05	150000	7500	0.05	0.005	17000	0.566667	30100	180	7.1928E-05	180000	9000	0.05	0.00598	17500	0.581395	0.776323	0.756322	0.883323	0.882233	-0.16667	-0.16667	-0.16667	0
31	31000	155	6.1938TE-05	155000	7750	0.05	0.005	17500	0.564516	31100	186	7.4326E-05	186000	9300	0.05	0.005981	18000	0.578778	0.772323	0.752322	0.879323	0.878233	-0.16667	-0.16667	-0.16667	0
32	32000	160	6.3936TE-05	160000	8000	0.05	0.005	18000	0.5625	32100	192	7.6723E-05	192000	9600	0.05	0.005981	18500	0.576324	0.768323	0.748322	0.875323	0.874233	-0.16667	-0.16667	-0.16667	0
33	33000	165	6.5934TE-05	165000	8250	0.05	0.005	18500	0.560606	33100	198	7.912E-05	198000	9900	0.05	0.005982	19000	0.574018	0.764323	0.744322	0.871323	0.870233	-0.16667	-0.16667	-0.16667	0
34	34000	170	6.7932E-05	170000	8500	0.05	0.005	19000	0.558824	34100	204	8.1518E-05	204000	10200	0.05	0.005982	19500	0.571848	0.760323	0.740322	0.867323	0.866233	-0.16667	-0.16667	-0.16667	0
35	35000	175	6.9930TE-05	175000	8750	0.05	0.005	19500	0.557143	35100	210	8.3916E-05	210000	10500	0.05	0.005983	20000	0.569801	0.756323	0.736322	0.863323	0.862233	-0.16667	-0.16667	-0.16667	0
36	36000	180	7.1928TE-05	180000	9000	0.05	0.005	20000	0.555556	36100	216	8.6314E-05	216000	10800	0.05	0.005983	20500	0.567867	0.752323	0.732322	0.859323	0.858233	-0.16667	-0.16667	-0.16667	0
37	37000	185	7.3926TE-05	185000	9250	0.05	0.005	20500	0.554054	37100	222	8.8711E-05	222000	11100	0.05	0.005984	21000	0.566038	0.748323	0.728322	0.855323	0.854233	-0.16667	-0.16667	-0.16667	0
38	38000	190	7.5924TE-05	190000	9500	0.05	0.005	21000	0.552632	38100	228	9.1109E-05	228000	11400	0.05	0.005984	21500	0.564304	0.744323	0.724322	0.851323	0.850233	-0.16667	-0.16667	-0.16667	0
39	39000	195	7.7922TE-05	195000	9750	0.05	0.005	21500	0.551282	39100	234	9.3508E-05	234000	11700	0.05	0.005985	22000	0.56266	0.740323	0.720322	0.847323	0.846233	-0.16667	-0.16667	-0.16667	0
40	40000	200	7.9920TE-05	200000	10000	0.05	0.005	22000	0.55	40100	240	9.5904E-05	240000	12000	0.05	0.005985	22500	0.561097	0.736323	0.716322	0.843323	0.842233	-0.16667	-0.16667	-0.16667	0
41	41000	205	8.1918TE-05	205000	10250	0.05	0.005	22500	0.54878	41100	246	9.8302E-05	246000	12300	0.05	0.005985	23000	0.559611	0.732323	0.712322	0.839323	0.838233	-0.16667	-0.16667	-0.16667	0
42	42000	210	8.3916TE-05	210000	10500	0.05	0.005	23000	0.547619	42100	252	0.00001007	252000	12600	0.05	0.005986	23500	0.558195	0.728323	0.708322	0.835323	0.834233	-0.16667	-0.16667	-0.16667	0

Table [12]: Hypothetical values in capture table metrics calculation

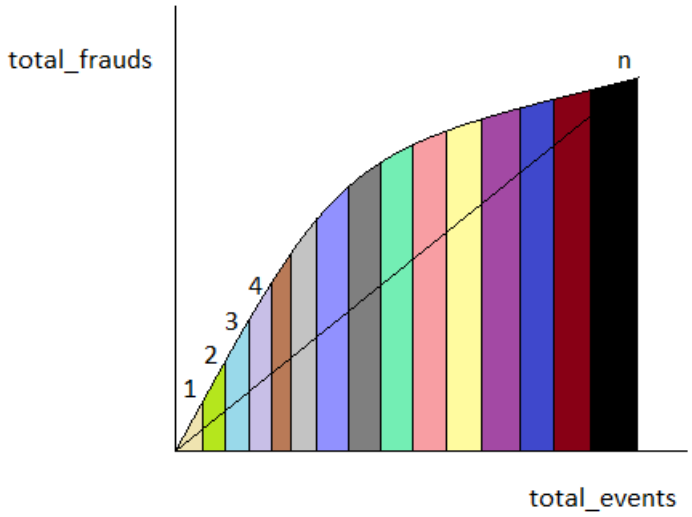


Figure [14]: This figure illustrates how we are calculating capture_table metrics for segregating frauds dollar amount saved or frauds caught or dollar amount increased or other such metrics as explained above with reference to gini metric calculation Lorenz curve. The multi-colored parts highlighted in this graph points to the cumulative values we are taking into consideration in the calculations. As such, x-axis points to the total number of transactions/events considered in real time decisioning system in fraud risk world or credit risk world and y-axis here denotes the total number of frauds caught or in the process of accounting for it while we increase the number of transactions.
Further explanation of this **figure [14]**, we say that we are considered multi-coloring to demonstrate the fact of creating cumulative values like cumulative_weights, cumulative_frauds, cumulative_weight_amounts, cumulative_fraud_amounts, cumulative_weight_amounts, cumulative_weight_scores.
For gini calculation, while referring to **Figure [10]**, we were considering the entire area under Lorenz curve while for capture table metrics calculation, we are considering part by part to reach till the entire count of total number of transactions is exhausted. To understand further, we add the part 1 highlighted in **figure[14]** as the first value of calculation, part 2 + part 1 would be considered further in calculation, part 1 + part 2 + part 3 further, and part 1 + part 2 + part 3 + … + part n till we reach total_events/total_transactions getting exhausted.

weight_amount (weighted amount)	amount_var*weights
fraud_amount	amount_var*weights
weighted_score_new	model_probability_value (for old model) * weights (for old model)
weighted_score_old	model_probability_value (for new model) * weights (for new model)
all_txns_weighted	sum of weighted transactions/sum of weights of all transactions
aggregate values of sum of weights	
sum of fraud_ind	total fraud transactions
sum of weighted_amount	sum of total weighted amount
sum of fraud_amount	sum of total fraud amount
sum of weighted_score_old	sum(weighted_score_old)
sum of weighted_score_new	sum(weighted_score_new)

Total_buckets	Num_parts (these are defined by total number of parts included in capture table metrics: it is generally 1000 or 10000), where total weighted transactions are distributed over the table started with 1 bps till 1000 bps or 10000 bps
Bucket_size	1000/total_buckets: these are defined by calculating 1000/total_buckets or 10000/total_buckets
Max_bucket	Print_buckets*bucket_size
Bps_list	[x for x in range(bucket_size, max_bucket+1, bucket_size)]
bps	Max_bucket
windowVal	Window.orderBy(df["model_prob_var1"].desc()).rangeBetween (Window.unboundedPreceeding,0)) cumulative_weights are calculated by doing sum(weights).over(windowval) max_value are calculated by filtering cumulative_weights >= {val} where val = value*bps*0.001 We filter input data on cumulative_weights <= {max_val}
cumulative_frauds are calculated with sum(fraud_ind).over(windowval)	Calculated field
cumulative_weight_amounts are calculated with sum(wgt_amount).over(windowval)	Calculated field
cumulative_frauds are calculated with sum(fraud_ind).over(windowval)	Calculated field
cumulative_weight_amounts are calculated with sum(wgt_amount).over(windowval)	Calculated field
cumulative_fraud_amounts are calculated with sum(fraud_amount).over(windowval)	Calculated field
cumulative_weight_scores are calculated with sum(weight_score_new).over(windowval)	Calculated field

Table [13]: Fields that were taken into consideration in calculation of Gini

We arrange data from input dataframes with respect to model_prob_scores in descending order such that transactions with highest model probability values are lying on the top of the bucket and transactions with lowest model probability are on the last bucket (bps).
Then we assign indexes based on the values defined in bps_list from 1to 1000 or from 1000 to 10000 (this is applicable for both models having old model probability and new model probability).
so overall fields that are being considered here are: rank_bps, count of transactions for old model, count of transactions for new model, count of fraud transactions for old model, count of fraud transactions for new model, fraud transactions coverage for old model, fraud transactions coverage for new model, dollar amount for new model, dollar amount for old model, fraud dollar amount for new

model, fraud dollar amount for old model, dollar rate for new model, dollar rate for old model, actual_old, actual_new, pred_old, pred_new, min_pred_old, max_pred_old, min_pred_new, max_pred_new, score_old_sum, score_new_sum, fraud transactions coverage lift, dollar lift, fraud dollar lift, dollar rate lift.
Finally we try to sort these values in descending order of model probability values
Interpretation of capture table metrics:

- Since we are taking into consideration all fraud transactions in first bps/bucket, the more the value of total transactions in first bucket/fraud transactions in first bucket, the more would be the value of the model being computed
- Since we are taking into consideration transactions having higher model probability values, we would get max_pred_old/max_pred_new values in the first bucket and eventually zero model probability values in the last bucket.
- Since we are taking into consideration transactions having lower model probability values, we would get min_pred_old/min_pred_new values in the first bucket and eventually zero model probability values in the last bucket.
- Similarly we get lift values indicating how has the new model performed as compared to old model (in this case, we do data modeling/model building process on ecommerce merchant clients belonging to different industries like Retail/Airline/Amusement_park/Travel/Food delivery/online delivery/Ticketing)

2. Rule automation tracking:
Purpose: This is another tool to track performance of how the decisioning system being setup in place is working for intended set of clients. We check rule performance through some graphs + rule performance reports if there are dips or faults that we are being able to find to check the rule performance. This helps in building strategy + performance leading up to a strategy of a client and if there are any changes that are needed to be done, rule fire count etc.

virtual_table_id	distinguishing field just like transaction_id (unique value would refer to unique transaction being made)
hierarchy_id	to distinguish between environments (test/prod/dev)
client_name (if we want to check rule fire count of a client)	Self-explanatory
review_ind	if monarch_decision==Review then value = 1 otherwise value = 0
review_rate	review_ind/txn_volume
review_rate_avg	group by time_period agg review_ind and then sum(sum(review_ind))
review_rate_avg	review_rate_avg/15
upper_limit_of_review_rate	review_rate_avg*1.25
lower_limit_of_review_rate	review_rate_avg*0.75
pr_ind	policy reject indicator which is defined by if monarch_decision == Policy Reject then value = 1 otherwise value = 0
policy_reject_rate	pr_ind/txn_volume
policy_reject_rate_avg	group by time_period agg pr_ind and then sum(sum(pr_ind))
policy_reject_rate_avg	policy_reject_rate_avg/15
upper_limit_of_policy_reject_rate	policy_reject_rate_avg*1.5
lower_limit_of_policy_reject_rate	policy_reject_rate_avg*0.5

Table [14]: Detailed explanation of fields which are involved for data pull to create and calculate rules.

historical values	df_hist as well call it since these are historical values associated with a certain client on which strategy performance needs to be checked

live values/current values	df_current/df_live as well call it since these are live values associated with a certain client on which strategy performance needs to be checked
scoring7_rules_tripped	rule_ids from real time decisioning system on which the values are -1/1 where the rules are getting tripped
resolution1_hostile	rule_ids from real time decisioning system where hostile marked transactions are considered
converted_transaction_amount	transaction_amount*exchange_rate (US/EURO) etc
Hostile_Fraud_Amount_ind	transactions consisting hostile fraud ind has been marked and the transaction amount associated with it; hostile_fraud_ind_amount is calculated when resolution1_hostile ==1 otherwise converted_transaction_amount
resolution1_indicator	resolution1 values
Rule_array_val	this column is created based on scoring7_rules_tripped
Rule_Id_1	cleaning up from rule_array_val
Fire_count	group by rule_id_1 aggregate rule_id_1 count
Hostile_Fraud_Count	group by rule_id_1 aggregate r1h_ind count
Hostile_Fraud_Amount	group by rule_id_1 aggregate Hostile_Fraud_Amount_ind count
Fire_Count_Rate_avg	((Fire_Count/txn_volume_avg)*100)/15
Hostile_Fraud_Rate_avg	Hostile_Fraud_Count/15
Hostile_Fraud_Dollar_Rate_avg	((Hostile_Fraud_Amount/Total_Amount)*100)/15
Fire_Count_avg	Fire_Count/15
Hostile_Fraud_Count_avg	Hostile_Fraud_Count/15
Total_Amount_avg	Total_Amount/15
Hostile_Fraud_Dollar_avg	Hostile_Fraud_Amount/15
Hostile_Fraud_Amount_ind	it's a conditional value when resolution1_hostile == 1 otherwise converted_transaction_amount
txn_volume_avg	df_hist.count()
Fire_Count_Rate	Fire_Count/txn_volume_avg
Hostile_Fraud_Rate	((Hostile_Fraud_Count/Fire_Count)*100)/15
Hostile_Fraud_Count	Hostile_Fraud_Count/15
Hostile_Fraud_Dollar_Rate	(Hostile_Fraud_Amount/Total_Amount)*100
Hostile_Fraud_Dollar	Hostile_Fraud_Amount
review_ind	monarch_decision=Review then 1 otherwise 0
pr_ind	monarch_decision=Policy reject then 1 otherwise 0
fraud_ind	resolution1_hostile=1 then 1 else 0
Pct_Dol_fraud_num	resolution1_hostile=1 and monarch_decision = Accept then converted_transaction_amount otherwise 0
Pct_Dol_fraud_den	resolution1_hostile=1 then converted_transaction_amount otherwise 0
review_rate	(review_ind/txn_volume)*100
policy_reject_rate	(pr_ind/txn_volume)*100
fraud_rate	(fraud_count/txn_volume)*100
Pct_Dol_Fraud_Accept	(Pct_Dol_Fraud_num/Pct_Dol_Fraud_den)*100

Table [15]: Detailed explanation of fields for checking rules performance

data	for strategy performance we take into consideration df_current/df_live + df_hist
review_ind	monarch_decision=Review then 1 otherwise 0
pr_ind	resolution1_hostile =1 then 1 otherwise 0
fraud_ind	resolution1_hostile=1 then 1 otherwise 0
Pct_Dol_Fraud_num	resolution1_hostile =1 and monarch_decision=Accept then converted_transaction_amount otherwise 0
Pct_Dol_Fraud_den	resolution1_hostile=1 then converted_transaction_amount otherwise 0
review_ind	groupby timeperiod sum(review_ind)
txn_volume	groupby timeperiod sum(transaction_id_corep)
pr_ind	groupby timeperiod sum(pr_ind)
fraud_ind	groupby timeperiod sum(fraud_ind)
Pct_Dol_Fraud_num	groupby timeperiod sum(Pct_Dol_Fraud_num)
Pct_Dol_Fraud_den	groupby timeperiod sum(Pct_Dol_Fraud_den)
review_rate	(review_ind/txn_volume)*100
policy_reject_rate	(pr_ind/txn_volume)*100
fraud_rate	(fraud_count/txn_volume)*100
Pct_Dol_Fraud_Accept	(Pct_Dol_Fraud_num/Pct_Dol_Fraud_den)*100

Table [16]: Detailed explanation of fields for strategy performance

	review_ind	monarch_decision=Accept then 1 otherwise 0
	pr_ind	monarch_decision= policy reject then 1 otherwise 0
	timeperiod	input defined
for historical data	review_rate	groupby timeperiod sum(review_ind)/groupby timeperiod sum(transaction_id_corep)*100
	policy_reject_rate	groupby timeperiod sum(pr_ind)/groupby timeperiod sum(transaction_id_corep)*100
	review_rate_avg	sum(review_rate)
	policy_reject_rate_avg	sum(policy_reject_rate)/15
	review_rate_avg	review_rate_avg/15
for live/current data	review_rate_current	groupby timeperiod sum(review_ind_current)/groupby timeperiod sum(transaction_id_corep)*100
	policy_reject_rate_current	groupby timeperiod sum(pr_ind_current)/groupby timeperiod sum(transaction_id_corep)*100
	review_rate_avg_current	sum(review_rate_current)/15
	policy_reject_rate_avg_current	sum(policy_reject_rate_current)/15
	review_rate_avg_current	review_rate_avg_current/15
	upper_limit_rr	review_rate_avg*1.25
	lower_limit_rr	review_rate_avg*0.75
	upper_limit_pr	policy_reject_rate_avg*1.5
	lower_limit_pr	policy_reject_rate_avg*0.5
some conditions	review_rate_current >=upper_limit_rr	Alert has been raised since upper threshold ("+str(upper_limit_rr)+") of review rate is breached. Last 15days avg review rate="+str(review_rate_avg)+"; Current day review rate = "+str(review_rate_current)
	review_rate_current <=lower_limit_rr	Alert has been raised since lower threshold("+str(lower_limit_rr)+") of review rate is breached. Last 15days avg review rate="+str(review_rate_avg)+"; Current day review rate = "+str(review_rate_current)

	policy_reject_rate_current >=upper_limit_pr	Alert has been raised since upper threshold("+str(lower_limit_rr)+") of policy reject rate is breached. Last 15days avg policy reject rate="+str(policy_reject_rate_avg)+"; Current day policy reject rate = "+str(policy_reject_rate_current)
	policy_reject_rate_current <=lower_limit_pr	Alert has been raised since lower threshold("+str(lower_limit_pr)+") of policy reject rate is breached. Last 15days avg policy reject rate="+str(policy_reject_rate_avg)+"; Current day policy reject rate = "+str(policy_reject_rate_current)
other final conditions	review_rate_current >= upper_limit_rr) | (review_rate_current <= lower_limit_rr) | (policy_reject_rate_current >= upper_limit_pr) | (policy_reject_rate_current <= lower_limit_pr)):	rules_performance calculation

Table [17]: Detailed explanation of fields for strategy threshold

Virtual_table_id	global column name to select a particular client
transaction_id	This is a unique identifier for every transaction that flows into a banking system with a real time rule based decisioning system
hierarchy_id	This segregates metadata of several ecommerce/merchants on the go with the environment we are working in which are production, development and test.
scoring7_rules_tripped	This list contains values of rule_id, which are assigned some values like -1,1 (tripped status/non-tripped status)
resolution1_hostile	This list contains values of rule_id which have been assigned to get resolution1_hostile values
converted_transaction_amount	transaction amount in USD/EURO/other countries internationally
ai_model_probability	this probability score value is calculated from AI model which is created on client side of things, Accertify Index Score is assigned from Accertify side of things
model_probability	this probability score value is calculated offline
total_score	This value helps to create monarch_decision column value like Accept/Review/Policy reject. For example, if total_score < 0 then Accept, total_score >=0 and total_score < 100000 then Review, total_score > 100000 then Policy reject
resolution1_missed	this field helps in marking for resolutions missed
challenger_flag	if the value of this challenger flag == Y, we take into consideration these records only
new_score	we fetch this field to get the new score values
monarch decision	this is created based on total score values
total_score < 0	accept
total_score >= 0 & total_score < 10000	review
total_score >=10000	policy reject

Table [18]: Fields considered in data pull process.

Table[14]-[18] explain the list of fields that are referred for rule build performance & tracking.
Note: How do we start with the process of decision made for strategy for clients on board:
we start with the datapull of the intended client (we tend to do data pull process to make a rules based strategic decision from live and historic tables)

2.1 Iteration for rule build performance:

Purpose: This approach is carried forward when the rules have been assigned as real-time decisioning system(ETD), and we are iterating the process to ensure that we are getting correct decision from strategy performance like good transactions are segregated in Accept bucket, supposedly fraud transactions are segregated in policy reject/review/reject bucket.
Steps that are followed in rule build process and the iterations that are performed:
We take into consideration development and production merged data after applying necessary tagging transformations to it. (We can consider test data only if it is significantly consisting of fraud transactions).
We segregate the data into development dataset (dataset on which model was trained) and validation datasets (model on which model is supposed to be tested where data is extracted from another set of months as compared to where data is extracted for development)

total_score	field created on client side (whichever ecommerce merchant client we are working on that point of time
real time decisioning (RTD)	a column created based on total_score, for example if total_score < 550 then RTD = accept, if 550 <= total_score < 300000 then review
merge process	we merge validation + development dataset
fraud_event	fraud_ind * 1
wgts	they are assigned 1 value for all transactions
fraud_dollar	fraud_ind * fn_amount_usd
bucketizer	we create bucketizer values consisting of a range of model probabilities to check the distribution of sum(fraud_ind), percentage of frauds over total_transactions: this gives an idea of how many frauds pertaining to certain model probability score range lies in a particular bucket
model_probability band values	we create this band values as follows: [-9999, 0.005, 0.02, 0.05, 0.1, 0.2, 0.35, 0.55, 0.725, 0.875, 0.95, 1.0]
conditions laid for model_probability band values are:	when model_prob <0.0005 then a.0-0.0005 when 0.005 <= model_prob < 0.02 then b.0.005-0.02 when 0.02 <= model_prob < 0.05 then c.0.02-0.05 when 0.05 <= model_prob < 0.1 then d.0.05-1.0 when 0.1 <= model_prob < 0.2 then e.0.1-0.2 when 0.2 <= model_prob < 0.35 then f.0.2-0.35 when 0.35 <= model_prob < 0.55 then g.0.35-0.55 when 0.55 <= model_prob < 0.725 the h.0.55-0.725 when 0.725 <= model_prob < 0.875 then i.0.725-0.875 when 0.875 <= model_prob < 0.95 then j.0.875-0.95 when 0.95 <= model_prob < 1.0 then k.0.95-1.0
then we create new_score which keeps on reiterating	This is defined as follows: [-9999, 0, 0.005, 0.02, 0.05, 0.1, 0.2, 0.35, 0.55, 0.725, 0.875, 0.95 1.0]

Conditions laid for model_probability band values are:	when model_prob < 0.005 then F.col("score") – 1600 when 0.005 <= model_prob < 0.02 then F.col("score") – 1000 when 0.02 <= model_prob < 0.05 then F.col("score") – 350 when 0.05 <= model_prob < 0.1 then F.col("score") – 100 when 0.1 <= model_prob < 0.2 then F.col("score") + 0 when 0.2 <= model_prob < 0.35 then F.col("score") + 50 when 0.35 <= model_prob < 0.55 then F.col("score") +150 when 0.55 <= model_pob < 0.725 then F.col("score") + 250 when 0.725 <= model_prob < 0.875 then F.col("score") + 550 when 0.875 <= model_prob < 0.95 then F.col("score") + 800 when 0.95 <= model_prob < 1.0 then F.col("score") + 1000
then we create a new column named new_RTD based on newly created new_score column:	when new_score > 550 we assign ACCEPT when 550 <= new_score < 300000 we assign REVIEW otherwise REJECT
then we create rule performance report based on these fields:	
xc_fin	this indicator is created after we apply business transformation logic
period	a time period defined to create rule build performance/check rule build performance
RTD	real-time decisioning
prob_band	model probability bands
new_RTD	newly created RTD
sum(fn_amount_usd)	total_transaction_amount in USD
count(1) as total_event	total number of transactions
sum(fraud_event)	total number of fraud transactions
sum(fraud_dollar)	total fraud dollar

Table [19]: Fields considered in rule performance iteration process for one set of model probability bands.

model_probability_band_values	[-9999, 0, 0.001, 0.003, 0.01, 0.025, 0.1, 0.2, 0.35, 0.6, 0.825, 0.925, 1.0]
conditions laid out for model_probability band values are:	when model_prob < 0.001 then a.0-0.001 when 0.001 <= model_prob < 0.003 then b.0.001-0.003 when 0.003 <= model_prob < 0.01 then c.0.003 – 0.01 when 0.01 <= model_prob < 0.025 then d.0.01 – 0.025 when 0.025 <= model_prob < 0.1 then e.0.025 – 0.1 when 0.1 <= model_prob < 0.2 then f.0.1-0.2 when 0.2 <= model_prob < 0.35 then g.0.2-0.35 when 0.35 <= model_prob < 0.6 then h.0.35-0.6 when 0.6 <= model_prob < 0.825 then i.0.6-0.825 when 0.825 <= model_prob < 0.925 then j.0.825-0.925 when 0.925 <= model_prob < 1.0 then k.0.925-1.0
then we create a new column named new_RTD based on	when new_score < 0.001 then F.col("score") – 3000

newly created new_score column:	when 0.001 <= new_score < 0.003 then F.col("score") – 1500 when 0.003 <= new_score < 0.01 then F.col("score") – 1000 when 0.01 <= new_score < 0.025 then F.col("score") – 800 when 0.025 <= model_prob < 0.1 then F.col("score") – 500 when 0.1 <= model_prob < 0.2 then F.col("score") – 250 when 0.2 <= model_prob < 0.35 then F.col("score") + 300 when 0.35 <= model_prob < 0.6 then F.col("score") + 600 when 0.6 <= model_prob < 0.825 then F.col("score") + 1000 when 0.825 <= model_prob < 0.925 then F.col("score") + 1500 when 0.925 <= model_prob < 1.0 then F.col("score") + 2000
then we create a new column named new_RTD based on newly created new_score column:	when new_score > 550 we assign ACCEPT when 550 <= new_score < 300000 we assign REVIEW otherwise REJECT
then we create rule performance report based on these fields:	
xc_fin	this indicator is created after we apply business transformation logic
period	a time period defined to create rule build performance/check rule build performance
RTD	real-time decisioning
prob_band	model probability bands
new_RTD	newly created RTD
sum(fn_amount_usd)	total_transaction_amount in USD
count(1) as total_event	total number of transactions
sum(fraud_event)	total number of fraud transactions
sum(fraud_dollar)	total fraud dollar

Table [20]: Fields considered in rule performance iteration process for second set of model probability bands and so on and so forth.

This report would help us know if we have made a correct decision by routing the intended transactions to review/reject/accept bucket (here review implies a particular transaction made in real time is supposed to be reviewed as being a fraudulent transaction, reject implies a particular transaction made in real time is supposed to be rejected as not being a fraudulent transaction, accept implies a particular transaction made in real time is supposed to be accepted as being a good transaction rather than a fraudulent transaction).
We also need to reach to create model_probability fields, these are the steps that we follow to reach to create this field which is basically assigning a probability score value to every transaction based on its nature of being fraudulent or non-fraudulent and that job is done by our model GBDT (gradient boosting decision tree).
As mentioned earlier in **point 13** related to GBDT script which runs on few parameters:

r	build a regression model
nt	number of threads (this field is of significance)
cn	minimum number of inputs to split a node, nodes with fewer inputs becomes leaf node
cvmin	minimum value to split a node, nodes with lesser value become leaf nodes (for classification only)

cvmax	maximum value to split a node
sg	surrogate splits to compute for each node, a surrogate is an alternative variable to use if the main split variable is missing during scoring
sc	compute surrogates only for categorical splits, only applicable if sg has been specified
pmml	output model in PMML format (in addition to the default binary format)
vi	number of variables to print in order of their importance in the model
w	weights to assign to instances of the positive class
wlv	compute leaf scores in a weighted manner

Table [21]: Detailed explanation of fields mentioned in GBDT model build process

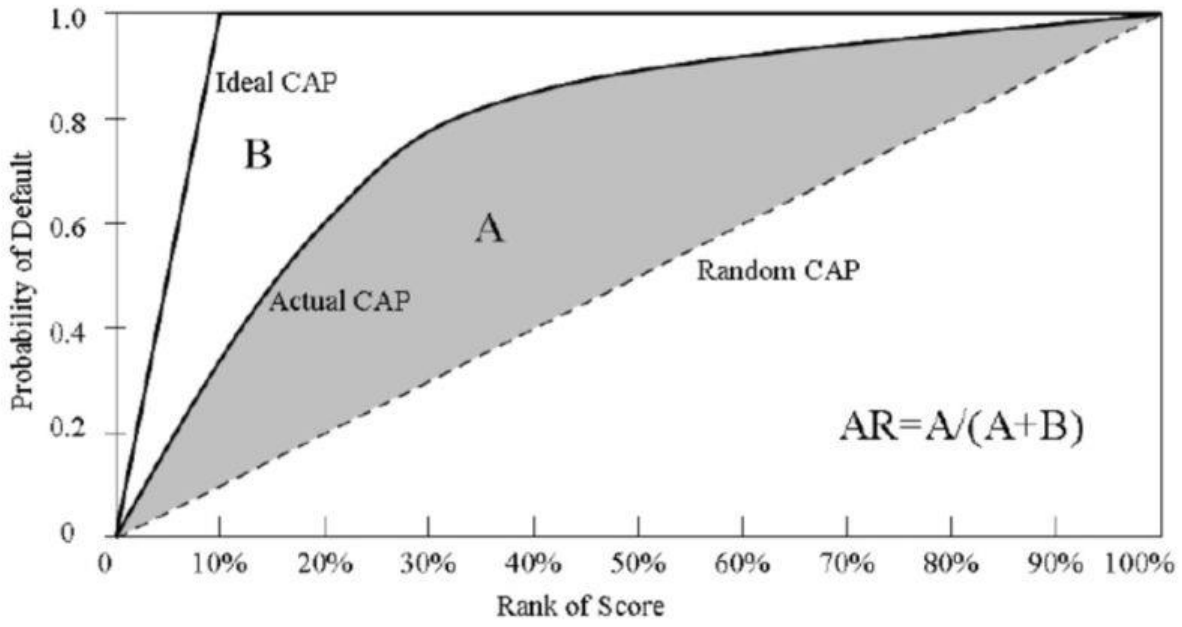
Results/Discussion:

Deciding clients based on industry/size/fraud_rate/products+services provided, months for model development and validation (holiday, non-holiday), sampling rate (current sampling is done taking into consideration the limitation of offline velocity compute and gini calculation).
Preliminary analysis of new features to be added or mapped from one industry to another. What will be its impact on data creation? Is new offline feature creation through libraries time-taking? Does its RnP/fraud trend justify its creation over choosing datapull of whatever features are available?
Data ETL (takes 70-80% of timelines if > 15 customers/clients (voluminous) are involved. Does ETL pipeline impacts client/customer selection in step 1. Always weigh in the trade-off between data needed for pull and the time it takes
QC of final modeling variables (overall volumes, numeric QC, categorical/character QC)
Model development with different hyperparameters:
:As mentioned the input parameters in the table above, here we would add explanation of how tuning of hyperparameters would help in establishing results
:Xmx - this helps in deciding how much heap size is needed by the Virtual Machine. Make sure that not more than 25% of the memory is blocked by this allocation, else other users' jobs would start getting affected. For example, Xmx100G can be used for a 100 GB training data
:XX:+UseG1GC - garbage first garbage collector: to manage pause time of the VM
Since we are using GBDT, number of trees and depth of each tree: these two arguments have the maximum impact on model performance. As a rule of thumb, a combination of 1000 trees and 5 depth is optimum. Reducing one/both these features reduces bias but increases variance. Increasing one/both these parameters increases bias but reduces variance. 800 trees are sufficient for a data size < 2MM and 1000-1200 trees are optimum for a data size of 10MM. Data size can be defined by: Number of transactions, Number of features: with more features (lets say 300) and lesser number of transactions - it is better to keep lesser depth (eg.4). With lesser features (eg, 100) and a greater number of transactions, more depth can be used (5/6)
Model performance (defined by Gini lift + fraud event capture lift + fraud dollar capture lift) increases with more depth/trees and then starts planning
Other 2 deciding factors for GBDT model performance are learning rate and minimum number of observations for a node's split. Learning rate is usually kept as 0.1. Increasing the learning rate will fasten the model build time (eg, a model built on a date with 150 variables, 10 million observations, 1200 trees, 5 depth and 0.1 learning rate takes 6-7 hours for completion). Each tree build takes 17-50 seconds depending on hyper-parameters.
Decreasing learning rate, increasing number of trees, increasing the depth slows down model build time and vice versa

Table [22]: Inferences after GBDT model script is run and model probabilities are assigned to every transaction. **Table [21]** explains the detailed fields given as an input to GBDT model, **Table [22]** explains the results after executing GBDT model.

3 Gini-coefficient interpretation in credit risk modeling calculation: Why is GINI coefficient important in credit risk default models? Before going into the gini coefficient, for some reason the banking industry has another term for Gini called as the accuracy ratio or AR. This term in my view is

not suitable, because Gini is not measuring accuracy of predictions, like how accuracy is measured by confusion matrix. The goal of a credit rating model is firstly to enable the ranking of borrowers as a relative measure. As long as the model can rank a good borrower better than a risky borrower, this is good for business. Business can perform risk assessment, and onboard customers based on the scores. The class/probability is not so critical. The Gini coefficient as has been mentioned in earlier points measures separation power similar to how it is used in economics to measure inequality. However for credit risk, the higher the Gini, the better. It means that amongst good and bad ratings, the separation is good which is also known as discriminatory power is. Another interpretation to it is good borrowers and risky borrowers are "unequal".
Ideally, GINI can be measured regardless of model which are referred to as machine learning model like linear regression or logistic regression or decision tree which are used for scoring (as mentioned in **Figure [15]** where x-axis is Rank of Score). Benchmark Gini values have minimum typically 40% for newly developed models which is too low and good model might have GINI above 60%. Hence, we say that Accuracy Ratio can be misleading. GINI translates to AUROC of 80% which is visually better suited for the term "accuracy".
Gini = 2*AUROC – 1
ROC (receiver-operator characteristics) tells us how well a classification model performs – in credit risk it refers to how well a credit risk model classifies borrowers in to defaulters and non-defaulters whereas AUROC is the summary ROC curve which measures the ability of a model (binary classification model) to separate borrowers in to defaulters and non-defaulters. AUC for a random classifier (which classifies entire sample in to 50% defaults and 50% non-defaults) is 0.5. AUC for perfect classifier (which classifies the sample data in to defaults and non-defaults exactly with full accuracy (as per observed status of borrower)) is 1.
AUC for a classifier that we built in practically in real life lies between 0.5 and 1 (between random classifier and perfect classifier). More the AUC, more the predictive power of the classifier. Some methods to improve model Gini are choosing the right set of features, appropriate algorithm selection, hyperparameter tuning, coarse classification etc.



Figure[15]: Interpretation of this figure: we arrange credit score range values in descending order, higher credit score range implies lower chances of customer defaulting and lower credit score implies higher chances of customer not defaulting/good customer/being able to repay loans on time. So according to CAP curve, for Ideal CAP (cumulative accuracy profile), for the customers having first 10% credit score range have the highest probability to default and then it saturates. Whereas in real time scenario (Actual CAP), as we parse through customers having higher credit score range (lying in

first 10%) would have the lowest probability of default and then we keep on increasing credit score range (till we reach 100%), we would have attained customers who are likely to default more. For Random CAP, we have linear rise in GINI with decrease in credit risk score. CAP curve used by banks and regulatory firms to analyze discriminatory ability of rating systems that evaluate credit risks is similar to Lorenz curve used by banks and regulatory firms to analyze discriminatory ability of rule marking/decisioning systems that evaluate fraud risk. The detailed explanation can also be seen in **[5]**.

Few other methods exist to synthetically inflate or degrade model performance by playing around with the bucketing of final PDs.

- Arbitrary bucketing: This involves dividing the model generated PDs into arbitrary intervals or bins, regardless of the data distribution or statistical properties. We can check distribution across buckets to see if the bins are equally sized or have variations which might distort ARs.
- Low frequency bucketing: This is when the number of bins are too small. While this approach can be useful for handling outliers and extreme values, it can also lead to loss of information and bias in the data, especially if the number of bins is too small depending on sensitivity and distribution of default data across bins. We can always check AR computation across multiple binning solutions including no bins/deciles.
- Surgical coarse classing: Most of the binary classification models today use WOE based bins in coarse classing. This makes it possible that within a miniscule change in the rating to PD mapping, a handful of defaults can be migrated from a lower rating grade to higher rating grade effectively changing GINI by 5%-10% depending on the sensitivity to default movement across higher buckets. We can check for AR sensitivity across risk grades by evaluating the change in AR by +/- 1 to 5 defaults in every risk grade. This would also help to understand if the calibration is extremely finetuned/overfitted that with a slight change in default distribution the model performance might stagger.

Input Distribution (to be updated)		
Buckets	**Bad Count**	**Good Count**
1	64.00	219.00
2	34.00	229.00
3	22.00	290.00
4	18.00	313.00
5	14.00	197.00
6	8.00	292.00
7	9.00	282.00
8	3.00	272.00
9	3.00	283.00
10	2.00	279.00

Impact Analysis on AR		
S.No	**Adjustment**	**Gini**
1	Base	54.27%
2	9 Buckets	52.74%
3	8 Buckets	52.39%
4	7 Buckets	51.85%
5	6 Buckets	48.91%
6	5 Buckets	51.45%
7	4 Buckets	46.12%
8	3 Buckets	46.61%
9	2 Buckets	32.41%
10	+2 Rollforward	57.20%
11	+4 Rollforward	59.25%
12	+6 Rollforward	61.31%
13	+8 Rollforward	63.36%
14	+2 Rollbackward	53.09%
15	+4 Rollbackward	51.03%
16	+6 Rollbackward	48.98%
17	+8 Rollbackward	46.92%

Max	**63.36%**
Min	**32.41%**

Sensitivity Analysis on AR's		
S.No	**Bucket Number**	**Sensitivity**
1	1-plus 1 default	0.158%
2	2-plus 1 default	0.063%
3	3-plus 1 default	-0.047%
4	4-plus 1 default	-0.174%
5	5-plus 1 default	-0.282%
6	6-plus 1 default	-0.385%
7	7-plus 1 default	-0.507%
8	8-plus 1 default	-0.624%
9	9-plus 1 default	-0.741%
10	10-plus 1 default	-0.860%
11	1-minus 1 default	-0.160%
12	2-minus 1 default	-0.065%
13	3-minus 1 default	0.046%
14	4-minus 1 default	0.175%
15	5-minus 1 default	0.285%
16	6-minus 1 default	0.389%
17	7-minus 1 default	0.512%
18	8-minus 1 default	0.630%
19	9-minus 1 default	0.749%
20	10-minus 1 default	1.748%

+1 Default	**Max Impact**	**0.16%**
	Min Impact	**-0.86%**
-1 Default	**Max Impact**	**1.75%**
	Min Impact	**-0.16%**

Figure[16]: Left most table consists of distribution of buckets from 1-10. PD bucketing is dependent on credit score range values (interpret values in 1000 – we have arranged credit score values in ascending order which is opposite to case of GINI as is explained in **Figure[16]**): lets say bucket 1 has credit score < 100 then PD = 20% (lets say) which also implies that bad count would be higher and good count would be lower as compared to other buckets, bucket 2 has 110 < credit score < 200 then PD = 15%, bucket 3 has 210 < credit score < 300 then PD = 12.5%, bucket 4 has 310 < credit score < 400 then PD = 10.0%, bucket 5 has 410 < credit score < 500 then PD = 6.0%, bucket 6 has 510 < credit score < 600 then PD = 5.0%, and so on last bucket has credit score > 910 then PD = 0.2%. This explanation represents the bucketing column. Bad count is the total number of defaulters

having a certain credit score and falling in that credit score range and hence in a particular bucket. Good count is the total number of non-defaulters having a certain credit score and falling in that credit score range in a particular bucket. So arbitrary bucketing would distort AR because of uneven distribution of good and bad counts.
Second table consists of impact analysis on AR where we can see adjustments being made to bucket values. As seen from the table, going from Base to 2 Buckets results in decreasing value of Gini. Here 2 Buckets implies first 20% in credit score range where GINI values are less (as can be seen from **Figure[16]**), 3 Buckets implies first 30% in credit score range where GINI is higher than 2 Buckets and so on and so forth, till Base where first 100% (total 100% or entire credit score range has been covered) implies highest GINI value as compared to other buckets. If this GINI value > 50% it would mean that model has not been able to score a customer defaulting or not defaulting depending on his transaction history, loan history and many other features considered while calculating score or PD values else if GINI value > 90% it would mean that model has been able to score a customer defaulting or not defaulting well. Furthermore, +2 RollForward implies hopping onto +2 buckets from previous considered bucket (lets say we consider Base here), +8 RollForward implies hopping onto +8 buckets from previous considered bucket (lets say we consider Base here again), so we are going beyond Base values which we have already established has the maximum GINI out of all buckets considered. Similarly, +2 RollBackward implies hopping to -2 buckets from previously considered bucket (lets say we consider Base here) hence the GINI value is justified (value between Base and 9 Bucket), if we go +8 RollBackward it would imply hopping to -8 buckets from previously considered bucket (lets say we consider Base here) hence the GINI value is justified (value between 3 to 4 Bucket). By this method, we would get maximum and minimum values of GINI.
Third table is sensitivity analysis on AR. Sensitivity is finding true positives amongst the set of relevant elements (true positives and false negatives). From the sensitivity column, we can see that as we increase bucket value (higher bucket value implies higher credit risk score and more likely to default), sensitivity drops implying the number of cases correctly identified as good customer (positive for not defaulting). In other words, as we go from 1-plus 1 default to 10-plus 1 default, AUC curve is bending from proportion of bads towards proportion of goods. Hence, according to GINI = 2*AUC -1 and referring to **Table 1**, we have increasing good count and decreasing bad count. This results in AUC = 0.5 + B (B is the area under curve bending towards proportion of bads) to 0, hence GINI/AR value turns to be negative and increasingly negative as we reach to 10-plus 1 default. As AUC = 0, ROC = 1 and has completely overlapped with FPR (false positive rate), making sensitivity bend towards negative curvature and hence we get negative values. For another set of bucket number (1 minus default to 10 minus default), we are essentially going from bending towards FPR towards TPR and increasing sensitivity. GINI would also go from negative values to positive value.
Illustration of CAP curve:
- To validate the rating model performance
- To generate probability of defaults

Rating model performance validation:
The validation of rating model performance is done by comparing the CAP curve of the rating model with the CAP curve of a perfect model and CAP curve of a random (imperfect) model. CAP curve of the rating model usually will be in between that of the perfect model and random model, occupying some part of the area formed by the CAP curve of the perfect model and that of the random model. The percentage area occupied by the CAP curve of the rating model is called the Accuracy Ratio (AR). An AR of around 40% to 60% is considered as a model with fair predictive power.
Example: Let us assume the following hypothetical portfolio with rating profile 1 to 8 where 1 is the best and 8 is the worst.

ACTUAL MODEL				
Rating	#good	#bad	#total	DR
8	38	2	40	20.00%
7	50	0	50	6.84%
6	59	1	60	1.25%
5	59	1	60	0.00%
4	55	0	55	0.00%
3	25	0	25	0.00%
2	5	0	5	0.00%
1	5	0	5	0.00%
TOTAL	296	4	300	3.47%

Table [23]: The above table has the rating grades sorted from worst to best, number of good borrowers, number of bad borrowers, total number of borrowers and finally the default rate, which is the ratio between the number of bad borrowers and the total number of borrowers in each grade.

The steps to construct CAP curve (and AR computation) are as under:

i) Sort the borrower rating pool from worst to best grade.

ii) Find the cumulative number of borrowers (x) and also cumulative number of defaults (y) against each grade. For e.g., x for rating grade 8 would be 40/300 = 13.3% and y for rating grade 8 would be 2 / 4 = 50%. Similarly x for rating grade 7 would be (50 + 40)/300 = 30% and y would be (2+0)/4 = 50%, and so on.

	Actual Model	
	Cpop (x)	Cdef(y)
	0	0
8	13.3%	50.0%
7	30.0%	50.0%
6	50.0%	75.0%
5	70.0%	100.0%
4	88.3%	100.0%
3	96.7%	100.0%
2	98.3%	100.0%
1	100.0%	100.0%

Table[24]: This table explains all the calculations of actual model for cumulative values of x and y

iii) Identify a perfect model in relation to the observed profile and construct cumulative number of borrowers (x) and cumulative number of defaults (y) for the perfect model. For eg., in the current example a perfect rating model would be a rating model if all the defaults emerge from the most inferior grade, 8.

PERFECT MODEL

ORR	#good	#bad	#total	DR
8	36	4	40	20.00%
7	50	0	50	6.84%
6	60	0	60	1.25%
5	60	0	60	0.00%
4	55	0	55	0.00%
3	25	0	25	0.00%
2	5	0	5	0.00%
1	5	0	5	0.00%
TOTAL	296	4	300	3.47%

Table[25]: So the profile of a perfect rating model would look like the below table: here the only striking difference is where #bad is different from #good (if we compare it with the values from actual model)

Perfect Model	
Cpop_per	Cdef_per
0	0
13.3%	100.0%
30.0%	100.0%
50.0%	100.0%
70.0%	100.0%
88.3%	100.0%
96.7%	100.0%
98.3%	100.0%
100.0%	100.0%

Table[26]: The cumulative number of borrowers (x) and cumulative number of defaults (y) for the above perfect rating model would be as under:

iv)The next step is to identify a random model (imperfect). The random model for the current example would a rating profile where the defaults are distributed across all grades equally.

RANDOM MODEL				
ORR	#good	#bad	#total	DR
8	37	0.50	38	20.00%
7	37	0.50	38	6.84%
6	37	0.50	38	1.25%
5	37	0.50	38	0.00%
4	37	0.50	38	0.00%
3	37	0.50	38	0.00%
2	37	0.50	38	0.00%
1	37	0.50	38	0.00%
TOTAL	296	4	300	3.47%

Table[27]: The following table shows the same:
here #bad and #good have the same values, hence we would have around same default ratio/default rates

The above numbers are just theoretical numbers as observing decimal number of defaults (0.5 in the above case) is meaningless. The idea is to arrive at the accuracy ratio using the observed rating model and the corresponding theoretical perfect and imperfect model.

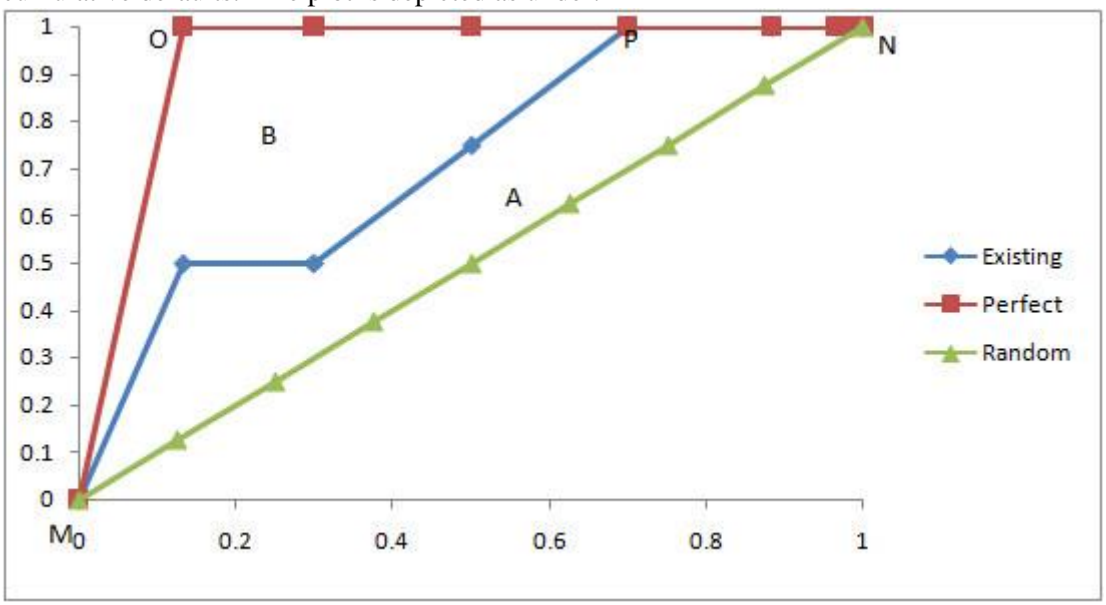
Random Model	
Cpop_rdm	Cdef_rdm
0	0
12.5%	12.5%
25.0%	25.0%
37.5%	37.5%
50.0%	50.0%
62.5%	62.5%
75.0%	75.0%
87.5%	87.5%
100.0%	100.0%

Table[28]: The cumulative number of borrowers and the cumulative number of defaults for the random model is given in the following table:

	Actual Model		Perfect Model		Random Model	
	Cpop (x)	Cdef(y)	Cpop_per	Cdef_per	Cpop_rdm	Cdef_rdm
	0	0	0	0	0	0
8	13.3%	50.0%	13.3%	100.0%	12.5%	12.5%
7	30.0%	50.0%	30.0%	100.0%	25.0%	25.0%
6	50.0%	75.0%	50.0%	100.0%	37.5%	37.5%
5	70.0%	100.0%	70.0%	100.0%	50.0%	50.0%
4	88.3%	100.0%	88.3%	100.0%	62.5%	62.5%
3	96.7%	100.0%	96.7%	100.0%	75.0%	75.0%
2	98.3%	100.0%	98.3%	100.0%	87.5%	87.5%
1	100.0%	100.0%	100.0%	100.0%	100.0%	100.0%

Table[29]: Putting everything together, the combined table would look like: we compare the values of actual/perfect/random models with respect to their cumulative value calculations and we get these results.

v) Construct a scatter plot (with line) using the above data series with x axis containing cumulative population (actual model, perfect model and random model) and y axis containing the corresponding cumulative defaults. The plot is depicted as under:



Figure[17]: The curve MPN represents the CAP curve for the actual rating model. The curve MOPN represents the CAP curve for the perfect rating model and the curve MN represents the CAP curve for an imperfect (random) rating model. The area represented by "A" (i.e, MPN) is the accuracy ratio. It represents a portion of the area formed by the perfect model (MOPN) and the random model (MN). The area MOPN is considered 100%. The percentage area occupied by MPN over MOPN is the accuracy ratio. Accuracy ratio for the above works out to be 44%.

Probability of defaults generation from CAP curve:

Another application of CAP curve is its ability to generate PDs.

Let us denote `k' as the concavity of the CAP curve (curve MPN). Higher the k, the more concave is the curve. The CAP curve in general can be represented by the following function (as in **Figure[17]**),

$$y = \frac{1-e^{-kx}}{1-e^{-k}} \qquad \text{Equation (1)}$$

where k is the concavity, y = cumulative number of defaults

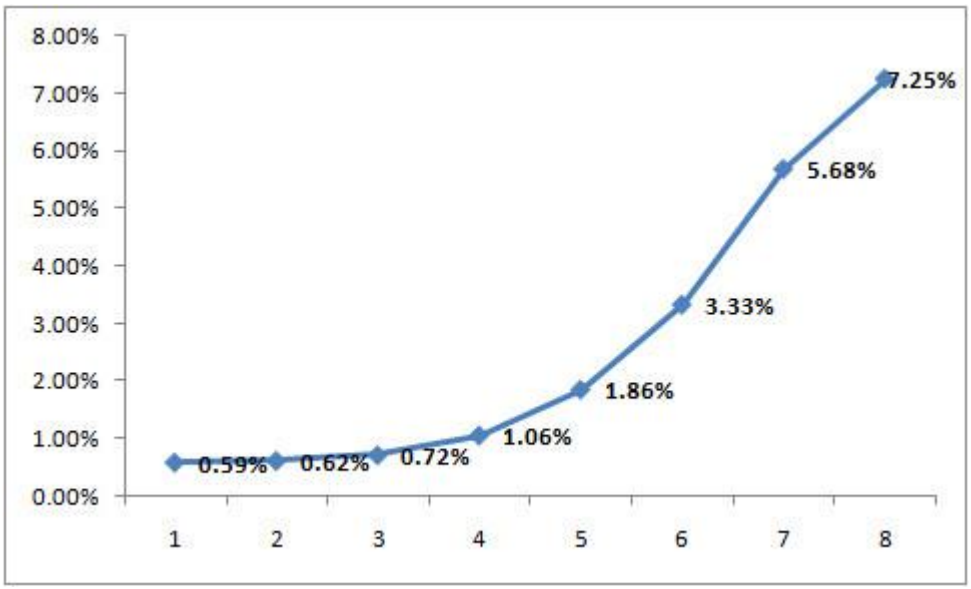
In other words, it denotes that the cumulative number of defaults (y) is a function of cumulative number of borrowers (x) and the function is the form of the equation stated as above.

dy/dx represents a small change in the cumulative number of defaults for a given small change in the

cumulative number of borrowers.
Applying differentiation principal of calculus,
dy/dx = $[1/(1-e^{-k}))]*[-e^{-kx}*(-k)]$
which when simplified gives the following equation,
dy/dx = $[k*e^{-kx}]/[1-e^{-k}]$ Equation (2)
The above denotes the change in defaults for the change in number of borrowers. This multiplied by the overall observed default rate will yield the probability of default for that grade.
Probability of default = (Overall default rate) * (dy/dx)

k	2.9163828		
Cpop	**Cdef**	**MSE**	**pd**
x	y=f(x)		f'(x) X DR
13.3%	34.1%	0.03	**7.25%**
30.0%	61.6%	0.01	**5.68%**
50.0%	81.1%	0.00	**3.33%**
70.0%	92.0%	0.01	**1.86%**
88.3%	97.7%	0.00	**1.06%**
96.7%	99.4%	0.00	**0.72%**
98.3%	99.7%	0.00	**0.62%**
100.0%	100.0%	0.00	**0.59%**
	min t_mse	0.05	
	using solver		

Table [30]: In the above table, x is the cumulative number of borrowers of the actual rating model from grade 8 to grade 1 (ref para 1, term iii, 2nd table)
y = f(x) is the corresponding value after applying equation (1). For eg, for x = 13.3%,
y = $[1-e^{(-k*0.133)}]/[1-e^{-k}]$
y is calculated initially by considering a random number for k across all grades. After calculating across all grades, k is changed in such a way that the difference between the resulting y and the actual observed y (ref para1, item iii, 2nd table) is minimum. MSE is the mean squared error between the resulting y and observed y. MSE is minimum when k = 2.92. In other words, the concavity of the CAP curve of the actual rating model is 2.92
dy/dx = $[2.92 * e^{-2.92*0.133}]/[1-e^{-2.92}]$
PD for the rating grade 8 = above * 3.47% = 7.25%, where 3.47% is the observed default rate (Ref item 1, titled Example)
Similarly for rating grade 7,
dy/dx = $[2.92*e^{-2.92*average(0.30,0.133)}]/[1-e^{-2.92}]$
PD for rating grade 7 = above * 3.47% = 5.7%
Note that term average (0.30, 0.133), instead of using 0.30 which is the cumulative number of borrower value for grade 7, it is advisable to use the average of cumulative number of borrower value of grade 8 and grade 7 in order to have a continuous function. Similarly for grade 6, average of cumulative number of borrower value of grade 7 and 6 is used.



Figure[18]: This PD curve is generated as explained above. This method can be applied specifically for low default portfolios (LDP)

Conclusion: We have explained various parts in this paper for these problem statements:
1) Gini computation/calculation and its significance in risk management based industries.
We reach to a conclusion that Gini helps in saving fraud dollar losses by applying fraud risk based mitigation strategies.
2) Steps involved in a real business world problem where we collate details related to many business nuances and create the final data to be further referenced in business metrics computation.
We have explained various steps which refer to business rules and logic that gets imbued to create the data which eventually is used for metrics computation.
3) Gini values calculation at critical bucket (significance of critical bucket lies in fraud risk/credit risk based business problems).
We have explained the implementation of Gini metrics computation and calculation.
4) Accuracy calculation for model with difference machine learning probabilities or across different time periods.
These metrics are referred to check the performance of machine learning model, but in case of fraud risk (whether its application is in underwriting credit risk models, credit risk based scoring model, fraud risk model), helps us to understand how the model probabilities assigned to every transaction has led to better outcome in detecting frauds.
5) Capture table metrics calculation to understand the spread of fraud transactions or total transactions.
This is rank order of model probabilities values' based bucketing to detect the riskiest fraud transactions without having to spend money to review the transactions falling in accept/review/reject buckets.
6) Rule performance building process and how it helps in building strategies for a specific customer.
We determine the steps that eventually help in checking rule performance, which in turn helps in developing strategy for customers.
7) Rule tracking and automation to optimize the rule build process.
We determine the steps that are involved in automation and optimization for rules implementation.
8) Gini metrics value/accuracy ratio value in credit risk world.
We determine the Gini metrics implementation and interpretation in credit risk based use-case scenario.

Funding information: This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

Ethics approval/informed consent: This article does not contain any studies involving human participants performed by any of the authors. If your work involves humans, it is necessary to include two additional statements.

Author's contribution: The author confirms sole responsibility for the following: study conception and design, data collection, analysis and interpretation of results, and manuscript preparation.

Conflict of interest: The author declares that they have no competing interests or conflict of interest.

Data availability statement: No new data were created or analysed in this study. Data sharing is not applicable to this article.

References:

[1] https://corporatefinanceinstitute.com/resources/economics/gini-coefficient/
[2] https://ourworldindata.org/what-is-the-gini-coefficient
[3], [4] See Anthony B Atkinson, 'On the Measurement of Inequality'. *Journal of Economic Theory* 2, no. 3 (1970): 244–63. Available here.
[5] https://www.bis.org/bcbs/events/oslo/liebigblo.pdf
[6] https://legal.thomsonreuters.com/blog/the-fundamentals-of-fraud-risk/#:~:text=One%20of%20the%20biggest%20risks,large%20corporations%2C%20and%20small%20businesses.

List of table references:

[1] Basic interpretation of fields required to make gini calculations
[2] cumulative column calculations
[3] steps and detailed notes to create datasets
[4] comprehensive calculation of fields leading to gini_new and similarly we would have fields for gini_old values
[5] briefly mentioned fields that are used in gini metrics calculation
[6] intermediate fields that we refer to for calculating gini metrics
[7] description of intermediate fields in gini calculation as explained in this table
[8] accuracy calculation metrics
[9] detailed explanation of fields that are mentioned in table
[10] detailed explanation of fields considered and calculated for critical bucket
[11] detailed explanation of capture table metrics
[12] hypothetical values in capture table metrics calculation
[13] fields that were taken into consideration in calculation of gini
[14] detailed explanation of fields which are involved for datapull to create and calculate rules
[15] detailed explanation of fields for checking rules performance
[16] detailed explanation of fields for strategy performance
[17] detalied explanation of fields for strategy threshold
[18] fields considered in datapull process
[19] fields considered in rule performance iteraton process for one set of model probability bands
[20] Fields considered in rule performance iteration process for second set of model probability bands and so on and so forth.
[21] Detailed explanation of fields mentioned in GBDT model build process
[22] Inferences after GBDT model script is run and model probabilities are assigned to every transaction.

[23] The above table has the rating grades sorted from worst to best, number of good borrowers, number of bad borrowers, total number of borrowers and finally the default rate, which is the ratio between the number of bad borrowers and the total number of borrowers in each grade.
[24] This table explains all the calculations of actual model for cumulative values of x and y
[25] So the profile of a perfect rating model would look like the below table: here the only striking difference is where #bad is different from #good (if we compare it with the values from actual model)
[26] The cumulative number of borrowers (x) and cumulative number of defaults (y) for the above perfect rating model would be as under
[27] The following table shows the same: here #bad and #good have the same values, hence we would have around same default ratio/default rates
[28] The cumulative number of borrowers and the cumulative number of defaults for the random model is given in the following table
[29] Putting everything together, the combined table would look like: we compare the values of actual/perfect/random models with respect to their cumulative value calculations and we get these results.
[30] In the above table, x is the cumulative number of borrowers of the actual rating model from grade 8 to grade 1 (ref para 1, term iii, 2nd table)

List of figure references:
[1] self explanatory figure for gini lorenz curve
[2] comparison between interpretations of definition of gini
[3] a scenario explaining the fields that go in real-time transaction database
[4] mapping of input variables stored in development/production mode with different fields of different data types
[5] univariate analysis
[6] velocity variable calculation
[7] sum velocity variable calculation
[8] automation script
[9] overall gini value calculation
[10] shaded area representing gini calculation
[11] gini metrics calculation of model with old/new model probabilities without transaction_id
[12] gini metrics calculation of model with old/new model probabilities with transaction_ids available
[13] critical bucket calculation
[14] capture table metrics
[15] accuracy ration calculation for credit risk
[16] different analysis methods
[17] CAP curve
[18] PD curve